ALTAIR

ONLY FORWARD

Altair Accelerator 2024.1.0

Administrator Guide

altair.com

Contents

Altair Accelerator Administrator Guide

This manual is written for the Accelerator system administrator who needs to configure and manage the use of this Altair Accelerator product after it is installed. This guide describes basic tasks, including submitting jobs, tracking job information, and analyzing and solving common problems.

This chapter covers the following:

The administrator is expected to understand UNIX system processes, the dynamics of UNIX interactive shells, shell scripting techniques and general trouble shooting concepts. As configuration is part of the role of the Accelerator administrator, knowledge of schedulers is also expected.

For details about the usage and capabilities of using Accelerator, refer to the *Altair Accelerator User Guide* and *Altair Accelerator User Tutorials*.

Note:

The terminology in this release has changed from the previous one.

The Accelerator products are built on platform called vov using a client-server architecture with remote-procedure-calls (RPC). The server software module is called vovserver. It communicates to clients using the vov protocol; vovservers can also be configured to respond to http requests: the REST API is implemented on top of http. There are several different client types, those that make requests to the vovserver are typically implemented using vovsh (the vov shell - a Tcl interpreter); those that respond to vovserver requests to run jobs or tasks are taskers and the software here is called vovtasker. The vovtasker can run on the same host as the vovserver or on a separate host; these hosts are typically referred to as compute nodes, compute hosts or execution hosts.

The architecture allows for multiple vovservers to communicate with each other via a vovagent. Examples of vovagents include vovwxd, indirect taskers and vovlad.

In the 2021.1.0 release, the term *slave* has been deprecated and has been replaced with the term *tasker*. The web user interface and the online documentation have been updated to reflect this change, as has the majority of the code base. Subsequent releases will complete the transition.

Accelerator Features

Accelerator is a high-performance, enterprise grade job scheduler designed for distributed High Performance Computing (HPC) environments. Accelerator provides a cost-effective, highly adaptable solution capable of managing compute infrastructures from small dedicated server farms to complex distributed HPC environments.

A full-featured scheduler, Accelerator is equipped with a comprehensive set of policy management features including FairShare, Preemption, and Reservations, which can be customized per organizational requirements to maximize resource utilization and throughput.

The services provided by Accelerator include job prioritization, automatic job queuing, license management, resource management and reporting the status of jobs as well as the usage and availability of resources.

The fields of application include hardware and software engineering, running calculations on a farm, electronic design automation (EDA) and other industries.

Accessing Accelerator

Accelerator can be accessed via the following media:

- **Web UI.** Configuring Accelerator properties, and viewing job status, configurations, available resources and more is available through the web user interface.
- **GUI.** Graphical user interface, independent of the web is also available for graphical views of current job and resource statuses.
- **CLI Command.** Commands are also available for configuration, viewing the status of jobs and resources. GUI and WebUI can be invoked through CLI commands.

Theory of Operation

During the initial setup, the Accelerator host server, vovserver, establishes a main port for communication and addition ports for web access and read-only access. The main process for the Accelerator vovserver is establishing a main port for communication plus additional ports for web and read-only access. Afterwards, the vovserver waits for and responds to incoming connection requests from clients.

Clients consist of *regular* clients that request a particular service, *taskers* (server farms) that provide resources, and *notify* clients that listen for events). As well as tasker-based resources, some clients provide central resources, which are stored in and accounted by the vovserver.

Regular clients can define jobs, or query data about jobs or system status. When a job is defined, it is normally placed in a scheduled state. Scheduled jobs are sorted into buckets. Jobs that have the same characteristics go in the same bucket. Buckets are placed in prioritized order for dispatching. This prioritization is based on FairShare, an allocation system. The top priority job in each bucket is dispatched when each of the defined resources (requests) for that job is available. The job requests can be fulfilled from the central pool as well as the tasker resources. When a tasker is found that completes the job's resource request, the job is dispatched to that tasker and the job status changes to running.

When the job has completed, the tasker notifies the vovserver. The resources, both tasker-based and central, are recovered, which allows subsequent jobs (queued in the buckets) to be dispatched. When completed, the job status is normally updated to either valid or failed.

As previously stated, in addition to dispatching jobs and processing their statuses, the vovserver responds to queries about system and job requests, publish events to notify clients, and continue to process incoming job requests.

Known Limitations

In the Windows environment, PowerShell is not supported; it is strongly recommend to avoid using PowerShell.

Related Documents

The following documents provide additional information that is related to using and configuring Accelerator:

- Altair Accelerator User Guide
- Altair Accelerator Training Guide
- Altair Accelerator Installation Guide
- Altair Monitor User Guide
- VOV Subsystem Reference Guide

Use Accelerator Help

Accelerator documentation is available in HTML and PDF format.

Access the Help when Accelerator is Running

When Accelerator is running, it displays the documentation through its browser interface. To access it from browser, you need to know which host and port Accelerator is running on. Ask your administrator, or find the URL for Accelerator with the following command:

```
% Accelerator cmd vovbrowser
http://comet:6271/project
```

In the example below, assume Accelerator is running on host comet, port 6271. The URL for Accelerator is:

```
http://comet:6271
```

To get the entire suite of Altair Accelerator documents, including FlowTracer™, Accelerator™, Monitor™ and the VOV subsystem, use the following URL:

```
http://comet:6271/doc/html/bookshelf/index.htm
```

Access the Help when Accelerator is not Running

All the documentation files are in the Altair Accelerator install directory, so you can access them even if vovserver is not running. To do this, open `/installation_directory/common/doc/html/bookshelf/index.htm` in your browser.

Tip: Bookmark the above URL for future reference.

Access the Help PDF Files

Altair Accelerator also provides PDF files for each of the guides. All the PDF files are in the directory `/installation_directory/common/doc/pdf`

Access the Help via the Command Line

The main commands of Accelerator are nc and ncmgr, with some subcommands and options. You can get usage help, descriptions and examples of the commands by running the command without any options, or with the -h option. For example,

```
% nc info -h
nc:
nc:  NC INFO:
nc:  Get information about a specific job or list of jobs.
nc:  USAGE:
nc:  % nc info <jobId> [options]...
nc:   -h      -- Show this message
nc:   -l      -- Show the log file
nc:
```

Access the Help via the vovshow Command

Another source of live information is using the command `vovshow`. The following options are often useful:

`vovshow -env RX`	Displays the environment variables that match the regular expression RX provided.
`vovshow -fields`	Shows the fields known to the version of VOV in use.
`vovshow -failcodes`	Shows the table of known failure codes.

For example, to find a variable that controls the name of the stdout/stderr files, without knowing the exact name of that variable, the following command can be used:

```
% vovshow -env STD
VOV_STDOUT_SPEC          Control the names of file used to save stdout and
                         stderr. The value is computed by substituting
                         the substrings @OUT@ and @UNIQUE@ and @ID@.
                         Examples: % setenv VOV_STDOUT_SPEC
                         .std@OUT@.@UNIQUE@ % setenv VOV_STDOUT_SPEC
                         .std@OUT@.@ID@
```

The output provides a description of all the variables used by the FlowTracer system that include the substring "STD". In this example, the output resultVOV_STDOUT_SPEC.

Accelerator Quick Start

Accelerator has two main commands, `nc` and `ncmgr`.

- `nc` is used to submit, query, and stop jobs. This command can also be invoked as `vnc`.
- `ncmgr` is used to start a queue: `ncmgr start`. By default, the queue (`vnc`) starts in a server working directory (SWD) that is a subdirectory in `$VOVDIR/../../vnc`.

The output of `ncmgr` start/stop is logged in `${VOVDIR}/local/logs/nc`, if it exists.

This page shows the usage messages that are generated by the `nc` and `ncmgr` commands.

nc

```
vnc: Usage Message
  Usage: nc [-q queuename] <command> [command options]

  Queue selection:
    The default queue is called "vnc".

    You can specify a different queue with the option -q <queuename>
    or by setting the environment variable NC_QUEUE.

  Commands:
    clean                Cleanup log files and env files.
    debug                Show how to run the same job without Accelerator.
    dispatch             Force dispatch of a job to a specific tasker.
    forget               Forget old jobs from the system.
    getfield             Get a field for a job.
    gui                  Start a simple graphical interface.
    help                 This help message.
    hosts                Show farm hosts (also called taskers).
    info                 Get information about a job and its outputs.
    list                 List the jobs in the system.
    jobclass             List the available job classes.
    kerberos             Interface to Kerberos (experimental).
    modify               Modify attributes of scheduled jobs.
    monitor              Monitor network activity.
    rerun                Rerun a job already known to the system.
    resources            Shows resource list and current statistics.
    resume               Resume a job previously suspended.
    run <job>            Run a new job (also called 'submit').
    preempt              Preempt a job.
    stop                 Stop jobs.
    submit <job>         Same as 'run'.
    summary              Get a summary report for all my jobs.
    suspend              Suspend the execution of a job.
    taskerlist           Show available tasker lists.
    wait                 Wait for a job to complete.
    who                  Report on who is using the system.
    why                  Analyze job status reasons.


  Unique abbreviations for commands are accepted.

  Advanced features:
    cmd <command>        Execute an arbitrary VOV command in the
```

```
                    context of the $product server.
    source <file.tcl>   Source the given Tcl file.
    -                   Accept commands from stdin.

  For more help type:
    % $::command <command> -h

  Copyright (c) 1998-2021, Altair Engineering.
```

ncmgr

This program manages the vovserver for Accelerator.

Usage

```
vncmgr: Usage Message

        This program manages the vovserver for Accelerator.
        Copyright (c) 1998-2022, Altair Engineering.

USAGE:
        ncmgr help|info|rehost|reset|start|stop|cm [OPTIONS]

ACTIONS:
        info   [-queue|-q <name>] [-v]
        reset  [-soft | -hard | -h ]
        rehost [-force] [-queue|-q <name>] -host <host>
        start  [-dir <server_working_dir>] [-force] [-queue|-q <name>]
               [-port <port> ] [-webport <port>] [-roport <port>]
               [-dbhost <host>] [-dbroot <path>] [-dbport <port>]
               [-prod nc|wx|he] [-basequeue <name>] [-dd]
               The default <server_working_dir> is
               <...>/vnc.
               This is the parent of the configuration (.swd) directory for
               the queue.
        stop   [-force] [-freeze] [-freeze_nocpr] [-queue|-q <name>]
               [-writeprdir <dirname>]
               -force        Do not prompt for confirmation
               -freeze       Instruct taskers to keep running and wait for a
                             new server
               -freeze_nocpr Instruct taskers to keep running and wait for a
                              new server, and do not compress PR file
               -writeprdir  Writes the PR file to the specified directory
               (which is created if necessary)
        cm     [-queue|-q name] <ACTION> [ARGUMENTS]
               Configuration Management. Pass "help" for detailed usage.

EXAMPLES:
        % ncmgr
        % ncmgr -h
        % ncmgr start -queue vnc2
        % ncmgr start -port 6699 -queue vnc99
        % ncmgr info
        % ncmgr reset -soft
        % ncmgr reset -hard
        % ncmgr cm help
```

```
EXAMPLE TO STOP AND RESTART SERVER:

        % ncmgr stop -freeze
        % ncmgr start -force
        % ncmgr stop -freeze -force -writeprdir /tmp/abc123
```

Command Line Interface

All user commands have the following structure:

```
% nc [-q qname] subcommand [options]
```

The command plus the subcommand is one of the following:

- nc clean
- cmd
- nc debug
- dispatch
- nc forget
- nc gui
- help
- nc hosts
- nc info
- nc list
- nc jobclass
- nc modify
- monitor
- nc rerun
- resume
- nc run
- source
- nc stop
- nc summary
- suspend
- nc wait

For example:

```
% nc help
% nc run sleep 10
% nc list
% nc forget -mine
```

The Exclamation Point (!) Special Operator

Some Accelerator subcommands accept a single exclamation point, and interpret it to mean 'most-recent job run in the current directory'. This is meant for interactive use to avoid typing or copying the nine digit job ID.

This is not recommend for use in scripts, because it involves a scan of the jobs in the system. Instead, save the job ID returned when submitting the job and use the ID in queries.

The Accelerator subcommands that support this are:

- `info`
- `getfield`
- `rerun`

For example:

```
% nc info !
% nc info -l !
```

Some VOV commands that support this may sometimes be useful in Accelerator context, by preceding them with `nc cmd`:

- `vovset`
- `vovfire`
- `vsx`, `vsy`

Any unique prefix for the subcommand is accepted, which allows abbreviated forms of commands to be used. For example:

```
% nc l
% nc li
% nc lis
% nc list
```

Quick Reference

Common Commands

help	Getting Accelerator help.
nc forget	Remove jobs from Accelerator queue.
nc getfield	Get detailed information on a job.
nc info	Get information on jobs.
nc list	Get a formatted list of jobs.
nc modify	Modifying jobs in the system.
monitor	Monitoring jobs and tasker machines.
notify	Email notification.
policy	Setting policy.
nc rerun	Re-running jobs.
nc run	Submitting jobs in Accelerator.
interactive	Running interactive jobs in Accelerator.
status	Getting status info in Accelerator.
nc stop	Stopping jobs in Accelerator.

Information Pages

Altair Accelerator User Guide	Introduction to Accelerator.
Installation Guide	Installation of Accelerator.
Manage	Managing Accelerator.
Test	Testing Accelerator after installation.
Troubleshoot	Troubleshooting Accelerator.

Administrative

Advanced Information	Advanced command usage.
Clean Up Log Files	Clean up log files.
Cross-platform	Cross-platform runs.

Environment Control	Controlling the VNC environment.
FairShare Groups	Setting up groups.
vnc_policy.tcl	VOV policy setup file.
Resource Management	Accelerator resource management.
Scheduled Jobs	Accelerator job queue.

Accelerator Server and System Customization

Configure Accelerator

Server Configuration

vovserver configuration parameter values may be changed in a running vovserver using the CLI, or prior to the starting the server via the `policy.tcl` file. An administator can configure the parameters in the running vovserver using the `vovservermgr` command or the `vtk_server_config` procedure.

Note: In usage, all commands and parameters are case insensitive.

Using `vovservermgr`:

```
% nc cmd vovservermgr config PARAMETER_NAME PARAMETER_VALUE
```

Using `vtk_server_config`:

```
% nc cmd vovsh -x 'vtk_server_config PARAMETER_NAME PARAMETER_VALUE'
```

Example of a configuration:

```
% nc cmd vovsh -x 'vtk_server_config timeTolerance 4'
% nc cmd vovsh -x 'vtk_server_config timeTolerance 4'
```

Note: A complete list of the current server configuration parameters is provided in the Server Configuration page.

Server configuration can be controlled by setting variables in the `policy.tcl` file. The variable can be set directly in the "config()" associative array, but it is best to set them with the procedure `VovServerConfig` as in:

```
VovServerConfig VARNAME VALUE
```

In either case, the name of the parameter *VARNAME* is case insensitive.

Example

The following is part of the `policy.tcl` file.

```
# This is part of the policy.tcl file.

# Example of parameters set with the procedure VovServerConfig
VovServerConfig readonlyPort  7111
VovServerConfig httpSecure       1

# Example of parameters set by assignment to array config().
set config(timeTolerance)        0
set config(maxBufferSize)       16000000
set config(maxNotifyBufferSize)  400000
set config(maxNotifyClients)     40;
set config(maxNormalClients)     400;

set config(maxAgeRecentJobs)     60;
set config(saveToDiskPeriod)     2h;
set config(autoShutdown)         2w;      # Shut down after 2 weeks of inactivity.
set config(autoLogout)           1h;      # Logout from browser interface.
set config(netInfo)              0;       # Do not collect net information (fs,
 hosts)

# Used by Accelerator for autoforget.
set config(autoForgetValid)      1h
set config(autoForgetFailed)     2d
set config(autoForgetOthers)     2d

set config(autoRescheduleThreshold) 2s

set config(preemptionPeriod) 3s
```

Below is an example of parameters set with the procedure `VovServerConfig`:

```
VovServerConfig readonlyPort  7111
VovServerConfig httpSecure       1
```

Tasker Configuration

Taskers

A tasker is a VOV client that provides computing resources, specifically CPU cycles, to the vovserver.

There are two types of taskers:

- **Direct taskers**: agents that offer for computation all the resources of the machine on which they are running
- **Indirect taskers**: agents that interface between a VOV project and a scheduler such as Accelerator.

A project can have a mix of direct and indirect taskers. Normally, Accelerator and Monitor use only direct taskers, while FlowTracer projects often interface to schedulers using one or more indirect taskers.

The list of taskers connected to a project is described in the file taskers.tcl, and the main utility to start and stop taskers is `vovtaskermgr`. Additional configuration can be specified with the `taskerClass.table` file.

Types of vovtasker Binaries

The main tasker client is called `vovtasker` but there are other variations of it:

- `vovtasker` can run jobs for more than one user; the success depends on file permissions.
- `vovtaskerroot` has the ability of switching user identity.

> **Note:** Accelerator is the only Altair Accelerator product that needs `vovtaskerroot`.

- vovtasker.exe has the ability of impersonating users on Windows, subject to a set of rules explained in *Vov Windows Impersonation*.
- `vovagent` is a temporary vovtasker that terminates upon a set of timeouts, and is used mostly in conjunction with LSF or SGE.
- `ftlm_agent` is a "thin-client" version of a vovtasker and can be used by Monitor to start and stop license daemons on remote machines.

vovtasker States

The vovtaskers will change states based on workload and operating environment. Generally, the tasker state will accompany the tasker name when displayed in the various user interfaces (CLI, GUI, WUI). The possible states are as follows:

- `BLACKHOLE`: temporarily paused due to a burst of job failures and cannot accept jobs
- `BUSY`: busy with internal operations
- `DEAD`: has disconnected and cannot accept jobs
- `DONE`: exiting after completing current jobs
- `FULL`: full and cannot accept more jobs
- `OVRLD`: overloaded and cannot accept jobs
- `NOLIC`: unlicensed and cannot accept jobs
- `NOSLOT`: configured to not accept job
- `OK`: idle and ready for jobs
- `PAUSED`: paused and cannot accept jobs
- `READY`: idle and ready for jobs
- `REQUESTED`: has been requested to start
- `SICK`: sick and possibly disconnected
- `SUSP`: suspended and cannot accept jobs
- `WARN`: in a warning state and should be checked
- `WRKNG`: working and can accept more job

Create a Tasker on Windows

1. Copy the vovtsd single file distributable (SFD) for Windows (`vovtsd.exe`) onto the Windows host that will be running one or more taskers.
2. Start `vovtsd` (use default port or specify a different one). Optionally, configure as a Windows service (see vovtsd documentation).
3. Register the Windows path to the server working directory. The server working directory is the parent of the `<queue-name>.swd` directory (aka SWD). The default queue name is “vnc”, so the default SWD is named `vnc.swd`. Registration is done via the `SWD/policy.tcl` file.

   ```
   vtk_swd_set windows <full path to server working directory>
   ```

   For example:

   ```
   vtk_swd_set windows n:/altair/vnc
   ```

4. After making the change, reread the policy via:

```
nc cmd vovproject reread
```

5. Update the following in `SWD/taskers.tcl`:

```
vtk_tasker_set_defaults \
    -rshcmd     vovssd \
    -vovssdport 16666 \
    -executable vovtasker \
    -vovdir     default \
    -logfile    default \
    -serverdir  n:/altair/vnc

vtk_tasker_define <windows host name>
```

   The default tasker name will be equal to the host name.

6. Start the newly defined tasker:

```
nc cmd vovtaskermgr start <tasker name>
```

7. Verify the tasker using this command:

```
nc hosts
```

   The Windows tasker will be remotely started via vovtsd and will enter a “ready” state once it is able to accept jobs:

```
# TASKER   LOAD   STATUS  JOBS  HB   RESERVE   MESSAGE
1 win16-1   0.00   ready   0/2   20s  None      Licensed for 2 slots
```

8. Submit the job:

```
nc run -e BASE -rundir <valid windows path> -l "'<log file in valid windows
 path>'" -r ARCH=win64 -- <cmd>
```

   For example:

```
nc run -e BASE -rundir c:/ -l "'c:/@JOBID@.log'" -r ARCH=win64 -- hostname
```

   The @JOBID@ keyword will be automatically substituted with the job’s ID, resulting in a uniquely-named log file.

UNIX User Impersonation

A normal vovtasker has the privileges and limitations of the account in which it runs. When a normal vovtasker executes a command, the effect is the same as when the vovtasker's owner runs the command.

For Accelerator, jobs may be submitted by many users. To obtain the correct permissions with respect to files needed by the job, the vovtasker must be able to switch to the account of the submitter. On UNIX and Linux machines, this is done by having vovtasker run with root privilege. A simple method is creating a copy of vovtasker called vovtaskerroot, which is owned by root and has the setuid flag set. vovtaskerroot can be set up by running the script `SETTASKERUID.csh`.

To verify that vovtaskerroot is properly installed, go to the `$VOVDIR/bin` directory and check the access flags of vovtaskerroot. Verify the "s" character is in the fourth column (instead of the "x" character) as shown below:

```
% cd $VOVDIR/bin
% ls -l vovtaskerroot
   -rwsr-sr-x   1 root    other     875092 Jan  5 11:57 vovtaskerroot
```

For security reasons:

- The `USER` field for each job cannot be edited by any user, including `ADMIN`.
- The vovtasker itself never executes any process as root. Instead, each job is executed as the user that first executed the job. If the user account does not exist on the tasker host, the job cannot be executed.

Other Methods of Starting vovtasker with Root Capability

Some sites may have security policies that prohibit setuid-root binaries, or prohibit setuid binaries from being mounted over NFS. Following are alternate methods to start vovtasker with root capability:

- To start vovtasker from a boot file, use the available example `.bat` files as a guide to create a script, and place it in the appropriate directory. Example startup files are provided in `$VOVDIR/etc/boot`. Choose the one that best fits your scenario.
- Use a setuid vovtaskerroot on a local disk.

Use a setuid vovtaskerroot Binary on a Local Disk

If the NFS filesystem (including the Altair Accelerator) is exported with the nosuid option, the `vovtaskermgr` command can still be used via a local setuid-root binary on each farm host.

Note: This method will add the cost of having to update the hosts individually when changing versions.

For example, the binary can be put at `/opt/rtda/some-version/linux/bin/vovtaskerroot`. It is helpful but not necessary if this path is the same on all farm hosts.

In this case, the setuid-root binaries must be created manually (the regular script is not useful. Following is an example of creating the setuid-root binary:

```
% ssh some-farm-host% /bin/su -
# cd /opt; mkdir -p rtda/CURRENT/linux/bin
# cd /opt/rtda/CURRENT/linux/bin
# cp /network-path-to-rtda/CURRENT/linux/bin/vovtasker ./vovtaskerroot
# chown root: ./vovtaskerroot
# chmod u+s ./vovtaskerroot
```

After creating the local vovtaskerroot, set up the taskers configuration file of the Accelerator to use the local vovtaskerroot. The file is located in `$VOVDIR/../../vnc/vnc.swd/taskers.tcl`.

If the path to the local vovtaskerroot binary is the same on all farm hosts, change the defaults at the beginning of the file as shown below:

```
if { [file executable /opt/rtda/CURRENT/linux/bin/vovtaskerroot] } {
    # Use vovtaskerroot from the local disk
    vtk_tasker_set_default  -executable /opt/rtda/CURRENT/linux/bin/vovtaskerroot
} else if { [file executable $env(VOVDIR)/bin/vovtaskerroot] } {
    vtk_tasker_set_default  -executable vovtaskerroot
} else {
    vtk_tasker_set_default  -executable vovtasker
}
```

If the path to the vovtaskerroot varies from host to host, add the -executable option to the definition for each host (instead of changing the default).

The taskers.tcl File

The `taskers.tcl` file describes the taskers for a project.

The `taskers.tcl` file is a Tcl script based on the procedure `vtk_tasker_define`. The synopsis for this procedure:

```
vtk_tasker_define hostname [options]
```

The two following examples both declare three taskers on the hosts apple orange and pear:

```
# Fragment of taskers.tcl file
vtk_tasker_define apple
vtk_tasker_define orange
vtk_tasker_define pear
```

```
# Fragment of taskers.tcl file
foreach host {apple orange pear} {
    vtk_tasker_define $host
}
```

The following procedure supports many options to define the characteristics of the tasker. The options include "-resources <string>" to set the resource list offered by a tasker and -CPUS n to define the number of CPUs in a machine. In the following example, the tasker on apple is set up to offer the resource "big_memory":

```
# Fragment of tasker file
vtk_tasker_define apple  -resources "@STD@ big_memory" -CPUS 2
vtk_tasker_define orange -resources "@STD@ big_memory" -CPUS 2
vtk_tasker_define pear   -resources "@STD@ big_memory" -CPUS 4
```

The default value for all options can be changed with the following procedure `vtk_tasker_set_defaults`, as shown below:

```
# Fragment of taskers.tcl file
vtk_tasker_set_default -resources "@STD@ big_memory" -CPUS 2
vtk_tasker_define apple
vtk_tasker_define orange
vtk_tasker_define    -CPUS 4
```

vtk_tasker_define

There are many options that can be used with vtk_tasker_define and vtk_tasker_set_default.

Option	Argument	Description
-capabilities	`string`	The capabilities that the tasker has. This results in license checkout attempts for the specified capabilities. Possible values: FULL, NC, PROCINFO, NETINFO, EXEC, RT. FULL includes all capabilities. NC contains EXEC, PROCINFO and NETINFO capabilities. Altair Accelerator enables runtime tracing for FlowTracer. Default is FULL.
-capacity	`int`	The number of concurrent jobs (job slots) that the vovtasker can handle.
-cpus	`int`	The number of CPUs in the machine, without affecting maxload and capacity. On most platforms, the number of CPUs is computed automatically.
-CPUS	`int`	Convenience option, equivalent to setting -cpus, -maxload and -capacity at the same time. If *N* is the number of CPUs, this options sets -cpus to *N*, -maxload to *N+0.5* and -capacity to *N*.
-coeff	`double`	The tasker coefficient. It is used as a divisor in computing the effective power of a vovtasker, e.g. a coefficient of 2.0 reduces the power by half.
-executable	`string`	The executable to use (default is vovtasker). For Accelerator, the default is vovtaskerroot.
-expiredate	`string`	Specifies the date and time after which the definition of this tasker is expired, and it cannot be started with vovtaskermgr command. The format of this parameter is year_month_day_hour_min_sec. Example: 2018_12_31_23_59_00
-failover		Passing this option will set the tasker's capacity to 0, which prevents the tasker from accepting jobs and pulling a license. This also acts as a flag to perform some failover configuration testing, such as checking `servercandidates.tcl` to make sure the tasker host is in the list, triggering a check to make sure the host has at least as many file descriptors as vovserver so it can operate at full capacity in the event of failover, and checking that the `server_election` directory is empty.
-host	`hostname`	The name to be used to connect to the vovserver (e.g. "localhost")
-indirect	`file`	Execute the jobs indirectly, using the Tasker* procedures described in the given file. This is used in indirect taskers.

Option	Argument	Description
-maxcapacity	`int`	The capacity of taskers can increase dynamically as a side effect of having suspended jobs. This limits the maximum capacity. The default value is twice the capacity.
-maxjobs	`int`	The maximum number of jobs a tasker can execute. Taskers will become suspended when the max is reached, and will exit once the last job is finished. Default: 0 (unlimited).
-maxidle	`timespec`	The maximum amount of time a tasker can be idle (having no jobs). Default: - (unlimited).
-maxlife	`timespec`	The maximum amount of time a tasker is allowed to run. Default: - (unlimited).
-maxload	`double`	The maximum load for the vovtasker. Above this, its power becomes zero, and the vovtasker does not accept new jobs until the load declines below this value. This helps avoid overloaded machines.
-message	`string`	Message to set on the vovtasker at startup. Should be brief.
-mindisk	`number`	Minimum disk space, in MB (for example, 100) or in percentage (0%-99%, for example, 10%) , on `/usr/tmp` below which the vovtasker will automatically be suspended.
-name	`string`	Name of the vovtasker. The default is the leaf name of the machine on which the vovtasker runs. May not contain the '.' (dot) character.
-nice	`int`	Run the tasker with niceness (reduced OS priority), for UNIX vovtasker only, ignored on Windows.
-power	`double`	The raw power to be used for this vovtasker. The default for this is 0.0, which implies that the raw power is computed automatically upon startup. You can use this to make machines know to be identically- provisioned to have the same power.
-repeat	`int`	Number of identical vovtasker on a host (obsolete).
-reserve	`reserve expression`	Reserve the vovtasker upon startup. The argument is a reservation expression. Example 1: `"-reserve /john/3w"` This means create a reservation for user john with a duration of three weeks at vovtasker startup.

Option	Argument	Description
		Example 2: `"-reserve memregr//4h"` This means create a reservation for group 'memregr' with a duration of 4h at vovtasker startup.
-resources	`string`	The tasker resources. This could be a list of literals like "bighost maingroup" or contain symbolic values like "@RAM@ @CPUS@". You must restart vovtaskers after changing values specified here. For a method that does not require restart, read about The taskerClass.table file.
-rshcmd	`string`	The command used to start a remote shell on the tasker machine. This is rsh by default, but it could be set, for example, to ssh. The known values for this option are: • rsh, the default value • ssh, the typical value • `vovtsd`
-serverdir	`dir`	Explicit path to the server directory for the tasker.
-taskergroup	`group`	Define the taskergroup in `taskers.tcl`. Often this is used to group similarly-provisioned machines.
-update	`timespec`	The update cycle time (heart beat) of the vovtasker. The default value is 60s. You can use shorter values to cause resources to be updated more frequently when using resource procedures, being mindful of the CPU load this brings to the vovserver.
-vovdir	`dir`	Explicit path to the VOV installation for the tasker.
-vovtsdport	`port`	Port number to be used when connecting to the VOV tasker service daemon, `vovtsd`.

Tasker Attributes

A tasker is characterized by several attributes. These attributes are controllable by means of the command line arguments to the vovtasker binary as well as by means of the procedure vtk_tasker_define in the `taskers.tcl` configuration file for a VOV project.

For cases where the vovtaskers are started by submitting the binary to a separate batch queue system, the system manager may create a copy of the vovtasker binary called `vovagent`. When invoked by this name, the binary will limit the values of some attributes based on information stored in the configuration file `$VOVDIR/local/vovagent.cfg`.

Attribute Name	vovtasker options	vtk_tasker_define option	Description
name	-a	-name	Name of the tasker. By default, it is the name of the host on which the tasker is running. The tasker name can contain alphanumeric characters, dashes and underscores. It must be less than 50 characters long. If the name ends with a "_r", it probably indicates a tasker that has reconnected to the tasker after a server crash. These taskers are used to terminate the jobs executing at the time of the crash. The name of a running tasker can be changed with `vovtaskermgr configure -name ...` or using the API `vtk_tasker_config $TASKERID name "newname"`.
capability	-b	-capabilities	The capabilities that the tasker has. This results in license checkout attempts for the specified capabilities. Possible values: FULL, NC, PROCINFO, NETINFO, EXEC, RT. FULL includes all capabilities. Accelerator contains PROCINFO and NETINFO capabilities. Default is FULL.
capacity	-T	-capacity	Maximum number of jobs that can be run by the tasker concurrently. Default is 1 slot per core detected or specified (see -C below).
maxload	-M	-maxload	Maximum allowed load on the tasker host (default N+0.5, where *N* is the number of cores detected). The maximum load is the point at which the host of the tasker is too busy to accept any more jobs. A machine is considered overloaded if its load average for either the last minute or the last five minutes exceeds this boundary.
loadsensor	-L	-loadsensor	Use an SGE style load sensors to control power of a tasker and other resources.
coefficient	-c	-coeff	The tasker coefficient (positive floating point number, with default 1.0). This attribute is used to adjust the raw power. A coefficient of 1.0 indicates the actual computed power of the tasker should be used. A coefficient of 4.0 indicates the actual power of the tasker should be divided by 4.

Attribute Name	vovtasker options	vtk_tasker_define option	Description
cpus	-C	-cpus	The number of CPUs in this machine. Default is the number of cores reported by the operating system. This attribute affects the computation of the actual load on the machine and by default, defines the capacity of the tasker (see -T above).
logfile	-l	-logfile	The name of the file for the tasker log. The file name is relative to the server working directory. The file name can contain also the following symbolic strings, which will be appropriately substituted: @NUMBER@ @TASKERHOST@ @SERVERHOST@ @PROJECT@.
reserve	-e	-reserve	The reservation expression for this tasker. The argument is in the format "GROUP/USER/DURATION", where the GROUP and USER fields are optional. Examples: `/john/2wusers//100dregression/john/2w`
resources	-r	-resources	The tasker resources offered by this tasker. The resource management determines the type of jobs the tasker may accept.
Remote Shell Command	n/a	-rshcmd	The command used to start a remote shell tasker on the tasker machine. The default is `rsh` (or `remsh`). Other possible values are: • `ssh` (most common value) • `vovtsd` (useful for Windows taskers), which requires `vovtsd` to be running on the remote machine.
Tasker Environment	n/a	-taskerenv	A space-separated list of VAR=VALUE elements that specify additional environment variables that need to be defined when starting the vovtasker. This parameter is only active when the tasker is started. To change the environment in a running vovtasker you have to use the vtk_tasker_config API.
Tasker Group	n/a	-taskergroup	The group(s) the tasker is in for purpose of viewing in the browser UI or the vovmonitor. This currently has no use other than making it easier to view groups of taskers.

Attribute Name	vovtasker options	vtk_tasker_define option	Description
timeleft	N/A	N/A	The time the tasker has left before it suspends itself. Also, the maximum expected duration for a job dispatched to this tasker. This attribute is available in the context of time variant resources and is controlled only by means of the procedure vtk_tasker_set_timeleft.
transient	-i	-transient	A transient tasker is destroyed when the vovtasker client is terminated. On the other hand, a non-transient tasker persists in memory even when vovtasker is terminated. Its status will be "DOWN".
Use Vovfire	-E	-usevovfire	With this option, a direct tasker may use `vovfire` to execute jobs instead of the direct execution of the job. Since `vovfire` does the directory change and the setting of the environment, the tasker does less work. The environment caching of the tasker is disabled in this mode. This is a development option that is useful mostly on Windows.
update	-U	-update	The period used by the tasker to update its status. The argument is in seconds. The default is 60 seconds.
mindisk	-D	-mindisk	The amount of free disk on `/usr/tmp` below which the vovtasker is suspended. The default is 5MB. Many system commands and some VOV ones depend on scratch space here. **Note:** The value can be set to 0 to turn off tasker suspension, but incorrect operation of some commands may occur.
maxidle	-z	N/A	After being idle for the given time, tasker does not accept new jobs and exits after completing active jobs. Value is a FlowTracer timespec, for example, 2m. If unspecified, idle time is unlimited.
maxlife	-Z	N/A	After the specified lifetime, tasker does not accept new jobs and exits after completing active jobs.

Attribute Name	vovtasker options	vtk_tasker_define option	Description
			Value is a FlowTracer timespec, for example, 2H. If unspecified, lifetime is unlimited.
maxjobs	-m	-maxjobs	The maximum number of jobs a vovtasker will execute during its lifetime. When the max is reached, vovtasker self-suspends so it stops accepting new jobs, and will exit once its last running job finishes. Default: 0 (unlimited).

Manage Tasker Lists

A tasker list is a named, ordered list of taskers. Tasker lists can be used to enhance performance or restrict usage.

Examples include:

- Make the scheduler more efficient on large farms.
- Pack jobs more tightly on the farm machines.
- Restrict jobs to selected taskers.

Every vovserver has at least one tasker list named *default*, which includes all the taskers in the system. The order of the taskers is determined by the connection order unless modified by the commands shown below. The default list cannot be deleted or recreated; taskers can be reordered in the default list.

A tasker list can contain all taskers or a subset of taskers. A tasker list can also be empty.

Note: Typically, a small number of tasker lists are setup for a system, approximately 20. A large number of tasker lists may result in inefficient searches for available resources.

An administrator can create, modify and delete tasker lists with the utility `vovtaskerlist`.

vovtaskerlist

Manipulate tasker lists.

```
vovtaskerlist: Usage Message

  DESCRIPTION:
      Manipulate tasker lists.

  USAGE:
      % vovtaskerlist ACTION [OPTIONS]

  OPTIONS:
      -h                    -- This help
      -v                    -- Increase verbosity
```

```
     Actions:
      create                 -- Create a new list
      append
      first
      last
      get                   -- Get taskers in a list
      list                  -- List all tasker lists
      delete                -- Delete specified tasker list
      bigram                -- Create list of taskers with a lot of ram
      smallram              -- Create list of taskers with little ram

     Note: actions can also have a dash (list and -list)

  EXAMPLES:
      % vovtaskerlist create planets "pluto jupiter uranus"
      % vovtaskerlist get  planets
      % vovtaskerlist list
      % vovtaskerlist first planets jupiter
      % vovtaskerlist last planets jupiter
      % vovtaskerlist delete  planets
      % vovtaskerlist -bigram 10000      ;; Make a list of taskers with more
                                           than 10GB of RAM
      % vovtaskerlist -smallram 2000     ;; Make list of all taskers with less
                                           than 2GB of RAM
```

nc taskerlist

Any user can view the tasker lists with the nc taskerlist command.

```
vnc: Usage Message


      NC TASKERLIST
          Support tasker lists.
          Taskerlists are named, ordered lists of taskers.

      USAGE:
          % nc taskerlist [OPTIONS]

      OPTIONS:
          -h        -- This help
          -v        -- Increase verbosity,
          -list     -- List all available taskerlists
          -get LIST -- Get the ordered taskers in the
                       specified LIST

      ADDITIONAL INFO:
        To manage the tasker lists, you have to be ADMIN
        and you can use the vovtaskerlist utility.


      EXAMPLES:
          % nc taskerlist -list
          % nc taskerlist -get NAMEOFLIST
          % nc run -r TaskerList:NAMEOFLIST ...  -- myjob
```

Create and Delete Taskerlists

The following example creates a taskerlist named `big` that contains five machines:

```
% vovproject enable vnc
% vovtaskerlist create big "lnxbig01 lnxbig02 lnxbig03"
% vovtaskerlist append big "lnxbig04 lnxbig05"
```

To delete the list, use the following command:

```
% vovtaskerlist delete big
```

A list can be reordered by moving selecting taskers `first` or `last`: to the beginning or to the end of the list, respectively. These commands can also be used on the default list.

```
% vovtaskerlist first big lnxbig05
% $VOVDIR last  big lnxbig01
```

The vtk API for Tasker Lists

```
vtk_taskerlist_create LISTNAME "list of taskers"
vtk_taskerlist_append LISTNAME "list of taskers"
vtk_taskerlist_first  LISTNAME "list of taskers"
vtk_taskerlist_last   LISTNAME "list of taskers"
vtk_taskerlist_get    LISTNAME
vtk_taskerlist_delete LISTNAME
vtk_taskerlist_list
```

The following example uses the vtk AP to create a vovtaskerlist that is in the reverse order of the default list.

```
% vovsh -x 'vtk_taskerlist_create reverse [lreverse [vovtaskerlist_get default]]'
```

Choose a Tasker List for a Job

To choose a tasker list, use the resource "TaskerList:LISTNAME" when submitting the job.

```
% nc run -r TaskerList:big -- sleep 100
```

If the specified list does not exist, the job will not run.

Control the Capacity of Taskers: Slots and Cores

The resources of a tasker include SLOTS, SLOTSTOTAL, CORES and CORESTOTAL.

For example, a machine with 8 cores is normally assigned the following resources:

CORESTOTAL#8	The physical number of cores in the machine.

CORES/8	A consumable resource to represent the number of available cores on the tasker.
SLOTSTOTAL#8	The total number of slots available. Most jobs use only 1 slot, although you can submit jobs that request more than 1 slot.
SLOTS/8	A consumable resource indicating the number of available slots in a tasker.

A simple single-threaded job consumes 1 slot and 1 core. A multi-threaded job may consume 2 or more cores, but conventionally it is assumed that each job consumes only 1 slot.

Persistent Capacity Configuration

The tasker capacity is normally configured prior to a tasker starting. Methods are shown below:

- -capacity or -cpus option to `vtk_tasker_define`
- -T or -C option to `vovtasker`
- Including SLOTS/N in the tasker resource specification or the `vovtasker` command line, where N is a positive integer number.

Refer to Tasker Attributes for more details about these methods.

Live Capacity Configuration

The behavior of manually overriding vovtasker cores and capacity has been improved. By default, the capacity follows the core count, but it can also be manually set via the -T option or by defining the SLOTS/N consumable resource via the -r option, where N is a positive integer. In all cases, the capacity directly affects the number of slot licenses that will be requested.

On occasion, it can be useful to change the number of jobs that are allowed to run on a tasker while it is live. For example, if you have an 8-core machine and you only want to run 4 jobs on it, you can configure the tasker on the fly as shown below:

```
% nc cmd vovtaskermgr configure -resources SLOTS/4 lnx123
```

Note: For backwards compatibility, the option -capacity in `vovtaskermgrconfigure` is still supported. This option is a shortcut method of setting SLOTS SLOTSTOTAL CORES and CORESTOTAL all to the same amount.

```
% nc cmd vovtaskermgr configure -capacity 4 lnx123
```

is the same as

```
% nc cmd vovtaskermgr configure -resources "SLOTS/4 SLOTSTOTAL#4 CORES/4
 CORESTOTAL#4" lnx123
```

Refresher: Submit Multi-threaded Jobs

A multi-threaded job consumes more than one core. In a 4-threaded job, a complete submission could look like this:

```
% nc run -r SLOTS/1 CORES/4 License:abc -- my_mt_job
```

If it is also important to count the multiple cores towards FairShare, the -fstokens 4 option can be used:

```
% nc run -r SLOTS/1 CORES/4 License:abc -fstokens 4my_mt_job
```

Stopped Tasker's Effect on a Newly Started Tasker

When a tasker has been requested to stop gracefully (allowing jobs to finish before exiting), it is suspended from scheduler consideration and its capacity is set to 0. If the same tasker is started on the same host before the stopped tasker exits, the slots consumed by the still-running jobs on the stopped tasker will be counted and deducted from the total slot capacity of the newly started tasker. This helps prevent overloading the machine on which the taskers are running.

Tasker Health Checks

By default, the vovtasker does not perform many health checks. If the jobs can be executed successfully, that is normally enough of a check.

In some large farms, it may be useful to activate additional checks, by invoking the tasker with the option `-H` [PpDdWwUu], where the lower case characters mean "disable" and the upper case characters mean "enable" of a particular check.

Built-in Checks in vovtasker

Check Name	Char	Description
DiskSpace	d	Check space in `/tmp` and `/usr/tmp`
WritePerm	w	Check write permissions in `/tmp` and `/usr/tmp`
Portmap	p	Check that the portmap daemon is responsive
UserScript	u	Check by executing an adminstrator-created, installation-wide script (see below)

From the command line, you can use the option `-H` in vovtasker, as in the following example, which activates only the DiskSpace test:

```
% vovtasker -H Dwpu -a my_test_tasker
```

From the `taskers.tcl` file, you can use the option `-health STRING` in a similar way:

```
## Fragment of taskers.tcl
vtk_tasker_define myHost -name my_test_tasker -health DWPU
```

User Created Script

When enabled, the vovtasker will run an adminstrator-created script placed in a specific location in the installation. There are security considerations in using this check, as the script will be run by any project that enables the UserScript checks.:

1. The script must be placed at `$VOVDIR/local/`tasker`/health_user_script.csh`
2. The script should be secured appropriately to prevent unauthorized modification.

3. The script will run as the user under which the project was started.

Configuration File for vovagent

For cases where the vovtaskers are started by submitting the binary to a separate batch queue system such as LSF or SGE, the system administrator may decide to use `vovagent`, which is a special copy of the `vovtasker` binary.

When `vovagent` is invoked, the binary limits the values of attributes that are based on the information stored in the configuration file `$VOVDIR/local/vovagent.cfg` to ensure compliance with the underlying batch queue system's FairShare policies.

The contents of the configuration file and relation to the command line and `taskers.tcl` file parameters are as follows:

Attribute Name	vovtasker options	vovagent.cfg option	Description
`maxidle`	-z	*`maxIdleTime`*	After being idle for the given time, tasker does not accept new jobs and exits after completing active jobs. Value is a VOV timespec, for example, 2m. The minimum value is 10s, and the maximum is the value of the *`maxlife`* parameter or 1hour, whichever is less. If the configuration file is absent or does not specify a value, the default is 1m.
`maxlife`	-Z	maxLifeTime	After the specified lifetime, tasker does not accept new jobs and exits after completing active jobs. The value is a VOV timespec, for example, 2H. The minimum value is 5m, and the maximum of unlimited is specified by giving 0 (zero) or a negative value. If the configuration file is absent or does not specify a value, the default is 2H.
`update`	-U	*`update`*	Specifies the update cycle time for the vovtasker agent, which is the interval at which tasker resource procedures are recalculated. The minimum value is 5s, and the maximum is half the value of the

Attribute Name	vovtasker options	vovagent.cfg option	Description
			`maxidle` parameter or 5 minutes, whichever is less. If the configuration file is absent or does not specify a value, the default is 1m.

An example of the `vovagent` configuration file is shown below.

Note: Set access permissions to ensure that only the VOV system manager account can modify this file.

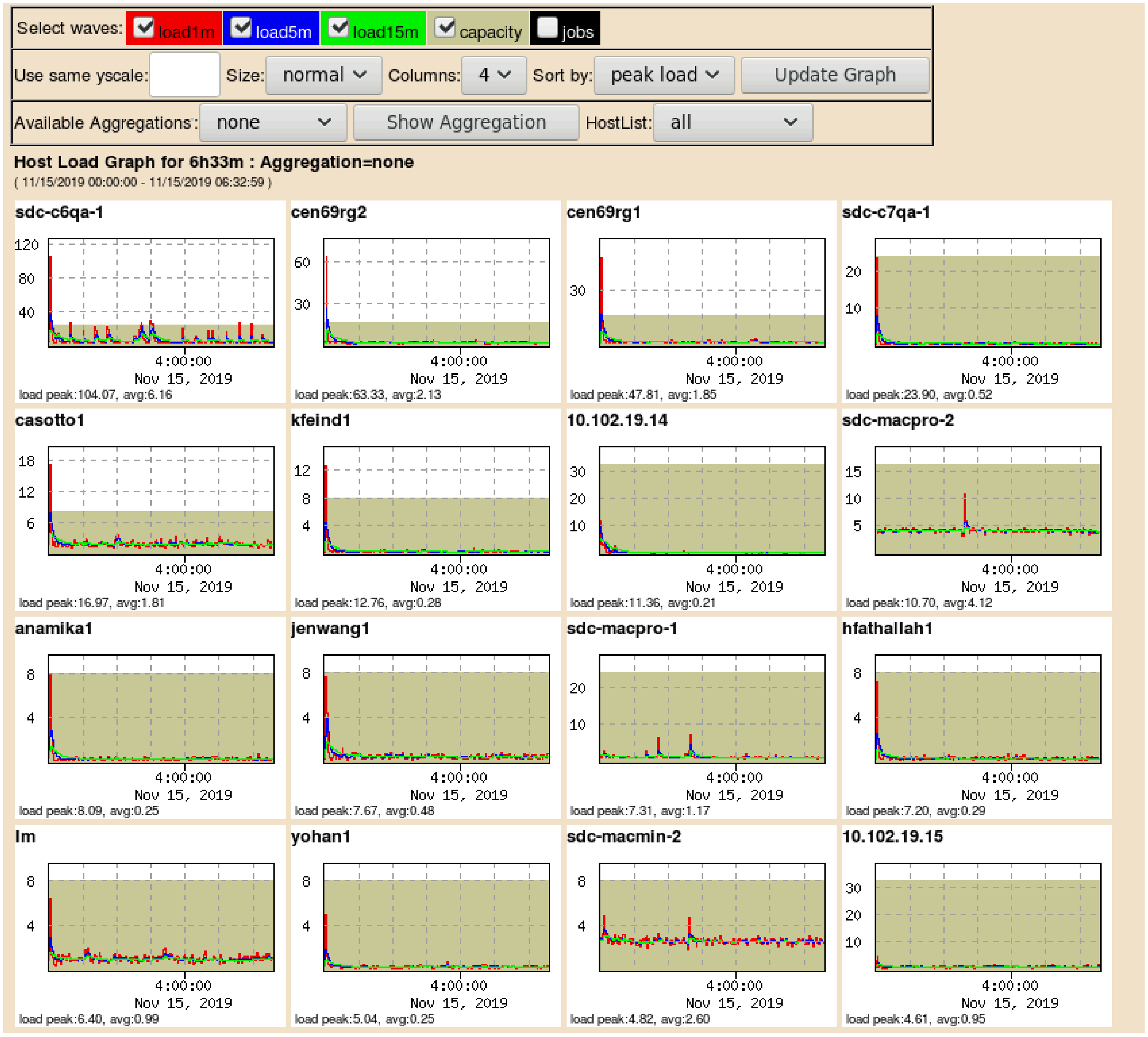
```
maxIdleTime  = 1m
maxLifeTime  = 1h
updateInterval =  15s
```

Tasker Load Reports

vovtasker continuously logs 1 minute, 5 minute and 10 minute load averages of the machine where the tasker is running. Tasker load reports are available on the Tasker Load page.

Set Up Aggregation

By default, the Tasker Load page shows the reports for all live taskers, one plot for each tasker, as shown below.



Select waves: ✓ load1m ✓ load5m ✓ load15m ✓ capacity ☐ jobs

Use same yscale: Size: normal Columns: 4 Sort by: peak load

Available Aggregations: none HostList: all

Host Load Graph for 6h33m : Aggregation=none
(11/15/2019 00:00:00 - 11/15/2019 06:32:59)

```
sdc-c6qa-1      load peak:104.07, avg:6.16
cen69rg2        load peak:63.33, avg:2.13
cen69rg1        load peak:47.81, avg:1.85
sdc-c7qa-1      load peak:23.90, avg:0.52
casotto1        load peak:16.97, avg:1.81
kfeind1         load peak:12.76, avg:0.28
10.102.19.14    load peak:11.36, avg:0.21
sdc-macpro-2    load peak:10.70, avg:4.12
anamika1        load peak:8.09, avg:0.25
jenwang1        load peak:7.67, avg:0.48
sdc-macpro-1    load peak:7.31, avg:1.17
hfathallah1     load peak:7.20, avg:0.29
lm              load peak:6.40, avg:0.99
yohan1          load peak:5.04, avg:0.25
sdc-macmin-2    load peak:4.82, avg:2.60
10.102.19.15    load peak:4.61, avg:0.95
```

Figure 1:

In addition to plots of individual taskers, plots can be aggregated to show the sum of the data of a selected group of machines (taskers). For example, the loads of the Linux machines could be aggregated and then compared to the summation of the loads of the HPUX machines. This optional feature can be set up in the taskerload configuration file.

For the Tasker Load page, the taskerload configuration file is located at: `projName.swd/taskerload_config.tcl`

For Accelerator with the queue name `vnc`, the taskerload configuration file is located at: `$VOVDIR/../../vnc/vnc.swd/taskerload_config.tcl`

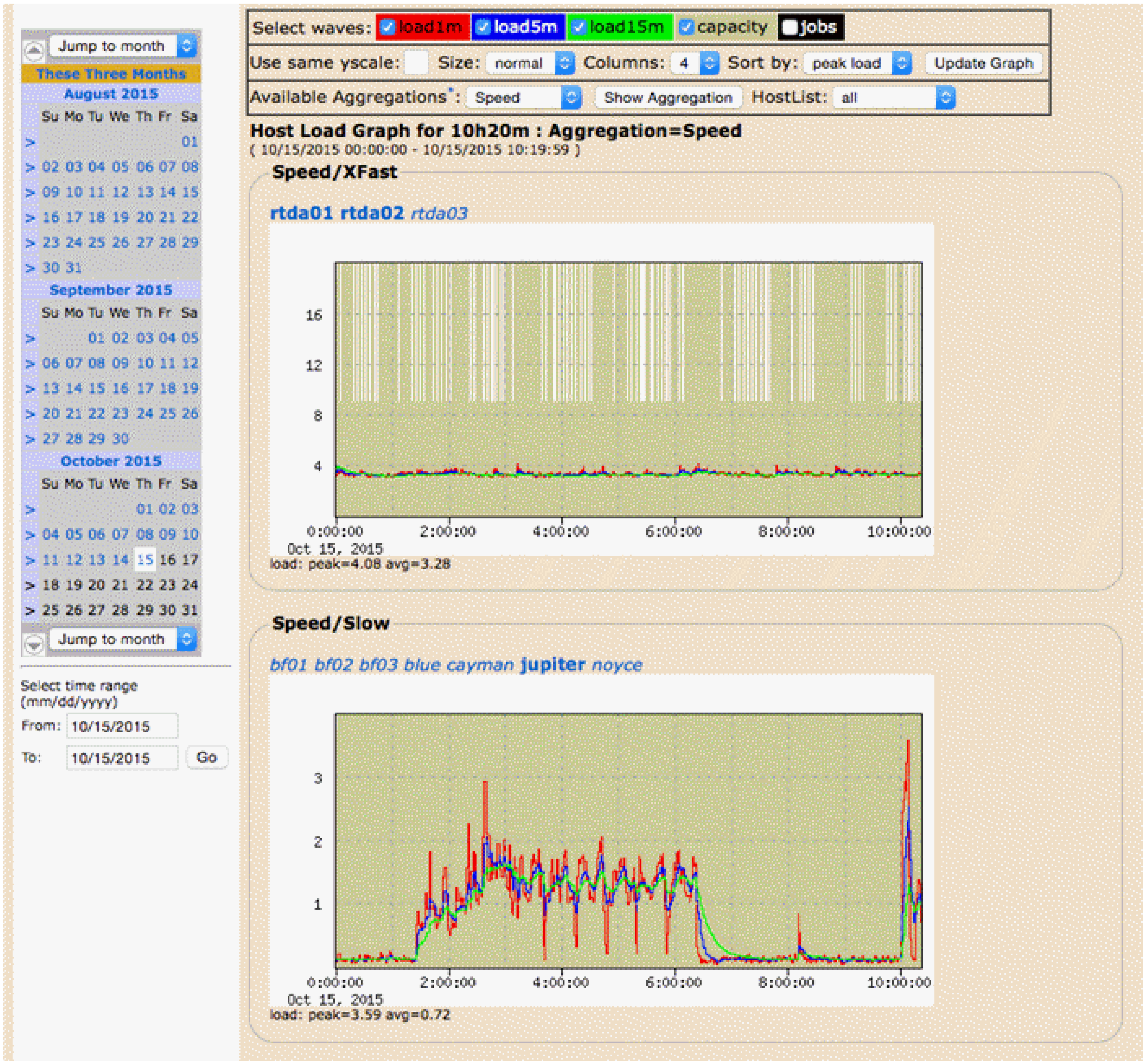
In the following example, three aggregation methods are set up: `Ownership`, `Type` and `Speed`.

```
# Sample of taskerload_config.tcl
set TASKERLOAD(aggregateby,Ownership)    "Development Regression Marketing"
set TASKERLOAD(aggregateby,Type)         "Linux"
set TASKERLOAD(aggregateby,Speed)       "Fast Slow"

set TASKERLOAD(taskers,Linux)   "farm01 farm02 farm03 farm04 farm05"

set TASKERLOAD(taskers,Fast)  "farm01 farm02 farm04 farm07 farm08"
set TASKERLOAD(taskers,Slow)  "farm12 farm13 farm14"

set TASKERLOAD(taskers,Development) "farm01 farm02 farm07 farm08 farm09"
set TASKERLOAD(taskers,Regression)  "farm03 farm04 farm05 farm 06 farm10 farm11
 farm13"
set TASKERLOAD(taskers,Marketing)   "farm12 farm14"
```

Setting up Aggregation by Speed

Two categories of `Speed` are available: `Fast` and `Slow`. The `Fast` category includes machine farm01, farm02 farm04 farm07 and farm08. The `Slow` category includes machine farm12, farm13 and farm14. In this case, choosing to have the report aggregated by `Speed` will result in two plots: one plot with the sum of the load of the machines defined in `Fast` category; one plot with the sum of the loads of the `Slow` machines.

The following is an example of reports that were aggregated with `Speed`:



Host Load Graph for 10h20m : Aggregation=Speed
(10/15/2015 00:00:00 - 10/15/2015 10:19:59)

Speed/XFast

rtda01 rtda02 rtda03

load: peak=4.08 avg=3.28

Speed/Slow

bf01 bf02 bf03 blue cayman jupiter noyce

load: peak=3.59 avg=0.72

Figure 2:

RAM Sentry

The RAM Sentry mechanism monitors the RAM utilization of the jobs and performs safety measures to prevent the tasker's memory resources from becoming saturated.

The RAM Sentry currently has one level of protection:

- **Suspend at swap**: For taskers that have multiple jobs running, if the tasker enters swap, the RAM Sentry suspends all jobs other than the job with the largest RAM footprint. When the large job completes, the smaller jobs will resume. While the RAM Sentry is active, the tasker itself is also suspended, preventing any new jobs from being accepted.

Enable RAM Sentry

To enable the RAM Sentry, set the variable VOV_RAM_SENTRY to 1 before starting a tasker. A simple choice is to add the following line to the `$VOVDIR/../../vnc/vnc.swd/setup.tcl` file, so that all taskers have this functionality enabled:

```
# Add this to the vnc.swd/setup.tcl file.
setenv VOV_RAM_SENTRY 1
```

After the tasker has started, you can control the mechanism for each individual tasker with `vovtaskermgr configure -ramsentry <boolean> ...,` as in the following examples:

```
% vovtaskermgr configure -ramsentry 1 linux010
% vovtaskermgr configure -ramsentry 0 linux010
```

Define Policies for Taskers

Host Availability Policy

In a typical network, compute servers are available around the clock while workstations are available only during the off-hours (for example, from 8pm to 8am and on weekends). This can be easily defined in the taskers file `vnc.swd/taskers.tcl` by adding a line like the following:

```
vtk_tasker_define workstation1 -resources  "VovResources::Offhours res1 res2"
```

In the above example, "workstation1" offers the resource list "res1 res2" during the off-hours. During the work-hours of the week, such a tasker will be suspended in the sense that it will not accept any new jobs. All jobs running at the time the suspension begins are carried out to completion.

In another scenario, a workstation is available whenever the owner is not actively using it. You can specify the following in the `vnc.swd/taskers.tcl` file:

```
vtk_tasker_define workstation1 -resources "VovResources::Workstation 5m 30m"
```

Here, "5m" indicates the minimum idle time required before any job is to be dispatched to "workstation1" and "30m" indicates that the tasker will not accept jobs with duration longer than 30 minutes.

Define a Custom Tasker Policy

You can define a custom policy for a tasker by adding a new procedure to the file `$VOVDIR/local/taskerRes.tcl`.

The following is an example of a simple procedure called `MyNight` for a tasker that behaves differently at night than during the day. The procedure is defined in the namespace `VovResources` and can be used by specifying the argument `VovResources::MyNight` for the option -resources of `vtk_tasker_define`.

```
# Fragment from the file $VOVDIR/local/taskerRes.tcl
namespace eval VovResources {
    namespace export MyNight
    proc MyNight { args } {
        #
        # During the night, the tasker accepts jobs up to 1 hour.
        # During the day, the tasker accepts jobs up to 2 minutes.
        #
        set HH [clock format [clock seconds] -format "%H"]
        regsub {^0} $HH "" HH; # Strip leading 0.
        if { $HH >= 6 && $HH < $19 } {
            vtk_tasker_set_timeleft 120
        } else {
            vtk_tasker_set_timeleft 3600
        }
        return "@STD@"
    }
}
```

The following should be noted:

- The resource list for the tasker is the value returned by the procedure
- The procedure `vtk_tasker_set_timeleft` *`n`* controls the maximum expected duration of jobs dispatched to the tasker. The value for *n* must be non negative. If the value is zero, the tasker is in the "SUSPENDED" state, that is it temporarily refuses to accept new jobs.

Time-Variant Taskers

Time-variant taskers resources can be configured using Tcl. Any procedure can be defined in the namespace `VovResources` and then used to compute the resource list.

If the first characters in the argument for the option -r is the sequence `VovResources::`, then the taskers resources are computed by executing the argument as a Tcl command.

For a list of predefined procedures, refer to *Predefined VovResources:: Procedures*.

A special case of a resource is the local disk space on a host, which varies according to the files stored there. There may be jobs that require a minimum amount of space to run successfully. The vovtasker implements `vtk_fs_stat` (see `vtk_fs`) to handle such requirements. An example is provided further below.

An alternative is to use load sensor. However, in the case of free disk space, `vtk_fs_stat` is recommended because it is more efficient.

For example, taskers can be set up to offer one set of resources during the day and another set of resources during the night. The following example is a script that computes the resource "nothing" between 5am and 8pm, and the standard resources during the night. In addition, `vtk_tasker_set_timeleft` is used to control the maximum expected duration of the jobs sent to the tasker.

```
namespace eval VovResources {
    proc Night { args } {
        # Return the standard resources during the night and suspend the tasker
        # during the day.  During the night, the tasker progressively reduces the
        # maximum length of the jobs it accept.
        set NIGHT_RESOURCES  "@STD@ $args"
        set EVENING_START  19
        set MORNING_END     6
        set HH [clock format [clock seconds] -format "%H"]
        regsub {^0} $HH "" HH; # Strip leading 0.
        if { $HH >= $MORNING_END && $HH < $EVENING_START } {
            # -- During the day, suspend the tasker.
            vtk_tasker_set_timeleft  0
        } else {
            # -- During the night, compute the time left.
            set hoursLeft  [expr $MORNING_END - $HH]
            set hoursLeft  [expr $hoursLeft >= 0 ? $hoursLeft : $hoursLeft + 24]
            set timeleft   [expr $hoursLeft * 3600]
            vtk_tasker_set_timeleft $timeleft
        }
        return $NIGHT_RESOURCES
    }
}
```

Example:

```
% vovtasker -r "VovResources::Night hspice"
```

These procedures are evaluated:

- Every minute, or at the interval that was selected with the option -U
- After the completion of each job.

The standard procedures are defined in the script `$VOVDIR/etc/tasker_scripts/taskerRes.tcl`. It is recommended to review these procedures before implementing your own procedures in `$VOVDIR/local/taskerRes.tcl`.

Free Disk Space

The following script monitors free disk space using `vtk_fs_stat`. This example script adds the resources `WORK` and `SCRATCH`; the values of these resources will be the amount of disk space in MB on the corresponding filesystem.

```
namespace eval VovResources {
    proc FsCheck { args } {
        # Usage:
        #   VovResources::FsCheck -fs WORK /work -fs SCRATCH /scratch -r  @STD@ -r xx
        #
        set resources {}
        while { $args != {} } {
            set arg [shift args]
            switch -- $arg {
                "-fs" {
                    set name [shift args]
                    set dir  [shift args]

                    set space [vtk_fs_stat $dir]
                    lappend resources "$name#$space"
                }
                "-r" {
                    lappend resources [shift args]
                }
            }
        }
        return [join $resources]
    }
}
```

Example:

```
% vovtasker -r "VovResources::FsCheck WORK /work"
```

Allocate Jobs to Machines Based on Percentages

All taskers in Accelerator offer a consumable resource called PERCENT. Every job in Accelerator requests at least PERCENT/1. If you want a job to have exclusive access to a tasker, the job should request PERCENT/100.

By default, each tasker provides PERCENT/100, but the total percentage can be adjusted for cases when part of a tasker needs to be pre-allocated for something outside of the queue. This is done either in The taskers.tcl File or in the taskerClass.table file by adding PERCENT/N to the resource specification, where N is a positive integer.

SGE Style Load Sensors

vovtaskers also support SGE style load sensors. Use option -L in vovtasker/vovtaskerroot to define the command line for the load sensor.

For example:

```
% cp $VOVDIR/etc/tasker_scripts/load_sensor_example.sh /tmp/myloadsensor.sh
% vovtaskerroot -L /tmp/myloadsensor.sh
```

For security reasons, a load sensor is accepted only if it is owned either by the user or by root.

Load sensors summary:

vovtasker option	-L *full_path_to_load_sensor*
taskers.tcl file	-loadsensor *full_path_to_load_sensor*

Execution

The tasker executes the load sensor:

- Every minute
- After the termination of a job

Special Variables

The implementation recognizes the following special variable names:

Variable name	Description
power	Overrides the effective power of the tasker.
maxload	Overrides the predefined value of max load allowed by the tasker.

All other variable names are added to the resource list for the tasker that is executing the load sensor.

Example of a Load Sensor

The following example can be found in `$VOVDIR/etc/tasker_scripts/load_sensor_example.sh`.

```
#!/bin/sh
### This is the classical example of a load sensor for SGE.
### Usage:  % vovtasker -L /full/path/to/load/sensor

myhost=`uname -n | awk -F. '{print $1}'`
myid=`id -u`

while [ 1 ]; do
     # wait for input
     read input
     result=$?
     if [ $result != 0 ]; then
          exit 1
     fi
     if [ "$input" == quit ]; then
          exit 0
     fi

     echo "Computing load sensor: $input"

     #
     # Send:
     #  1. Number of users logged in
     #  2. Another number, just for fun.
     #  3. Change the power of the tasker to affect the preference
     #
     logins=`who | cut -f1 -d" " | sort | uniq | wc -l`
     logins=`/bin/echo $logins` # Trim left white space.
     echo begin
     echo "$myhost:logins:$logins"
     echo "$myhost:id:$myid"
     echo "$myhost:power:42222"
     echo end
done

# We never get here.
exit 0
```

Start a Remote UNIX Tasker

On UNIX, the script `vovtaskerstartup` is used to start a tasker on a remote machine. This script ensures that the tasker runs in a valid environment by sourcing the following scripts.

- `$VOVDIR/local/scripts/vovtaskerstartup.aux`, if available, to perform site specific initialization:

```
# Example of $VOVDIR/local/scripts/vovtaskerstartup.aux
echo "This is vovtaskerstartup.aux on `uname -n`"
switch ( $VOVARCH )
 case "linux*":
   unlimit openfiles
   breaksw
 default:
endsw
```

- `$VOVDIR/etc/std.vov.aliases`, to define all standard aliases (for example, `ves`).
- `setup.tcl` in the server configuration directory to initialize the project environment.

If the -view option is used, the script also starts the taskers in the given ClearCase view.

The -descriptors option triggers a check to ensure the host has at least as many descriptors as vovserver, which ensures it can operate at full capacity in the event of failover. It also checks that the `server_election` directory is empty.

Note: For this function to work, the -failover option must be used with `vtk_tasker_define`. For more information, refer to vtk_tasker_define.

Taskers on Windows

Accelerator can dispatch jobs to taskers running on Windows hosts. Starting the taskers on Windows hosts, however, is less automated than on UNIX hosts.

To prepare the Windows host:

1. Mount the filesystem with the Accelerator installation on the Windows machine in the form of a drive letter. For example, assume that the installation is mounted as `f:\rtda\<version>`. Enter the location depending on your network setup.
2. Start a Windows `cmd` shell . Initialize it by executing the `vovinit.bat` script that is in the installation directory under `win64/bat`.

   For the example, use:

   ```
   % f:\rtda\<version>\win64\bat\vovinit.bat
   ```

   The `vovinit.bat` file adds the Altair Accelerator commands to the PATH and sets other environment variables used by the software.

   You may consider adding the following shortcut to your desktop to facilitate this operation in the future:

   ```
   cmd /k f:\rtda\<version>\win64\bat\vovinit.bat
   ```

   Refer to the Windows documentation to learn how you can add a shortcut to the desktop.
3. Start the Tasker Server Daemon with:

   ```
   % vovtsd -user vncadmin
   ```

   where `vncadmin` should be replaced by the login name of the Accelerator administrator who is running the Accelerator vovserver.
4. Mount all other drives required to perform the tasks.

   Basically, the Windows host must be capable of performing the tasks that are about to be dispatched by the Accelerator vovserver. This means that the filesystem with the run directory must be mounted at a drive letter, since you may not change directory to a UNC path.

Configure Taskers on Windows

1. Define the Windows host in `vnc.swd/taskers.tcl`.

2. Be sure to add the host to the list of Windows hosts, not the UNIX hosts. These hosts need to use special values for the option -serverdir and -vovdir of `vtk_tasker_define`. All of this is also explained by comments in the sample `vnc.swd/taskers.tcl` file.
3. Start the tasker using, for example, a soft reset:

```
% ncmgr reset -taskers
```

Windows User Impersonation

VOV supports Windows in addition to many UNIX variants. This section describes cross-platform jobs between UNIX/Linux and Windows.

User Accounts that Run Jobs on Windows

The persistent Windows vovtasker runs as the user that started it, but can switch to other accounts if it is supplied with the account credentials by the vovserver. These credentials are supplied with the job over the VOV-protocol connection between the vovserver and the vovtasker.

Every Altair Accelerator project, including Accelerator, is a collection of jobs and files managed by a vovserver. A project may be multi-user, and the vovserver stores information about the users who have jobs in the project. Each user has an account name that is related to a user ID, and possibly to a Windows account name. The vovserver uses the operating system's mechanism to authenticate users. A user must already have authenticated to the OS before being able to start a vovserver, and the vovserver runs in that user's account.

There is no superuser identity such as root that can switch to other accounts without providing any credentials. For Accelerator on Windows, a different method is used to run jobs as the submitter.

The vovtasker calls a Windows API to create a process with a username and password. The username is mapped from the UNIX/Linux username, and the password is stored in encrypted form in the vovserver and passed to the vovtasker. The password is destroyed immediately after use. It is possible for a UNIX/Linux user to run jobs as another Windows user, by logging onto the machine locally with the account name and password.

After the Windows process is running as the correct user, it may need to point to different drive letters as, as each user has their own set in recent Windows versions. For details, refer to *Vov Windows Impersonation*.

Enter the Windows Password - only needed if you want to switch the user running the job

In the `vovconsole` GUI, you may enter the Windows password using the **Tools** > **Windows Password** menu item. From the command line, you can use the `vovauxpasswd` command.

Batch Files

In the course of running the job, the Accelerator tasker on Windows creates a batch file, and may use an optional pre- and post-job batch file. Batch files cannot be edited by you, but the pre and post job batch files can be used for debugging purposes.

The following is an example of the pre-job batch file that if present, will be called by the vovtasker's job-bootstrap batch file. This example records the variables and drive letters in the file `pre-hook.txt`.

Example script that you must write and put in a directory that you have to create yourself: `vovbtpre.bat` that record variables and drive letters into a file for troubleshooting:

```
set & pre-hook.txt
net use && pre-hook.txt
```

Must be in C:/tmp directory, so please create this directory

If defined, the batch file `vovbtpost.bat` is called after the job. Use the same technique as the pre- file to create and use this fiile

The following example uses the `choice` command to keep the window visible until a key is pressed.

Example script: `vovbtpost.bat`:

```
REM Example script: vovbtpost.bat
REM Use choice cmd to make window stay around
choice /M "Press Y to finish"

echo done
```

Manage Remote Taskers without SSH/RSH Capabilities

The program `vovtsd` is a daemon written as a Tcl script that runs using the VOV `vtclsh` binary. This daemon can be used also to start various types of agents on any type of Windows or UNIX machine.

In Windows, `vovtsd` is started from the command line and then runs in a Windows `cmd` shell. Each vovserver connects to `vovtsd` via TCP/IP to start the vovtasker process.

vovtsd

This utility listens for requests to launch taskers for various projects, but always for the same user. The requests typically come from `vovtaskermgr`.

Usage

```

vovtsd: Usage Message

  VOVTSD: Vov Tasker Service Daemon
      This utility listens for requests to launch taskers for
      various projects, but always for the same user.
      The requests typically come from vovtaskermgr.

  USAGE:
      % vovtsd [OPTIONS]

  OPTIONS:
      -v                      -- Increase verbosity.
      -h                      -- Print this help.
      -debug                  -- Generate verbose output.
      -help                   -- This message.
      -normal                 -- Start a normal daemon (for current user)
      -expire <TIMESPEC>      -- Exit from vovtsd after specified time.
```

```
    -user  <user>          -- Specify the user that should be impersonated.
                              vovtsd computes the port number by
                              hashing the user name.
    -port  <n>             -- Specify port to listen to.

EXAMPLES:
    % vovtsd -normal
    % vovtsd -port 16666
    % vovtsd -user john -port 16000
```

Start a Remote vovtasker with vovtsd

To use `vovtsd` on Windows, follow these steps:

1. Start a command shell on the Windows workstation as the user who is supposed to run the taskers. See below if this user needs to be different from the user logged in on the screen.
2. Set up the shell to use VOV with the `vovinit` command. This sets the needed environment variables, including PATH.

   ```
   c:\temp> \<install_path>\win64\bat\vovinit
   ```
3. Mount all filesystems using the appropriate drive letter; these need to agree with the values of serverdir and vovdir in the `taskerRes.tcl` file for the VOV project.
4. Start `vovtsd`, possibly using a new window. The -normal option says to use a TCP/IP port calculated from the username. You may specify the port explicitly by using the -port option.

   ```
   c:> start vovtsd -normal
   ```

Run vovtsd as a Different User

The vovtasker started by `vovtsd` will run as the user running `vovtsd`. If you need for this to be different from the user logged in at the keyboard and screen, you have several options.

On Windows 2000 and Windows XP, you can use the `runas.exe` command included with the operating system. For example, on Windows XP, logged in as 'user1', you could start a command shell using:

```
C:\temp> runas /user:domain-name\username cmd
```

Note: You may need to mount the filesystems for that user. On Windows NT, the drive letters are shared, but on later versions, each user can mount a different filesystem on a given drive letter.

Black Hole Detection

A *Black Hole* is a tasker that appears healthy but is unable to execute jobs. All jobs sent to that tasker quickly fail, and the tasker appears ready to execute the next job in queue, although all jobs submitted to that tasker fail.

On a given tasker, if a number (`blackholeFailedJobs`) of consecutive jobs fail within a relatively short time (`blackholeDiscardTime`), the tasker is potentially a black hole. The tasker is certainly a black hole only when we know that a large fraction (`blackholeFailRate`) of those jobs succeed on other taskers. When a tasker is a potential back hole, it is suspended for a short amount of time (`blackholeMaybeTime`), typically around 10 seconds. When a tasker is a black hole, it is suspended for a longer period (`blackholeSuspendTime`) typically around 10 minutes.

To activate the functionality, use:

```
% nc cmd vovsh -x 'vtk_server_config blackholedetection 1'
```

To disable the functionality, use:

```
% nc cmd vovsh -x 'vtk_server_config blackholedetection 0'
```

To check whether black hole detection is active, use:

```
% nc cmd vovsh -x 'vtk_generic_get policy a; parray a' | grep blackhole
```

Troubleshooting a SICK Tasker

The vovserver marks a vovtasker SICK when it has not received the vovtasker's heartbeat message for three consecutive update cycles.

Possible causes include:

- The machine has crashed
- The machine got disconnected
- The top-level `vovtaskerroot` process has crashed or was killed

Since there is no single solution to this problem, here is a short debugging guide.

1. Is vovtasker SICK?

   If vovtasker is SICK, use:

   ```
   % nc cmd vovtaskermgr stop name-of-SICK-vovtasker
   % nc cmd vovtaskermgr start name-of-SICK-vovtasker
   ```

   Otherwise, vovtasker will not start.
2. Is the machine running?
   - No: you have a network problem: call IT
   - Yes: continue
3. Is vovtasker/vovtaskerroot stuck?

   a) On Linux, check the process status with:

   ```
   # root privilege is needed
   % strace -p PID% pstack PID
   ```

   where PID is the PID of the vovtasker/vovtaskerroot process.
   - No: continue

- Yes: often, the output of `strace` and `pstack` help diagnose the problem (e.g. a bad NFS mount, an unresponsive LDAP, ...).

Sometimes you may not be able to figure out what is holding up the vovtasker. Submit a support request at Altair Community for assistance.

Tasker Reservations

A vovtasker may be reserved for one or more:

- Users
- FairShare groups
- OS groups
- Jobclasses
- Job projects
- Jobs
- Buckets

The reservation is always for a specific period of time, starting at any time and ending some time in the future. If the end time is in the past, the reservation is ignored and removed. The reservation can be very long, for example thousands of days.

A tasker can have multiple reservations at any time. Jobs that do not match any reservation are not sent to the tasker if the tasker is reserved. For example:

- If the tasker is reserved for a user, only jobs submitted by that user may be dispatched to that tasker.
- If the tasker is reserved for a group, only jobs from that group may be dispatched to that tasker.
- If the tasker is reserved for a user and a group, only jobs submitted by that user who belongs to that group may be dispatched to that tasker.

Negated Reservations

Taskers may also be reserved for a negated set, in order to prevent access to the tasker by jobs that match members of the set. To negate a tasker reservation, simply place a not symbol "!" at the beginning of the reservation expression. For example, the command `vovtaskermgr reserve -user '!john,mary' -duration 1h tasker1` will allow the tasker named `tasker1` to run jobs from any user except `john` and `mary` for 1 hour. Any valid reservation expression can be negated.

Note: If you are negating a list, use the not symbol ONLY at the beginning of the list; you should NOT place a "!" before every item.

