SANDIA REPORT

SAND2010-7109
Unlimited Release
Printed October 2010

OVIS 3.2 User's Guide

J. Brandt, A. Gentile, C. Houf, J. Mayo, P. Pébay, D. Roe, D. Thompson, and M. Wong

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories

SAND2010-7109
Unlimited Release
Printed October 2010

OVIS 3.2 User's Guide

J. Brandt
Sandia National Laboratories
M.S. 9159, P.O. Box 969
Livermore, CA 94551, U.S.A.
brandt@sandia.gov

A. Gentile
Sandia National Laboratories
M.S. 9152, P.O. Box 969
Livermore, CA 94551, U.S.A.
gentile@sandia.gov

C. Houf
Sandia National Laboratories
M.S. 9012, P.O. Box 969
Livermore, CA 94551, U.S.A.
cahouf@sandia.gov

J. Mayo
Sandia National Laboratories
M.S. 9159, P.O. Box 969
Livermore, CA 94551, U.S.A.
jmayo@sandia.gov

P. Pébay
Sandia National Laboratories
M.S. 9159, P.O. Box 969
Livermore, CA 94551, U.S.A.
pppebay@sandia.gov

D. Roe
Sandia National Laboratories
M.S. 9152, P.O. Box 969
Livermore, CA 94551, U.S.A.
dcroe@sandia.gov

D. Thompson
Sandia National Laboratories
M.S. 9159, P.O. Box 969
Livermore, CA 94551, U.S.A.
dcthomp@sandia.gov

M. Wong
Sandia National Laboratories
M.S. 9152, P.O. Box 969
Livermore, CA 94551, U.S.A.
mhwong@sandia.gov

Abstract

This document[1] describes how to obtain, install, use, and enjoy a better life with OVIS version 3.2.

[1]Last Revision: October 9, 2010.

Contents

Figures

1 Introduction

This document is the user's guide for OVIS version 3.2. It is an update of the OVIS 2 User's Guide [6].

The OVIS project [11] targets scalable, real-time analysis of very large data sets. We characterize the behaviors of elements and aggregations of elements (e.g., across space and time) in data sets in order to detect meaningful conditions and anomalous behaviors. We are particularly interested in determining anomalous behaviors that can be used as advance indicators of significant events of which notification can be made or upon which action can be taken or invoked.

The OVIS open source tool (BSD license) is available for download at `ovis.ca.sandia.gov`. While we intend for it to support a variety of application domains, the OVIS tool was initially developed for, and continues to be primarily tuned for, the investigation of High Performance Compute (HPC) cluster system health. In this application it is intended to be both a system administrator tool for monitoring and a system engineer tool for exploring the system state in depth.

OVIS 3.2 provides a variety of statistical tools for examining the behavior of elements in a cluster (e.g., nodes, racks) and associated resources (e.g., storage appliances and network switches). It provides an interactive 3-D physical view in which the cluster elements can be colored by raw or derived element values (e.g., temperatures, memory errors). The visual display allows the user to easily determine abnormal or outlier behaviors. Additionally, it provides search capabilities for certain scheduler logs. The OVIS capabilities were designed to be highly interactive – for example, the job search may drive an analysis which in turn may drive the user generation of a derived value which would then be examined on the physical display.

The OVIS project envisions the capabilities of its tools applied to compute cluster monitoring. In the future, integration with the scheduler or resource manager will be included in a release to enable intelligent resource utilization [5, 7, 2, 3]. For example, nodes that are deemed less healthy (i.e., nodes that exhibit outlier behavior with respect to some set of variables shown to be correlated with future failure) can be discovered and assigned to shorter duration or less important jobs. Further, HPC applications with fault-tolerant capabilities would respond to changes in resource health and other OVIS notifications as needed, rather than undertaking preventative measures (e.g. checkpointing) at regular intervals unnecessarily.