Persistent Reservations

Reservations are persistent. If you stop a tasker and restart another one with the same name, the new tasker will inherit the persistent reservation of the old tasker. Expired reservations are removed frequently.

If a tasker is renamed, the corresponding reservations get updated too. So, the reservations do not get lost.

Reservation rules
Only reservations with end time later than the current time are valid.

There can be multiple reservations on a vovtasker. But there is only one active reservation. If there is already an active reservation, a new reservation overlapping time with the active reservation can be created but ignored. When the active reservation expires (end time passes current time), a new active reservation with earliest start time is picked.
Reservation only applies to normal (direct) taskers or BPS agents. Making reservations on indirect taskers is not allowed.

Duplicate reservations

A new reservation can be created if the reservation is different from existing reservations. Two reservations are the same if the following attributes of reservations are the same. All of these attributes can be specified when creating a reservation using `vtk_reservation_create`. If a reservation is created through vovtaskermgr, vtk_tasker_reserve, and `taskers.tcl`, `start time` and `end time` are not used as identifiers. The existing reservation with the same other attributes are updated with the new start time and end time. If there is no existing reservation, a new reservation is created.

`type`	Always `tasker` if the reservation is for tasker.
`what`	List of tasker names that this reservation is reserving
`start time`	Reservation start time
`end time`	Reservation end time
`user`	Reservation is for these users
`group`	Reservation is for these groups
`osgroup`	Reservation is for these osgroups
`jobclass`	Reservation is for these jobclasses
`jobproj`	Reservation is for these jobprojects
`bucketid`	Reservation is for these bucket IDs
`id`	Reservation is for these job IDs

Reserve a vovtasker

There are several ways to create a new reservation.

vovtaskermgr, `taskers.tcl`, and vtk_tasker_reserve creates a new reservation, but if there is an overlapping reservation with the same parameters, the existing one is updated with new `start_time` and `end_time`. These are the same with running `vtk_reservation_create` with the -update option.

On the Command Line: vovtaskermgr reserve

The syntax is:

```
% vovtaskermgr reserve [options] <taskers_list>
```

where the options could be:

-user	Comma separated list of users
-group	Comma separated list of groups

-jobproj	Comma separated list of job projects
-jobclass	Comma separated list of job classes
-osgroup	Comma separated list of OS groups
bucketid	Comma separated list of bucket IDs
-id	Comma separated list of job IDs
-start	Start time
-end	End time
-cancel	
<tasker_list>	List of tasker names to reserve.

For example, the following command reserves taskers `jupiter` and `alpaca` for user john for 3 hours starting from now.

```
% vovtaskermgr reserve -user john -duration 3h jupiter alpaca
```

With `-cancel` option, all reservations on the specified tasker(s) will be removed.

```
% vovtaskermgr reserve -cancel jupiter alpaca
```

The following command shows all reservations for taskers.

```
% vovtaskermgr reserveshow
```

In the taskers.tcl File

This is useful to reserve a tasker from the instant it is created. Use option -reserve of vtk_tasker_define.

The reservation expression argument to the -reserve option takes space-separated list of key value pairs, where the key is one of `USER`, `GROUP`, `JOBCLASS`, `JOBPROJ`, `BUCKET`, `DUR`, `QUANTITY`. If the key is `DUR`, the value is a time spec. If not specified, the default duration is 1 year. For the other keys, the value is a comma-separated list.

Examples:

```
vtk_tasker_define jupitar -reserve "USER john,mary JOBCLASS spectre"
vtk_tasker_define alpaca -reserve "JOBPROJ chipa,chipb DUR 3w"
```

The old form `GROUP/USER/DURATION` is accepted. The `GROUP` and `USER` parts are optional, but the separators ('/', the forward-slash character) must be present. The duration is expressed as a VOV timespec, e.g. 2d for two days. If only digits are present, the value is interpreted as seconds.

-reserve is passed to vovtasker executable as -e option. If advanced users want to run a tasker with -e option for initial reservation, the syntax is the same as `vtk_tasker_define -reserve` option.

Reserving a Tasker via the Browser

Open the tasker page to reserve a particular tasker. It is at the URL `/tasker/taskerId`. Fill out a simple form, indicating which user, group, OS group, job class, and/or job project for this tasker is to be reserved, as well as start time and duration. After you select the duration, the form will be submitted.

Click **Forget** to cancel the tasker reservation, if you are ADMIN.

vtk_tasker_reserve Tcl Interface

With Tcl, you can use tasker reservations. The syntax is:

```
vtk_tasker_reserve taskerId [-user <user1,user2,...>]
                          [-group <group1,group2,...>]
                          [-osgroup <osgroup1,osgroup2,...>]
                          [-jobclass <jobclass1,jobclass2,...>]
                          [-jobproj <jobproj1,jobproj2,...>]
                          [-bucketid <bucketid1,bucketid2,...>]
                          [-id <jobid1,jobid2,...>]
                          [-start <starttime>]
                          [-end <endtime>]
                          [-duration <reserved_duration>]
```

For example, the following line will clear (cancel) all reservations on tasker 00001230, if there is one. Otherwise, this doesn't have any effect.

```
vtk_tasker_reserve 00001230
```

The following line reserves tasker 00001230 for user `john` for 3 hours starting from now.

```
vtk_tasker_reserve 00001230 -user john -duration 3h
```

This reserves tasker 00001230 for user `john` in group `alpha` for 2 weeks starting from one hour from now.

```
vtk_tasker_reserve 00001230 -user john  -group alpha  -start [clock scan "1 hour"]  -
duration 2w
```

vtk_reservation_create Tcl Interface

You can use `vtk_reservation_create` in a Tcl interface. This is a new interface introduced in 2017 version to support multiple reservations per tasker.

The syntax is:

```
vtk_reservation_create <type> <what> <quantity> <start_time> <end_time> [options]
```

where:

<what>	Comma separated list of tasker names
<quantity>	Not used for tasker reservations
<start_time>	Reservation start time
<end_time>	Reservation end time

And the options could be:

-user	Comma separated list of users
-group	Comma separated list of groups
-osgroup	Comma separated list of OS groups
-jobclass	Comma separated list of jobclasses
bucketid	Comma separated list of bucket IDs
-id	Comma separated list of job IDs
-update	

For example, the following Tcl script creates a reservation for host1 and host2. It reserves 4 slots of each host. It returns ID for the reservation. The ID can be used to update and delete the reservation.

```
set now [clock seconds]
vtk_reservation_create tasker host1,host2 1 $now [expr $now+3600] -user brian
```

If all attributes are the same as one of existing reservations, `vtk_reservation_create` will return `nochange`.

With -update option, it looks for a reservation which has same attributes except `start_time` and `end_time` but the reservation period is overlapping. If there is one, the existing reservation is updated with `start_time` and `end_time`.

```
set now [clock seconds]
set end [expr $now+3600]
set end2 [expr $now+7200]
vtk_reservation_create tasker localhost 1 $now $end -user john #creates a new
 reservation
vtk_reservation_create tasker localhost 1 $now $end -user john #returns "nochange"
vtk_reservation_create tasker localhost 1 $now $end -group g1  #creates a new
 reservation
vtk_reservation_create tasker localhost 1 $now $end2 -user john -update #updates the
 first
```

Manage Reservations

To show all existing reservations, use:

```
% vovshow -reservations
```

To forget all reservations, use:

```
% vovforget -allreservations
% vovforget <reservationId>
```

Reservations data is accessible with `vovselect` as well.

```
% vovselect id,reserveuser,reservestart,reserveend from reservations
```

vtk_reservation_get Tcl Interface

`vtk_reservation_get <reservationId> <variable>` provides the details about a reservation.

```
vtk_reservation_get 21673 info
parray info
```

Prints out the following information:

```
info(id)                = 21673
info(quantity)          = 1
info(reservebucketid) =
info(reservecreated)  = 1513026518
info(reservedby)      = jin
info(reserveend)      = 1513199318
info(reservegroup)    =
info(reserveid)       =
info(reservejobclass) =
info(reservejobproj)  =
info(reserveosgroup)  =
info(reservestart)    = 1513026518
info(reserveuser)     = jin
info(type)            = tasker
info(what)            = local2
```

`vtk_reservation_update <reservationId> <fieldname> <new_value>` updates a field of reservation with a new value. Available field names can be found by `vovselect fieldname from reservations` on the command line.

```
vtk_reservation_update 21673 RESERVEUSER robert
```

`vtk_reservation_delete <reservationId>` removes the reservation.

```
vtk_reservation_delete 21673
```

Number of Reservations and System Performance

Many reservations on each tasker may slow down the system. Upon choosing the right tasker to run a job, the algorithm considers all reservations. It is recommended to use the limited number of reservations per tasker. By default, the maximum number of reservations per tasker is set as 10 and this is configurable through a server parameter *`tasker.max.reserve`* in `policy.tcl`.

Clean up Processes Left Behind by Completed Jobs

Some third-party software has a tendency to spawn child processes but do not ensure that they are cleaned up once the main process ends. This behavior can lead to overloaded, and in extreme cases, unresponsive hosts. Accelerator can be configured to enable automatic cleanup of such processes. This functionality is supported on Linux only.

Note: While automated cleanup is an effective strategy for combating this problem, the behavior should not be considered as normal, and it is recommended to report it to the third-party software vendor when it is encountered.

If enabled, for each job that has ended, the vovtasker will parse the environment metadata for each process that exists on the host. If the VOV_JOBID environment variable exists in the environment for a process, and its value matches that of the job that has ended, the process will be marked as an orphan process that must be cleaned up. Since child processes inherit their parent's environment, the vovtasker will be able to identify related child processes hierarchically. Once all orphan processes have been identified, the vovtasker will send the KILL signal to each one and will print a corresponding message in the tasker log.

Note: Processes that create their own environment from scratch, as well as ones that explicitly remove the VOV_JOBID variable from the environment will not be cleaned up by this feature.

1. To enable this feature, add the following line to the `$SWD/policy.tcl` file:

```
set config(tasker.childProcessCleanup) 1
```

2. Once the file has been edited, reread the policy via:

```
% nc cmd vovproject reread
```

3. To confirm the feature is enabled:

```
% nc cmd vovselect param.tasker.childProcessCleanup from server
```

   A value of 0 indicates the feature is disabled (default), whereas a value of 1 indicates the feature is enabled.

Web Server Configuration

HTTP Access Models

There are 3 HTTP access models:

- Legacy
- Internal/External
- Nginx

Legacy Webserver

The Legacy webserver is the basic web server that is internal to vovserver and serves content directly to web browser clients.

All traffic is transmitted using HTTP protocol and is unsecured. This method is approriate for REST versions up to version 2.0.

This is the case when

- `webport=0`, or
- `webport != 0` and `webprovider=nginx`

Internal Webserver

The Internal webserver is an enhanced web server that is internal to vovserver, for secure pages and all REST versions.

The Internal webserver is established when

- `webport != 0` and `webprovider=internal`

To specify the web port at product start, refer to the product-specific documentation for startup. To change the port in an already-running product instance, see *Advanced Control of the Product Ports*. To enable SSL support (HTTPS), follow the steps in Configure the TLS/SSL Protocol.

You get REST v3 API support from this webserver, and we still transparently delegate some HTTP requests to the old web server on the VOV port.

The Internal server securely handles all incoming traffic, decrypting it before handing it off to the locally running vovserver. Likewise, any response that is sent back to the browser is routed through the Internal webserver, which encrypts the response and sends it to the browser. This implementation is known as an SSL termination proxy.

nginx Webserver

The vovserver serves content to a proxy webserver (nginx), which communicates to web browser clients. Under this model, SSL can be enabled, securing all traffic using the HTTPS protocol.

The nginx web server is enabled when the web port is configured with a non-zero value. To specify the web port at product start, refer to the product-specific documentation for startup. To change the port in an already-running product instance, see *Advanced Control of the Product Ports*. To enable SSL support (HTTPS), follow the steps in Configure the TLS/SSL Protocol.

For experts only, advanced customizations to the nginx configuration can be made by modifying its configuration template. Configuration templates are searched for in the following locations:

Order	Type	Path
1.	Instance-specific	`$SWD/vovnginxd/conf/nginx.conf.template`
2.	Site-wide	`$VOVDIR/local/config/vovnginxd/nginx.conf.template`
3.	Installation-specific(edits not recommended)	`$VOVDIR/etc/config/vovnginxd/nginx.conf.template`

If customizations are intended, it is recommended to start with a copy of the default configuration template shown at location 3 above and place into either location 1 or 2.

Note:

- The configuration template is copied into the nginx configuration directory located at `$SWD/vovnginxd/conf`, named as `nginx.conf`. The copy is made upon product start, as well as any time the web port or SSL configuration is changed.
- Changes to the actual configuration file can be read into nginx via the `vovdaemonmgr reread vovnginxd` command, but such changes will be overwritten the next time the configuration template is copied.
- The configuration template contains keywords surrounded by @ signs, such as @WEBPORT@, that are dynamically substituted with values during the copy process. Removal of these keywords is not recommended, as it may effect the ability for nginx to be reconfigured in the event of a vovserver failover.

Configure the TLS/SSL Protocol

The internal and nginx webservers support TLS/SSL Protocol communication via "https" - prefixed URLs when configured correctly.

The vovserver serves content to a proxy webserver (nginx), which communicates to web browser clients. Under this model, SSL can be enabled, securing all traffic using the HTTP protocol.

When SSL is enabled, nginx will look for an SSL certificate/key pair in the following locations:

Order	Type	Path	Files
1.	Site-wide wildcard	`$VOVDIR/local/ssl`	`wildcard-crt.pem` `wildcard-key.pem`
2.	Host-specific	`$SWD/config/ssl`	`hostname-crt.pem` `hostname-key.pem`
3.	Host-specific (auto-generated and self-signed)	`$SWD/config/ssl`	`hostname-self-crt.pem` `hostname-self-key.pem`

Note:

- For hostname, use the actual host name that will be used to access the web UI. This will be the value of VOV_HOST_HTTP_NAME that was set in the configuration. If not defined, the value of VOV_HOST_NAME is used instead.

  To use the fully qualified domain name, the value of VOV_HOST_HTTP_NAME must be set.
- Self-signed certificates will present security warnings in most browsers.

Updating the TLS/SSL cert requires restarting the webserver so that the cert files can be re-read. For the internal webserver, see, "Restarting the Webserver" below.

Guest Access Port

The vovserver can be configured to enable a guest-access port, also called the read-only port due to the limited privileges allowed by the port. This port bypasses the login prompt and provides the user with a READONLY security principle, which disallows access to writable actions as well as certain pages in the UI.

To specify the guest access port at product start, refer to the product-specific documentation for startup. To change the port in an already-running product instance, follow the steps in *Advanced Control of the Product Ports*.

Transition from nginx Webserver to Internal

To transition from external (nginx) to the internal web server, follow these steps:

1. Shut down nginx with the command `vovdaemonmgr stop vovnginxd`.
2. Delay for 5 seconds with the command `sleep 5`.
3. Start the internal web server with `vovservermgr config webprovider internal`.

Restarting the Webserver

Complete the following steps to restart the webserver without bringing down vovserver.

1. Enter the following:

   ```
   vovservermgr config webport 0
   ```
2. Wait five seconds, then enter:

   ```
   vovservermgr config webport $VOV_WEB_PORT_NUMBER
   ```

Altair Accelerator Configuration

Many aspects of Accelerator behavior can be customized.

- Caching of `nc list` results may be configured in the file `$VOVDIR/local/vnclist.config.tcl`

```
# Example content of vnclist.config.tcl
# enable 'nc list' local caching
set NCLIST(cache,enable) 1

# Other variables used in global array NCLIST (not settable)
#set NCLIST(cache,timeout,default)
#set NCLIST(cache,cacheFile)
#set NCLIST(cache,cacheFp)

# environment variables that influence 'nc list' caching
# NC_LIST_FORMAT
# NC_LIST_CACHE_DIR  ; # default ~/.vov/vnclist.caches/<project>...
# NC_LIST_CACHE_TIMEOUT
```

Job Submission Policy

The job submission behavior of Accelerator or Accelerator Plus can be controlled by the file `vnc_policy.tcl`, which resides in the vovserver configuration directory.

This file is used to define the procedures that are listed below.

Note: `vnc_policy.tcl` can now reside in `vnc.swd/vnc_policy.tcl` as well as `$VOVDIR/local/vnc_policy.tcl`.

When placed in the configuration directory, it only affects that Accelerator instance. When placed in the 'local' directory, it affects all Accelerator instances.

Procedures for Customizing Job Submission

Procedure	Args	Description
VncPolicyDefaultPriority	`{ user }`	Assign the default priority to a job based on the user.
VncPolicyDefaultResources	`{}`	Compute the default resources required by a job.
VncPolicyGetJobInfo	`{ key }`	Retrieve job information. Following are the available key values: **tool** Tool or command name, such as hsim **command** Complete command line (without wrapper) **user** Login name submitting the job **setName** Name of the set in which the job is to be placed **group** The group the submitter requested

Procedure	Args	Description
		inputs Inputs to the job **outputs** Output files of the job **mailuser** Email address, if notification was requested **wrapper** Name of the FT wrapper program, such as `vw` **priority,default** Default submission priority **priority** Requested submission priority **resources** Requested submission resources **env** Name of requested job run environment **xdur** Expected duration of job
VncPolicyUserPriority	`{ user schedPriority }`	Limit the scheduling priority based on the maximum allowed to the user.
VncPolicyUserPriorityExec	`{ user execPriority }`	Limit the execution priority for a job. By default, this returns the priority that has been passed in.
VncPolicyValidateCommand	`{ commandLine }`	Make sure that the command line for a job obeys site-specific rules.
VncPolicyValidateEnvironment	`{ envName }`	Make sure that the environment name for a job obeys site-specific rules.
VncPolicyValidateOptions	`{ subCommand argv }`	Ensures the arguments obey site-specific rules. Returns a modified list of options. See example below.
VncPolicyValidateResources	`{ reslist }`	Ensure that the resource list for a job obeys rules defined by the Accelerator administrator.

Example for `VncPolicyValidateOptions`:

```
proc VncPolicyValidateOptions { subCommand argv } {
    set nargv []
    set maxAutoKill [VovParseTimeSpec 30d]
    while { $argv ne {} } {
        set arg [shift argv]
        switch -glob -- $arg {
            "-autokill" {
                lappend nargv $arg
                set value [VovParseTimeSpec [shift argv]]
                if { $value > $maxAutoKill } {
                    lappend nargv "30d"
                } else {
                    lappend nargv $value
                }
            }
            default {
                lappend nargv $arg
            }
        }
    }
    return $nargv
}
```

The `VncPolicy*` procedures are called at job submission time, and may cause the job entered into the server to have modified resources or priority compared to what the submission requested.

The following is an example for `vnc_policy.tcl`:

```
# This is an example of vnc_policy.tcl
proc VncPolicyDefaultResources {} {
    global env
    return "$env(VOVARCH) RAM/50"
}

proc VncPolicyValidateResources { resList } {
    #
    # This policy adds a minimum RAM requirement
    # for all submitted jobs.
    #    global VOV_JOB_DESC
    if { $VOV_JOB_DESC(tool) == "vovresgrab" } {
# Do not touch this type of jobs (see vovresreq).
       return $resList
    }

    if [regexp "RAM/" $resList] {
        # Job already has a RAM constraint.
    } else {
        # Add a RAM constraint.
        lappend resList "RAM/100"
    }
    return $resList
}
```

The following is an example using the tool name. This can be used to send jobs of a certain tool to specific hosts. A Tcl `catch{ }` is used in case someone uses this file with an older version by mistake.

Fragment of `$VOVDIR/local/vnc_policy.tcl`:

```
# This is a second example of vnc_policy.tcl
proc VncPolicyDefaultResources {} {
    global env
    return "$env(VOVARCH)"
}

proc VncPolicyValidateResources { resList } {
    #
    # This policy sends tharas jobs to vovtasker hosts offering 'tharas_host'
    # and keeps other kinds of jobs off those hosts
    #
    catch {
        set jtool [VncPolicyGetJobInfo tool]
        if { "$jtool" == "tharas" } {
            lappend_no_dup resList tharas_host
        } else {
            lappend_no_dup resList "!tharas_host"
        }
    }
    return $resList
}
```

Throttling Job Submission Rate

There is also a way to throttle users who have submitted a number of jobs over a configurable threshold. This was implemented so that users trying to submit too many job in a small time frame can not overload vovserver. The process adds a delay to job submission for users that have gone over that threshold.

Note: Although mentioned in this section because they affect job submission, these values are set in the vovserver configuration file `policy.tcl`.

Procedure	**Args**	**Description**
`hog.protection.enable`	`( )`	The default is 0, which means it is disabled. Add a 1 to enable it.
`hog.protection.jobcountth`	`( N )`	N represents the number of jobs which will trigger this threshold. Default is 100000. Min value is 1000. Max value is 999999.
`hog.protection.clientdela`	`( S )`	S is the number of seconds to delay the submit of a user how has triggered this rule. The default is 1 second. Min is 1 second. Max is 600.

nc run Command

The `nc run` command has built-in default features that include checking the validity of the run directory, enabling job profiling, etc. This section describes how the Accelerator administrator can use the file `$VOVDIR/local/vncrun.config.tcl` to modify some defaults.

This file does not exist by default; it must be created when needed.

The defaults for job characteristics are controlled by entries in the VOV_JOB_DESC array variable. The `vncrun.config.tcl` file is loaded after the defaults are set; these defaults can be overridden.

For additional information, refer to Define Jobclasses for details about VOV_JOB_DESC.

Examples

When submitting a job, the default is to check for a logical name (equivalence) for the filesystem where the run directory is located. This is controlled by the `check,directory` slot.

To change the default to not check the directory, add the following to the `vncrun.config.tcl` file:

```
set VOV_JOB_DESC(check,directory) 0
```

When submitting a job, the default is not collecting profile information, because the data can be large, unless the -profile option is used. To make profile collection the default setting, add following to the config file.

```
set VOV_JOB_DESC(profile) 1
```

An example file is included below that shows some other commonly-used settings:

```
# Example content of vncrun.config.tcl
set VOV_JOB_DESC(check,directory) 0

# Other settings that may be useful.
# set VOV_JOB_DESC(priority,default) [VncPolicyUserPriority $username]
# set VOV_JOB_DESC(priority,sched)   $VOV_JOB_DESC(priority,default)
# set VOV_JOB_DESC(priority,exec)    $VOV_JOB_DESC(priority,default)

# set VOV_JOB_DESC(autokill)      0
# set VOV_JOB_DESC(autoforget)    1
# set VOV_JOB_DESC(legalExit)     "0"
# set VOV_JOB_DESC(mailuser)      ""
# set VOV_JOB_DESC(wrapper)       "vw"
# set VOV_JOB_DESC(preemptable)   1
# set VOV_JOB_DESC(profile)       0
# set VOV_JOB_DESC(schedule,date) 0
# set VOV_JOB_DESC(xdur)          -1
```

Configure Callbacks with vnccallbackaction

In some jobclasses, it may make sense to call custom procedures:

- Right after a job has been created, i.e. as soon as we know its VovId
- Right after a job or set has been scheduled

This behavior is controlled by `VncCallbackAction`. The behavior of this procedure is documented by the following example:

```
# Example of a jobclass with custom callback to be invoked after the
# job has been created.

set VOV_JOB_DESC(resources) "unix RAM/100"
# ... other jobclass stuff


### Callback section.
proc MySpecialProcedure { jobId } {
   vtk_transition_get $jobId jobInfo
   if { $jobInfo(env) eq "SNAPSHOT" } {
      set jobInfo(env) "MYENV"
      vtk_transition_set $jobId jobInfo
   }
}

proc MySpecialCleanup { args } {
  VncCallbackAction del run post_create MySpecialProcedure
  VncCallbackAction del run finish      MySpecialCleanup
}


# VncCallbackAction verbose
# VncCallbackAction quiet
  VncCallbackAction add run post_create   MySpecialProcedure
# VncCallbackAction add run post_schedule MySomeOtherProc
```

```
   VncCallbackAction add run finish        MySpecialCleanup
```

Speed up nc run

The overhead in the `nc run` commands consists mostly in the on-demand compilation of Tcl code and secondarily on the number of round-trips to vovserver.

(Experimental feature) To amortize the Tcl compilation across multiple job submission, you can use the nc - option and then pipe a large number of `run ...` commands, as in the following example:

```
% mkfifo /tmp/vovfifo$USER.$$
% nc - < /tmp/vovfifo$USER.$$
% echo run hostname >> /tmp/vovfifo$USER.$$
```

To limit the number of round trips, you can define a number of environment variables that define semi-constant values, namely:

- NC_URL: This is the main URL for the vovserver and is used to compute the full URL for each submitted job. If not defined, the code in `nc run` needs to query the server about the HTTP server name and the WEB port.
- NC_DEFAULT_JOBCLASS: This is the name of the jobclass to be invoked by default. It can be set to the empty value. This is supposed to have the same value as

  ```
  `vovprop GET 1   NC_DEFAULT_JOBCLASS`
  ```
- NC_VALID_DIRECTORIES: The value is a list of directories from which it is acceptable to submit jobs, all in addition to directories that contain the `.vnc` file. It can be set to the empty value. This is supposed to have the same value as

  ```
  `vovprop GET 1   NC_VALID_DIRECTORIES`
  ```

These variable can be set, for example, in the `setup.tcl` file:

```
# Fragment of the setup.tcl file
# Used to speedup nc run
setenv NC_URL              https://nchost:6271
setenv NC_DEFAULT_JOBCLASS normal
setenv NC_VALID_DIRECTORIES ""
```

nc list Command

This section describes how the Accelerator administrator can use the file `$VOVDIR/local/vnclist.config.tcl` to modify some defaults for the `nc list` command. This file does not exist by default; it must be created when needed.

Enable List Cache

By default, list results are obtained from the server in real-time. In large-scale workload environments, repeated queries can impact server performance. To reduce this impact, a list result cache can be enabled:

```
# Example content of vnclist.config.tcl
# enable 'nc list' local caching
set NCLIST(cache,enable) 1
```

Configure List Cache Expiration

If the list cache is enabled, list results will be written to a client-side file, and subsequent list requests will be obtained from this file, up to the cache expiration. The default expiration is 30s from creation. After this time, the cache file will be regenerated upon the next list request. To set the cache expiration to a different value:

```
set NCLIST(cache,timeout,default) 60
```

```
# Other variables used in global array NCLIST (not settable)
#set NCLIST(cache,timeout,default)
#set NCLIST(cache,cacheFile)
#set NCLIST(cache,cacheFp)
```

Disable Listing by Job Name

Another list operation that can affect server performance in a large-scale workload environment is listing by job name. This is due to the need to compare string values across many jobs. Listing by job name can be disabled entirely by setting:

```
set NCLIST(listbyjobname,enable) 0
```

Influential Environment Variables

Environment variables that influence `nc list` caching:

```
# NC_LIST_FORMAT
# NC_LIST_CACHE_DIR  ; # default ~/.vov/vnclist.caches/<project>...
# NC_LIST_CACHE_TIMEOUT
```

nc wait Command

This section describes how the Accelerator administrator can use the file `$VOVDIR/local/vncwait.config.tcl` to modify some defaults for the `nc wait` command. This file does not exist by default; it must be created when needed.

Examples

Add Polling for Wait Calls

```
# Use 2s polling for all wait calls instead of listening to the event stream (-
p)
set polling 2000
```

Show Job Info

```
# Always show job info (-jobinfo)
set VNCWAIT(showJobInfo) 1
```

Register a Call-Back Command

```
# Register a call-back command (-callback)
set VNCWAIT(callbackCmd) {$VOVDIR/local/myWaitCallback.sh}
```

Accelerator GUI

The Accelerator GUI is implemented in Tcl/Tk. By editing the file, the appearance can be customized and extra elements can be added. The customization file is `gui.tcl`, which located in the server configuration directory (normally `$VOVDIR/../../vnc/vnc.swd/gui.tcl`).

In the following example, a button is added at the bottom of the GUI, which is only available when the GUI is started with environment variable SIMPLEDEMO set.

```
#
# This is a fragment of vnc.swd/gui.tcl
# to demonstrate the customizability of the NC GUI.
#
if [info exists env(SIMPLEDEMO)] {
    button .demo -text "Simple Customization Demo" -command {
        puts "You can run any command you want."
    }
    pack .demo -side bottom -fill x -expand 0
}
```

To test and verify the customization, use the following command:

```
% env SIMPLEDEMO=1 nc gui &
```

Web Interface

Some features of the Accelerator web interface can be configured by the administrator. Configuration for these items is performed in the `vnc.swd/config/web.cfg` file. The complete list of customizable items is shown below.

Example Configuration: `$VOVDIR/etc/config/nc/web.cfg`

```
### NODE VIEWER SETTINGS
#### Use a select (drop-down) menu for priority-based retrace controls
# Set to 0 to disable, set to 1 or comment-out to enable (default)
# set nodeviewer(retraceSelect) 1

# Use a select (drop-down) menu for preemption controls
# Set to 0 to disable, set to 1 or comment-out to enable (default)
# set nodeviewer(preemptSelect) 1
```

Configuration Management (CM) Guide

It is a recommended best practice to employ change control on your project configuration files.

There are 2 primary reasons for using configuration management.

- To impose strict change control on your project configuration files, and enable the ability to revert to a previous known good configuration.
- To quickly and efficiently model your production configuration in a test setup. Refer to Use CM in a Test Instance below.

An open source utility called "Fossil" has been integrated with the `ncmgr` command as a set of subcommands under the "cm" option. Keep in mind that as you are using the `ncmgr` utility for any capacity, you will need to first set the default queue name if it is not "vnc". This can be accomplished by one of 2 ways:

1. Set the NC_QUEUE variable:

   ```
   setenv NC_QUEUE queue_name
   ncmgr cm <cm_command>
   ```

2. Use either one of the -q or -queue options to `ncmgr`.

   ```
   ncmgr cm -q queue_name <cm_command>
   ```

There is a minimum set of files in the Server Working Directory (SWD) of your project that are best under CM control. These include, but are not limited to the following files:

`taskers.tcl`
`policy.tcl`
`equiv.tcl`
`exclude.tcl`
`resources.tcl`
`security.tcl`
`setup.tcl`

Following this initial setup, additional files (or entire directories within the SWD, such as the config directory) can be added using the command:

```
ncmgr cm add<filename>
```

Functions Available

`add`	Add a file to the repository and commit the change immediately. The repository must be previously initialized.
`cat`	Show the displayable contents of a file.
`commit`	Commit any outstanding file changes to the repository.
`del`	Delete a file from the repository. The file in the working directory is unaffected.
`diff`	Display any local changes that have been made to a file or files.
`help`	Display help text.
`init`	Initialize the CM repository for use with an Altair Accelerator project. This creates the repository (`*.cmrepo`) file in the NC_CONFIG_DIR directory (usually `$VOVDIR/local/vncConfig/`)
`ls`	List project files that are currently under change control (use `-v` to also show file status).
`open`	Open a repository and check out the files to the current working directory.

`quickstart`	Initialize the repository and add the most common files (as listed in the previous section). This is typically done on a single queue to act as master.
`revert`	Roll back any existing local changes to a file or files.
`status`	Show the status of the local checkout set in the working directory.
`timeline`	Show the timeline of changes to the repository.
`undo`	Undo previous update or revert action.
`update`	Update local file(s) with the latest from the repository, or a specific revision.

CM Command Usage

```
vncmgr: Usage Message

  DESCRIPTION:
        "ncmgr cm ..." is the utility for performing configuration management
        actions.  To use configuration management, you need to be in the
        server-working directory (SWD) of the current project.
        To work with a non-default queue name, either set the NC_QUEUE
        environment variable to the target queue name or use either of the
        "-queue" or "-q" arguments.

  USAGE:
          % ncmgr cm [-queue|-q QUEUE] <ACTION> [ARGUMENTS]

  ACTIONS: (arguments are either <>=required or []=optional)

          add       <FILE>  -- Add a file to the repository.
          cat       [-r REVISION] <FILE>
                            -- Display contents of a file.
          commit    <-m "MSG"> [FILE]
                            -- Commit changes to the repository.
          del       <FILE>  -- Delete a file from the repository.
          diff      [-r REVISION] <FILE>
                            -- Display local changes.
          help              -- Display this help.
          init              -- Initialize repository.
          ls        [-v] <FILE>
                            -- List managed files.
                               Use -v to also show file status.
          open              -- Open a repository and check out files to the
                               current working directory.
          quickstart        -- Initialize repository and add most common files.
                               Typically done on a single queue to act as
                               master.
          revert    [FILE]  -- Roll back local changes.
          status            -- Show status of the local checkout.
          timeline  [FILE]  -- Show timeline of changes.
          undo      [FILE]  -- Undo previous update or revert action.
          update    [-r REVISION] [FILE]
                            -- Update local file(s) with the latest from the
                               repository, or a specific revision.

        NOTE: For configuration management, you must be in the server working
        directory (SWD) of the project being managed.
        If you are unsure of this, please run the following commands:
        % vovproject enable <project_name>
        % cd `vovserverdir -p .`

  EXAMPLES
          % ncmgr cm quickstart
          % ncmgr cm init
          % ncmgr cm add policy.tcl
          % ncmgr cm commit -m "Added new tasker." taskers.tcl
          % ncmgr cm add scripts/myscript.tcl
          % ncmgr cm commit -m "Added custom script to CM"
                       scripts/myscript.tcl
```

Use CM in a Test Instance

Implementing a configuration management process for your testing workflow can save a lot of time and hassle in keeping configuration files synchronized.

An example follows with a production instance called "vnc" and a test instance called "vnctest". For purposes of this example it is assumed that the repository has been previously set up, all desired config files have been added, and that two files have been modified and need to be synced to the test instance. The process consists of the following 2 steps:

1. Enable the production instance.

   Snapshot the selected production config files with the `ncmgr cm commit` command.

   ```
   vovproject enable vnc
   cd `vovserverdir -p .`
   ncmgr commit -m 'save changes to taskers.tcl and policy.tcl' taskers.tcl
    policy.tcl
   ```

2. Enable the test instance.

   Restore (synchronize) the same config files to the test setup using the `ncmgr cm open` command.

   Use the production instance as the path to the repository.

   ```
   vovproject enable vnctest
   cd `vovserverdir -p .`
   ncmgr cm -q vnc open
   ```

Browser-based Setup

Once the Accelerator vovserver has been started, you can view and manage it using your web browser.

The Setup Page

The URL for the Accelerator vovserver can be found with the following command:

```
% nc cmd vovbrowser
http://somehostname:6271/project
```

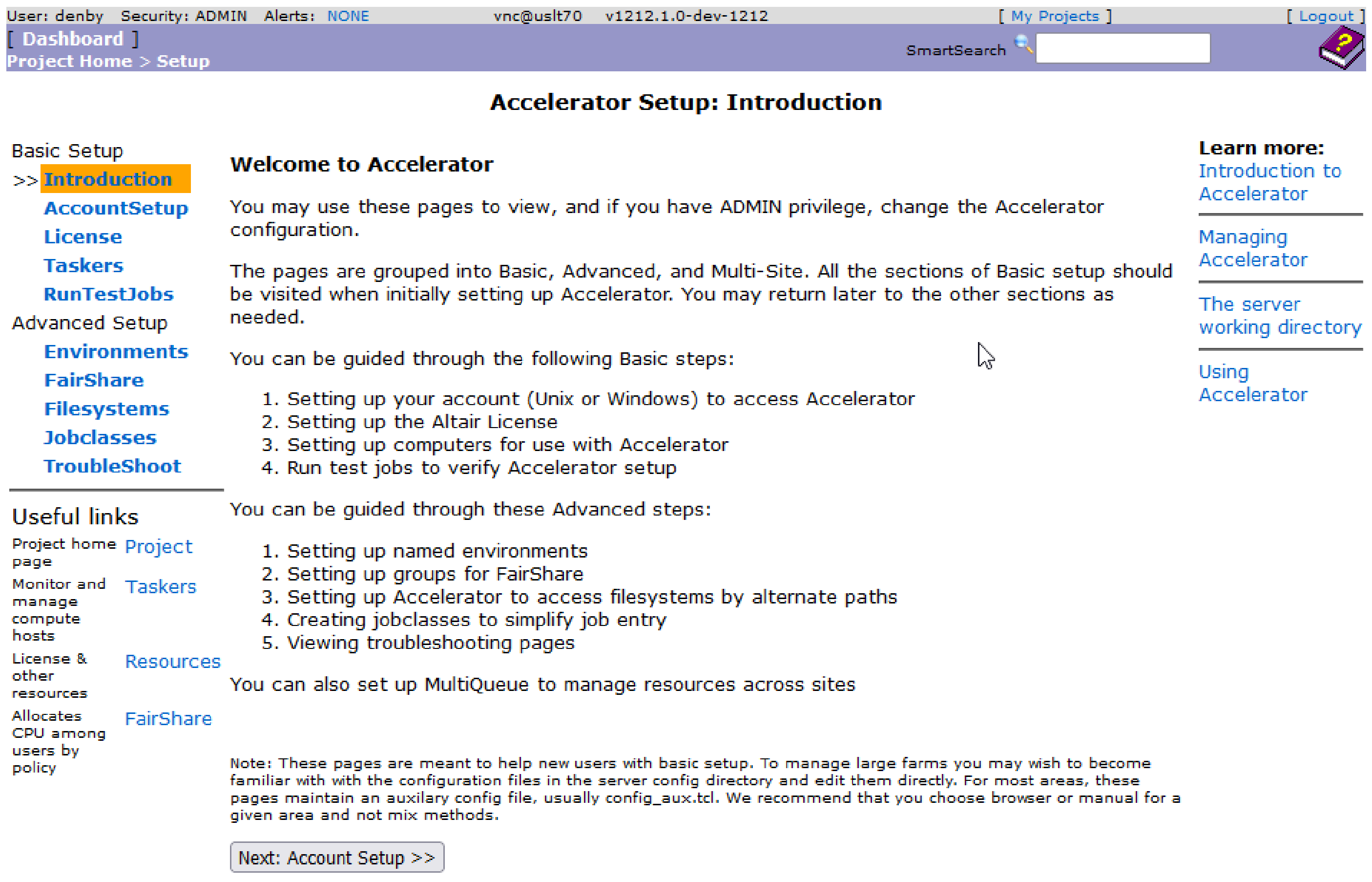
The setup page is available at the URL /cgi/setup.cgi.

```
% nc cmd vovbrowser -url /cgi/setup.cgi
```

```
http://hostname:6271/cgi/setup.cgi
```

Web Interface Screens

The **Introduction** page provide an overview of the available information. As indicated in the left column, menus and guidelines are available for setting up and using the Accelerator features. More detailed information and advanced methods are provided in this document in other chapters.


User: denby Security: ADMIN Alerts: NONE vnc@uslt70 v1212.1.0-dev-1212 [My Projects] [Logout]
[Dashboard]
Project Home > Setup SmartSearch

Accelerator Setup: Introduction

Basic Setup
>> Introduction
AccountSetup
License
Taskers
RunTestJobs
Advanced Setup
Environments
FairShare
Filesystems
Jobclasses
TroubleShoot

Useful links

Project home page	Project
Monitor and manage compute hosts	Taskers
License & other resources	Resources
Allocates CPU among users by policy	FairShare

Welcome to Accelerator

You may use these pages to view, and if you have ADMIN privilege, change the Accelerator configuration.

The pages are grouped into Basic, Advanced, and Multi-Site. All the sections of Basic setup should be visited when initially setting up Accelerator. You may return later to the other sections as needed.

You can be guided through the following Basic steps:

1. Setting up your account (Unix or Windows) to access Accelerator
2. Setting up the Altair License
3. Setting up computers for use with Accelerator
4. Run test jobs to verify Accelerator setup

You can be guided through these Advanced steps:

1. Setting up named environments
2. Setting up groups for FairShare
3. Setting up Accelerator to access filesystems by alternate paths
4. Creating jobclasses to simplify job entry
5. Viewing troubleshooting pages

You can also set up MultiQueue to manage resources across sites

Note: These pages are meant to help new users with basic setup. To manage large farms you may wish to become familiar with with the configuration files in the server config directory and edit them directly. For most areas, these pages maintain an auxilary config file, usually config_aux.tcl. We recommend that you choose browser or manual for a given area and not mix methods.

Next: Account Setup >>

Learn more:
Introduction to Accelerator
Managing Accelerator
The server working directory
Using Accelerator

Figure 3:

Regulate Access to Accelerator

To regulate access to Accelerator, the security file `vnc.swd/security.tcl` must be edited. To deploy the changes after editing the file, Accelerator must be reset.

The format of the security file is:

```
vtk_security username|-group vovusergroupsecurity-levelhost1 ...
```

The security levels are USER, LEADER, and ADMIN. Security-level roles can be assigned to users or groups of users (VovUserGroups). VovUserGroups are created from user lists manually, or by associating them with existing UNIX or LDAP groups. For more information about VovUserGroups, refer to *VovUserGroups* in the *VOV Subsystem Administrator Guide*.

Both the administrator and root should be granted the ADMIN role; all others should be granted the USER role.

Note: For vovtaskers to work correctly, root must also be granted the ADMIN role.

Disable Security

Security can be disabled by allowing unrestricted access to all users to Accelerator; assign everyone ADMIN privileges from all hosts.

```
# This is the vnc.swd/security.tcl file needed to disable security.
# The first + means "everybody"
# The second + means "from all hosts"
vtk_security + ADMIN +
```

Enable Security

To control access to Accelerator, activate security. The minimum requirement is that both the administrator and the root of Accelerator must have the ADMIN level of privilege from all hosts.

Shown below is an example of a security file. In this example, the system uses five computers. User "john" can only use the system from host `ws1`, and user "susie" can use the system from hosts `ws1` and `ws2`. Users in the VovUserGroup `mygroup` have ADMIN rights on the set of hosts known as `$allHosts`.

```
# Example of vnc.swd/security.tcl file
set allHosts { ws1 ws2 apple orange pear }
vtk_security vncadmin ADMIN  $allHosts
vtk_security root     ADMIN  $allHosts
vtk_security john     LEADER ws1
vtk_security susie    LEADER ws1 ws2
vtk_security -group mygroup  ADMIN  $allHosts
vtk_security +        USER   +
```

After configuring the security file, Accelerator must be reset to apply those the changes:

```
% ncmgr reset
```

Autostart Directory

With the command `vovautostart`, on vovserver startup, scripts can be specified to execute automatically.

In UNIX, the scripts can be written in either C-shell or Tcl syntax.

Note: For the script to work in Windows, Tcl syntax must be used. Guidelines follow:

- Create a directory named `autostart` in the server working directory.
- For both UNIX and Windows:

  - Create a script with the suffix `.tcl` in the `autostart` directory.
- For UNIX only, CSH scripts are also supported:
  - Create a script with the suffix `.csh` in the `autostart` directory.
  - Ensure the script has the appropriate executable permissions.

Each script in the `autostart` directory is called with one argument, which is the word *`start`*. This argument is usually ignored in OEM scripts, but can be used to in custom scripts to enforce different behaviors between a manual call on the CLI versus an automated call by the vovserver.

Examples are available in the directory `$VOVDIR/etc/autostart.`

vovautostart

The scripts are launched by the utility `vovautostart`. To repeat the execution of the `autostart` scripts, `vovautostart` can be executed from the command line.

```
vovautostart: Usage Message

  DESCRIPTION:
      Execute the scripts in the *.swd/autostart directory.
      There are two types of scripts that get executed:
      1.  Scripts that match *.csh are executed directly (Unix only)
      2.  Scripts that match *.tcl are executed by vovsh.

      The scripts are executed in alphabetical order in the background, with a 5s
      delay between successive scripts.

      This utility is normally invoked by vovserver upon launching.

  USAGE:
      % vovautostart [optional directory spec]

  EXAMPLES:
      % vovautostart
```

Autostop Directory

With the command `vovautostop`, on vovserver shutdown, scripts can be specified to execute automatically.

In UNIX, the scripts can be written in either C-shell or Tcl syntax.

Note: For the script to work in Windows, Tcl syntax must be used. Guidelines follow:

- Create a directory named `autostop` in the server working directory.
- For both UNIX and Windows:
  - Create a script with the suffix `.tcl` in the `autostop` directory.
- For UNIX only, CSH scripts are also supported:
  - Create a script with the suffix `.csh` in the `autostop` directory.
  - Ensure the script has the appropriate executable permissions.

Each script in the `autostop` directory is called with one argument, which is the word `stop`. This argument is usually ignored in OEM scripts, but can be used to in custom scripts to enforce different behaviors between a manual call on the CLI versus an automated call by the vovserver.

Examples are available in the directory `$VOVDIR/etc/autostop`.

The vovautostop Command

The scripts are launched by the utility `vovautostop`. To repeat the execution of the `autostop` scripts, `vovautostop` can be executed from the command line.

```
vovautostop: Usage Message

  DESCRIPTION:
      Execute the scripts in the *.swd/autostop directory.
      There are two types of scripts that get executed:
      1.  Scripts that match *.csh are executed directly (Unix only)
      2.  Scripts that match *.tcl are executed by vovsh.

      The scripts are executed in alphabetical order,
      in the background, and
      with a 5s delay between successive scripts.

      This utility is normally invoked by vovserver upon shutdown.

  USAGE:
      % vovautostop [OPTIONS] [optional directory spec]

  OPTIONS:
     -v            Increase verbosity.
     -h            This usage message.

  EXAMPLES:
      % vovautostop
```

Test Altair Accelerator

You may wish to run a few simple jobs to confirm that the system is running properly.

1. Submit a simple job with the following commands:

```
% nc run sleep 30
```

```
Resources= linux
Env      = SNAPSHOT(vnc_logs/envjohn36829.csh)
Command  = sleep 30
Logfile  = vnc_logs/20020602/105731.27873
JobId    = 00002539
nc: message: Scheduled jobs: 1       Total estimated time: 0s
```

- The `JobId` in this example is 2539. The leading zeros are insignificant.
- The environment used for the job execution is a snapshot of the current environment, stored in the file `vnc_logs/envjohn36829.csh`
- The default resource list for the job is `linux` which, in this example, is the platform from which the job was submitted. Later, you will learn how to control the resources assigned to a job.

Note: If the following dialog appears, see *Qualify the Working Directory of Jobs* in the Altair Accelerator Administrator Guide.

```
WARNING:
 The current directory '/export/scratch/john' may not be a valid
directory path for all hosts in the cluster.

 EXPLANATION:

 If the job runs on a host other than the local node, and the current
directory is not shared on that host, unpredictable results may occur.

 OPTIONS:
   (1)   Continue.
         By choosing this option, you assert that the path
         is valid everywhere.  If it is not, the job is likely
         to fail, because the remote host cannot reach
         the current directory.
         This option causes the creation of a flag file
         called   .vnc    which has the purpose of
         avoiding the repetition of this question for this
         directory and its subdirectories
   (2)   Abort.
         Please ask your Accelerator administrator to
         change the equiv.tcl file to define the rules that
         give a logical name to the current directory.

  Please reply: [1,2] >
```

2. Check the status of the job:

```
% nc list
00002539 Done    sleep 10
% nc summary
...output omitted
% nc info 2539
Id,User,Group    00002539,john,users
Environment      SNAPSHOT(vnc_logs/envjohn36829.csh)
Directory        ${HOMES}/john
Command          sleep 10
Status           Running
Host          alpaca
Resources     linux
QueueTime     0s
```

```
RunningTime  9s
```

3. Check the output of the job, which in this case is empty, since it is the output of `sleep`.

```
% nc info -l 2539
Log file is: '${HOMES}/john/vnc_logs/20020602/105731.27873'
```

4. Rerun the job.
   The first time Accelerator notices that the job is already Done (=VALID) so it does not run it again. The second time, the option -f forces Accelerator to rerun the job even if it is already Done.

```
% nc rerun 2539
nc rerun 00002539
nc: message: Not rerun: 00002539
% nc rerun -f 2539
nc: message: Job 00002539 is already VALID.
nc: message: Scheduled jobs: 1       Total estimated time: 15s
```

5. Stop the running job. The job will fail.

```
% nc stop 2539
```

6. Forget the job:

```
% nc forget 2539
nc: message: Forgetting 1 jobs
```

7. Get a quick report:

```
% nc summary
Accelerator Summary For User john
TOTAL JOBS        105      Duration: 4m17s
Failed             53
Queued             50
Idle                2

JOBS   GROUP    TOOL        WAITING FOR...
50   pi       vtclsh     'hpux#1 '
```

Server Error Conditions

VOV and the jobs it runs depend on external resources such as available licenses, RAM, swap, tmp and disk space, which may become exhausted, damaged or otherwise unavailable. When users are subject to disk quotas, writes may fail when the disk is not completely full.

VOV License Violation

Listed below are possible causes for a license violation:

- Invalid license file, such as wrong version or damaged
- Expired license file
- License server never started
- No license server running, deliberately stopped or crashed
- Licensed quantities exceeded, such as too many vovtaskers

Server License Violation Behavior

When a violation is detected, retracing and dispatching new jobs are stopped.

Disk Full and Quota - vovserver

The `vovserver` program tracks the free disk space on the file system where its working directory (such as `PROJECTNAME.swd`) is placed, and also on `/usr/tmp` where some temporary files are stored. The vovserver issues loud alerts when disk resources are exhausted.

The vovserver disk full warning is sticky, and persists after the disk has been cleaned up and has free space. The warning is sent to any client that tries to connect to the vovserver until the warning state has been cleared. The disk full warning causes most vovsh-based commands, such as `vovconsole`, to fail.

Because disk full is a severe error and can cause many cascading errors, to ensure the issue is noticed and addressed, this warning is deliberately sticky; this prevents a potentially critical alert from becoming a static message that is buried and unnoticed in a vovserver log file.

The `vovproject sanity` command can be used to clear the warning and return to normal operation. An example is shown below.

To clear disk full warning:

```
% vovproject enable your-project
thishost your-project@srvhost ENV dir> vovproject sanity
```

For Accelerator:

```
% /bin/su - FT-admin  # login as the owner of Accelerator
% nc cmd vovproject sanity
```

Disk Full and Quota - vovtasker

The vovtasker program also checks for free space on `/usr/tmp` on the host where it runs, and suspends itself (refuses to accept new jobs) if the amount falls below a configurable amount. The default is 5MB, which can be configured using the -mindisk option of vtk_tasker_define.

The tasker suspended condition is not sticky. After the tasker host disk has been cleaned up so that free space is above the threshold, the vovtasker will automatically resume and enter the ready state.

Configure a Failover Server Replacement

If a server crashes suddenly, VOV has the capability to start a replacement server on a pre-selected host. This capability requires that the pre-selected host is configured as a failover server.

The configuration instructions follow.

Note: The `vovserverdir` command only works from a VOV-enabled shell when the project server is running.

1. Edit or create the file `servercandidates.tcl` in the server configuration directory. Use the `vovserverdir` command with the -p option to find the pathname to this file.

   ```
   % vovserverdir -p servercandidates.tcl
   ```

```
/home/john/vov/myProject.swd/servercandidates.tcl
```

The `servercandidates.tcl` file should set the Tcl variable *ServerCandidates* to a list of possible failover hosts. This list may include the original host on which the server was started.

```
set ServerCandidates {
    host1
    host2
    host3
}
```

2. Install the `autostart/failover.csh` script as follows:

```
% cd `vovserverdir -p .`
% mkdir autostart
% cp $VOVDIR/etc/autostart/failover.csh autostart/failover.csh
% chmod a+x autostart/failover.csh
```

3. Activate the failover facility by running `vovautostart`.

```
% vovautostart
```

For example:

```
% vovtaskermgr show -taskergroups
ID          taskername        hostname          taskergroup
000404374  localhost-2      titanus          g1
000404375  localhost-1      titanus          g1
000404376  localhost-5      titanus          g1
000404377  localhost-3      titanus          g1
000404378  localhost-4      titanus          g1
000404391  failover         titanus          failover
```

Note: Each machine listed as a server candidate must be a vovtasker machine; the vovtasker running on that machine acts as its agent in selecting a new server host. Taskers can be configured as dedicated failover candidates that are not allowed to run jobs by using the -failover option in the taskers definition.

Preventing jobs from running on the candidate machine eliminates the risks of machine stability being affected by demanding jobs. The -failover option also enables some failover configuration validation checks. Finally, failover taskers are started before the regular queue taskers, which helps ensure a failover tasker is available as soon as possible for future failover events.

Refer to the tasker definition documentation for details on the -failover option.

How vovserver Failover Works

If the vovserver crashes, after a period of time, the vovtasker process on each machine notices that it has had no contact from the server, and it initiates a server machine election.

In this election, each vovtasker votes for itself (precisely, the host that this particular tasker runs on) as a server candidate. The election is conducted by running the script `vovservsel.tcl`.

After the time interval during which the vovtaskers vote expires, (default 60 seconds) the host that appears earliest on the list will be selected to start a new vovserver.

In the following example, the `servercandidates.tcl`, file contains three hosts:

```
set ServerCandidates {
    host1
    host2
    host3
}
```

When the server crashes, if there are vovtaskers running on host1, host2 and host3, then these hosts will be voted as server candidates. Then host2 will be the best candidate and a new vovserver will be started on host2. This server will start in crash recovery mode.

Note: For failover recovery to be successful, an active `vovtasker` process must be running on at least one of the hosts named in the `ServerCandidates` list. Usually, these vovtaskers have been defined with the -failover option so they can not accept any jobs, and are members of the `failover` taskergroup.

The failover vovserver will read the most-recently-saved PR file from the `.swd/trace.db` directory, and then read in the transactions from the 'cr*' (crash recovery) files to recover as much of the pre-crash state as possible.

The vovserver writes a new `serverinfo.tcl` file in the `.swd` that vovtaskers read to determine the port and host. When it starts, the failover vovserver appends the new host and port information to the `$NC_CONFIG_DIR/<queue-name.tcl>` as well as to the `setup.tcl` in the server configuration directory. The vovserver then runs the scripts in the autostart directory. This should include the `failover.csh` script, which resets the failover directory so that failover can repeat. This script removes the registry entry, and removes the `server_election` directory and creates a new empty one. At the end, it calls `vovproject reread` to force the failover vovserver to create an updated registry entry.

The failover vovserver remains in crash recovery mode for an interval, usually one minute, waiting for any vovtaskers that have running jobs to reconnect:

- For Accelerator, Accelerator Plus, Monitor and Allocator, vovtaskers wait up to 4 days for a new server to start.
- For FlowTracer, vovtaskers wait up to 3 minutes for a new server to start.

After reconnecting to vovserver, vovtaskers automatically exit after all of their running jobs are completed. After the vovserver transitions from crash recovery mode to normal mode, it will try to restart any configured vovtaskers that are not yet running.

Any of the following conditions will prevent successful failover server restart:

- The filesystem holding the `.swd` directory is unavailable.
- The file `servercandidates.tcl` does not exist.
- The `ServerCandidates` list is empty.
- There is no vovtasker running on any host in the `ServerCandidates` list when the server crashes.
- The `autostart/failover.csh` script file is not in place.

In this case, the failover server will not be automatically started; the server will have to be manually started.

Tips for Configuring Failover

Following are tips for failover configuration:

- Make the first failover host the regular one. This way, if the vovserver dumps core or is killed by mistake, it will restart on the regular host.

- Configure special vovtaskers only for failover by passing the -failover option to `vtk_tasker_define`.
- Test that failover works before depending on it.

Migrating vovserver to a New Host

The failover mechanism provides the underpinnings of a convenient user CLI command that can be used to migrate vovserver to a new host:

```
ncmgr rehost -host  NEWHOST
```

The specified NEWHOST must be one of the hosts eligible for failover of vovserver.

Crash Recovery Mode

Crash Recovery Mode is activated the next time the server is restarted, if the server was not shut down cleanly. Crash Recovery Mode is part of the Failover Server capability of VOV, which is mainly used in Accelerator. This capability allows VOV to start a new server to manage the queue of a server which has crashed unexpectedly.

When you shut down an Accelerator instance cleanly using the `ncmgr stop` command, the server will save its database to disk just before exiting. When the server is restarted, it will read the state of the trace from disk, and immediately be ready for new work.

Sometimes the vovserver will be stopped unexpectedly, such as due to a hardware problem like a machine crash or memory exhaustion. In such cases, the server will not have a chance to save the project database before terminating.

- In VOV, the main concern is usually the state of the trace, which stores the status all the jobs in your project.
- In Accelerator, there is no trace, and the important thing to preserve is the state of the queue, so jobs do not lose their position and need to be re-queued in the case of a server crash.

Journal Files

The vovserver keeps crash recovery journal files of the events that affect the state of the server. These 'CR' files are flushed whenever the trace data are saved to disk. During crash recovery, the vovserver first reads the last saved state of the trace from the disk data, then applies the events from the CR files.

Crash Recovery Restart

When the server is next restarted after such a crash, the server enters what is called Crash Recovery Mode, which usually lasts about two minutes, but may take longer if the CR files are very large. During this period:

- The server waits for vovtaskers with running jobs to reconnect.
- No jobs are dispatched.
- The server does not accept VOV or HTML TCP connections from `vovsh` or browser clients.
- At the end, the server performs a global sanity check.
- A crash_recovery_report <timestamp> logfile is written. It logs any jobs lost by the crash recovery sequence.

If the server is properly shut down, the next time it restarts, it will not enter Crash Recovery Mode, and will be immediately functional.

Note: If the sanity check command cannot connect, the server is still recovering its state from the DB and CR files. Check the size of the `vnc.swd/trace.db/CR*` files.

Example commands:

```
% vovproject enable <project-name>
% vovproject sanity
```

Accelerator Daemons

Additional functionality in Accelerator is provided by external daemons, which are described in the table below.

The status of the daemons can be viewed on the Daemons page.

Daemon	Who needs it?	Description
`vovnotifyd`	If you want e-mail notification	This daemon is necessary to receive email notifications. The email can be triggered by job events (such as a job that completes) or an unusual condition that is detected by the daemon. For more information, see Health Monitoring and vovnotifyd.
`vovresourced`	Everybody	This daemon manages the resources managed by the vovserver. It is controlled by the `resources.tcl` file. Historically, this is the first daemon to be developed. It still stands out from the other daemons because how it is managed is slightly different. This daemon is started automatically by the server. For more information, see Resource Daemon Configuration.

Access Control List

An Access Control List (ACL) is a list of permissions that are attached to an object. The list defines who can access the object (an agent) and what actions the agent can perform on the object.

Overview

The VOV software implements compartmentalized access control with Access Control Lists (ACLs). Each ACLs is a triplet:

- A VOV object
- An *agent*, which is VOV security role name or an individual user name
- A *capability*, which is a controlled activity

For any user to be authorized to perform a controlled capability or action on a VOV object, an ACL must exist that contains that user or role, the controlled action, and the VOV object.

Objects

Every ACL is associated with a VOV object. Types of objects currently include:

- FairShare groups
- Resource maps
- Nodes (transitions, aka jobs, places, aka files)
- Node sets
- Reservations

Agents

Permission to perform a controlled action depends on the user ID, and the VOV role associated with that user. The `SWD/security.tcl` file defines the association of Users with Roles.

VOV's roles serve two purposes:

1. Control queue/instance/project operations (via the VOV protocol)
2. Establish high-level permissions on VOV objects

VOV has these named roles:

ADMIN	Can do just about anything. By default, the "owner" of the queue/instance/project is the only admin.
LEADER	Can do lots of things, but not everything an ADMIN can.
USER	Can create and manage their own objects.
READONLY	Can view most things, but not create.
ANYBODY	Very limited, mainly used for testing.
NOBODY	Nothing, mainly used for testing.
UNKNOWN	The last resort, in case we somehow encounter someone who doesn't fall into any of the above roles, also can do nothing.

An ACL is expressed in terms of operations that are permitted to an *agent* acting on the object. An agent may be a USER (login account), an OS group (OSGROUP), a FairShare group (FSGROUP), a machine (HOST) or one of the symbolic agents EVERYBODY, OWNER, ADMIN. The most powerful agent is the SERVER.

ACLs support the following "agents", which provide the identities of the persons involved:

USER *name*	OS authorized user
OWNER - role	The queue/instance/project owner
USER - role	The user that owns the VOV object
OSGROUP *name*	Members of the specified OS group
FSGROUP *name*	Members of the specified FairShare group
HOST	Anyone with a client connected from a specific host
EVERYBODY - role	Everybody
ADMIN - role	Anyone with the ADMIN role
SERVER	The vovserver process, specifically
LEADER	Anyone with the LEADER role
USERGROUP *name*	Members of the specified VOV user group
UNDEF	The last resort, in case there is someone who doesn't fall into any of the above agent types.

For the agents that are groups, membership in the group confers the operations permitted by that ACL. For example, if the login `joe` is a member of the OS group `dvregr`, and OSGROUP `dvregr` has APPEND on a `fsgroup`, then `joe` may add ACLs to that `fsgroup`.

To bypass the ACL, you must be the logged in on the host running vovserver as the user that is running vovserver, and you must change VOV_HOST_NAME to "localhost".

ACL Management

To perform ACL management, use a utility with the following syntax:

```
% vovacl [OPTIONS]  <Objects>
```

The following utilities are available for ACL management:

Utility	Description
vovacl	Script to manage ACLs in VOV.

ACL Commands

ACL management consists of the following commands:

Command	Description
APPEND	Add ACLs to an object.
DELETE	Delete an ACL element from an object. The element is identified by the agent and name fields.
GET	Get current ACLs on an object. It shows you the current ACLs that are associated with an object, if the ACL permits you to VIEW it.
RESET	Reset ACLs on an object to defaults. It removes all the object's current ACLs and replaces them with the default values. `ACL  1: OWNER      ""   ATTACH DETACH` ` EDIT VIEW FORGET DELEGATE EXISTS` `ACL  2: EVERYBODY  ""   ATTACH VIEW`

ACL Actions

Following are the actions that can be controlled via ACLs:

Action	Description
ATTACH	Create a relationship between objects
CHOWN	Change ownership of an object
CREATE	Create an object
DELEGATE	Assign ACLs on an object
DETACH	Destroy a relationship between objects
EDIT	Modify properties of an object
EXIST	The agent is aware of the existence of the object
FORGET	Forget an object
RESUME	Resume a suspended job
SIGNAL	Send a signal to a job.
STOP	Stop an object.

Action	Description
SUSPEND	Suspend an object
VIEW	View properties of an object

Note: Not all actions apply to all objects. In the case of FairShare groups, applicable actions include: ATTACH, EDIT, VIEW, DELEGATE. The actions RETRACE, STOP, SUSPEND, FORGET are reserved for use with jobs in future releases.

Obtain SERVER Credentials

For some ACL operations, you will need the most powerful credentials, i.e. SERVER, which are only available to the owner of the vovserver process when connected on the loopback interface.

- Login on the vovserver host as the user that is running vovserver.
- Enable the project with `vovproject enable PROJECTNAME`.
- Change the VOV_HOST_NAME to `localhost`

  ```
  % setenv VOV_HOST_NAME localhost
  ```
- Now your clients act as the SERVER agent with respect to the ACL.

vovacl

```
vovacl: Usage Message

 DESCRIPTION:
  Manage access control lists (ACLs).

 USAGE:
    % vovacl [OPTIONS] <Object>

 OPTIONS:
    -h         -- This help
    -v         -- Increase verbosity
    -agent     -- Association of specified ACL. One of the following types and
                  formats. ACL capabilities for ALL pertinent agents for a
                  given user are aggregated.
                      "USER name"              OS user name
                      "FSGROUP name"           VOV FairShare group
                      "USERGROUP" name         A VOV user group
                      "OSGROUP" name           Unix primary group (Linux only)
                       OWNER
                       ADMIN
                       LEADER
                  Different VOV object types honor different subsets of the
                  agents listed above. To see which apply, see the Agents Table
                  below.

    -actions   -- Capabilities list for this ACL, space delimited. A list of
```

```
                  access control capabilities is shown in the Capabilities
                  Table below.
     -append   -- Add specified capabilities to ACL for specified agent
     -set      -- Replace existing ACLs for specified agent
     -delete   -- Remove specified capabilities from specified agent
     -show     -- Show current ACL for specified objects.
     -reset    -- Reset ACL to default values

  OBJECTS:
      <setName>
      <fairshareGroupName>
      <vovId>  -- Where vovId can apply to a job, set, FairShare group, or
                  resourcemap

  AGENTS TABLE
      The agents list specified with the -agent option is one of the following
      that is valid for the object type.

      Agent Type               Type of Object

                     Set/Job        FS Group        Resourcemap

      USER              y              y                 y
      USERGROUP         n              n                 y
      FSGROUP           n              y                 n
      OSGROUP           y              n                 n
      OWNER             y              y                 y
      ADMIN             y              y                 y
      LEADER           n/a            n/a               n/a
      EVERYBODY         y              y                 y

  CAPABILITIES TABLE
      The capabilities list specified with the -agent option is one of the
      following that is valid for the object type.

      Capability Name              Type of Object

                         Set/Job        FS Group        Resourcemap

      ATTACH               n/a           attach           use
      CHOWN                n/a            n/a             n/a
      CREATE               n/a          create sub-group  n/a
      DELEGATE             n/a            n/a             n/a
      DETACH               n/a           detach           n/a
      EDIT                modify         modify          modify
      EXISTS           preq-for-all      prereq          prereq
      FORGET              delete         delete          delete
      RESUME              resume          n/a             n/a
      RETRACE              run            n/a             n/a
      SIGNAL              signal          n/a             n/a
      STOP               job stop         n/a             n/a
      SUSPEND            suspend          n/a             n/a
      VIEW                view           view            view

  EXAMPLES:
    % vovacl -agent ADMIN -append -actions "VIEW RETRACE STOP FORGET" MySetName
    % vovacl -agent "USER cadmgr" -append -actions STOP  /system/processcontrol
    % vovacl -agent "USERGROUP designers" -append -actions STOP,FORGET DesignSet
    % vovacl -agent "OSGROUP designers" -append -actions STOP  00123456
    % vovacl -reset 00123456

  NOTE: The SUSPEND action is not applicable to an FSGROUP object.
```

Client Limitation and Tuning

The maximum number of clients - the combination of vovtaskers, user interfaces and proxies, that can be concurrently connected to a vovserver is limited by the number of file descriptors available.

This is an operating system parameter, and is inherited from the shell that starts vovserver. It can not be changed after vovserver starts.

There are two kinds of limits, a hard limit and a soft limit. Limits are imposed to reduce the likelihood of exhausting system resources. A soft limit may be set by shell commands so long as a value less than or equal to the hard limit is selected. The hard limits for descriptors and other resources may vary by user, group, and other attributes.

On UNIX, this number is operating system dependent. In most UNIX installations, the hard limit for file descriptors is 1024 or more.

On Windows NT, VOV sets the limit at 256 file descriptors.

On Linux, `root` can change the limits in the file `/etc/security/limits.conf`. Example:

```
* hard nofile 8192
* soft nofile 2048
```

The above example sets the soft limit for all users to 2048, and the hard limit to 8192. The '*' character could be replaced by that of the Accelerator owner account, e.g. 'cadmgr'.

Background

Each operating system offers a limited number of file descriptors for each process. In modern systems, this limit may be up to 65000. The vovserver can handle as many clients as the "descriptors limit" allows. It is also possible to reduce the number by setting a soft limit using the methods described above.

To allow a large number of clients, vovserver must be started with a high limit. The `ncmgr` command reports the number at startup time, please read it carefully before replying 'yes'. Example:

```
% limit descriptors 16000
% ncmgr start
```

The file descriptors are used by the vovserver to communicate with the clients: vovtaskers, GUI, browser, interactive jobs, etc.

The utilization of file descriptors are approximately:

- The server by itself needs about 10
- The other descriptors less than 40 are not used
- Each vovtasker needs 1
- Each running batch job needs 1
- Each running interactive job needs 2
- Each vovconsole needs 2
- Each nc monitor needs 1

Example: On a loaded farm with 500 vovtaskers, each with four CPUs, with half jobs interactive, the estimated number of descriptors needed is:

```
10 + 500 + 500 * 2  + 500 * 2 * 2 =  3510 file descriptors
```

This leaves descriptors for about 580 monitors and GUIs.

Behavior with Exhausted File Descriptors

The exhaustion of file descriptors rarely occurs. Altair Accelerator's main concern is preserving the integrity of the vovserver. Commands that attempt a new connection to the vovserver fail to connect and return an error message too many `clients in the system`. The vovserver will then post an alert showing `too many open files`. In that condition, ordinary commands such as `nc hosts` and `nc list` will not work because the `vovsh` that runs those commands cannot connect to vovserver.

Reserved Connection on the localhost

When file descriptors are exhausted, you can connect to vovserver using a special method through the software loopback interface (lo0, 127.0.0.1). This is achieved by setting VOV_HOST_NAME to localhost. Example:

```
% vovproject enable vncNNNN
% setenv VOV_HOST_NAME localhost
% vsi
```

Solutions to File Descriptor Exhaustion

A short-term solution is to stop some of the clients is to lower the demand for file descriptors. Transient GUI clients such as VovConsole, monitors, and Accelerator GUI should be stopped first. Any idle vovtaskers should also be stopped. Guidelines:

- Check how users are submitting jobs. There are some limits on *`maxNormalClients`* and *`maxNotifyClients`* in the `policy.tcl` file to prevent accidental or malicious denial-of-service attacks. Sometimes we have seen jobs submitted with the -wl option and placed in background, each consuming 2 descriptors.
- Next you should first find whether it is possible to raise the descriptor limit on the current host. If not, a longer-term solution is to move the vovserver to another host that offers more file descriptors. A newer version of UNIX is a good candidate as it offers 65K descriptors.

It is possible to continue operation, even in the presence of interactive jobs, by moving the vovserver and making the new queue the default queue so that newly-submitted jobs go to the new default queue. The server on the host with limited descriptors will finish all jobs, and it may then be shut down.

Client Service Modes

On Linux-based systems, there are two client servicing modes from which to choose: poll (default) and epoll. The mode chosen specifies which POSIX mechanism the vovserver will use to determine which client file descriptors are ready for use. The mode can be specified by setting `config(useepoll)` to 0 (poll, default) or 1 (epoll) in the `SWD/policy.tcl` file.

Generally, the epoll mode should result in more efficient processing of service requests. As of this version, epoll mode is a new feature and is therefore disabled by default.

Enabling Time Series Data Stream

Users are now able to leverage existing Kafka systems and compatible reporting tools to monitor VOV projects.

When Kafka is used as part of an infrastructure, multiple vovservers can now be enabled to provide time series data and events, without negatively impacting server performance/scalability. This allows users to capture time varying data in order to see how usage evolves over time.

Events Frequency

The metrics events are published at the same rate that metrics are calculated in the server. This will vary by load but should be at most every 10s on an active server, and possibly longer if the server is heavily loaded such that the scheduling cycle takes longer than this. The project data is expected to be relatively static and is published every 4 hours. The command: `vovservermgr config sds.readconfig 1` will cause the updated project record to be published on execution.

Project IDs

To enable data from multiple projects to be collected on the same kafka infrastructure, each event will contain the field **projId** which identifies the project which published the event. The projId field is formed by concatenating the project name, a hyphen, and the vovserver instance's numeric generated unique id. e.g. "vnc-12345678"

SDS Configuration

On startup, the vovserver will create and/or update the following configuration items:

- SDS configuration directory, at the server working directory (SWD)**/config/publishers/sds**, for example, `../vnc.swd/config/publishers/sds`
- SDS configuration file in the SDS configuration directory, `sds.cfg`

If the `sds.cfg` file does not already exist, the following default `sds.cfg` file is created:

```
{
  enabled = 0;
  kafka_servers = "",
  format = "json",
  site = "",
  group = "",
  events = {
    project = {
      schemaId = 0,
      topic = "vov-projects",
    },
    taskers = {
      schemaId = 0,
      topic = "vov-metrics-taskers",
    },
    jobs = {
      schemaId = 0,
      topic = "vov-metrics-jobs",
    },
    scheduler = {
      schemaId = 0,
      topic = "vov-metrics-scheduler",
    },
  },
}
```

Configuration File Parameters

Service level configurable parameters in `sds.cfg`:

Parameter	Values	Default	Description
enabled	0 or 1	0	Disable/enable SDS on startup/readconfig
format	"json" "confluent"	"json"	Specifies the Kafka payload format to use, "confluent" = confluent Avro dialect with the schema registry id
site	string	""	User definable string to be delivered with the project record
group	string	""	User definable string to be delivered with the project record
debug	0 or 1	0	Disable/enable SDS debug logging on startup/readconfig

Change the Config File for the First Time

In order to use SDS for the first time, the user must perform the following operations:

1. Set the *`kafka_servers`* parameter in the `sds.cfg` file to the bootstrap server(s) for their kafka installation; for example, `kafka_servers = "kafkahost:9092"` or `kafka_servers = "kafkahost1:9092,kafkahost2:9092"`
2. If publishing using the Confluent Schema Registry, then the following steps are also needed:
   a) Upload the schema files to the schema registry and note the IDs assigned to each schema.
   b) Assign the schema registry IDs discovered in step 1 to the events in the `sds.cfg` file (see Event specific configuration below)

For the initial release the Kafka published events are:

Event Name	Description	Schema File	Default Topic
project	(relatively) Static project information that may be useful to join with time series data	vov-projects	
taskers	Metrics related to the state and capacity of the taskers	metrics.taskers.avsc	vov-metrics-taskers

Event Name	Description	Schema File	Default Topic
jobs	Metrics related to the number of jobs in specific states and rate of dispatch/completion	metrics.jobs.avsc	vov-metrics-jobs
scheduler	Metrics related to scheduler performance, sizes, clients, innerloop timers	metrics.scheduler.avsc	vov-metrics-scheduler

Change the Config File at Run Time

The SDS configuration may be changed while the server is running.

1. The SDS service may be enabled/disabled by using the command:

```
 $ vovservermgr config sds.enabled 1/0
```

2. Update the config file for the running server and/or publish a new project event with the following command:

```
$ vovservermgr config sds.readconfig 1
```

3. The debug setting may be enabled using:

```
$ vovservermgr config set_debug_flag SDS
$ vovservermgr config reset_debug_flag SDS
```

Troubleshooting

If the kafka_servers cfg parameter is not set correctly, the server log will contain entries like the following:

```
%3|1610382360.585|FAIL|rdkafka#producer-1| [thrd:foo:9092/bootstrap]: foo:9092/
bootstrap: Failed to resolve 'foo:9092': Temporary failure in name resolution (after
 1033ms in state CONNECT)
%3|1610382360.585|ERROR|rdkafka#producer-1| [thrd:foo:9092/bootstrap]: 1/1 brokers
 are down
%3|1610382363.544|FAIL|rdkafka#producer-1| [thrd:foo:9092/bootstrap]: foo:9092/
bootstrap: Failed to resolve 'foo:9092': Temporary failure in name resolution (after
 993ms in state CONNECT, 1 identical error(s) suppressed)
```

If the kafka servers are not running or reachable, the server log will contain entries like the following:

```
%3|1610383215.659|FAIL|rdkafka#producer-2| [thrd:hecto:9092/bootstrap]: hecto:9092/
bootstrap: Connect to ipv4#127.0.1.1:9092 failed: Connection refused (after 0ms in
 state CONNECT, 1 identical error(s) suppressed)
```

Start and Stop Accelerator

To start the Accelerator vovserver, login as user `rtdamgr` (or the user chosen as Accelerator administrator) on the machine that is designated as the vovserver, and the use the following command:

```
% ncmgr start
```

To stop the Accelerator vovserver, use the following command:

```
% ncmgr stop
```

Note: To stop the vovserver while there is an active workload (running jobs), use the option -freeze. When -freeze is used, all currently running jobs are preserved. If -freeze is not used, all vovtaskers are stopped along with vovserver, and any running job is also terminated. The -freeze option feature can be important when upgrading Accelerator.

```
% ncmgr stop -freeze
% ncmgr start -force
```

Reset Accelerator and Restart Taskers

Changes made to Accelerator's configuration files can be read in by performing a reset of the queue. The default behavior is to re-read all configuration files. There are also two optional reset types: soft and hard.

Soft Reset (-soft)	A soft reset instructs all running taskers to re-read minor configuration changes and starts configured taskers that are not running at the time. Running jobs are not disturbed.
Hard Reset (-hard)	A full reset stops and restarts all taskers. This is a forceful command and will kill all running jobs.

```
vncmgr: Usage Message


        DESCRIPTION:
        Utility to reread configuration files and optionally start/restart
        taskers.

        By default, "ncmgr reset" rereads and reprocesses the settings in the
        policy.tcl and security.tcl configuration files in the Server Working
        Directory.

        USAGE:
        % ncmgr reset [OPTIONS]

        OPTIONS:
        -help               -- Print this message
        -queue <name>       -- Specify the queue name. Default is $NC_QUEUE if
                               set, and otherwise vnc.

        -soft               -- Also starts any stopped, but configured taskers
        -taskers            -- Same as -soft

        -hard           *   -- Also stops and starts all taskers (jobs are lost)
        -full           *   -- Same as -hard

        * Warning: These options will forcefully stop all running jobs.

        EXAMPLES:

        % ncmgr reset
        % ncmgr reset -taskers
        % ncmgr reset -full

        % ncmgr reset -soft
        % ncmgr reset -hard
```

```
% ncmgr reset                  - Reread configuration files
% ncmgr reset -queue vnc2      - Reread configuration files for vnc2 queue
% ncmgr reset -soft            - Reread configuration files and start taskers that
 are not running
% ncmgr reset -hard            - Reread configuration files and restart all taskers
```

For an example of using the full reset option, refer to Directories and Files.