More information about the OVIS project and publications describing the OVIS research in more detail can be found at `ovis.ca.sandia.gov` [11].

The OVIS team can be reached at `ovis-help@sandia.gov`.

1.1 New Features

New features in the OVIS 3.2 release (beyond OVIS 2) include:

- revisions to the database schema – particularly for metric data – that co-locate the data and timestamp, greatly improving performance;
- support for multiple Shepherds for both data collection and analysis;
- the ability to create tables of derived metrics for use in analysis and visualization;
- job log exploration; and
- revisions to the sampler classes and ovdb file specifications.

These are covered in more detail in this document.

The outlier detection functionality present in OVIS 2 is not yet incorporated in OVIS 3.2 but will be in a future release. More information on this feature can be found in [6].

2 Installation

This section contains build instructions for the OVIS release. You will be required to obtain and install supporting software that is not part of OVIS in a manner appropriate to your platform. Necessary supporting software is listed below. System settings for running OVIS are also given.

2.1 Hardware Requirements

These requirements apply to computers that will run the OVIS Baron GUI.

1. A Core 2 Duo or newer CPU and 2 GB or more RAM are recommended.
2. An NVIDIA graphics card, preferably a GeForce 8800 or newer or a Quadro FX (not NVS), is recommended.

2.2 Supporting Software

1. If possible, use a recommended operating system: Fedora 9 or newer, or Red Hat Enterprise Linux 5 or newer. Mac OS X can also be used.
2. Obtain and install a C compiler, a C++ compiler, and OpenGL version 1.2 or newer. These are probably already installed on your system.

   Regarding OpenGL: Video cards with NVIDIA chipsets for 3-D rendering combined with the NVIDIA-provided, closed-source OpenGL drivers are all we test and support, given the lack of features and/or stability other solutions currently provide. The `akmod-nvidia` package from `rpmfusion.net` is an alternative to manually installing NVIDIA drivers on Fedora and other RPM-based Linux distributions. Installing the `xorg-x11-drv-nvidia-libs` and `xorg-x11-drv-nvidia-devel` packages provides the remaining elements for OpenGL support.
3. Obtain and install database software – either MySQL or PostgreSQL will work. As of this writing the current RPMs for Fedora will work.

   MySQL:

   (a) Obtain and install MySQL at least version 5.0.77 (install client, server, devel, and shared)
   (b) Obtain and install MySQL-python

   PostgreSQL:

   (a) Obtain and install PostgreSQL at least version 8.2.9
   (b) Set the following in your environment:

```
setenv PGSQL_HOME /usr/local/pgsql
set path=($PGSQL_HOME/bin \$path)
```

4. Obtain and install Qt using the latest stable version of 4.6.x. As of this writing the current RPM for Fedora will work (Qt version 4.6.x). You will need the qt4-devel RPM and the RPM for the appropriate database plugin as well.

5. Obtain and install CMake at least version 2.8.1 from `http://www.cmake.org/`

6. Obtain and install Boost at least version 1.42.0 from `http://www.boost.org/`

7. You must provide a multicast domain name service (mDNS) service discovery (SD) daemon. This may be either Bonjour or Avahi. If you plan to use Avahi,

   - Obtain and install libdaemon.
   - Obtain and install dbus.
   - Obtain and install Avahi at least version 0.6.22. This may require you to get intltool for the install. On 64-bit systems, you will need version 0.6.23 or newer. Whatever version of Avahi you use, you must build the avahi-qt4 library if you plan to build the OVIS Baron GUI.
   - Build Avahi and install it as a system service on all nodes which will run any OVIS software. Configure Avahi as follows:

     ```
     setenv PKG_CONFIG_PATH /usr/local/lib/pkgconfig:$QTDIR/lib/pkgconfig
     ./configure --disable-python-dbus --disable-mono --disable-python \
        --disable-gtk --disable-qt3
     ```

   - If you use the RPM's install avahi-devel as well.

   If you plan to use Bonjour,

   - Obtain and install mDNSResponder on all nodes which will run any OVIS software.

2.3 OVIS Install

1. Obtain the OVIS source from `ovis.ca.sandia.gov`.

2. `mkdir /path/to/OVISBuildDir`

3. `cd /path/to/OVISBuildDir`

4. `ccmake /path/to/OVISSrcDir`

5. Set the following variables within ccmake to the values shown. (Type 'c' to configure and then 't' to see these options; ignore initial warnings.)

   `CMAKE_BUILD_TYPE: Debug`

`OPENGL_INCLUDE_DIR`: Location of NVIDIA headers (e.g., `/usr/lib64/nvidia`)

`OPENGL_gl_LIBRARY`: Location of NVIDIA OpenGL library (e.g., `/usr/lib64/nvidia/libGL.so`)

6. Type 'c' to configure again until you have a 'g' to generate option.
7. Type 'g' to generate.
8. `make`
9. `make install`

   # optional. Running as root may be needed

There is an additional parameter, `OVIS_SQL_FP_RTOL`, relating to the analyses, that can be set in ccmake. It is described further in Section 7.

The VTK source is included in the OVIS source, and VTK will get built as part of the OVIS build.

You are now done building OVIS 3.2!

Note that OVIS will place the plain text configuration file `${HOME}/.config/Sandia/ovis.conf` on your system. This will contain Baron state information (as described in §8) and database usernames and passwords should you choose to explicitly save them. (The ServerConnection window of Figure 17, described in §8 and §9.1, contains the control on whether this information is saved or not.)

2.4 MySQL Settings

1. If MySQL is not currently running (however, it should be if you are running on a system with Fedora 8 or greater installed):

   ```
   /sbin/chkconfig --level 345 mysqld on # run MySQL daemon at boot
   /sbin/service mysqld start # run MySQL daemon now
   ```

2. You should create a database user named ovis. This user will need full administrative privileges on the local machine and the ability to alter tables and insert records from machines where data will be collected. The administrative privileges are required to load a shared library (`libovis-mysql.so`) that contains functions used to signal the OVIS Shepherd process when rows in certain tables are inserted or modified. For connections from network interfaces that face outside the cluster, you may require a password for the ovis user. Remote connections from external networks will still require permission to insert records into the database in order to request analyses of collected data. For example, if you will be using OVIS to store data into a database called OVIS_Cluster:

```
mysql -u root -p
mysql> GRANT ALL PRIVILEGES ON OVIS_Cluster.* TO 'ovis'@'localhost';
mysql> GRANT ALL PRIVILEGES ON OVIS_Cluster.* TO 'ovis'@'localhostsipaddress';
       #similiarly for all addresses (including virbr0)
mysql> GRANT INSERT,DELETE,UPDATE,EXECUTE,SELECT ON OVIS_Cluster.* TO
 'ovis'@'someremotehost' IDENTIFIED BY 'somepassword';
mysql> flush privileges;
```

The remotehosts should include that of the nodes running the Sheep §5 that will insert into a given database, and that of all other Shepherds that may be participating in an analysis §7.2.

Note that you should configure mysqld so that no password is required to access the database from the private (administrative) network of your cluster or the local host where mysqld and the OVIS Shepherd process will run. This way, Sheep and Shepherd processes do not need to be configured with any database passwords. You can require a password for the machine that is running the Baron.

3. There is a user defined function that must be loaded into the MySQL database. This will go into the `mysql.func` table. It needs only to be added once – not on a per OVIS_Cluster basis. If you use the database effector described in §4.2, this will occur with the `-t 4` flag. If you load a mysqldump (such as the example one for the Glory database that comes with the release), ideally the function should be loaded as part of this process, however in practice the authors have seen that this may be dependent on the MySQL version. It is thus recommended that you run the database effector at this point. For MySQL, the database effector will run as the user `mysql` and will require that `/libovis-mysql.so`, built during the install be accessible by that user; directories that are so accessible are specified in, possibly, `/etc/ld.so.conf.d/mysql-x86_64.conf`, which may contain the entry `/usr/lib/mysql/plugin`.