Start Accelerator at System Boot Time

The instructions in this section are valid for Linux only. This part of the installation requires root permission.

Note: This step is optional.

The Accelerator vovserver can be restarted at reboot by installing the proper script in both the `/etc/rc3.d` and `/etc/rc5.d` directories.

Run the following commands on the host that was selected as the Accelerator vovserver.

```
% /bin/su
% cp $VOVDIR/etc/boot/S99nc /etc/rc2.d/S99nc
% chmod 755 /etc/rc2.d/S99nc
% vi /etc/rc2.d/S99nc
....
Edit configurable items as needed.
```

Note: `sudo` should be used where configured. To avoid Trojan Horse programs, `su` should always be called by full path `/bin/su`.

```
% ./S99nc start
```

Messages about the vovserver starting should be displayed. An example is shown below:

```
% ./S99nc status
```

The output of `nc info` should be displayed. An example is shown below:

```
% ./S99nc stop
```

Note: Re-start the Accelerator server with the command `S99nc start` after testing the `stop` capability.

Job Management

Job Status

In Accelerator, each job goes through a number of states until completion.

The states are described in the following table:

Status	Color	Description
Idle	BlueViolet	If the node is a job, either it has not been run successfully yet or it needs to be run again, because one of its inputs has been modified since the last time the job was executed. If the node is a file, it is the output of a job that is not Idle.
Queued	Light blue	The job is scheduled to be run. It may be already queued or it will go in the queue as soon as all its inputs are ready.
Running	Orange	The job is currently being retraced; it has been dispatched to one of the taskers. All the outputs of such a job are either RETRACING or RUNNING.
Done	Green	If the node is a job, it has run successfully. If the node is a file, it is up-to-date with respect to all other files and jobs on which it depends.
Failed	Red	The job ran and failed.
Transfer	Cream	The job is being transferred to another cluster and it is not yet running.
Suspended	Pink	The job was running (or retracing) and one of the processes belonging to the job is currently suspended.
Sleeping	Black	Either the job caused an output conflict upon submission (bad dependencies) or the job was not reclaimed by any tasker upon crash recovery.
Withdrawn	Gray	A job has been withdrawn after dispatching, such as by the preemption daemon. **Note:** This status occurs rarely and tends to be hard to observe.

The normal sequence for a successful job is **Idle** > **Queued** > **Running** > **Done**

The normal sequence for a failing job is **Idle** > **Queued** > **Running** > **Failed**

Job Persistence

An important feature of Accelerator is *job persistence*. After a job completes, its information remains in vovserver's memory until the job is forgotten, manually or automatically.

Jobs are automatically forgotten to limit vovserver memory consumption.

Note: The Accelerator default is to automatically forget jobs after a configurable time interval, as described in Automatic Forgetting.

You may override automatic forgetting by submitting jobs using the -keep or -keepfor options.

The benefits of job persistence:

- You can re-execute a job using `nc rerun jobID` without the need to type the job command line again.
- Duration information can be used to execute the job on the appropriate taskers (enough time left)
- Commands are easily edited in the GUI or the browser UI
- Preserves the job info for documentation and auditing
- Jobs in vovserver memory have full information accessible via the VTK API

To enable persistence, use the -keep option when you submit the job, as shown below.

Important: Jobs submitted with this option remain in vovserver memory until they are explicitly forgotten, so there is a tradeoff vs. memory usage, and jobs should only be kept when needed.

Keep Jobs for Longer than the Default

If your Accelerator administrator has arranged to support it, you may also use the -keepfor option when submitting jobs. This option takes a VOV timespec, e.g. 4h, 14d. Jobs will be automatically removed from the system when older than the specified age.

This is supported by a liveness script which examines the jobs in memory periodically. The example script may be found in `$VOVDIR/etc/liveness/live_keepfor_jobs.tcl` and should be copied into the Accelerator vovserver's 'tasks' subdirectory.

The scripts in the tasks subdirectory are triggered every vovserver update cycle, about once a minute. The -keepfor script uses a property KEEPFOR_LASTRUN_TS to record the last time it was active, and an optional KEEPFOR_FREQUENCY to determine how frequently to clean up kept jobs. The default is 1800s, or 30m. These properties are on the object having ID 1.

When you submit a job using the -keepfor option, Accelerator attaches a KEEPFOR property to the job to record how long the job should be kept. This is silently limited to the range 0..32000000 seconds (just over 1 year).

Example commands:

```
% nc run -r unix -t BASE -keep runregression daily
```

```
% nc run -r unix -e BASE -keepfor 2w runregression quick
```

CPU Effort Considerations

The script that implements -keepfor needs to make a scan through all jobs in the system that are in the Done or Failed state, so this takes some CPU time, and this time will increase with the total number of jobs in the system. You should generally not need to reduce the scan interval below the default of 30m, and in most cases, you can make it longer, perhaps to 8-24h depending on the rate of kept job creation in your system.

Forget Jobs from vovserver Memory

You can forget jobs from vovserver's memory using the following command. Common values for <job-spec> include a jobID, '-mine' and '-set' with a set name.

For more usage information, see `nc forget`.

```
% nc forget <job-spec>
```

Automatic Forgetting

Accelerator's default is to automatically forget jobs as listed below:

- Successful (Done) jobs are forgotten after 1 hour
- Failed jobs are forgotten after 2 days
- Unscheduled (Idle) jobs are forgotten after 2 days

To control the time jobs are kept in the system, edit the VovServer configuration in the `policy.tcl` file. The parameters controlling this are *`autoForgetValid`*, *`autoForgetFailed`*, and *`autoForgetOthers`*.

For more details, refer to Autoforget Jobs.

Autoforget Jobs

The autoforget flag sets up a job to automatically be forgotten by the system after a certain time, (not including suspension time) if and only if the job is done, failed, or idle. Jobs that are scheduled, running, suspended or transfer are never autoforgotten.

Global auto-forget Parameters

There are three different auto-forget parameters:

- *`autoForgetValid`*
- *`autoForgetFailed`*
- *`autoForgetOthers`*

In Accelerator, the autoforget flag is set by default, which can be unset by using the option -keep in `nc run`. In Flow Design Language (FDL), the variable *`make(autoforget)`* controls the flag.

- The autoforget flag on the job is true

- The job is done, failed, or idle.

Jobclass Specific auto-forget

- The job belongs to a jobclass with the `AUTOFORGET` property set to a positive value.
- The job is done, failed, or idle.

Note: The autoforget flag on the job is irrelevant.

If a jobclass has a specific auto-forget property, then the jobs in that jobclass will be forgotten after that specified time.

For example, to set the autoforget property on a jobclass called *abc*, use the `vtk_jobclass_set_autoforget` API:

```
% nc cmd vovsh -x "vtk_jobclass_set_autoforget abc 2m"
```

To disable this functionality for a jobclass, set the value of autoforget to a non-positive value, such as:

```
% nc cmd vovsh -x "vtk_jobclass_set_autoforget abc 0"
```

Auto-forget Log Files

If the parameter *`autoForgetRemoveLogs`* is true and the parameter *`disablefileaccess`* is false, the vovserver tries to delete the log file of the jobs that are being auto-forgotten. The success of the deletion depends on the file permissions.

Note: Accessing files makes the vovserver vulnerable to NFS problems.

Auto-forget Examples

For this example, the default autoforget policy is to forget jobs after 1h. Other jobs in the jobclass "Regression" should be retained for 10days. Submit the jobs with the -keep option (no autoforget flag) and then set the AUTOFORGET property in the set `Class:Regression` to 864000.

```
### Done by an ADMIN
% nc cmd vovsh -x 'vtk_jobclass_set_autoforget Regression 10d'

% nc run -C Regression -keep  ./my_test
```

Conversely, if the retention policy keeps the jobs for a long time (such as 3 days), some jobs in the jobclass "Quick" may be set to be forgotten more promptly, (such as after 5m) of completion. In this case, set the AUTOFORGET property in the jobclass set as follows:

```
### Done by an ADMIN
% nc cmd vovsh -x 'vtk_jobclass_set_autoforget Quick 5m'
```

Schedule Jobs

The scheduling process controls the order of job execution.

The scheduler in Accelerator is event-driven. When an event occurs that can cause a job to be placed onto a tasker, the scheduler is called. The types of events includes:

- job submission
- job termination
- increased availability in a resource map
- expiration of a reservation
- and many others

The jobs that are scheduled to be run, the queued jobs, are organized into *buckets*. Each bucket contains jobs that have identical scheduling parameters.

The buckets are assigned a rank based on the FairShare statistics. Starting from the buckets of rank 0 (zero), the scheduler attempts to dispatch the top job in the bucket to the best available tasker. Then the scheduler looks at buckets with rank 1 and so on.

Exceptions to FairShare Scheduling

There are a few ways to bypass the FairShare scheduling: manual dispatch of jobs or Job Cohorts.

Seamless Transition to a Cycle-based Scheduler

The scheduler typically executes in a few milliseconds. The scheduler effort required in each cycle increases with the number of buckets, the number of taskers, and the complexity of the resource expressions. It is always possible to overload the scheduler, meaning that the scheduler requires a large fraction of the total CPU time used by vovserver.

The vovserver parameter to control scheduler cycle behavior is called *`schedSkip`*, and is expressed in seconds. If the effort required to run the scheduler exceeds the *`schedSkip`* threshold, action is taken to reduce the duty cycle allocated to scheduling. To reduce the duty cycle, the vovserver starts skipping scheduler calls, which frees up computing power to be used to service other requests such as listings and job terminations. The result is that the scheduler functionality is effectively bypassed, regardless of bucket priority, until the target "scheduler duty cycle" percentage is attained.The ratio of the computing power allocated to scheduling, which we call the "scheduler duty cycle", is controlled by the parameter *`schedMaxEffort`*, an integer that represents the percentage of time we want to allocate to the scheduler.

The default value of *`schedSkip`* is 0.1 seconds, while the default duty cycle reserved for scheduling is set to 20%, represented by *`schedMaxEffort`* value of 20.

Other Parameters that Control the Scheduler

There are a few more parameters that control the scheduler behavior and could impact performance on large workloads.

Note: There is a built-in dynamic server tuning feature for the maximum jobs dispatched per queue bucket when server is under heavy load conditions.

`sched.maxpostponedjobs`

Controls the exit from the scheduler when it is hard to dispatch jobs to taskers, i.e. when the scheduler has to postpone many jobs because it cannot find suitable taskers for them. Normally this parameter is set to be much larger than the maximum number of buckets in the system. Our default value is 10,000; this value can be decreased if you have very homogeneous farms where all taskers are identical.

> **Note:** This parameter will disappear in future implementations of the scheduler.

`fairshare.maxjobsperbucket`

Controls how many jobs are allowed to be dispatched from the same bucket. The default value for this is 20 jobs. After 20 jobs have been dispatched from a bucket, the scheduler moves to the next bucket. Smaller values give a more accurate FairShare accounting. Larger values give a faster dispatch.

> **Note:** This parameter will disappear in future implementations of the scheduler.

`fairshare.maxjobsperloop`

Controls how many jobs can be dispatched in a single scheduler loop (this includes the scanning of possibly all the buckets). The default value is 20 jobs per loop (note: this could mean that all 20 jobs are from the same bucket). Smaller values give a more accurate FairShare accounting. Larger values give a faster dispatch.

> **Note:** This parameter will disappear in future implementations of the scheduler.

To control the scheduler, an admin can use the command line or the `policy.tcl` file. For example, to increase the threshold to morph into a cycle-based schedulers from the default 0.1 to 0.3 seconds and to increase the duty cycle from 20% to 40%, the following command line could be used:

```
% vovsh -x "vtk_server_config schedSkip 0.3"
% vovsh -x "vtk_server_config schedMaxEffort 40"
% vovsh -x "vtk_server_config sched.maxpostponedjobs 10000"
% vovsh -x "vtk_server_config fairshare.maxjobsperloop 20"
% vovsh -x "vtk_server_config fairshare.maxjobsperbucket 20"
```

Alternatively, the `policy.tcl` file can be modified:

```
# This is a fragment of policy.tcl
set config(schedSkip) 0.3
set config(schedMaxEffort) 40
set config(sched.maxpostponedjobs) 10000
set config(fairshare.maxjobsperloop) 20
set config(fairshare.maxjobsperbucket) 40
```

Job Cohorts

> **Note:** This is a new experimental concept in the 2017 release.

A *job cohort* is a collection of jobs that require special FIFO scheduling. These jobs are always given a FairShare rank of 0 and are therefore scheduled "as soon as possible".

These type of jobs typically occurs in conjunction with large distributed parallel jobs. If a wide DP job has been partially dispatched, it is important that the rest of the partial jobs get dispatched right away, else we waste a lot of time waiting for the rendezvous. In the current implementation of partialTool if at least 25% of the DP job has been dispatched, then all the remaining DP job components become a cohort and will be dispatched "as soon as possible," therefore bypassing other scheduling rules.

Another application of job cohorts is to schedule sets of jobs that are relatively short, but only have value when all of them have been executed, for example a "smoke regression test", those short regressions that many organizations require of their engineers before a change to the input data can be checked into the repository. If the smoke test begins and is, say, 50% dispatched, then it becomes very valuable to make sure that the other jobs in the smoke test also get executed, bypassing the FairShare rules.

It is possible to abuse this concept and say that all my jobs are "cohort" jobs. Abuses will be detected and warnings will be issued.

How to Turn On/Off Cohorts

There is only one low level method to activate cohorts, via the **`vtk_set_cohort`** API.

```
vtk_set_cohort $setId  1 ;#   Set the cohort flag on all jobs contained in set.
vtk_set_cohort $setId  0 ;#   Reset the cohort flag on all jobs contained in set.
```

Database

Accelerator stores historical information about jobs in a relational database. As of version 2015.09, the database is fully integrated and managed as part of Accelerator. This section provides an overview of the components that run and manage the database.

The database, vovdb, is based on PostgreSQL™, a performant and reliable open-source database engine with decades of wide-scale use. There are two ways to configure and manage vovdb: through the web UI and via the command line.

It should never be necessary to run `vovdb` directly; once configured and enabled, Accelerator will automatically start and stop the database as needed.

Daemon

Certain database tasks are managed via the `vovdbd` daemon: a background process that can load data, perform maintenance, or trigger backups. These tasks can be configured through the database administration web UI.

Refer to the *Daemons* documentation for more information on how Accelerator manages daemons.

Tasker

Note: The information in this section does not apply to Windows-based installations.

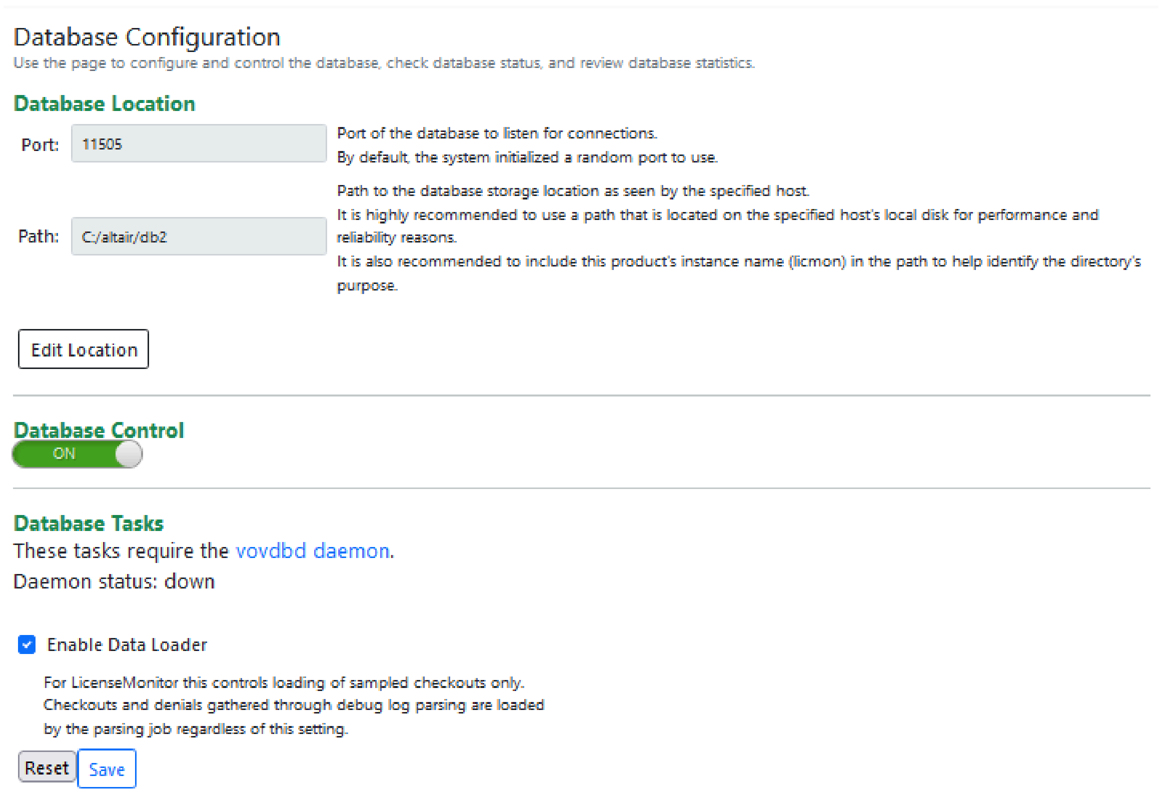
A vovdbd tasker will be created as necessary to manage the database. This occurs when certain database commands (starting, stopping, and performing backups) are run from a host different than the configured database host.

This typically occurs when the database is configured with a remote host, but may also occur with a local database if command line operations are run from another machine. In either case, the command is automatically spawned to the vovdbd tasker as an interactive job; the command behaves identically whether run locally or remotely.

The vovdbd tasker does not consume a license, and is reserved for database-related jobs only.

Set Up

For the first start of Accelerator, click **Admin** > **System** > **Database** to configure the database. You must have ADMIN privileges on the Accelerator server to configure and control the database.


Database Configuration
Use the page to configure and control the database, check database status, and review database statistics.

Database Location

Port: 11505
Port of the database to listen for connections.
By default, the system initialized a random port to use.

Path: C:/altair/db2
Path to the database storage location as seen by the specified host.
It is highly recommended to use a path that is located on the specified host's local disk for performance and reliability reasons.
It is also recommended to include this product's instance name (licmon) in the path to help identify the directory's purpose.

Database Control

Database Tasks
These tasks require the vovdbd daemon.
Daemon status: down

[✓] Enable Data Loader

For LicenseMonitor this controls loading of sampled checkouts only.
Checkouts and denials gathered through debug log parsing are loaded by the parsing job regardless of this setting.

Figure 4: System Database Configuration

Database Location

In this section, specify the host (if a separate database host is desired), port (if a specific port is desired), and the path to the database directory as seen by the specified database host. The configuration page will default to using the local host, a port randomly selected, and a path that is located in the installation area.

If configuring a separate database host, it is assumed that an SSH connection can be made between the Accelerator server host and the remote database host without a password prompt appearing. Consult your IT organization to set up an SSH key environment if needed.

Note:

- Separate database hosts are not supported for Windows-based installations.
- It is highly recommended to utilize the local file system for the database directory. While remote file systems will work, the resulting performance will be far less than with the local file system.

The database host and path can also be configured from the command line. See *Configure the Database from the Command Line* for more information.

Database Control

After the database host and location have been specified, drag the slider switch to the ON position to create and start the database. The database will now be started automatically upon each start of Accelerator.

To stop the database, drag the slider switch to the OFF position. When the database is not running, data will continue to be collected by Accelerator but it will not be loaded until the database is restarted.

The database can also be started and stopped from the command line. See *Configure the Database from the Command Line* for more information.

Database Tasks

This section allows configuration of the database tasks. These tasks cannot be configured from the command line.

The tasks are run by the `vovdbd` daemon. Status is shown and updated every 30 seconds. For more information, refer to *Altair Accelerator Daemons*.

The Data Loader checkbox controls the automatic loading of jobs. The loader runs continuously, but is automatically disabled when doing maintenance or backup.

The Maintenance settings enable daily database maintenance and specify the time window to perform maintenance. The database is still available during the maintenance window; however performance may be impacted. For this reason, it is best to schedule maintenance for off-peak hours.

Database Backups enables automated backups. The frequency of the backups, backup location, time window in which to start the backup, and how many backups to keep can be specified. As with maintenance, the database is still available while a backup is being made, however performance will be impacted.

Note: Network storage is acceptable for database backups.

Configure the Database from the Command Line

The command line utility `vovdb_util` can be used to configure most aspects of the Accelerator database. You must have ADMIN privileges on the Accelerator server to configure and control the database.

To execute the utility, first set up the CLI:

```
% vovproject enable vnc
```

vovdb_util

Utilities for use with the VOV database.

```
vovdb_util: Usage Message
  DESCRIPTION:
    Utilities for use with the VOV database.

  USAGE:
    % vovdb_util <COMMAND> [COMMON OPTIONS] [OPTIONS]

  COMMANDS:
    help         -- Shows help information.
    backup <dest>
                 -- Backs up the VOV database to the specified destination.
                    This command must be run on the same machine as the
                    database.  If the destination exists it must be empty.
                    The database must also be running.
    clearcfg [-noconfirm]
                 -- Resets the VOV database configuration to the initial,
                    unconfigured state.  Pass -noconfirm to skip confirmation.
    exportconfig <fileout>
                 -- Exports VOV database configuration
                    properties DB_* to obfuscated file.
    exportpasswords <fileout>
                 -- Exports VOV database passwords
                    from project to obfuscated file.
    importconfig <filein>
                 -- Imports VOV database configuration
                    properties DB_* from obfuscated file.
    importpasswords <filein>
                 -- Imports VOV database passwords
                    from obfuscated file to project.
    configure [-v] [-reset] [-noconfirm] <host> <root> [<port>]
                 -- Sets the host, root data path, and port for the VOV
                    database. Pass -reset to overwrite existing settings.
                    Pass -noconfirm to skip confirmation.
    showcfg      -- Prints out current VOV database configuration.
    startdb      -- Starts the VOV database.  Will restart a running database.
    status       -- Prints current VOV database status.
    stopdb       -- Stops the VOV database.
    upgrade [-noconfirm] [-sdb sourcedir] [-spgsw pg_software_dir]
                 -- Upgrades the VOV database to use the newest version of the
                    PostgreSQL engine.
                    Pass -noconfirm to skip confirmation.
                    -sdb to specify path to Source database
                    -spgsw path to older version of PG
                     binaries compatible with Source database

      The following commands are only supported for NetworkComputer and
      LicenseMonitor.

    dump [-pre201509] [-start <YYYYMMDD>] [-end <YYYYMMDD>]
                 -- Generate data files, optionally limited to the start and
                    end times specified.  Pass -pre201509 to dump the database
                    that was used prior to upgrade to 2015.09 (or beyond.)

    trim <YYYYMMDD>
                 -- Deletes data prior to the given date.

  COMMON OPTIONS:
    -v           -- Increase verbosity.
```

Database Configuration Options

Database configuration is handled by the `vovdb_util configure` command. Pass -reset to overwrite an existing configuration. Pass -noconfirm to skip confirmation.

To show the current configuration, use `vovdb_util showcfg`. To clear an existing configuration, use `vovdb_util clearcfg`.

Some examples:

```
% vovdb_util configure localhost /data/rtda/licmon
vovdb_util 08/17/2023 16:15:03: message: Setting VOV database configuration to:
 Host:      srv1
 Data Path: /data/rtda/licmon
Set configuration (yes/no)? yes
vovdb_util 08/17/2023 16:15:08: message: Configuration saved.

% vovdb_util configure -reset -noconfirm localhost /data/rtda/db/licmon
vovdb_util 08/17/2023 16:15:53: message: Configuration saved.

% vovdb_util showcfg
vovdb_util 08/17/2023 16:16:00: message: The VOV database configuration is:
 Host:      srv1
 Data Path: /data/rtda/db/licmon
 Status:    stopped

% vovdb_util clearcfg -noconfirm
vovdb_util 08/17/2023 16:15:24: message: Configuration has been cleared.
```

Database Control Options

Once the database is configured, it can be controlled with the `vovdb_util startdb` and `stopdb` commands. There will be additional output the first time the database is started as the on-disk structure is created.

Starting the database:

```
% vovdb_util startdb
vovdb 08/17/2023 16:22:55: message: Creating database
The files belonging to this database system will be owned by user "admin".
This user must also own the server process.

The database cluster will be initialized with locale "en_US.UTF8".
The default database encoding has accordingly been set to "UTF8".
The default text search configuration will be set to "english".

Data page checksums are disabled.

creating directory /data/rtda/db/licmon/dbdata9_4 ... ok
creating subdirectories ... ok
<snip>
vovdb 08/17/2023 16:22:58: message: Starting database
vovdb 08/17/2023 16:22:58: message: LOG:  database system was shut down at 2023-08-17
 16:22:56 CDT
vovdb 08/17/2023 16:22:58: message: LOG:  MultiXact member wraparound protections are
 now enabled
vovdb 08/17/2023 16:22:58: message: LOG:  database system is ready to accept
 connections
vovdb 08/17/2023 16:22:58: message: Database engine is ready.
vovdb 08/17/2023 16:22:58: message: Configuring database...
vovdb 08/17/2023 16:22:58: message: Creating user 'rtdamgr'
vovdb 08/17/2023 16:22:58: message: Creating user 'rtdausr'
vovdb 08/17/2023 16:22:58: message: Creating database 'rtda'
vovdb 08/17/2023 16:22:58: message: Database configured.
vovdb 08/17/2023 16:22:58: message: Loading schema for LicenseMonitor...
vovdb 08/17/2023 16:22:58: message: Creating table 'metadata'...
vovdb 08/17/2023 16:22:58: message: Granting RO privileges to rtdausr...
vovdb 08/17/2023 16:22:58: message: Creating table 'loadinfo'...
vovdb 08/17/2023 16:22:58: message: Granting RO privileges to rtdausr...
<snip>
vovdb 08/17/2023 16:22:59: message: Schema loaded
vovdb 08/17/2023 16:22:59: message: Database is ready.
```

```
vovdb_util 08/17/2023 16:22:59: message: Database started.
```

Stopping the database:

```
% vovdb_util stopdb
vovdb_util 08/17/2023 16:31:22: message: Stopping database...
vovdb_util 08/17/2023 16:31:23: message: Database stopped.
```

Database Tasks

At present, database tasks such as the automatic loader, daily maintenance, and automatic backups can only be configured through the web interface. Refer to *Set Up* for more information.

Database Engine Versions and Upgrades

Accelerator includes version 14.4 of the PostgreSQL database engine that is used by the VOV database daemon `vovdb`. This section describes how to determine if a database upgrade will be needed when upgrading the Accelerator product software.

Database Engine Versions

PostgreSQL 14.4 is an improved and updated database version. An existing database created with PostgreSQL 9.4 or 9.6 cannot be used directly in 14.4; it must be upgraded. The upgrade process is described in Database Upgrades. When you plan to upgrade an existing Accelerator project from a Accelerator software version with PostgreSQL 9.4 or 9.5 to a Accelerator version with PostgreSQL 14.4, you will need to choose a database upgrade strategy from the options described in the *Altair Accelerator Software Installation Guide*.

To determine the running Accelerator project's PostgreSQL version, visit the **System** > **Database Configuration** web UI page. If the PostgreSQL version is 14.4, then you may simply upgrade Accelerator via the Software Upgrade Instructions section in the manual.

Load Data

This section details the methods that Accelerator uses to load data into the database.

Automatic Loading

Checkout data files generated from sampled data will be automatically and continuously loaded if the daemon-based loader is enabled.

Job data files generated by the server are automatically and continuously loaded when the daemon-based loader is enabled. Refer to *Set Up* for more information.

Manual Loading

Manual loading of job data files can be accomplished by using the `vovsql_load_checkouts` and `vovsql_load_denials` command line utility.

To execute the utility, set up the CLI:

```
% vovproject enable vnc
```

The checkout loader's syntax is:

```
vovsql_load_checkouts: Usage Message

  USAGE:
      % vovsql_load_checkouts [OPTIONS] <listOfFiles>

  OPTIONS:
      -h                -- This help.
      -v                -- Increase verbosity.
      -q                -- Quiet.
      -origin <N>       -- Specify origin of data (see below)
      -f                -- Same as -force.
      -force            -- Force reloading of data files.
      -convert          -- Convert the old data files to current format.

  REQUIRED ARGUMENTS:
      <listOfFiles>     -- A list of checkout data files. Can also be specified
                           as a glob expression (e.g. checkouts/2009.12.*)
                           or can contain the tokens @TODAY@ and @YESTERDAY@.

  ORIGIN:
      1                 -- The data comes from the sampling (default).
      2                 -- The data comes from debug log parsing, to be merged.
      6                 -- The data comes from debug log parsing,
                           not to be merged.

  EXAMPLES:
      % vovsql_load_checkouts           licmon.swd/data/checkouts/2007*
      % vovsql_load_checkouts -force    licmon.swd/data/checkouts/2007*
      % vovsql_load_checkouts -origin 2
          licmon.swd/data/checkouts/MGC/mgcld/master/2010*
      % vovsql_load_checkouts           licmon.swd/data/checkouts/@TODAY@
```

```
  vovsql_load_checkouts: Usage Message

  USAGE:
      % vovsql_load_checkouts [OPTIONS] <listOfFiles>

  OPTIONS:
      -h                -- This help.
      -v                -- Increase verbosity.
      -q                -- Quiet.
      -origin <N>       -- Specify origin of data (see below)
      -f                -- Same as -force.
      -force            -- Force reloading of data files.
      -convert          -- Convert the old data files to current format.

  REQUIRED ARGUMENTS:
      <listOfFiles>     -- A list of checkout data files. Can also be specified
                           as a glob expression (e.g. checkouts/2009.12.*)
                           or can contain the tokens @TODAY@ and @YESTERDAY@.

  ORIGIN:
      1                 -- The data comes from the sampling (default).
      2                 -- The data comes from debug log parsing, to be merged.
      6                 -- The data comes from debug log parsing,
                           not to be merged.

  EXAMPLES:
      % vovsql_load_checkouts           licmon.swd/data/checkouts/2007*
      % vovsql_load_checkouts -force    licmon.swd/data/checkouts/2007*
      % vovsql_load_checkouts -origin 2
          licmon.swd/data/checkouts/MGC/mgcld/master/2010*
      % vovsql_load_checkouts           licmon.swd/data/checkouts/@TODAY@
```

Export Data

One function of the utility `vovdb_util` is to export the database into jobs data files. The exported files are saved in the `vnc.swd/data/dump` directory.

To execute the utility, first set up the CLI:

```
% vovproject enable vnc
```

vovdb_util

```

vovdb_util: Usage Message
  DESCRIPTION:
    Utilities for use with the VOV database.

  USAGE:
    % vovdb_util <COMMAND> [COMMON OPTIONS] [OPTIONS]

  COMMANDS:
    help         -- Shows help information.
    backup <dest>
                 -- Backs up the VOV database to the specified destination.
                    This command must be run on the same machine as the
                    database.  If the destination exists it must be empty.
                    The database must also be running.
    clearcfg [-noconfirm]
                 -- Resets the VOV database configuration to the initial,
                    unconfigured state.  Pass -noconfirm to skip confirmation.
    exportconfig <fileout>
                 -- Exports VOV database configuration
                    properties DB_* to obfuscated file.
    exportpasswords <fileout>
                 -- Exports VOV database passwords
                    from project to obfuscated file.
    importconfig <filein>
                 -- Imports VOV database configuration
                    properties DB_* from obfuscated file.
    importpasswords <filein>
                 -- Imports VOV database passwords
                    from obfuscated file to project.
    configure [-v] [-reset] [-noconfirm] <host> <root> [<port>]
                 -- Sets the host, root data path, and port for the VOV
                    database. Pass -reset to overwrite existing settings.
                    Pass -noconfirm to skip confirmation.
    showcfg      -- Prints out current VOV database configuration.
    startdb      -- Starts the VOV database.  Will restart a running database.
    status       -- Prints current VOV database status.
    stopdb       -- Stops the VOV database.
    upgrade [-noconfirm] [-sdb sourcedir] [-spgsw pg_software_dir]
                 -- Upgrades the VOV database to use the newest version of the
                    PostgreSQL engine.
```

```
                     Pass -noconfirm to skip confirmation.
                     -sdb to specify path to Source database
                     -spgsw path to older version of PG
                      binaries compatible with Source database

       The following commands are only supported for NetworkComputer and
       LicenseMonitor.

     dump [-pre201509] [-start <YYYYMMDD>] [-end <YYYYMMDD>]
                  -- Generate data files, optionally limited to the start and
                     end times specified.  Pass -pre201509 to dump the database
                     that was used prior to upgrade to 2015.09 (or beyond.)

     trim <YYYYMMDD>
                  -- Deletes data prior to the given date.

   COMMON OPTIONS:
     -v           -- Increase verbosity.
```

Database Export

Database export is done using the `vovdb_util dump` command. Pass the optional -start YYYYMMDD and -stop YYYYMMDD to restrict the dump to after the specified start date and before the specified end date.

Dump Pre-2015.09 Databases

If updating Accelerator from a version prior to 2015.09, it is possible to pass the -pre201509 option to `vovdb_util dump` to generate data files from the old database. This can be used to save old data if the original data files are no longer present.

Exported Jobs Directory

The `vnc.swd/data/dump/jobs` directory contains the exported jobs data. The dumped data files can be loaded into a new database using the `vovsql_load_jobs` utility.

Database Backup

There are two ways to do direct backups of the database: automatic and manual. In addition, the original source data files may be saved.

Automatic Backups

Automatic backups are configured through the administration web UI. When enabled, the backups are generated in subdirectories of the configured Backup Location named for the time of backup.

If the backup location is configured as:

```
/data/rtda/db_backup/vnc
```

The backup started on August 21st, 2023 at 1:00AM will be located in:

```
/data/rtda/db_backup/vnc/20230821_010000
```

Manual Backups

Manual backups are generated by running the `vovdb_util` utility. Backups are generated directly in the directory specified; it is recommended to specify the time of backup in the directory name for future reference. Example:

```
% vovdb_util backup /data/backup/vnc/2023Aug22_10AM
```

Source Data Backups

In addition to backing up the database directly, the source data files used to populate the database can be saved. The source files are located in vnc`.swd/data` and can be used to rebuild a database from scratch. No utility is provided to save these files.

Restore a Database from Backups

The procedure to restore from backup is the same whether the backup was generated automatically or manually.

1. Ensure the current database is stopped, either through the web UI or via the command line.
2. Delete or move the existing database.
   For example, if the database is located at `/data/rtda/db/`vnc, it can be moved with:

   ```
    % mv /data/rtda/db/vnc /data/rtda/db/vnc.bad
   ```
3. Copy or move the backup to the original database location.

   ```
   # Using the example of an automatic backup from above.
   % mv /data/rtda/db_backup/vnc/20160821_010000 /data/rtda/db/vnc

   # Using the example of a manual backup from above.
   % mv /data/backup/vnc/2016Aug22_10AM /data/rtda/db/vnc
   ```
4. Restart the database.

Rebuild a Database from Source Data Files

1. Ensure the current database is stopped, either through the web UI or via the command line.
2. Delete or move the existing database.
   For example, if the database is located at `/data/rtda/db/`vnc, it can be moved with:

   ```
   % mv /data/rtda/db/vnc /data/rtda/db/vnc.bad
   ```
3. Reload the source data files.

   ```
   % vovproject enable vnc
   % cd `vovserverdir -p data/jobs`
   % vovsql_load_jobs *
   ```

Track Job Commands

Each job has a field called `tool`, which is stored in the database and is used for reporting. This field is automatically computed to be the tail of the first command line argument that is not a known wrapper.

For example, if the command line is `vw /bin/cp aa bb`, the `tool` field has the value `cp` because `vw` is a wrapper and `/bin/cp` is the first argument that is not a wrapper.

Problem: The Number of Tools Explodes

Scripts may be generated automatically with unique names assigned to each script, containing elements such as timestamps or random seeds. This method affects the meaning of `tool`. It appears that each tool is used only once and there is a very large number of tools, which leads to excessively slow reporting times. In this scenario, the value of the field `tool` needs to be controlled.

Realizing a problem with `tool` often emerges late in the deployment of Accelerator. A solution for this problem has two components:

- A post-processing script to modify the `tool` field in the SQL database. This script is called `vovsql_normalize_field`.
- A facility to control the value of `tool` at job creation time using FDL. This only requires setting the variable *`make(tool)`* before the job is created.

The vovsql_normalize_field Utility

The utility `vovsql_normalize_field` changes the value of the `toolid` field in the jobs table in SQL. The recommendation is to use the utility is to create a configuration file called `db_rename.tcl` in the directory `vnc.swd/db` and then call the utility as follows:

```
% vovsql_normalize_field -file vnc.swd/db/normalize.tcl
```

This can also be done automatically with a liveness script (XXXXX TO BE DONE).

The file `db_rename.tcl` contains calls to a procedure called `SQL_NC_FIX_TOOL` which takes 2 parameters:

- A regular expression to select the bad tool names. This expression is in SQL syntax; is must include the character '%' to match any substring.
- The new value of the tool name. Using a simple name is recommended.

```
Example of *.swd/db/normalize.tcl file.
SQL_NORMALIZE_TOOL vrun%               VRUN
SQL_NORMALIZE_TOOL gen_script%seed%  SIM
```

Control the Tool Field at Job Creation Time

In Accelerator, control the `tool` field with the option -tool in `nc run`. In a job class definition, the value of the variable *VOV_JOB_DESC(tool)* can be set.

```
% nc run -tool SIM ./sim/sim_seed_123487897_ts_12122212_aa_to_bb
```

The `tool` field is also important from a scheduling point of view because the tool field can be associated with a resource map. In the following example, if there is a resource map called `Tool:SIM`, it will be honored for all jobs that have a tool value equal to `SIM`.

```
# In a jobclass definition file
...
set VOV_JOB_DESC(tool) "SIM"
...
```

Plot Jobs

The script `jobplots.cgi` can generate plots of jobs over a period of time. The jobs are organized by project, user, host, or jobclass.

Following is an example of a plot by project:

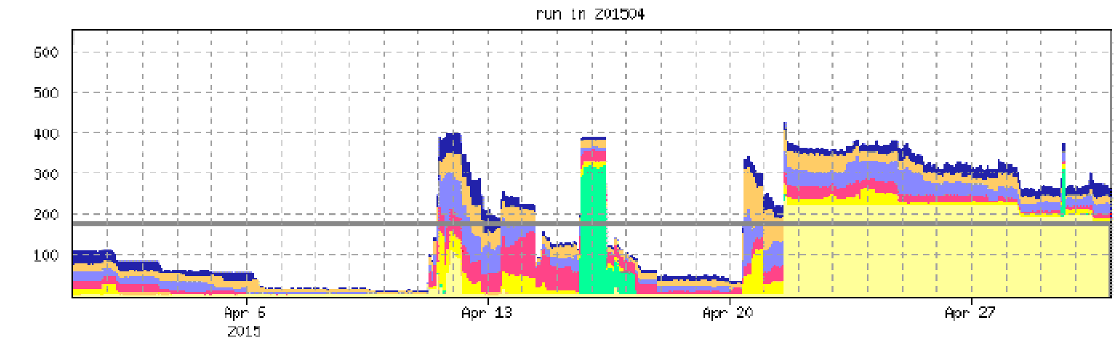


Figure 5: Example: running jobs

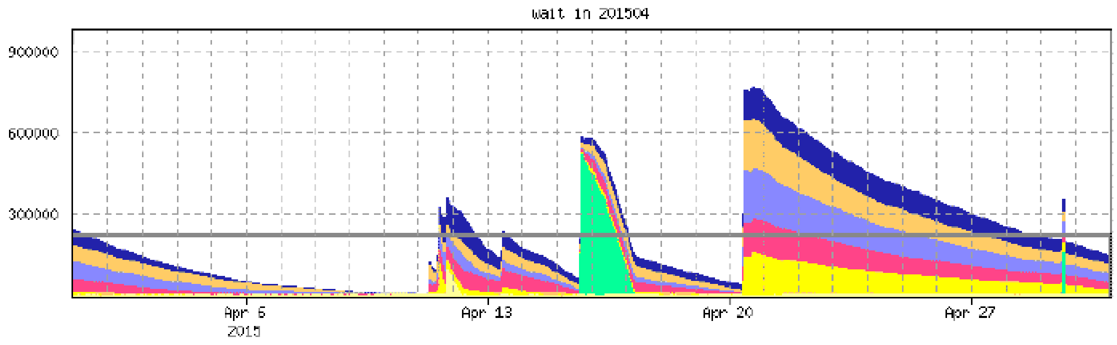


Figure 6: Example: waiting jobs

Customize Plot Colors

The script `jobplots.cgi` uses a small palette of 12 colors. Depending on what is shown, it is possible that the same entity is plotted in a different color. The file `plot.config.tcl` in the SWD (e.g. `vnc.swd/plot.config.tcl`) can be used to specify the color to be used for a specific project, host, user, jobclass, etc.

```
# Example of plot.config.tcl   used by   jobplots.cgi
# Use symbolic names for colors (see rgb.txt)
JobPlotColor projects arm3  pink
JobPlotColor projects rout5 yellow

# Use hex value for colors (no # and no 0x please)
JobPlotColor jobclasses hsim 33FF88
JobPlotColor jobclasses dc   EE44AD
```

Generate Custom Reports

The Accelerator database is open for custom queries using SQL.

Database Structure

The main table in the database is called `jobs`. As the table is expected to grow very large (billions of records), it is composed of integer fields only, which are used as references into auxiliary tables. Refer to Database Schema for more information on the database structure.

Writing Custom Queries

Arbitrary reports can be generated using `vovsql_query`. (Some knowledge of SQL is required.)

The following example shows the host and users for all the jobs that ran for more than one hour.

```
SELECT hostid, userid, endtime - starttime AS duration
    FROM jobs
    WHERE duration > 3600
    ORDER BY duration DESC
    LIMIT 100;
```

There are two ways to pass the SQL command:

1. Directly on the command line:

   ```
   % vovsql_query -e "SELECT hostid, userid, ...."
   ```

2. Within a file:

   ```
   % vovsql_query -f file_with_your_sql_query
   ```

Database Schema

The SQL schema used by this version of Accelerator is visualized in the following entity-relationship diagram.

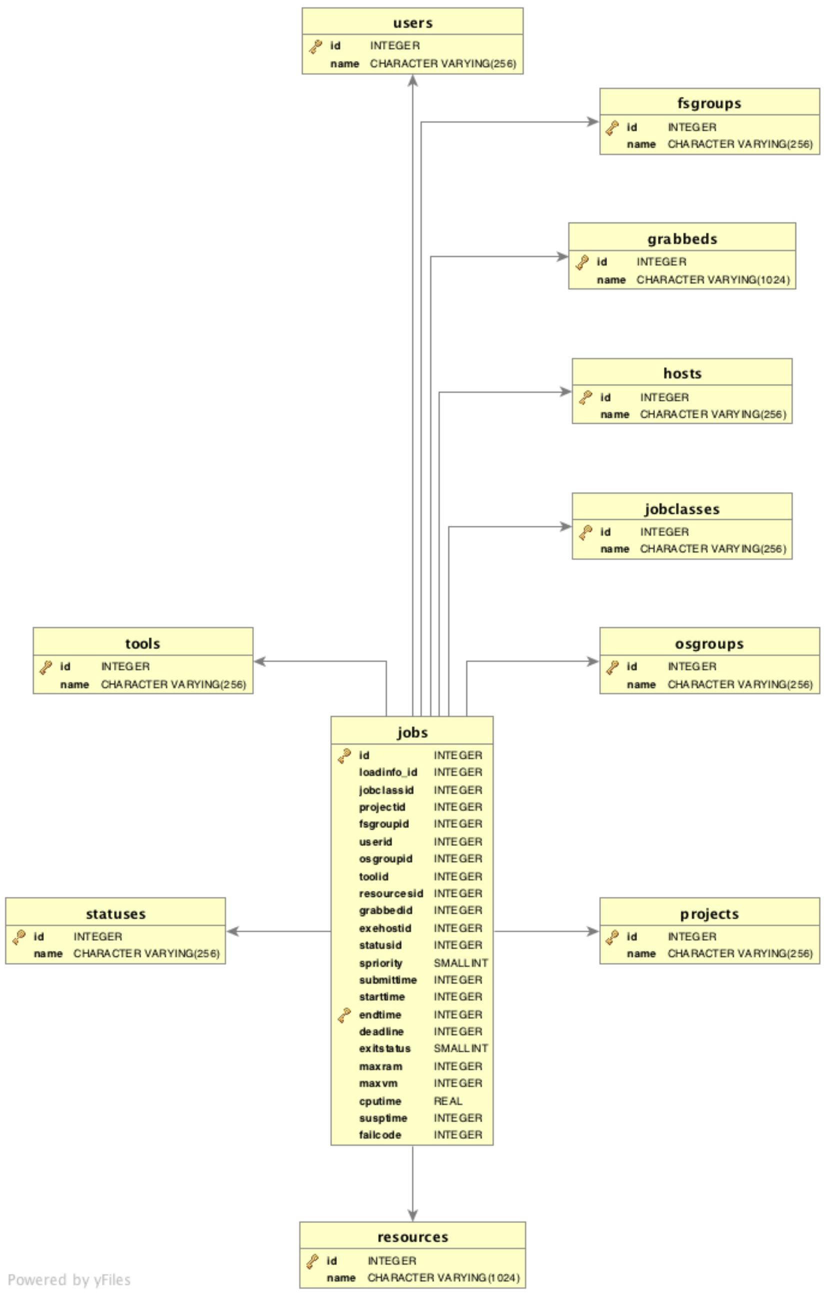


Figure 7: The Accelerator Database Schema

FairShare

FairShare allocates CPU cycles among groups and users according to policies defined by the administrators. FairShare is the dominant criteria in the scheduler, more important than job priorities. The fairness of CPU time allocation is computed using a multi-level FairShare tree. Each node in the tree is called a *FairShare Group* (abbreviated `fsgroup`) and is characterized by a *name*, a *weight*, and a *time window*.

Each job belongs to one and only one `fsgroup`. The contribution of each job to the FairShare mechanism is controlled by the parameter *`fstokens`*, which is 1 by default. If *`fstokens`* is 0, then the job will not contribute anything to the actual share (this is only rarely useful). If *`fstokens`* is 2, then the job contributes twice as much as a regular job with *`fstokens`* set to 1.

An `fsgroup` is "active" if some of its jobs are either running or queued, or, recursively, if any of its children is active.

The FairShare tree can be surprisingly large. Some organization have more than 16,000 nodes in the tree, on account of the large number of projects and users, although typically, at any one time, only less than 100 `fsgroups` are active.

The goal of the FairShare algorithm is to allocate resources to the active `fsgroup` so that the actual share of resources is as close as possible to the target share defined by the weights. To be clear, the `fsgroups` that are not active are not considered in the FairShare algorithm.

- The **target share** is computed from the `fsgroup` weights and allocated to all active `fsgroups`. The inactive fsgroups get a target of 0%
- The **actual share** is computed from the contribution of each job according to the overlap of the job execution and the time window multiplied by the *`fstokens`* parameter. The actual share consists of two components: the "running actual share" based on the number of jobs currently running, weighted by *`fstokens`*, and the "historical actual share" based on the overlap of the job execution time and the time window, also weighted by *`fstokens`*.

Each active `fsgroup` is assigned a 'rank' computed from the difference between its target share and actual share. That rank is then assigned implicitly to all jobs that belong to that `fsgroup`. The `fsgroup` that has the highest deficit will get rank of 0, while the `fsgroup` with the largest excess share will get a large rank (depending on current number of active `fsgroups`). The rank determines which jobs are preferred for dispatch, so that the scheduler first considers dispatching the jobs that have lower rank, i.e. jobs from `fsgroups` under their target share. If those jobs cannot be dispatched, because of other constraints such as RAM or limits, then the scheduler considers jobs with higher rank.

The default FairShare window is 2 hours meaning that we consider the time interval starting 2 hours before the present. Normally all nodes in the FairShare tree have the same time window, but that is not a requirement. In particular, it is possible to set the time window to zero in a node to disable FairShare for nodes under that node, by setting the rank of all children to the same value.

Selecting the appropriate window size is a balance between responsiveness and accuracy. A wide time window required more computation than a narrow window. The average job length and overall daily workload should be taken into account when selecting an appropriate window size.

As a rule of thumb, if your workload is small, i.e. under 100,000 jobs per day, do not worry about the FairShare window. If your workload exceeds 100,000 jobs per day, perhaps you want to use a shorter time window, such as 10 minutes. Workloads of millions of jobs per day can benefit with a time window of 2 minutes. Also relevant here is the frequency of update of the actual shares, which is controlled by the parameter *`fairshare.updatePeriod`*. The default value for this parameter is 0, meaning that the FairShare data is updated a frequently as needed, perhaps multiple times a second. For large workloads it may be a good idea to set that parameter to 3 or 5 seconds.

FairShare Rank

A `fsgroups's` rank ranges from zero upward. Jobs from `fsgroups` ranked closest to zero are preferred for dispatch. The rank is computed by ordering the `fsgroups` by a combined 'distance' between its target share and actual share. This distance has a 'running' component and a 'historical' component, based on jobs in the window. The vovserver configuration parameters *`fairshare.relative`* and *`fairshare.relative_alpha`* control the influence of the historical versus running distance on the actual rank. Refer to Server Configuration for details.

FairShare features:

- Multiple multi-level FairShare trees are supported. The default number of levels is 2.
- Each node in a FairShare tree has its own window size and weight.
- Ability to disable FairShare for a sub-tree by setting the window size to zero.
- Privileges are controlled with Access Control Lists (ACLs) for fine grained control.

FairShare Tree Naming Conventions

Each `fsgroup` has a hierarchical name where the components are separated by a "/", similar to a file name. The default `fsgroup` is `/time/users`. The name can take one of the following three forms:

Type	Form	Example
FS-Group	HIERARCHICAL_GROUP_NAME	/time/users
FS-User	HIERARCHICAL_GROUP_NAME.USEI	/time/users.joe
FS-Subgroup	HIERARCHICAL_GROUP_NAME.USEI	/time/users.joe:myregression1

Each component in the name has to be alpha-numeric, and can contain _. The . character is not allowed except in the FS-User component. The / and : are not allowed anywhere.

Type	Example
FS-Group	/proj/sanjose/library/qa
FS-User	/proj/sanjose/library/qa.john
FS-Subgroup	/proj/sanjose/library/qa.john:mytest1

Each node in the FairShare tree has an owner who has the authority to set the weights for all the subnodes in the tree. For example, the owner of group `/time/med` can set the weights for `/time/med/sanjose` and any other nodes of the form `/time/med/*`.

Define FairShare Groups

The FairShare tree is dynamic and can be changed at any time. If you like a configuration, you can save it into a file and then you can reload it at a later time. The main tool to perform these actions is `vovfsgroup`

FairShare Command Line Utilities

To view all FairShare groups, use `vovfsgroup` show:

```
ID         GROUP                                                OWNER WEIGHT   WINDOW
 RUNNING    QUEUED
000000016 /                                                  (server)      0    1h00m
 0        0
000001012 /system                                              cadmgr    100       0s
 0        0
000001050 /system/taskers                                       cadmgr    100       0s
  0        0
000001053 /system/taskers/messages                              cadmgr    100       0s
  0        0
000001056 /system/taskers/reservations                          cadmgr    100       0s
  0        0
000001006 /time                                                cadmgr    100    1h00m
 0        0
000001009 /time/users                                          cadmgr    100    1h00m
 0        0
000001081 /time/users.cadmgr                                   cadmgr    100    1h00m
 0        0
```

An older method still available is to use `vovshow -groups`:

```
% vovshow -groups
  ID        GROUP                           WEIGHT   WINDOW
  02223424 /system                             100    1m00s
  02223422 /time                               100    1h00m
  02223423 /time/users                           1    2h00m
  02223435 /time/users.cadmgr                  100    1h00m
```

For a specific `fsgroup`, you can use an additional argument to `vovfsgroup show FSGROUPNAME`:

```
% vovfsgroup show /time/users
Id:       000001009
FullName: /time/users
Owner:    cadmgr
Weight:   100
Window:   1h00m
Rank:     -1
        ACL   1: OWNER      ""   ATTACH DETACH EDIT VIEW STOP FORGET DELEGATE EXISTS
        ACL   2: EVERYBODY  ""   ATTACH VIEW
 000001081 /time/users.cadmgr                          100          cadmgr
```

Permission is required to create `fsgroups` and to change their weight. You can try the following commands as the ADMIN user for your Accelerator instance:

```
% vovproject enable vnc
% vovfsgroup create /app/primetime
% vovfsgroup create /app/spice
% vovfsgroup create /app/other
% vovfsgroup modify /app/primetime weight 300
% vovfsgroup modify /app/spice     weight 100
% vovfsgroup modify /app/other     weight  20
% vovfsgroup modrec /app           window 1h
% vovfsgroup exists /app/other
% vovfsgroup exists /app/not_there
% vovfsgroup genconfig  saved_my_cool_config.tcl    ### Important to use the .tcl
 extension
% vovfsgroup delete /app
% vovfsgroup loadconfig saved_my_cool_config.tcl
```

For more information, refer to Configure FairShare via the vovfsgroup Utility.

Monitor FairShare

From the command line, start the monitors with

```
% nc monitor
```

From the browser interface, visit the Project Home page and then select the FairShare link.

Target Share Example

In the following example, it is assumed two groups are defined, the default group `/time/users` and another group named `/time/regr`. The users `maureen` and `murali` are members of the `/time/users` group. User `john` is a member of the `/time/regr` group. It is also assumed that all users have jobs queued. Following is how the target shares would be determined using the two-tier method.

There are two groups in the queue, each group with a weight of 100. Therefore, at the start, the FairShare percentage of each can be calculated as such:

```
/time/regr              share = 100/(100+100)    50%
/time/users             share = 100/(100+100)    50%
```

Now assume that within the `/time/regr` group, only `john` has jobs. That user gets 100% of the group's share or 50% of the overall cycles.

```
/time/regr.john         share = 100/(100+100)  50% * 100% = 50%
/time/users             share = 100/(100+100)    50%
```

Within the `/time/users` group, two users have jobs as shown below:

```
 /time/users.maureen     share = 10/(10+10)       50% * 50% grp = 25%
 /time/users.murali      share = 10/(10+10)       50% * 50% grp = 25%
```

If another user `suresh`, who is a member of the `users` group submits jobs that are queued, the target shares would change as follows:

```
/time/users.maureen     share = 10/(10+10+10)    33% * 50% grp = 16.7%
/time/users.murali      share = 10/(10+10+10)    33% * 50% grp = 16.7%
/time/users.suresh      share = 10/(10+10+10)    33% * 50% grp = 16.7%
```

Because the user `suresh` just entered the queue, his actual share will probably be much less than the target share. Therefore, his jobs will be launched ahead of the other users as the system tries to bring his actual share up to his target share. An example of the overall FairShare picture is shown below (target shares shown):

```
/time/regr.john         share = 100/(100+100)   100% * 50% grp = 50.0%
/time/users.maureen     share = 10/(10+10+10)    33% * 50% grp = 16.7%
/time/users.murali      share = 10/(10+10+10)    33% * 50% grp = 16.7%
/time/users.suresh      share = 10/(10+10+10)    33% * 50% grp = 16.7%
```

Configure FairShare via the vovfsgroup Utility

The `vovfsgroup` utility is used to manage FairShare groups (`fsgroups`) from the command line. This includes creating, deleting and listing them with attributes and ownership.

`Fsgroups` have an associated owner and an Access Control List (ACL) to describe the operations that may be performed on the group. ACLs permit management of `fsgroups` to be delegated and distributed among multiple users.

The `vovfsgroup` command is also used to manage the ACLs of `fsgroups`. (see also `vtk_acl_op`).

vovfsgroup

Create, show, and modify attributes of FairShare groups. The script also controls Access Control Lists (ACL) for FairShare groups.

```
vovfsgroup: Usage Message

      DESCRIPTION:
      Create, show, and modify attributes of FairShare Groups.
      The script also controls Access Control Lists (ACL) for
      FairShare groups. See information on ACL for additional detail.

      SYNOPSIS:
      % vovfsgroup <action> <group> ...

      WHERE:
      <action>      is one of "acl, aclrec, create, exists, delete, genconfig,
                                 loadconfig, modify, modrec, normalize, show"
      and is case-insensitive
      <group>       is the name of the FairShare group

      USAGE:
      vovfsgroup acl <group> GET
                    // Retrieve FairShare Group ACLs
      vovfsgroup acl <group> RESET
                    // Reset FairShare Group ACLs to Default
      vovfsgroup acl <group> APPEND OWNER "Privilege List"
                    // Append Privs to FairShare Group
      vovfsgroup acl <group> APPEND EVERYBODY "Privilege List"
                    // Append Privs to FairShare Group
      vovfsgroup acl <group> APPEND USER <user> "Privilege List"
                    // Append Privs to FairShare Group

      Same as above, but apply setting recursively
      to all nodes in the FairShare group:
      vovfsgroup aclrec <group> GET
                    // Retrieve FairShare Group ACLs
      vovfsgroup aclrec <group> RESET
                    // Reset FairShare Group ACLs to Default
      vovfsgroup aclrec <group> APPEND OWNER "Privilege List"
                    // Append Privs to FairShare Group
      vovfsgroup aclrec <group> APPEND EVERYBODY "Privilege List"
                    // Append Privs to FairShare Group
      vovfsgroup aclrec <group> APPEND USER <user> "Privilege List"
                    // Append Privs to FairShare Group

      vovfsgroup create <group>
```

```
                    // Create a new FairShare group

       If the owner-user of the product instance is performing the clone:
         vovfsgroup clone -take <group> <new group>
                    // Clone an existing group and all subgroups to a new group,
                    //   with the current user taking ownership of the new group
                    //   and all subgroups
         vovfsgroup clone -preserve <group> <new group>
                    // Clone an existing group and all subgroups to a new group,
                    //   with ownership of the new group and all subgroups
                    //   copied from the original group
       If any other user is performing the clone:
         vovfsgroup clone <group> <new group>
                    // Clone an existing group and all subgroups to a new group,
                    //   with the current user taking ownership of the new group
                    //   and all subgroups

       NOTE: the owner-user of the product instance MUST specify either -take
             or -preserve with the clone command.  Users other than the
             owner-user cannot use the -preserve argument.  Only subgroups to
             which the user has access will be cloned.

       vovfsgroup exists <group>
                    // Exit status = 0 if group exists
       vovfsgroup delete <group>
                    // Delete a FairShare group
       vovfsgroup delete -unused
                    // Remove unutilized FairShare groups

       vovfsgroup modify <group> weight <integer-value>
                    // Change the weight of a FairShare group
       vovfsgroup modify <group> window <time-spec>
                    // Change the window size of a FairShare group
       vovfsgroup modify <group> owner  <owner-name>
                    // Change owner (requires SERVER status)
       vovfsgroup modify <group> flatten <0|1>
                    // Changed the flattened/non-flattened state of the group
                    //  (0 by default). If a group is flattened, its target
                    //  share is calculated differently; instead of the usual
                    //  hierarchical weighting, all non-leaf node weights are
                    //  ignored and leaf nodes are weighted against each other
                    //  as though they were all part of the same level of
                    //  hierarchy.  Note that vovfsgroup modrec should not be
                    //  used with flatten; it will work but is inefficient,
                    //  as vovfsgroup modify will already propagate the flag
                    //  to any child groups.


       Same as above, but apply setting recursively
       to all nodes in the FairShare group:
       vovfsgroup modrec <group> weight <integer-value>
                    // Change the weight of a FairShare group.
       vovfsgroup modrec <group> window <time-spec>
                    // Change the window size of a FairShare group.
       vovfsgroup modrec <group> owner  <owner-name>
                    // Change owner (requires SERVER status).
       vovfsgroup genconfig <NEW_CONFIG_FILE>
                    // Generate config file (name specified)

       EXAMPLES:
       % vovfsgroup help
       % vovfsgroup create /class/sim
       % vovfsgroup modify /time/users weight 121
```

```
      % vovfsgroup modify /time/users window 8h
      % vovfsgroup modify /time/users owner  mary
      % vovfsgroup modify /time/users flatten 1
      % vovfsgroup modrec /time/users window 8h
      % vovfsgroup exists /time
      % vovfsgroup show
      % vovfsgroup show   /time/users
      % vovfsgroup delete /class/sim /class/verilog
      % vovfsgroup delete -unused
      % vovfsgroup acl    /class/sim  GET
      % vovfsgroup acl    /class/sim  RESET
      % vovfsgroup acl    /class/sim  APPEND EVERYBODY "ATTACH VIEW"
      % vovfsgroup acl    /class/sim  APPEND USER jong "ATTACH VIEW"
      % vovfsgroup acl    /class/sim  APPEND USER jong "ATTACH VIEW"
      % vovfsgroup acl    /           SET    OWNER "ALL"
      % vovfsgroup normalize /time/projects 1000
      % vovfsgroup genconfig
      % vovfsgroup genconfig -leaf MyGroupsIncludingLeafNodes.tcl
      %
      % vovfsgroup genconfig myconfig.tcl
      % vovfsgroup loadconfig myconfig.tcl
      %
      % vovfsgroup genconfig -serial myconfig.txt
      % vovfsgroup loadconfig myconfig.txt
```

vovfsgroup Examples

This section provides examples of using `vovfsgroup`.

- Show the existing FairShare groups
- Show details of an existing FairShare groups
- Create a new FairShare group
- Modify a FairShare group
- Delete a FairShare group
- Set default for FairShare group
- Set ACL for a FairShare group
- All children inherit the values of Window and Weight.

In all examples below, the command `nc cmd` as a prefix to make it clear that we want the command to be executed in the Accelerator project. The prefix can be omitted if you first do a `vovproject enable vnc`. You also need to have ADMIN privileges for most of these commands to work.

Note: Configuration files can be viewed in various scripting languages. By default, the scripting language is Tcl. The option -serial allows using serial text languages such as Perl. Example:

```
% vovfsgroup genconfig -serial    #content is saved in serial text format,
 one group per line
% vovfsgroup genconfig            #content is saved in hierarchical  tcl
 format
```

The `vovfsgroup create` command will now copy the parent ACL when creating a subgroup:

```
vovfsgroup create /abc/def
```

The above command will create a new group `/abc/def`, with ACL permissions copied from the group `/abc`. If there is no applicable parent group, the default ACLs will be used.

Show the Existing FairShare Groups

The following example shows the summary of the existing `fsgroups`, with their owner, weight, and time window. The `/time` and `/system` FairShare trees are built-in. The `/system` tree is used by Accelerator and should not be modified or extended.

```
% nc cmd vovfsgroup show
ID       GROUP                                         OWNER WEIGHT   WINDOW
00001142 /system                                     cadmgr      100    1m00s
00001140 /time                                       cadmgr      100    1h00m
00001144 /time/production                            cadmgr      700    1h00m
00001145 /time/production.joe                        cadmgr        1    1h00m
00001147 /time/regression                            cadmgr      300    1h00m
00001141 /time/users                                 cadmgr       10    2h00m
00001185 /time/users.joe                             cadmgr      100    1h00m
```

Show Details of Existing FairShare Groups

The following example shows the details of the `/time/users` FairShare group. It shows that the owner has all ACL privileges, and that everybody has the ATTACH and VIEW privileges. These privileges are needed to automatically create the default `/time/users.<user-name>` group the first time a user submits a job without specifying a group name.

```
% nc cmd vovfsgroup show /time/users
    OWNER "" {ATTACH DETACH EDIT VIEW RETRACE STOP SUSPEND FORGET DELEGATE}
EVERYBODY "" {ATTACH VIEW}
00001185 /time/users.joe                              100         joe
```

Create a New FairShare Tree

This command creates the new fsgroup /division/project/block. The intermediate level /division/project is created automatically. You must be logged in as a user with ADMIN privilege level in Accelerator's `security.tcl` file.

```
% nc cmd vovfsgroup create /division/project/block
```

When you create a new `fsgroup`, you become its owner. At present, there is no way to change ownership except to delete the `fsgroup`, then re-create it as the desired owner. See DELEGATE.

The initial ACLs assigned to an `fsgroup` when it is created are:

```
DEFAULT_ACL  1: OWNER      ""   ATTACH DETACH EDIT VIEW FORGET DELEGATE EXISTS
DEFAULT_ACL  2: ADMIN      ""   ATTACH DETACH EDIT VIEW FORGET
DEFAULT_ACL  3: EVERYBODY  ""   ATTACH DETACH VIEW
```

Modify a FairShare Group

The first command modifies the fsgroup /division/project/block. The weight is set to 200. The share assigned to this group will be the weight divided by the sum of the weights of other active `fsgroups` at this level (i.e. all other groups that have running or queued jobs).

The second command modifies the fsgroup /division/project/block and sets its time window to 4 hours. This is the interval over which recent jobs continue to contribute to FairShare rank.

```
% nc cmd vovfsgroup modify /division/project/block weight 200
% nc cmd vovfsgroup modify /division/project/block window 4h
```

Append or Delete an ACL Element from a Group

APPEND can be used to append the defined privileges to the ACL list. DELETE is used to do the exact opposite of it, deleting only what is specified.

For example, an fsgroup /mygroup with one of the ACLs being

```
USER "john"   ATTACH DETACH VIEW
```

and then executing

```
$ nc cmd vovfsgroup /mygroup APPEND USER john EDIT
```

will append "EDIT" to this ACL and leads to

```
USER "john"   ATTACH DETACH EDIT VIEW
```

Now executing

```
$ nc cmd vovfsgroup /mygroup DELETE USER john "EDIT"
```

will DELETE the "EDIT" privilege from the ACL again and leads to

```
USER "john"   ATTACH DETACH VIEW
```

Set Default Weight for a FairShare Group

The following creates the fsgroup /division/project/default, and assigns it a weight of 200. When another `fsgroup` is created at the same level, that is, a sibling of `default`, it automatically is assigned the weight from the `default` group at that level.

```
% nc cmd vovfsgroup create /division/project/default
% nc cmd vovfsgroup modify /division/project/default weight 200
```

Set Access Control Lists for a FairShare Group

The following sequence of commands first gets the current ACL of the group /time/regression for inspection. The second command resets the ACL to its default value.

```
% nc cmd vovfsgroup acl /time/regression GET
% nc cmd vovfsgroup acl /time/regression RESET
% nc cmd vovfsgroup acl /time/regression APPEND  EVERYBODY "ATTACH VIEW"
% nc cmd vovfsgroup acl /time/regression APPEND  USER regrmgr "ATTACH DETACH VIEW
 EDIT FORGET DELEGATE"
```

The third and fourth commands add the ACL for EVERYBODY and regrmgr.

```
ACL  1: OWNER      ""   ATTACH DETACH EDIT VIEW FORGET DELEGATE EXISTS
ACL  2: EVERYBODY  ""   ATTACH VIEW
```

Configure FairShare via File

The options are:

-w (weight)	Specify the relative weight of the group (default given by variable $FSGROUP(weights), which is normally 100).
-t (timeWindow)	Specify the time window used in the computation of the actual shares (default given by variable $FSGROUP(windows) which is normally 7200, which is 2 hours). **Note:** A time window of duration zero can be used to disable FairShare for a subset of the FairShare tree.
-u (List-Of-Users)	Specify a list of users that have exclusive access to that FairShare group. This access is controlled by an Access Control List.
-user	Specify that the group is a USER level group

A FairShare configuration file that represents current FairShare state can be created in the web user interface. Click **FairShare** > **Hierarchical Configuration**. The `vovfsgroup genconfig` command provides a way to generate a FairShare cofiguration file from the CLI.

Tcl Example

```
% vovfsgroup genconfig  config.mysetup.tcl
% vovfsgroup genconfig -leaf config.mysetup_with_leaf_nodes.tcl
```

A FairShare configuration file (as generated by the example above) can be imported into the system with the command `vovfsgroup loadconfig`, as follows:

```
% vovfsgroup loadconfig config.normal.tcl
```

Where `config.normal.tcl` contains the FairShare configuration declared in Tcl as follows:

```
# Copyright (c) 1995-2023, Altair Engineering
# All Rights Reserved.
# $Id: //vov/branches/2023.1.2/src/etcDir/config/fairshare/config.normal.tcl#2 $

#
# This is an example of config for fairshare.
#
FSGROUP "class" {

    FSGROUP "sim" -w 0 -t 0 {
 # In this branch, fairshare is disabled
 # because the timewindow is zero (-t 0)
 FSGROUP jolly
 FSGROUP cronos  -w 100 -t 3h
    }
    FSGROUP "urgent" -window 10m {
 FSGROUP h2p -w 300
 FSGROUP jolly
 FSGROUP cronos
    }
    FSGROUP projects -w 133 {
 FSGROUP jolly  {
     FSGROUP normal -w 200 {}
     FSGROUP random -w  80 {}
 }
 FSGROUP cronos {
     FSGROUP normal
     FSGROUP random
 }
    }
}
```

Serial Example

The `vovfsgroup genconfig -serial` command provides a way to generate a FairShare configuration file in the "serial" format.

```
% vovfsgroup genconfig -serial config.mysetup.txt
% vovfsgroup genconfig -leaf -serial config.mysetup_with_leaf_nodes.txt
```

A FairShare configuration file in serial format (as generated by the example above) can be imported into the system with the command `vovfsgroup loadconfig`, as follows:

```
% vovfsgroup loadconfig config.normal.txt
```

Where the content of file `config.normal.txt` is as follows:

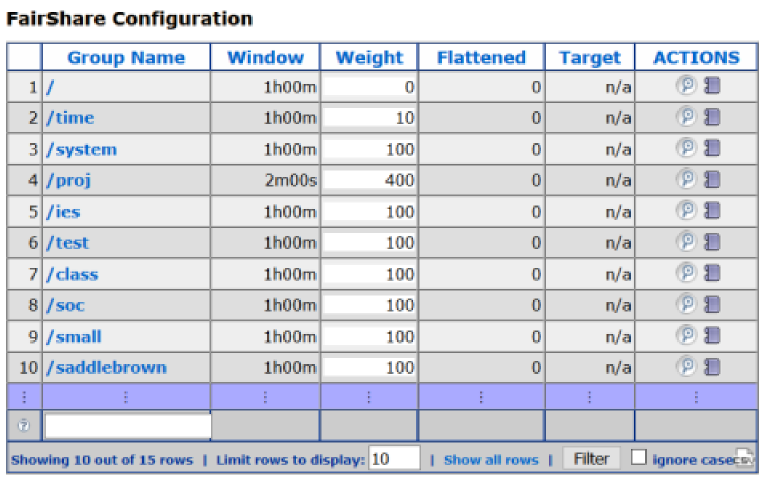
```
# FairShare configuration file created by vovfsgroup on Mon May 02 16:55:09 EDT 2022
# Output is in tab-separated key=value format
# ACL values are in AGENT/NAME/ACTIONS format, with multiple ACLs separated by a
 comma

group=/ weight=16777215 window=-1s flatten=0
group=/time weight=100 window=1h00m flatten=0
group=/time/users weight=100 window=1h00m flatten=0 acl=OWNER//ATTACH DETACH EDIT
 VIEW STOP FORGET DELEGATE EXISTS CREATE,ADMIN//ATTACH DETACH EDIT VIEW FORGET
 CREATE,EVERYBODY//ATTACH DETACH VIEW
group=/system weight=100 window=0s flatten=0
group=/system/taskers weight=100 window=0s flatten=0
```

```
group=/system/taskers/closures weight=100 window=0s flatten=0
group=/system/taskers/messages weight=100 window=0s flatten=0
group=/system/vovdbd weight=100 window=0s flatten=0
```

FairShare Weights Control Methods

The FairShare weight is capped at a maximum value of 100,000. The current implementation uses an integer scale. The use of large ratios within a FairShare group, often as an attempt to implement a priority based queuing system, is discouraged.

Set FairShare Weights Via the Web Interface



FairShare Configuration

	Group Name	Window	Weight	Flattened	Target	ACTIONS
1	/	1h00m	0	0	n/a	
2	/time	1h00m	10	0	n/a	
3	/system	1h00m	100	0	n/a	
4	/proj	2m00s	400	0	n/a	
5	/ies	1h00m	100	0	n/a	
6	/test	1h00m	100	0	n/a	
7	/class	1h00m	100	0	n/a	
8	/soc	1h00m	100	0	n/a	
9	/small	1h00m	100	0	n/a	
10	/saddlebrown	1h00m	100	0	n/a	

Showing 10 out of 15 rows | Limit rows to display: 10 | Show all rows | Filter | ignore case

Figure 8:

Accelerator Admins can change the FairShare group weights from the browser. The weights can either be changed by using the +/- buttons on the right, which changes the weight by 10%, or the weight can simply be entered under the weight column. As with many configurations in Accelerator, the changing of weights is only permitted by Accelerator Admins.

You can also use `vovfsgroup` to change the FairShare weights at any time.

Multiple Tokens in FairShare

In FairShare, the number of license tokens for each job is counted. Sometimes there are jobs that use four tokens of a license and other jobs that use only one token of a license, each to access different specific tool functionalities. For the best of use of resources and scheduling, it may be undesirable to account for both types of jobs in the same way. To specify how selected jobs contribute to FairShare, use the field `FSTOKENS` in the `nc run ...` submission string.

The `FSTOKENS` field can be set on each job of which the status is neither Failed nor Done. The default value is 1. Typical values are small integers, such as 2, 3, 4 or 8. The maximum value is 50000. The minimum value is 0, which indicates that the job does not contribute to the FairShare value.

`FSTOKENS` can be applied for jobs in which the status is not Failed or Done.

To set `FSTOKENS` in Accelerator use the option -fstokens in `nc run`. An example using a "MMSIM" license follows:

```
% nc run -fstokens 4 -r License:MMSIM#4 -- my_job_script_that_uses_4_tokens
```

The above will result in an decreased impact when FairShare is calculated for the job. This strategy can be used for multi-token tools to help them attain a more favorable FairShare positioning.

Note: Setting this up for a job type is easily accomplished by implementing a job class that includes the appropriate token argument to `-fstokens`. See Jobclasses for details.

FairShare Recommendations

Initially: Do Nothing

The first recommendation is to do nothing. Start with the simplest FairShare tree, where every job belongs to the fsgroup /time/ users.USERNAME where USERNAME is of course the owner of the job. This means that CPU cycles are allocated to all users in equal parts.

Later: Design a FairShare Tree for Both Applications and Projects

Over time, as you acquire experience in FairShare, you may want to design a more complex FairShare tree, one that accounts for allocations of expensive licenses and for all the projects going on in your organization.

Our recommendation is to organize your FairShare tree first by application, then by project, so that a typical name for a FairShare group could be /class/spice/ChipA.joe

```
## Example of design of a production fairshare tree.
## This file could be  vnc.swd/fairshare/main_fs_tree.tcl

set listOfApps     "spice dc pt"
set listOfProjects "X3 X5 X8 RT SW chameleon"
FSGROUP / -w 0 -t 1h {
    foreach app $listOfApps {
         FSGROUP $app  -w 100 -t 1h  {
               foreach proj $listOfProjects {
                    FSGROUP $proj -w 100 -t 1h {}
               }
               FSGROUP other  -w 10 {}  ;#  All jobs not in a project
               FSGROUP default -w 1 {}  ;#  In case a new projects appear, give a
 small weight
         }
    }
    FSGROUP extra   -w 10 -t 1h {}      ;# All jobs not in an app.
    FSGROUP default -w 1        {}      ;# In case new apps appear, give them a small
 weight.
}
```

To load this FairShare tree, use:

```
% vovfsgroup delete -unused
% vovfsgroup loadconfig .../main_fs_tree.tcl
```

This will add the tree in addition to all currently used groups in your system.

The users would then submit jobs using the appropriate fsgroup, for example with:

```
% nc run -g /app/spice/chameleon -r License:spice -- myspice -i ckt.spi
```

To simplify the submission, we also recommend using Jobclasses, where both resources and FairShare groups can be defined together, as in the following example:

```
# This could be file  vnc.swd/jobclass/spice.tcl
set VOV_JOB_DESC(resources)  "License:spice RAM/200"
set VOV_JOB_DESC(group)      "/app/spice/$env(PROJECT)"
```

With the jobclass, the submission is greatly simplified:

```
% nc run -C spice myspice -i ckt.spi
```

At some point, it may happen that one project enters a critical phase, e.g. the tapeout of a chip, so it may benefit from a larger share of resources. This can be achieved by changing the weights of all fsgroups for that project, for example with a script like this which sets all fsgroups for project "chameleon" to 300:

```
# Assuming C-shell syntax
foreach g ( `vovselect fullname from fairshare -where name==chameleon`)
   vovfsgroup modify $g weight 300
end
```

Disable FairShare

Reducing the FairShare window or completely disabling FairShare may improve the performance of the scheduler.

Note: This configuration is only effective when the workload is very large, such as several million jobs per day.

FairShare can be disabled on an individual node in the FairShare tree by setting the time window size to zero. When FairShare is disabled, jobs that belong to the disabled branches of the FairShare tree are scheduled using a round-robin scheduler that considers the oldest buckets first.

For example, to disable FairShare on the subtree rooted at `/class`, use the following command:

```
% vovfsgroup modrec /class window 0
```

This can be done at any time and has immediate effect, which is that all jobs in any subgroup of `/class` will be considered for scheduling based on the time a job has been waiting in the queue.

Re-enable FairShare

To re-enable FairShare on a subtree of the FairShare tree, set the FairShare window to something different from zero. Example:

```
% vovfsgroup modrec /class window 2h
% nc cmd vovproject sanity
```

Note: As of release 2015.09, all FSGROUP properties are persistent between Accelerator restarts. If you are using a release prior to that, the above command line changes will be temporary until the next restart.

FairShare Parameters

The FairShare system is controlled by the following parameters:

`fairshare.allowAdminBypass`	Allow ADMIN to bypass ACL when modifying a FairShare group. Default is 0. Set this to 1 if you want to allow any ADMIN to change any FairShare setting.
`fairshare.default.weight`	Default weight assigned to new FairShare groups, typically set to 100. Can also be controlled by a sibling group called 'default'. In other words, if a new FairShare group is created, its weight will be the same as that of a sibling group with name "default", else it will be determined by this parameter *`fairshare.default.weight`*.
`fairshare.default.window`	Default window assigned to new FairShare groups. Normally the groups inherit the window from their parent of from a sibling group called 'default' (*`fairshare.default.weight`*)
`fairshare.relative`	Controls how FairShare ranks are computed. In the following formulas, assume: • *t* = target allocation • *h* = historic allocation in window • *r* = running allocation n • *d* = distance from equilibrium, ultimately used to compute ranking of FairShare groups. The legal values for this parameter are: • If set to 0, compute FairShare distance by a simple difference between actual and target *d*=(*r*-*t*)+(*h*-*t*) • If set to 1, compute FairShare distance relative to the FairShare target *d*=((*r*-*t*)+(*h*-*t*))/*t* (of course assuming that the target *t*>0). • If set to 2, compute FairShare as a weighted sum of *d* = (*r*-*t*) + #(*h*-*t*), where # is the parameter *`fairshare.relative_alpha`* explained below
`fairshare.relative_alpha`	Weight of historic distance relative to running distance in computing value for FairShare ranking, but only when *`fairshare.relative`* is set to 2. The default value for this is 10, meaning that history counts 10 times more than current allocation.
`fairshare.updatePeriod`	Update FairShare stats no more frequently than specified period. The default is 0, meaning a period of 0 seconds which means constant update, i.e. that FairShare

	stats are updated as frequently as desired. A value of 1 or 2 seconds may improve server responsiveness.
fairshareMode	Obsolete, was used to turn FairShare on and off.

Control FairShare Tree Access

By default, the FairShare tree is extendable; new nodes can be added without default user permissions. This enables users to add their own nodes, including those implicitly added at submission time by inclusion in the command line.

For example, here is a minimal FairShare tree:

```
mac12 vncaux@mac12.local DEFAULT vnc/vncaux.swd > vovfsgroup show
ID         GROUP                                                OWNER WEIGHT   WINDOW
 RUNNING    QUEUED
000000016 /                                                  (server)      0    1h00m
 0         0
000001006 /system                                              taylor    100    1m00s
 0         0
000001004 /time                                                taylor    100    1h00m
 0         0
000001005 /time/users                                          taylor     10    2h00m
 0         0
000108049 /time/users.pistol                                   pistol    100    2h00m
 0         0
```

When users submit jobs, the jobs are added to the /time/users branch by default. In this case, the user `pistol` has recently submitted a job and a node was created for him, and his job was attached to it. This model allows users to run jobs without special settings, which requires a minimal amount of setup work for the administrator. By default, each job will get an equal share of the / time/users node.

However, a user can also run a job outside the /time/users node by using the -g option to the `nc run` command. In the following example, a job is running on the `/garterinn` node:

```
[localhost:~] falstaff% nc run -g /garterinn sleep 500
Fairshare= /garterinn.falstaff
Resources= macosx
Env      = SNAPSHOT(vnc_logs/snapshots/falstaff/macosx/env55747.env)
Command  = vw sleep 500
Logfile  = vnc_logs/20131015/161821.9853
JobURL   = http://mac12:6349/cgi/node.cgi?id=000108058
JobId    = 000108058
```

Looking at the FairShare tree summary:

```
mac12 vncaux@mac12.local DEFAULT vnc/vncaux.swd > vovfsgroup show
ID         GROUP                                                OWNER WEIGHT   WINDOW
 RUNNING    QUEUED
000000016 /                                                  (server)      0    1h00m
 1         0
000108056 /garterinn                                         falstaff    100    1h00m
 1         0
000108057 /garterinn.falstaff                                falstaff    100    1h00m
 1         0
```

```
000001006 /system                                        taylor    100     1m00s
 0        0
000001004 /time                                          taylor    100     1h00m
 0        0
000001005 /time/users                                    taylor     10     2h00m
 0        0
```

The above data shows that `/garterinn` has been added, which allows the job to run. Additionally, the weight for this node is set to the default value of 100. The total resources at the top level is now divided in 3 (`garterinn`, `system`, `time`) each with equal weights of 100. With this setup, `falstaff` has a FairShare allocation of 1/3rd of the compute resources. In other words, with unrestricted access, arbitrary users can easily tie up an inordinate proportion of the total resources.

Note: For this reason, it is often desirable to restrict access to the top level.

The example below shows the default permissions for the top level:

```
mac12 vncaux@mac12.local DEFAULT vnc/vncaux.swd > vovfsgroup show /
Id:       000000016
FullName: /
Owner:    (server)
        ACL   1: OWNER       ""   ATTACH DETACH EDIT VIEW FORGET DELEGATE EXISTS
        ACL   2: ADMIN       ""   ATTACH DETACH EDIT VIEW FORGET
        ACL   3: EVERYBODY   ""   ATTACH DETACH VIEW
```

The key value is the third permission listed, `ACL 3`. `ATTACH` indicates that `EVERYBODY` can attach node to this root location. When a job runs, it needs `ATTACH` permission. It would be desirable to reduce the `EVERYBODY` ACL to `VIEW`. However, there is no direct mechanism to selectively remove ACLs at this level of granularity. Instead, the solution is to go back to a zero ACLs and then add more.

Zero ACL

Changing to a zero ACL state on the root node '/' is problematic as that would remove your own permissions to edit the ACL. The workaround is to get the `SERVER` role. `SERVER` is a super user mode that ignores the restrictions implied by the ACLs. As well as resolving the issue of zero ACLs on the root node, the `SERVER` role also allows correcting other lock-out scenarios that may occur due to administration errors.

Important: In ACL terms, the `SERVER` role is the highest level of access, and is valuable as a *last resort* back door access.

To access the `SERVER` role requires an active login shell on the same host as the Accelerator server process (such as ssh into the Accelerator server host). Additionally, Accelerator must be accessed through the loopback interface 127.0.0.1. To do so, set `VOV_HOST_NAME=localhost`.

Once in the `SERVER` role, the root '/' node permissions can be fixed. This is done with three actions:

1. Set the `OWNER`
2. Set the `ADMIN`
3. Set `EVERYBODY` ACLs

Note: There is a side effect of working on the top level node '/'. The ACL change is applied recursively to all nodes, which will need to be fixed. Here is a transcript of the top level change:

```
[cadmgr@rtda01 ~]$ vovfsgroup acl / SET OWNER ALL
[cadmgr@rtda01 ~]$ vovfsgroup show /
Id:       000000016
FullName: /
Owner:    (server)
        ACL  1: OWNER      ""   ATTACH DETACH EDIT VIEW FORGET DELEGATE
 EXISTS
[cadmgr@rtda01 ~]$ vovfsgroup acl / APPEND ADMIN ALL
[cadmgr@rtda01 ~]$ vovfsgroup show /
Id:       000000016
FullName: /
Owner:    (server)
        ACL  1: OWNER      ""   ATTACH DETACH EDIT VIEW FORGET DELEGATE
 EXISTS
        ACL  2: ADMIN      ""   ATTACH DETACH EDIT VIEW FORGET DELEGATE
 EXISTS
[cadmgr@rtda01 ~]$ vovfsgroup acl / APPEND EVERYBODY VIEW
[cadmgr@rtda01 ~]$ vovfsgroup show /
Id:       000000016
FullName: /
Owner:    (server)
        ACL  1: OWNER      ""   ATTACH DETACH EDIT VIEW FORGET DELEGATE
 EXISTS
        ACL  2: ADMIN      ""   ATTACH DETACH EDIT VIEW FORGET DELEGATE
 EXISTS
        ACL  3: EVERYBODY  ""   VIEW
```

With the above setup, the user falstaff will be unable to move to the boarshead. Following is the message to expect when a user tries to submit a job to a nonexistent FairShare node, and the parent's node has been locked down.

```
[localhost:~] falstaff% nc run -g /boarshead sleep 500
vnc 10/17/2013 12:51:38: Error: Problem joining fairshare group /boarshead
 Please check with administrator to see that you have
 permissions to join the group.
vnc 10/17/2013 12:51:38: FATAL ERROR: Failed to submit batch job.
```

Since actions on / are currently recursive, it may be necessary to relax the restriction on the default group /time/users:

```
rtda01 vnc@rtda01 [BASE] 854 > vovfsgroup acl /time/users SET EVERYBODY "ATTACH VIEW"
```

Clean Up After an Escape

While the unpermitted attach was prevented (as described above), deleting the offending node is problematic. A node can only be deleted when it is empty; all jobs, including valid jobs, must be forgotten. When this is not possible, the alternative action is to reduce the weight of the offending node as shown below:

```
mac12 vncaux@mac12.local DEFAULT vnc/vncaux.swd > vovfsgroup modrec /garterinn weight
 1
mac12 vncaux@mac12.local DEFAULT vnc/vncaux.swd > vovfsgroup show
ID        GROUP                                                OWNER WEIGHT   WINDOW
 RUNNING   QUEUED
000000016 /                                                 (server)      0    1h00m
 0        0
```

```
000108056 /garterinn                                   falstaff      1     1h00m
 0        0
000108057 /garterinn.falstaff                          falstaff      1     1h00m
 0        0
000001006 /system                                        taylor    100     1m00s
 0        0
000001004 /time                                          taylor    100     1h00m
 0        0
000001005 /time/users                                    taylor     10     2h00m
 0        0
000128088 /time/users.taylor                             taylor    100     2h00m
 0        0
```

Jobclasses

Jobclasses provide the following advantages:

- Simplifying the command line for job submission, which can prevent errors and omissions.
- Emulate the concept of *queues*, which is similar to the processes of other batch processing systems. This queue emulation enables additional behaviors such as:
    - Automatic revocation of resources that have been grabbed by jobs in the jobclass but are not used
    - Automatic warning and termination of jobs that are stuck: jobs that have been dispatched to a vovtasker but appear to be using no CPU time

A jobclass represents a collection of `nc run` options that are needed to run a type of jobs, such as VCS regression jobs.

Membership in a jobclass can be used to differentiate between jobs in preemption: preempt jobs in a regression jobclass to free up resources for jobs in an interactive jobclass.

If more than one -C option is given, the jobclasses are processed left-to-right as the command line is parsed. This method requires great care and planning.

Create Jobclasses

The administrator of Accelerator can define jobclasses using one of the following methods:

- Logged in as ADMIN, click the **Job classes** link in the **Workload** section of the Accelerator main page. This page displays all of the available jobclasses, and allows creating and editing jobclasses, and allows the administrator to designate a default jobclass.
- Directly add Tcl syntax files in the directory `jobclass` under the server working directory, which is typically `$VOVDIR/../../vnc/vnc.swd/jobclass`.

Each file in the `jobclass` directory manipulates the submission parameters defined in the Tcl array VOV_JOB_DESC so as to define a jobclass. See the following table for the meanings of the items in this data structure. The variable *`classDescription`* holds a string used for documentation, i.e. a one-line summary of the jobclass. The variable *`classEditable`* holds a boolean value that controls whether the jobclass can be edited using the jobclass web UI page.

An optional procedure `initJobClass` can be defined to do any initializations needed for the jobclass to perform correctly. Often, this is used to create any Limit: resources that may be used by the jobclass.

The files in the jobclass directory are parsed by `vovresourced` when it starts, and any `initJobClass` procedures are evaluated once.

The jobclass definition files are located using a search path.

The built-in search path is computed by a procedure `VncJobClassSearchPath`, and is shown in the table below.

This procedure adds any directories named by the environment variable VOV_JOBCLASS_DIRS to the beginning of the path, analogous to VOV_ENV_DIR.

This procedure is defined in `$VOVDIR/tcl/vtcl/vovutils.tcl`. You may change the search path for jobclass files by including a modified definition in `vnc.swd/resources.tcl`, and also in `vnc_policy.tcl`, if used. See example.

Search path for jobclass definitions:

```
#directories named by VOV_JOBCLASS_DIRS
<project>.swd/jobclass
$VOVDIR/local/jobclass
$VOVDIR/etc/jobclass
```

Use Additional Jobclass Directories

If you have an extensive Accelerator setup, you may wish to manage jobclasses in a more-centralized way than placing their definitions into each `vncNNN.swd/jobclass` directory. The following example shows one way to accomplish this.

For this example, we want to implement a system where jobclasses are searched for first in the specific Accelerator queue, then in a site-specific directory, and finally in a global area.

We implement this by using the regular queue-specific directory, and two symlinks called *`jobclass_site`*, and *`jobclass_global`* under `vncNNN.swd` that resolve to the site-specific and global directories for jobclasses. You will need to arrange for the jobclass directories to be available and up to date at each site.

Additionally, some generic code is shown that may be dropped into `vncNNN.swd` to automatically compute the value of the VOV_JOBCLASS_DIRS environment variable.

Code to compute VOV_JOBCLASS_DIRS:

```
# This is file jcdirs.tcl
# NOTE: vovwait4server runs in vtclsh, and sources setup.tcl
# so vovGetProjectFileName may not be used here

# If env-var VNCSWD is set, use its value, it is much simpler
if { [info exists env(VNCSWD)] && [file isdirectory $env(VNCSWD)] } {
   set cfgdir [file join $env(VNCSWD) $env(VOV_PROJECT_NAME).swd]
} else {
    # compute path to the .swd directory
    # setup.tcl needs to be source-able in vtclsh, must use exec
    if { [catch {set sdp [exec vovsh -x {puts [vtk_server_dir -physical]}]} em] } {
        VovError "determining config directory -- $em"
    } else {
        set cfgdir [file join $sdp "$env(VOV_PROJECT_NAME).swd"]
    }
}

set jcdirs {}
foreach dn {jobclass jobclass_site jobclass_global} {
    set tdir [file join $cfgdir $dn]
    if { [file isdirectory $tdir] } {
        lappend jcdirs $tdir
        VovMessage "added jobclass dir '$tdir'" 3
    } else {
        VovMessage "non-existing jobclass dir '$tdir'" 3
    }
}
if { $jcdirs != {} } {
    setenv VOV_JOBCLASS_DIRS [join $jcdirs ":"]
}
```

The VOV_JOBCLASS_DIRS need to be set in all Accelerator-related processes. Sourcing the above script from `vncNNN.swd/setup.tcl` will arrange this for vovserver and the vovtaskers, which also inherit their environment from a shell that has sourced `setup.tcl`. To have it set in the clients, source it from `vnc_policy.tcl`, which is sourced by the top-level `vnc` command. The following example places the drop-in code in `vncNNN.swd/scripts/`:

```
# This is file jcdirs.tcl
# source the drop-in code for VncJobClassSearchPath{}
# compute the value of VOV_JOBCLASS_DIRS
if { [info exists env(VNCSWD)] } {
    set jcsetup $env(VNCSWD)/$env(VOV_PROJECT_NAME).swd/scripts/jcdirs.tcl
}
if { [file readable $jcsetup] } {
    if { [catch {source $jcsetup} smsg] } {
        VovError "jcsetup error -- $smsg"
    } else {
        VovMessage "jcsetup OK" 3
    }
} else {
    VovError "jcsetup not found -- $jcsetup"
}
```

Define a Default Jobclass

When defined, a default jobclass is evaluated for each job as it is submitted before any other `nc run` options are parsed. The default jobclass should be simple and limited to actions such as supplying basic values for RAM/ and CORES/.

When a default jobclass is in effect, the values it establishes may be changed if a jobclass is later named by the -C option of the `nc run` command.

There are two methods to designate a jobclass as the default:

- Via the Jobclass page web page.
- Setting a property from the CLI.

The default jobclass is determined by the value of the property NC_DEFAULT_JOBCLASS attached to the trace (VovId=1). You can use the utility `vovprop` to set this property. In the following example, `normal` is set as the default jobclass.

```
% nc cmd vovprop SET -text 1 "NC_DEFAULT_JOBCLASS" "normal"
```

View the default jobclass from the CLI by:

```
% nc cmd vovprop GET 1 NC_DEFAULT_JOBCLASS
```

Jobclass Definitions Examples

Example of job class "SHORT"

```
# This is file short.tcl
set classDescription  "Jobs taking less than 30s"
set classEditable     true; # Allow editing via web UI
set VOV_JOB_DESC(xdur)           30
set VOV_JOB_DESC(autokill)        1
set VOV_JOB_DESC(priority,sched) 8
set VOV_JOB_DESC(env)             "BASE+D(VOV_LIMIT_cputime=30)"

proc initJobClass {} {}
```

Example of jobclass "INTERACTIVE"

```
# This is file interactive.tcl
set classDescription         "Interactive Jobs"
set classEditable            false; # Disallow editing via web UI

set VOV_JOB_DESC(resources)       ""

# Make the environment unique for each interactive job,
# so that multiple submissions of the same command in the same
# directory will result in multiple jobs
set VOV_JOB_DESC(env) "BASE+D(uniquify=[clock seconds])"

set VOV_JOB_DESC(priority,sched) 9
set VOV_JOB_DESC(interactive,useXdisplay) 1

# We want Crtl-C and similar commands to be handled by the remote host.
set VOV_JOB_DESC(interactive,flag) "tty_remote"

# We do not want any wrapper for interactive jobs,
# to allow stdout and stdin to go directly to the TTY.
set VOV_JOB_DESC(wrapper) ""
```

Reconcile Unused Resources

Some tools have a complex behavior for acquiring and releasing licenses. For example, some tools may acquire a license for a short term and then release that license. Other tools may or may not acquire a specific license.

In some cases, if it is uncertain if licenses are needed, the best method is to request the licenses, and then reconcile (release) the licenses that are not used.

This method can be accomplished with jobclasses and the procedure `vtk_jobclass_set_revocation_delay $JOBCLASS_NAME $DELAY_SPEC`, which is typically called from within the `initJobClass` procedure in the jobclass definition.

Global Setting for Reconciliation

If jobclasses are not being used, or setting a global default value for automatic reconciliation is desired, set the variable *RESD(revokeDelay)* in the `vovreconciled/config.tcl` file.

For example:

```
set RESD(revokeDelay) 4m
```

Find and Remove "Stuck Jobs" using a Jobclass

To find stuck jobs, the jobs that are running but are burning no CPU longer than the timing threshold set by -maxNoCpuTime. A notification email is sent when job stuck time exceeds the threshold.

If the jobclass has a property `IDLE_KILL_DELAY` set, the job will be killed when the stuck time exceeds this property setting. An email will be sent to the owner at the time of killing the job.

To define the behavior of Accelerator with stuck jobs, use the procedure `vtk_jobclass_set_idle_delays`. This procedure is called inside the `initJobClass` procedure. This procedure requires three arguments as shown below:

```
vtk_jobclass_set_idle_delays $JOBCLASS_NAME $WARN_TIMESPEC $KILL_TIMESPEC
```

Define Jobclasses

This VOV_JOB_DESC data structure is an associative array that describes the characteristics of a job. The array has a number of slots that hold the values describing the job. The following table shows the array slot fields.

Field in Array	Description
autokill	Set the autokill flag (option `-kill`)
check,directory	Set it to 0 to disable checking of canonicalization of current directory (option `-D`)
env	Environment of the job (option `-e`). Set this to "" or to DEFAULT to force the use of an environment snapshot.
force	Force the job to be rescheduled (option `-F`)
group	Group the job belongs to (option `-g`)
group,final	Group the job belongs to including the user subgroup (option `-G`)
inputs	List of input files (dependencies) (option `-i`)
interactive,flag	Used for interactive jobs, with values `tty_remote` (option `-Ir`) or `tty_local` (option `-Il`)
interactive, useXdisplay	Set if the job requires an X display (option `-Ix`)
logfile	Name of the log file (option `-l`)
mailuser	Specification of who gets the e-mail notification (option `-M`)
osgroup	The UNIX group this job. This field is read-only and cannot be changed (see also *user*)
outputs	List of output files (option `-o`)
preemptable	A suggestion to determine if the job is preemptable
priority, default	Default priority (NOT USED)
priority, exec	Execution priority

Field in Array	Description
priority, sched	Scheduling priority
proplist	Properties to be added to the job (option `-P`)
resources	The resources of the job (option `-r`)
rundir	The running directory for the job (normally)
schedule, date	NOT SUPPORTED YET
setName	The set to which the job belongs (option `-set`)
user	The user for this job. This field is read-only and cannot be changed (see also *osgroup*)
wait	Boolean: set it to `1` to wait for the job to complete (option `-w`)
wait,options	When waiting, these options are passed to the `nc wait` command. For example, set it to `-l` to view the log file
wrapper	Wrapper used for the job (option `-wrapper`)
xdur	Expected duration of the job (option `-X`)

The following is an example of the populated VOV_JOB_DESC array.

```
VOV_JOB_DESC(autoforget)              = 0
VOV_JOB_DESC(autokill)                = 1
VOV_JOB_DESC(check,directory)         = 1
VOV_JOB_DESC(env)                     = BASE+RTSIM
VOV_JOB_DESC(force)                   = 0
VOV_JOB_DESC(group)                   = users
VOV_JOB_DESC(inputs)                  =
VOV_JOB_DESC(interactive,flag)        = none
VOV_JOB_DESC(interactive,useXdisplay) = 0
VOV_JOB_DESC(logfile)                 = vnc_logs/20050920/131409.25563
VOV_JOB_DESC(mailuser)                =
VOV_JOB_DESC(osgroup)                 = guests
VOV_JOB_DESC(outputs)                 =
VOV_JOB_DESC(priority,default)        = 4
VOV_JOB_DESC(priority,exec)           = 4
VOV_JOB_DESC(priority,sched)          = 8
VOV_JOB_DESC(proplist)                =
VOV_JOB_DESC(resources)               = linux
VOV_JOB_DESC(rundir)                  = .
VOV_JOB_DESC(schedule,date)           = 0
VOV_JOB_DESC(setName)                 =
VOV_JOB_DESC(user)                    = mary
VOV_JOB_DESC(wait)                    = 0
VOV_JOB_DESC(wait,options)            =
VOV_JOB_DESC(wrapper)                 = vw
VOV_JOB_DESC(xdur)                    = 30
```

Use Jobclasses

A jobclass allows multiple job parameters to be set in a single object that can be requested at submission time.

For example, there may be a job that requires 3 different licenses, 4GB of RAM, and 4 cores. Instead of requesting all 3 licenses, a jobclass can be created that is called with the -C submission option to the `nc run` command. Jobclasses are often used to emulate queues that are found in other batch processing systems.

Note: A jobclass can only be created by an Accelerator administrator.

Find Jobclasses

To list the available classes from the command line, use the `jobclass` subcommand of the `nc` command.

For example:

```
% nc jobclass
  1 short
  2 interactive
```

The `jobclass` subcommand accepts the repeatable option -l. The first option includes the description, and the second option shows the values to which VOV_JOB_DESC slots will be set.

In addition, Accelerator provides the Jobclass page. This page shows a table of the jobclass, with links to the definitions of each class, and to the sets containing the jobs in that class. It also shows the pass/fail status as a bar graph.

nc jobclass

List classes defined for job submission

```
vnc: Usage Message
  NC JOBCLASS:
          List classes defined for job submission
  USAGE:
          % nc jobclass [OPTIONS]
  OPTIONS:
          -h                     -- This help
          -l                     -- Long format (with description)
          -ll                    -- Longer format.
          -v                     -- Increase verbosity.
  EXAMPLES:
          % nc jobclass
          % nc jobclass -l
          % nc jobclass -ll
```

Submit Jobs Using Jobclasses

To submit a job in a given class, use the option -C of `nc run`.

```
% nc run -C short sleep 10
```

Jobs in a class are automatically added to a set named after the class, for example `Class:interactive`.

The options to `nc run` are parsed sequentially, so it is possible to do a command line override of the parameters set in the jobclass. For example, the following commands behave differently:

```
% nc run -C verilog -e DEFAULT -- run_sim chip
% nc run -e DEFAULT -C verilog -- run_sim chip
```

In the first invocation, the option -e overrides the specifications for the environment to be used for the job. In the second invocation, the environment is determined by the definition of the *verilog* jobclass.

Resources That Change Over Time

`TIMEVAR` should be defined in the `$SWD/config/timevars.tcl` file.

The procedure `TIMEVAR` takes two arguments: a label and a list. The list is similar to a *switch* context, an even number of elements. In each pair, the first element is a time condition and the second element is an executable script.

The time condition can be one of the following:

- HH:MM-HH:MM
- Sun, Mon, Tue, Wed, Thu, Fri or Sat

The example below utilizes `TIMEVAR` to control the level of the `hsim` simulation resource. Between 2:00am and 6:00am on weekdays, the available resources are restricted. In this example, the restriction is an allowance that allows nightly backups to finish. Any other time, the number of allowed resources is doubled.

```
# Fragment of the definition of the jobclass 'hsim'
proc initJobClass {} {
    TIMEVAR hsim {
        Sat,Sun {
            vtk_resourcemap_set_limit  Limit:u_hsim_@USER@ 20
        }
        02:00-06:00 {
            vtk_resourcemap_set_limit  Limit:u_hsim_@USER@ 10
        }
        default {
            vtk_resourcemap_set_limit Limit:u_hsim_@USER@ 20
       }
    }
}
```

A new liveness task reads the `timevars.tcl` file and processes the `TIMEVAR` procedure. The task script is `live_execute_timevars.tcl`, and it is automatically enabled when you start Accelerator on version 2021.2.0. Other products can enable this task, if desired, by manually copying the task script from `$VOVDIR/etc/liveness` into `$SWD/tasks`.

Jobclass Examples

Short Jobs

In this example, a jobclass is set up for short jobs. All jobs have the strict CPU limit of 10 seconds.

```
set classDescription "Jobs taking less than 10 seconds"
set VOV_JOB_DESC(env)                     "BASE+D(VOV_LIMIT_cputime=10)"
set VOV_JOB_DESC(priority,sched)          8
set VOV_JOB_DESC(resources)               "unix"
set VOV_JOB_DESC(xdur)                    10

proc initJobClass {} {
  # No actions needed to initialize this job class.
}
```

Night Jobs

In this example, a jobclass is set up for jobs to run at night or on weekends. For each users, the limit is set for 10 jobs per night.

```
set classDescription "Jobs to run at night or during weekends"
set VOV_JOB_DESC(env)              "SNAPSHOT"
set VOV_JOB_DESC(priority,sched)  2
set VOV_JOB_DESC(resources)       "Queue:night Limit:night_@USER@"

proc initJobClass {} {
   vtk_resourcemap_set Limit:night_@USER@ 10

   TIMEVAR night {
      Sat,Sun {
         vtk_resourcemap_set Queue:night unlimited
      }
      20:00-24:00,00:00-07:30 {
         vtk_resourcemap_set Queue:night unlimited
      }
      default {
         vtk_resourcemap_set Queue:night 0
      }
   }
}
```

Preemption

Preemption is the process of reserving or revoking resources from other jobs in order to help "important jobs" finish quickly.

It is normally triggered by queued jobs, but can also be triggered by a change in a resource or by a manual request from the user. Preemption may require killing or suspending running jobs, but it can also be "gentle," only generating events or reserve resources such as taskers or licenses.

The queued job that triggers preemption is called the *preempting* job, while the job or jobs from which the resources are revoked are called the *preempted* jobs.

Preemption can be used with licenses that are constantly used by background regression jobs. This way, engineers who need licenses for interactive jobs do not have to wait for a regression job to finish.

Preemption Directory and Files

Summary information for preemption:

Working directory	`vnc.swd/vovpreemptd`
Config file	`vnc.swd/vovpreemptd/config.tcl`
Info file	`vnc.swd/vovpreemptd/info.tcl`
Aux directory	`vnc.swd/preemption`

Monitoring Preemption Behavior

To monitor the behavior of the preemption mechanism, view the Preemption Status page.

You may also find it helpful to run separate Accelerator GUIs, one for the set of preemptable jobs, one for the set of preempting jobs, etc. This setup enables observing jobs that enter the system, jobs that are preempted, and other jobs that are running with the new resources.

Set Up the Optional Preemption Ruler Compiler Daemon

The Accelerator preemption mechanism is activated by default. If the preemption rules are stored within the `config.tcl` file, then `vovpreemptd` will need to be started in order to read and monitor the preemption rules. Follow these steps to enable and configure the preemption daemon.

1. Create the directories `preemption` and `vovpreemptd` inside the server configuration directory (.swd). The `vovpreemptd` directory is the run directory for the preemption daemon.
2. Open up permissions for `preemption`, because auxiliary jobs will be run in this directory.

```
% vovproject enable vnc
% cd  `vovserverdir -p .`
% mkdir vovpreemptd
% mkdir preemption
% chmod a+rwx preemption
```

3. Copy the configuration file template, `$VOVDIR/etc/config/vovpreemptd/config.tcl` into the newly-created `preemption` directory.
4. Edit the file to define the preemption rules to be used, using the examples below as a guide.
5. Start the `vovpreemptd` daemon process using `nc cmd vovdaemonmgr start vovpreemptd`, or manually via:

```
% vovproject enable vnc
% cd `vovserverdir -p vovpreemptd`
% vovpreemptd >& vovpreemptd.log &
```

Tip: Set up an autostart script to automatically start `vovpreemptd` when the vovserver is started.

```
% cd `vovserverdir -p autostart`
% cp $VOVDIR/etc/autostart/vovpreemptd.csh .
```

There are two procedures that support the preemption rules, which can be used in `vovpreemptd`'s `config.tcl` file.

- VovPreemptRule defines which jobs can preempt other jobs.
- VovPreemptMethod defines the methods used to revoke specific resources.

File: `$VOVDIR/etc/config/vovpreemptd/config.tcl`

```
#
# Example of a config.tcl file for vovpreemptd.
#

#
# Set to 0 (zero) if you do not have the old fashion policies.
# Obsolete in 2013.09
#
set vovpreempt(doPolicies)       0
set vovpreempt(maxPreemptPerJob) 6

#
# The max time we wait for a jobcontrol action to have effect.
# On slow networks (e.g. slow LDAP), you may want to increase
# this to 20 or more seconds.
# Obsolete in 2013.09
#
```

```
set vovpreempt(timeout,safejobcontrol) 8

#
# This rule fires when there are jobs in the
# queue that have priority >= 8 and that have been
# waiting in the queue for more than 1 minute.
# The preemptable jobs are those in the same jobclass
# as the preemting job and have priority less than 4.
#
VovPreemptRule -rulename GenericPriorityWithinClass \
    -preempting  "priority>=8"  -bucketage 1m \
    -preemptable "jobclass==@JOBCLASS@ priority<4" \
    -method      AUTOMATIC

#
# Example of an ownership contract (Obsolete)
#
VovPreemptDefineOwnershipContract -contractname "SampleOwnership" \
    -ownertable {
 /sample/urgent      0   40
 /sample/normal      0   20
 /sample/regression  0   10
    }
```

Preemption Rules

Preemption is controlled by persistent objects called *preemption rules*, which collectively define:

- The resources that can be preempted
- Conditions under which preemption should be triggered
- Methods used to revoke those resources

There are two ways to manage preemption rules:

1. Create and edit the rules with Accelerator's web interface. The rules are stored within the vovserver and saved to disk from time to time. The rules are automatically loaded when Accelerator is restarted.
2. Define the rules in the Tcl syntax configuration file at `vnc.swd/vovpreemptd/config.tcl`. When the `vovpreemptd` daemon starts, it reads the `config.tcl` file containing the preemption rule information. The daemon monitors this file and reads it again upon changes. The daemon also creates the `info.tcl` file which contains information about the daemon and serves as a lock file to prevent two instances of the daemon from running. The daemon tracks the modification time of the `info.tcl` file, and will exit if the time is changed, for example by another instance of `vovpreemptd`.

Preemption rules define the conditions under which preemption is to be performed. These rules are either defined using the `VovPreemptRule` command (defined below) and/or via Accelerator's web page interface.

The preemption rules are grouped into different pools.

- Only one rule in a pool fires for a given iteration; rules are considered in order (as defined with the -order <N> option). This order of executing rules can use used to set up *escalation*. For example, if the current rule has not fired, then consider the next rule. The first rule could be for a small set of preemptable jobs, the second rule could be for a much larger set of preemptable jobs.
- Multiple pools allow different preemption strategies to be considered in parallel; one pool could be for Design Verification jobs, while another pool could be for spice simulation jobs.

By default, all rules are added to the pool called *mainpool*. For multiple rules to fire during each preemption cycle, the rules must be organized into different pools.

Every preemption rule must have a unique name within it's preemption pool specified with the -rulename NAME option.

Note: If the same name is used for multiple rules, the last definition prevails.

Preemption Conditions

The preemption condition can be defined in one of the following ways.

- There is a bucket of jobs that matches a given selection rule (use the `-preempting` and `-bucketage` options)
- There is a bucket of jobs that is waiting for at least one of a list of resources (use the -waitingfor and -bucketage options)
- There is a resource map that is controlled by MultiQueue and MultiQueue is currently requesting a drastic reduction (at least 10% of the current "in-use" S count) in the amount allocated to this queue (see option `-multiqueueres`). The 10% threshold can be controlled by means of the option `-mqthresh`.

The weakest types of preemption are the `RESERVE_RESOURCES` and `RESERVE_TASKERS` methods. `RESERVE_RESOURCES` simply reserves some resources for the job at the top of the bucket and `RESERVE_TASKERS` simply reserves a tasker for the job at the top of the bucket. It is the intended behaviour that while the resources or `taskers` are reserved, some other jobs will terminate and enable the job at the top of the bucket to be dispatched. The reservation is controlled by the options `-reservetime`, `-reservefor`, and `-reservenum`.

If the preemption type is not `RESERVE_RESOURCES` or `RESERVE_TASKERS`, then the system looks for jobs that can be preempted, i.e., that can be either killed or suspended.

The strongest type of preemption is `FREE_TASKERS` which looks at ways to preempt all jobs currently running on a tasker.

Search for Preemptable Jobs

In this search for preemptable jobs:

- Exclude jobs that have the `preemptable` flag set to zero;
- Exclude jobs that are "system" jobs (like job resumers, zip jobs, ...)
- Exclude jobs that are labeled as *top job*, which are jobs that have caused a preemption in the past, because they were at the top job in a preempting bucket. The jobs are left alone for at least 10 minutes, a time interval that can be controlled with the option `-donotdisturb` <timeSpec>.

Searching for preemptable jobs is done in the following order:

- Look for jobs that can be killed, or more precisely *withdrawn*, and resubmitted, which are the jobs that satisfy the `-preemptable` selection rule and that are younger than `-killage`. These jobs also must be useful in the sense that they hold some resources requested by the preempting job. Preemption will not kill jobs that are not considered useful for the preempting job.
- If no job can be killed, then look for jobs that satisfy the `-preemptable` selection rule and are also useful. If any such job is found, preemption is attempted using the method specified by the `-method` option. Some jobs are resilient to some preemption methods, so care is applied to validate that the method has been effective.
- If the preempted job is successfully suspended, then a **resumer job** associated with the suspended job is created. The resumer job is an invocation of the script `vovjobresumer`. The resumer job inherits the grabbed resources from the suspended job, meaning that it will be executed only when all resources grabbed from the suspended job become available. The resource list of the resumer job can also be augmented with the option `-resumeres`.

@KEYWORD* Expansion

A few of the preemption options supported *KEYWORD* expansion against a *preempting* or *preemtped* job, following the structure set by the option itself. The following rules apply:

`-resumeres`

The default for the resumer job resources (if `-resumeres` is NOT specified at all) is to use the preempted job's STOLENRESOURCES field value. Anything entered into the `-resumeres` option will be added to this default. One sensible example might be to add @SOLUTION@. This would cause the scheduler to require both the SW and HW resources of the preempted job to be available in order for the resumer job to be dispatched to a tasker for execution.

`-preemptable`

The `-preemptable` option expects job metadata and expansion is performed using the *preempting* job instead of the *preempted* job. The purpose of this option is to narrow down the list of jobs to be considered for preemption by attempting to match it up to the *preempting* job in some way. Examples would be: "JOBCLASS==@JOBCLASS@", "PRIORITY<@PRIORITY@", "FSGROUP==@FSGROUP@", and "JOBPROJ==@JOBPROJ@". Values for these fields can also be pre-determined instead of using keyword expansion.

`-preempttaskerspec`
: This option expects tasker-specific information and is used for the FREE_TASKERS and RESERVE_TASKERS rule types exclusively. This can be a predetermined notion of taskers/hosts, such as "TASKERLIST:taskerlist1", "TASKERNAME=tasker1", and "HOST=host1", or a dynamically-populated notion of a tasker/host, such as "TASKERNAME=@TASKERNAME@" or "HOST=@HOST@", which would be dynamically-populated from the *preempting* job.

`-reservefor`
: Used with the RESERVE_RESOURCES and RESERVE_TASKERS rule types exclusively. This option expects reservation targets that will be used to reserve a resource/tasker upon firing of the preemption rule and expansion is performed using the *preempting* job. Valid targets are: USER, GROUP, JOBCLASS, JOBPROJ, and OSGROUP. Examples would be: "USER @USER@", "JOBCLASS @JOBCLASS@", and so on for the other allowed targets. Values for these fields can also be pre-determined instead of using keyword expansion, such as "USER mary" and "JOBCLASS jobclass1".

Preemption Rule Types

Preemption was previously defined as the process of revoking resources from a running job or reserving resources in order to start a 'more urgent' queued job that needs specific resources that are not currently available. Consequently, rules are divided into specific **types** based on how the resources are to be made available to such 'more urgent' job.

The rule type is specified via the -ruletype option in `VovPreemptRule`, and the default rule type is "GENERIC".

Some of options in `VovPreemptRule` are only meaningful for certain rule types. The following options apply to all rule types.

- `-pool`
- `-rulename`
- `-ruletype`
- `-order`
- `-debug`
- `-enabled`
- `-fireonce`
- `-preempting`
- `-bucketage`
- `-waitingfor`

RESERVE_TASKERS

The rules with type `RESERVE_TASKERS` are the simplest and least intrusive. Such rules when fired add a reservation of some specified tasker(s) for some specified period of time. When jobs terminate on a reserved tasker, those open slots are reserved to the jobs in the preempting bucket.

Note: A preempt rule is recomputed every few seconds; because of this, a short reserve time, such as 10 seconds or so, is typically sufficient.

The options in `VovPreemptRule` required for specifying this rule type are:

- `-reservenum`
- `-reservetasker`
- `-reservefor`
- `-reservetime`
- `-preempttaskerspec`

Since no jobs are preempted, options such as `-method` and `-preemptable` are not needed and will be ignored.

For example, the following rule reserves a number of taskers for one minute for any job in the `hsim_critical` jobclass:

```
#
# Reserve some machines for one minute if there is a critical hsim job.
#
VovPreemptRule -rulename "ReserveOnlyHsimHw" \
   -preempting "jobclass==hsim_critical"     \
   -ruletype RESERVE_TASKERS                  \
   -reservetasker "taskerlist:dram4"        \
   -reservefor   "JOBCLASS hsim_critical"    \
   -reservenum   1  \
   -reservetime  "1m"
```

RESERVE_RESOURCES

The preemption rules with type RESERVE_RESOURCES are similar to the RESERVE_TASKERS type; however, instead of reserving taskers, these rules reserve resources for some specific reservation period. The resources reserved are the resources that the preempting job is waiting for and will be reserved for the preempting job for the time specified via the `-reservetime` option. Since no jobs are preempted, the options `-method` and `-preemptable` are ignored. The following is meaningful options for RESERVE_RESOURCES rule type.

- `-reservetime`

In the following example, licenses are reserved for a high priority job of class LargeJob that has been waiting for more than 5 minutes.

```
VovPreemptRule       \
    -pool     "mainpool"     \
    -rulename "ReserveLicenseLargeJobHighPriority"  \
    -ruletype "RESERVE_RESOURCES"    \
    -preempting "Priority>8 JOBCLASS=LargeJob"   \
    -bucketage "5m"      \
    -waitingfor "License:*"     \
    -reservetime 2m
```

MULTIQUEUE

Options for this type are:

- `-multiqueueres`
- `-mqthresh 0.25`
- `-donotdisturb`
- `-preemptable`
- `-killage`
- `-method`
- `-skipresumedjob`

- `-resumeres`
- `-numjobs`
- `-maxattempts`
- `-sortjobsby`

In the following example, if the difference between the multiqueue allocation of License:hsim and actual is greater than 25%, then the jobs using that resource are preempted using the automatic method.

```
VovPreemptRule -rulename "mqPreemptHsim" \
    -ruletype MULTIQUEUE                 \
    -multiqueueres License:hsim          \
    -mqthresh 0.25                       \
    -pool multiqueue                     \
    -method AUTOMATIC
```

GENERIC

The preemption type `GENERIC` is the most common. It is designed to find running jobs that can be preempted to provide for resources in order to dispatch the preempting job.

Options for this type are:

- `-donotdisturb`
- `-preemptable`
- `-killage`
- `-method`
- `-skipresumedjob`
- `-resumeres`
- `-numjobs`
- `-maxattempts`
- `-sortjobsby`

For reference, it may be best to review the options for the vovpreemptrule command for more detailed explanation of the options available for the command.

An example rule follows:

```
#
# Preempting rule is activiated when any job with priority greater than
# or equal to 8 AND is waiting for License:hsmi AND has been waiting
# more than 2 minutes.  It will preempt any job with priority less
# than the preempting job AND is using resource License:hsmi.
# The preemptable job will be killed and resubmitted if it has been
# running less than 1 minute.  Otherwise, it will be preempted via the
# AUTOMATIC method.
#
VovPreemptRule -rulename "priority"    \
    -ruletype GENERIC                  \
    -preempting "PRIORITY>=8"          \
    -waitingfor "License:hsim"         \
    -bucketage  2m                     \
    -preemptable "PRIORITY<@PRIORITY@" \
    -killage    1m                     \
    -method SIGTSTP
```

FAST_FAIRSHARE

`FAST_FAIRSHARE` preemption is intended to help speed up FairShare. The rule type is mainly used by the `NC` web page preemption rule entry page to pre-enter the interesting FairShare related fields for the preempting and preemptable conditions in the selection rules. Internally, it is processed exactly the same as the `GENERIC` rule type.

FREE_RESOURCES

The FREE_RESOURCES preemption rule type is similar in spirit to the FREE_TASKERS rule type in that it can preempt all jobs using a specific set of reserved SW resources to allow the preempting job to run. This type of rule is intended to support hardware emulation jobs and is only supported for the Hero product type. Use extra care when selecting appropriate parameters to ensure that they are consistent with each other.

Options for this type are:

`-method`	Specifies the method used to preempt the preemptable jobs.
`-preemptable`	Specifies which jobs can be preempted by this rule.
`-preempting`	Specifies which jobs can trigger a preemption.
`-reservation`	Reservation id that specifies a collection of emulator leaf resources and a time interval. This preemption rule will only trigger a preemption while the reservation is active.
`-reservetime`	When a preemption is triggered, a temporary reservation is made for the preempting job that will prevent other jobs starting on the leaf resources for the duration specified by the `-reservetime` option. As soon as the preempting job starts the temporary reservation is removed.

The caller must ensure that the reservation target, `-preempting` and `-preemptable` parameters are consistent.

The preemption rule will trigger when the following are true:

1. The reservation is active (that is, within it's start/end time)
2. There is a job matching the `-preempting` selection rule waiting to run
3. The only jobs currently using the reserved resources are those matching the `-preempting` and `-preeemptable` selection rule
4. All the jobs currently using the reserved resources that match the `-preeemptable` selection rule have the preemptable flag set.

When the rule triggers, the specified method will be be used to perform the preemption. Note that the `vovsh` executable must be excluded by the method when using `hero_adapter`; see the example below for an illustration.

Care should be taken to ensure the preemption signal cascade allows the emulation job to clean up before the `hero_adapter` wrapper terminates. This may take several minutes, the `-reservetime` should account for the time taken to preempt the jobs.

Some EDA tools start child processes that are not terminated when the main tool process exits and this can result in unnecessary CPU load. The `tasker.childProcessCleanup` configuration parameter be used to address this issue.

Here is an example rule:

```
# This example assumes that some resources have been reserved for
# jobs whose job project is one of interactiveProj, regressionProj or
# otherProj, for example, using the command
# nc -q hero cmd hero_reserve 'HERO:LEAF_myemulname_(0..8).(0..7)' <start time> <end
 time> -jobproj interactiveProj,regressionProj,otherProj
# and the rule id is assumed to # be 00001234 below for the -reservation
# parameter.
#
# The intent is that if there is an interactive job waiting to run, and
# the only jobs using the reserved resources are interactive & regression
# jobs (whose preemptable flag is set) then all regression jobs using the
# reserved resources will be preempted.  If any other running job is using
# the reserved resources, then a preemption will not be triggered.
#
VovPreemptRule \
    -pool "mainpool" \
    -rulename "emulatorFree" \
    -ruletype FREE_RESOURCES \
    -preempting "jobproj==interactiveProj" \
    -preemptable "jobproj==regressionProj" \
    -reservation 00001234 \
    -reservetime 60s \
    -method "0:*:TERM,,vovsh,0 4:WAIT:NOP 30:WITHDRAWN:RESUBMIT"
```

FREE_TASKERS

This is one of the strongest preemption types, because it can preempt all jobs on a tasker at the same time to make space for the preempting job. This type preempts necessary number of taskers and jobs on those taskers enough to run jobs in the preempting bucket. Also the number of taskers to get preempted does not exceed `-preempttaskernum`.

Options for this type are:

- `-donotdisturb`
- `-preemptable`
- `-preempttaskerspec`
- `-preempttaskernum`
- `-method`
- `-skipresumedjob`
- `-resumeres`

In the following example, high priority jobs in the jobclass "design" requiring 4 or more cores are allowed to preempt groups of "regression" jobs. With a bucketage of 10 seconds, this rule fires about once every 10 seconds for each bucket.

```
VovPreemptRule     \
    -pool     "mainpool"   \
    -rulename "taskerFree"   \
    -ruletype "FREE_TASKERS"   \
    -preempttaskerspec "TASKERLIST:default"  \
    -waitingfor HW    \
    -bucketage  10 \
    -preempting "JOBCLASS==design PRIORITY>=8 REQCORES>=4"   \
    -preemptable "JOBCLASS==regression" \
```

Preemption Timing

-numjobs N

The number of jobs preempted concurrently is determined by the expression:

```
max(N, min(max(M/2, 1), 80))
```

Where:
N = `numjobs` value
M = number of jobs queueing in the bucket

-maxattempts N

The maximum number of attempts to match the rule for a preempting job.
Setting this to zero (0) disables the check, meaning that the rule can be matched an unlimited number of times which is useful for example < for RESERVE_* type rules.
Default value is 10

The `-maxattempts` limits the number of times the preemption rule will be applied to the top job in a preempting bucket **after** no `preemptable` targets are found.

The default preemption cycle length is 3s. Since this is short it may appear that more than one job is being preempted during a given cycle. The `preemptionPeriod` parameter can be set in `policy.tcl` to a longer period to make the number of jobs preempted more apparent. For example:

```
set config(preemptionPeriod) 10s
```

The number of jobs preempted per cycle is also limited to a fraction the size of the preempting bucket.

For example, consider a situation with the following characteristics:

- `preemptionPeriod` of 10s
- SIGTSTP method
- a central resource with `-total` 4
- 4 preemptable jobs that consume a single resource and 10 preempting jobs that consume 4 resources each
- `-numjobs` 1
- `-maxattempts` 3

The preemptable jobs are running before the preempting jobs are added. When the preempting job runs, it runs indefinitely (the others were added just to have a sufficiently large preempting bucket to test `-numjobs`). Initially no preempting job is running and the rule triggers for the top job 000001112 in the preempting bucket. Since it needs 4 resources, it runs 4 cycles preempting one job at a time allowing job 000001112 to execute. Subsequently the rule fires for job 000001117 but there are no suitable preemptable jobs available, so after the third cycle (`-maxattempts` 3) it will no longer apply this rule to job 000001117.

```
Rule triggers for job 000001112 in bucket 000001114 (jobproj==urgent_job_53244).
GENERIC PreemptRule Rule_53244 trying to preempt up to 1 jobs.  4 preemptable targets
 found
Rule triggers for job 000001112 in bucket 000001114 (jobproj==urgent_job_53244).
GENERIC PreemptRule Rule_53244 trying to preempt up to 1 jobs.  3 preemptable targets
 found
Rule triggers for job 000001112 in bucket 000001114 (jobproj==urgent_job_53244).
GENERIC PreemptRule Rule_53244 trying to preempt up to 1 jobs.  2 preemptable targets
 found
Rule triggers for job 000001112 in bucket 000001114 (jobproj==urgent_job_53244).
```

```
GENERIC PreemptRule Rule_53244 trying to preempt up to 1 jobs.  1 preemptable targets
 found
Rule triggers for job 000001117 in bucket 000001114 (jobproj==urgent_job_53244).
GENERIC PreemptRule Rule_53244 trying to preempt up to 1 jobs.  0 preemptable targets
 found
Rule triggers for job 000001117 in bucket 000001114 (jobproj==urgent_job_53244).
GENERIC PreemptRule Rule_53244 trying to preempt up to 1 jobs.  0 preemptable targets
 found
Rule triggers for job 000001117 in bucket 000001114 (jobproj==urgent_job_53244).
GENERIC PreemptRule Rule_53244 trying to preempt up to 1 jobs.  0 preemptable targets
 found
Permanently skip this preemption rule for top job 000001117 since it exceeds maximum
 attempts of 3
```

Command Line Interface for Preemption Rules

Here are some useful commands to manage preemption rules.

```
% vovshow -preemptrules
002772887 test          testFreeTaskerRule           KILL+RESUBMIT    101
002774006 Micron        mic_pri                     KILL+RESUBMIT    102
002774004 Micron        mic_mq                      KILL+RESUBMIT    101
002775275 mainpool      FormalRegressions           SUSPEND          101
002774085 mainpool      testStealResource           AUTOMATIC        102
002775412 mainpool      PreemptAth                  AUTOMATIC        103
002774511 mainpool      byPriority                  SUSPEND          101
002777727 RegrTestPool  RegrTestPriority1523033980  KILL+RESUBMIT     50
002778828 RegrTestPool  RegrTestThomas              0:*:EXT,KILL
 10:WITHDRAWN:RESUBMIT   50
002777743 RegrTestPool  ReserveTaskersForTest        AUTOMATIC        101
002774829 mainpool      HelpStiffJobs               AUTOMATIC        101
002778200 mainpool      pRule4613                   SUSPEND          101
002778064 mainpool      rr41523034013               AUTOMATIC         50
002775429 HERO          Test_Priority_Same_User     SUSPEND          101
002777656 TESTPOOL      TestMethodnormal            AUTOMATIC        103
002778839 RegrTestPoolMQ  RegrTestMQ1523034523      SUSPEND           55
% vovforget -preemptrules
```

If you know the VovId of a preemption rule, you can use it in these commands:

```
% vovshow ID_OF_PREEMPT_RULE
...
% vovforget ID_OF_PREEMPT_RULE
...
```

Preemption methods are created only in `policy.tcl`:

```
% vovshow -preemptmethods
 1 JOBHANDLER_VOVSH     0:*:EXT,SIGTSTP,vovsh 5:SUSPENDED:NOLMREMOVE
 2 SIGTSTP+LMREMOVE     *:RETRACING:SIGTSTP 5:WAIT:SUSPEND 10:SUSPENDED:LMREMOVE
 20:LMREMOVED:DONE
 3 SUSPEND              *:*:SUSPEND
 4 SIGTSTP+SUSPEND      *:RETRACING:SIGTSTP 5:WAIT:SUSPEND
 5 SIGTSTP              *:*:TSTP
 6 KILL+RESUBMIT        0:*:KILL 3:WAIT:NOP 30:WITHDRAWN:RESUBMIT
 7 LMREMOVE             *:*:SUSPEND 10:SUSPENDED:LMREMOVE 20:LMREMOVED:DONE
 8 AUTOMATIC            *:*:*
 9 JOBHANDLER           0:*:EXT,SIGTSTP,tclsh* 5:SUSPENDED:NOLMREMOVE
```

Tcl Interface to Preemption Rules

To dump the rules to a file, use the command `VovDumpPreemptionRules`:

```
# This is Tcl.
VovDumpPreemptionRules NameOfFile.tcl
```

At the low level, you can use these procedures to manipulate preemption rules:

```
% vovshow -api preempt
vtk_preemptrule_create DESCRIPTION_ARRAY
vtk_preemptrule_modify DESCRIPTION_ARRAY
vtk_preemptrule_forget ID
vtk_preemptrule_delete ID
vtk_preemptrule_find   POOL RULENAME
vtk_preemptrule_get    ID RESULT_ARRAY
vtk_preemptrule_delete_all
vtk_preemptrule_forget_all
```

To preempt a specific job, call:

```
# This is Tcl.
vtk_transition_preempt  jobId [-noop] [-manualresume] [-method METHOD] [-resumeres
 RESLIST]
```

vovpreemptrule

```
Usage: VovPreemptRule -rulename NAME [OPTIONS]

Options:
    -pool               POOLNAME     -- The rule belongs to a pool of rules.
                                        At most one preemption can occur for each pool
                                        in each preemption cycle (default: mainpool)
    -rulename           NAME         -- Required.
    -ruletype           TYPE         -- The type of preemption rule. Allowed values are
                                        GENERIC, FAST_FAIRSHARE, MULTIQUEUE,
 RESERVE_RESOURCES,
                                        RESERVE_TASKERS, and FREE_TASKERS (default:
 GENERIC)
    -order              INTEGER      -- Specify the order of evaluation of rules within
 the same pool.
                                        Rules are evaluated from low to high order.  If
 not specified
                                        the order is assigned automatically based on
 order of declaration.
                                        Typical range is small positives from 0 to 1000,
 but the order
                                        can be any integer.
    -enabled            BOOL         -- To enable and disable the rule.
    -enable             BOOL         -- Same as -enabled (obsolete).
    -debug              BOOL         -- To control debugging flag for this rule.
    -fireonce           BOOL         -- To control the fire-once flag.
    -preempting         SELRULE      -- A selection rule for the top job in a bucket.
    -waitingfor         RESLIST      -- If set, the top job in the bucket must be
 waiting
```

```
                                   for at least one of the
                                   given resources in order to trigger a
preemption.
                                   If the RESLIST contains the string 'HW', then
                                   preemption is triggered if a job waits for a
slot.
   -bucketage         TIMESPEC    -- Only apply the preemption if the bucket age
                                   is greater than the specified value.

   -multiqueueres     RESLIST     -- Trigger preemption if a multiqueue resource
(rank>20)
                                   is imbalanced.
   -mqthresh         THRESHOLD    -- Percent reduction in MQ allocation that triggers
                                   preemption. Default 0.1=10%

   -donotdisturb      TIMESPEC   -- Do not preempt a job that was a top-job (i.e. a
job
                                  that triggered some preemption) for at least the
specified
                                  time (default 10m)
   -preemptable       SELRULE    -- A selection rule for the running jobs
                                  that should be preempted.  Any field of the
                                  form @FIELD@ is replaced by the corresponding
                                  value for the top job in the bucket.

   -preempttaskerspec SPEC      -- If preempting job is waiting for hardware,
                                  preempt taskers that match the given SPEC.
                                  The SPEC may include
"TaskerList:NAMEOFTASKERLIST"
                                  and selection rules for taskers, like
"HOSTNAME=lnx01,lnx02 RANDOM>5000"
                                  Used for FREE_TASKERS rules.
   -preempttaskersnum  N         -- For FREE_TASKERS rules, how many taskers to
preempt for each bucket
                                  that matches the preempting rule. In any case, we
never preempt more
                                  taskers than there are jobs in the bucket.
Default is 1.
                                  Use a higher number if you are preempting many
jobs for better performance.

   -killage           TIMESPEC   -- Jobs younger than this age are simply killed
                                  and resubmitted.  Limited to 7 days max and
default
                                  is 0 which implies that killage is not used.
   -method            METHOD     -- The method to be used to recover license
                                  resources from the job. Allowed values are
                                  SUSPEND, LMREMOVE, RESERVE, AUTOMATIC.
                                  If AUTOMATIC, then each license is removed using
                                  the specific method defined with
VovPreemptMethod.
                                  Default: AUTOMATIC
   -skipresumedjob    TIMESPEC   -- Do not preempt jobs that have been resumed
                                  no more than TIMESPEC ago.
                                  Default: 2m

   -reservetime       TIMESPEC   -- How long resources should be reserved for the
                                  top job when attempting preemption (default 20)
   -reservetype       RESTYPE    -- Deprecated. Use reservefor.
   -reservenum        N          -- How many taskers are to be reserved for
RESERVE_TASKERS rule type.
                                  Default: 1
   -reservetasker   TASKERNAMES -- If a job is waiting for hardware, this is a
 space-separated list
                                   of taskers to reserve for the job.  It is also
 possible to include
                                   a tasker list by using the keyword
 'TASKERList:NAME_OF_TASKER_LIST'.
                                   -preempttaskerspec is used if the field is empty.
   -reservefor       RESSPEC    -- Specify how to reserve a tasker. The RESSPEC is
                                   a space-separated list of KEY VALUE, where KEY is
                                   one of BUCKET USER GROUP JOBCLASS JOBPROJ OSGROUP
 JOBID and VALUE
                                   is a comma-separated list of values (also
 symbolic like @USER@).
                                   VALUE of BUCKET and JOBID entered here is ignored
 and
                                   preempting job ID and bucket ID are used.
                                   Default is BUCKET.

   -resumeres        RESLIST    -- List of resources to append to the resumer job.
                                   RESLIST can contain field references (e.g.
 @HOST@)
                                   which are taken from the preempted job.
   -resumedelay      TIMESPEC   -- Set the minimum delay before executing the
 resumer job,
                                   where TIMESPEC is the span of time between job
                                   suspension and future time when the resumer job
                                   will be considered for scheduling again.
                                   Default: 5s

   -numjobs          N          -- The maximum number of jobs preempted per bucket
 per
                                   preemption cycle.
   -maxattempts      N          -- The maximum number of attempts to match the rule
 for a
                                   preempting job. Setting this to zero (0) disables
 the check
                                   meaning that the rule can be matched an unlimited
 number of times
                                   which is useful for example for RESERVE_* type
 rules.
   -sortjobsby       N          -- Criteria to sort/order potential preemptable
 jobs.
                                   Format is:
                                   <fieldname> [ASC|DESC] [, <fieldname> [ASC|
DESC]]*.
                                   Default is 'PRIORITY ASC, AGE ASC'.
```

Debug VovPreemptRule

To make sure the rules works as intended, it is useful to look at how the preemption algorithms work in detail. Detailed logs will be written into a log file separate from server log as named server_preemption_DATE.log.

Set the server parameter `preemption.log.verbosity` to a number between 0 and 10. For preempt rules of interest, turn on the debug flag through Web UI. To log all preempt rules, set the server parameter `preemption.log.allrules` to 1.

The following shows kinds of messages shown for each verbosity level.

- Taskers preempted, jobs preempted, reservations made on taskers and resources, and durations of preempted jobs.

- Preempt rules that trigger for jobs in each bucket. Reasons why rules get disabled. Time taken to process rules if it is significant.
- Which taskerlist is used. Which tasker is missing.
- Why each tasker is not selected. Reasons may be bad tasker status, HW not compabible, already reserved, or select rule not applicable.
- Report all job status being preempted. Each job preempted with which plan. Reserving critical resource. Skip job after max attempt.
- Report how all preempted jobs are handled. Which jobcontrol method is applied.
- Why preempt rule is not triggered. Why taskers are not chosen (already reserved, invalid reserve spec.).
- Details about choosing preemptable target such as waiting for HW and SW, running jobs that have resources managed by Allocator, jobs that have useful resources, wait reasons, preemptable analysis, missing resources.
- Time taken to process preempt rules.
- Which rule is disabled. Miscellaneous messages.

Preemption Examples

Preemption by Priority

In the following example, all high priority jobs requesting `License:abc` can preempt all low priority jobs. It is implied that the preempted low priority job must provide the resource `License:abc`, otherwise it will not be preempted.

```
# An implementation of the old "preemption by priority" policy.
VovPreemptRule -rulename "SimplePriority" \
      -ruletype    GENERIC  -preempting  "priority>=8"    -waitingfor License:abc \
      -preemptable "priority<4 "
```

The next example generalizes the preemption by priority, within a specific job class. In this case, the preempting job needs to be in the jobclass called `hsim`, and the preemptable jobs are selected only if the have the same jobclass as the preempting job:

```
# A more generic preemption by priority for jobs in the hsim jobclass.
VovPreemptRule -rulename "Priority_hsim" \
      -ruletype    GENERIC  -preempting  "jobclass==hsim priority>=8" \
      -preemptable "jobclass==@JOBCLASS@ priority<@PRIORITY@"
```

Preemption Across Jobclasses

In this example, the jobclass `urgent` can preempt jobs in the jobclass `regression` but only for jobs that belong to the same project, as expressed by `jobproj==@JOBPROJ@`. Also, a relatively long kill age of 10 minutes is allowed; the preemptable jobs that are younger than 10 minutes are withdrawn and resubmitted, while jobs that are older are suspended and later resumed.

```
# A preemption between jobclasses, but within the same project
VovPreemptRule -rulename "Urgent_vs_regression" \
       -preempting  "jobclass==urgent"  -preemptable "jobclass==regression
 jobproj==@JOBPROJ@"  -killage 10m
```

Reserve Resources for Token-based Jobs

In the following example, a *weak preemption* type called `RESERVE_RESOURCES` is used to help dispatch jobs that require multiple tokens in the presence of other jobs that compete for the same tokens. In this case, if there is a job in the class `ultrasim` with priority greater than 4, the system reserves 6 tokens of whatever the job requires for 3 minutes.

```
VovPreemptRule -rulename "UltrasimWeak" \
    -preempting "jobclass==ultrasim priority>4" \
    -ruletype RESERVE_RESOURCES -reservetime 3m -reservenum 6 \
    -pool Ultrasim -order 10
```

Note: There is no mention of the resource required by the jobs in the preempting jobclass, because the resources are computed automatically.

Such a rule (no mention of required resources) may be combined with a stronger rule. For example, such a rule can actively preempt other jobs by the same user, provided that the ultrasim job has been waiting for at least 4 minutes.

```
VovPreemptRule -rulename "UltrasimStronger" \
    -preempting "jobclass==ultrasim priority>4" -bucketage 4m \
    -preemptable "jobclass==spectre user==@USER@ priority<@PRIORITY@" \
    -pool Ultrasim -order 20
```

MultiQueue Preemption

MultiQueue is the older name for the system now called Allocator™. The preemption system still refers to these rules with the old name.

In the following example, assume that the resources `WAN:abc` and `WAN:hsim` are managed by Allocator. You want the preemption daemon to preempt a job that uses those resources when Allocator requires a "substantial reduction" in the number of available resources for this site. In the example, a different rule is created for each resource and each rule is assigned to a different pool. The substantial reduction is defined a 5% change in the allocated resources.

```
foreach mqRes {
    WAN:abc
    WAN:hsim
} {
    VovPreemptRule -rulename "MultiQueueLicense_$mqRes" \
        -multiqueueres $mqRes  -mqthresh 0.05  -pool POOL$mqRes
}
```

Use the -resumeres Option

In many cases, the preemption occurs to make a saturated license available to an important job. Upon suspension of the preempted job, the license becomes available to the preempting job, while the resumer job, which also wants the same license, waits because it typically has lower priority than the preempting job.

In some cases, however, the license that is being preempted is not saturated, and preemption occurs on account of the lack of slots (also known as hardware preemption). In such case, the resumer job could be executed immediately on any available slot, which may be undesirable. The resources specified by the -resumeres option can be added to the resumer job as a way to better control its execution.

Note: When preemption occurs because of hardware, the `-resumeres` option is highly recommended.

Note: See @KEYWORD* Expansion for details on using @KEYWORD@ expansion with `-resumeres`.

For example, assume that jobs in the class *C* can only execute on taskers that offer the resource *R*. A low-priority job in the class C is preempted. The corresponding resumer job requires only the resources grabbed by the preempted job. Since the resumer job can execute on any host and the resources grabbed by the suspended job are not saturated, the resumer job fires immediately, leading to a premature resumption of the suspended job and to the possible overloading of the host on which the resumed job is running. However, if the preemption rule is specified with the option `-resumeres HOST=@HOST@`, then the resumer job is forced to execute only on host called `@HOST@`, which is mapped to the name of the host on which the preempted job was originally running.

All fields in the preempted job can be used in the `-resumeres` argument. The most useful are `@TASKERNAME@` (or `@NAME@` for short), `@TASKERHOST@` (or `@HOST@` for short) and `@HWRAM@`, `@HWPERCENT@`, `@HWSLOTS@`, `@HWCPUS@`, and `@HWSWAP@`, which are computed from the `SOLUTION` property of the preempted job. The open slot goes to the preempting job because of priority, and the resumer job waits for at least one slot to open up on the host of the suspended job.

```
#
# The resumer job must execute on the same host as the
# preempted job.
#
VovPreemptRule -rulename XX  -ruletype    GENERIC  -preempting "jobclass==abc
 priority>=8" -waitingfor "HW"  -preemptable
"jobclass==abc priority<4"  -resumeres "TASKERNAME=@TASKERNAME@ HOST=@HOST@ RAM/
@HWRAM@ SLOTS/@HWSLOTS@"
```

```
#
# The resumer job must execute on the same tasker as the
# preempted job.
VovPreemptRule -rulename YY  -ruletype    GENERIC  -preempting "jobclass==abc
 priority>=8" -waitingfor "HW"  -preemptable
"jobclass==abc priority<4"  -resumeres "TASKERNAME=@TASKERNAME@"
```

Preempt Ultrasim

Ultrasim uses Virtuoso_Multi_mode_Simulation, typically 4 or 6 tokens. To preempt Ultrasim, you can use this example:

```
VovPreemptMethod License:Virtuoso_Multi_mode_Simulation SIGTSTP

VovPreemptRule -rulename ultrasim \
      -ruletype GENERIC \
      -killage    10 \
      -waitingfor "License:Virtuoso_Multi_mode_Simulation" \
      -preempting "priority>8" \
      -preemptable "priority<@PRIORITY@" \
      -method AUTOMATIC
```

Use Preemption to Reserve Taskers

In the following example, if there are scheduled jobs from the FairShare group "`/class/ABCD`", then we reserve 3 taskers called `linux02`, `linux04`, and `linux06` for 1 minute. If, in that 1 minute, the tasker becomes available, the preempting job will have exclusive access to that tasker. Since the rule is recomputed every few seconds, it is sufficient to have short reserve times, such as one minute or even less.

```
VovPreemptRule -rulename reserveHardware \
      -ruletype    RESERVE_TASKERS \
      -preempting "GROUP~/class/ABCD" \
      -reservetime 1m \
      -reservefor  "GROUP @FSGROUP@" \
      -reservetasker "linux02 linux04 linux06"
      -reservenum 2
```

Preemption Methods

A preemption method is a set of actions that are applied to the preemptable job to revoke its resources. Some methods are provided with the Accelerator software. Additional methods can be implemented by writing procedures in Tcl using the Accelerator API calls.

`VovPreemptMethod` can be used in the configuration file to associate a method with a resource to be revoked.

The supplied preemption methods are:

AUTOMATIC	This is the default method and means that the actual preemption method is computed on the basis of the licenses (not the grabbed resources) actually held by the preempted job.
SIGTSTP	The job's process group is signalled with the operating system TSTP signal. Many software tools will first give up their licenses, then self-suspend after receiving this signal. Verify that your tool responds as desired when using this method. This method sends `TSTP` to all processes in the process tree of the job. If the job requires `TSTP` to be sent to only a few processes, as in the case of ModelSim, the `EXT` method needs to be used.
SIGTSTP+LMREMOVE	The same as above but also calls `lmremove` to remove all license checkouts detected for the job being preempted. This is useful for tools that use 3rd-party licenses, such as a piece of IP provided by a 3rd-party vendor, that is used in a simulation.
SUSPEND	The job's process group is signalled with the operating system STOP signal. This causes it to be inactive until it is resumed by the operating system `CONT` signal.
SUSPEND+LMREMOVE	Like `SIGTSTP+LMREMOVE`, but uses the `STOP` signal instead of `TSTP`.
STOP	The job is dequeued or stopped, and rescheduled. Mainly used with jobs that have only been running a short time.
EXT	This method uses an EXTernal script to send a specified list of signals to one or more of the processes in the job. This method is used, for example, to preempt ModelSim jobs.
EVENTS-ONLY	The `EVENTS-ONLY` method does not actually preform any preemption but only issues preemption events. This method can be useful for those users that wish to perform specific preemptions themselves using custom scripts.

Custom preemption methods can be created as well (see the next section on preemption plans). To list all preemption methods, along with their respective plans, use the following command:

```
% nc cmd vovshow -preemptmethods
 1 SIGTSTP+LMREMOVE     *:RETRACING:SIGTSTP 5:WAIT:SUSPEND 10:SUSPENDED:LMREMOVE
 20:LMREMOVED:DONE
 2 SUSPEND              *:*:SUSPEND
 3 SIGTSTP+SUSPEND      *:RETRACING:SIGTSTP 5:WAIT:SUSPEND
 4 SIGTSTP              *:*:TSTP
 5 KILL+RESUBMIT          0:*:KILL 10:WITHDRAWN:RESUBMIT
 6 LMREMOVE               *:*:SUSPEND 10:SUSPENDED:LMREMOVE 20:LMREMOVED:DONE
 7 AUTOMATIC              *:*:*
 8 JOBHANDLER             0:*:EXT,SIGTSTP,tclsh* 5:SUSPENDED:NOLMREMOVE
```

Select the Preemption Method

There are a few methods to specify the preemption method:

- If the job uses `SmartSuspend`, then the property `SSR_STATUS_PATH` exists on the job and the method `SMARTSUSPEND` will be used.
- Set the property `VOVPREEMPT_METHOD` on the job. For example:

  ```
  % nc run -P VOVPREEMPT_METHOD=LMREMOVE -r License:abc -- myscript mychip.x
  ```
- Set the property `VOVPREEMPT_METHOD` on the jobclass. This can be set using the procedure `vtk_jobclass_set_preemption_method`, as in this example:

  ```
  # This could be in vovpreemptd/config.tcl
  vtk_jobclass_set_preemption_method hsim_lo STOP
  ```
- Assign a preemption method to each resource map, using the procedure `VovPreemptMethod` in the file `vnc.swd/vovpreemptd/config.tcl`.

The procedure `VovPreemptMethod` is used to specify which method to use for each resource this is used by the preempted job.

LMREMOVE

While the method `LMREMOVE` works well with a majority of licenses, some licenses are harder to get and this procedure allows specifying how this should be done.

For example, some tools react to the `SIGTSTP` signal, but it can only be sent to a specific process in the process tree. ModelSim is one such tool, which wants the `SIGTSTP` signal to be delivered only to the `vish` process.

```
VovPreemptMethod License:msimhdlsim  EXT -signal TSTP -include vish
```

The following statement is useful for the Cadence tokens. In this example, to recover the resource `License:simtoken`, presumably derived from the FlexNet Publisher feature Virtuoso_MultiMode_Simulator, the `EXT` method (external) and sending the signal `SIGTSTP` are required, but only for the processes either `spectre` or `ultrasim`:

```
VovPreemptMethod License:simtoken EXT -signal TSTP -include "spectre ultrasim"
```

The following example is for `Incisive_Enterprise_Simulator`:

```
VovPreemptMethod License:Incisive_Enterprise_Simulator EXT -signal TSTP -include
 "ncsim ncvlog_main"
```

vovpreemptmethod

Specify which method to use for each resource this is used by the preempted job.

```
Usage: VovPreemptMethod resourcemap methods [OPTIONS]
```

```
WHERE: methods -- A list of one or more of
                  EXT SUSPEND MODELSIM LMREMOVE SIGTSTP JOBHANDLER

OPTIONS:
    -signal <SIGNAME>     -- TSTP, USR1, USR2, ...
    -include patternList  -- Only send signal to processes that match one of
                             the pattern in the list
    -exclude patternList  -- Do not send signal to processes that match one of
                             the pattern in the list
    -process patternList  -- Synonym for -include.

EXAMPLES:
        VovPreemptMethod License:msimhdlsim      EXT
            -signal TSTP -include vish
        VovPreemptMethod License:Virtuoso_Multi_mode_Simulation   EXT
             -signal TSTP -include "spectre ultrasim"
        VovPreemptMethod License:Virtuoso_Multi_mode_Simulation   EXT
             -signal TSTP -include "spectre* *ultra*"
        VovPreemptMethod License:DesignCompiler  LMREMOVE
```

Preemption Plans

Preemption plans are the building blocks of preemption methods and allow the implementation of custom, complex multi-step preemption sequences.

Preemption plans are defined in the `SWD/policy.tcl` file, using the following configuration command:

```
vtk_preemptionplan_set <METHOD-NAME> <PREEMPTION-PLAN>
```

A preemption plan is a space-separated list of instructions. Each instruction consists of 3 colon-separated fields: `TIME`, `STATE`, and `ACTION`

`TIME` can be:

- `*` = any time
- N = N seconds after start of preemption has begun. More specifically, this is the longest time we would wait for the job to reach the required state. If the specified time has elapsed and the job is not in the required state, the preemption plan is considered to have failed.

`STATE` can be:

- `*` = any state
- `SUSPENDED` = the job is known to be in a suspended state
- `RETRACING` = the job is known to be still retracing (orange)
- `RUNNING` = the job is known to be still running (yellow)
- `WITHDRAWN` = the job has been killed by preemption
- `LMREMOVED` = the job has the property PREEMPT_LMREMOVED
- `WAIT` = enter a wait state for the specified number of seconds in the TIME field
- `SSRSUSPENDED` = the job has the property SSR_STATUS (SmartSuspend integration) (OBSOLETE)

`ACTION` can be:

- `NOP` = No operation, mainly used for no-op wait states
- `SUSPEND` = Suspend job (SIGSTOP)
- `SIGTSTP` or `TSTP` = Send `SIGTSTP`
- `SIGUSR1` or `USR1` = Send `SIGUSR1`
- `SIGUSR2` or `USR2` = Send `SIGUSR2`
- `EXT,<SIG>[,includeRx][,excludeRx]` = Send a signal using the EXTernal method
- `LMREMOVE` = Remove licenses using vovlmremove
- `NOLMREMOVE` = Do not call the utility `vovlmremove` on the preempted job. This is essentially the same as `NOP`
- `DONE` = Similar to `NOP` but also signifies the end of preemption the plan
- `RESUBMIT` = Resubmit a withdrawn job (not yet implemented)

The legacy method `MODELSIM` is essentially a one-step preemption plan of the form `*:*:EXT,TSTP,vish`, which means "at any time, in any state, send the signal `TSTP` to the `vish` process.

Note: The preemption will fail if the time has passed and an expected state have not been attained. In the following example, if the job is not suspended at 10 seconds, or if the license is not removed at 20 seconds, the preemption will fail.

```
*:*:SUSPEND 10:SUSPENDED:LMREMOVE  20:LMREMOVED:DONE
```

Examples

Below are some examples of 3-step preemption plans:

```
*:RETRACING:TSTP  10:WAIT:SUSPEND  15:SUSPENDED:DONE
```

1. At the beginning, if the job is RETRACING, send its process tree the TSTP (temporary stop) signal.
2. Wait at least 10 seconds from preemption start and send the SUSPEND signal.
3. Up to 15 seconds from preemption start, if the state is SUSPENDED, the preemption is done. Otherwise, preemption fails.

```
*:*:SUSPEND  10:SUSPENDED:LMREMOVE  20:LMREMOVED:DONE
```

1. At any time, in any state SUSPEND a job.
2. At any time up to 10 seconds from preemption start, if the job is SUSPENDED, remove license(s) from the job.
3. At any time up to 20 seconds from preemption start, if the licenses have been removed, preemption is done. Otherwise, preemption fails.

```
*:*:EXT,HUP,licd  10:*:EXT,TSTP,licd  20:SUSPENDED:LMREMOVE
```

1. At any time, in any state, use the external method to send the HUP (hangup) signal to the `licd` process.
2. At any time up to 10 seconds from preemption start, whether or not the previous step (HUP) was successful, use the external method to send the TSTP signal to the `licd` process.
3. At any time up to 20 seconds from preemption start, if the job is SUSPENDED, remove license(s) from the job.