   - Either
     - add the path to `libovis-mysql.so` to `mysql-x86_64.conf`
     - `/sbin/ldconfig -v`
       to update the paths

     or

     - add a link for, or copy, `libovis-mysql.so` to an existing accessible directory, such as `/usr/lib/mysql/plugin`
     - also, if applicable, use

       `chcon`

       to set the security context to that of other files that exist there.
   - after you run the database effector, check that the functions are indeed present in the `mysql.func` table.

.

4. Note: Some users have reported that VTK's MySQL interface cannot find `mysql.sock` despite its location being specified in the `/etc/my.cnf` configuration file. If this occurs, do the following:

   ```
   cd /tmp
   ln -s /var/lib/mysql/mysql.sock mysql.sock
   ```

2.5 PostgreSQL Settings

If you wish to use a PostgreSQL database to hold OVIS information, you will need to configure it appropriately.

1. You will need to edit the configuration file `/var/lib/pgsql/data/postgresql.conf` and change the `listen_addresses` setting to allow incoming connections from remote machines (both Sheep inserting measurements of cluster behavior and users connecting with the Baron to perform analysis). Unless you have a reason to specifically avoid a particular network interface, we suggest listening on all interfaces. We also recommend turning off informational messages printed by clients.

   ```
   listen_addresses = '*' # listen on all network interfaces
   client_min_messages = warning # print only warnings+errors
   ```

2. In addition to requesting that the daemon listen on network interfaces, you must specify how authentication should occur in `/var/lib/pgsql/data/pg_hba.conf` . For local connections or from network interfaces on the administrative network of a cluster, you should require no password so that Sheep and Shepherd processes may connect. For connections from network interfaces that face outside the cluster, you may require a password for the ovis user. As an example, consider the following lines:

   ```
   # TYPE  DATABASE     USER  CIDR-ADDRESS          METHOD
   ## Connections on the local machine
   local   ovis         ovis                        trust
   host    ovis         ovis  127.0.0.1/32          trust
   local   OVIS_Cluster ovis                        trust
   host    OVIS_Cluster ovis  127.0.0.1/32          trust
   ## Connections on private cluster admin network
   host    OVIS_Cluster ovis  192.168.1.254/24      trust
   ## Connections from remote sites. Requires password
   host    OVIS_Cluster ovis  74.125.19.19/24       ident
   ```

3. If the PostgreSQL postmaster daemon was running and you changed any of the configuration files above, you should restart it:

```
/sbin/service postgresql restart
```

Otherwise, if the daemon was not currently running, set it to run on reboot and then start it manually:

```
/sbin/chkconfig --level 345 postgresql on # run daemon at boot
/sbin/service postgresql start # run PostgreSQL daemon now
```

4. You should create a database user named ovis. This user will need full administrative privileges on the local machine and the ability to alter tables and insert records from machines where data will be collected. The administrative privileges are required to load a shared library (`libovis-psql.so`) that contains functions used to signal the OVIS Shepherd process when rows in certain tables are inserted or modified. For example, if you will be using OVIS to store data into a database called OVIS_Cluster:

```
createuser -s -d -l -P ovis  # You will be prompted for a password.
createdb -p -U ovis ovis     # Enter the password for ovis.
createdb -p -U ovis OVIS_Cluster
```

5. There is a user defined function that must be loaded into the PostgreSQL database. It needs only to be added once – not on a per OVIS_Cluster basis. If you use the database effector described in §4.2, this will occur with the `-t 4` flag.

2.6 Additional General System Settings

Finally, many Linux distributions will need some system settings changed, links created, and daemons turned on or off.