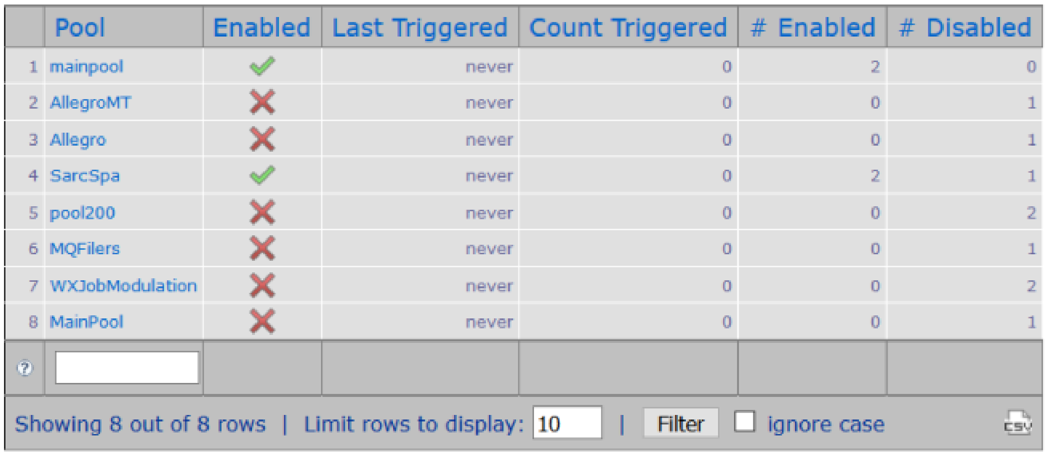
Web-Based Interface for Preemption

The Accelerator web interface provides a form-based method of entering preemption rules and methods, and viewing statistics regarding the activation of previously entered preemption rules.

To access the online preemption information, from the Project Home page of the Accelerator web interface, under Workload, click **Preemption**. By default, the Preemption page opens on the Pools tab.

Pools tab

The Pools tab lists summary of all pools of preemption rules. Click the desired pool names to view the listing of all the preemption rules in that pool.



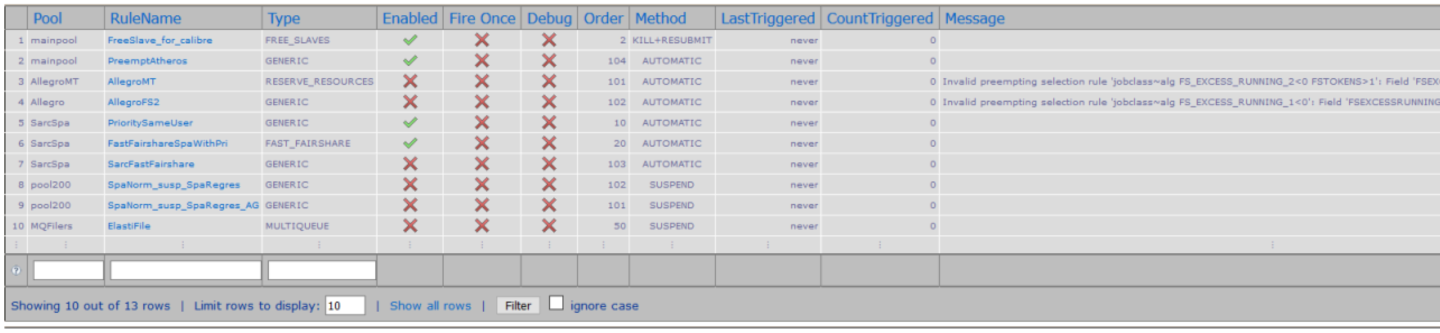
	Pool	Enabled	Last Triggered	Count Triggered	# Enabled	# Disabled
1	mainpool	✓	never	0	2	0
2	AllegroMT	✗	never	0	0	1
3	Allegro	✗	never	0	0	1
4	SarcSpa	✓	never	0	2	1
5	pool200	✗	never	0	0	2
6	MQFilers	✗	never	0	0	1
7	WXJobModulation	✗	never	0	0	2
8	MainPool	✗	never	0	0	1

Showing 8 out of 8 rows | Limit rows to display: 10 | Filter ☐ ignore case

Figure 9:

Preemption Rules tab

The Preemption Rules tab lists the preemption rules that are currently set.



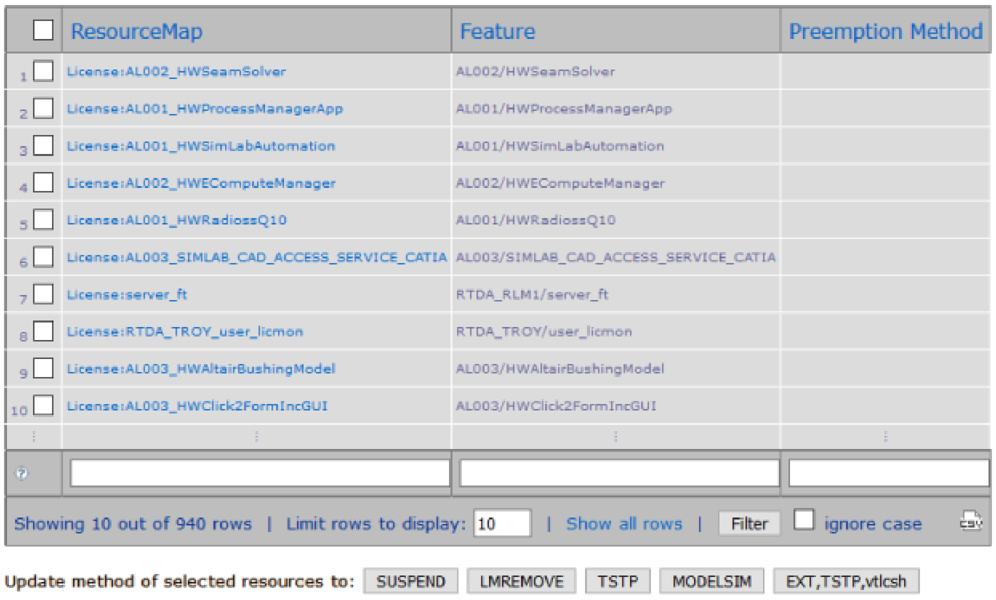
	Pool	RuleName	Type	Enabled	Fire Once	Debug	Order	Method	LastTriggered	CountTriggered	Message
1	mainpool	FreeSlave_for_calibre	FREE_SLAVES	✓	✗	✗	2	KILL+RESUBMIT	never	0	
2	mainpool	PreemptAtheros	GENERIC	✓	✗	✗	104	AUTOMATIC	never	0	
3	AllegroMT	AllegroMT	RESERVE_RESOURCES	✗	✗	✗	101	AUTOMATIC	never	0	Invalid preempting selection rule 'jobclass~alg FS_EXCESS_RUNNING_2<0 FSTOKENS>1': Field 'FSEX
4	Allegro	AllegroFS2	GENERIC	✗	✗	✗	102	AUTOMATIC	never	0	Invalid preempting selection rule 'jobclass~alg FS_EXCESS_RUNNING_1<0': Field 'FSEXCESSRUNNING
5	SarcSpa	PrioritySameUser	GENERIC	✓	✗	✗	10	AUTOMATIC	never	0	
6	SarcSpa	FastFairshareSpaWithPri	FAST_FAIRSHARE	✓	✗	✗	20	AUTOMATIC	never	0	
7	SarcSpa	SarcFastFairshare	GENERIC	✗	✗	✗	103	AUTOMATIC	never	0	
8	pool200	SpaNorm_susp_SpaRegres	GENERIC	✗	✗	✗	102	SUSPEND	never	0	
9	pool200	SpaNorm_susp_SpaRegres_AG	GENERIC	✗	✗	✗	101	SUSPEND	never	0	
10	MQFilers	ElastiFile	MULTIQUEUE	✗	✗	✗	50	SUSPEND	never	0	

Showing 10 out of 13 rows | Limit rows to display: 10 | Show all rows | Filter ☐ ignore case

Figure 10:

Methods for Resources

The Methods for Resources tab summarizes the preemption rules that are used for each resource.



	ResourceMap	Feature	Preemption Method
1	License:AL002_HWSeamSolver	AL002/HWSeamSolver	
2	License:AL001_HWProcessManagerApp	AL001/HWProcessManagerApp	
3	License:AL001_HWSimLabAutomation	AL001/HWSimLabAutomation	
4	License:AL002_HWEComputeManager	AL002/HWEComputeManager	
5	License:AL001_HWRadiossQ10	AL001/HWRadiossQ10	
6	License:AL003_SIMLAB_CAD_ACCESS_SERVICE_CATIA	AL003/SIMLAB_CAD_ACCESS_SERVICE_CATIA	
7	License:server_ft	RTDA_RLM1/server_ft	
8	License:RTDA_TROY_user_licmon	RTDA_TROY/user_licmon	
9	License:AL003_HWAltairBushingModel	AL003/HWAltairBushingModel	
10	License:AL003_HWClick2FormIncGUI	AL003/HWClick2FormIncGUI	

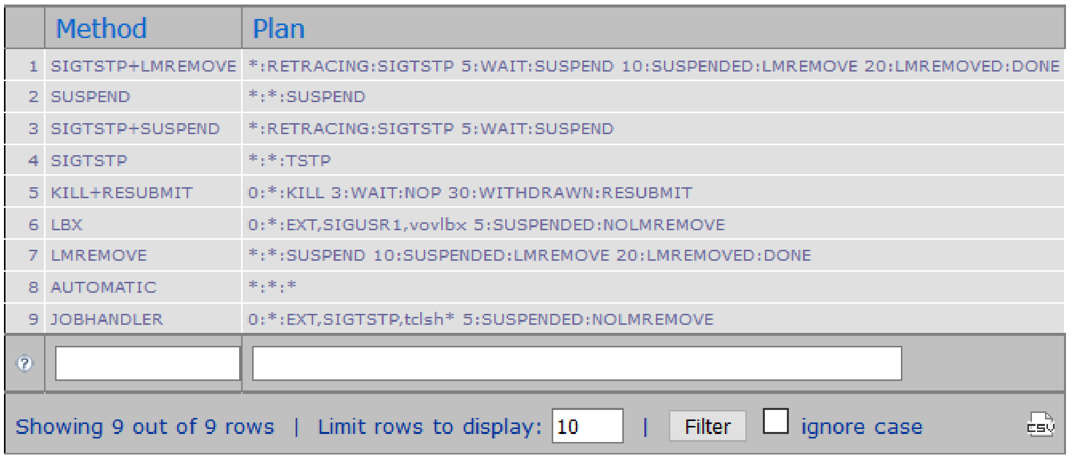
Showing 10 out of 940 rows | Limit rows to display: 10 | Show all rows | Filter ignore case

Update method of selected resources to: SUSPEND LMREMOVE TSTP MODELSIM EXT,TSTP,vtlcsh

Figure 11:

Plans for Methods

The Plans for Methods tab lists the rules of the method plans.



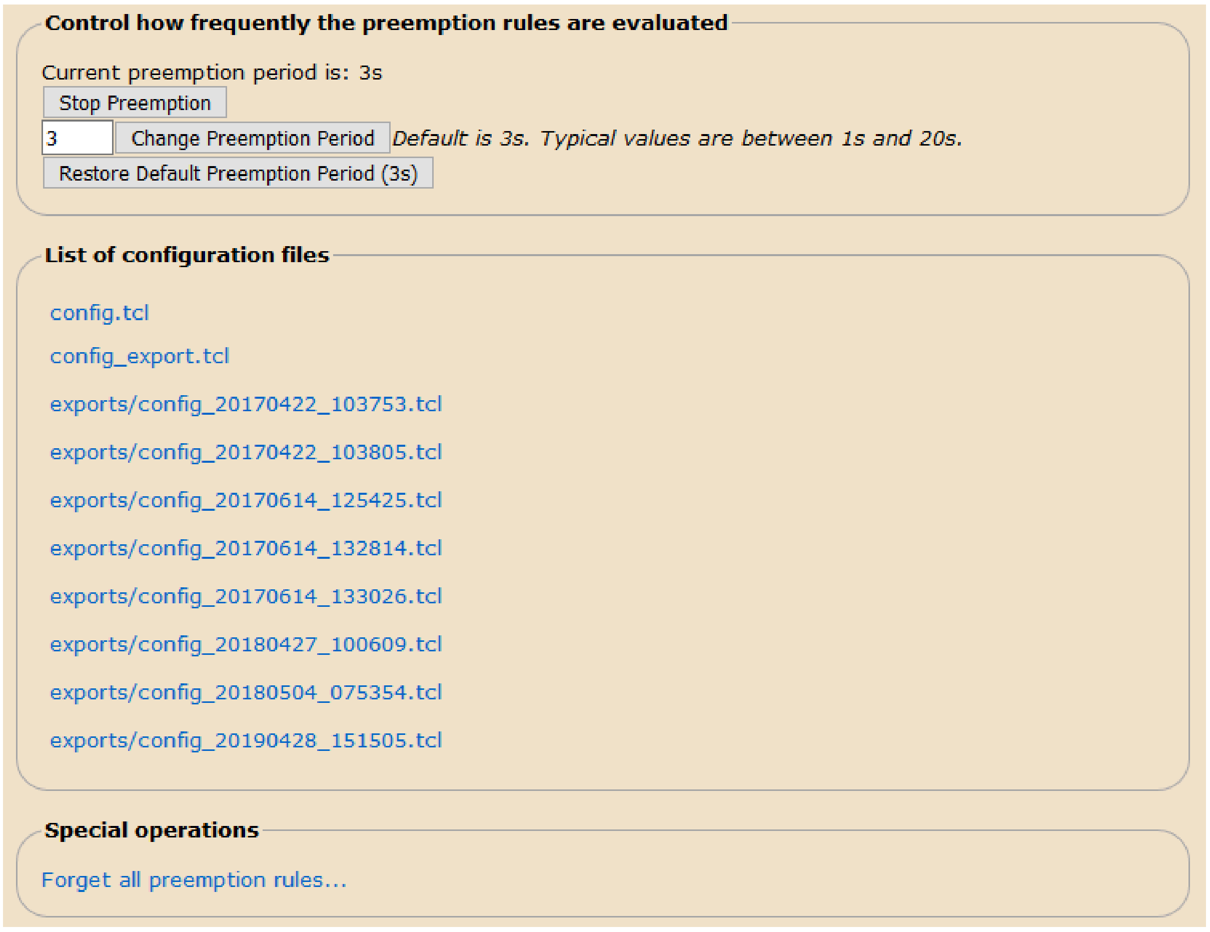
	Method	Plan
1	SIGTSTP+LMREMOVE	*:RETRACING:SIGTSTP 5:WAIT:SUSPEND 10:SUSPENDED:LMREMOVE 20:LMREMOVED:DONE
2	SUSPEND	*:*:SUSPEND
3	SIGTSTP+SUSPEND	*:RETRACING:SIGTSTP 5:WAIT:SUSPEND
4	SIGTSTP	*:*:TSTP
5	KILL+RESUBMIT	0:*:KILL 3:WAIT:NOP 30:WITHDRAWN:RESUBMIT
6	LBX	0:*:EXT,SIGUSR1,vovlbx 5:SUSPENDED:NOLMREMOVE
7	LMREMOVE	*:*:SUSPEND 10:SUSPENDED:LMREMOVE 20:LMREMOVED:DONE
8	AUTOMATIC	*:*:*
9	JOBHANDLER	0:*:EXT,SIGTSTP,tclsh* 5:SUSPENDED:NOLMREMOVE

Showing 9 out of 9 rows | Limit rows to display: 10 | Filter ignore case

Figure 12:

Configuration

The Configuration tab allows exporting preemption rules and creating new preemption rules.


Control how frequently the preemption rules are evaluated
Stop Preemption
Change Preemption Period
Restore Default Preemption Period (3s)
List of configuration files
Special operations
Forget all preemption rules...

Current preemption period is: 3s

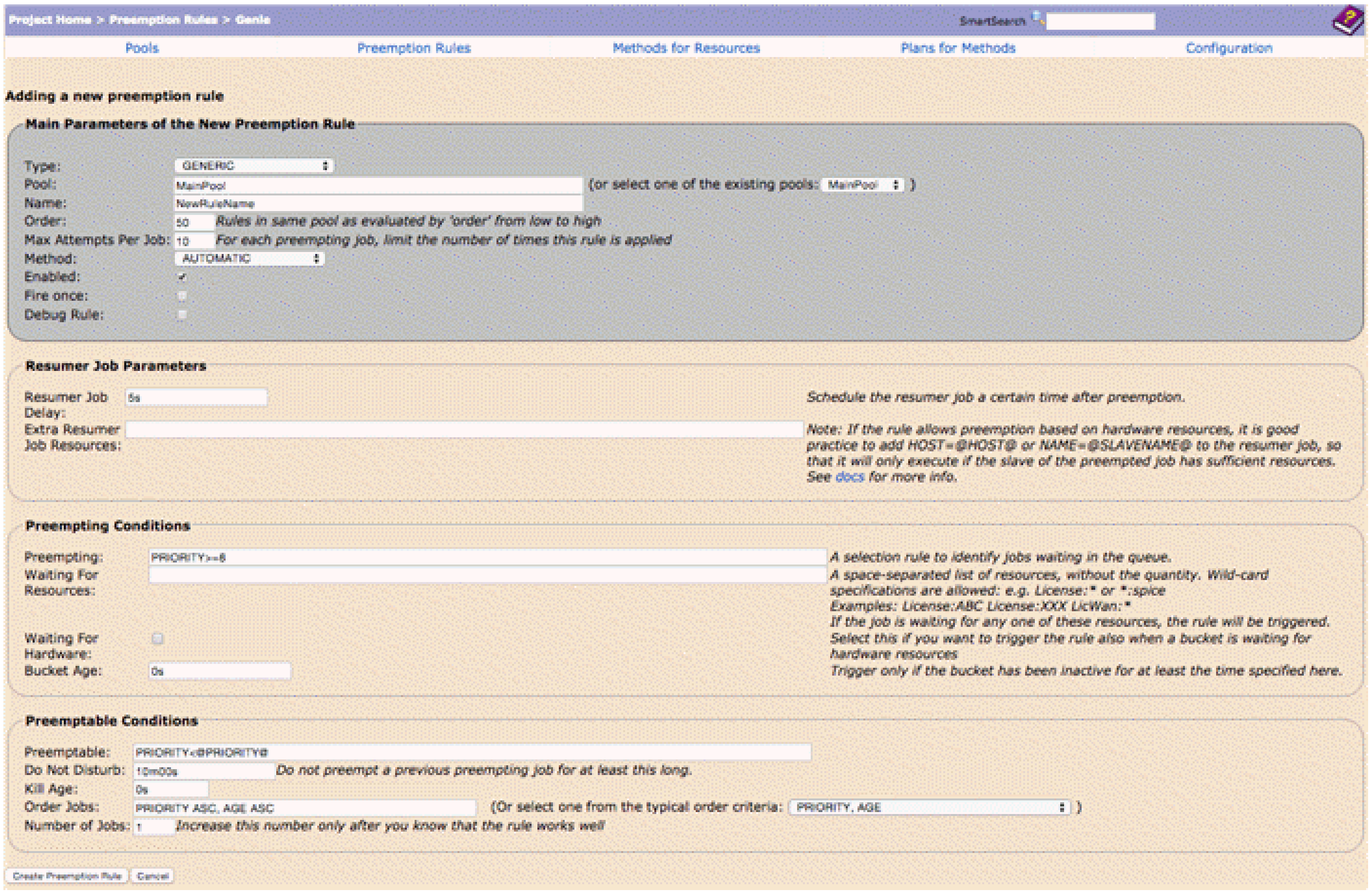
3 *Default is 3s. Typical values are between 1s and 20s.*

config.tcl
config_export.tcl
exports/config_20170422_103753.tcl
exports/config_20170422_103805.tcl
exports/config_20170614_125425.tcl
exports/config_20170614_132814.tcl
exports/config_20170614_133026.tcl
exports/config_20180427_100609.tcl
exports/config_20180504_075354.tcl
exports/config_20190428_151505.tcl

Figure 13:

Create a New Preemption Rule Using the Web UI

1. To enter a new rule, click **Add new preemption rule** at the bottom of the Preemption window.
   The Preemption Rule form opens.



Adding a new preemption rule

Main Parameters of the New Preemption Rule

Type: GENERIC
Pool: MainPool (or select one of the existing pools: MainPool)
Name: NewRuleName
Order: 50 *Rules in same pool as evaluated by 'order' from low to high*
Max Attempts Per Job: 10 *For each preempting job, limit the number of times this rule is applied*
Method: AUTOMATIC
Enabled: ✓
Fire once:
Debug Rule:

Resumer Job Parameters

Resumer Job Delay: 5s *Schedule the resumer job a certain time after preemption.*
Extra Resumer Job Resources: *Note: If the rule allows preemption based on hardware resources, it is good practice to add HOST=@HOST@ or NAME=@SLAVENAME@ to the resumer job, so that it will only execute if the slave of the preempted job has sufficient resources. See docs for more info.*

Preempting Conditions

Preempting: PRIORITY>=8 *A selection rule to identify jobs waiting in the queue.*
Waiting For Resources: *A space-separated list of resources, without the quantity. Wild-card specifications are allowed: e.g. License:* or *:spice*
*Examples: License:ABC License:XXX LicWan:**
If the job is waiting for any one of these resources, the rule will be triggered.
Waiting For Hardware: *Select this if you want to trigger the rule also when a bucket is waiting for hardware resources*
Bucket Age: 0s *Trigger only if the bucket has been inactive for at least the time specified here.*

Preemptable Conditions

Preemptable: PRIORITY<@PRIORITY@
Do Not Disturb: 10m00s *Do not preempt a previous preempting job for at least this long.*
Kill Age: 0s
Order Jobs: PRIORITY ASC, AGE ASC (Or select one from the typical order criteria: PRIORITY, AGE)
Number of Jobs: 1 *Increase this number only after you know that the rule works well*

Figure 14:

2. Select the type of preemption rule to implement.
   The window shows the appropriate options for the type.
3. Fill out the form and click **Create Preemption Rule** to save the new rule.

For examples on the types of preemption rules available, see Preemption Rule Types.

Preemption Rules to Speed Up FairShare

Every node in the FairShare tree represents a FairShare group. Each job belongs to one and only one FairShare group. Every node in the FairShare tree is assigned a **target share**, which depends on both the weights assigned to the nodes in the tree and on the activity of the nodes. A FairShare node is considered active if it has at least one job that is queued, running or suspended. All nodes that are not active are assigned a FairShare target of zero.

The target share of a FairShare node is accessible by the field `FS_TARGET` for any job that belongs to that FairShare node. The FairShare target is a fractional number less than 1.0, but the `FS_TARGET` field is an integer in the range from 0 to 10,000 obtained by scaling up the FairShare target by 10,000. For example, a `FS_TARGET` of 8000 indicates that the FairShare node has a target share of 80%(=0.8).

The field `FS_RUNNING` represents the fraction of running jobs in a FairShare group, relative to all running jobs in the system. This field is also scaled up by a factor of 10,000. The difference between `FS_RUNNING` and `FS_TARGET` is `FS_EXCESS_RUNNING`.

```
FS_EXCESS_RUNNING  := FS_RUNNING - FS_TARGET
```

This measures how much a group is above or below its target. A positive number of `FS_EXCESS_RUNNING` means that the FairShare group is running more jobs than it should.

The field `FS_RUNNING_COUNT` is the number of jobs a FairShare group is running.

The field `FS_HISTORY` represents the fraction of jobs that have been run by a FairShare group in the FairShare window (typically 2 hours) relative to all other jobs that have been run in the system. The difference between `FS_HISTORY` and `FS_TARGET` is `FS_EXCESS_HISTORY`, and is similar to `FS_EXCESS_RUNNING` explained above.

```
FS_EXCESS_HISTORY := FS_HISTORY - FS_TARGET
```

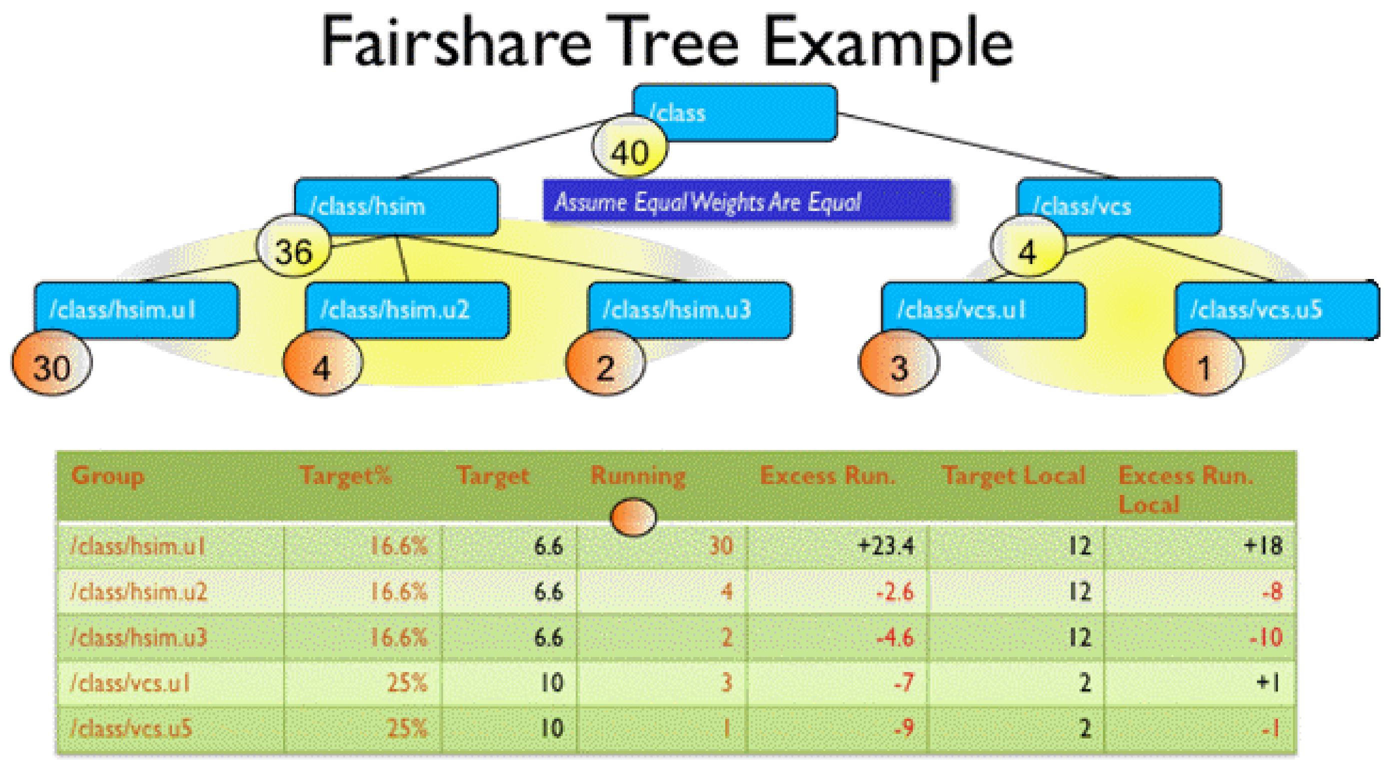
The field `FS_RANK` is computed by the scheduler and assigned to each FairShare group that has jobs in the queue. The jobs are dispatched to taskers in ascending order of rank, starting from the group of rank zero (0). Groups that have no jobs in the queue are assigned the conventional rank -1. For FairShare and preemption to work harmoniously, it is important that the rank of the preempted job is greater than the rank of the preempting job, which is why the preemption rules should contain a term of the form `FSRANK>@FSRANK@` in the -preemptable option. Since you also want to allow the preemption of jobs that are running but have no queued jobs in the same group, use the field "FS_RANK9", which is the same as FS_RANK, except that the value of FS_RANK9 for groups that have no queued jobs is 9,999,999 instead of -1, which makes for an easier comparison the preemptable rule `FSRANK9>@FSRANK9@`.

The Special Field FS_EXCESS_RUNNING_LOCAL

The picture below illustrates the difference between the `FS_EXCESS_RUNNING` field and `FS_EXCESS_RUNNING_LOCAL`. While the first considers the total number of running jobs in the system, the second field only considers the balance of running jobs at each local level. In the pictures, the nodes of interest are `/class/hsim` and `/class/vcs`.

The node `/class/vcs` has a total of 4 running jobs and 2 children, with user u1 running 3 jobs and user u5 running 1. Assuming that all weights are the same in all branches, the target share for `/class/vcs.u1` and `/class/vcs.u5` is exactly the same. Looking at the `FS_EXCESS_RUNNING`, it is negative for both nodes because the node `/class/hsim` has a large proportion of the running jobs. In this scenario, a preemption rule based on `FS_EXCESS_RUNNING` as shown below will not fire:

```
VovPreemptRule -rulename RuleThatDoesNotFire \
   -preempting   "JOBCLASS==vcs FS_EXCESS_RUNNING<0" \
   -preemptable  "JOBCLASS==vcs FS_EXCESS_RUNNING>0 FSRANK9>@FSRANK9@" \
    -pool fastfairshare -ruletype FAST_FAIRSHARE
```



Group	Target%	Target	Running	Excess Run.	Target Local	Excess Run. Local
/class/hsim.u1	16.6%	6.6	30	+23.4	12	+18
/class/hsim.u2	16.6%	6.6	4	-2.6	12	-8
/class/hsim.u3	16.6%	6.6	2	-4.6	12	-10
/class/vcs.u1	25%	10	3	-7	2	+1
/class/vcs.u5	25%	10	1	-9	2	-1

Figure 15:

On the other hand, with a local view of `/class/vcs`, it is apparent that the distribution of jobs is not balanced. To use preemption to speedup the achievement of balance, the `FS_EXCESS_RUNNING_LOCAL` field can be used as follows:

```
VovPreemptRule -rulename RuleThatFires \
   -preempting   "JOBCLASS==vcs FS_EXCESS_RUNNING_LOCAL<0" \
   -preemptable  "JOBCLASS==vcs FS_EXCESS_RUNNING_LOCAL>0 FSRANK9>@FSRANK9@" \
    -pool fastfairshare  -ruletype FAST_FAIRSHARE
```

Practical FairShare Driven Preemption

Frequently used fields that are used in preemption are:

- FS_EXCESS_RUNNING_LOCAL
- FS_RANK and FS_RANK9
- FS_RUNNING_COUNT

Other fields are described in *Node Fields* in the Altair Accelerator User Guide; those fields also begin with `FS_`.

Preemption Based on FairShare

Preemption can be used as a method to accelerate the FairShare mechanism, so that instead of waiting for a job to finish and a slot to open up, the preemption daemon can detect imbalances in the FairShare and preempt a job of a group that has excess share in favor a another group that has a deficit in the share.

A reference rule for this type of preemption can be found in `$VOVDIR/etc/config/vovpreemptd/rules/fastfairshare.tcl`, as shown below:

```
# Copyright (c) 1995-2020, Altair Engineering
# All Rights Reserved.

# $Id: $

# Use of preemption to speed-up fairshare.
#
# We assume a workload organized in jobclasses where the each jobclass
# has its own fairshare node called /class/$JOBCLASS.
#
# The preempting is triggered if there is a job which has a locally a
# deficit in the number of running jobs (FSEXCESSRUNNINGLOCAL<0) and has been waiting
 for at least 10 seconds.
# Also, if a fairshare group already has at least 4 jobs running, do not preempt.
#
# We do preemption within the same jobclass  (JOBCLASS==@JOBCLASS@)
# and we target the groups that have excessive share of running jobs
 (FSEXCESSRUNNINGLOCAL>0) and
# also a higher rank (FSRANK9>@FSRANK9@).  We use FSRANK9 instead of FSRANK to
# simplify the comparison of the ranks to include groups that have no rank.
# We do not want to preempt if the group has only one running job
 (FS_RUNNING_COUNT>1).
# We also consider priority (PRIORITY<=@PRIORITY@) to avoid preempting a job of
 higher priority.
#
```

```
VovPreemptRule -rulename FastFairshare \
    -preempting  "FSEXCESS<0 GROUP~/class FS_RUNNING_COUNT<=3" \
    -bucketage   10 \
    -preemptable "FSEXCESS>0 JOBCLASS==@JOBCLASS@ FS_RUNNING_COUNT>1
 FSRANK9>@FSRANK9@ PRIORITY<=@PRIORITY@" \
    -killage 2m  \
    -pool FastFairshare \
    -ruletype FAST_FAIRSHARE
```

Preemption Over Altair Allocator

The preemption daemon can be used to speed up the release of licenses at one site. The release of licenses is within the constraints imposed by Allocator. Constraints are enforced by specifying a preemption rule, which states that a specific resource is managed by Allocator and should be preempted.

```
VovPreemptRule -rulename "mqPreemptHsim"  -ruletype MULTIQUEUE  -multiqueueres
 License:hsim  -pool multiqueue
```

Preempt Jobs with Unrequested Resources

It is possible for resources to be used without being applied. To control the usage properly, a preemption rule can be created that prevents such issues. For this preemption rule, use the field `LM_HANDLES_NRU`, where `NRU` represents "Not Requested / Used". An examples is shown below.

In this example, the preempting condition is a job waiting for a license, `License:abc`. In addition, the queued job is in the specific jobclass `abc`.

```
 -preempting "JOBCLASS==abc"  -waitingfor "License:abc"
```

The preemptable set is any job that uses `License:abc` but does not ask for it:

```
 -preemptable "LM_HANDLES_NRU~License:abc"
```

For example:

```
VovPreemptRule -rulename PunishCheatersAbc \
   -ruletype GENERIC \
   -preempting "JOBCLASS==abc"  -waitingfor "License:abc" \
   -preemptable "LM_HANDLES_NRU~License:abc" \
   -method AUTOMATIC
   -pool   PunishPool
```

Control Whether a Job is Preemptable

This field is a boolean and identifies the jobs that can be preempted.

This flag can be set using the variable *make(preemptable)* in a FDL file.

In Accelerator, all jobs are preemptable by default. To disable the flag, use the option -preemptable 0 at submission time.

```
% nc run -preemptable 0 sleep 100
```

Note: See @KEYWORD* Expansion for details on using @KEYWORD@ expansion with `-preemptable`.

Preemption Timing

The preemption subsystem has a complex behavior, controlled by many delays that are described in this section.

Preemption Cycle

A `preempt` cycle is set within the `policy.tcl` server configuration file. The unit is in seconds. The default value 3 seconds,

No Preemption After Resumption

Job resumption is supported by the property `PREEMPTRESUME`. In `VovPreemptRule`, this function is supported by `-skipresumedjob TIMESPEC`. The default value is 2 minutes.

Job Too Young to Preempt

This delay depends on the resource. This function is represented by the variable *PREEMPT($resmap,delay)*. The option is -delay in `VovPreemptPolicy`. The default value is 5 seconds.

Minimum Bucket Age to Trigger Preemption

Each preempting job is waiting in the queue and therefore belongs to a bucket. The age of the bucket depends on the time of the last dispatch of a job from that bucket, or the bucket creation, whichever is younger.

Note: If the age of the bucket is less than this minimum age, the preemption is not triggered. This is represented by `PRULES($rule,bucketage)`, controlled by the option -bucketage in `VovPreemptRule`. The default value is zero; there is no minimum age for firing the rule.

Resource Reservation Time

When the resource is reserved for the preempting job, the preempting method used is RESERVE. With preemption rules, this reservation time is represented by `PRULES($rule,reservetime)`, which is controlled by the option -reservetime in `VovPreemptRule`. The default value is 30 seconds.

Too Early to try lmremove

If the age of a checkout assigned to a job is less than 2 minutes. This is hard coded.

Do not Preempt a Resumed Job

If a job has just been resumed, it is recommended to not allow the job to immediately be preempted. Instead, allow the job time to settle and give Accelerator time to figure out which licenses the job is using. This preempt method is represented by `PRULES($rule,skipresumedjob)`, which can be set with option -skipresumedjob TIMESPEC in `VovPreemptRule`

The default value is 2 minutes.

Kill Instead of Suspend a Job

If a job is younger than a certain age, we prefer killing and resubmitting it rather than suspending and resuming it. This is represented by `PRULES($rule,killage)` and is controlled by the -killage option in `VovPreemptRule`.

The default value is 2 minutes. A value of zero disables the killing.

Do not Disturb a Top Job

If a job successfully triggered a preemption, you do not want that job to be bothered for some time. This is represented by `PRULES($rule,donotdisturb)`, which is controlled by the option -donotdisturb in `VovPreemptRule`.

The default value is 10 minutes.

Safe Job Control Timeout

This is the time preemption waits for the signals to have effect. Typically the signals have immediate effects. However it has been noticed that SIGTSTP may take many seconds to take effect. This is represented by the variable `vovpreempt(timeout,safejobcontrol)`. The default value is 30s. The valid range is between 1s and 2m, and it is silently enforced.

Failed Preemption, Job-to-Watch Timeout

If a job cannot be preempted, it is put into a "jobs-to-watch" list for the time specified by `vovpreempt(timeout,jobstowatch)`. The default value is 30m.

Start the Preemption Rule Compiler Daemon vovpreemptd

Note: The following commands must be executed on the host where the vovserver is running.

The daemon `vovpreemptd` can be started or stopped using `vovdaemonmgr`.

```
% vovdaemonmgr start vovpreemptd
% vovdaemonmgr stop vovpreemptd
```

For debugging, it can be useful to start the daemon in the foreground.

```
% cd `vovserverdir -p vovpreemptd`
% vovpreemptd -v -v
```

Note: When a new instance of the preemption daemon is started, the previously running instance is automatically terminated.

Automatically Start the Preemption Daemon

Note: If autostart scripts are not yet in use, it may be necessary to create the autostart directory as a subdirectory of the `.swd` for Accelerator.

To start the preemption daemon when the server is started, add the following executable script to the Autostart Directory for Accelerator. It is expected that the `lmremove` command is available in the path.

```
% cd `vovserverdir -p autostart`
% cp $VOVDIR/etc/autostart/vovpreemptd.csh .
```

vovpreemptd

Main preemption daemon, based on the C++ implementation of preemption.

```

vovpreemptd: Usage Message

  DESCRIPTION:
     Main Preemption Daemon, based on the C++ implementation
     of preemption.

  USAGE
      % vovpreemptd [OPTIONS]

  OPTIONS:
     -h       -- This help
     -v       -- Increase verbosity
```

```
      -n      -- Normal (no-op)

   EXAMPLES:
      % vovpreemptd -h                   -- usage message
      % vovpreemptd                      -- normal start
      % vovpreemptd -n                   -- same as above
      % vovpreemptd -v -v                -- for debugging
```

Manual Preemption

Manual preemption can be used in addition to or instead of automatic preemption.

In automatic preemption, the preemptable jobs are identified by `vovpreemptd`; however, in manual preemption, the ID of the preemptable job is required with the `nc preempt` command.

It is sometimes advantageous to manually preempt a job. For example, the licenses are available, but all the CPU slots are taken by other jobs and an important job must run now. Preempting a job can only be applied by the owner of the job or by having ADMIN privileges.

A job can be submitted with very high priority, such as `high` or `top`. Later, if needed, a running job can be preempted with `nc preempt jobId` if the high level job is waiting in the queue.

Unless the method is specified with the -method, manual preemption uses the configured preemption method for the resources held by the preempted jobs, which is the same as applied for automatic preemption. The preemption method is defined in the configuration file for the automatic preemption daemon. See Automatically Start the Preemption Daemon for the location of this file.

Examples:

```
% nc preempt 34567
% nc preempt -manualresume 45678
```

Manual Preemption with Manual Resumption

In normal preemption, the preempted job is suspended and another job, called the resumer job, is created. The resumer job is scheduled to be executed as soon as the required licenses become available.

```
% nc preempt 12345
### No need to call nc resume
```

With the option -manualresume of `nc preempt`, the resumer job is created but not scheduled. To schedule the resumer job, use `nc resume`.

```
% nc preempt -manualresume 12345
# ... later, you have to remember to resume the job...
% nc resume 12345
```

nc preempt

Preempt the specified running jobs.

```

vnc: Usage Message

  NC PREEMPT:
      Preempt the specified running jobs.
      Preemption means that:
      1.  The job is stopped or suspended (depending on method and age)
      2.  The resources of the job are revoked
      3.  If needed, a 'resumer job' is scheduled to
```

```
        restart the job as soon as the revoked
        resources are again available.

        If a job is not running, an error is reported.

    USAGE:
        % nc preempt [OPTIONS] <jobId> ...

    OPTIONS:
        -h                     -- This help.
        -v                     -- Increase verbosity.
        -method METHOD         -- Specify preemption method. Default is AUTOMATIC.
                                  Common values:
                                  AUTOMATIC, KILL, KILL+RESUBMIT, SUSPEND,
                                  SIGTSTP+SUSPEND
                                  Other values and example of preemption plans
                                  (see docs): MODELSIM, BEGIN:RETRACING:EXT,
                                  TSTP, vish.
                                  Bad values are currently ignored.
        -manualresume          -- The resumer job is not scheduled;
                                  A 'nc resume' is required to restart the
                                  preempted job.
        -resumeres RESLIST     -- Specify resources to be added to the
                                  resumer job. The resources are expanded.
                                  Examples:
                                  @HOST@ RAM/@RAM@
                                  @PROP.SOLUTION@
                                  Any job field can be used, but here are
                                  some common fields that can be useful:
                                  @TASKERNAME@
                                  @HWRAM@ @HWCORES@ @HWPERCENT@ @HWSLOTS@

    EXAMPLES:
        % nc preempt 123456
        % nc preempt -v 123456

        % nc preempt -method SIGTSTP 123456
        % nc preempt -method KILL+RESUBMIT 123456
        % nc preempt -method BEGIN:RETRACING:EXT,TSTP,vish 123456
        % nc preempt 03076307 03076311 03076315

        % nc preempt -manualresume 123456
        % nc preempt -manualresume -resumeres @PROP.SOLUTION@ 123456
        % nc preempt -manualresume -resumeres "@HOST@ RAM/@MAXRAM@" 123456
        % ...
        % nc resume  123456
```

Options

Most of the options are explained by the summaries in the brief usage output shown above. The `-v` option enables printing of additional messages that may be helpful in troubleshooting.

The `-method` option may be used to specify the preemption method used to revoke the resources of the preempted jobs. This overrides any method in the configuration file. This option may be helpful in troubleshooting. The `-l` option is handy to get a list of known preemption methods. The names of the methods are *case-sensitive*.

Only users with the Altair Accelerator ADMIN privilege or the owner of the preempted job can run this command.

Preempting Tokens

There are families of tools that use different numbers of tokens per license. The number of tokens used depends on the application. For example, the family Cadence Multi-Mode simulators contains tools such as Spectre, AMS, and UltraSim, using 1, 2, and 6 tokens respectively. These families of tools require special attention when configuring preemption.

In the following examples, it is assumed there are eight groups of users (My1, ..., My8) competing for the tokens, represented by the resource map `MyTokens`. It is also assumed that the jobs are assigned to different jobclasses: `spectre`, `ams`, and `ultrasim`.

Three types of preemption will be set up:

- Based on priority, restricted to jobs belonging to the same user
- Based on ownership
- Based on avoiding starvation of multi-token jobs by reserving tokens

Priority-based Preemption for Tokens

In this rule, a high priority job is allowed to preempt a low priority job that is in the same jobclass and is owned by the same user. The preempting jobs need to be waiting for the `MyTokens` resource. Jobs younger than 20 seconds are killed and resubmitted; they are not suspended or resumed.

```
    # Fragment of vnc.swd/vovpreemptd/config.tcl
    VovPreemptRule -rulename MyPriSameUser \
        -preempting "priority>=8" \
        -waitingfor MyTokens \
        -preemptable "jobclass==@JOBCLASS@ user==@USER@ priority<4" \
        -killage 20 \
        -pool contract:My
```

The only difference between the preemption on tokens and the preemption on regular license is the use of the pool `contract:My`. This pool is used for the rules based on Ownership, which is described in the following section. In a given preemption cycle, the ownership rules will not fire if the `MyPriSameUserK` rule is fired.

Avoid Starvation for Tokens

With mixed workloads of `spectre` and `ultrasim` jobs, it is difficult for an `ultrasim` job to be scheduled. It is also difficult for six tokens to be available at the same time, as `spectre` jobs will likely take all the tokens as the tokens become available. This *starvation* for the `ultrasim` jobs can be prevented by reserving tokens for the `ultrasim` jobs as described in the following example.

In this example, for jobs in the `ultrasim` class, if a job that has at least normal priority and has been waiting for more than 2 minutes, 6 tokens are reserved for the top job for a period of 1 minute. As this reservation is renewed at every cycle, the reservation period does not need to be long. A subordinate rule in the same pool takes care of the jobs in the `ams` class.

Note:

- The resource reservations for a specific job are automatically dismissed when the job is no longer in the queue because it is either dispatched or descheduled.
- It is not necessary to specify the resources to be reserved, as that is automatically computed.

The same pool, `My:reserve`, is used for both `ultrasim` and `ams`. This is to not have both rules to fire in each cycle. Having reserved tokens for the `ultrasim` jobs, it is not desirable to also reserve tokens for `ams`, as `ams` would likely prevail.

```
# Fragment of vnc.swd/vovpreemptd/config.tcl
VovPreemptRule -rulename MyAntiStarvationUltrasim \
    -preempting "jobclass==ultrasim priority>=4" \
    -bucketage  "2m" \
    -method  "RESERVE" \
    -reservetime  1m -reservenum 6 \
    -pool "My:reserve"

VovPreemptRule -rulename MyAntiStarvationAms \
    -preempting "jobclass==ams priority>=4" \
    -bucketage  "2m" \
    -method  "RESERVE" \
    -reservetime  1m -reservenum 2 \
    -pool "My:reserve"
```

Remove Licenses from a Preempted Job

The utility `vovlmremove` is used to release license resources from a suspended job. This utility is normally called automatically by the preemption code; it can also be called from the command line. The usage is simple: pass the VovId of the jobs from which to release the license resources, as shown below.

```
% vovlmremove 00012345 00012355
```

The selection of utility calls `lmremove`, `rlmremove`, or `vlmremove`, is related to the licenses that currently match to the job.

Utility	**Related License**
lmremove	FlexNet Publisher license
vlmremove	License emulation

Automatic Call of vovlmremove for Suspended Jobs

The utility `vovlmremove` is automatically called by vovserver when a preempted job is in the `SUSPENDED` state and matches some existing license handles. The utility attempts to free up the handles that match. This functionality is called with a frequency controlled by the parameter *`preemptionResourceRelease`*, which defaults to 90 seconds. This functionality can be disabled by setting this parameter to 0.

The utility `vovlmremove` is called during the following conditions:

- A job has been preempted at least 60 seconds before and is currently in the `SUSPENDED` state (system controlled).
- The job does not have the property `DONOTLMREMOVE` (user or system controlled).

vovlmremove

This utility removes all the license features associated with the given job.

```
vovlmremove: Usage Message

  USAGE:
     % vovlmremove [OPTIONS] <jobId> ...

     This utility removes all the license features associated with the given job.
     It uses 'lmremove' for FLEXlm licenses.
     It uses 'rlmremove'  for licenses that use a tag containing RLM.
     It also works with Altair Engineering license emulation using 'vlmremove'.

     Due to FLEXlm behavior, it is possible and expected for lmremove to fail if
     it is called too early (typically 2 minutes after a checkout has occurred).
     In such cases, vovlmremove should be retried after adequate time has passed.

  OPTIONS:
     -h      Print this help message.
     -v      Increase verbosity
     -prop   Add a property called "PREEMPT_LMREMOVE" to the job to describe what
```

```
             has been done.

EXAMPLES:
   % vovlmremove 00012345
   % vovlmremove 00012345 00022334
   % vovlmremove -v 00012345
   % vovlmremove -prop 00012345
   % vovlmremove -h
```

Note

`vovlmremove` uses `lmremove` for FlexNet Publisher licenses. It uses `rlmremove` for licenses that use a tag containing RLM. It also works with Altair Engineering license emulation using `vlmremove`.

Because of the FlexNet Publisher behavior, it is possible and normal for `lmremove` to fail if it is called too early (typically before 2 minutes after a checkout). In such case, you can try to run `vovlmremove` again a bit later.

Resource Management

Altair Accelerator includes a subsystem for managing computing resources. This allows the design team to factor in various constraints regarding hardware and software resources, as well as site policy constraints.

This mechanism is based on the following:

- Resources required by jobs
- Resources offered by taskers
- Resource maps, as described in the file `resources.tcl`

There are several types of resources, which are listed below:

Resource type	**Representation**	**Explanation**
Job Resources	`name`	A resource required by a job. If the quantity is not shown, the default is 1; "unix" is the same as "unix#1".
Uncountable Resources (also called Attributes)	`name`	These resources represent attributes of a tasker that are not countable. For example, a tasker may have attributes such as "unix" or "linux". The quantity is not shown for these resources and it defaults to MAXINT; "unix" is equivalent to "unix#MAXINT".
Quantitative Resources	`name#quantity`	Example: The resource `RAMTOTAL#2014` on a tasker indicates the total amount of RAM on that machine. On a job, it says that the job requires at least the shown amount of RAMTOTAL.
Consumable Resources	`name/quantity`	Example: RAM/500 assigned to a job indicates that the job consumes 500 MB of the consumable resources RAM.
Negated Resources	`!name`	Example: "unix !linux" on a job indicates that the job requires a UNIX machine but not a Linux one.

The definition of the quantity is related to the context of the resource. If the context is a tasker, quantity represents how much of that resource is available from the tasker. If the context is a job, quality represents how much of that resource is required by the job.

Note: Negated resources are allowed only for the context of a job.

The *unit of measure* is determined by convention for each resource. For example, the resource `RAMTOTAL` is measured in MB. By default, quantity is assumed to be 1; the notation `foo` is equivalent to `foo#1`.

A *resources list* is a space-separated list of resources, which are typical resources offered by the taskers. The following example indicates that a job requires at least 128 MB of RAM and a UNIX host, but not a Linux host.

```
RAMTOTAL#128 unix !linux
```

A *resources expression* is a space separated list of resources and operators: typical resources requested by the jobs or mapped in the resource map set. Operators can be one of the following: `<blank space>`, &, `|`, `OR`, `AND`, `!`, and `NOT`. The operators are defined in the table below.

Note: Logical `AND` has precedence over logical `OR` operations.

Operator	Description
`<blank space>`	implicit logical AND
`&`	explicit logical AND
`AND`	explicit logical AND
`|`	explicit logical OR
`OR`	explicit logical OR
`!`	explicit logical negation
`NOT`	explicit logical negation

For example, a job may have the following resource requirements:

```
RAMTOTAL#128 unix !linux | RAMTOTAL#512 & linux
```

This job requires either a UNIX host with at least 128 MB of RAM, but not a `linux` host or a Linux host with at least 512MB of RAM.

Also in this Section

Hardware Resources

All taskers offer a predefined set of hardware resources that can be requested by jobs.

All taskers offer a predefined set of hardware resources that can be requested by jobs. These resources are listed in the following table.

Hardware Resource	Type	Description
ARCH	STRING	The VOV architecture of the machine, for example "linux64", "win64", "armv8"

Hardware Resource	Type	Description
CORES	INTEGER	Consumable resource: the number of logical CPUs/processors used by a job.
CORESUSED	INTEGER	The total number of cores used by the running jobs. It is assumed that each job uses at least one core.
CLOCK	INTEGER	The CPU-clock of at least one of the CPUs on the machine in MHz. If the machine allows frequency stepping, this number can be smaller than expected.
GROUP	STRING	The tasker group for this tasker. Each tasker can belong to only one tasker group.
HOST	STRING	The name of the host on which the tasker is running. Typically this is the value you get with `uname -n`, except only the first component is taken and converted to lowercase, so that if `uname -n` returns `Lnx0123.my.company.com` the value of this field will be `lnx0123`.
LOADEFF	REAL	The effective load on the machine, including the self-induced load caused by jobs that just started or finished.
L1	REAL	On UNIX, the load average in the last one minute.
L5	REAL	On UNIX, the load average in the last five minutes.
L15	REAL	On UNIX, the load average in the last fifteen minutes.
MACHINE	STRING	Typically the output of `uname -m`.
MAXNUMACORES	INTEGER	Highest total number of NUMA cores in a single NUMA node.
MAXNUMACORESFREE	INTEGER	Highest number of free cores in a single NUMA node. Note that free NUMA cores are correctly accounted for only if the user specified **-jpp pack** or **-jpp spread** for all jobs on the tasker.

Hardware Resource	Type	Description
NAME	STRING	The name of the tasker.
OS	STRING	The name of the operating system: "Linux" or "Windows".
OSCLASS	STRING	This can be `unix` or `windows`.
OSVERSION	STRING	The version of the OS. On Linux, this can usually be found in `/etc/system-release`.
OSRELEASE	STRING	Typically the output of `uname -r`.
PERCENT	INTEGER	Consumable resource: The percentage of the machine that is still available.
POWER	INTEGER	The effective power of the tasker, after accounting for both raw power and the effective load.
RAM	INTEGER	A consumable resource expressing the remaining RAM available to run job: RAMTOTAL-RAMUSED, in MB.
RAMFREE	INTEGER	The amount of RAM available to run other jobs. This metric comes from the OS, and on linux it includes both free memory and buffers. In MB.
RAMTOTAL	INTEGER	The total amount of RAM available on the machine, in MB.
RAMUSED	INTEGER	The aggregate quantity of RAM used by all jobs currently running on the tasker, in MB. For each job, the amount of RAM is calculated as the maximum of the requested RAM resource (REQRAM) and the actual RAM usage of the job (CURRAM).
RELEASE	STRING	On Linux machines, this is the output of `lsb_release -isr`, with spaces replaced by dashes. For example, `CentOS-6.2`
SLOT	INTEGER	A consumable resource indicating how many more jobs can be run on the tasker.

Hardware Resource	Type	Description
SLOTS	INTEGER	Same as SLOT
SLOTSUSED	INTEGER	Corresponding to the number of jobs running on the tasker.
STATUS	ENUMERATED TYPE	Possible values are `BLACKHOLE`, `BUSY`, `DEAD`, `DONE`, `FULL`, `OVRLD`, `NOLIC`, `NOSLOT OK`, `PAUSED`, `READY`, `REQUESTED`, `SICK`, `SUSP`, `WARN`, `WRKNG`
SWAP	INTEGER	A consumable resource. The swap space in MB.
SWAPFREE	INTEGER	The amount of free swap.
SWAPTOTAL	INTEGER	Total about of swap configured on the machine.
TASKERNAME	STRING	Same as *NAME*
TASKERHOST	STRING	Same as *HOST*
TIMELEFT	INTEGER	The number of seconds before the tasker is expected to exit or to suspend. This value is always checked against the expected duration of a job.
TMP	INTEGER	On UNIX, free disk space in /tmp, in MB.
USER	STRING	The user who started the vovtasker server, which is usually the same user account associated with the vovserver process.
VOVVERSION	STRING	The version of the vovtasker binary (such as '2015.03').

Request Hardware Resources

Each job can request hardware resources.

Note: The consumable resources are `CORES`, `CPUS`, `PERCENT`, `RAM`, `SLOT`, `SLOTS`, and `SWAP`.

- To request a machine with the name `bison`, request `NAME=bison`. To request any linux64 machine, request `ARCH=linux64`.

- Consumable resources are added together. For example `-r CORES/2 CORES/4 CORES/6` is a request for a total of 12 cores.
- If redundant resources are specified, the largest value will be taken. For example, if `-r RAMTOTAL#2000 RAMTOTAL#4000` is specified then `RAM#TOTAL4000` will be the resource that is used.

Request examples are listed in the following table:

Request Objective	**Syntax for the Request**
A specific tasker	`NAME=bison`
Not on bison	`NAME!=bison`
One of two taskers	`NAME=bison,cheetah`
A preference: bison, if it is available; otherwise, cheetah	`(NAME=bison OR NAME=cheetah)`
A specific architecture, such as Linux	`ARCH=linux`
A specific tasker group, such as prodLnx	`GROUP=prodLnx`
2 GB of RAM	`RAM/2000`
Two cores	`CORES/2`
Two slots	`SLOTS/2`
Exclusive access to a machine	`PERCENT/100`
1 minute load less than 3.0	`L1<3.0`

Wildcard Tasker Resources

A tasker can also offer resources that contain a wildcard. The wildcard is '*' and can be used instead of the name or the type. Legal values for wildcard resources are:

Wildcard	**Description**
*	Matches all resources that have no type
:	Matches all resources
*:hsim	Matches all resources with name "hsim"
License:*	Matches all resources of type "License"

Wildcard	**Description**
JobType:*	Matches all resources of type "JobType"

These resources are particularly useful for indirect taskers, which are used to transfer jobs from FlowTracer to Accelerator.

Resource Mapping

As the vovservers determines which tasker is most suited to execute a particular job, it performs a *mapping* of the job resources, followed by a *matching* of the mapped resources.

When dispatching a job, the vovservers does the following:

- Gets the list of resources required by a job.
- Appends the resource associated with the priority level, such as `Priority:normal`.
- If it exists, it appends the resource associated with the name of the tool used in the job (reminder: the tool of a command is the tail of the first command argument after the wrappers). The tool resource has type `Tool` and looks like this: `Tool:`*`toolname`*.
- Appends the resource associated with the owner of the job, such as `User:john`.
- Appends the resource associated with the group of the job, such as `Group:time_regression`.
- Expands any special resource, i.e. any resource that starts with a "$".
- For each resource in the list, the vovservers looks for it in the resource maps. If the resource map is found and there is enough of it, that is, the resource is available, the vovservers maps the resource. This step is repeated until one of the following conditions is met:
  - # The resource is not available. In this case, the job cannot be dispatched and is left in the job queue.
  - # A cycle in the mapping is detected; in this case the job cannot be dispatched at all and is removed from the job queue.
  - # The resource is not in the resource map.
- VOV appends the resource associated with the expected job duration to the final resource list. For example, if the job is expected to take 32 seconds, the resource `TIMELEFT#32` will be appended.
- Finally, the vovservers compares the resulting resource list with the resource list of each tasker. If there is a match - all resources in the list are offered by the tasker - the tasker is labeled as eligible. If there is no eligible tasker, the job cannot be dispatched at this time and remains in the queue; otherwise, the server selects the eligible tasker with the greatest effective power.

Local Resource Maps

Resource maps can be designated as *local*, using the `local` flag.

Important: This flag is only available and supported for a FlowTracer installation, utilizing `vovwxd` and an LSF interface.

Resource maps designated as local will be managed on the "local" (FT) side of the `vovwxd` connection instead of the normal case where resource specifications are expected to be managed on the base queue side.

For example, to limit jobs to running 5 at a time from a specific FlowTracer project, do the following:

1. Enable local resources with `run: vovservermgr configure vovwxd.localresources 1`

   Note: Alternatively, you can add the following to the `policy.tcl` file:

   ```
   set config(vovwxd.localresources) 1
   ```

2. Create the local resource.
   a. Run `vovresourcemgr set mylocallimit -max 5 -local`

      Note: Alternatively, you can add the following to the `resource.tcl` file:

      ```
      vtk_resourcmap_set mylocallimit -total 5 -local
      ```

This results in limiting running jobs with the local resource `mylocallimit` to a maximum of 5 jobs at a time.

For example, FDL to use a local resource named "mylocallimit:

```
R mylocallimit

J vov /bin/sleep 0
```

Resources Representing the Sum of Others

The procedure `vtk_resourcemap_sum` is used to define a resource map as the sum of other resource maps. It takes two arguments:

- The name of the resource map
- The list of the resource components

For example, suppose you have a resource map called `License:a` and another called `License:b`. You can create a sum resource called `License:sum` using:

```
# This code fragment typically goes into resources.tcl
vtk_resourcemap_sum License:sum [list License:a License:b]
```

This results in a resource `License:sum` defined as follows:

```
# This is the result of using vtk_resourcemap_sum....
vtk_resourcemap_set License:sum -max [expr $qa+$qb] -map "License:a OR License:b"

## ... or this map if you activate "commas" (see Commas vs. ORs in
       Resources).
vtk_resourcemap_set License:sum -max [expr $qa+$qb] -map "License:a,License:b"
```

Where `qa` and `qb` are the current max values for `License:a` and `License:b` respectively. The sums are recomputed by `vovresourced` about once a minute, or by Allocator every 30 seconds.

Commas vs. ORs in Resources

Motivation for Commas

For various reasons, the current Accelerator scheduler suffers from an growth of complexity depending on the number of OR's in a resources expression. For example, this expression has 2 ORs:

```
"( License:A OR License:B  OR License:C ) RAM/200"
```

Practical considerations limit the allowed number of ORs to about 20. However, in many cases, you do not need to use the expensive OR when instead we are trying to request any resource in a list of resources. For this purpose, there is the "comma operator", and the equivalent expression above can be rewritten as:

```
"License:A,License:B,License:C RAM/200"
```

The scheduler will pick any one of `License:A`, `License:B` or `License:C`, in that order. Such expression is much cheaper to compute.

As of version 2016.09u15, full support of this comma operator is offered.

Activation of Commas in vovresourced

To activate the use of commas in resource expressions in `vovresourced`, add the following setting in the config file (either `resources.tcl` or `vovresourced/config.tcl`)

```
# Add this to vovresourced/config.tcl
set RESD(useCommas) 1
```

Activation of Commas in Allocator

For the time being, the use of commas to represent groups of resources is activated by setting the environment variable VOV_USE_COMMAS_IN_MAPS to 1 when calling `vtklanc`. This is most easily accomplished by setting the variable in the `setup.tcl` file and restarting the taskers.

```
# Set this in   la.swd/setup.tcl
setenv VOV_USE_COMMAS_IN_MAPS 1
```

It is expected that finer control for this will be provided in future versions of Allocator.

Automatic Resource Limits

The resources of type `Limit` are treated specially by VOV. When a job is created or submitted, if the name of the job resource contains one or more of these tokens `@USER@`, `@GROUP@`, `@JOBCLASS@`, `@JOBPROJ@` then each token is replaced by the value of the corresponding field for the job. The resource map with the token is called the **symbolic limit** while the derived resource map is called the **specific limit**.

For example, if user "john" submits a job with the resources `Limit:abc_@USER@`, the following happens:

- The resource requirement for the job is changed so that `Limit:abc_@USER@` is replaced with `Limit:abc_john`

- A new resource map called `Limit:abc_john` is created. This resource map will be assigned a maximum amount equal to the maximum of the resource map called `Limit:abc_@USER@`, if such resource map exists, or just 1 if the resource does not exist.

To use the automatic limit resources, the admin needs to create the symbolic resource. For example:

```
# In resources.tcl
vtk_resourcemap_set Limit:queue_normal_@USER@ 10
```

To change the value of all limits derived from a symbolic limit, you should use the procedure `vtk_resourcemap_set_limit`. This procedure normally sets all derived limits to the same new value, but it also allows the specification of different limits for selected users or the reduction of specific limits based on the out-of-queue usage of a given license.

```
# Example 1: set all derived limits to 15:
vtk_resourcemap_set_limit Limit:queue_normal_@USER@ 15

# Example 2: set all derived limits to 15, with a few exceptions:
vtk_resourcemap_set_limit Limit:queue_normal_@USER@ 15 -special {
   Limit:queue_normal_mary 20
   Limit:queue_normal_john  3
}

# Example 3: set all derived limits to 15, but consider out-of-queue usage
#            Since this limit changes over time, we put it inside a TIMEVAR
#            procedure, so it is computed once every minute.
TIMEVAR hsim_ooq {
   default {
       vtk_resourcemap_set_limit Limit:queue_hsim_@USER@ 15 -ooq License:hsim
   }
}
```

Resource Daemon Configuration

vovresourced

Table 1:

Working directory	`vnc.swd/vovresourced`
Config file	`vnc.swd/vovresourced/resources.tcl`
Auxiliary config file	`$VOVDIR/local/resources.tcl`
Info file	`vnc.swd/vovresourced/resourced.pid`

The daemon `vovresourced` is the main agent that defines the resources of the vovservers. The configuration file is `resources.tcl`, which is located in the server configuration directory. This file defines which resources are to be used by the server by calling the procedure `vtk_resourcemap_set`. Examples are available in the `vnc.swd/resources.tcl` file.

The procedures `vtk_flexlm_monitor` and `vtk_flexlm_monitor_all` are used to define resources that are derived from licenses. If the `resources.tcl` file calls for monitoring FlexNet Publisher features with the command `vtk_flexlm_monitor`, or when Monitor receives notification of an event from LICMON, `vovresourced` then retrieves the information about the event. Refer to the `vnc.swd/resources.tcl` file for examples.

Refresh Rate

The frequency of the checks can be configured in the `resources.tcl` file as shown below:

```
set RESD(refresh)  10000;   #Set refresh time to 10,000 milliseconds.
```

Starting vovresourced

The program `vovresourced` is normally started automatically by the server. `vovresourced` can also be invoked manually from any directory. After reading the `vovresourced` configuration file, `vovresourced` then reads the auxiliary configuration file `$VOVDIR/local/resources.tcl` (if the auxiliary file has been created). When the `resources.tcl` file is changed, the `vovresourced` daemon is restarted with the following commands:

```
% vovproject reread          ; # Generic method for any vovserver.
% nc cmd vovproject reread   ; # Specific for Accelerator.
% ncmgr reset                ; # Specific for Accelerator.
```

Manage Resources with the CLI

`vovresourcemgr` is a utility for managing VOV resource maps. It may be used to create, modify, forget, and reserve resource maps.

Resouce map names support most of the ASCII characters except `# / =`. Using alphanumeric characters is strongly recommended.

```

vovresourcemgr: Usage Message

USAGE:
  % vovresourcemgr COMMAND  [options]

COMMAND is one of:
   show                 Show summary info about all resource maps
   show    [R1..RN]     Show info about specified resource map(s)
   matches RESMAP       Show license matching info
   ooq     RESMAP       Show out of queue license handles
   create  RESMAP map-options
                        Create a new resource map
   set     RESMAP map-options
                        Create a new or modify an existing resource map
   reserve RESMAP TYPE WHO HOWMANY HOWLONG WHY [-exclusive]
                        Place a reservation on a resource map
   forget  [-force] R1 [R2..RN]
                        Remove resource map(s) from the system

MAP-OPTIONS:
   -expire specify expiration (timespec) relative to now
   -max    specify quantity
   -map    specify map-to value
```

```
    -rank   specify rank when setting
    -noooq  do not track out-of-queue
    -local  specify that this is a local resource (when using vovwxd)

For reserve, TYPE is one of: USER,GROUP,JOBCLASS,JOBPROJ,JOBID.

EXAMPLES:
  % vovresourcemgr show
  % vovresourcemgr show  Limit:abc
  % vovresourcemgr matches  Limit:abc
  % vovresourcemgr create License:spice -max 8
  % vovresourcemgr set License:spice -max 10
  % vovresourcemgr set License:spice -map "Policy:spice"
  % vovresourcemgr ooq License:spice
  % vovresourcemgr reserve License:spice USER john,jane 3 3d ""
  % vovresourcemgr reserve License:spice USER bill 1 1w "" -exclusive
  % vovresourcemgr forget  License:spice
  % vovresourcemgr forget  -force License:spice
```

Note: The `vovresourcemgr` utility command connects to and acts on the VOV project enabled in the shell where it is launched. To act on Accelerator, use `vovproject enable vnc`, or precede it with `nc cmd` as shown in the examples below.

Dynamic Resource Map Configuration

Persistent resource maps are defined in the `resources.tcl` configuration file for a project. The `vovresourcemgr` command is useful to make changes to the resource maps on the fly.

Note: Unlike resource maps defined in `resources.tcl`, changes made with `vovresourcemgr` do not persist across restarts of vovserver.

The `create` command checks for existence of the named resource map and exits with a message if it already exists. The `set` command will create or replace an existing resource map with the given values with no confirmation.

The following example creates a new resource map named `Limit:spice`, which is created with a quantity of `10` and an empty map-to value.

```
% nc cmd vovresourcemgr set License:spice -max 10
```

Resource Map Reservation

Following is an example of using `vovresourcemgr` to place a reservation on a resource map. In this case, two of the resource maps called `License:spice` are reserved for user `john` for an interval of 4 hours. The resource map reservation will automatically expire after 4 hours.

```
% nc cmd vovresourcemgr reserve License:spice USER john 2 4h "library char"
```

Workaround for Misspelled Resource

Sometimes users submit jobs to Accelerator that request nonexistent resources, which causes the jobs to be queued indefinitely. Such jobs can be made to run by creating the missing resource, or by modifying the jobs to request the correct resources. The following example creates four temporary `License:sspice` resources that are mapped to the correct `License:spice` resource. *`License:sspice`* is an incorrect request - that resource does not exist. A temporary resource is created with that name that will be mapped to the correct resource, *`License:spice`*

```
% nc cmd vovresourcemgr create License:sspice -max 4 "License:spice"
```

Forgetting Unneeded Resource Maps

Continuing the above example - the temporary resource map may be removed after the malformed jobs have run. Or, you can just let it expire.

Note: There is no confirmation; the command acts immediately.

```
% nc cmd vovresourcemgr forget License:sspice
```

Reconciliation Daemon Configuration

Summary information for `vovreconciled`:

Working directory	`vnc.swd/vovreconciled`
Config file	`vnc.swd/vovreconciled/config.tcl`

The daemon `vovreconciled` periodically checks all running jobs and looks for resources that are either "Requested/Not Used" or "Not Requested/Used". When the daemon is reasonably sure about the resource mismatch, it will reconcile the grabbed resources list for the running jobs by calling `vtk_resourcemap_change_grab`.

```
vovreconciled: Usage Message

DESCRIPTION:
    vovreconciled is a daemon that detects "requested/not_used"
    resources for running jobs and removes them from the
    "grabbed resources" list after a certain amount of time,
    called "RevocationDelay"

    The RevocationDelay is set to the smallest value
    found in the following places:

    1. The property AGGRESSIVE_SCHEDULING_DELAY
                           (old) attached to the job class object, if defined
    2. The property REVOKE_DELAY
                           (new) attached to the job class object, if defined
    3. The property REVOKE_DELAY
                           attached to the resourceMap, if defined
    4. The value of RESD(revokeDelay), if defined.


    NO revocation is performed if any of the following are true
      1. If RevocationDelay  < 1
```

```
       2. If RevocationDelay  > 10000000
          (or 115d17h)
       3. If the resource is not derived from an external license.
       4. If the resource type is not "License" or a legal member
          of License
       5. If the number of revocations for a license on a job  >
          $RESD(maxRevokes)=50
       6. If the CHANGEGRAB property exceeds RESD(maxPropLength)
       7. The job is younger than the RESD(revokeDelay)

     The config.tcl file must exist but it can be empty.
     The config file allows the user to set some additional options

     RESD(maxRevokes)    N  N is the maximum number of times a license on a
                            job can be revoked.  Default is 50
                            To see the number of times a specific license has
                            been revoked for a given job, view the
                            REVCNT_<license> property that will exist on the
                            job, where <license> is the name of the specific
                            license of interest.
     RESD(maxPropLength) N  N is the number of characters the CHANGEGRAB
                            property can be.  Default 130000
     RESD(emailSkips)    N  1 enables/0 disables emailing the job owner and
                            optionally admins that a license could have been
                            revoked but was not, because the maximum number
                            of revoke was reached or the CHANGEGRAB property
                            is too long.  Default 1
     RESD(adminEmails)   S  A comma-separated string of userId's that are sent
                            emails on skips. Default ""
     RESD(revokeDelay)   T  number of seconds a job must be running before it
                            can be considered to have a license revoked.
                            Default 10000000 seconds or 115d17h
     RESD(loopTime)      T  How often to run the check on all jobs.
                            Default 30 seconds
     RESD(typeList)      S  A space separated list of license types that will
                            be handled by vovreconciled.  Default is {License}.
                            The types Limit, Policy, User, Group and Priority
                            are not supported and will be ignored. The type
                            License will be added if not specified.

OPTIONS:
     -v                      -- Increase verbosity.
     -h                      -- Show this help.
     -loop <TIMESPEC>        -- Default 30s
     -inert                  -- Run in inert mode where nothing changes
                                for the job.

EXAMPLES:
     % vovreconciled
     % vovreconciled -h
     % vovreconciled -loop 2m
     % vovreconciled -v
```

vovreconciled Operations

This daemon, if activated, runs continuously and checks all running jobs every 30 seconds. It looks at running jobs whose age is greater than the `RESD(revokeDelay)` or from the most recent resumption. If one of such jobs has an RNU resource (Requested but Not Used) for longer than a certain reconciliation time (Treconcile), then the job is flagged for reconciliation. If the condition persists for 3 consecutive cycles, then the resource is removed from the list of grabbed resources for the job.

The reconciliation time Treconcile is computed as the list of:

- The value of the property `REVOKE_DELAY` attached to the resource map (a TIMESPEC)
- The value of the property `REVOKE_DELAY` attached to the jobclass (a TIMESPEC)
- The value of `RESD(revokeDelay)` in `config.tcl`

Later on, if a job is found to use a resource that was previously reconciled away, that resource is restored to the job.

Override Delays

For each running job, `vovreconciled` looks at what it can do only after a certain amount of time has elapsed from the start of the job. This amount of time is called REVOKE DELAY and it is defined, by default, as the least of:

- The value of the property `REVOKE_DELAY` in the jobclass
- The value of the property `REVOKE_DELAY` in the resource map
- The global variable `RESD(revokeDelay)`

Some customers may want to change this behavior. A possibility is to override the procedure `VovGetRevokeDelay` in the file `config.tcl`. Both the default implementation of this procedure as well as an example for an override are shown below:

```
####
#### DEFAULT IMPLEMENTATION
####
proc VovGetRevokeDelay { jobClass res displayMessage } {
    global RESD
    set revokeDelayOld    [VovJobClassGetProperty $jobClass
 AGGRESSIVE_SCHEDULING_DELAY 10000000]
    set revokeDelayNew    [VovJobClassGetProperty $jobClass REVOKE_DELAY
   10000000]
    set revokeDelayResMap [VovResMapGetProperty   $res      REVOKE_DELAY
   10000000]

    set revokeDelay [FindLeastDelay $revokeDelayOld $revokeDelayNew
 $revokeDelayResMap $RESD(revokeDelay)]

    if { $displayMessage > 0 } {
        set msg "    FindLeastDelay\n"
        append msg "\tAggressiveClass:        $revokeDelayOld\n"
        append msg "\tREVOKE_DELAY in class:  $revokeDelayNew (jobclass=$jobClass)\n"
        append msg "\tREVOKE_DELAY in ResMap: $revokeDelayResMap (resource=$res)\n"
        append msg "\tGlobal:                 $RESD(revokeDelay)\n"
        append msg "\tResult revokeDelay:     $revokeDelay"
        VovMessage $msg 5
    }

    return $revokeDelay
}
```

```
####
#### EXAMPLE OVERRIDE (to be implemented in vovreconciled/config.tcl
####
proc VovGetRevokeDelay { jobClass res displayMessage } {
    global RESD
    set revokeDelayClass    [VovJobClassGetProperty $jobClass REVOKE_DELAY 10000000]
    if { $revokeDelayClass != 1000000 } { return $revokeDelayClass }

    set revokeDelayResMap [VovResMapGetProperty   $res      REVOKE_DELAY 10000000]
    set revokeDelay [FindLeastDelay $revokeDelayResMap $RESD(revokeDelay)]

    return $revokeDelay
}
```

Add Resources

Generic resources are added to Altair Accelerator via the `vtk_resourcemap_set` procedure call:

```
vtk_resourcemap_set <name> <quantity> [map]
```

Below are two examples:

```
vtk_resourcemap_set myres 2
vtk_resourcemap_set myunlimitedres UNLIMITED
```

Example: Node Locked License

In the scenario of this example, a license does not utilize FlexNet Publisher or another dynamic license management solution, but does require a tool to run only on one specific host. In this example, the tool is `spice`, the host is `pluto`, and the license is for two concurrent instances of spice. Following are the steps to correctly handle this constraint:

1. Choose a name (in the form `name` or `type:name`) to represent the node locked resource (such as `License:spice_pluto`) and a name to be announced to the users (such as `License:spice`). In this way, it is hidden to the users that the spice license is locked to a given node. Also, if there is an upgrade to a floating license or multiple node-locked licenses, that can be carried out without having to announce it to the users.
2. Add the following lines to the `resources.tcl` file:

   ```
   vtk_resourcemap_set License:spice UNLIMITED License:spice_pluto
   vtk_resourcemap_set License:spice_pluto 2 pluto
   ```

   ```
   % nc cmd vovproject reread
   ```
3. Let the job declare that it requires the resource `License:spice`; use option -r in `nc run` as shown below:

   ```
   % nc run -r License:spice -- spice -i chip.spi
   ```

License-based Resources

This section describes the Accelerator interface to Monitor, an application that monitors license servers and makes the in-use information available to Accelerator.

Many software products in Electronic Design Automation use FlexNet Publisher licensing by Flexera or other vendor-specific license mechanisms.

Monitor provides a centralized interface between vendors' license daemons and Accelerator. The benefits of this approach are:

- Faster response, with reduced load on the license daemons
- Improved consistency of license in-use information

- Individual projects need not be concerned with license details
- Browser-based interface to access information about licenses in-use

Accelerator is shipped with an edition of Monitor that is licensed to monitor **current license activity only**, and provide that information to Accelerator. This edition is referred to as LMS (Monitor Small).

Monitor can also store, report, and graph historical license usage and denial activity.

Note: A full Monitor license from Altair is required to enable historical and denial information.

Configuration

Refer to *Installation Guide* for details about installing and configuring the Monitor product.

By default, Accelerator assumes that the Monitor server is running on the same machine as the Accelerator server on port 5555, under the VOV project name of `licmon`. If this is not the case, the following statements will need to be placed at the top of the `resources.tcl` file to inform the Accelerator resource daemon of Monitor's details:

```
# Enable Accelerator to see Monitor.
# This is a security feature.
set lm(ssl) true

# Fragment of resources.tcl
# This is the default configuration.
set LM(flexlmd) localhost:5555
set LM(licmon)  licmon

# You may want to specify a different Monitor
# running on a different host, a different port
# with a different name.
set LM(flexlmd) someOtherHost:25555
set LM(licmon)  licmonTest
```

Accelerator uses `vtk_flexlm_monitor` and `vtk_flexlm_monitor_all` statements in its `resources.tcl` configuration file in order to communicate with the Monitor product to obtain license utilization information.

The procedure `vtk_flexlm_monitor` takes from one to three arguments:

1. **feature** - the name of the license feature. This is the name of any feature monitored by Monitor. The name may include a specific Monitor tag. If no tag is specified, the cumulative count for all tags containing the feature will be used.
2. **resource** - the Accelerator resource name. If the resource name is specified, then this name is the actual resource name used. If the resource name is not specified, the name defaults to License:<feature>.
3. **map** - an optional resource to which the resource should be mapped.

For example:

```
vtk_flexlm_monitor Design-Compiler

# Pick up a specific tag for Design-Compiler
# This maps the feature Design-Compiler to License:Design-Compiler
vtk_flexlm_monitor SNPS/Design-Compiler

# Specify a different resource name
vtk_flexlm_monitor SNPS/Design-Compiler License:dc

# Additionally specify that all jobs using License:dc need to also use
# a linux resource.
```

```
vtk_flexlm_monitor SNPS/Design-Compiler License:dc linux
```

As an alternative to individually calling `vtk_flexlm_monitor` for each feature to monitor, `vtk_flexlm_monitor_all` can be used. The default behavior of this procedure is to create resources for all features that are known to Monitor. This procedure has a number of options:

Table 2: vtk_flexlm_monitor_all Options

Option	Description
`-daemon host:port`	Specifies the host and TCP/IP port where Monitor is to be contacted to get license data via HTTP. If not given, the procedure reads the `info.tcl` in `$VOVDIR/../../licmon/licmon.swd/vovlmd` to locate the daemon. If the daemon cannot be located, the procedure returns 0.
`-tag tag`	Only use features from the source having this tag. The default is to use features from all license sources. This option may be repeated.
`-tags list_of_tags`	Same as above, only with a list of tags to be included. This option may be repeated.
`-I regexp`	Appends `regexp` to the list of regexps evaluated for feature inclusion. If no -I options are given, all features from the given source are included.
`-X regexp`	Appends `regexp` to the list of `regexps` evaluated for feature exclusion. If no -X options are given, no features are excluded.
`-It regexp`	Appends `regexp` to the list of `regexps` evaluated for tag inclusion. If no -It options are given, all tags from the given source are included.
`-Xt regexp`	Appends `regexp` to the list of `regexps` evaluated for tag exclusion. If no -Xt options are given, no tags are excluded.
`-fproc fproc-name`	Specifies the name of the user-defined Tcl filter procedure to apply to the features. The default is `vtk_flexlm_monitor_filter {tag feature}`. This procedure takes two parameters, a tag and a feature name, and returns a Boolean, where 1 means to include the feature, and 0 means to exclude it. The default procedure always returns 1.
`-rproc rproc-name`	Specifies the name of the user-defined Tcl procedure that returns the resource map name for a feature. The default is `vtk_flexlm_monitor_resname {tag feature}`. This procedure takes two parameters, a tag and a feature name, and returns a string, which is the VOV resource name for the feature. The default procedure prepends `License:` to the feature name.
`-mproc mproc-name`	Specifies the name of the user-defined Tcl procedure that returns the right-hand-side of the resource map name for a feature. The default is `vtk_flexlm_monitor_mapname {tag feature}`. This procedure takes two parameters, a tag and a feature name, and returns a string, which is the

Option	Description
	right-hand-side of the VOV resource map for the feature. The default procedure returns "", which means no mapping.

`-order list_of_tags`	This options controls the order in which multiple tags (think of "license files") are listed in Accelerator. This applies when there are multiple tags for the same feature. For example, if the feature `abc` is in two tags, `SNPS_BLR/abc` and `SNPS_US/abc`, you will get a resource map called `License:abc`, which is the OR of the two resource maps associated to each feature, as in
	`License:abc #  License:SNPS_US_abc  OR License:SNPS_BLR_abc`
	However, it is also possible to get the following map, with inverted order of the tags:
	`License:abc #  License:SNPS_BRL_abc  OR License:SNPS_US_abc`
	The -order option allow controlling which map will be used. For example: `-order "SNPS_BLR SNPS_US"` will use `SNPS_BLR` before `SNPS_US`. The option can be repeated multiple times.

This procedure can make getting started with license management much easier and faster. Use this procedure with caution, especially if it is used with any user-defined Tcl procedures. Place the procedure definitions in the `resources.tcl` file, and be sure to specify the names carefully.

More examples for vtk_flexlm_monitor and vtk_flexlm_monitor_all

Fragment of `resources.tcl` file:

```
# Monitor the feature PrimeTime
vtk_flexlm_monitor  PrimeTime

# Monitor the feature PrimeTime, control the order of the tags
vtk_flexlm_monitor -order "SNPS_US SNPS_CH SNPS_FR" PrimeTime

# Monitor the feature  for Design-Compiler.  Internally VOV
# uses the token dc_shell_license.
vtk_flexlm_monitor  Design-Compiler  dc_shell_license

# The FlexNet Publisher feature "pathmill" maps to the VOV resource "pathmill"
# which, in turn, maps to the resource "sun7"
vtk_flexlm_monitor  pathmill         pathmill         sun7
```

Example : Monitoring all features

```
#
# Monitor all the features gathered by the Monitor at grove:5555#
vtk_flexlm_monitor_all -daemon grove:5555

#
```

```
# Monitor all the features gathered by the Monitor,
# control the order of the tags
#
vtk_flexlm_monitor_all -order "SNPS_US SNPS_FR" -order "MGC_US MGC_FR MGC_UK"
```

Example: Monitor some features, user-defined map proc

```
#
# Monitor some features gathered by the LicenseMonitor at grove:5555
## Make all the Fintronic tools go to the finfarm vovtaskers
# by defining a RHS-procedure

proc fintronic_mapname {tag feature} {
    set rval ""
    if { [regexp {^fin} $feature] } {
        set rval "finfarm"
    }
    return $rval
}

# Only monitor features from tags REAL and Altair Accelerator products
vtk_flexlm_monitor_all -daemon grove:5005 -tag REAL -tag RTDA  -rproc
 fintronic_mapname
```

In the above example, features related to the Fintronic Finsim Verilog simulator, recognized by the feature name beginning with 'fin', will have a right-hand-side of finfarm' added to the resource map, so that such jobs will go to vovtaskers offering the 'finfarm' resource.

License Sharing Support

With *license sharing*, the same license can be shared by multiple jobs. Depending on the policy selected by the vendor, there are licenses that can be shared among different jobs provided that some characteristics are the same, for example the execution host, the user or the display.

Host + User Sharing

With this type of sharing, if a user is already running a job on a machine, that same user is allowed to run any number of jobs on the same machine without consuming another license. This can be very advantageous for the end user especially when using machines with 8 or more CPUS.

This configuration is supported with two steps:

1. Define a jobclass for the jobs that should take advantage of license sharing.

   ```
   # This is file   vnc.swd/jobclass/spectrerf.tcl
   set classDescription "SpectreRF class"
   set VOV_JOB_DESC(resources) "Share:spectrerf_[vtk_logname] OR License:spectreRF"
   ```

2. Add a procedure to `resources.tcl` to dynamically create the resource maps represent what licenses can be shared.

   ```
   source $env(VOVDIR)/tcl/vtcl/vovsharedresourcesproc.tcl
   # Configuration of the procedure for shared licenses.
   SHRaddRule spectrerf  Share:spectrerf_@USER@
   ```

Automatic Setting of LM_LICENSE_FILE

The capability described in this section is useful when the same license is provided by more than license server.

Problem Description

In this example, there is a license called "spice" that is provided by two daemons:

- The daemon `1234@lan.company.com` has 3 licenses for LAN use
- The daemon `9999@wan.company.com` has 10 licenses for WAN use

In a typical setup, the scheduler is told to use first the LAN licenses and then the WAN licenses, and this is easily accomplished by setting the environment variable LM_LICENSE_FILE to the following value:

```
setenv LM_LICENSE_FILE 1234@lan:9999@wan
```

For this example, the scheduler has decided that a given job should use a WAN license. When the job is launched, however a LAN license has just become available, and the tool actually checks out a LAN license instead of a WAN license.

This causes a problem: the *double counting* of the licenses in use:

- One WAN license is considered in use by the scheduler (requested but not yet used).
- One LAN license is actually in use (not requested but currently used).

In this situation, it would be desirable for the scheduler to decide that a job should use a WAN licenses and the value of LM_LICENSE_FILE should be only 9999@wan. However, if the scheduler chooses the LAN license, then the value of LM_LICENSE_FILE should be only 1234@lan. This arrangement works best if the license checkouts need to be queued by FlexNet Publisher.

The farm would work more efficiently if the value of LM_LICENSE_FILE were controlled at execution time on the basis of the resources grabbed for each individual job. A solution is described in the following section.

Solution: set VOV_LM_VARNAMES

If the environment variable VOV_LM_VARNAMES is set to a comma-separated list of names of environment variables, the "vw" wrapper will automatically set or prepend to each variable in the list with the list of license daemons that are associated with the resources that have been grabbed for the job.

Important: The functionality activated by setting the VOV_LM_VARNAMES environment variable is implemented by the VOV wrapper program `vw2` and its aliases. Interactive jobs that have a PTY do not use this wrapper, so the VOV_LM_VARNAMES capability is only available for batch jobs.

A typical value of VOV_LM_VARNAMES is LM_LICENSE_FILE, but sometimes it is necessary to control other variables such as CDS_LICENSE_FILE (for Cadence tools). At times, it may be desirable to set VOV_LM_VARNAMES to another variable name which is then processed by one of the wrapper scripts.

There may be cases where it is desired to exclusively set license variables instead of prepending to existing values. To accomplish this, set the VOV_UNSET_VARNAMES environment to a comma-separated list of environment variables to unset and they will be unset before the `vw` wrapper populates them.

Following is an example of using this variable:

```
setenv VOV_LM_VARNAMES LM_LICENSE_FILE
```

More examples are shown below:

```
# Examples of uses of VOV_LM_VARNAMES
setenv VOV_LM_VARNAMES CDS_LICENSE_FILE
setenv VOV_LM_VARNAMES CDS_LICENSE_FILE,LM_LICENSE_FILE
setenv VOV_LM_VARNAMES MY_VAR
```

Job submission with VOV_LM_VARNAMES

In the example below, a feature called "abc" is provided by two FlexNet Publisher daemons, `1234@lan` and `9999@wan`. Both daemons are being monitored by Monitor, which uses tags "LAN" and "WAN" respectively. In turn, Accelerator, which is connected to Monitor, is aware of those two resources and has a resource map called "License:abc" which is the sum of "License:LAN_abc" and "License:WAN_abc". In Accelerator, License:abc maps to "License:LAN_abc OR License:WAN_abc". You want to push the utilization of the abc resource to the edge of saturation, so instead of defining LM_LICENSE_FILE to the value "1234@lan:9999@wan," let Accelerator control the value of LM_LICENSE_FILE by having an environment for the tool execution where the variable VOV_LM_VARNAMES is set to LM_LICENSE_FILE. Call the environment "MYENV". In the file `MYENV.start.csh`, you will have a line that says:

```
setenv VOV_LM_VARNAMES LM_LICENSE_FILE
```

To submit a job that uses "abc", enter:

```
% ves MYENV
% nc run -r License:abc -- abcTool ...
```

In cases in which you absolutely want to use one of the daemons, specify the appropriate resource:

```
% nc run -e MYENV -r License:WAN_abc -- abcTool ...
```

LM_VAR_NAME - deprecated

Note: Using LM_VAR_NAME is not recommended; it has been deprecated.

For backwards compatibility, the variable LM_VAR_NAME could be used to set the name of a single environment variable.

vovgetflexlmdaemons

In some situations, using other methods to set the LM_VAR_NAME variable on the fly may be preferred. The utility `vovgetflexlmdaemons` can be used to locate the daemons that need to be used based on the resources grabbed by Accelerator at dispatch time.

```
vovgetflexlmdaemons: Usage Message

  DESCRIPTION:
      Processes VOV_UNSET_VARNAMES, VOV_LM_VARNAMES, and/or LM_FILE_VAR
      environment variables and generates script for the specified shell that
      will unset the variables listed in VOV_UNSET_VARNAMES and then set
      LM_LICENSE_FILE to match the resources used by the job whose ID is in
      VOV_JOBID or those provided on the command line.
```

```
USAGE:
    % vovgetflexlmdaemons [OPTIONS] [RESOURCES]

OPTIONS:
   -v               -- Increase verbosity
   -h               -- This help
   -sh              -- Specify the shell for the output script, default is csh
   -bash
   -ksh
   -csh
   -tcsh

EXAMPLES:
    % vovgetflexlmdaemons -h
    % vovgetflexlmdaemons
    % vovgetflexlmdaemons -bash
    % vovgetflexlmdaemons -bash License:lic_drc

EXAMPLE OUTPUT:
   ### With -bash option.
   LM_LICENSE_FILE=6306@mac05;
   export LM_LICENSE_FILE;

   ### With -csh option.
   setenv LM_LICENSE_FILE 6306@mac05

USAGE IN SCRIPTS:
    eval `vovgetflexlmdaemons`;
```

Limit Users

Limiting users is typically not the best choice, as alternative approaches may be used to enforce policies without incurring low resource utilization issues. However, in some cases, limiting users may be the best solution and this section discusses several ways in which an administrator can limit the number of jobs that a user can run.

Control the User:x Resource

For every user 'x' there is a resource called "User:x" that controls the overall number of jobs that the user can run. The default value is "unlimited" and an administrator can set the resource to a different value with `vtk_resourcemap_set`, as in the following examples:

```
vtk_resourcemap_set User:john 5
vtk_resourcemap_set User:mary unlimited
vtk_resourcemap_set User:phil 0
vtk_resourcemap_set User:hoag -max 2 -map "linux64 RAM/200"
```

These VTK calls can go into the `resources.tcl` file or they could be executed directly with `vovsh -x ...`:

```
% nc cmd vovsh -x 'vtk_resourcemap_set User:tarzan 88'
```

Similarly, you can control the Group:* resource to limit the jobs within a FairShare group or the Priority:* resource to limit all jobs that use certain ranges of priorities. The use of these limits is highly discouraged.

The default value of the User:* resource is controlled by the server configuration parameter *`resMapUserDefault`*, which can be set in the `policy.tcl` file, as in this example:

```
# Fragment of policy.tcl file.
set config(resMapUserDefault) 30
```

The configuration applies to all new resources that are created after the default has been changed.

Control the Tool:* Resource

While the resources mentioned in the previous section are always used, the resource Tool:x is only used if it is defined for a given tool. The "tool for a job" is defined as the tail of the command line argument that is not a known VOV wrapper. For example, for the command `vw /bin/cp aa bb`, the tool is the string `cp`.

For example, if you want to limit the number of `cp` jobs that are executed at any one time, or you want to route them to a specific set of machines, you can use the resource map "Tool:cp", as in:

```
vtk_resourcemap_set Tool:cp -max 3
```

Limits in Jobclasses

If you are using a jobclass, it is easy to define limits for both the jobclass and also for the users of that jobclass. This is best shown with this complete example taken from `$VOVDIR/etc/jobclass/examples/hsim.tcl`:

File: `hsim.tcl`

```
set classDescription "A template for an hsim class"
set classEditable     1
lappend VOV_JOB_DESC(resources) License:hsim
lappend VOV_JOB_DESC(resources) Limit:q_hsim_@USER@
lappend VOV_JOB_DESC(resources) Limit:r_hsim_@GROUP@
lappend VOV_JOB_DESC(resources) CPUS/1 percent/1 RAM/20

proc initJobClass {} {
    # Executed by vovresourced at startup.
    # vtk_resourcemap_set License:hsim 8
    # vtk_flexlm_monitor hsim License:hsim

    # Revoke requested/not-used resources after 2 minutes.
    vtk_jobclass_set_revocation_delay "hsim" 2m

    # Warn after 2h, kill after 4h of idleness.
    vtk_jobclass_set_idle_delays    "hsim" 2h 4h

    vtk_jobclass_set_max_reschedule "hsim"  2

    TIMEVAR hsim {
 Fri,Sat,Sun {
     vtk_resourcemap_set_limit Limit:q_hsim_@USER@  3
     vtk_resourcemap_set_limit Limit:r_hsim_@GROUP@ 5
 }
 20:30-24:00,0:00-5:30 {
     vtk_resourcemap_set_limit Limit:q_hsim_@USER@  1
     vtk_resourcemap_set_limit Limit:r_hsim_@GROUP@ 2
 }
 default {
     vtk_resourcemap_set_limit Limit:q_hsim_@USER@  1
     vtk_resourcemap_set_limit Limit:r_hsim_@GROUP@ 2
 }
```

```
    }
}
```

Self Limiting

Some users may like to self limit the jobs that can be executed concurrently by using the option -limit in `nc run`:

```
% nc run -limit 3 -f listOfJobs
```

License Overbooking

Advantages of Overbooking

If there are 10 licenses of a simulator, the scheduler (that is, Accelerator) dispatches 10 jobs using those licenses. However, many jobs do not use the licenses for 100% of their lifespan. If one checks how many licenses are checked out at any one time, for example with `lmstat`, one may find out that occasionally there are less that 10 licenses in use. Experienced users who look directly at the license daemon statistics may wonder why there are jobs in the queue while licenses are available. From Accelerator's point of view, those licenses are not available because they are reserved for those running jobs, which may check out the licenses at any time.

This problem is greatly amplified if instead of 10 licenses there are 1,000 licenses. In such a case, you may notice that 1,000 licenses are never fully checked out, although there are 1,000 running jobs at all times. For example, one customer had about 2,000 licenses of a simulator and even with 2,000 running jobs, only 1,750 to 1,900 licenses were checked out.

Those unused licenses create an opportunity to run more jobs than licenses, which is accomplished by "license overbooking".

Activate Overbooking

For overbooking to work well, you have to activate *vendor-queueing* for the license. If a job cannot find a license, it waits for the license to become available instead of failing. Allocator and Accelerator are capable of overbooking licenses so that only a small number of running jobs are waiting for a license.

Note: The activation of vendor queueing is application dependent.

Overbooking Operation

The normal approach for managing licenses in Accelerator is setting the maximum number of licenses equal to the number available from the license daemons. Allocator and/or Accelerator issue jobs using this fixed number as an upper bound. In overbooking, tell Allocator and Accelerator to keep issuing jobs until it sees the actual license count (from the license daemon) fully depleted. The overbooking function acts as a classic control loop:

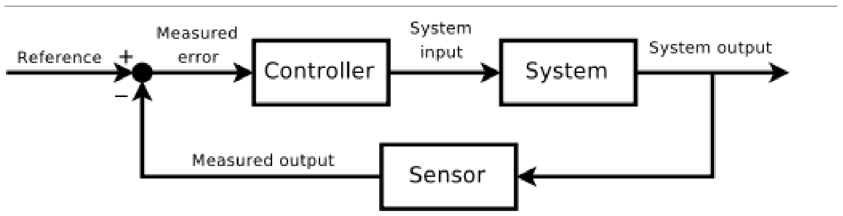


Figure 16:

The **Reference** is the number of licenses available, the **Measured output** is the number of licenses in use and the **Measured error** is the difference in the two. The **Controller** is the actual overbooking function that converts the license available into the number of jobs to issue - this is the **System input**. The **System output** is the launched jobs and the **Sensor** is the Monitor.

The Overbooking function (Controller) is `vtk_flexlm_overbook` for `vovresourced` and `LA::AddResource` in Allocator. These procedures have a number of tuning parameters for overbooking.

License Overbooking in vovresourced

The Overbooking function (Controller) in `vovresourced` is called `vtk_flexlm_overbook` and has a number of tuning parameters:

-thresh real
: This is the threshold at which overbooking becomes active. For example if the value is set to 0.8 it means that when the job count reaches 80% of the allowed total, overbooking starts increasing the number of submitted jobs. The default value for the parameter is 0.9.

-factor real
: This is number is used to scale the Measured Error into the additional number of jobs to submit. Common values are 0.8. The default value for this parameter is 1.0.

-headroom int
: When the license in-use counts is greater than the maximum number of licenses less the headroom, the overbooking quantity is throttled. Negative headroom values are often used. The default value of headroom is 0.

-queued int
: When Allocator sees this number of licenses queued (for example, in vendor-queuing) then the overbooking quantity is throttled. Generally this should be a small number. The default value is 1.

-lowpass int
: To smooth the effect of overbooking, the actual correction is low-pass filtered. This option controls the delay of the filter. The larger the number, the longer the delay and the smoother the correction. The default value is 8.

-enable BOOL
: Simple way to enable/disable an overbooking rule.

-verbose BOOL
: Increase verbosity.

-v

Same as -verbose 1.

Tuning Overbooking

The optimal values for the overbooking parameters are workload specific. While the default values work in many cases, it is worthwhile to review the overbooking operation. Use the -verbose 1 option to monitor the behavior of the overbooking routines in the resource daemon log file. The goal should be to increase the license utilization close to 100% but without pushing too many jobs into vendor queueing. Vendor queuing implies that a hardware slot is taken but the job is stalled waiting for that license; normally a few slots being idle for a few seconds is a reasonable tradeoff for high license utilization. However, administrators should be aware that having too many jobs in vendor queueing can cause the license daemons to stall, dramatically reducing overall throughput.

If your workload ends up pushing too many jobs into vendor queuing before backing off, even with a small value of -queued then consider reducing the factor from 1.0 to 0.9 or lower, and increasing the headroom from 0 to 2-5% of the maximum license quantity.

If your workload struggles to get any vendor queuing, it may be because the threshold is not reached or maintained - particularly when the new total (maximum number of jobs) has been increased. Reducing the threshold to 0.8 or 0.7 will enable overbooking to continue to be active. For a large number of short duration jobs, there can be many jobs "in-flight": jobs for which license matching by Accelerator has not yet occurred. In these cases, it may make sense to increase -factor N to about 1.2 and have a negative value of headroom equal to 2-5% of the total license count.

While overbooking exhibits the self compensation characteristics expected from a control loop, tuning the parameters is often worthwhile for optimal usage.

Example of Overbooking in vovresourced

```
# Fragment of resources.tcl
# Example of overbooking of hspice.

vtk_flexlm_monitor_all   ; ## This must be called before vtk_flexlm_overbook.

#
# Overbook the feature hspice up to the point where there are 5 jobs in vendor-queue.
vtk_flexlm_overbook hspice -queued 5

## All valid options.
vtk_flexlm_overbook myspice -thresh 0.9 -factor 1.0 \
     -headroom 2 -queued 3 \
     -enable 1 \
```

After changing `resources.tcl`, remember to restart the `vovresourced` daemon:

```
% ncmgr reset
```

Resource Management with RDS

This is a usage guide for the Resource Data Service (RDS), an alternative service for managing License resources and License-first scheduling in Accelerator.

Overview

Accelerator and Monitor run in "classic resource management" mode by default. In classic mode, the `vovresourced` daemon manages licenses and other resources as configured in the `resources.tcl` configuration file.

A second mode for managing resources, termed "RDS mode", may be chosen and configured as described in this section of the manual. In RDS mode, resource management is performed by an RDS thread instead of the vovresourced daemon, and the configuration is described in the `resources.cfg` file instead of `resources.tcl`.

Resource Management Background

The Resource Data Service (RDS) is a new solution to manage job resources associated with limits or software licenses. The Accelerator and Monitor products work together to provide advanced software licensing feature scheduling with flexible and powerful capabilities to adapt license resource allocation to asynchronous and out-of-queue license usage and to variable license usage across a job's lifespan.

When an Accelerator batch job is submitted, a list of *resource maps*, also known as *resources*, are specified to guide scheduling. An important type of resource is a *license-based resource*, which is managed by Accelerator resource management with input from Monitor, which actively monitors actual license usage in real time. The classic service within Accelerator that provided this capability is the `vovresrouced` daemon, which is active by default.

Configuring RDS

The RDS configuration file is located at `<SWD>/resources.cfg`. This file sets up RDS with configuration information which includes connections to license monitoring and definition of limit resource maps, license resource maps, and mapping resource maps. In summary, the `resources.cfg` file will give you the ability to configure what classic resource management has configured in the Tcl based `resources.tcl` file. The Attribute Value Stream (AVS) format and schema for RDS `resources.cfg` is described Configuration File Format.

Important configuration notes:

- For advanced license scheduling, Accelerator needs to link to at least one running RDS-enabled Monitor instance. See Server Configuration Parameter to Configure Accelerator for RDS.
- The LICENSE_MONITORS attribute in `resources.cfg` specifies the Monitor instance that will be linked to Accelerator.
- A common way to specify a Monitor instance in Accelerator's `resources.cfg` file is with the following simple `resources.cfg` content:

```
{
LICENSE_MONITORS = [ { NAME = "monitor_name" } ],
RESOURCE_MAPS = [ { FEATURES = "*" ]
}
```

This specification will often be enough. However, if Monitor is not using the same registry as Accelerator, then Monitor needs the EVENT_PORT parameter declared. The selected/specified port will be placed in the LM's registry file so that normally only the LM name is needed to establish the RDS to LM connection. For example:

```
% lmmgr start -eventport <port number>
```

- The RESOURCE_MAPS attribute structure in `resources.cfg` uses explicit feature names and wildcard patterns to specify exactly what license features to import into Accelerator for the purpose of advanced license scheduling.
- Full details for configuration syntax of LICENSE_MONITORS and RESOURCE_MAPS in `resources.cfg` are shown in Configuration File Format.

Note: For a full description of the AVS data language, see AVS Syntax.

Resources.cfg

The Resource Data Service (RDS) capability is Accelerator's next-generation resource management. The new RDS service is configured by a new configuration file --`resources.cfg`-- in the Server Working Directory. The new configuration file uses an Attribute Value Stream (AVS) syntax, a JSON-like syntax which is quite different from the Tcl format of the `resources.tcl` file used to configure classical Accelerator resource management. This document will guide you in configuring RDS and in translating an existing `resources.tcl` file into a properly formatted `resources.cfg` file.

Server Configuration Parameter to Configure Accelerator for RDS

When you have an RDS-enabled Monitor instance running, you are ready to configure Accelerator for RDS.

Within Accelerator, when the `rds.enable` server configuration parameter is set to "1", Accelerator will be RDS-enabled, and a new server thread named RDS will activate in place of the classic `vovresourced` daemon. The RDS thread will process the `resources.cfg` file, which can be written according to the following instructions.

Edit/create the `SWD/resources.cfg` file. Substitute your LM name for `licmon` below if different:

```
{
    LOG_LEVEL = 5,
    LICENSE_MONITORS = [
      { NAME = "licmon", }
 ],
RESOURCE_MAPS = [
     { FEATURES = "*" },       # monitor all features

     { TYPE = "Priority", NAME = "low" },
     { TYPE = "Priority", NAME = "normal" },
     { TYPE = "Priority", NAME = "high" },
     { TYPE = "Priority", NAME = "top" },
     { NAME = "diskio" },
  ]
}
```

See Configuration File Examples for more complex `resources.cfg` options.

Start Accelerator as normal using `ncmgr start`. It will locate the LM ports using the licmon name in the registry.

Configuration File Format

The RDS config file consists of a single object in Attribute-Value Stream (AVS) format. AVS objects begin with an open curly brace '{' and end with a close curly brace '}'. Array attributes are expressed as a comma separated list inside square brackets []. In most cases array attributes with a single element may be expressed as the single element (TAGS = foo is equivalent to TAGS = [foo]). String type attributes are double quoted.

There are 3 types of attributes in the top level AVS object:

- RDS Global Attributes
- The LICENSE_MONTIORS array
- The RESOURCE_MAPS array

The objects within the RESOURCE_MAPS array may be one of the following object types. The first is identified by a NAME attribute, and is called a *non-license resource definition.* The second object type is identified by the presence of a FEATURE attribute and is called a *license feature resource creation rule*.

Flag values are generally indicated as on/true simply by their presence but may alternately be indicated on or explicitly off by setting the attribute to 0/1, off/on, false/true.

RDS Global Attributes (aka - Top/Daemon Level Attributes)

Attribute	Description	Default
DROPPED_RESMAP_DELETE_DELAY	When RDS is initializing, the amount of time in seconds that it will wait before deleting feature resource maps that are no longer represented in the license monitor data. This allows for delays in license monitor # license server recovery.	900
FLAGS	Array of flag overrides.	
HEARTBEAT_THRESHOLD	How long to wait for a missing LM heartbeat before signaling a resync, in seconds.	5 minutes
INTERNAL_HEARTBEAT_THRESHOL	(Internal for testing) overrides above with unlimited values.	
LICENSE_MONITORS	LM project name, or an LM object.	
LOG_LEVEL	The RDS logging level 0-7: Fatal=1	4

Attribute	Description	Default

	Error=2
	Warning=3
	Info=4
	Verbose=5
	Debug=6
	Engineering=7	

MATCHING_PERIOD	How often to perform matching in seconds.	60
MAX_ORS_IN_RESOURCE_SUM	Max number of OR operators allowed in a sum resource using legacy OR operators.	5
RESOURCE_MAPS	Array of resource map and/or feature rule objects.	
RM_RANK_DEFAULT	The default RANK of non-license resource maps.	3

Non-License Resource Definition Object Attributes

Attributes for regular resource maps not related to the monitoring of LM features. Most of these attributes correspond to the same named fields in the vov resource maps meta data. Usually each non-licensed ResourceMap object in the `resource.cfg` file will create one resource map. See examples section below for more details.

Attribute	Description	Default
MAP	The MAP field of the resource map, string	none, ""
NAME	The NAME of the resource map, string	None (required)
NOLOG	flag, sets the NOLOG field in the resource map	off
OWNER	The OWNER of the resource map. string	"(rds)"
RANK	The RANK of the resource map, integer	3
SPECIAL	Special exceptions to user specific resource maps, object	none
TOTAL	The TOTAL/max available for this resource, integer or UNLIMITED	UNLIMITED

Attribute	Description	Default
TYPE	The TYPE of the resource map, string	none, ""

License Feature Resource Rule Object Attributes

Feature rules specify which tag/feature pairs are to be monitored, how/if they are matched, and how/if they are grouped into sum resources. When a specific tag/feature pair is reported by the license monitor, the feature rules are consulted in order to determine the behavior for that tag/feature. Once a feature rule fires, no further rules will apply to that tag/feature. Generally there will be a single sum resource map per rule per matching feature. See Configuration File Examples for more details.

Attribute	Description	Default
EXCLUDE_FEATURES	Exceptions to FEATURES	None
EXCLUDE_TAGS	Exceptions to TAGS	None
FEATURES	Array of string patterns naming LM features to be matched by this rule. The array is optional for a single pattern.	None (required)
LEGACY_SUM_RESMAPS	Flag that indicates the legacy OR operator should be used in the sum resource map's MAP field instead of the comma operator.	off
MAP	A resource map MAP expression. It is applied to the leaf resource maps created by a FEATURE rule.	
ORDER	The order in which to place the various tags in the sum resourcemap map field. The default is TAGORDER which indicates the same order as in the TAGS attribute. Alternately, REVERSE_TAGORDER, or a partial explicit array of tags may be specified.	TAGORDER
SUM	Array of Custom Sum Resource Rule objects. See description below.	License:@FEATURE@ mapped to all matching tags in TAGORDER
TAGS	Array of string patterns naming Tags to be matched by this rule. The array is optional for a single pattern	"*"
TOTAL	Overrides the total available for the corresponding feature resourcemap rather than using the total available from LM.	

Attribute	Description	Default
	This value will also be used in the sum resourcemap total calculation unless that is also overridden.	

The following are flag attributes for feature rules. Their presence in the rule signifies that they are "On", their absence signifies "Off". Alternately they may be expressed as <name> = 1, or <name> = 0 for on/off respectively. Default: all are off.

Flags	Description
DONOTSHARE	Not used by RDS but it signifies something to LA ?
EXCLUDE	Do not monitor tag/features matched by this rule (stops further rule evaluation for a matching tag/feature).
LEGACY_SUM_RESMAPS	Use the legacy OR operator between members of the sum resource map in stead of the comma operator.
NOALSOMATCH	Do not calculate additional matches for best matched jobs.
NOMATCH	Do not perform matching.
NOOOQ	Do not calculate Out-Of-Queue values, aka VovResourceMap "OTHERS" field.
NORECENT	Do not perform matching against recently finished jobs.
NOSUM	Do not create a sum resource for this rule.
NOVQASOOQ	Do not count vendor-queued licenses as OOQ.
SUM1	Create a sum resource even if the rule uniquely identifies a single feature.

Custom Sum Resource Rule Object Attributes

By default, each feature rule will result in the creation of a single sum resourcemap for each matching feature containing all the matching tags. SUM-affecting attributes contained in the feature rule will be inherited by the "default" sum rule implicit in each feature rule. Grouping of tag/features into sum resources can generally be achieved by creating separate feature rules for each group. An explicit array of Sum resourcemap rule objects may be specified when multiple sums with overlapping tags is desired, or in order to override the sum's total or map fields.

Attribute	Description	Default
EXCLUDE_TAGS	Array of tag patterns to exclude from this sum	Inherited
LEGACY_SUM_RESMAPS	Use the legacy OR operator between members of the sum resource map in stead of the comma operator	Inherited
MAP	Override the map field	Auto maps each contained tag/feature in specified order
NAME	The NAME of the sum resource to be created	"@FEATURE@"
ORDER	The order for the tags in the map	Inherited
TAGS	Array of tag patterns to include in this sum	Inherited
TOTAL	Overrides the total number available	Auto sums the totals from tag/features contained
TYPE	The TYPE of the sum resource to be created	Inherited

Configuration File Examples

Resource map definitions and feature rules may now be intermixed to support common usage in customer examples where related entities are grouped together.

```
########################
# new default
########################
{
  #LICENSE_MONITORS = { NAME = "licmon" },
  RESOURCE_MAPS = [
    #{ FEATURES = ALL }, # monitors all features/tags
    #{ FEATURES = "feature1", TAGS = "tag1" }, # monitors tag1/feature1
    # monitor all combinations of feature2, feature3, tag2, tag3
    #{ FEATURES = [ "feature2", "feature3"], TAGS = [ "tag2", "tag3" ] },
    { TYPE = "Priority", NAME = "low" },
    { TYPE = "Priority", NAME = "normal" },
    { TYPE = "Priority", NAME = "high" },
    { TYPE = "Priority", NAME = "top" },
    { NAME = "diskio" }
  ],
}
#######################
# end new default
```

```
######################
```

```
###################################
# -- examples with documentation
###################################
{
  #
  # optional daemon/top level config params
  #

  RM_RANK_DFLT = 3,      # default rank for non-lm resourcemaps created from this
 configuration, default is 3

  MATCHING_PERIOD = 60,   # period in seconds for which to accumulate lm changes
 before performing matching,
                        # evaluated per feature, default: 60
  LOG_LEVEL = 4,         # level of logging detail to use for RDS, default is 4
 (Info), see rds_config.hh for levels

  MAX_ORS_IN_RM_SUM = 5, # maximum number of OR operators allowed in a legacy sum
 resource

  HEARTBEAT_THRESHOLD = 300, # max seconds after which no messages from an lm will
 trigger a reconnect/resync
                            # attempt, limit 2-15 minutes, default 300 (5 minutes)

  LEGACY_SUM_RESMAPS = off, # whether or not to use the legacy OR operator by default
 in the sum resource maps instead
                           # of new default comma operator

  # internal optional
  INTERNAL_HEARTBEAT_THRESHOLD = 50,  # overrides HEARTBEAT_THRESHOLD for testing
 without limits, default none

  #
  # section identifies license monitor instances, may be array or a single
 object )currently only a single
  # license monitor is supported)
  #
  LICENSE_MONITORS = [
    # preferred usage, use registry to locate the named licmon
    { NAME      = "licmon" },  # lm info is looked up in registry

    # alternate for when no access to shared registry
    {
      NAME = "licmon2",
      HOST = "localhost",
      EVENT_PORT = 12345,
     }
  ],


  #
  # Resource map section:
  #

  # The resource map section contains rules that specify feature/tag combinations to
 be monitored and
  # resource map definitions.  Each entry may be a feature rule, a resourcemap
 definition, or a collection of
  # related feature rules/resourcemap definitions that optionally share common
 parameter definitions.
  # A feature rule contains the "FEATURES" parameter, when a new tag/feature
 combination is reported from an LM,
  # it should be evaluated in order against all feature rules until a match is found.
  If a match is found (that
  # is not an exclusion rule) a resource map for that specific  tag/feature will be
 created, and that tag/feature
  # will be monitored and included in the appropriate sum resources.  Once a specific
 tag/feature is matched with
  # a rule, no further rules are evaluated for that tag/feature pair.
  # A feature rule containing the flag EXCLUDE, indicates that matching tag/feature
 combinations are not to be monitored
  # and should be excluded from further rule matching.

  # A collection contains a RESOURCE_MAPS parameter, and may contain default
 parameter definitions for the contained
  # rules.

  # A resource map definition contains neither FEATURES nor RESOURCE_MAPS parameters,
 has at least a NAME parameter.

  RESOURCE_MAPS = [

    # feature rule examples
    { FEATURES = "foo" }, # monitor feature foo across all lms and tags with all
 default values
    { FEATURES = [ "foo1", "foo2" ], TAGS = [ "bar1", "bar2 " ] }, # monitor all
 combinations
    { FEATURES = "foo?", TAGS = "bar?" }, # "

    # collection examples:
    {
      TAGS = [ "CADENCE_AUS_DV01", "CADENCE_AUS_DV02", "SYNOPSYS_LDC01",
 "CADENCE_AUS_VIP01", "CADENCE_AUS_VIP02",
               "CADENCE_AUS_VIP03", "SYNOPSYS_AUS_DV01", "MENTOR_SCV_GWAN06",
 "MENTOR_SCV_GWAN05", "MENTOR_WAN01",
               "INTERRAD_WAN01", "APACHEDA_WAN01", "SYNOPSYS_US_GWAN01",
 "SYNOPSYS_US_GWAN03", "SYNOPSYS_US_GWAN02",
               "CONCEPTENG_SOC01", "REALINTENT_SOC02", "CADENCE_SCV_GWAN02",
 "CADENCE_SCV_GWAN03", "CADENCE_SOC04",
               "CADENCE_SCV_FV01", "SYNOPSYS_SOC01", "SMARTDV_SCV_GWAN01",
 "FRACTAL_WAN01", "SYNOPSYS_SCV_DV01" ],

      # all entries in this RESOURCE_MAPS array, share the above LICMONS and TAGS
 defintions unless overridden:
      RESOURCE_MAPS = [

        {
          FEATURES = "Formality*",
          TAGS = [ "SYNOPSYS*"], # TAGS specified here completely replaces the upper
 level array of TAGS
          SUM = [  # create 2 custom sum resourcemaps for each matching feature (note
 that the same tag may appear in multiple sums this way
            { TAGS = [ "SYNOPSYS_US_GWAN01", "SYNOPSYS_US_GWAN03" ], NAME =
 "@FEATURE@" },   # ex. License:SYNOPSYS1 will map to these tags
            { TAGS = [ "SYNOPSYS_DEV??", "SYNOPSYS_US_GWAN01" ], NAME =
 "@FEATUR@E_Dev" }    # ex. License:SYNOPSYS1_Dev will map to these tags
          ]
        },

        { FEATURES = "Incisive*",   TAGS = "CADENCE_AUS_DV*" },  # override TAGS with
 glob
```

```
       { FEATURES = "Primtime",    NOMATCH, NOOOQ, MATCHRECENT = 0 }, # override
flags, MATCHRECENT is explicitly off
       { FEATURES = "Primtime-SI", NOMATCH, NOOOQ, MATCHRECENT = 0 },

       # exclusion rule prevents matching feature/tags from triggering further rules
(ex. CADENCE_SCV_GWAN02/Incisive)
       # note that exclusion applies to all remaining rules in the config, not just
in this collection
       { FEATURES = [ "Formality*", "Incisive*", "Conformal*" ], EXCLUDE },

       # overbooking TBD V2 (internal)
       { FEATURES = "VCSRuntime_Net", OVERBOOK = { THRES = 0.5, FACTOR = 1.2, QUEUED
= 1, HEADROOM = -20 } },

       # Specify partial order
       { FEATURES = "Conformal*",  TAGS = "CADENCE_*", ORDER = [ "*AUS*", "*SOC*",
"*SCV*" ] },  # tags matched by the ORDER array will be mapped before

            # other tags

       { FEATURES = "*" } # all features not already matched above (inherited TAGS
def still applies)
     ]
   },
   {
     TAGS = [ "SYNOPSYS_WANLIC", "SYNOPSYS_EVALLIC" ],
     TYPE = "Mytype", # default "License", can override - ( was "" in customer
example, not allowing "" )
     RESOURCE_MAPS = [
       { FEATURES = "1TA-OPT-SPY-GuiPkg", NAME = "atrenta_optspygui" }, # renames
sum because multiple tags
       { FEATURES = "Advanced_CDC",       NAME = "atrenta_advancedcdc" },
       { FEATURES = "BasePolicySO",       NAME = "atrenta_basepolicyso" },
       { FEATURES = "adv_checker",        NAME = "atrenta_advchecker" },
       { FEATURES = "checker",            NAME = "atrenta_checker" },

       # related resourcemap definitions
       { NAME = "atrenta_spyglass",
         MAP = "atrenta_advancedcdc atrenta_basepolicyso atrenta_advchecker
atrenta_checker" },
       { NAME = "atrenta_advancedcdc",  MAP = "Mytype:atrenta_advancedcdc" }, # add
typeless maps
       { NAME = "atrenta_basepolicyso", MAP = "Mytype:atrenta_basepolicyso" },
       { NAME = "atrenta_advchecker",   MAP = "Mytype:atrenta_advchecker" },
       { NAME = "atrenta_checker",      MAP = "Mytype:atrenta_checker" },
     ]
   },

   {
     EXCLUDE_TAGS = "*BETA*", # all tags except those that match "*BETA*"
     RESOURCE_MAPS = [
      # sum Formality1 as normal/default
      { FEATURES = "Formality1" },
      # sum other Formality* features with DEV tags as @FEATURE@_Dev and disable
logging for these features
      { FEATURES = "Formality*", TAGS = [ "SYNOPSYS_DEV??" ], NORECENT, NAME =
"@FEATURE@_Dev"  },
      # sum other Formality* features with PROD tags as @FEATURE@_Prod and disable
ooq for these features
      { FEATURES = "Formality*", TAGS = [ "SYNOPSYS_PROD??" ], NOOOQ, NAME =
"@FEATURE@_Prod"  },
      # monitor remaining Formality* tags and sum under _misc, but exclude tag EVAL
from the sum
```

```
        { FEATURES = "Formality*", NAME = "@FEATURE@_misc", SUM = { EXCLUDE_TAGS =
"EVAL" } },
     ],
   },

   # all features that start with "Bar", except "BarTest", (all tags)
   { FEATURES = "Bar*", EXCLUDE_FEATURES = "BarTest" },

   # more customer examples:

   # customer example
   {
     TAGS = "ARTWORK",
     RESOURCE_MAPS = [
       { FEATURES = "ACS3520", NAME = "gdsplot" }, # renames specific to
License:gdsplot
       { FEATURES = "ACS5003", NAME = "qckvugds" },
       { FEATURES = "ACS58IO", NAME = "qckvu3gdsII_full" },
       # create resource map that combines features
       { TYPE = "License", NAME = "qckvu", MAP = "License:qckvu3gdsII_full |
License:qckvugds" },
     ]
   },

   # customer example
   {
     TAGS = [ "SYNOPSYS_WANLIC", "SYNOPSYS_EVALLIC" ],
     TYPE = "Mytype", # default "License", can override - ( was "" in customer
example, not allowing "" )
     RESOURCE_MAPS = [
       { FEATURES = "1TA-OPT-SPY-GuiPkg", NAME = "atrenta_optspygui" }, # renames
sum because multiple tags
       { FEATURES = "Advanced_CDC",       NAME = "atrenta_advancedcdc" },
       { FEATURES = "BasePolicySO",       NAME = "atrenta_basepolicyso" },
       { FEATURES = "adv_checker",        NAME = "atrenta_advchecker" },
       { FEATURES = "checker",            NAME = "atrenta_checker" },
       { NAME = "atrenta_spyglass",
         MAP = "atrenta_advancedcdc atrenta_basepolicyso atrenta_advchecker
atrenta_checker" },
       { NAME = "atrenta_advancedcdc",  MAP = "Mytype:atrenta_advancedcdc" }, # add
typeless maps
       { NAME = "atrenta_basepolicyso", MAP = "Mytype:atrenta_basepolicyso" },
       { NAME = "atrenta_advchecker",   MAP = "Mytype:atrenta_advchecker" },
       { NAME = "atrenta_checker",      MAP = "Mytype:atrenta_checker" },
     ]
   },

   # order overrides
   {
     TAGS = [ A, B, C ],
     RESOURCE_MAPS = [
       { FEATURES = "orderExample1" }, # default TAGORDER
       { FEATURES = "orderExample2", ORDER = REVERSE_TAGORDER },
       { FEATURES = "orderExample3", ORDER = SHUFFLE },
       { FEATURES = "orderExample4", ORDER = [ B, C ] } # explicit partial order
     ]
   },
   # sum overrides
   {
     TAGS = [ D, E, F ],
     RESOURCE_MAPS = [
       { FEATURES = "sumExample4" }, # create default sum if multiple tags exist
       { FEATURES = "sumExample1", SUM = { TAGS = [ D, E ] } }, # exclude F from sum
```

```
        { FEATURES = "sumExample2", NOSUM }, # do not create sum
        { FEATURES = "sumExample3", SUM1 }, # create sum even if only one of the tags
 exists
        { FEATURES = "sumExample5", SUM = { NAME = "sum5", RANK = 4  } }, # create
 default sum, override name and rank
        { FEATURES = "sumExample6", LEGACY_SUM_RESMAPS }, # create default sum except
 use legacy ORs
        { FEATURES = "sumExample7", LEGACY_SUM_RESMAPS = 0 }, # create default sum
 except don't use legacy ORs (useful when when LEGACY_SUM_RESMAPS is the default)
      ]
    },

    # feature with a map
    { FEATURES = "mappedFeature", SUM1, NOOOQ, MAP = "linux64" },


    # example resource map definitions not related to features
    {
      TYPE = "Priority",
      RESOURCE_MAPS = [
        { NAME = "low" }, # default UNLIMITED
        { NAME = "normal" },
        { NAME = "high" },
        { NAME = "top" },
      ]
    },

    { NAME = "diskio" }, # default TYPE = ""
    { TYPE = "Tool", NAME = "footool" }, # override TYPE
    { NAME = linux, MAP = "centos | ubuntu" },

    # Limit resource examples:
    {
      TYPE = "Limit",
      RESOURCE_MAPS = [
        # create Limit:random_@USER@ with total = 305 on demand for each user
        {
          NAME = "random_@USER@",
          TOTAL = 305,
          SPECIAL = [  # override totals for these specific users:
            { NAME = "random_a_mur", TOTAL = 5000 },
            { NAME = "random_segdv_total_limit", TOTAL = 20500 },
            { NAME = "random_sbenarye", TOTAL = 600 },
            { NAME = "random_nmalki", TOTAL = 600 },
            { NAME = "random_tgreenshtein", TOTAL = 600 }
          ]
        },
        # create map to this limit, UNLIMITED by default
        { NAME = "random_segdv", MAP = "Limit:regression_segdv_total_limit" },
      ]
    }

  ], # end outer RESOURCE_MAPS

  # FLAGS section overrides flags for specific resmaps
  FLAGS = [
    { TYPENAME = "License:bar_foo", NORECENT },
    { TYPENAME = "Mytype:atrenta_advancedcdc", NOOOQ, DONOTSHARE }
  ]

}
```

```
# end of resource.cfg
```

Switching From Classic Resource Management Mode to RDS Mode

Accelerator activates the legacy resource management daemon vovresoruced by default. To switch to RDS resource management mode, follow the procedure below.

Steps 1 - 5 will configure your LM project:

1. Append this line to the LM project `<SWD>/policy.tcl`:

   ```
   set config(rds.enable) 1
   ```

2. Enable the shell and activate these changes in `policy.tcl`:

   ```
   % vovproject enable LM_PROJECT_NAME
   % vovproject reread
   ```

3. Use the following command to see if the RDS Init and Update ports are set to appropriate port numbers for your system:

   ```
   % vovselect project,initport,updateport from server
   ```

4. If the new Init and Update port numbers need to be changed, restart LM with the `-inport` and `-upport` options specified with the `lmmgr start` command.

The following steps configure your NC queue.

5. Set the NC_QUEUE environment variable to the name of your Accelerator queue.
6. Append this line to `<SWD>/policy.tcl`:

   ```
   set config(rds.enable) 1
   ```

7. Provide a translation of Tcl-based `resources.tcl` into an AVS-based `<SWD>/resources.cfg` file. See Resources.cfg.
8. Shut down `vovresourced`.

   ```
   % nc cmd vovdaemonmgr stop vovresourced
   ```

9. Start the RDS service by activating the `policy.tcl` setting:

   ```
   % ncmgr reset
   ```

Revert from RDS Mode to Classic Resource Management Mode

If RDS is active and you want to revert operation to legacy resource management, follow the procedure below:

Steps 1 – 6 are performed on your NC queue.

1. Set the NC_QUEUE environment variable to the name of your Accelerator queue.

2. Append this line to `<SWD>/policy.tcl`:

```
set config(rds.enable) 0
```

3. Provide or restore Tcl based configuration file `<SWD>/resources.tcl`.
4. Activate the updated RDS-off setting:

```
% ncmgr reset
```

5. Erase any resource maps that RDS had created while processing the configuration file.

```
% nc cmd vovforget -rdsresources
```

6. Start `vovresourced`:

```
% nc cmd vovdaemonmgr start vovresourced
```

Steps 6 - 9 will revert LM to classic mode. This is optional, because LM in RDS mode can interact with NC in either mode.

7. Enable the shell for this LM project.

```
% vovproject enable LM_PROJECT_NAME
```

8. Append this line to `<SWD>/policy.tcl`:

```
set config(rds.enable) 0
```

9. Activate the new RDS-off setting:

```
% vovproject reread
```

Monitoring RDS Function and Performance

RDS Log Files

The RDS service logs various events in a special log file in the server working directory. The file is `<SWD>/logs/rds.log`. Log entries have verbosity levels associated with them: (4=info (default)). To control the level of logging in the `rds.log` file, specify the desired maximum message log level in the `LOG_LEVEL` variable in `resources.cfg`. The log levels are defined as follows:

Level	Log Level Number
Fatal	1
Error	2
Warning	3
Info	4 (default)
Verbose	5

Level	Log Level Number
Debug	6
Internal	7

Inner Loop Timers

The vovserver tracks how the time is spent in the inner loop. The statistics are accumulated over multiple loops and are rotated every 10 seconds, so what you normally see in the stats is the overall time spent in the past 10 to 20 seconds. With RDS a new timing category "rds" is added.

If you are ADMIN, you can check the timers with `vovshow`:

```
% nc cmd vovshow -innerlooptimers
```

Accelerator CLI Command Syntax

lmmgr start

The `lmmgr start` command has two new options to permit the administrator to specify the port number for the Monitor INIT and UPDATE ports:

- `-inport`
- `-upport`

Explicit port numbers may be specified or a special keyword "any" may be specified to request that a free port be found and assigned.

Wildcard-Capable Attributes

The following matched attribute pairs accept two optional wildcard characters. The wildcard characters are '*', matches 0 or more characters, and '?', matches exactly 1 character. When a wildcard character is present in an attribute value, the value must be quoted.

Inclusion	Exclusion
FEATURES	EXLUDE_FEATURES
TAGS	EXCLUDE_TAGS

The inclusion attribute is applied first, followed by the exclusion attribute. Thus, exclusion wins regardless of order.

Configure Container Integration

Container Support

Linux containers can be leveraged to constrain the amount of system resources used by jobs. Accelerator's container support is designed to be agnostic of the container solution. The examples provided are for Docker specifically.

Note: It is currently recommended to exclude contained jobs from preemption. This can be done at job submit time using the "-preemptable 0" submission option or by writing preemption rules to exclude jobs that request a "Container:X" resource.

Named Container Configurations

Containers are enabled by the administrator through named configurations that can be requested as a job resource. Each named configuration will contain a recipe of hooks to call to setup the container, and limits to impose upon it. Hooks are stand-alone shell scripts, each meant to perform a certain task, with specific UNIX permissions (root | user).

Named configuration files must be stored in the SWD/containers directory and end with a `.cfg` extension. Examples named configuration files are provided in the `$VOVDIR/etc/config/containers` directory, such as:

File: Container C1 definition:

```
### Example named-container configuration c1.
## Hook run directory
#
# The default behavior is to switch to the job directory before running the
# hooks. If this directory is not accessible until the hooks have been executed
# though, the job will fail. This issue can be resolved by setting
# containerHooksRunDir to a location that is guaranteed to be accessible prior
# to the hooks being executed. This causes the subtasker to switch to that
# directory instead, and the original job directory will be made available to
# the hooks via the VOV_CONTAINER_JOB_RUNDIR environment variable. The value of
# this variable can then be used by the hooks to specify the working directory
# for the container instance.
# containerHooksRunDir "/path/to/run/directory"

## Hook definitions
#
# containerHook <type> <mode> <path> <signature>
#   <type>      = setup | enter | cleanup | teardown
#   <privilege> = user  | root
#   <file>      = absolute path to hook file
#   <signature> = output of vovsignfile command
#
# Specifies which hook(s) should be called to interact with the container
# platform throughout the job's life cycle:
#
# 1. The setup hook is required if the container must be created prior to
#    spawning the job. This hook will always be used if root privileges are
#    required to create containers.
# 2. The enter hook is always required and is responsible for placing the
#    job into the container. If root privileges are not required to create
#    containers and the container platform supports it, the enter hook can
#    also create the container, as would be done via the "docker run" command,
#    for example. The enter hook must block for the duration of the job.
# 3. The cleanup hook can be used to remove temporary artifacts that are
#    generated by the job.
# 4. The teardown hook is required to stop and/or remove the container. This
#    hook would normally be used if a setup hook is used to create the
#    container.
#
# Any root-mode hook must be owned by root on the filesystem.
#
# The tasker sets at least the HOST, VOV_PROJECT_NAME, VOV_JOBSLOT, and
# VOV_CONTAINER_NAME environment variables in the job execution environment.
# It is intended for the hooks to use these variables to uniquify container
# instances. Hooks can be tested by running them manually in the shell as long
# as the required environment variables are set to some value. The only hook
# that is called with arguments is the enter hook, which expects the job command
# line to be passed in. An example test run of the enter hook may look like:
#
# % env HOST=foo VOV_PROJECT_NAME=vnc VOV_JOBSLOT=0 VOV_CONTAINER_NAME=c1 \
# /path/to/c1-enter.sh whoami
# containerHook setup    root "/path/to/c1-setup.sh"    1211238920
containerHook enter    user "/path/to/c1-enter.sh"    97261574
# containerHook cleanup  user "/path/to/c1-cleanup.sh"  5376821904
# containerHook teardown root "/path/to/c1-teardown.sh" 8237156649
```

Specify the Taskers that Support Containers

Each named container configuration will require a Container:X resource to be offered by every tasker that supports that specific container configuration. This can be done via the tasker.tcl file or the taskerClass.table file.

When a user includes Container:c1 in the resource request for their job, the request will be passed to the tasker that is selected to execute the job and the tasker will process the recipe defined in the configuration. If the recipe references a hook that does not exist, is not executable, or has a different signature than the one specified in the configuration, the job will fail.

Hooks

It is recommended to manually test each hook before using them in production environment. Use the tasker hosts for testing and production. An example test run of the "enter" hook may look like:

```
#% env HOST=foo VOV_PROJECT_NAME=vnc VOV_JOBSLOT=0 VOV_CONTAINER_NAME=c1 \
#/path/to/c1-enter.sh whoami
```

Multiple hooks can be utilized to integrate with the container solution. At a minimum, an "enter" hook will be required, as long as the container solution provides a single command to setup the container, run the job, and remove the container afterward. For container solutions that do not provide this feature, separate setup and teardown hooks can be configured. For either method, an optional cleanup hook can be configured to remove artifacts generated by the job from the file system, for example.

Hooks must also be stored in the `SWD/containers` directory and can be in script or binary form. Example hook scripts are provided in the `$VOVDIR/etc/config/containers` directory, such as:

File: c1-enter.sh

```
!/bin/bash -fxv
# Note: The enter hook must block for the duration of the job.
#
# Container c1 example enter hook: an all-in-one script that creates a
# container, launches a job inside of it, then exits and removes the container.
#


# The following environment variables are available and should be used to avoid
# container and/or host name conflicts:
#
#   HOST                (string)
#   VOV_PROJECT_NAME    (string)
#   VOV_JOBSLOT         (number)
#   VOV_CONTAINER_NAME  (string)
#
# The following environment variables are available if defined in the named
# container configuration file:
#
#   VOV_CONTAINER_CORES (number)
#   VOV_CONTAINER_RAM   (megabytes)
#   VOV_CONTAINER_TMP   (megabytes)

# For containers handling interactive jobs (-I), uncomment the following line.
# set -m
containerName=${VOV_PROJECT_NAME}_${HOST}_${VOV_CONTAINER_NAME}_${VOV_JOBSLOT}
uid=$(id -u ${USER})
# Handle the run directory specified in the named container configuration file.
if [[ -d $VOV_CONTAINER_JOB_RUNDIR ]]; then
  workDirOptions="--workdir $VOV_CONTAINER_JOB_RUNDIR --mount type=bind,source=
${VOV_CONTAINER_JOB_RUNDIR},target=${VOV_CONTAINER_JOB_RUNDIR}"
else
  workDirOptions="--workdir $PWD"
fi
# Process limits into Docker options.
limitOptions=""
if [[ -n $VOV_CONTAINER_CORES && $VOV_CONTAINER_CORES > 0 ]]; then
  limitOptions+=" --cpus $VOV_CONTAINER_CORES"
fi
if [[ -n $VOV_CONTAINER_RAM && $VOV_CONTAINER_RAM > 0 ]]; then
  ramSpec="${VOV_CONTAINER_RAM}m"
  limitOptions+=" --memory $ramSpec"
fi
if [[ -n $VOV_CONTAINER_TMP && $VOV_CONTAINER_TMP > 0 ]]; then
  # Use in-memory tmpfs for /tmp in container.
  tmpBytes=$(($VOV_CONTAINER_TMP*1048576))
  limitOptions+=" --mount type=tmpfs,destination=/tmp,tmpfs-size=$tmpBytes"
fi
# Capture job environment in a file for Docker to import.
envFile=/tmp/${containerName}.env
env > $envFile

# Use the Docker "run" command to create a container based on the "myImage"
# container image, setup networking, specify the user, capture the environment,
# and bind-mount the required directories for the job. Finally, the job itself
# is passed in for Docker to execute. The image must have the ability to resolve
# the job owner's UID, have access to the VOV software installation, and be able
# to execute the vw job wrapper along with the job command.
# The "$@" variable will contain "vw <jobCmd> <jobArgs>".
```

Note: Pay close attention to the comments in each hook example. Failure to follow the guidelines provided will likely result in failure to integrate with the container solution, as well as job failure. Any root-mode hook must be owned by root on the filesystem.

Hook Signatures

The configuration for each hook that is defined must contain a signature. The signature is used by the tasker to verify the hook has not been tampered with since it was defined by the administrator. The signature can be obtained by using the `vovsignfile` utility. In the following example, the signature is highlighted in red:

```
# containerHook setup    root "/path/to/c1-setup.sh"    1211238920
containerHook enter    user "/path/to/c1-enter.sh"    97261574
# containerHook cleanup  user "/path/to/c1-cleanup.sh"  5376821904
# containerHook teardown root "/path/to/c1-teardown.sh" 8237156649
```

vovsignfile

Utility to obtain a security signature for files.

```
vovsignfile: Usage Message

      Utility to obtain a security signature for files.

      USAGE:

        vovsignfile [OPTIONS] <FILE>

      OPTIONS:

        -h    -- Show usage syntax.

      EXAMPLES:

        % vovsignfile -h
        % vovsignfile /path/to/file
```

Streaming Data Service

The Streaming Data Service (SDS) publishes a time series data stream that can be consumed by existing Kafka systems and compatible reporting tools to monitor VOV projects.

When Kafka is used as part of an infrastructure, multiple vovservers can now be enabled to provide time series data and events, without negatively impacting server performance/scalability. This allows users to capture time varying data in order to see how usage evolves over time.

Events Frequency

The metrics events are published at the same rate that metrics are calculated in the server. This will vary by load but should be at most every 10s on an active server, and possibly longer if the server is heavily loaded such that the scheduling cycle takes longer than this. The project data is expected to be relatively static and is published every 4 hours. The command: `vovservermgr config sds.readconfig 1` will cause the updated project record to be published on execution.

Project IDs

To enable data from multiple projects to be collected on the same kafka infrastructure, each event will contain the field **projId** which identifies the project which published the event. The projId field is formed by concatenating the project name, a hyphen, and the vovserver instance's numeric generated unique id. e.g. "vnc-12345678"

SDS Configuration

On startup, the vovserver will create and/or update the following configuration items:

- SDS configuration directory, at the server working directory (SWD)**/config/publishers/sds**, for example, `../vnc.swd/config/publishers/sds`
- SDS configuration file in the SDS configuration directory, `sds.cfg`
- Avro schema files (updated each time the server starts)

If the `sds.cfg` file does not already exist, the following default `sds.cfg` file is created:

```
{
  enabled = 0,
  kafka_servers = "",
  format = "json",
  site = "",
  group = "",
  enable_jobdata = 0,
  events = {
    project = {
      schemaId = 0,
      topic = "vov-projects",
    },
    taskers = {
      schemaId = 0,
      topic = "vov-metrics-taskers",
    },
    jobs = {
      schemaId = 0,
      topic = "vov-metrics-jobs",
    },
    scheduler = {
      schemaId = 0,
      topic = "vov-metrics-scheduler",
    },
    tasker = {
      schemaId = 0,
      topic = "vov-metrics-tasker",
    },
    taskerlist = {
       schemaId = 0,
       topic = "vov-metrics-taskerlist",
    },
    jobdata = {
       schemaId = 0,
       topic = "vov-jobdata",
   },
   deletejob = {
      schemaId = 0,
      topic = "vov-deletejob",
   },
  },
}
```

Configuration File Parameters

Service level configurable parameters in `sds.cfg`:

Parameter	Values	Default	Description
enabled	0 or 1	0	Disable/enable SDS on startup/readconfig
format	"json" "confluent" azure"	"json"	Specifies the Kafka payload format to use: "json": use plain text using the JSON format "confluent": use AVRO encoding with the schema registry identifier used in the Confluent services "azure": use the AVRO encoding with the schema registry identify used in Azure's EventHub service.
site	string	""	User definable string to be delivered with the project record

Parameter	Values	Default	Description
group	string	""	User definable string to be delivered with the project record
debug	0 or 1	0	Disable/enable SDS debug logging on startup/readconfig
enable_jobdata	0 or 1	0	Disable/enable job events

Event Specific Configurable Parameters in sds.cfg

Each event has its own configuration section in the config file, for example, for the project event:

```
...
events = {
  project = {
    schemaId = 0,
    topic = "vov-projects",
  },
...
```

Parameter	Values	Default	Description
schemaId	integer or string	0	N is a string, publish using Azure EventHub encoding where N is the schema's registry ID N < 0, do not publish this event N = 0, publish using single object encoding N > 0, publish using Confluent encoding where N is the schema's registry ID
topic	string	see event table	The name of the Kafka topic to which these events are published.
format	"" "json" "confluent"	""	Overrides format for this event "" = use format specified at the overall service level

Parameter	Values	Default	Description
	"azure"		

Kafka Configuration

The Kafka libraries enable some flexibility with configuration. For example, in the `publishers/sds/sds.cfg` file:

```
  # kafka properties forwarded to rdkafka library for client config
  kafka_properties = {
    security = {
      protocol = "ssl",  # security.protocol=ssl
    },
    ssl = {
      ca = {
        location = "/home/rhenry/Proj/confluent/ssl2/ca-cert" # ssl.ca.location=/
home/rhenry/Proj/confluent/ssl2/ca-cert
      },
      certificate = {
        location = "/home/rhenry/Proj/confluent/ssl2/client_hecto_client.pem" #
 ssl.certificate.location
      },
      key = {
        location = "/home/rhenry/Proj/confluent/ssl2/client_hecto_client.key", #
 ssl.key.location
        password = "abcdefgh" # ssl.key.password
      }
    }
  }
```

Kafka uses `security.protocol = "ssl"`

However, there is flexibity to include an equivalent as security `{ protocol = "ssl" }`.

The vovserver/SDS is passing the configuration through to the underlying Kafka libraries and network layer, making this feasible.

Change the Config File for the First Time

In order to use SDS for the first time, the user must perform the following operations:

1. Set the *`kafka_servers`* parameter in the `sds.cfg` file to the bootstrap server(s) for their kafka installation; for example, `kafka_servers = "kafkahost:9092"` or `kafka_servers = "kafkahost1:9092,kafkahost2:9092"`
2. If publishing using the Confluent Schema Registry, then the following steps are also needed:
   a) Upload the schema files to the schema registry and note the IDs assigned to each schema.
   b) Assign the schema registry IDs discovered in step 1 to the events in the `sds.cfg` file (see Event Specific configuration below)

For the initial release the Kafka published events are:

Event Name	Description	Schema File	Default Topic
project	(relatively) Static project information that may be useful to join with time series data	vov.projects.avsc	vov-projects
taskers	Metrics related to the state and capacity of the taskers	metrics.taskers.avsc	vov-metrics-taskers
jobs	Metrics related to the number of jobs in specific states and rate of dispatch/completion	metrics.jobs.avsc	vov-metrics-jobs
scheduler	Metrics related to scheduler performance, sizes, clients, innerloop timers	metrics.scheduler.avsc	vov-metrics-scheduler

The following events have been added in the 2022.1.0 release:

Event Name	Event Type	Description	Schema File	Default Topic
deletejob	DELETEJOB	Information about deleted job like jobid, user, name, host, etc/	vov-deletejob.avsc	vov-deletejob
jobdata	INITIAL	Initial job data like jobid, command, jobclass, status, etc.	vov-jobdata.avsc	vov-jobdata
jobdata	UPDATE	Updated job data like jobid, command, jobclass, status, etc.	vov-jobdata.avsc	vov-jobdata
jobdata	PROPERTYADD	Information about job new properties in the format <name value>	vov-jobdata.avsc	vov-jobdata
jobdata	PROPERTYMODIFY	Information about job modified properties in the format <name value>	vov-jobdata.avsc	vov-jobdata
jobdata	PROPERTYDELETE	Information about job deleted properties in the format <name value>	vov-jobdata.avsc	vov-jobdata

Changing the Config File at Run Time

The SDS configuration may be changed while the server is running.

1. The SDS service may be enabled/disabled by using the command:

   ```
    $ vovservermgr config sds.enabled 1/0
   ```

2. Update the config file for the running server and/or publish a new project event with the following command:

   ```
   $ vovservermgr config sds.readconfig 1
   ```

3. The debug setting may be enabled using:

   ```
   $ vovservermgr config set_debug_flag SDS
   $ vovservermgr config reset_debug_flag SDS
   ```

Troubleshooting

If the kafka_servers cfg parameter is not set correctly, the server log will contain entries like the following:

```
%3|1610382360.585|FAIL|rdkafka#producer-1| [thrd:foo:9092/bootstrap]: foo:9092/
bootstrap: Failed to resolve 'foo:9092': Temporary failure in name resolution (after
 1033ms in state CONNECT)
%3|1610382360.585|ERROR|rdkafka#producer-1| [thrd:foo:9092/bootstrap]: 1/1 brokers
 are down
%3|1610382363.544|FAIL|rdkafka#producer-1| [thrd:foo:9092/bootstrap]: foo:9092/
bootstrap: Failed to resolve 'foo:9092': Temporary failure in name resolution (after
 993ms in state CONNECT, 1 identical error(s) suppressed)
```

If the kafka servers are not running or reachable, the server log will contain entries like the following:

```
%3|1610383215.659|FAIL|rdkafka#producer-2| [thrd:hecto:9092/bootstrap]: hecto:9092/
bootstrap: Connect to ipv4#127.0.1.1:9092 failed: Connection refused (after 0ms in
 state CONNECT, 1 identical error(s) suppressed)
```

Environment Management

To ensure the correct and repeatable behavior of the tools, the environment must be controlled. This chapter explains how VOV supports multiple reusable environments.

Environments Overview

When using the UNIX shell, it is standard procedure to establish a working environment by setting environment variables within the shell login script, such as the `.profile` or `.cshrc` file.

The Altair Accelerator products are built to establish control and direction from the values of environment variables. The set of such controls is long enough that it makes it hard to manage all the values. A technique is supported to help manage the complexity so that a person does not use a single monolithic login script to set every possible environment variable.

This technique is to partition the set of control environment variable into working groups that are appropriate for use in certain situations and by certain activities.

Each group is given a name, and tools are provided to modify the environment to add or delete environment variables by group name.

The technique of using FlowTracer tools to set a particular working environment by name is used instead of doing a native UNIX sourcing of a variety of different scripts as needed, or of one large script sourced at login.

Definitions and Benefits

In both UNIX and Windows platforms, there is a mechanism that allows processes to communicate information to their subprocesses environment variables. For example, in a shell, the following command can be used set the value of the environment variable VOVDIR.

```
% setenv VOVDIR /remote/VOV/<version>/<platform>
```

All jobs started by the shell inherit the environment. This enables finding the root directory of the VOV installation by looking up the value of the variable VOVDIR. The behavior of many programs is affected by such environment variables.

Environment indicates the collection of all the environment variables. A single environment that can execute all tools required in a project is desirable but not always feasible. For example, if a project requires tools from many vendors, the PATH variable may become too long. In other cases, there may be conflicting requirements for the value of some variables (for example, LM_LICENSE_FILE). for these reasons, multiple environments are a valuable asset.

To address these issues, some users have developed a more or less unstructured collection of setup files, scattered around the file system, leaving it up to the individual designers to remember to use those files when needed. Others have developed a system in which the designers, instead of invoking the tools directly, invoke specialized wrappers. Wrappers set up the appropriate environment for the tool on the fly before invoking the actual tool.

If you have scattered setup scripts, the VOV environment management facilities offers a way to organize them under a logical framework. If you have wrappers, you can keep using them as a complement to the VOV environment management facilities.

VOV environment facilities let you:

- Precisely control the environment used by each job in your flow
- Load each environment on demand rather than use an overloaded environment
- Simplify the shell startup script (such as `~/.cshrc`)
- Create many small environments that are optimized and easy to maintain

- Share the environments across multiple shells (ksh, tcsh, ...)
- Share the environments among the project team members
- Share the environments among multiple projects

Environment Basics

Each VOV environment has a name. This name is an alphanumeric string, usually in uppercase. Underscores are also allowed. For example, an environment could be named `TEX` for running Latex, and another environment named `SYNOPSYS` for running the Synopsys tools.

The name of the current environment is represented by the environment variable VOV_ENV. If this variable is not set, VOV assumes that the name is `DEFAULT`.

Environment definitions are found in the directory `$VOVDIR/local/environments`. The optional variable VOV_ENV_DIR can be used to identify other directories where additional environments can be found. The value of this variable is a list of directories separated by colons ":" on UNIX systems and by semicolons ";" on Windows.

The environment files may be written in C-shell, in Bourne Shell, or Tcl. Regardless of the syntax used to describe it, any environment can be used in any shell; an environment written in C-shell can be used, even if ksh is being used.

VOV stores the name of the environment that is used to execute each job. Before re-executing a job, VOV switches to the appropriate environment. The switch of environment is actually performed on the taskers. Taskers cache environments, resulting in instantaneous switches between environments.

Parameterized Environments

An environment definition may accept parameters. This is useful, for example, to select different versions of some tool.

Parameters are passed to the script either one of syntaxes shown below. The older syntax uses parentheses, which need quoting when used from the shell

In this syntax, parentheses are used:

```
environmentName(parameter1[,parameter2]...)
```

In this 'comma' example, parentheses are not used:

```
environmentName,parameter1[,parameter2]
```

The parameters are a comma-separated list of tokens that are placed in parentheses after the environment name or after the first comma. No spaces are allowed. Arguments cannot contain commas, spaces, quotes, or other special characters. Proper quoting must be used when switching to a parameterized environment from the command line.

Examples

These examples show how to add a parameterized environment. The two pairs of lines have the same effect.

```
% ves '+D(DISPLAY=tahoe:0.0)'
% ves '+D,DISPLAY=tahoe:0.0'
% ves '+SYNOPSYS(1998.08)'
% ves '+SYNOPSYS,1998.08'
```

From inside the environment script, the parameters are accessible by means of the variable '$argv' in C-shell or the list `$argv` in Tcl. Parameters are passed to both the `start.tcl` script, and the `end.tcl` script.

For example, this is the definition of the standard environment `D`:

```
# This is D.start.tcl
# An environment to define variables.
# Usage: ves +D,VAR1=value,VAR2=value,...
foreach arg $argv {
    if [regexp {([^=]+)=(.*)} $arg all var value] {
        setenv $var $value
        lappend env(D_env_vars) $var
    }
}
```

Multiple environment variables can be set while launching a job using `D` using parentheses, like this:

```
% nc run -e "D(VOV_LIMIT_maxproc=8192,VOV_LIMIT_openfiles=8192)" env
```

Multiple environment variables can also be set for individual jobs by using the comma notation without parentheses and without quotes.

```
% nc run -e D,VOV_LIMIT_maxproc=8192,VOV_LIMIT_openfiles=8192 env
```

Inside a jobclass definition file, a parametrized environment can be specified like this:

```
set VOV_JOB_DESC(env) "SNAPSHOT+D,VOV_LIMIT_maxproc=8192,VOV_LIMIT_openfiles=8192"
```

Curly braces are also supported in the use of environment variables. This permits the use of commas, among other special characters. For example:

```
nc run -e 'D(FOO={value,with,commas},BAR=normal_value)' ...

and

ves 'D(FOO={bar,baz})'
```

Another example:

```
set VOV_JOB_DESC(env) "SNAPSHOT
+D,VOV_LIMIT_maxproc=8192,MY_CSV_VAR={a,b,c},VOV_LIMIT_openfiles=8192"
```

Composite Environments

Complex environments can be built via *composition*; use the operator "+" with `ves`.

For example, if you there are two environments `E1` and `E2`, they can be combined by switching to the environment `E1+E2` as shown below:

```
% ves E1+E2
```

Order of Environment Components

The order of the environment components in environments is significant, because some environment definitions can be destructive, while others may be in conflict with each other.

For example, the environment `BASE` sets the variable PATH to a well-defined list of directories, ignoring any previous value. For this variable, the environments `BASE` and `E1+BASE` are identical, because it is completely determined by the `BASE` environment. Note that in general, the environment `BASE+E1` is a true composite environment (assuming that E1 is not destructive).

For example, to use tools from Synopsys and Virage, they can be run in the combined environment `SYNOPSYS+VIRAGE` as shown below:

```
mars chip@mercury BASE src/vhdl > ves SYNOPSYS+VIRAGE
mars chip@mercury SYNOPSYS+VIRAGE src/vhdl >
```

Environment Examples

The following is an example of a start script for the environment named `SYNOPSYS`. Typically, this script is stored in: `$VOVDIR/local/environments/SYNOPSYS.start.csh`:

```
###################################################
# Typical Synopsys Environment: SYNOPSYS.start.csh
###################################################

setenv SYNOPSYS        /home/eda/synopsys/synopsys3.0
setenv SIM_ARCH        sparc
setenv LD_LIBRARY_PATH  `vovenv APPEND -: $SYNOPSYS/$SIM_ARCH/sim/lib
 $LD_LIBRARY_PATH`
setenv MANPATH          `vovenv APPEND -: $SYNOPSYS/doc/sim/man  $MANPATH`

set path = `vovenv APPEND $SYNOPSYS/$SIM_ARCH/sim/bin   $path`
set path = `vovenv APPEND $SYNOPSYS/$SIM_ARCH/motif/bin $path`
set path = `vovenv APPEND $SYNOPSYS/$SIM_ARCH/syn/bin   $path`
set path = `vovenv APPEND $SYNOPSYS/$SIM_ARCH/sge/bin   $path`
```

The following is another example of building a composite environment:

```
# -- A combined environment: call it COMBI.start.csh
# -- Get Synopsys and EPIC tools together.
source $VOVDIR/etc/std.vov.aliases

# Show two different ways to use ves.
ves BASE+SYNOPSYS
ves +EPIC
```

Refresh Environments

To save time while switching environments, taskers cache up to eight environments, which makes switching instantaneous. The cached environments can be refreshed either through the GUI by clicking **Project** > **Taskers control** > **Refresh environments**, or by entering:

```
% vovtaskermgr refresh
```

Refreshing is necessary if you modify an environment after the taskers have cached it. Restarting the taskers achieves the same result at a slightly higher cost.

CAUTION: On Windows NT, refresh does not work. You must restart the taskers instead.

Develop Environments

Each environment can be described with Tcl, C-shell, or Bourne-shell scripts, which allows the re-use of existing scripts. The Tcl syntax is recommended; the resulting environment can be used on both UNIX and Windows systems, and Tcl supports aliases.

When necessary, the environment definition, written in any of the above languages, is automatically converted to use with Bourne-shell and the derivatives of Korn-shell and bash, C-shell and derivatives, Tcl scripts, and DOS prompt.

Each environment is described by the files shown in the following table. A minimal description of an environment consists of the `start*` script and the DOC file.

The `.end*` script is usually needed only for environments that are used with `ves` from the command line. The vovtasker binary caches eight recently used environments. When all eight cache slots are full and a new one is needed, the least-recently used environment is discarded without calling any of the `.end*` scripts.

Some C-shell implementations have small limits on some important variables, such as the length of the `path`. If environments are needed that exceed those limits and tcsh is on the hosts, the `.tcsh` script suffix can be used.

Note: A vovtasker does not execute jobs using any shell. Instead, a vovtasker uses the `execve()` system call. The shell implied by the environment script suffix is only used to compute the environment.

Suffix	Language	Description
`start.csh`	C-shell	Initialization scripts, executed before entering the environment. If multiple scripts exist for the same environment, VOV will prefer in the following order, Tcl, C-Shell, Bourne-Shell, tcsh.
`start.sh`	Bourne-shell	
`start.ksh`	Korn-shell	
`start.tcl`	Tcl	
`start.tcsh`	tcsh	

Suffix	Language	Description
`end.csh`	C-shell	Termination scripts, read when exiting the environment. See comment above about choice of language
`end.sh`	Bourne-shell	
`end.tcl`	Tcl	
`end.tcsh`	tcsh	
`pre.tcl`	Tcl	Executed before the execution of the job.
`post.tcl`	Tcl	Executed after the execution of the job.

General Rules

A good environment definition is minimal, incremental and reversible.

Note: These general rules are recommendations; they are not requirements. VOV works well with environments that are not minimal, incremental, or reversible.

- **Minimal**: The definition adds only the minimum number of variables necessary to correctly execute a certain class of tools.
- **Incremental**: It builds upon the original environment.
- **Reversible**: It is possible to restore the original environment.

Rules to Write Environments in c-shell

In this example, an environment is created. The environment is named `MYENV`, which contains the directory `/usr/local/bin` in the path. The start script for this environment is `$VOVDIR/local/environments/MYENV.start.csh`.

In C-shell, either the shell variable `path` or the environment variable PATH can be set. An example follows:

```
# -- This is MYENV.start.csh
set path = ( /usr/local/bin $path )
```

Note: This solution has a disadvantage. Switching to the `MYENV` environment, the resulting PATH may contain duplicates of `/usr/local/bin`. In the long run, it is possible for the PATH variable to exceed its maximum allowed length (about 1kB), which can be imposed by some implementations of csh.

A better solution avoids duplicates. For this purpose, use `vovenv`, which is a script to manipulate environment variables. The usage for `vovenv` is:

```
vovenv OPERATION [-colon] wordlist
```

Operation	Description
DELETE	Deletes word from list.

Operation	Description
APPEND	Adds the word at the end of the list.
PREPEND	Adds the word at the beginning of the list.

If -colon is used, the list is assumed to be colon-separated, as for the environment variable PATH. Otherwise, it is a space-separated list such as the C-shell variable `path`; Instead of -colon, -: can be written.

For example:

```
# -- This is a better MYENV.start.csh
set path = `vovenv PREPEND /usr/local/bin $path`
```

In the `MYENV.end.csh` file, revert the changes made by the start script with the operation DELETE of `vovenv` as shown below.

Note: This practice should be applied to all the environment variables that describe lists of files or directories, such as PATH, MANPATH, LD_LIBRARY_PATH and LM_LICENSE_FILE.

For example:

```
# -- This is MYENV.end.csh
set path = `vovenv DELETE /usr/local/bin $path`
```

Rules to Write Environments in Tcl

If you are familiar with Tcl, consider writing the environment definitions in this language. The advantage is the portability between UNIX and Windows.

Note: Tcl must be used to describe environments for Windows.

To write an environment in Tcl, it is important to remember that all environment variables are available through the associative array `env()`. For example, the value of the variable VOVDIR is accessible as `$env(VOVDIR)`. You also need to become familiar with the following Tcl procedures supplied by VOV:

```
setenv name value
unsetenv name ...
vovenv name separator op arg ...
alias name words ...
```

These procedures look similar to their C-shell equivalent. In fact, they are Tcl procedures that are defined in `$VOVDIR/tcl/vtcl/vovenvutils.tcl`.

The procedures `setenv` and `unsetenv` behave as their C-shell counterparts. The procedure `vovenv` has the same functionality as the shell utility `vovenv`, but with a different syntax. Refer to the VOV/Tcl book for more information about these procedures.

Error handling: if errors are detected while processing of the environment definition, do not call `exit`. Instead, use the call `error`.

The environment `MYENV` that was described in the previous section can be described with the Tcl syntax as shown below.

Note: Because the colon ":" is used as a path separator, this example only works for UNIX. (The example shown after this works with Windows.)

```
# This is MYENV.start.tcl
vovenv PATH : PREPEND /usr/local/bin
```

```
# This is MYENV.end.tcl
vovenv PATH : DELETE /usr/local/bin
```

To have an environment that also works on Windows the following form can be used:

```
# This is MYENV.start.tcl
if { $::tcl_platform(platform) eq "windows" } {
    # Quote ; because it is the command separator in Tcl.
    vovenv PATH ";" PREPEND c:/local/bin
} else {
    vovenv PATH : PREPEND /usr/local/bin
}
```

For Windows environments, care must be taken in dealing with case insensitivity and with the confusion between backward and forward slashes. The variables *Temp* and *TEMP* are indistinguishable in Windows, because they differ only in case. In Tcl, however, `env(Temp)` and `env(TEMP)` are distinct and only one of the two can be used. If the value of an environment variable is needed, first call the procedure `nt_preprocess_env` to create an upper-case only version of the variable:

```
set tmpdir $env(TEMP)  ;# May not work

nt_preprocess_env
set tmpdir $env(TEMP)  ;# Guaranteed to work.
```

Another useful procedure is `nt_slashes`, which is used to convert the direction of slashes in file names. Example:

```
nt_preprocess_env
set tmpdir [nt_slashes $env(TEMP)]
```

Support for Aliases

Some customers desire the ability to define aliases in environments. Aliases are useful shorthands and reduce typing. They are useful only in command shells. Aliases are not used when taskers execute jobs.

To define an alias, you have to describe an environment using Tcl syntax. Aliases defined in the environment become available to the following shells: C-shell, Tcsh, Korn-shell. They are not available in Bourne-shell or in DOS.

The synopsis to define an alias is:

```
alias NAME WORD ....
```

For example: define an alias called 'l' for 'ls -sF':

```
# At the end of $VOVDIR/local/environments/BASE.start.tcl
alias lll ls -sF
```

```
% ves BASE% alias lll
ls -sF
```

Pre and post Conditions

As part of environment definition, you can prepare two scripts, called `NameOfEnv.pre.tcl` and `NameOfEnv.post.tcl`, which can be used to take care of pre- and post-conditions on a job by job basis.

Pre-Command and Post-Command Job Conditions

When a job is being submitted, a pre-condition and/or a post-condition can be specified.

- **pre-condition**: a script that is executed before the job is executed.
- **post-condition**: a script that is executed after the job has completed. The post-condition is typically used to perform cleanup, such as deleting temporary files in `/usr/tmp`.

Example scripts are available in the following directories: `$VOVDIR/etc/pre` and `$VOVDIR/etc/post`.

Pre-condition

A pre-condition is executed before the job is run. It is invoked with a single argument: the ID of the job. A pre-condition is executed with the same credentials as the job (userid, os-groupid) and is in the same directory of the job.

- If the precondition script fails by exiting with a status different from 0 (zero), the job will not be run and the exit status of the job will be the exit status of the pre-condition script.
- If the exit status of the pre-condition script is within the range 201-215, the automatic rescheduling condition will occur and the job will be rescheduled on a different host or on a different tasker.

Post-condition

The post-condition script is invoked with two arguments: the ID of the job and the exit status of the job. The post-condition is executed with the same credentials as the job (userid, os-groupid) and in the same directory of the job.

- When the post-condition script is invoked, the job is still running.
- The post-condition is executed after the job, even if the job fails, but it is not executed if the pre-condition fails.
- The exit status of the post condition overrides the exit status of the job. It needs to explicitly return the exit status of the job when that is the requested behavior (see the example scripts).

Submit Jobs with Conditions

Use the options -pre and -post with `nc run` to specify the pre- and post- conditions.

```
% nc run -pre $VOVDIR/etc/pre/pre_check.sh sleep 10
% nc run -post $VOVDIR/etc/post/post_cleanup.sh sleep 10
```

Log Files

The standard output from the pre- and post-commands is saved in log files. The location of the log files is determined by the value of the environment variable NC_LOGDIR. If NC_LOGDIR is not set, the files are stored in the directory `./vnc_logs`, relative to the current launch directory.

In the following example, NC_LOGDIR is not set, and the run directory is `~/testrundir`:

```
[goetz@goetz1 ~/testrundir]$ pwd
/home/goetz/testrundir
```

```
[goetz@goetz1 ~/testrundir]$ ls
vnc_logs
[goetz@goetz1 ~/testrundir]$ ls -a vnc_logs/
. .. 20210726 .precmd.000083865.log .precmd.000083885.log snapshots
```

The log files are created with zero size if the pre- and post-commands redirect all the output of the files. At the end of the job, if these files are zero length, they are automatically deleted to reduce disk space overhead.

The log files are named according to the following rules:

```
.precmd.$jobID.log
.postcmd.$jobID.log
```

The pre- and post-command log files can optionally be located in the same directory as the job logfile. For example:

```
nc run -pre "myprecommand > @JOBLOGDIR@/@JOBID@_pre.out" -l path/to/an/existing/
directory/mycommand.out -- mycommand
nc run -post "mypostcommand > @JOBLOGDIR@/@JOBID@_post.out" -l path/to/an/existing/
directory/mycommand.out -- mycommand
```

This would result in the respective pre- and post-command logfiles being written to the directory `path/to/an/existing/directory`.

Note: When using the `nc run` command after forgetting jobs that have pre- and/or post-commands, it does not automatically remove the pre- and post-command `.log` files. If these files are not zero length, they must be removed manually.

Manage Umask

The umask feature is used on UNIX to set the permissions on new files and directories. VOV supports umask with the environment variable `VOV_UMASK`. This variable is checked by the wrappers (vw, vov, etc.). When set, the wrapper adjusts the umask accordingly.

The environment variable VOV_UMASK is automatically set to the value of the umask in the submit environment when using an environment snapshot in Accelerator.

If using a named environment, VOV_UMASK may need to be set separately. To do so, add `D(VOV_UMASK=value)` to the environment specification.

Note: The logfile of the job is created by the vovtasker, and its mode is controlled by VOV_UMASK. However, the date-stamped directory `YYYYMMDD` under `vnc_logs` is created at job submit time by the `nc run` command; the file permission (mode) of the logfile is controlled by the umask in the submit shell.

Examples with VOV_UMASK

This section provides examples of using `VOV_UMASK`.

In the following example, the umask is set in the current shell to a value that is different from the one in the `~/.cshrc` file. The 077 umask removes all permissions from group and others.

```
% umask 077
% umask
77
% grep umask ~/.cshrc
umask 022
% nc run -v 0 -r unix -wl csh -c umask
----STARTING ON some-host-name----
22
----END OF LOG----
----EXIT STATUS 0----
```

The next example shows the umask value that is set in the `~/.cshrc` file, since the csh ran that file when it started. This value overrides the value that is captured in the environment snapshot.

```
% nc run -v 0 -r unix -wl csh -fc umask
----STARTING ON some-host-name----
77
----END OF LOG----
----EXIT STATUS 0----
```

The following example shows the umask value that is set in the current shell, captured in the snapshot environment and reproduced in by vw2. That shell does not run the `~/.cshrc` file because of the -f option. The snapshot file contains the following:

```
...
VOV_UMASK='077'
export VOV_UMASK
...
```

The next example shows a umask value that is different from the current shell, that was most likely inherited from the one set up in the startup script of the owner of Accelerator. It could also be set by VOV_UMASK in the startup script for the BASE environment. The BASE environment shipped by Altair does not set VOV_UMASK.

```
% nc run -v 0 -e BASE -r unix -wl csh -fc umask
----STARTING ON some-host-name----
22
----END OF LOG----
----EXIT STATUS 0----
```

The following example shows the umask value that is set by the VOV_UMASK environment variable, which is different from both the current shell, and the shell startup file.

```
% nc run -v 0 -e 'BASE+D(VOV_UMASK=055)' -r unix -wl csh -fc umask
----STARTING ON some-host-name----
55
----END OF LOG----
----EXIT STATUS 0----
```

The following example creates the logfile named by the command, and possibly the 20060713 subdirectory of `vnc_logs`, if it did not exist. The subdirectory is mode 0700, since it was initialized from the umask in the current shell.

The logfile itself, 162014.8525, has mode 0620 because it was initialized from the OR of umask 055 and 666 in the wrapper vw2.

```
% nc run -e 'BASE+D(VOV_UMASK=055)' -r unix -wl date
Resources= unix CPUS/1
```

```
Env      = BASE+D(VOV_UMASK=055)
Command  = vw date
Logfile  = vnc_logs/20060713/162014.8525
JobId    = 00424353
----STARTING ON some-host-name----
Thu Jul 13 15:57:56 PDT 2006
----END OF LOG----
----EXIT STATUS 0----
% ls -ld vnc_logs/20060713
drwx------  2 cadmgr rtda 4096 Jul 13 15:57 vnc_logs/20060713
% ls -l vnc_logs/20060713
-rw--w----  2 cadmgr rtda 29 Jul 13 15:57 vnc_logs/20060713/162014.8525
```

Environment Debugging

A faulty or incomplete environment definition can cause problems with running jobs. Example: A job succeeds when executed directly from the command line but fails when executed by the taskers.

The utility `taskerdebug` can be used to debug the environments used by the `taskerdebug`. This utility prints all environment variables, aliases and equivalences into the file that is given as its first argument. In the following example, the environment named BASE is debugged, and `base.out` is the output file:

```
% ves BASE
% vov taskerdebug base.out
```

The command can now be retraced on selected taskers and check the file `base.out` for clues about the problem with the environment. If necessary, use the resource mechanism to direct the job to the desired tasker.

Note: The `taskerdebug` command is implemented as a `csh` script on UNIX, and a `.bat` script for Windows. The command `machinfo` can also be used, which is implemented in Tcl. Implemented in Tcl allows using this command in either UNIX or Windows. This may be preferred, as the `machinfo` output provides more information than the command `taskerdebug` on Windows.

Environment Checking

To verify if an environment is good, use `vovenvcheck`, which checks that all variables set in the `start.*` file are properly unset in the `end.*` file. Example:

```
% vovenvcheck env_name
```

Environment Management: Limits

In addition to environment variables, hard and soft limits can affect tool behavior. Hard and soft limits are set in the shell and are imposed by the operating system. Both UNIX and Windows provide a mechanism for processes to communicate information to their subprocesses via environment variables.

VOV uses special environment variables to communicate limit information. The environment variables are named VOV_LIMIT_<name>. `name` is the name of the limit, such as VOV_LIMIT_cputime.

A complete list of VOV_LIMIT_<name> environment variables include:

- VOV_LIMIT_cputime
- VOV_LIMIT_datasize
- VOV_LIMIT_descriptors
- VOV_LIMIT_filesize
- VOV_LIMIT_maxproc
- VOV_LIMIT_memorylocked
- VOV_LIMIT_memoryuse
- VOV_LIMIT_openfiles
- VOV_LIMIT_stacksize
- VOV_LIMIT_vmemoryuse

The variables are interpreted in the wrapper program `vw2` which uses the C-language `getrlimit()/setrlimit()` system calls to set the limits for the child process when the job runs.

The value of the variable must be an sequence of digits followed by an optional unit indicator letter (case insensitive), or the string value 'unlimited'. If there is no unit indicator, the units are kilobytes (1024 bytes). The recognized unit indicators are:

- k or K, kilobytes (same as no unit indicator)
- M, Megabytes (multiply by 1024)
- G, Gigabytes (multiply by 1024*1024)
- T, Terabytes (multiply by 1024*1024*1024)

In Accelerator, the limit can be set at submission time. In the following example, a limit of 20 seconds of CPU is set for a job.

```
% nc run -e 'BASE+D(VOV_LIMIT_cputime=20)' .... shortjob.csh
% nc run -e 'BASE+D(VOV_LIMIT_vmemoryuse=5G)' .... bigjob.csh
% nc run -e 'BASE+D(VOV_LIMIT_vmemoryuse=5G)' .... bigjob.csh
```

In the following example, the command lines provide the roughly equivalent limit of 5GB for a job.

```
% nc run -e 'BASE+D(VOV_LIMIT_vmemoryuse=5000M)'    .... bigjob.csh
% nc run -e 'BASE+D(VOV_LIMIT_vmemoryuse=5G)'       .... bigjob.csh
% nc run -e 'BASE+D(VOV_LIMIT_vmemoryuse=5000000k)' .... bigjob.csh
```

The multipliers used with the memory specifications for limits are as follows: 'k' (KiB, 1024=2^{10}, kibibytes) 'M' (MiB, 2^{20}, mebibytes) 'G' (GiB, 2^{30}, gibibytes) and 'T' (TiB, 2^{40}, tebibytes). The multipliers are case-insensitive.

If an integer has no unit specification, kilobytes are the units used. For example, 1024K is the same as 1024. In addition, using `unlimited` as a value is acceptable.

The following two examples show how to find the current limits.

For `csh/tcsh`:

```
% limit
cputime         unlimited
filesize        unlimited
datasize        unlimited
```

```
stacksize       unlimited
coredumpsize    0 kbytes
vmemoryuse       unlimited
descriptors     1024
memorylocked    unlimited
maxproc         2048
openfiles       1024
```

For `sh/bash`:

```
bash-2.05$ ulimit -a
core file size (blocks)     unlimited
data seg size (kbytes)      unlimited
file size (blocks)          unlimited
max locked memory (kbytes)  unlimited
max memory size (kbytes)    unlimited
open files                  1024
pipe size (512 bytes)       8
stack size (kbytes)         unlimited
cpu time (seconds)          unlimited
max user processes          5119
virtual memory (kbytes)     unlimited

## Use option -H to get the hard-limits
bash-2.05$ ulimit -a -H
...
```

In most cases, everything in the shell startup file can be set as unlimited. This setup gives the tools the greatest possibility of a successful run. It is extremely rare for this method to not work.

When an environment snapshot is used when submitting jobs to Accelerator, the limits in the submission environment are captured automatically.

If using a named environment, the following Tcl code in the `ENV.start.tcl` script can be used to set the variables for limits and umask. This is the same as what is done in the environment snapshot.

```
if { [info command vtk_umask_get] != {} } {
    setenv VOV_UMASK [vtk_umask_get]
    catch {
        vtk_limits_get limit
        foreach n [array names limit] {
            setenv VOV_LIMIT_$n $limit($n)
        }
    }
}
```

Tcl-language API

VOV provides two API procedures to get and set the limits, `vtk_limits_get` and `vtk_limits_set`. Both procedures take a single array parameter, which contains the limits, keyed by name.

Because different platforms have different limits available, the VTK procedures support only a common subset of limits.

The following names are supported:

VTK Limit Procedure Name	Description
stacksize	Size of process stack segment, bytes

VTK Limit Procedure Name	Description
datasize	Size of process data segment, bytes
cputime	Maximum process CPU time, seconds
filesize	Maximum file size, bytes
coredumpsize	Maximum core dump file size, bytes

In the following example, Tcl code is used to eliminate `core` files by setting the `coredump` size limit to zero:

```
catch {
    vtk_limits_get L ;      # get existing limits into array L
    set L(coredumpsize) 0 ; # set coredump limit
    foreach n [array names L] {
        setenv VOV_LIMIT_$n $limit($n) ; # propagate limits to envVars
    }
}
```

The SNAPSHOT and SNAPPROP Environments

The SNAPSHOT Environment

If the environment variable VOV_ENV is not defined or if it is defined and it contains the keyword SNAPSHOT, the submission procedure creates a snapshot file with all current environment variables, excluding some troublesome variables, listed later in this section. The snapshot file is typically in `vnc_logs/snapshots/$LOGNAME/$VOVARCH/envNNNNN.env` where `NNNNN` is a hash of the content of the file, and it is used to quickly share existing snapshots between files. After the snapshot file has been created, the environment of the job is modified so that the SNAPSHOT environment gets a parameter which is a full name or a relative name to the snapshot file. The snapshot file is chosen based on the value of the variables NC_LOGDIR and NC_SNAPSHOTDIR.

For example:

```
% nc run -e SNAPSHOT sleep 10
Fairshare= /time/users.john
Resources= macosx
Env      = SNAPSHOT(vnc_logs/snapshots/john/macosx/env13378.env)
Command  = vw sleep 10
Logfile  = vnc_logs/20121013/124703.55776
JobId    = 007470870
```

The SNAPPROP Environment

It has been proven that under heavy load, many NFS servers are not fast enough to deliver the SNAPSHOT file to the remote host on which jobs need to be executed. This is particularly tough to debug because by the time one is ready to investigate a job failure caused by a bad snapshot file, the file has become available on the remote host.

Accelerator implements an alternative way to deliver the snapshot information to the remote job which does not rely on NFS files but rather on properties attached to the job. The downside of this approach is an increase in memory use by the main vovserver in

Accelerator, such on the order of one additional GB for about 100,000 jobs, although this does not seem to be much of a problem on current hardware.

Currently, the only way to use the SNAPPROP environment is to use option -ep in `nc run`.

```
% nc run -ep  sleep 10
Fairshare= /time/users.john
Resources= macosx
Env       = SNAPPROP(@JOBID@)
Command   = vw sleep 10
Logfile   = vnc_logs/20121013/123218.26892
JobId     = 007470866

% nc info !
Id,User,Group   007470866,john.staff,/time/users.john
Environment     SNAPPROP(007470866)
Directory       /Users/john
Command         sleep 10
Resources       macosx
Submitted from  mac09
Submitted at    Sat Oct 13 12:32:18 PDT 2012
Priorities      schedule=normal execution=normal
Status          Done
   Host         mac09
   Queue Wait   0s
   CPU Time     0.00
   Max RAM      0MB
   Duration     11
   Age          9s
   AutoForget   1
```

Customize SNAPSHOT Behavior

Some aspects of the environment snapshot procedure can be customized by the administrator via the `$VOVDIR/local/vovenv.config.tcl` file. The settings in this file apply to both file and property-based snapshots.

Variable Blacklisting

It may be desirable to prevent some environments variables from being carried over from the submission environment to the job execution environment. A blacklist can be established via a Tcl list variable named *badVarList*.

- ANIMALS
- HOST
- HOSTNAME
- LC_COLLATE
- LC_CTYPE
- LC_MESSAGES
- LC_MONETARY
- LC_NUMERIC
- LC_TIME
- LS_COLORS
- OSREV
- OSTYPE

- PROMPT
- PWD
- SHELL
- SHLVL
- TERMCAP
- TK_TABLE_LIBRARY
- USERNAME
- VOVARCH
- VOVDIR
- VOVSAVEPROMPT
- VOV_ENV
- VTCL_LIBRARY
- VTIX_LIBRARY
- VTK_LIBRARY
- WINDOWIDTERM
- WINDOW_TERMIOS
- _

Other Settings

maxEnvSize

Specifies the maximum size, in bytes, the submission environment is allowed to be for the environment snapshots.

Example File

```
# Example of $VOVDIR/local/vovenv.config.tcl  file.
lappend badVarList SITE
set maxEnvSize 10000
```

Directories and Files

Working Directories and Equivalences

The directory where the top level job is executed must be visible and accessible to both the Accelerator vovserver, which is running on UNIX, and the remote Windows NT machine.

This is normally not a limitation, since there is always at least one directory that satisfies this requirement, the one of the Accelerator installation itself. Nothing prevents you from changing to any other directory as part of the job.

It is imperative that you explain to the Accelerator vovserver the naming equivalences between UNIX files and Windows NT files. Keeping up with our examples, the path to the Accelerator installation is `/usr/local/rtda` on UNIX and `f:\rtda` on Windows NT. This can be described in the file `vnc.swd/equiv.tcl` as:

```
# Fragment of vnc.swd/equiv.tcl
vtk_equivalence NCROOT /usr/local/rtda
vtk_equivalence NCROOT f:/rtda;  # Notice the forward slashes!!
```

where `NCROOT` is the "logical name" we want to use for both the directory `/usr/local/rtda` and `/usr/local/rtda`.

After you change the `vnc.swd/equiv.tcl` file, you must always do a full reset:

```
% ncmgr reset -full
```

Canonical and Logical File Names

VOV clients and server exchange dependency information by using file names; each file needs a single name that is valid on both the client and the server.

It may seems that each file, could use its full path as its unique name. However, a file may and will have more than one name for the following reasons:

- Links, both hard and symbolic, allows multiple full paths for the same file.
- For any file, it is possible to generate an infinite number of full paths by using the "dot" and "dot-dot" notation (for example, `/usr/bin/ls` can also be written as `/usr/../usr/bin/./ls`).
- The same file may have different full paths on different hosts due to how the file systems are network mounted.

Canonical Names

VOV defines the *canonical name* of a file to be the full path obtained by removing all symbolic links and all "dots".

In this example, a file system contains the following link:

```
/users/john/projects --> /sandbox/projects
```

With the relative path `~/projects/vhdl/vtech/../syn/vtech.v`, with respect to the user `john`, the following transformations would apply:

The non-canonical path	`~/projects/vhdl/vtech/../syn/vtech.v`
after tilde expansion becomes	`/users/john/projects/vhdl/vtech/../syn/vtech.v`
after removing the symbolic link becomes	`/sandbox/projects/vhdl/vtech/../syn/vtech.v`
by removing the double dot becomes canonical	`/sandbox/projects/vhdl/syn/vtech.v`

Logical Names

A canonical name is then turned into a *logical name*. A logical name is one in which the file name begins with the value of a variable.

For example, the name `${HOME}/foo.c` is logical, while `/users/home/john/foo.c` is not.

The use of logical names is critical because the value of the variable used in the name is allowed to be different on different hosts. This is to account for the different ways the file systems are mounted across the network.

For example, the variable `${HOME}` may point to `/users/home/john` on a UNIX machine and to `h:/john` on a Windows NT machine.

All filenames in VOV are logical and canonical names. The logical names are formed according to the rules defined in the equiv.tcl file.

There are two further advantages in using logical canonical names:

- The average length of names is reduced, which reduces the storage requirements for the trace.
- The trace can be easily moved from one file system to another.

Define Equivalences for File Names

There are multiple methods to define the equivalences used to compute the canonical names of files and directories.

For example:

- Instruct the server to parse the `equiv.tcl` file and provide entries to clients. This is the default behavior. Note that for this case, equivalences that reference an environment variable should not resolve the variable in this file, in environments that will have both UNIX and Windows clients. Instead, they will need to be resolved by the client upon receipt. This is done by enclosing the equivalence value inside curly braces and referring the environment variable as $VARNAME as opposed to the Tcl format of $env(VARNAME).
- Instruct clients to read the file directly. This is a legacy method that requires that all clients have access to the server working directory so they can parse the `equiv.tcl` file for entries, and read/write access to the `equiv.caches` directory so the entries can be written to a host-based cache file for future use. In this mode, environment variables may be resolved in this file, but the behavior will be the same as not allowing them to be resolved. To resolve them in this file, the equivalence value should not be wrapped with curly braces and the environment variable should be referred to in the Tcl format of $env(VARNAME). This method is enabled by setting the VOVEQUIV_CACHE_FILE environment variable to "legacy".
- Instruct clients to read a specific cache file only. This is a special method used in corner cases where directories may not be the same but should be forced to be considered the same. This is utilized mainly by Monitor agent single-file

distributables. In this mode, environment variables may be resolved in this file, but the behavior will be the same as not allowing them to be resolved. To resolve them in this file, the equivalence value should not be wrapped with curly braces and the environment variable should be referred to in the Tcl format of $env(VARNAME). This method is enabled by setting the VOVEQUIV_CACHE_FILE environment variable to a valid equivalence cache file path.

Equivalence File

The equivalence file (`equiv.tcl`) defines the rules to generate logical names. This file is used by all clients as well as by the server. This file is a Tcl script. The fundamental procedure used in this script is `vtk_equivalence`.

The procedure `vtk_equivalence` has the following purposes:

- The main purpose is to define an equivalence between a logical name and a physical path, as in:

  ```
  vtk_equivalence TOP /export/projects/cpu
  vtk_equivalence TOP p:/cpu
  ```

  Note: The physical path need not be canonical. The definition is silently ignored if the physical path does not exist.

- The secondary purpose is to control the case sensitivity for file names, using the options -nocase or -case. With -nocase, all names are canonicalized to lowercase, which is useful when the vovserver is running on a Windows NT machine.
- The third purpose is to control whether the AFS paths should be supported. If the -afs option is used, then all paths of the type `/.automount/hostname1/root/aaa` become `/net/hostname1/aaaa`

The procedure `vtk_equivalence` also has side effects:

- The environment variable corresponding to the logical name is set, if it does not exist already (that is, the variable *$env(TOP)*).
- The Tcl global variable corresponding to the logical name is set to the value of the environment variable (that is, the variable *$TOP*).

Example Uses of vtk_equivalence

See the following example:

```
# -- HOME should not be used in multi-user projects, because it has a
# -- different value for each user. Use it only in single-user projects.

vtk_equivalence HOME $env(HOME)

# -- VOVDIR is always defined and these equivalences are always useful.

vtk_equivalence VOVDIR $VOVDIR
vtk_equivalence VOVDIR $VOVDIR/../common

# -- Data directories.

vtk_equivalence TOP /export/projects/cpu; # This is for Unix
vtk_equivalence TOP p:/cpu              ; # This is for Windows

# Uncomment this if you need AFS paths.

# vtk_equivalence -afs
```

For another example of equivalence file, see the default file for the "generic" project type `$VOVDIR/local/ProjectTypes/generic/equiv.tcl`.

Define Host-specific Overrides for the Server-side Equivalence Cache

The server-side equivalence cache can be accessed via the VTK Tcl API using `vtk_equivalence_get_cache OPTION`. When passing a host name in for `OPTION`, the equivalences for that host will be returned. When passing an empty string in for `OPTION`, the list of host names that have entries is returned. By default, a special host name of "_default_" is used for the server-side cache that applies to all clients.

The server-side equivalence cache can be set with via the VTK Tcl API using `vtk_equivalence_set_cache HOSTNAME VALUES`, where `HOSTNAME` is the name of a host or the "_default_", and `VALUES` is a Tcl list with an even number of elements in the form `LOGICAL_NAME PHYSICAL_PATH.`

```
vtk_equivalence_set_cache lin0201 "HOMES /homes VOVDIR /tmp_mnt/tools/rtda/current/"
```

The equivalences can also be viewed and managed via the web UI on the Equivalences page.

Historical Job Data Files

vovserver creates a CSV (comma-separated variable) file containing data about each job that ran. This format may be directly imported into many spreadsheet and database programs. The jobs files are stored in the `jobs` subdirectory of the vovserver configuration directory.

For example, for the default setup of Accelerator, the job files are stored in the directory as shown below:

```
$VOVDIR/../../vnc/vnc.swd/jobs
```

The jobs files are rotated automatically each day by the vovserver, and older ones are compressed. Each file is named according to the day it applies to, in the form `YYYY.MM.DD`, where `YYYY` is the year, `MM` is the 2-digit month, and `DD` is the two-digit day of the month.

Each file contains a few header lines that identify the format version and the order of the fields. In the following example, the backslashes indicate line breaks inserted for readability. The comment denoting the fields is actually one long line. The available fields may vary by jobs file version as shown below:

```
# V 1
# T 1173942608
# FIELDS: ID,JOBCLASS,PROJECT,GROUP,USER,OSGROUP,DP,DIR,ENV,TOOL,JOBNAME,
SPRIORITY,XPRIORITY,RESOURCES,GRABBED,LICENSES,SUBMITHOST,EXEHOST,DPHOSTS,
SUBMITTIME,STARTTIME,ENDTIME,STATUS,EXIT,MAXRAM,CPUTIME,QUEUE
```

Jobs File Fields Description

The following table shows the type and meaning of the fields in the jobs file.

Field Name	Data Type	Description
ID	%d	Numeric ID of the job

Field Name	Data Type	Description
JOBCLASS	%s	Name of jobclass in which job was submitted; may be empty
PROJECT	%s	Name of management project on behalf of which job was run
GROUP	%s	Name of FairShare group in which job was run
USER	%s	Name of user as which the job ran
OSGROUP	%s	List of user.group
DP	%d	Boolean; 1 if distributed parallel job, 0 if not
DIR	%s	Working (run) directory of the job
ENV	%s	Name of the environment in which the job was run
TOOL	%s	Name of the program (first word on command line) that the job ran
JOBNAME	%s	User-assigned name of the job; may be null
SPRIORITY	%d	User-assigned scheduling priority of the job
XPRIORITY	%d	User-assigned execution priority of the job
RESOURCES	%s	Resources requested by the job
GRABBED	%s	Resources actually assigned to the job, after all mapping
LICENSES	%s	License features used by the job
SUBMITHOST	%s	Name of host from which job was submitted
EXEHOST	%s	Name of host on which job was executed
DPHOSTS	%s	List of host names on which distributed parallel job was executed

Field Name	Data Type	Description
SUBMITTIME	%d	Time job was submitted, seconds since the Epoch
STARTTIME	%d	Time job started running, seconds since the Epoch
ENDTIME	%d	Time job finished running, seconds since the Epoch
STATUS	%s	String describing status of the job, e.g. 'Done'
EXIT	%d	Exit status of the job, e.g. 0
MAXRAM	%d	Maximum RAM used by the job, in MBytes
CPUTIME	%d	Cumulative CPU time used by the job, in seconds
QUEUE	%d	Time the job waited in queue, in seconds

Journals

The vovserver records all events in a journal file that resides in a the subdirectory `journals/` of the server configuration directory.

Each journal has a name in the form `YYYY.MM.DD.jrn`. A new journal is started each day; older journals are compressed automatically, but not removed. For a long-running VOV project, it may be necessary to set up the vovcontrab or another means to manage the size of the journal's directory.

In the current release, the journals are intended for machine consumption, and are terse and cryptic. These journals are to be used for auditing and troubleshooting.

The journals can be browsed on the Journals page. This page can display only the events that are related to a specific node, or all events. The events are arranged into groups by timeslice.

Alerts and Notifications

Notification daemon: vovnotifyd

The `vovnotifyd` daemon is used to deliver notifications to selected recipients about functions that are related to job events.

On Accelerator, this delivery function is associated with the MAILTO property of the jobs. (This association does not apply to other Altair Accelerator products.)

The notifications are related to health checks, such as taskers that are down, jobs that are stuck or waiting too long, and so on. These conditions are detected by Accelerator or Monitor. Predefined system health check procedures are included with the Altair Accelerator.

The content and the email delivery of the notifications can be configured by using the forms and text fields on the browser, or by creating and editing files that use CLI commands.

The following table lists the files that are related to `vovnotifyd`.

Table 3: Summary of vovnotifyd Files

Config files	`vnc.swd/vovnotifyd/config.tcl`
	`vnc.swd/vovnotifyd/config_smtp.tcl`
	`vnc.swd/vovnotifyd/config_export.tcl`

Info file	`vnc.swd/vovnotifyd/info.tcl`

Auxiliary files	`$VOVDIR/tcl/vtcl/vovhealthlib.tcl`
	`$VOVDIR/local/vovhealthlib.tcl`
	`vnc.swd/vovnotifyd/vovhealthlib.tcl`

Timing of Notifications

You can use the TIMEVAR definitions to control the timing at which license expiration and other emails are sent.

In vovhealthlib.tcl, you can define a TIMEVAR such as:

```
source $env(VOVDIR)/tcl/vtcl/vovflexlmdlib.tcl
package require vovurlutils

global HEALTHLIB_PRODUCT_MAP HEALTH_PROCS env

set HEALTHLIB_PRODUCT_MAP(doTestHealthMyFeature) "nc"
# registerHealthCheck doTestHealthMyFeature -checkfreq 10 -forceCheckfreq -mailfreq
 10 -forceMailfreq
lappend HEALTH_PROCS(list) doTestHealthMyFeature

proc doTestHealthMyFeature { args } {
    global HEALTH_PROCS
    set homeUrl $HEALTH_PROCS(homeURL)
    # VovMessage "now running doTestHealthMyFeature"
    set subject "TestMyHealthFeature"
    set body "\nHi,\n\nWe ran TestMyHealthFeature.\n"
    doMailNotify doTestHealthMyFeature "@ADMIN" $subject $body
}

TIMEVAR doTestHealthMyFeature {
    Tue {
        suppressMail doTestHealthMyFeature 1
    }
    06:00-08:00 {
        suppressMail ALL 0
        suppressMail doTestHealthMyFeature 0
    }
    default {
        suppressMail ALL 1
        suppressMail doTestHealthMyFeature 1
    }
}
```

Where calling `suppressMail` with a "1" for the specified health check will suppress the mailings for that TIMEVAR. `suppressMail` accepts either the name of a specific health check, or "ALL" to control all defined health check routines.

Configure vovnotifyd via the Browser

1. On the menu bar, click the Administrations icon (which looks like a gear).
   This takes you to Admin page.
2. In the left column on the Admin page under Administration, select **- Daemons**.
3. In the Daemons page, select **config** for `vovnotifyd`.
4. The next windows provides the options to edit, enable and disable the desired features: Health Checks, SMTP Configuration or E-Mail Maps
   After configurations have been set, the option to view the current configurations will be available in the Config File column: Show config file.

   CAUTION: Configurations can be modified in the this text field. To avoid errors, it is recommended to instead configure the parameters in the GUI fields that are described below.

Health Checks
By default, all procedures are monitored. Procedures that are designated as not required can be disabled.

SMTP Configuration
SMTP Configuration is used to configure the notification system.

Note: To query LDAP for email addresses, LDAP must first be configured. For details about LDAP configuration, refer to *LDAP Integration*.

E-Mail Maps

E-Mail Maps are used to add, update or remove the email addresses of the users that receive notifications. By default, email addresses are sent directly via either the user ID or user@sourcedomain. In addition, alternate email addresses can also be entered per recipient.

Configure vovnotifyd via the CLI

Note: For information about configuring health checks, refer to *Health Monitoring and vovnotifyd*.

1. To manually configure `vovnotifyd`, create the directory `vovnotifyd` inside the server working directory (`.swd`).
2. Copy the configuration file template into the newly-created directory.

```
cp $VOVDIR/etc/config/vovnotifyd/config.tcl
```

3. Modify the configuration template to match the settings of your mail server environment.

```
% cd `vovserverdir -p .`
% mkdir vovnotifyd
```

Example of the `$VOVDIR/etc/config/vovnotifyd/config.tcl` file:

```
# Notification configuration file.
# Should be placed in the vovnotifyd directory of the .swd.
# All settings are required unless specified otherwise.
# Unused optional settings should be commented out.

# Create an e-mail address map, stackable, optional
addUserToEmailAddressMap  rtdamgr john@mydomain.com

### AltairMonitor-specific settings
# See notification configuration documentation in Altair Monitor Admin guide
# ConfigureTag     TAG OPTION VALUE
# ConfigureFeature FEATURE OPTION VALUE

### Examples:
# ConfigureTag MGC -poc { john mary }
# ConfigureFeature EDA/MATLAB -longcheckout 2d -userlongcheckout john 1w -mincap
 5 -triggerperc 90
# ConfigureFeature SIMULINK -poc bob -mincap 10 -triggeruse 12
```

4. To start the daemon, either enter the command `nc cmd vovdaemonmgr start vovnotifyd`, or start it manually from the `vovnotifyd` directory as shown below:

```
% vovproject enable vnc
% cd `vovserverdir -p vovnotifyd`
% vovnotifyd
```

Autostart vovnotifyd

In the directory `vnc.swd/autostart` set up your script `start_vovnotifyd.tcl` to run with autostart:

```
% cd `vovserverdir -p .`
% mkdir autostart
% cp $VOVDIR/etc/autostart/start_vovnotifyd.tcl autostart
```

Configure Email Addresses

Use the `config.tcl` file to set the email addresses to be used for each user.

Two methods are available:

- Call addUserToEmailAddressMap USERNAME EMAIL
- Override the entire procedure getEmailAddress

```
# Fragment of vovnotifyd/config.tcl file.

# Method 1.
addUserToEmailAddressMap john  John.Smith@my.company.com

# Method 2. Assume you can get an address from LDAP
#           The LDAP subsystem needs to be configured.
proc getEmailAddress { user } {
    set email [VovLDAP::getEmail $user]
    if { $email != "" } {
       return $email
    } else {
       return $user
    }
}
```

Write Localized Health Checks

The `vovnotifyd` command runs in the `vovsh` binary, so all the VTK API procedures are available to you.

The standard checks procedures are defined in the file `$VOVDIR/tcl/vtcl/vovhealthlib.tcl`.

The health check procedures are loaded by using a search path. They are loaded first from the file given above, then from `$VOVDIR/local/vovhealthlib.tcl`, and last from `vovhealthlib.tcl` in the `vovnotifyd` working directory. This permits redefining health check procedures on a site-wide or a project-specific basis.

Note: Local procedure names should begin with '`doTestHealth`' such as the system names; some platforms require on this convention.

To have local procedures shown by the browser UI, add a line into your `vovhealthlib.tcl` file. An example is shown below:

```
set HEALTHLIB_PRODUCT_MAP(doTestHealthYourProcedure)        "nc"
```

The product names are those returned by the procedure `vtk_product_get_info -name`: nc, lm, ft, wa.

Ensure your procedures are robust and handle error conditions, such as catching all `exec{}`, `open{}` and other procedures that may fail.

Note: Changes made to any of the `vovhealthlib.tcl` files will not take effect until the `vovnotifyd` daemon is restarted.

Alternate Method of Sending Email

If SMTP is not available, a mailer program could be used instead.

Use a Mailer Program

The standard `sendMail` procedure checks the Method setting in the SMTP Configuration section of the `vovnotifyd` configuration web UI. The method may be set to **PROGRAM** in the web UI to allow you specify a custom mailer command.

Notification of Job Status

The Accelerator `vovnotifyd` notification daemon accesses the server's event stream and then sends a notification for jobs that request it.

To enable this notification, the MAILTO property must be set: use the option -m or -M option with the `nc run` command. An example is shown below:

```
% nc run -m sleep 10
% nc run -M ":ERROR"  simulate chip.spi
```

The format of the property of MAILTO can be configured as follows:

```
recipientList
recipientList : verbList
recipientList : ALL
: verbList
```

recipientList is the list of the e-mail recipients. *verbList* is the list of verbs for which notifications must be sent. The supported verbs are listed below.

```
DESCHEDULE - Job has been dequeued.
DISPATCH   - Job has left the queue and has been routed to an execution host.
ERROR      - Job has exited with a failure.
FORGET     - Job has been forgotten.
RESUME     - Job has been resumed.
STOP       - Job has exited successfully.
SUSPEND    - Job has been suspended.
```

If the *recipientList* is empty, a notification is sent to the owner of the job. If the *verbList* is empty, then a notification is sent only when the job terminates.

For example:

```
john : ERROR   - Send mail to the user 'john' if the job terminates in error.
```

```
: STOP ERROR   - Send mail to the job owner when the job terminates.
john mary: ALL - Send mail to the users 'john' and 'mary' for anything that happens
 to the job.
```

Change the MAILTO Property After Job Submission

To change the MAILTO property, use the `vovprop` utility. The following are examples of getting, setting, and deleting the property:

```
% nc cmd vovprop get 000012345 MAILTO
% nc cmd vovprop set -text 000012345 MAILTO "mary : STOP ERROR"
% nc cmd vovprop delete 000012345 MAILTO
```

Job Status Triggers

The daemon `vovtriggerd` taps the event stream and executes commands that are based on selected events.

A typical application is updating an external SQL database when a job is completed.

Triggers are different from post-commands. Triggers are executed by `vovtriggerd`, which is normally run by the user who owns vovserver. The owner of the account that was used to start vovserver is the *owner* of vovserver. Post-commands are executed by the user who owns each job.

The following table summarizes the information about `vovtriggerd`:

Config file	`vnc.swd/vovtriggerd/config.tcl`
Sample config file	`$VOVDIR/etc/config/vovtriggerd/config.tcl`
Info file	`vnc.swd/vovtriggerd/info.tcl`

Set Up vovtriggerd

`vovtriggerd` is a daemon that is configured as follows:

- Create a subdirectory called `vovtriggerd` in the server configuration directory
- Create a configuration file called `config.tcl` with the main purpose of overriding the procedure called `triggerCallBack`
- Start the daemon:

```
% mkdir `vovserverdir -p vovtriggerd`
% cd `vovserverdir -p vovtriggerd`
% mkdir autostart
% cp $VOVDIR/etc/config/vovtriggerd/config.tcl .
% vovdaemonmgr start vovtriggerd
```

The TRIGGER Property

The default trigger handler looks for the property `TRIGGER` attached to the object mentioned in the event. If the property exists, it is assumed to be the name of a trigger procedure to be called. There are three arguments for the trigger procedure: `id`, `subject`, `verb`.

The trigger procedures are defined in the `config.tcl` file. Following are the guidelines for implementing a trigger:

- The trigger is stateless.
- The trigger is fast; it should complete within a few seconds.

Following is an example of using the `TRIGGER` property:

- Create a trigger call-back in the `config.tcl` file. In the following example, it is named `trigShowJobEventCB`.

```
#
# This goes in PROJ.swd/vovtriggerd/config.tcl
#
proc trigShowJobEventCB { id subject verb } {
    puts "TrigShowJobEventCB: Just got the event $id $subject $verb"
}
```

- Attach the property TRIGGER to jobs in the flow. Example:

```
% vovprop set -text 000123456 TRIGGER trigShowJobEventCB
% vovprop set -text 000234567 TRIGGER trigShowJobEventCB
```

Trigger Events

Following are the events that are processed by `vovtriggerd`:

- JOBID "JOB" "DISPATCH", when the job is dispatched to a tasker.
- JOBID "JOB" "ERROR", if the job fails.
- JOBID "JOB" "STOP", if the job succeeds.

Handling the OVERFLOW Event

If the `vovtriggerd` daemon receives an overflow event (the verb is the string `OVERFLOW`), the procedure `overflowCallBack` is called with no arguments. The overflow event is an indication of a buffer overflow inside the vovserver, which is typically caused by `vovtriggerd` being too slow in processing the events. Depending on the situation, it may be useful to reinitialize the trigger callbacks.

Example of Submission of Jobs with Triggers

The following example applies to Accelerator:

A trigger can be submitted by setting the `TRIGGER` property. Knowing the name of the trigger callback routine to call is required. In the following example, the name of the trigger callback is `updateDbCallBack`.

```
% nc run -P TRIGGER=updateDbCallBack sleep 10
```

```
# Example of updateDbCallBack
# This procedure is defined in *.swd/vovtriggerd/config.tcl
# Here we update a table called "mytable" based on the
# value of a property called MYPROP.
proc updateDbCallBack { jobid subject verb } {
    switch $verb {
        "STOP" - "ERROR" {
            if [catch {set value [vtk_prop_get $jobid "MYPROP"]}] {
                set value -1
            }

            set    stmt "INSERT INTO mytable (id,value)"
            append stmt " VALUES ( $jobid, $value)"
            VovSQL::init
            set handle [VovSQL::open]
            VovSQL::query $handle $stmt
            VovSQL::close $handle
        }
    }
}
```

Alerts

VOV issues an "alert" when an event requires attention. An alert can range from information that does not require action to an urgent fault that requires immediate action.

Depending on the event that occurs, an alert may require attention from a system administrator.

VOV supports four alert levels, which are defined in the table below.

INFO	Information only, no action required.
WARN	Warning, a limit is about to be reached.
ERROR	A fault in one of the subsytems. Example: a syntax error in one of the configuration files.
URGENT	A major fault that compromises the behavior of the system and requires immediate attention. Examples: a license violation or a disk full condition.

Alerts can be viewed on the browser or the command line interface (CLI). In addition, alerts are stored in log files.

For the administrator, VOV permits two actions with respect to an alert:

- Acknowledge the alert.
- Delete the alert from view.

Note: Every alert is stored in a log file; deleting an alert from viewing does not delete the record of the alert in the log file.

Maximum Number of Alerts

The vovserver keeps up to a defined maximum number of alerts in view. The maximum number is defined by the parameter *`alerts.max`*. The default value is 50. If the number of alerts exceeds the maximum, the oldest alert with the lowest level is deleted from the view.

Note: The record of the alert is not deleted from the log file.

Manage Alerts

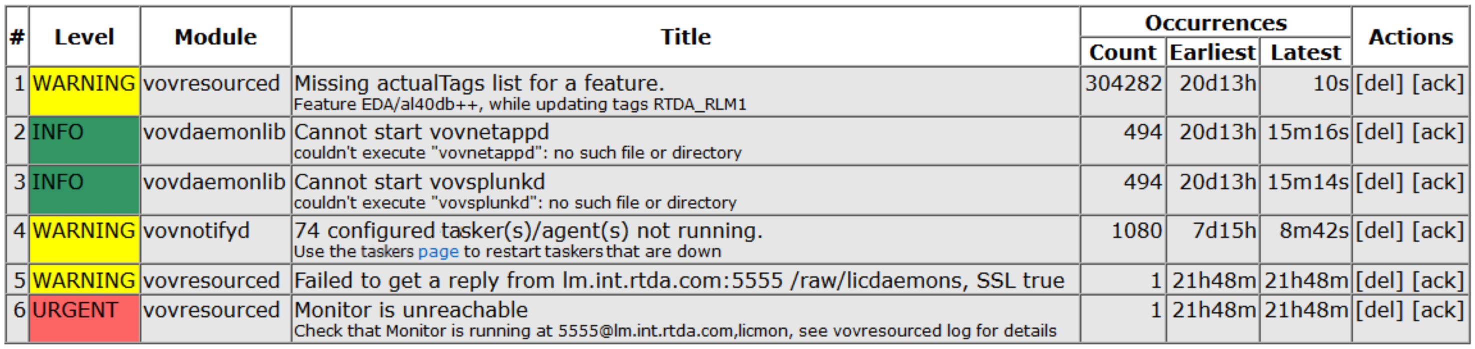
For viewing alerts, the level of the most severe alert is visible in the title bar of the browser user interface and in the VOV GUI. The most recent alerts can be viewed from the command line interface with the following commands:

- `vsi` for short format
- `vovshow -alerts` for full format

The following is an example of alerts as shown by `vsi`:

```
Alerts:
URGENT          Imminent license violation: us      3   12d20h   12d20h
WARNING         License is expiring in less th   2613   12d20h    9d13h
ERROR           License is expiring in less th    558    9d13h    8d13h
URGENT          License has expired               270    8d13h    8d01h
URGENT          License violation                  24    8d13h    8d01h
WARNING         License violation: too many sl    336    8d13h    8d01h
```

The following example shows the same alerts as seen on the browser:



#	Level	Module	Title	Occurrences			Actions
				Count	Earliest	Latest	
1	WARNING	vovresourced	Missing actualTags list for a feature. Feature EDA/al40db++, while updating tags RTDA_RLM1	304282	20d13h	10s	[del] [ack]
2	INFO	vovdaemonlib	Cannot start vovnetappd couldn't execute "vovnetappd": no such file or directory	494	20d13h	15m16s	[del] [ack]
3	INFO	vovdaemonlib	Cannot start vovsplunkd couldn't execute "vovsplunkd": no such file or directory	494	20d13h	15m14s	[del] [ack]
4	WARNING	vovnotifyd	74 configured tasker(s)/agent(s) not running. Use the taskers page to restart taskers that are down	1080	7d15h	8m42s	[del] [ack]
5	WARNING	vovresourced	Failed to get a reply from lm.int.rtda.com:5555 /raw/licdaemons, SSL true	1	21h48m	21h48m	[del] [ack]
6	URGENT	vovresourced	Monitor is unreachable Check that Monitor is running at 5555@lm.int.rtda.com,licmon, see vovresourced log for details	1	21h48m	21h48m	[del] [ack]

Figure 17:

If viewing the documentation from a live vovserver, refer to the Alerts page.

Alerts are also logged in the logs directory in files with names that are formatted as `alert.YYYY.MM.DD.log`. Old alert files are compressed.

Some alerts may not require immediate action. However, it is good practice to acknowledge the alert. The `[ack]` link on the alerts page can be used to indicate that the alert has been acknowledged. The login name of the person acknowledging the alert will be shown on the Alerts page.

Clear Alerts from View

An alert is automatically cleared from view about one day after the last occurrence that triggered the alert. A selected alert can be removed from view by using the `[del]` link from the browser UI.

All alerts can also be cleared from view with the following command:

```
% vovforget -alerts
```

Tcl API

There are two Tcl API procedures in `vovsh` that handle alerts:

- `vtk_generic_get alerts A` - Get alert data into array A
- `vtk_alert_add sev title` - Add an alert

To add an alert from the Tcl interface, use the command `vtk_alert_add`.

To get data for all alerts in Tcl, use the command

```
vtk_generic_get alerts array
```

The following code example shows how `vsi` formats the alerts:

```
# This is how the vsi cmd formats alerts
vtk_generic_get alerts alerts
if { $alerts(count) > 0 } {
    append output "\nAlerts:\n"
    for { set i 0 } { $i < $alerts(count) } { incr i } {
        append output [format "  %-10s %-10s %-30s" $alerts($i,level)
                        [string range $alerts($i,module) 0  9]
                        [string range $alerts($i,title)  0 29] ]
        if { $alerts($i,count) > 1 } {
            append output [format "%7d %8s %8s"
                            $alerts($i,count)
                            [vtk_time_pp [expr $now - $alerts($i,first)]]
                            [vtk_time_pp [expr $now - $alerts($i,last) ]] ]
        }
        append output "\n"
    }
    append output "\n"
}
```

System Tasks

Run Periodic Tasks with vovliveness

If the directory "tasks" exist in the Server Working Directory, the server calls the `vovliveness` script once per minute.

The script executes all the tasks contained in the "tasks" directory.

```
vovliveness: Usage Message

  DESCRIPTION:
      This script is called by vovserver about once a minute.
      It can be used to perform maintenance tasks.

  USAGE:
      % vovliveness [OPTIONS] <taskdirectory> <timestamp>

  WHERE:
     task_directory     -- is the directory with the tasks
                           to be executed. The tasks are those
                           that match the expression "live_*.tcl".
     timestamp          -- Currently ignored.

  OPTIONS:
     -v                 -- Increase verbosity.
```

There are many uses for `vovliveness`. Examples are available in the directory `$VOVDIR/etc/liveness`.

To activate this functionality, create the directory `tasks` and add some tasks files with a name matching the expression `live_*.tcl`. The Tcl interpreter has access to all `vtk_*` procedures. Example:

```
% cd `vovserverdir -p .`
% mkdir tasks
% cd tasks
% cp $VOVDIR/etc/liveness/live_start_taskers.tcl .
```

Following an example of the script `live_start_taskers.tcl` to restart any down taskers, once per hour:

```
#
# Copyright © 2007-2021, Altair Engineering
#
# All Rights Reserved.
#
# Directory : src/scripts/liveness
# File      : live_start_taskers.tcl
# Content   : Start down taskers once an hour.
# Note      :
#
# $Id: //vov/branches/2019.01/src/scripts/liveness/live_start_taskers.tcl#3 $
#

set now [clock seconds]
```

```
# Get or initialize period
if { [catch {set period [vtk_prop_get 1 LIVE_START_TASKERS_PERIOD]}] } {
    set period 3600
    catch {vtk_prop_set 1 LIVE_START_TASKERS_PERIOD $period}
}

# Get age
if { [catch {set lastRun [vtk_prop_get 1 LIVE_START_TASKERS_LAST]}] } {
    set lastRun 0
}
set age [expr {$now - $lastRun}]

if { $age >= $period } {

    # Start down taskers
    if { [catch {exec vovtaskermgr start >&@ stdout} errmsg] } {
        VovError "Failed to start taskers: $errmsg"
    }

    # Reset the last run TS
    catch {vtk_prop_set 1 LIVE_START_TASKERS_LAST $now}

}
```

Alerts from Liveness Tasks

Alerts may occur that are related to liveness tasks such as "The previous liveness script is still connected", especially in Monitor.

Note: In previous releases, there is no control these occurrences; such occurrences cause no harm.

The liveness tasks system is designed to support short jobs that are triggered frequently (about once per minute) by the vovserver so long as it is running. It was also used for the database loading task for Monitor checkouts and Accelerator jobs; sometimes these jobs run significantly longer.

In later releases the debuglog parsing, batch reports and other maintenance items are converted to periodic jobs that run on a dedicated vovtasker named 'maintainer', so these alerts should no longer appear.

Run Periodic Tasks with vovcrontab

The UNIX utility `crontab` is used to perform regularly scheduled tasks such as retracing an entire project each night or storing a back-up of the trace every Saturday. `vovcrontab` is a VOV utility that simplifies the creation of cron rules for a project. Directions are provided in this section.

Usage: voncontrab

```
vovcrontab: DESCRIPTION:
vovcrontab:     Interface to the UNIX utility crontab.
vovcrontab:
vovcrontab: USAGE:
vovcrontab:     % vovcrontab [option]
vovcrontab:
vovcrontab: OPTIONS:
vovcrontab:   -help         -- Get this message
vovcrontab:   -new          -- Install the crontab for this project
vovcrontab:                    Also used to update the scripts/vovdir.csh script.
```

```
vovcrontab:   -noautostart -- Do not install autostart script to update
vovcrontab:                   vovdir.csh; the default is to install it.
vovcrontab:   -reinstall   -- Reinstall current crontab for this project
vovcrontab:   -clear       -- Clear the current crontab
vovcrontab:   -show        -- Show the crontab
vovcrontab:   -type <type> -- Specify project type
vovcrontab:
vovcrontab: NOTE:
vovcrontab:     Please remember to copy
vovcrontab:     $(VOVDIR)/etc/autostart/update_crontab_vovdir.csh
vovcrontab:     into your autostart directory if needed.
vovcrontab:     It is installed by default.
```

Enable a Project

Enable a project in a shell via:

```
'vovproject enable <PROJECT>'
```

Create crontabs

Execute `'vovcrontab -new'` to create crontabs.

```
% vovcrontab -new
vovcrontab: Creating vnc.swd/scripts/cron.csh
vovcrontab: Creating cron table vnc.swd/crontab.lion
no crontab for john
vovcrontab: Installing new crontab.
vovcrontab: Installing updated crontab
```

This program prepares the scripts `$SWD/scripts/cron.csh` and `$SWD/crontab.hostname`.

Display Current crontab Definition

Running `vovcrontab -show` shows the current crontab definition.

```
% vovcrontab -show
#### (vovcrontab) START PROJECT vnc ####
#
# ... some lines omitted...
#
# Every hour at 5 minutes before the hour.
55 * * * *  /home/john/vov/vnc.swd/scripts/cron.csh hourly
#
# Every day: at 23:15
15 23 * * *     /home/john/vov/vnc.swd/scripts/cron.csh daily
#
# Every week: on Saturday at 7:00am
0  7  * * 6     /home/john/vov/vnc.swd/scripts/cron.csh weekly
#
# Every month: on the 1st at 3:00am
0  3  1 * *     /home/john/vov/vnc.swd/scripts/cron.csh monthly
#### (vovcrontab) END PROJECT vnc ####
```

Customize the crontab

The crontab can be customized by editing either `$SWD/crontab.hostname` or `$SWD/scripts/cron.csh`. Afterwards, `vovcrontab -reinstall` will need to be run to take the modifications into consideration.

```
% vovcrontab -reinstall
vovcrontab: vnc.swd/scripts/cron.csh  exists already.
vovcrontab: vnc.swd/crontab.lion exists already.
vovcrontab: Installing new crontab.
vovcrontab: Installing updated crontab
```

Delete the Current crontab Definition

To delete current crontab definitions, use:

```
'vovcrontab -clear'
vovcrontab: vnc.swd/scripts/cron.csh  exists already.
vovcrontab: vnc.swd/crontab.lion exists already.
vovcrontab: Removing the crontab
```

Complete the Cleanup

To complete the cleanup, remove the `crontab.HOSTNAME` file in the SWD directory of the project.

```
% rm crontab.lion
```

vovgetnetinfo

The program `vovgetnetinfo` is used with Accelerator and Monitor to fill in information about the hosts. `vovgetnetinfo` is to be run periodically from the scripts created by the `vovcrontab` command.

`vovsh -s netinfo` is run as a system job on each of the vovtasker hosts, which gathers information about clock offset, filesystems, OS version, memory, etc., and sends information to the vovserver.

```

vovgetnetinfo: Usage Message

USAGE:
    % vovgetnetinfo [OPTIONS]

OPTIONS:
    -autoforget   -- Set the autoforget flag on the jobs created by this script.
    -delay <ms>   -- Add a delay between submission of jobs. In milliseconds.
    -h            -- Help usage message.
    -hosts <list> -- Restrict operation to list of named hosts. Requires a
                     tasker on each host.
    -netinfo      -- Compute host and filesystems information (default).
    -nolog        -- Disable output log. Unless disabled, an output log will be
                     written in the parent of the SWD.
    -procinfo     -- Compute process status information.
    -v            -- Increase verbosity.

EXAMPLES:
   % vovgetnetinfo -h
   % vovgetnetinfo -procinfo
```

```
    % vovgetnetinfo -netinfo
```

`vovgetnetinfo` can also be run as a single time from the command line as shown below:

```
% nc cmd vovgetnetinfo
```

Information about a single host can be updated as follows:

```
% rsh some-host-name nc cmd vovsh -s netinfo
```

vovinfo

A utility related to `vovgetnetinfo` is `vovinfo`, which performs the same function as `vovsh -s netinfo`, but with a smaller binary. This binary can only get host and process information, and does not have Tcl/TK or the VTK API.

`vovinfo` is to be used to provide monitoring and can run standalone in an infinite loop to provide periodic updates to a vovserver. The following example updates the vovserver `vnc@jupiter:6271` with clock offset information every hour for 60 days. The environment variable settings do not require access to the server's `.swd` directory.

```
% setenv VOV_SWD_KEY none
% setenv VOVEQUIV_CACHE_FILE vovcache
% vovinfo -all h jupiter -p vnc -P 6271 -l 3600 -i 1440 &
```

```
usage: vovinfo [-ac] [-s what] [-p project] [-h host] [-P port] [-l looptime]
                    [-i iterations] [-Vv]
   -a:         Ignore PROCS_TO_TRACK property and track all processes if
               procinfo is enabled.
   -c:         Use cpu_* licensing instead of host_* licensing (deprecated)
   -s:         What info to get: netinfo hostinfo procinfo clockinfo all. May
               be repeated. The keywords 'netinfo' and 'hostinfo' are
               equivalent.
   -p:         The project name (or the value of VOV_PROJECT_NAME
   -h:         The host name (or the value of VOV_HOST_NAME
   -P:         The port number (or the value of VOV_PORT_NUMBER)
   -l:         The looptime (a time-specification): defaults to 0
   -i:         Exit after this many iterations (use with -l)
   -V:         Print version and exit
   -v:         Verbose flag
```

The `vovinfo` program is licensed, and must be run by a user having enough privilege with respect to the vovserver to update the server's clock, process, and other information about the server.

Manage Processes

This section describes how to use the command line and find all processes that are not currently managed by Accelerator. VOV can use a vovtasker to collect information about all processes from all hosts in a farm.

Processes that are descendents of vovtasker, orphans of vovtasker, and external processes can also be found. Foster jobs can be created for discovered orphans; these jobs can be accounted for by a tasker on the same host, and tracked for the rest of their lifetime.

```
vovprocessmgr: Usage Message

  USAGE:
      % vovprocessmgr [OPTIONS]

  Report on and manage processes on hosts where an Altair Engineering vovtasker
  is running, for example, in Accelerator.

  OPTIONS:
      -h              -- Show brief help.
      -v              -- Increase verbosity.
      -w              -- Wide output (tab-separated, no truncation in names).
      -refresh [-orphans [-host HOST[,HOST]...] [-nohost HOST[,HOST]...]]
                         -- Refresh the process info. Refreshes all process info
                         unless the -orphans option is also passed, which
                         refreshes the process info for orphaned processes only.
                         The -host/-nohost options apply when refreshing orphaned
                         processes only, otherwise, all hosts are included.
                         Orphaned processes are determined by the presence of the
                         VOV_JOBID variable in the environment of the process.
                         Note that this is an asynchronous operation that is sent
                         to remote taskers, requesting them to gather and send
                         process information to the server. The timeliness of the
                         response depends on the loading of both the taskers and
                         of the server. For this reason, some amount of time
                         should be allowed between a refresh request and
                         reporting on processes of any type. For reports
                         involving only a few taskers, this could be measured in
                         seconds. For requests involving hundreds or thousands of
                         hosts, it may take several minutes for every tasker to
                         report in. Refreshing process info is an expensive
                         operation that can result in a significant amount of
                         communication and loading on the vovserver process and
                         therefore should be used only when necessary.
      -external       -- Filter to processes that are not an descendant of
                         tasker.
      -descendants    -- Filter to current descendant processes of tasker.
      -orphans        -- Filter to former descendant processes of tasker. For
                         accurate results, refresh the process info using the
                         -orphans option prior to running an orphan report. An
                         orphan report will also include fostered jobs as well.
                         Note that if orphan processes are common, it is
                         recommended to enable automatic child process cleanup
                         via the tasker.childProcessCleanup configuration
                         parameter in the policy.tcl file.
      -fostered       -- Filter to orphans currently being fostered.
      -all            -- Show all processes.

      -user "USER[,USER]..."              -- Filter to specified users.
      -host "HOST[,HOST]..."              -- Filter to specified hosts.
      -exe  "executableName[,exname]..."  -- Filter to specified executable
                                             names.

      -noheader      -- Suppress header.
      -noresv        -- Exclude reserved taskers.
      -noexternal    -- Exclude processes that are not an descendant of tasker.
      -nodescendants -- Exclude current descendant processes of tasker.
      -noorphans     -- Exclude former descendant processes of tasker.
      -nofostered    -- Exclude orphans currently being fostered.

      -nouser "USER[,USER]..."              -- Exclude specified users.
      -nohost "HOST[,HOST]..."              -- Exclude specified hosts.
      -noexe  "executableName[,exname]..."  -- Exclude specified executable
                                               names.

      -age   "TIMESPEC"  -- default age 10m.
      -maxrecursion "N"  -- Limit in recursive check of parents (default 100).

      -foster            -- Create a foster job for each top-most orphaned
                            process. Note that if orphan processes are common,
                            it is recommended to enable automatic child process
                            cleanup via the tasker.childProcessCleanup
                            configuration parameter in the policy.tcl file.

      -clear             -- Forget all information about processes from the
                            server (frees up memory). Must be the only option.

  EXAMPLES:
     % vovprocessmgr -refresh
     % vovprocessmgr -refresh -orphans
     % vovprocessmgr -orphans
     % vovprocessmgr -all -noexternal
     % vovprocessmgr -external -onlyuser john,mary,bob
     % vovprocessmgr -orphans -age 3h
     % vovprocessmgr -clear
```

The process information is accumulated in the vovserver and is released after approximately 5 minutes or until it is refreshed, whichever occurs first. To refresh information about all processes, use the following commands:

```
% vovproject enable vnc
% vovprocessmgr -refresh
```

`vovprocessmgr` sends a message to all the taskers to update the information about all processes and deliver the data collection to the vovserver. Sending all the data may take a few seconds.

Note: This command only works for the owner of Accelerator.

All processes can now be computed that are not children of a `vovtaskerroot` process with:

```
% vovprocessmgr -orphans
```

For example:

```
> vovprocessmgr -orphans
vovprocessmgr 07/08/2016 11:46:18: message: Analyzing 438 processes on 1 hosts that
 are older than 10m00s

  Mininum process age: 10m00s
  Exclude user:  apache avahi canna daemon dbus gdm
   haldaemon haldeamon htt mysql named nobody
   ntp oracle postfix postgres root rpc
   rpcuser smmsp xfs
  Filter to orphans

Host          Pid       User        Executable      Age       State     RAM       CPU
  Relation
titanus       23382     john        vovsh           S         10        10s       orphan
titanus       23383     john        vovsh           S         9         0s        orphan
titanus       23421     john        postgres        14d00h    S         212       5s
  orphan
```

To create foster jobs for discovered orphans:

```
% vovprocessmgr -orphans -foster
```

To track fostered jobs:

```
% vovprocessmgr  -fostered
```

To find only the processes that are older than a specified time, for example 1 day, use the option -age as shown below:

```
% vovprocessmgr -orphans -age 1d
```

Removing unwanted processes from the farm can be necessary. For security, `vovprocessmgr` only provides the list of suspected orphans. Only an administrator with root privileges has the authority to access the machines to kill the processes that were listed in the information.

Stopped Taskers and Foster Jobs

Taskers account for jobs running on a stopped tasker that is on the same host. When a tasker is started, if there is a matching tasker in the stopped condition (waiting on its jobs to finish), the new tasker will adopt any jobs on the stopped tasker by using foster jobs. This prevents host overloading.

Job Fostering

Job fostering is the processes of artificially consuming a job slot on a tasker in order to represent an externally running job or process.

The `vovfosterjob` utility runs as a job on a tasker and monitors an external job or process. When the external job or process exits, the `vovfosterjob` utility will also exit, freeing up the slot in which it was running.

vovfosterjob

```

vovfosterjob: Usage Message

  DESCRIPTION:
    A system utility to tell a tasker to watch a PID or a JOB by creating
    a foster job that runs for the lifetime of the entity being watched.
    Resulting foster jobs are stored in the System:Orphanage set, which
    can be displayed by passing the set name to 'nc list -alljobs -set'.
    This utility is mainly used to prevent overloading of a tasker that
    has been started on a host that also has a tasker in the process of
    shutting down gracefully. Such taskers will only exit once any jobs
    running on them are complete. Fostering such jobs onto the newly started
    tasker will result in job slots being consumed, preventing additional
    jobs from landing on the host.

  OPTIONS:
    -fromtasker TASKERNAME -- Foster all jobs on specified tasker. Normally used
                              to foster jobs on a tasker that has been requested
                              to gracefully stop after its jobs are finished.
                              This helps to prevent overloading if the tasker is
                              restarted before the job attrition process is
                              complete. Compatible only with, and requires, the
                              -totasker option.
    -h                     -- Help usage message.
    -host HOSTNAME         -- Specify host to which the foster job should be
                              dispatched. The resulting foster job will be
                              dispatched to the first tasker found on the
                              specified host. Compatible with the -pid and
                              -job options.
    -job JOBID             -- Foster a job. Without the -host or -totasker
                              option, the resulting foster job will be
                              dispatched to the first tasker found that is
                              running on the same host as the job being
                              fostered. Compatible with the -host and -totasker
                              options.
    -pid PID               -- Foster a process. Without the -host or -totasker
                              option, the resulting foster job will be
                              dispatched to the first tasker found that is
                              running on the same host where the vovfosterjob
                              command is executed. If the PID does not exist,
                              the foster job will be created but will exit
                              immediately. Compatible with the -host and
                              -totasker options.
    -stoppedtaskers TASKERNAME --
                           -- Foster jobs running on all prior instances of the
                              specified tasker that are in the process of
                              stopping gracefully. The resulting foster jobs
                              will be dispatched to the specified tasker. Not
                              compatible with any other option.
    -totasker TASKERNAME   -- Specify tasker to which the foster job should be
                              dispatched. Compatible with the -pid, -job, and
                              -fromtasker options.
    -v                     -- Increase output verbosity. Repeatable.

  EXAMPLES:
    % vovfosterjob -job 000123456
    % vovfosterjob -job 000123456 -totasker titan
    % vovfosterjob -pid 6789
    % vovfosterjob -pid 6789 -host titan
    % vovfosterjob -fromtasker titan_stopped -totasker titan
    % vovfosterjob -stoppedtaskers titan
```

Query the vovserver

The `vovselect` command provides a way to retrieve specific data from the vovserver, with filtering done on the server side. This method is differs from some of the VTK calls, which get all data and require processing on the client side to get the data of interest.

Attention: In an upcoming major release of Accelerator Products, the `vovselect *` wildcard select feature will be dropped. To prepare for this change, users should update scripts and REST requests to issue vovselect requests using a specified list of field names. For example:

```
nc cmd vovselect statusnc,id,command from jobs
```

An easy way to find out what fields are in an object type is by using `vovselect fieldname`. For example:

```
nc cmd vovselect fieldname from jobs
```

Many types of objects in the vovserver may be queried. See the help information below for the supported objects.

Note: `vovselect` supports the "*" wildcard to signify all fields of a particular object. Be sure to quote the * character as required by your shell, e.g.: vovselect '*' from jobs where idint==12345

Run `vovselect fieldname,fieldtype from <object>` to see the list of fields for that object. Multiple fields may be requested by separating them with a comma. Some fields represent a data collection that can be broken down using a format of FIELD.X, such as:

```
KEY.<KEYNAME>         (metric objects)
PARAM.<PARAMNAME>     (server object)
PROP.<PROPNAME>       (all objects)
RESOURCES.<RESNAME> (tasker objects)
```

`GRABBEDRESOURCES.<name>` only returns a value for the corresponding central resource when the job is currently running.

`SOLUTION.<name>` returns a value for the corresponding hardware resource after the job has started running. The value persists after the job has terminated.

`RESOURCES.<name>` attempts to determine a value for the corresponding requested resource. If the job is running, then the actual value of the allocated resource is returned. If the job is not running, the query will estimate a value by looking for the first matching value in the requested resource string. This may result in an underestimate, or an incorrect value.

For example, with the request: `-r "RAM/20 RAM/30", RESOURCES.RAM` may return `"20"` or `"50"` depending on the scheduling phase of the job. A contrived example which illustrates the difficulty of computing a value would be `-r "( RAM/100 CLOCK/10 ) OR ( RAM/50 CLOCK/20)"`. RESOURCES.RAM returns exactly the same value as REQRAM, and similarly for CORES, PERCENT, SLOTS & SWAP.

Examples

Queries you can run include the following:

- RESOURCES.<RESNAME> estimates the requested resource value of RESNAME. It attempts to determine a value for the corresponding requested resource. If the job is running, then the actual value of the allocated resource is returned. If the job is not running, the query will estimate a value by looking for the first matching value in the requested resource string. This may result in an underestimate, or an incorrect value.

For example, with the request `-r "RAM/20 RAM/30", RESOURCES.RAM` may return `"20"` or `"50"` depending on the scheduling phase of the job. A contrived example which illustrates the difficulty of computing a value would be `-r "( RAM/100 CLOCK/10) OR ( RAM/50 CLOCK/20 )"`. RESOURCES.RAM returns exactly the same value as REQRAM, and similarly for CORES, PERCENT, SLOTS & SWAP.

- GRABBEDRESOURCES.<RESNAME> returns the current value of RESNAME in a job's grabbed resources. It only returns a value for the corresponding central resource when the job is currently running.
- SOLUTION.<RESNAME> returns the value of RESNAME in a job's solution. It returns a value for the corresponding hardware resource after the job has started running. The value persists after the job has terminated.

For example, the following job submission

```
$ nc run -r "RAM/30 License:MATLAB/2" – sleep 1000
```

`vovselect` produces the following output:

```
$ nc cmd vovselect -header id,RESOURCES.RAM,RESOURCES.License:MATLAB from jobs
id RESOURCES.RAM RESOURCES.License:MATLAB
000001366 30 2
```

If the query is unable to determine a value, it will return an empty string.

If the query is used inside a 'where' clause, it may need to be quoted, such as, `-where '"grabbedresources.License:MATLAB">1'`.

For example, if the following job executes:

```
$ nc run -r License:MATLAB/2 – sleep 1000
```

`vovselect` produces the following output:

```
$ nc cmd vovselect -header id,statusnc,GRABBEDRESOURCES.License:MATLAB,SOLUTION.RAM
 from jobs -where
'"GRABBEDRESOURCES.License:MATLAB">1'
id statusnc GRABBEDRESOURCES.License:MATLAB SOLUTION.RAM
000001202 Running 2 20
```

vovselect

```
vovselect: Usage Message

    Utility to query vovserver data.

    USAGE:

      vovselect <FIELDSPEC> from <OBJECT> [OPTIONS]

    OPTIONS:

      -h               -- Show usage syntax.
      -v               -- Increase verbosity.
      -where <FILTER>  -- Filter the results.
      -order <COLUMN> [ORDER]  -- Sort the output by the specified column
                                  and ordering. Ordering is either "asc"
                                  (ascending)  or "desc" (descending).
                                  Default ordering is ascending.  When
                                  specifying an ordering, place the column
```

```
                                 and ordering in quotes, e.g.
                                 -order 'name desc'.
    -limit <N>       -- Limit the output to N rows.
    -distinct        -- Return distinct rows.

    -header          -- Displays column headers in the output.
    -cache 0/1       -- Control cacheing of query (default is 1).
                        Tech Note: use cache 0 for small results
                        (less than a few thousand rows)

    If option values contain shell-sensitive characters, such as ">",
    enclose them with single quotes (Linux) or double quotes (Windows).

    The from parameter will accept queryable object names (as listed by
    "vovselect objectname from objects"), individual object identifiers
    (as listed by "vovselect idint from <object>"), or set names
    (as listed by "vovselect name from sets").  This parameter can also
    accept the following:
       SUBSETS.<SETID>
       MATCHES.<RESMAPID>
       MATCHES.<RESMAPNAME>

  SUPPORTED OBJECTS:
    Run "vovselect objectname from objects" to see
    the list of queryable objects.

  SUPPORTED FIELDS:

    Run "vovselect fieldname,fieldtype from <object>" to see the list of
    fields for that object. To see a list of fields with descriptions run,
    "vovselect fieldname,fielddesc from <object>". Multiple fields may be
    requested by separating them with a comma. Some fields represent a data
    collection that can be broken down using a format of FIELD.X, such as:
       GRABBEDRESOURCES.<RESNAME>     (job objects)
       KEY.<KEYNAME>                  (metric objects)
       PARAM.<PARAMNAME>              (server object)
       PROP.<PROPNAME>                (all objects)
       RESOURCES.<RESNAME>            (tasker & job objects)
       SOLUTION.<RESNAME>             (job objects)

  SUPPORTED FILTERS:

    Use selection rule operators in conjunction with field names to filter
    queries. See operator list at URL/doc/html/vov/topics/vov/operators.htm
    via web browser. To get the current the URL for current instance,
    execute the vovbrowser command.

  EXAMPLES:

    % vovselect -h
    % vovselect objectname from objects
    % vovselect fieldname from server
    % vovselect id,name from users -order name -limit 10 -header
    % vovselect id,name from users -where 'name==joe'
    % vovselect id,name from 12345
    % vovselect id,name from subsets.23456
    % vovselect matchtype,host from matches.License:spice
    % vovselect idint,name from users -where 'idint>3600'
                -order 'idint desc'
    % vovselect id,age from System:running
    % vovselect id,age -from System:running -cache 0
```

Use vovselect for Querying

The `nc hosts` command can be used for querying, but it can sometimes take several minutes to return results, which causes some nodes to show up as "N/A". `nc hosts` will query the server and return significant amounts of data, but the server loading will directly affect the response time of the command.

In order to avoid such delay, you can use `vovselect` to run the query, as it prefilters the output server-side before returning it to the client.

Use the table below to understand the mapping of fields between the `nc hosts` and `vovselect` commands.

nc hosts	vovselect from TASKERS	vovselect from HOSTS
ARCH	ARCH	ARCH
CAPABILITIES	CAPABILITIES	NA
CAPACITY	CAPACITY	CPUS
CLASSRESOURCES	CLASSRESOURCES	NA
CLOCK	CLOCK	CPUCLOCK
COEFF	COEFF	NA
CONSUMABLES	CONSUMABLES	NA
CORES	CORESAVAIL	NA
CORESAVAIL	CORESAVAIL	NA
CORESTOTAL	CORESTOTAL	CPUS
CORESUSED	CORESUSED	NA
CPUS	CPUS	CPUS
CURLOAD	CURLOAD	NA
DOEXEC	DOEXEC	NA
DONETINFO	DONETINFO	NA
DOPROCINFO	DOPROCINFO	NA
DORTTRACING	DORTTRACING	NA
EFFLOAD	NA	NA
EXTRAS	EXTRAS	NA

nc hosts	vovselect from TASKERS	vovselect from HOSTS
FULLINFO	FULLINFO	NA
GROUP	GROUP	NA
HB	NA	NA
HBPP	NA	NA
HEARTBEAT	HEARTBEAT	NA
HOST	HOST	NAME
ID	ID	NA
IDINT	IDINT	NA
LASTJOBID	NA	NA
LASTUPDATE	LASTUPDATE	NA
LIFETIMEJOBS	LIFETIMEJOBS	NA
LOAD1	NA	NA
LOAD15	NA	NA
LOAD5	NA	NA
LOADEFF	NA	NA
MACHINE	MACHINE	MACHINE
MANUALPOWER	NA	NA
MAXLOAD	MAXLOAD	NA
MESSAGE	MESSAGE	NA
MESSAGESYS	MESSAGESYS	NA
MESSAGEUSER	MESSAGEUSER	NA
MODEL	MODEL	NA
NAME	NAME	NAME
NUMJOBS	NA	NA
OSCLASS	OSCLASS	NA

nc hosts	vovselect from TASKERS	vovselect from HOSTS
PERCENT	PERCENT	NA
PERSISTENT	PERSISTENT	NA
PID	PID	NA
POWER	POWER	NA
RAM	RAM	NA
RAMFREE	RAMFREE	NA
RAMTOTAL	RAMTOTAL	RAMTOTAL
RAWPOWER	NA	NA
RELEASE	RELEASE	NA
RESERVEDBY	RESERVEDBY	NA
RESERVEEND	RESERVEEND	NA
RESERVEFORBUCKETID	RESERVEFORBUCKETID	NA
RESERVEFORID	RESERVEFORID	NA
RESERVEGROUP	RESERVEGROUP	NA
RESERVEJOBCLASS	RESERVEJOBCLASS	NA
RESERVEJOBPROJ	RESERVEJOBPROJ	NA
RESERVEOSGROUP	RESERVEOSGROUP	NA
RESERVESTART	RESERVESTART	NA
RESERVEUSER	RESERVEUSER	NA
RESOURCECMD	RESOURCECMD	NA
RESOURCES	NA	NA
RESOURCESEXTRA	NA	NA
RESOURCESPEC	RESOURCESPEC	NA
RUNNINGJOBS	RUNNINGJOBS	NA
SLOTS	NA	NA

nc hosts	vovselect from TASKERS	vovselect from HOSTS
SLOTSTOTAL	SLOTSTOTAL	NA
STATSREJECTCORES	STATSREJECTCORES	NA
STATSREJECTOTHER	STATSREJECTOTHER	NA
STATSREJECTRAM	STATSREJECTRAM	NA
STATSREJECTRESERVED	STATSREJECTRESERVED	NA
STATSREJECTSLOTS	STATSREJECTSLOTS	NA
STATSVISITS	NA	NA
STATUS	NA	NA
SWAP	SWAP	NA
SWAPFREE	SWAPFREE	NA
SWAPTOTAL	SWAPTOTAL	NA
TASKERGROUP	TASKER	NA
TASKERNAME	TASKERNAME	NAME
TASKERSLOTSSUSPENDABLE	TASKERSLOTSSUSPENDABLE	NA
TASKERSLOTSSUSPENDED	TASKERSLOTSSUSPENDED	NA
TASKERSLOTSUSED	TASKERSLOTSUSED	NA
TASKERTYPE	TASKERTYPE	NA
TIMELEFT	TIMELEFT	NA
TMP	TMP	NA
TYPE	TYPE	NA
UPTIME	NA	NA
UPTIMEPP	UPTIMEPP	NA
USER	USER	NA
VERSION	VERSION	NA
VOVVERSION	VOVVERSION	NA

Upgrade Accelerator

This section provides instructions to change the software version on which Accelerator runs.

Often you will do this to switch all the components (vovserver, vovtasker, vovsh) to a newer version, but you may also switch the vovserver and vovtasker versions separately. For the discussion below, *current* refers to the version before the change, and *new* for the version after it.

You will find instructions for three common scenarios below.

Cold Upgrade	Used when you are also shutting down the Accelerator computers, as for making OS, hardware upgrades. All components will be on the new version.
Hot Upgrade	Used to replace vovserver while allowing jobs to continue to run on the current vovtasker version. vovserver will be on the new version, but vovtaskers stay on the current version.
Rolling Hot Upgrade	Used to allow the current software version to run jobs after the new version is installed and starts running jobs. This method is recommended when the normal workload consists of large-scale projects.

Note: The upgrade process includes downloading software archive files from the Altair website, which are then unpacked or untarred into a temporary installation directory. The installer will ask for a destination directory.

When choosing the destination for the downloaded software, be careful to select a non-destructive installation path. This ensures the current Altair Accelerator is not overwritten or otherwise damaged before the new software is installed and activated. Usually the new version will be installed as a sibling of the current version, such as `/tools/rtda/2013.09` for the current and `/tools/rtda/2016.09` for the new.

Cold Upgrade

This method is usually implemented with a physical overhaul, a part of a major IT event. This kills all running jobs, which can be highly disruptive.

To reduce the impact, Accelerator can be instructed to stop accepting and dispatching new jobs for a period prior to the shutdown event, enabling some of the running jobs to complete. How many jobs will complete depends on the jobs' duration and the time allowed before shutdown.

1. Download the Accelerator upgrade software.
2. Install the new software.
3. Using the new version, create a separate, temporary test queue (to validate the new version while production continues).
4. Validate the installation using the test queue that you created.
5. Schedule and announce the upgrade.

6. If you have multiple Accelerator queues, it is recommended that you set thNC_QUEUE to the name of the queue that is undergoing maintenance. This helps prevents accidentally shutting down the wrong queue. Use the command:

   ```
   setenv NC_QUEUE vncNameOfQueue
   ```

7. Optional: Suspend the vovtaskers with the command below. This command puts the vovtaskers in the SUSP state; running jobs will continue, but vovtasker will not accept new jobs. When vovtasker completes its current set of jobs, it will exit.

   ```
   nc cmd vovtaskermgr stop
   ```

8. Optional: At the point of the scheduled downtime, document the IDs of running jobs as those jobs will be terminated forcefully. This list can be used to inform users that their jobs were terminated by the maintenance event.

   ```
   nc list -r -a -O @ID@ @USER@ @COMMAND@
   ```

9. Optional: To automatically identify jobs when the queue is restarted, place the jobs in a special set.
   Example:

   ```
   nc cmd vovset create "ImpactedByQueueRestart" "isjob status==RETRACING"
   ```

10. Optional: Terminate these running jobs with the -force option, which should terminate the remaining taskers within a few minutes.
    Example:

    ```
    vovtaskermgr stop -force -all
    ```

11. Stop the vovserver of the queue with the command `ncmgr stop`
12. Proceed with any necessary infrastructure maintenance.
13. Restart the queue.
14. Ensure that your shell is configured to support the correct number of file descriptors.

    Note: This value cannot be changed after starting the queue.

15. Ensure that you are pointing to the appropriate version of Accelerator with the command `which nc`.
16. Start the queue with the command `ncmgr start`.
    A confirmation dialog will open.
17. Review the parameters carefully (especially number of file descriptors) before replying 'yes'. (Starting the queue will automatically start the taskers but this will take some time, be patient.)
18. Validate that jobs are dispatching normally.

    Note: Restarting a large compute farm will take several minutes.

19. Optional: Re-queue the jobs that were impacted by the shutdown. Use the following command:

    ```
    nc rerun -f -set ImpactedByQueueRestart
    ```

Hot Upgrade

This method reduces upgrade impact -- less downtime for productivity, less obtrusive for the users. The vovserver and vovtaskers are moved separately to the new version.

With this method, vovtaskers with running jobs are temporarily renamed, by appending `_stopped_<timestamp>` to their regular names. This allows active jobs to finish, and restarted vovtaskers to use the regular names.

> **Note:** If you are using a large value of VOV_RELIABLE_TIMEOUT you should suspend job dispatch (see Suspend Accelerator Job Dispatch) for some time, 30-60s, before initiating the cutover.

Follow the steps below for the preparation portion of the procedure:

1. Download the Accelerator upgrade software.
2. Install the upgrade software.
3. Create a separate, temporary test queue (to validate installation).
4. Validate the installation by starting the temporary test queue with a few vovtaskers on the new version and running test jobs.

Cutover process:

5. Notify your Accelerator users of the scheduled upgrade.
6. Get a shell as the Accelerator owner on the Accelerator vovserver host with current-version Altair Accelerator commands in the PATH.
7. Suspend job dispatching as in Suspend Accelerator Job Dispatch.
8. Stop the vovserver with the following command:

   ```
   ncmgr -q nc-queue stop -freeze
   ```

   > **Note:** The vovtaskers with jobs will continue to run and will be renamed. The ones that have no jobs will exit.

9. Ensure that the shell you are using has a sufficiently high limit for file descriptors.
10. Source the Altair Accelerator setup file for the new version, or use a separate shell set up for the new version.
11. Restart the vovserver with the new software version.

   ```
   ncmgr -queue nc-queue start
   ```

12. Optional: Restart a subset of the taskers. Use the following command:

   ```
   nc cmd vovtaskermgr stop tasker1...taskerN
   ```

13. After the vovtaskers have finished their jobs and exited, run the following command:

   ```
   nc cmd vovtaskermgr start tasker1...taskerN
   ```

Rolling Hot Upgrade

In this method, a new queue is brought up with the new version of the software; the previously existing queue remains fully functional.

This approach permits both administrators and users to thoroughly test the new version of the software and selectively move workloads to the new queue. The vovtasker machines are progressively moved from one queue to another by suspending them on the old queue, allowing jobs to *drain off* (complete) and then restarting the tasker on the new queue. The advantage of this approach is reducing risk: The old queue remains available during transition. Some disadvantages: The queue host:port and possibly queue name changes is not transparent to users; the entire upgrade process may take many days; you may need additional licenses during the transition interval.

1. Download the Accelerator upgrade software.
2. Install the new upgrade software.
3. Create a new queue name.
4. Start the queue (created in step 1) on either the existing host machine for Accelerator or a new host.
5. Configure the new queue with the various policy and job class settings.
6. Suspend a number of the taskers on the old queue and when they have terminated (when existing jobs have completed), add these taskers to the new queue.
7. Test and verify the software upgrade; transfer other remaining taskers after verifying new Accelerator.
8. After all taskers have been transferred, stop the old queue.

Stop Accelerator Job Acceptance

When preparing to stop Accelerator, you may wish to stop accepting new jobs so the queued jobs can drain before the shutdown.

You can do this by implementing the `vnc_policy.tcl` file shown below. Since `vnc_policy.tcl` is interpreted inline during the `nc run` command, this policy refuses new jobs with an informative message.

```
proc VncPolicyValidateResources { resList } {
    #
    # This policy prevents submission of new jobs with a message
    #
    VovWarning "Job not not submitted; NC closed by admin"
    exit
    return $resList; # not reached
}
```

Suspend Accelerator Job Dispatch

You can suspend job scheduling without stopping the Accelerator vovserver.

```
 % nc [-q qname] cmd vovsh -x 'vtk_server_config scheduler suspend'
```

This will suspend the dispatching of jobs.

To resume, use:

```
% nc cmd vovsh -x 'vtk_server_config scheduler resume'
```

(in fact, any word other than 'suspend' will resume the scheduler)

Run Multiple Versions of Altair Accelerator

Accelerator can be run using a different software version than the software version used by the other Altair Accelerator projects.

Allocator

Specify the version used by a queue using the -version option. For more information, refer to the Altair Allocator user documentation.

Indirect Taskers

For indirect taskers to work with a Accelerator server that uses a different version, set the environment variable VNC_NEWVERSION to the full path of the root of the installation used by Accelerator.

For example, if the Accelerator server is using the software installed in `/tools/RTDA/2019.01/macosx/bin/vovserver`, do the following:

```
setenv VNC_NEWVERSION /tools/RTDA/2019.01
```

Storage Aware Scheduling

This daemon `vovfilerd` replaces older daemons like `vovisilond`, `vovnetappd` and `vovregulatord`. Also the utility `vovfiler_setup` is no longer needed. The visualization of the filer information is now done either via the `vovfilerd.cgi` page or via the utility `vovfilerdgui`.

Storage Aware Grid Acceleration (SAGA) with vovfilerd

`vovfilerd` is used to make Accelerator aware of the status of the filers.

These filers can be of any type, but special treatment is done for Isilon filers (https://www.dellemc.com/en-us/storage/isilon/index.htm), NetApp filers, and ElastiFile filers.

For this daemon to work, you need:

- Access to the `curl` binary
- A user/password pair on each filer to be able to use the RestAPI (ask your Admin to help you with this)

Start and Configure the Daemon

```
% cd `vovserverdir -p vovfilerd`
% vi config.tcl
# This is an example for  vovfilerd/config.tcl
####################
### Global settings
####################
DefineFilers {
   AddFiler FS1 -probe 1 -testdir /net/fs120-ch/export/probes -period 1m
   AddFiler FS2 -probe 1 -testdir /remote/fs2/users/cadmgr    -period 20
   AddFiler IFS -probe 1 -testdir /mnt/isilon/dev         -host ifs1 -user admin -
passwd pw
}
```

The usage message for the procedure `AddFiler` is the following:

```
DESCRIPTION OF PROCEDURE:
   AddFiler <NICKNAME> <ARGS>...

        This procedure is only available in the config.tcl file for vovfilerd.

        This procedure has to be called from inside DefineFilers {}

        The first argument is the short nickname you want to use for the filer.
        It will be useful in invoking the GUI and it is reported in all  interfaces.
        The rest of the arguments are options:

        The filer can be in one of these states based on the measured value of
 latency
        and on the values of the 3 limits l1 l2 l3

        State   S1    Open-Loop         if  l <= l1
        State   S2    Steady            if  l > l1 && l <= l2
        State   S3    Feedback           if  l > l2 && l <= l3
        State   S4    Preempt            if  l > l3

   OPTIONS:
     -doc                 -- This message
     -t <TYPE>            -- Type of filer: Known types are :Isilon Netapp Elastifile
 Generic
                             Case insensitive.  Default 'Generic'
     -type <TYPE>         -- Same as -t
     -probe <BOOL>        -- Activate probe for filer using vovfsprobe.  Default 0.
     -testdir <DIR>       -- Test directory for the vovfsprobe.  This must be a
 writable directory
                             that sits on the filer being probed.
     -period <TIMESPEC> -- How often you want the probe to run (last arg in
 vovfsprobe)
                             Default 30s
     -preempt <BOOL>      -- Control whether preemption is activated when we reach
 state S4.
                             Default 0
     -min <N>             -- Minimum value for controlling resource. Default 20
     -max <N>             -- Maximum value for controlling resource. Default 1000
     -limits <LIST l1 l2 l3>  -- Define the boundaries of the state of the filer
 based on
                                   measured latency.  The limits are in milliseconds.
                                   The default values are  { 5.0 10.0 18.0 }

   ISILON OPTIONS:
     -host <HOSTNAME>   -- Used for Isilon to connect to correct host with wget
     -user <USERNAME>   -- Used for Isilon to connect to host with correct user name
     -passwd <PASSWD>   -- The password in Isilon to access the

   EXAMPLE:
        DefineFilers {
           AddFiler F1 -probe 1 -testdir /remote/filer1/test
        }
```

A simple way to start `vovfilerd` for testing is the following:

```
% cd `vovserverdir -p vovfilerd`
% vovfilerd >& vofilerd.log &
```

Then start the GUI with:

```
% vovfilerdgui -f NAME_OF_FILER &
```

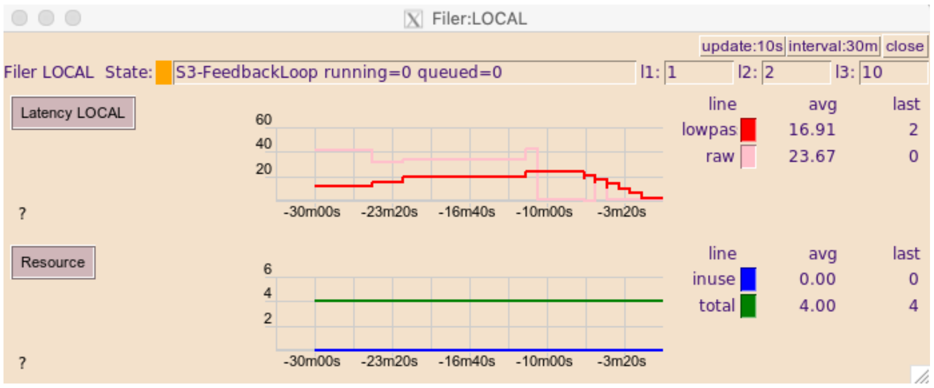


Figure 18:

In this dialog, you can see the status of the selected filer, and a graph showing the raw/filtered value of the measured latency (pink/red) and another graph showing the value of the controlling resource for that filer, i.e. the resource called Filer:NAME_OF_FILER.

vovfilerd Behavior

The `vovfilerd` daemon:

- Monitors several metrics on one or more filers, such as the average latency per operation; the main metric used to represent the load on the filer is the "Latency", which is the "average latency per operation" measured in milliseconds;
- Throttles the load on the filer by means of a resource map called "Filer:NAME_OF_FILER"
- Preempt jobs to lower the load if the load on the filer exceeds a threshold.

Both the Latency signal extracted from the filer and the value of the controlling resource computed by `vovfilerd` are low-pass filtered to smooth out their behavior.

The values of L1, L2, and L3 can be defined by the administrator using the option -limits in `AddFiler` and can be different for each filer. The limits represent latency values in milliseconds. They have to be at least 1.0 and each has to be larger than the previous.

State of Filer	Condition	Description
S1#OpenLoop	Latency # L1	The controlling resource is enough to accommodate the demand (running + queued); in this state of low latency, the resource is allowed to grow rapidly
S2#Steady	L1 < Latency # L2	The controlling resource is set to about 2% above the current running count, representing a slow growth of the number of running jobs
S3#FeedbackLoop	L2 < Latency # L3	The controlling resource is decreased based on the distance of latency from (L2+L3)/2. This state tries to maintain the latency in the middle between L2 and L3

State of Filer	Condition	Description
S4#Preemption	L3 < Latency	This state is the same as S3-FeebackLoop with the addition of preemption in a pro-active attempt to lower the latency on the driver. At every cycle, up to 5% of jobs impinging on the filer are preempted. This requires preemption to be activated for this filer with the option -preempt 1 in `AddFiler`

The selection of the jobs to be preempted is based on a number of metrics:

- Only jobs that declare the resource "Filer:NAME_OF_FILER" are considered
- The age, priority of the job: young jobs are more likely to be preempted, and low priority jobs are also preferred
- The latency experienced by the host on which the job is executing
- The directory in which the job was started, based on the load on the filer by path

All preempted jobs are suspended with SIGSTOP, their licenses are recovered, and then they are restarted when the load on the filer had decreased and the required licenses are again available.

Start vovfilerd

Once `vovfilerd` is configured, it can be started and stopped like any other daemon using `vovdaemonmgr`:

```
% vovdaemonmgr start vovfilerd
% vovdaemonmgr status vovfilerd
OK
% vovdaemonmgr stop vovfilerd
```

Usage: `vovfilerd - h`

```

vovfilerd: Usage Message

DESCRIPTION:
   A daemon to monitor filers and control Filer:XXX resources.

USAGE:
    % vovfilerd \[OPTIONS\]

OPTIONS:
    -h                -- Help usage message.
    -v                -- Increase verbosity.


EXAMPLES:
   % vovfilerd
   % vovfilerd  -v -v
```

Tuning vovfilerd

You may need to refine the tuning of the L1,L2,L3 limits that define the behavior of `vovfilerd` for each specific filer. We suggest a simple experiment that will help you see the effect of storage-aware-scheduling and also may help you decide on the tuning.

The experiment consists of running two I/O intensive workload (using dd) first with Storage Aware Scheduling and then without. The difference in behavior can be monitored in the browser interface. The suggested workload consists of only "write-to-disk" operations, so it is by definition an extreme workload. Real workload will behave better that this experiment.

First, bring up the GUI for the filer you want to test (in this example we assume the nickname for the filer is FS1), so you can keep an eye on the measured latency on the filer.

```
% vovfilerdgui -show
% vovfilerdgui -f FS1 &
```

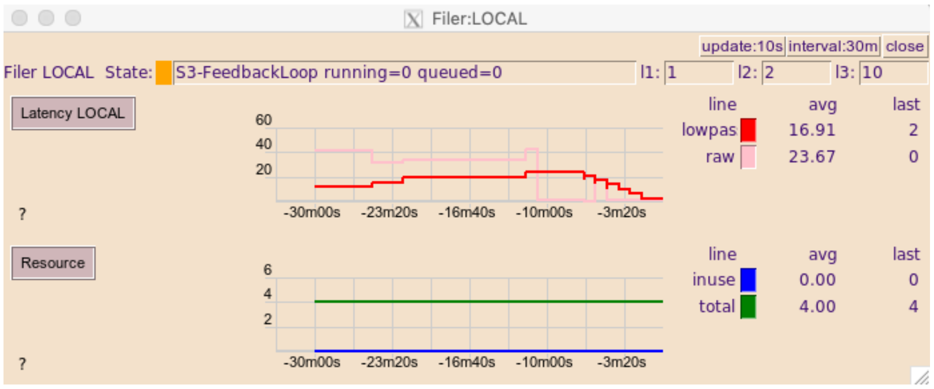


Figure 19:

Next run the workload **with** storage-aware-scheduling, i.e. using the resource that represents the filer, in this case Filer:FS1:

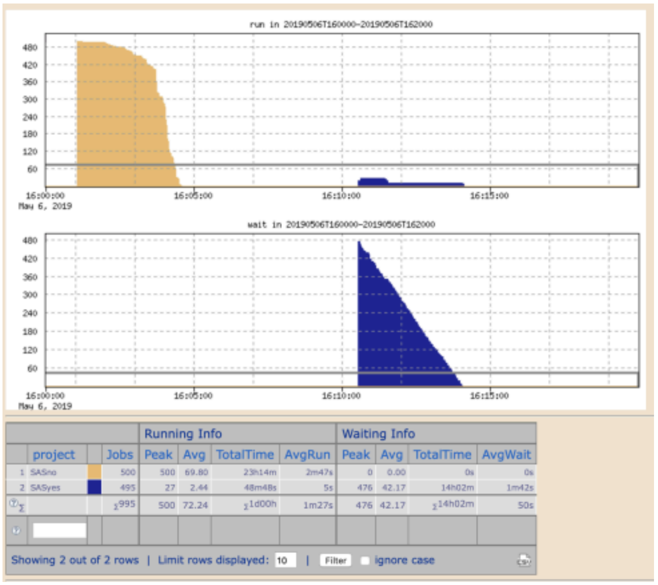
```
% cd some/directory/on/filer/FS1
% mkdir OUT
% setenv VOV_JOBPROJ SASyes
% time nc run -w -r Filer:FS1 -array 5000 dd of=OUT/dd_@INDEX@.out
if=/dev/zero count=1024 bs=102400
```

Next, run the same workload **without** the resource, so no restraint will be placed on it:

```
% cd some/directory/on/filer/FS1
% mkdir OUT
% setenv VOV_JOBPROJ SASno
% nc run               -array 5000 dd of=OUT/dd_@INDEX@.out if=/dev/zero
count=1024 bs=102400
```

If you have a big enough farm, say close to 1000 cores, you will see that this second workload uses all machines you have and creates even larger level of latency on the filer. The jobs, which should take about 1s each, may end up taking minutes to run because of such congestion. If the jobs also required a license, that license would be used by the job for way longer than it was earlier with storage aware scheduling.

To quantify the difference between the two modes of operation, lets navigate the browser interface (**Home** > **Workload** > **Job Plots**) to look at this report (you will have to define a precise time range for the report, something of the form "20190506T150000-20190506T170000", then report by project with no binning)



				Running Info				Waiting Info			
	project		Jobs	Peak	Avg	TotalTime	AvgRun	Peak	Avg	TotalTime	AvgWait
1	SASno	🟧	500	500	69.80	23h14m	2m47s	0	0.00	0s	0s
2	SASyes	🟦	495	27	2.44	48m48s	5s	476	42.17	14h02m	1m42s
Σ			Σ995	500	72.24	Σ1d00h	1m27s	476	42.17	Σ14h02m	50s

Showing 2 out of 2 rows | Limit rows displayed: 10 | Filter ignore case

Figure 20:

In this report you see the two workloads. (In this plot the SASno experiment comes before the SASyes experiment). Look at the average run time of the jobs. In SASyes experiment, the average duration is 5 seconds, while in the SASno experiment the average duration is 2m47s, i.e. 33 times longer!

Frequently Asked Questions and Troubleshooting Tips

I'm doing an installation and configuration in a Windows environment - can I use PowerShell?

PowerShell is not supported; we strongly recommend not using PowerShell.

How do I contact Altair Engineering to get additional support, report a bug, or request a feature?

You can contact Altair Engineering at: https://www.pbsworks.com/ContactSupport.aspx.

Why can't I access Monitor's historical license usage through Accelerator?

Accelerator ships with a version of Monitor that is licensed to monitor current license activity only. This edition is called LMS (Monitor **S**mall). To access Monitor's historical license usage information, you must have the full version of Monitor.

What do I do in the event of a server failover or crash?

You can find a checklist for system recovery on the System Recovery page. You can find this address with the command:

```
nc cmd vovbrowser -url /cgi/sysrecovery.cgi
```

Where is the policy.tcl file? What about the taskers.tcl file? The resources.tcl and security.tcl files?

All `.tcl` configuration files for Accelerator are located at `$VOVDIR/../../vnc/vnc.swd`

How do I enable the retrace of more than 400 jobs at a time?

The limit to how many jobs can be run/retraced at any given moment is defined by the *maxNormalClients* config variable. To change the variable, you can use the command:

```
vtk_server_config "maxNormalClients" maxnumberofjobs
```

How do I receive email notifications on job completion?

To receive automatic notification of major FlowTracer and Accelerator events, you should use the `vovnotifyd` daemon.

How do I track the memory usage of taskers?

VOV automatically keeps track of tasker memory usage. vovtasker keeps logs of 1 minute, 5 minute, and 10 minute load averages of the machines where taskers are running on. The tasker reports are available on the Tasker Load page. The Accelerator URL can be found with the command:

```
nc cmd vovbrowser -url /cgi/taskerload.cgi
```

Why are my jobs taking so long?

There are multiple reasons why FlowTracer jobs may be retracing slowly. Fortunately, Accelerator produces reports to help diagnose any problems. Read about available reports at *Resource Plots* in the Altair Accelerator User Guide..

How do I change to another version of Accelerator?

To upgrade Accelerator software, refer to Upgrade Accelerator.

How do I access information on license usage?

Accelerator does not have this functionality. This functionality belongs to Monitor. If you have Monitor installed and fully working, you can display license information at on the FTLM page.

Why are my licenses not fully utilized? I'm sure they're completely booked.

Your licenses are not fully utilized probably because they are not overbooked.

Essentially, the jobs being run do not use a license 100% of the time. Because there are jobs booked for licenses 100% of the time, there will be times where licenses are not utilized. This is because one or more jobs will still be running, but be done with the license that was booked. To rectify this, jobs are queued for more than 100% of the licenses, allowing another job to start and utilize the open license.

How do I share licenses between jobs in queue?

Read more on license at License Sharing Support.

My tasker is sick! What do I do?

Your tasker is sick because it has not sent out a heartbeat for at least 3 minutes. This may mean your tasker has crashed or disconnected. Once you have identified a sick tasker, you can proceed to troubleshoot it to fix the problem.

This list may be helpful:

- Check to make sure the machine itself is healthy. Make sure it is running, connected to the network, and not jammed.
- Check to see if vovtasker or vovtaskerroot is still running. If it isn't, then the tasker program itself has crashed. You should restart the tasker program with:

  ```
  vovtaskermgr start
  ```
- Check to see if vovtasker or vovtaskerroot is stuck. If it is, Linux commands such as `strace` and `pstack` should provide you with enough information to fix it.

My tasker is healthy, but all jobs sent to the tasker come out failed. What is going on?

Your tasker is what is called a black hole. It appears healthy, but is in fact unable to execute jobs. There is functionality to enable automatic detection of black holes in the Black Hole Detection page.

When a black hole is found, it would be prudent to send a simple job such as `cp` or `sleep` to the tasker to confirm its black hole state.

I want to give a different amounts of resources to different sites. How can I do that?

FairShare is a mechanism to allocate CPU cycles among groups and user according to a policy. This would be your best bet.

How do I limit a resource for a particular user?

Although it is not recommended, information on limiting users can be found on the Limit Users page.

My job was killed because it failed to start within 1m00s!

This can be caused by a bad NFS mount point, or an automounter that is so overloaded, that it fails to mount the run directory for the job in under a minute. Although this is a hardware problem, there is a workaround by changing the variable VOV_MAX_WAIT_NO_START to a value over 1 minute.

How do I setup prioritized licence usage?

For example, to use the licence FOO_BAR_A first, then the licence FOO_BAR_B second, use:

```
vtk_resourcemap_set FB-lic UNLIMITED "Licence:FOO_BAR_A  OR Licence:FOO_BAR_B"
```

In the jobclass. To set it in a resource map, use:

```
set VOV_JOB_DESC(resources) "Licence:FOO_BAR_A  OR Licence:FOO_BAR_B"
```

Why am I missing the plots when I look at a resource or license report?

Probably, what is causing the plots to be missing is a name resolution issue. To fix this, make sure VOV_HOST_HTTP_NAME is set correctly. If all else fails, set this to the host's IP address, not network name. To update a running server, you must use the command:

```
vtk_server_setenv VOV_HOST_HTTP_NAME XXX
```

vovresourced is not starting, says 'Failed to source'' too many resources'!

Most likely, you have exceeded the limit for resource maps in use. To raise this limit, change the maxResMap value in `policy.tcl`.

How do I ensure that a tool is preemptable robustly?

Sometimes a tool will crash when preempted. To test whether this is Altair, or the tool vendor, try and run the tool without Altair binaries (pure UNIX code) and see if the tool still crashes. The steps to do this are as follows:

1. Start the EDA tool(s) which you wish to test.
2. Use the UNIX command `ps` to find the PID of the EDA tool(s):

```
% ps | grep firefox
PID TTY          TIME CMD
349 ?        00:24:19 firefox
```

3. Send TSTP and CONT signals 10 seconds apart repeatedly. Try this in your shell:

```
% kill -TSTP 349 ; sleep 10 ; kill -CONT 349 ; sleep 10 ; kill -TSTP 349 ;
 (etc...)
```

Following these steps, if the tool crashes, then the problem is independent of Altair, as not a single line of Altair code was executed.

I set a configuration in the policy.tcl file, but it is not taking effect!

Most likely, the file has not been read yet. Try a:

```
% nc cmd vovproject sanity
```

I have a lot of log files, how can I remove the older files?

An easy way to remove files that are over 60 days old is using the `vovcleanup` command:

```
% nc cmd vovcleanup -proj
```

Note: When preemption is heavily used, log files tend to build up.

How do I test a policy change before releasing it to production?

To test policy changes you can use the soft release mechanism. Here's a summary:

- Create a test queue.
- Set up the test to use files from the repository of the master queue.
- Test a hot file in a sandbox, identify and fix the errors before releasing it to the production domain.

I am upgrading the software - how do I suspend Accelerator from dispatching jobs?

Typically, to minimize the impact of upgrading the overhauling the system, the vovserver is stopped from dispatching new jobs, while jobs that are running are allowed to complete on the vovtasker. There is more than one way to do this: Cold Upgrade, Hot Upgrade and Rolling Hot Upgrade. For more information and instructions, refer to Upgrade Accelerator.

HPC Advice

This section provides recommendations to obtain the maximum performance from your Accelerator. As Accelerator is a fast system, fine-tuning performance may only be needed when running several hundreds of thousands of jobs daily.

Use the Latest Altair Accelerator Release

The performance of the Accelerator scheduler is frequently updated. Using the most current version is recommended.

Use the `vwn` Wrapper

The wrapper `vwn` (alias for `vw -d`) is a faster wrapped because it avoids communication with vovserver. The regular `vw` checks the timestamp of the outputs after the job is done, whereas `vwn` does not. An example is shown below:

```
% nc run -wrapper vwn -array 100 sleep 0
```

To further push performance of the scheduler, you may want to use two options:

- -nolog: this disables the creation of the log file
- -nodb: this disables the logging of the job execution used for adding job info to the database

```
% nc run -wrapper vwn -nodb -nolog -array 100 sleep 0
```

- The benefit of using `vwn` is speed.
- The disadvantage is that jobs that require the -wl option cannot be run. However, this disadvantage may be not be significant, as -wl adds a relatively high load for what it does: -wl requires an extra *notify client* to handle the event generated when the job terminates.

Reduce the FairShare Window

When running millions of jobs per day, it is not important to keep a long FairShare history. Typically, a window of 2 to 5 minutes tracks sufficient history. An example follows:

```
% nc cmd vovfsgroup modrec /some/fs/tree  window 2m
```

Reduce the autoForget Times

By forgetting jobs more quickly, the memory image of vovserver is kept smaller. An example is shown below:

```
# In policy.tcl
set config(autoForgetValid)  3m
set config(autoForgetFailed) 1h
set config(autoForgetOthers) 1h
```

Disable Wait Reasons

If analyzing what causes *wait time* in the workload, the wait reason analysis can be disabled as shown below:

```
# In policy.tcl
set config(enableWaitReasons) 0
```

Wait time analysis can then be re-enable as needed as shown below:

```
% nc cmd vovsh -x 'vtk_server_config enableWaitReasons 1'

### collect some data for a few minutes, then

% nc cmd vovsh -x 'vtk_server_config enableWaitReasons 0'
```

Disable File Access

Disabling file access is mostly a high-reliability option. By disabling file access, the vovserver never looks at any of the files in the user workspaces, which avoids the risk of disk slowness or disk unavailability. An example is shown below:

```
% nc cmd vovsh -x 'vtk_server_config disablefileaccess 2'
```

Reduce Update Rate of Notify Clients

Notify clients, clients that are tapping the event stream from vovserver (such as `nc gui`, `voveventmon` or `nc run -wl`), are updated immediately in the inner loop of the scheduler. If the environment includes hundreds of such clients, it may be beneficial to slow down the update rate by setting the parameter *`notifySkip`*. The default value is 0: no skip. Typically, the more events that take place, the more events that can be skipped without notice. For example, if several events are taking place, setting `notifySkip` to 100, fewer updates may not be noticed. If the number of events is small, a one-second delay may be noticed in some updates of the GUI. skipped without notice.

Note: Regardless of the setting, the maximum time between updates is one second.

```
# In policy.tcl
set config(notifySkip) 100
```

NVIDIA™ GPUs Support in Accelerator

If you have machines with multiple GPUs, you can harness the power of those devices by following these guidelines, which have been tested with up to 8 GPUs per machine.

1. Start a "vovtaskerroot" in each machine with one consumable hardware resource for each of the GPUs on that machine. Call these resources `GPU:Tesla<N>` where `N` is an index starting from 0.

```
# In taskers.tcl
set res4gpus "GPU:Tesla0/1 GPU:Tesla1/1 GPU:Tesla2/1 GPU:Tesla3/1"
vtk_tasker_define nv001 -resources $res4gpus
vtk_tasker_define nv002 -resources $res4gpus
# ...
vtk_tasker_define nvXXX -resources $res4gpus
```

2. Define job resources of type `G:Tesla<J>` where `J` is the number of GPUs that are requested by the job. For each `J` you need to define the maps from the job resource to the HW resources of type `GPU:Tesla`*`N`*.
For example, `G:Tesla1` is easy and it needs to map to the OR of any of the available GPUs, while `G:Tesla4 is also` easy because it needs to map to the AND of all 4 devices.

```
# In resources.tcl
set mapOR  "GPU:Tesla0/1 OR GPU:Tesla1/1 OR GPU:Tesla2/1 OR GPU:Tesla3/1"
set mapAND "GPU:Tesla0/1    GPU:Tesla1/1    GPU:Tesla2/1    GPU:Tesla3/1"
vtk_resourcemap_set G:Tesla1 -max unlimited -map $mapOR
# ...
vtk_resourcemap_set G:Tesla4 -max unlimited -map $mapAND
```

The other maps for `G:Tesla2` and `G:Tesla3` are more complex, and for larger values of `N` it is not feasible to use all possible combinations of devices. To help compute those maps, and to reduce the number of combinations to a workable subset, we provide a procedure called `findCombinations` in `$VOVDIR/scripts/hero/nvidia/hero_nvidia_resources.tcl`. Feel free to copy that file into your `resources.tcl` file.

```
proc findCombinations { list n } {
    #
    # Recursive procedure to find all combinations of 'n' elements from 'list'.
    #
    set result {}
    set l [llength $list]
    if { $l >= $n } {
        set inc 1
        for { set i 0 } { $i < $l } { incr i $inc } {
            set elem      [lindex $list $i]
            set subList   [lreplace $list 0 $i]
            set subCombos {}
            if { $n > 1 } {
                set subCombos [findCombinations $subList [expr $n-1]]
                if { $l > 2 && [llength $subCombos] > 1 } {
                    ### When the combinations are too-many, use only the first
 combo.
                    set subCombos [lrange $subCombos 0 0]
                }
                foreach subCombo $subCombos {
                    lappend result [concat $elem $subCombo]
                }
            } else {
                lappend result $elem
            }
        }
    }
    return $result
}

## Maximum number of GPUS in any of the farm machines.
## Call the GPUS  "GPU:Tesla0 ... GPU:Tesla3 ..."
set MAX  8
set GPUS {}
for { set i 0 } { $i < $MAX } { incr i } {
    lappend GPUS "GPU:Tesla$i/1"
}

# A resource to count how many GPUs are in use.
vtk_resourcemap_set G:TeslaNum -total unlimited

for { set i 1 } { $i <= $MAX } { incr i } {
    set options [findCombinations $GPUS $i ]
    set optionsWithParentheses ""
    set sep ""
    foreach opt $options {
        if { [llength $opt] > 1 } {
            append optionsWithParentheses "$sep ( $opt )"
        } else {
            append optionsWithParentheses "$sep $opt"
        }
        set sep " OR"
    }
    vtk_resourcemap_set G:Tesla$i -total unlimited -map "G:TeslaNum#$i
 ( $optionsWithParentheses )"
}
```

3. Submit your workload using the wrapper `vovgpu` which is a script that interprets the "SOLUTION" computed by the scheduler and passes the selected list of devices to the application via the environment variable VOV_GPUSET or with the macro @GPUSET@. Note that VOV_GPUSET gives you a space-separated list of devices, while @GPUSET@ gives you a comma-separated list.

With this setup, you can submit arbitrary workloads which request any number of GPUs.

```
% nc run -r G:Tesla1 -- vovgpu some_job_requiring_gpu -devices=@GPUSET@ -arg1 -arg2
% nc run -r G:Tesla2 -- vovgpu some_job_requiring_gpu -devices=@GPUSET@ -arg1 -arg2
% nc run -r G:Tesla3 -- vovgpu some_job_requiring_gpu -devices=@GPUSET@ -arg1 -arg2
% nc run -r G:Tesla4 -- vovgpu some_job_requiring_gpu -devices=@GPUSET@ -arg1 -arg2
```

If you are new to Accelerator, it is worth remembering that you can get project tracking if you use the -jobproj option. For FairShare, use the -g and -sg options in `nc run`, as in this example:

```
% nc run -r G:Tesla4  -jobproj MachineLearnAboutCats  -g /bu/ai/rd   vovgpu
 my_ml_app
```

Simulation Scripts

This section describes job simulation scripts that emulate jobs. Such scripts are often used by developers as well as business systems analysts.

Typically, these scripts perform no real functions and do not access licenses; they emulate the appearance of actual usage. These scripts are often used to debug a system or configuration issue, test the capacity of the system, checking if the resources are available for upcoming jobs, and setting benchmarks for dispatching jobs, such as 1000 calls of `nc run /bin/date`.

Why Developers Use Simulation Job Scripts

Sometimes developers need to test a flow under realistic conditions to ensure that all settings are correct that users have access to resources, permissions and quotas, to run the jobs that they intend to run.

- Developers may not have access to the tools; they need to create simulated jobs for realistic testing in an artificial environment.
- Developers may have access, but in an earlier stage of development, it may be preferable to create placeholder tools, thus avoiding the use and cost of licenses.

Using Job Script Simulations for Troubleshooting and Planning

Running tests with simulated jobs can help identify hardware bottlenecks or other system limitations. Using test scripts with proportional values help generate profiles very quickly, such as usage over time. Such scripts can be used with scaled memory/time requirements, such as 1 Megabyte of memory of a test script represents 1 Gigabyte represents 1 Megabtyte of actual usage, or 1 minute of a test script represents 1 hour of actual usage.

For more basic flows in which each stage consists of similar types of jobs, test scripts may not be needed. However, for more complex flows with jobs that have different characteristics and dependencies, estimating the longest path, how often job requirements result in conflict and so on, are difficult to estimate without running tests that provide results to analyze.

```
% sleep   x
% cp  aa bb
% vovmemtime
```

Guidelines for Simulation Job Scripts

Frequently Used Code

- `array`
- `cp` *`file1file2`*: Emulates I/O data transfer.

  Note: To successfully view a data transfer on a job profile, very large files must be used; transactions and other usages must continue at least one minute to be visible.

- `sleep` *`x`*: Do nothing during the specified time ***x***. For tests and evaluations, it is best to include a random number generator. Used alone, sleep jobs complete at known, precise times - based on the specified timing, several jobs could complete simultaneously, which does not occur in actual job runs. For information about job profiles, refer to *Job Profiling* in the Altair Accelerator User Guide..
- `vovmetime`: Allocates memory, also uses CPU.
- `vtool`: Used for calling licenses. `vtool` can be used to emulate calling licenses. For information, refer to *Wrap Unlicensed Tools* in the Altair Monitor Administrator guide.

Simple examples of scripts

Note: Using the sleep command alone may cause unrealistic behaviors, such as all jobs completing at the same time. Due to the scheduling of jobs and the availability of resources exact timing is unlikely. For more realistic behaviour, including a random variation of timing is recommended.

bash version

```
% #!/bin/bash
% dur=$[ ( $RANDOM % $1 / 5 ) + $1 ]s
% echo "Sleeping for $dur"
$% sleep $dur
```

Sanity Check for vovserver

The command `sanity` is used to perform checks on the consistency of the trace and of other internal data structures.

Use sanity check when the server appears confused about the status of the trace.

```
% vovproject sanity
```

Use the `reread` command to re-read the server configuration. The files read are `policy.tcl`, `security.tcl`, `equiv.tcl`, `setup.tcl`, and `exclude.tcl`.

You need not use `reread` after changes to `taskers.tcl`, it is not a vovserver config file. It is used by `vovtaskermgr`.

```
% vovproject reread
```

`sanity` does a wide variety of checks, cleanups, and rebuilds of internal data structures. Check the vovserver log file for messages that include `sanity`. Here are some of the main things that it does:

- Clears all alerts
- Flushes journal and crash recovery files
- Clears IP/Host caches
- Stops and restarts resource daemon (`vovresourced`)
- Checks and cleans internal object attachments
- Verifies all places and jobs have sensible status
- Resets user statistics and average service time
- Checks the contents of system sets like System:jobs
- Removes older jobs from recent jobs set
- Makes sure all jobs in the running jobs set are actually running
- Verifies all sets have the correct size
- Clears the barrier-invalid flag on all nodes and recomputes it
- Clears empty retrace sets
- Checks preemption rules

- Checks all tasker machines, marking them sick if they are not responding
- Checks for rebooted tasker machines and terminates jobs attached to them
- Checks filesystems on tasker machines and verifies mount points
- Clears resource list caches from jobs
- Clears and rebuilds job class sets
- Creates limit resources for ones that are missing
- Verifies grabbed resources (non running jobs should not have any)
- Makes sure only running jobs have stolen resources
- Reserves resources for all running jobs
- Create any missing resource maps for groups and priorities
- For each job with I/O, makes sure outputs are newer than inputs
- Makes sure any file with running status has an input job with running status
- Verifies the status of all nodes
- Checks for stuck primary inputs (primary inputs should only be VALID or MISSING)
- If a file is invalid or missing, but the input job is VALID, turn the job INVALID
- Finds running jobs without tasker and changes the status to SLEEPING
- Makes sure all input files of a VALID job are also VALID
- Makes sure all output files of a job have the same status as the job
- Recomputes waitreason counts
- Checks job queue buckets
- Verifies link between job queue buckets and resource maps
- Makes sure all queued jobs have job queue buckets
- Checks FairShare groups
- Checks for a license

Disable Regular User Login

This section provides guidelines to disable the ability for those with the USER level of privilege to log onto selected tasker machines. Most often, for better throughput, this is applied to use selected machines as part of a computing resource pool exclusively through Accelerator.

Note: Disabling the login to a vovtasker is not a normal or supported use of VOV functionality.

Disabling user login to the selected tasker machines is done in two phases:

1. Disable user logins
2. Set up vovtsd

Phase 1

1. Disable all user logins except for the superuser or root on the selected machine.
2. Create the file `/etc/nologin`. The content of the file will be the message the users receive when they try to login.

An example of `/etc/nologin` on host `h01` is shown below:

```
% cat /etc/nologin
Login disabled. Contact admin for help.
```

When a regular (non-root) user tries to login, the message shown to the user will be as follows:

```
% rlogin -l john h01
john's Password:
Login disabled. Contact admin for help.

login:
```

Phase 2: Set Up vovtsd When login is disabled for the USER level and VOV ADMIN is typically not a root user, it is not possible to use rsh or ssh to either start or stop taskers from remote machines, such using the command `vovtaskermgr`. In this scenario, `vovtsd` can be used to manage the tasker machines remotely.

3. Log onto the machine as `root`, switch to `VOV ADMIN` and then start `vovtsd`.
4. From a remote machine, start or stop a tasker on this machine using a previously used method, such as using the command `vovtaskermgr` or `ncmgr reset`, the GUI or a browser.

```
% su - vncadmin
% vovtsd -normal
```

5. Step 4 assumes that the shell for user "vncadmin" is set up to run the VOV software. Some accounts do not have this setup. In that case, the `vovboot` script could be used to start `vovtsd`.

```
% su - root
% /full/path/to/vov/installation/common/scripts/vovboot vovtsd -normal
```

To start `vovtsd` at boot time, consider deployment of the `S99vovtsd` script, a copy of which can be found in the directory `$VOVDIR/etc/boot/S99vovtsd`. Copy this script into `/etc/rc3.d/S99vovtsd` and customize it to fit the installation requirements.

- Ensure that `vovtsd` is running.
- Automatically restart `vovtsd` on reboot of the machine. This enables the machine to provide continuous computing power without having to log in as root and manually start `vovtsd`.

Note: If `vovtsd` is already running on a host and starting another host is tried, that second host will not start because the port is already occupied; starting `vovtsd` on a regular basis is a good way to ensure it is always running. Keeping the host running can be done with a cron job as shown in the example below:

```
% su - vncadmin% crontab -e
# Start vovtsd every thirty minutes
1,31 * * * *  /full/path/to/VOV/common/scripts/vovboot vovtsd -normal > /
dev/null 2>&1
```

Auxiliary Group Membership

Theory

If VOV_USE_INITGROUPS is set, the subtasker calls `initgroups()`. This is an OS call that sets all (or max 16) auxiliary groups. The resulting list of groups is not cached. Another job will call `initgroups()` again.

The default is to not call `initgroups` because it may load the name services too much.

By default, the vovtasker calls the external utility `vovgetgroups`, which uses the value of VOV_ALARM to decide how long to wait for a reply (default 10 seconds). The VOV_USE_VOVGETGROUPS environment variable can be used to control this behavior:

Set to 0 to disable the call to the external utility and use the `getgrent()` POSIX API function to find all groups that are valid for a user. If there are more than 16, the list is truncated to the first 16. The list is cached by vovtasker, so only the first job for a user causes traffic with the name services. This is only recommended in small environments, as this method can create significant delays, and even blocking conditions, in complex environments (e.g. Linux with LDAP).

Set to 2 to continue to use the external utility, but instruct the utility to call the `getgrent()` POSIX API function instead of the default call to `getgrouplist()`. This is mainly for debugging purposes, since this mode of operation results in slower processing of group information.

History

Prior to 2016.09 & 2015.09u8

If VOV_USE_VOVGETGROUPS was set to any value, when a tasker needs to get group data it will use the `vovgetgroups` external utility (a separate executable). This utility is robust to LDAP errors or timeouts which would otherwise cause the `getgrent` library call to hang indefinitely (and block the tasker from issuing further jobs).
Prior to customers switching to Centos6.x and SSSD name service, the use of VOV_USE_VOVGETGROUPS was recommended. After the switch to Centos6.x/SSSD, a bug was found that prevented all groups from being fetched. Switching to VOV_USE_INITGROUPS=1 and leaving VOV_USE_VOVGETGROUPS unset appeared to fix the problem, but at the probable cost of reduced performance and increased name service load.

2016.09 & 2015.09u8 and Later Versions

If VOV_USE_VOVGETGROUPS was set to any value other than 1, it would behave like pre 2016.09 code and use `getgrent()`. If VOV_USE_VOVGETGROUPS was set to 1, it would use `getgrouplist()`, which is a newer utility (but still old) to get group information with higher performance.
The downside to setting VOV_USE_VOVGETGROUPS=1 in 2016.09 is that there may be some off-beat OS's that don't support it. However, it seems to be faster, work with SSSD, and doesn't load the name service as much.
The recommendation based on the review of the history and the code is the following:

- Use VOV_USE_VOVGETROUPS=1 and leave VOV_USE_INITGROUPS unset if you are on <2015.09u8 earlier and not using CentOS6.6 with SSSD (uses non blocking `getgrent`)
- Leave VOV_USE_VOVGETGROUPS unset and set VOV_USE_INITGROUPS=1 if you are on < 2015.09u8 and want to use CentOS6.6/SSSD (uses an extra group init & `getgrent`)
- Set VOV_USE_VOVGETGROUPS=1 and leave VOV_USE_INITGROUPS unset if you are on 2016.09 or >2015.09u7 and running a common OS (non blocking, `getgrouplist`)
- Set VOV_USE_VOVGETGROUPS=1 and leave VOV_USE_INITGROUPS unset if you are on 2016.09 or >2015.09u7 and running an uncommon OS (non blocking, `getgrent`).

If both are set then VOV_USE_VOVGETGROUPS dominates.

Troubleshooting

Answers to common questions when using Monitor.

The Server Doesn't Start

1. Make sure you have a valid RLM license. Type:

   ```
   % rlmstat -a
   ```

2. Check if the server for your project is already running on the same machine. Do not start an Accelerator project server more than once.

   ```
   % vovproject enable project% vsi
   ```

3. Check if the server is trying to use a port number that is already used by another vovserver or even by another application. VOV computes the port number in the range [6200,6455] by hashing the project name. If necessary, select another project name, or change host, or use the variable VOV_PORT_NUMBER to specify an known unused port number. The best place to set this variable is in the `setup.tcl` file for the project.
4. Check if the server is trying to use an inactive port number that cannot be bound. This can happen when an application, perhaps the server itself, terminates without closing all its sockets.

   The server will exit with a message similar to the following:

   ```
   ...more output from vovserver...
   vs52 Nov 02 17:34:55         0                3      /home/john/vov
   vs52 Nov 02 17:34:55 Adding licadm@venus to notification manager
   vs52 Nov 02 17:34:55 Socket address 6437 (net=6437)
   vs52 ERROR Nov 02 17:34:55 Binding TCP socket: retrying 3
   vs52 Nov 02 17:34:55 Forcing reuse...
   vs52 ERROR Nov 02 17:34:58 Binding TCP socket: retrying 2
   vs52 Nov 02 17:34:58 Forcing reuse...
   vs52 ERROR Nov 02 17:35:01 Binding TCP socket: retrying 1
   vs52 Nov 02 17:35:01 Forcing reuse...
   vs52 ERROR Nov 02 17:35:04 Binding TCP socket: retrying 0
   vs52 Nov 02 17:35:04 Forcing reuse...
   vs52 ERROR Nov 02 17:35:04
   PROBLEM:  The TCP/IP port with address   6437   is already being used.

   POSSIBLE EXPLANATION:
           - A VOV server is already running   (please check)
           - The old server is dead but some
             of its old clients are still alive  (common)
           - Another application is using the
             address  (unlikely)

   ACTION: Do you want to force the reuse of the address?
   ```

a) If this happens, list all VOV processes that may be running on the server host and that may still be using the port. For example, you can use:

```
% /usr/ucb/ps auxww | grep vov
john   3732  0.2  1.5 2340 1876 pts/13   S 17:36:18  0:00 vovproxy -p
 acprose -f - -b
john   3727  0.1  2.2 4816 2752 pts/13   S 17:36:16  0:01 vovsh -t /rtda/
VOV/5.4.7/sun5/tcl/vtcl/vovresourced.tcl -p acprose
...
```

b) Wait for the process to die on its own, or you can kill it, for example with `vovkill`.

```
% vovkill pid
```

c) Restart the server.

5. You run the server as the Accelerator administrator user. Please check the ownership of the file `security.tcl` in the server configuration directory `vnc.swd`.

UNIX Taskers Don't Start

Accelerator normally relies on remote shell execution to start the taskers, using either `rsh` or `ssh`.

- If using `rsh` try the following:

  ```
  % rsh host vovarch
  ```

  where `host` is the name of a machine on which there are problems starting a tasker.

  This command should return a platform dependent string and nothing else. Otherwise, there are problems with either with the remote execution permission or the shell start-up script.
- If the error message is similar to "Permission denied", check the file `.rhosts` in your home directory. The file should contain a list of host names from which remote execution is allowed. See the manual pages for `rsh` and `rhosts` for details. You may have to work with your system administrators to find out if your network configuration allows remote execution.
- If using `ssh`, perform the test above but use `ssh` instead of `rsh`. For more details about `ssh` see SSH Setup in the *VOV Subsystem Administrator Guide*.
- If you get extraneous output from the above command, the problem is probably in your shell start-up script. If you are a C-shell user, check your `~/.cshrc` file. The following are guidelines for a remote-execution-friendly `.cshrc` file:

  # Echo messages only if the calling shell is interactive. You can test if a shell is interactive by checking the existence of the variable prompt, which is defined for interactive shells. Example:

  ```
  # Fragment of .cshrc file.
  if ( $?prompt ) then
  echo "I am interactive"
  endif
  ```

  # Many `.cshrc` scripts exit early if they detect a non interactive shell. It is possible that the scripts exit before sourcing `~/.vovrc`, which causes Accelerator to not be available in non-interactive shells. Compare the following fragments of `.cshrc` files and make sure the code in your file works properly:

The following example will not work properly for non-interactive shells:

```
if ( $?prompt ) exit
source ~/.vovrc
```

This example is correct, source `.vovrc` and then check the prompt variable:

```
source ~/.vovrc
if ( $?prompt ) exit
```

This example is also correct:

```
if ( $?prompt ) then
# Define shell aliases
...
endif
source ~/.vovrc
```

# Do not apply `exec` to a sub shell. This will cause the `rsh` command to hang.

```
# Do not do this in a .cshrc file
exec tcsh
```

License Violation

Accelerator is licensed by restricting the number of taskers. This is the number of all unique hosts that run taskers in all instances of Accelerator servers that use the same license.

You can find out the capacity of your license with the following command:

```
% rlmstat -avail
```

The file `$VOVDIR/../../vnc/vnc.swd/taskers.tcl` defines the list of hosts that are managed by the server. Make sure the number of tasker hosts is within the license capability.

Crash Recovery

In the event of a crash or failover, you can find a checklist of what to do on the System Recovery page.

This address can found using the command:

```
nc cmd vovbrowser -url /cgi/sysrecovery.cgi
```

Backwards Compatibility and Migrating from Previous Versions

This chapter provides information about deprecated features that are still supported. This information is provided should you see older commands in use after a software upgrade or migrating to a newer Accelerator.

As these older features, commands and options may soon become unsupported, using the current commands and options instead is strongly recommended.

Tasker Resources

The resources offered by a tasker are represented by a space-separated list of tokens. Resources can be **explicit** or **symbolic**.

The resources described in this section have been obsoleted. The new information about Hardware Resources in provided in Hardware Resources. The resources described in this section are still supported.

The resource list is the concatenation of two sublists:

- The resources specified with option `-r`, which can be either a static list (explained below) or dynamically computed list as explained in Time-Variant Taskers.
- The resources specified in the `taskerClass.table` file.

The option `-r` in vovtasker defines the resources offered by the tasker.

Examples of explicit resources:

```
% vovtasker -r "unix diskio RAM/512"
% vovtasker -r "RAM/3000 REGRMACHINE"
```

Symbolic Resources for Taskers

The following table describes all symbolic resources defined for taskers. By default, a tasker provides the resources corresponding to the symbolic resource `@STD@`.

Symbolic Resource	Description
@ARCHITECTURE@	It has value 'win64' on Windows, and the same as @MACHINE@ on UNIX.
@CPUS@	A consumable resources indicating the number of CPUS in the tasker (example for a 4-CPU machine: "CPUS/4").
@DISPLAY@	The name of the X display accessible by the tasker and derived from the value of the environment variable DISPLAY.
@HOST@	The name of the host on which the tasker runs.
@MACHINE@	The name that identifies the machine type . On Windows, this have value 'x86'.

Symbolic Resource	Description
@MODEL@	Linux: an abbreviated version of the "model name" line in the output of `/proc/cpuinfo`, otherwise the value is "Model:unknown". Not available on the other platforms.
@OS.VERSION@	The name of the operating system, including the version number (for example, Linux.1.0).
@OS@	The operating system name (for example, Linux).
@OSCLASS@	Either "unix" or "windows".

@RAMFREE@	The amount of available physical memory (RAM), in MB. On a Linux system, this is computed by opening `/proc/meminfo` and by adding up the values for MemFree, Buffers, and Cached.
	`% cat /proc/meminfo`
	`MemTotal:      6100392 kB`
	`MemFree:       1848576 kB`
	`Buffers:        188596 kB`
	`Cached:        2686368 kB`
	`SwapCached:         16 kB`
	`Active:        1847404 kB`
	`...`

@RAMTOTAL@	The total amount of physical memory (RAM), in MB.
@RAM@	This is a consumable resource representing the residual amount of RAM after taking into account the jobs running on the tasker.
@RELEASE@	This is mostly used on Linux to represent the name of the Linux distribution. On most systems, this is computed by running `lsb_release -isr`.
@SMARTSUSPEND@	Either the string "smartsuspend" if SmartSuspend (by Jaryba) is available on the machine or the null string "".
@STD@	The default for each tasker, which corresponds to the set @CPUS@ @DISPLAY@ @USER@ @HOST@ @RAM@ @MACHINE@ @VOVARCH@ @OSCLASS@ @OS.VERSION@ @VIEW@ @RELEASE@.
@SWAPFREE@	The amount of available swap space, in MB.
@SWAPTOTAL@	The total amount of swap space, in MB.

Symbolic Resource	Description
@SWAP@	This is a consumable resource representing the residual amount of SWAP after taking into account the jobs running on the tasker.
@USER@	The name of the user that owns the tasker. This is the value of either the variable LOGNAME or of the variable USER.
@VIEW@	The view name as defined by ClearCase.
@VOVARCH@	The value of the environment variable VOVARCH, normally set with `vovarch`.

Pre and Post Conditions

The execution of Pre and Post conditions based on the environment definition have been deprecated as of version 8.2.0. For current information, refer to Pre-Command and Post-Command Job Conditions. The information provided in this section describes behavior that is supported for backward compatibility.

As part of the environment definition, two scripts can be prepared to take care of pre- and post-conditions on a job-by-job basis. These scripts are called `NameOfEnv.pre.tcl` and `NameOfEnv.post.tcl`.

Configure FairShare via the policy.tcl File

The FairShare Groups policy configuration described in this section have been deprecated. *It is strongly advised to not use the described in this section. The features described in this section are currently available, but may soon become unavailable and/or unsupported.*

Refer to vovfsgroup for the current implementation.

To create new groups, define them in the `policy.tcl` file in the `PROJECT.swd` directory using *vtk_group_set*. This procedure is only valid in the `policy.tcl` file.

```
# Example of definition of some groups in the policy.tcl file.
# Reminder: this is considered obsolete. Use FSGROUP instead.
vtk_group_set /time/med/sanjose         -weight 200
vtk_group_set /time/med/sanjose/library -weight  40
vtk_group_set /time/med/sanjose/systems -weight  60
vtk_group_set /time/med/sanjose/systems/qa   -users { john mary }
vtk_group_set /time/med/sanjose/systems/dev  -users { bob suresh }
vtk_group_set /time/med/sanjose/systems/prot -unixgroup acxx
```

If a group is defined without a leading /, the group becomes part of the /time group by default.

Set Up FairShare

1. The `policy.tcl` file defines the FairShare weight for each fsgroup. The script uses the following Tcl procedures:
   - `vtk_fairshare_set -window <timeSpec>`

     Sets the value of the keyref window for the top group.
   - `vtk_group_set <group_name> [-weight w] [-users userlist] [-useasdefault] [-max N] [-map RESEXPRESSION] [-owner USER]`

     The default weight for a group is 100. The default user list is empty.
   - Example:

```
vtk_fairshare_set -window 4h
vtk_group_set /queue/alpha -weight 500 -users {mary frank}
vtk_group_set /queue/gamma -weight 800 -users {frank john alan}
vtk_group_set /queue/gamma -weight 800 -unixgroup acx
```

2. Each fsgroup with queued jobs is assigned a target share proportional to the FairShare weight.
3. The fsgroups are ranked based on how far they are from the target share taking into account:
   - The number of jobs currently running for each fsgroup;
   - The total run-time logged by each fsgroup over the FairShare window.
4. The farther a fsgroup is from the target, the lower its rank and the earlier their jobs are checked for scheduling.
5. As soon as a slot becomes available, the scheduler searches for the next job to dispatch starting from the user with lowest rank, and then increasing the rank until a job to dispatch is found.

Migration of Preemption from 2013.03 and Prior Versions

Starting in version 2013.09, Accelerator has a new implementation of the preemption functionality. Within this section, the terminology "older version" of preemption is used to indicate Accelerator versions 2013.03 and prior and "newer version" of preemption to indicate Accelerator versions 2013.09 or later.

Optional vovpreemptd Daemon

The Preemption section discussed some of the changes in preemption implementations as compared to prior versions of Accelerator. Specifically, the preemption functionality for older versions, a stand-alone daemon monitored the file-based preemption rules and that daemon also preformed the actual preemption. For newer version of preemption, the functionality is performed by the server directly and preemption rules are stored within the persistent representation of Accelerator so that this information is preserved across server restarts. There is an optional daemon that can be used to monitor preemption rules that are stored within the preemption `config.tcl` file.

For new users of preemption and for previous preemption users upgrading from older versions of Accelerator, the initial decision for preemption configuration is to decide if preemption rules will be file-based or entered via Accelerator. If the preemption rules are file-based and the administrator wishes for Accelerator to monitor the rules, then the optional `vovpremeptd` daemon will need to be started.

Server Configuration

As previously mentioned, Accelerator is now managing the preemption of jobs internally. The administrator can configure the server to specify the frequency of how often Accelerator is to attempt preemption of jobs. This is via the `preemptionPeriod` server configuration variable as described in Server Configuration. The default value is that Accelerator evaluates the preemption rules every 3 seconds. Setting this configuration value to 0 will disable preemption in the server and setting the value to a larger number will obviously cause preemption to occur on a less frequent basis.

VovPreemptRule Command Changes

`VovPreemptRule` has some new command options and some slightly modified commands option beginning in 2013.09. The two modified command options are -killage and -method.

The default of -killage was changed from a default of 1 minute to 0. Previously, by default, any preempted job younger than 1 minute was killed and resubmitted. Now, this is not the default and young jobs will only be killed and re-submitted if the -killage command option is specified.

The -method option previously supported RESERVE and FREE_TASKERS as preemption methods. A new concept called Preemption Rule Types factors RESERVE and FREE_TASKERS out as -method and are now valid values for -ruletype. Consequently, any preemption rules that use these values as preemption method need to instead specify the values as preemption type.

Convert Old VovPreemptPolicy into VovPreemptRule

An old and currently unsupported preemption policy is used for this example:

```
VovPreemptPolicy License:ncverilog -type OWNERSHIP:GROUP -delay 150 \
    -method SUSPEND+LMREMOVE -forcelmremove \
    -owners {
        Group:AAA 7
        Group:BBB 8
        Group:CCC 10
    }
## Note: this type of preemption policy is no longer supported because it
## results in a quadratic number O(N**2) of rules  where N is the
## number of owners.
```

This is replaced by a FairShare driven preemption. Assume that the jobs to be balanced are in FairShare groups called /xxx/AAA /xxx/BBB and /xxx/CCC and that these groups have been assigned weights of 700 800 and 1000 respectively

Two implementations are suggested:

Implementation for scripted preemption daemon (pre 2013.09)

```
## For preemption daemon before 2013.09
VovPreemptRule -rulename fsConversion \
    -preempting "FSGROUP~/xxx/ FS_EXCESS_RUNNING<0 FS_COUNT_RUNNING<2"
    -waitingfor "License:ncverilog" \
    -preemptable "FSGROUP~/xxx/ FS_EXCESS_RUNNING>0 FS_COUNT_RUNNING>2
FS_RANK9>@FS_RANK9@"
    -method SUSPEND+LMREMOVE
```

Implementation for binary preemption daemon (2013.09 and later)

```
## For preemption daemon 2013.09  and later
VovPreemptRule -rulename fsConversion  -poolname somePoolName \
    -ruletype "FAST_FAIRSHARE" \
    -preempting "FSGROUP~/xxx/ FS_EXCESS_RUNNING<0FS_COUNT_RUNNING<2" \
    -waitingfor "License:ncverilog" \
    -preemptable "FSGROUP~/xxx/ FS_EXCESS_RUNNING>0 FS_COUNT_RUNNING>2
FS_RANK9>@FS_RANK9@"
    -method SUSPEND+LMREMOVE
```

FairShare Groups

Note: The FairShare information described in this chapter has been deprecated. *It is strongly advised to not use the information described in this chapter. The features described in this section are currently available, but may soon become unavailable and/or unsupported.*

Refer to FairShare and Configure FairShare via the vovfsgroup Utility for the current implementations.

Each job belongs to both a user and a group. Groups are used as part of the FairShare mechanism.

To set the FairShare group of a job, set the environmenVOV_GROUP when creating the job using `vovbuild` or when executing the job for the first time. If the variable is missing, the FairShare group name is `/time/users` by default. The group of a job cannot be changed after the job has been created.

The VOV administrator defines the share of resources to allocate to a group by means of the procedure `vtk_group_set` in the `policy.tcl` file. The option -weight is used to specify a relative weight of the group. Example:

```
vtk_group_set /time/alpha -weight 500
vtk_group_set /time/beta  -weight 200
vtk_group_set /time/reg   -weight 200 -unixgroup regression
vtk_group_set /time/prod  -weight 200 -unixgroup production -useasdefault
```

To define the users that belong to a FairShare group, use the option -users or -unixgroup. Example:

```
# Explicit list of users:vtk_group_set /time/alpha -users { john mary joe }

# Take list from the Unix definition of groups:vtk_group_set /time/beta  -unixgroup
 guests
```

To define the share of resources to allocate to a user within a group, use `vtk_user_set` and the -group option. Example:

```
vtk_user_set john -weight 20   # Default is group "users"
vtk_user_set john -group /time/alpha -weight 100
vtk_user_set john -defaultGroup /time/beta
```

To see the list of all groups, refer to the /groups page.

Legal Notices

Intellectual Property Rights Notice

Copyrights, trademarks, trade secrets, patents and third party software licenses.

Altair Simulation Products

Altair® AcuSolve® ©1997-2023

Altair Activate®©1989-2023

Altair® Battery Designer™ ©2019-2023

Altair Compose®©2007-2023

Altair® ConnectMe™ ©2014-2023

Altair® EDEM™ © 2005-2023

Altair® ElectroFlo™ ©1992-2023

Altair Embed® ©1989-2023

Altair Embed® SE ©1989-2023

Altair Embed®/Digital Power Designer ©2012-2023

Altair Embed® Viewer ©1996-2023

Altair® ESAComp® ©1992-2023

Altair® Feko® ©1999-2023

Altair® Flow Simulator™ ©2016-2023

Altair® Flux® ©1983-2023

Altair® FluxMotor® ©2017-2023

Altair® HyperCrash® ©2001-2023

Altair® HyperGraph® ©1995-2023

Altair® HyperLife® ©1990-2023

Altair® S-TIMBER™ © 1995-2023

Altair® S-FOUNDATION™ © 1995-2023

Altair® S-CALC™ © 1995-2023

Altair® S-VIEW™ © 1995-2023

Altair® Structural Office™ © 2022-2023

Altair® SEAM® © 1985-2023

Altair® SimLab® ©2004-2023

Altair® SimLab® ST © 2019-2023

Altair SimSolid® ©2015-2023

Altair® ultraFluidX® ©2010-2023

Altair® Virtual Wind Tunnel™ ©2012-2023

Altair® WinProp™ ©2000-2023

Altair® WRAP™ ©1998-2023

Altair® GateVision PRO™ ©2002-2023

Altair® RTLvision PRO™ ©2002-2023

Altair® SpiceVision PRO™ ©2002-2023

Altair® StarVision PRO™ ©2002-2023

Altair® EEvision™ ©2018-2023

Altair Packaged Solution Offerings (PSOs)

Altair® Automated Reporting Director™ ©2008-2022

Altair® e-Motor Director™ ©2019-2023

Altair® Geomechanics Director™ ©2011-2022

Altair® Impact Simulation Director™ ©2010-2022

Altair® Model Mesher Director™ ©2010-2023

Altair® NVH Director™ ©2010-2023

Altair® NVH Full Vehicle™ © 2022-2023

Altair® NVH Standard™ © 2022-2023

Altair® Squeak and Rattle Director™ ©2012-2023

Altair® Virtual Gauge Director™ ©2012-2023

Altair® Weld Certification Director™ ©2014-2023

Altair® Multi-Disciplinary Optimization Director™ ©2012-2023

Altair HPC & Cloud Products

Altair® PBS Professional® ©1994-2023

Altair Data Analytics Products

Third Party Software Licenses

AcuConsole contains material licensed from Intelligent Light (www.ilight.com) and used by permission.

For a complete list of Altair Accelerator Third Party Software Licenses, please click here.

Technical Support

Altair provides comprehensive software support via web FAQs, tutorials, training classes, telephone and e-mail.

Altair One Customer Portal

Altair One (https://altairone.com/) is Altair's customer portal giving you access to product downloads, Knowledge Base and customer support. We strongly recommend that all users create an Altair One account and use it as their primary means of requesting technical support.

Once your customer portal account is set up, you can directly get to your support page via this link: www.altair.com/customer-support/.

Altair Training Classes

Altair training courses provide a hands-on introduction to our products, focusing on overall functionality. Courses are conducted at our main and regional offices or at your facility. If you are interested in training at your facility, please contact your account manager for more details. If you do not know who your account manager is, e-mail your local support office and your account manager will contact you

Telephone and E-mail

If you are unable to contact Altair support via the customer portal, you may reach out to the technical support desk via phone or e-mail. You can use the following table as a reference to locate the support office for your region.

When contacting Altair support, please specify the product and version number you are using along with a detailed description of the problem. It is beneficial for the support engineer to know what type of workstation, operating system, RAM, and graphics board you have, so please include that in your communication.

Location	Telephone	E-mail
Australia	+61 3 9866 5557 +61 4 1486 0829	anz-pbssupport@altair.com
China	+86 21 6117 1666	pbs@altair.com.cn
France	+33 (0)1 4133 0992	pbssupport@europe.altair.com
Germany	+49 (0)7031 6208 22	pbssupport@europe.altair.com
India	+91 80 66 29 4500 +1 800 208 9234 (Toll Free)	pbs-support@india.altair.com
Italy	+39 800 905595	pbssupport@europe.altair.com
Japan	+81 3 6225 5821	pbs@altairjp.co.jp
Korea	+82 70 4050 9200	support@altair.co.kr

Location	Telephone	E-mail
Malaysia	+91 80 66 29 4500 +1 800 208 9234 (Toll Free)	pbs-support@india.altair.com
North America	+1 248 614 2425	pbssupport@altair.com
Russia	+49 7031 6208 22	pbssupport@europe.altair.com
Scandinavia	+46 (0) 46 460 2828	pbssupport@europe.altair.com
Singapore	+91 80 66 29 4500 +1 800 208 9234 (Toll Free)	pbs-support@india.altair.com
South Africa	+27 21 831 1500	pbssupport@europe.altair.com
South America	+55 11 3884 0414	br_support@altair.com
United Kingdom	+44 (0)1926 468 600	pbssupport@europe.altair.com

See www.altair.com for complete information on Altair, our team and our products.

Index