1. While it is possible to run the Shepherd process on systems with SELinux enabled, it is beyond the scope of this document to cover all of the configuration issues required. You may wish to configure your Shepherd nodes to run in permissive rather than enforcing mode.

2. Place the following lines in your iptables configuration file (`/etc/sysconfig/iptables` on most systems):

```
# Allow mDNS (also known as Avahi, Zeroconf, Bonjour)
-A INPUT -m state --state NEW -m udp -p udp --dport 5353 \
    -d 224.0.0.251 -j ACCEPT
# OVIS
-A INPUT -m state --state NEW -m udp -p udp --dport 49154 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 53170 -j ACCEPT
```

You may also need to add entries to allow PostgreSQL (port 5432) or MySQL (port 3306) connections, depending on which distribution of Linux you use and which database you prefer.

3. Set symbolic links for libraries.

   - If you have root and if you have done `make install` (note in the below that `/path/to/ovisInstallDir` is `/usr/local`):

     ```
     cd /usr/lib64 # if you are on a 64 bit machine, or /usr/lib if you are not
     ln -s /path/to/ovisInstallDir/lib64/libovis-mysql.so \
       /usr/lib/mysql/plugin/libovis-mysql.so
       # your mysql-plugin directory or similiar, as described earlier
     ln -s /path/to/ovisInstallDir/lib64/vtk-5.7/libvtksys.so.5.7 \
       libvtksys.so.5.7
     ln -s /path/to/ovisInstallDir/lib64/libovis-mysql.so libovis-mysql.so
     ln -s /path/to/ovisInstallDir/lib64/libvtkCommon.so.5.7 \
       libvtkCommon.so.5.7
     ln -s /path/to/ovisInstallDir/lib64/libvtkFiltering.so.5.7 \
       libvtkFiltering.so.5.7
     ln -s /path/to/ovisInstallDir/lib64/vtk-5.7/libvtksys.so.5.7 \
       libvtksys.so.5.7
     ```

   - Or if you have not done `make install`:
     - do the above *for the mysql-plugin directory only*, replacing `/path/to/ovisInstallDir` with `/path/to/ovisBuildDir`
     - you may also have to add, or copy, `libvtkCommon.so.5.7`, `libvtkFiltering.so.5.7`, and `libvtksys.so.5.7` to the mysql-plugins directory as well.
     - put `/path/to/ovisBuildDir` in your `LD_LIBARY_PATH`.

3 OVIS Components

In this section we describe the components of OVIS and their general interaction. For more information on the OVIS architecture see [4].

3.1 OVIS Components

OVIS uses a whimsical and intuitive analogy for naming its components.

- **Sheep** - the components which report (bleat) data values. Details on setting up the Sheep data collectors can be found in §5.2.
- **Shepherds** - the components which maintain the databases to which the Sheep report. Details on setting up the database can be found in §4.1 and §4.2.
- **Haruspices (singular Haruspex)** - analysis engines, used to determine potentially portentous abnormal behaviors. These are named after *haruspicy*, the practice of examining Sheep entrails for divination. Haruspices are described in more detail in §7.
- **Baron** - the GUI from which the data can be viewed and analyzed. Features of the Baron are given in §8 with an example in §9.1.

These components are shown in Figure 1. The Shepherds and the Haruspices that operate on the Shepherd's database are located on the same machine.

OVIS 3.2 supports multiple simultaneous Shepherds, each hosting (non-replicated) database backends for data insertion and analysis. Analysis occurs on each database separately, with the results aggregrated and the aggregate result presented in the Baron.

Visualization of the data on the 3-D display currently only works for the database (Shepherd) to which the Baron is connected. Concurrent visualization of data from all Shepherds is in work and will be part of a future 3.X release.

3.2 Running OVIS

Executables for the Sheep, Shepherd, Baron, and the associated database effector are in `/path/to/ovisBuildDir/bin`. A quick command reference is shown below for the test example described in §9.2:

- Running the database effector:

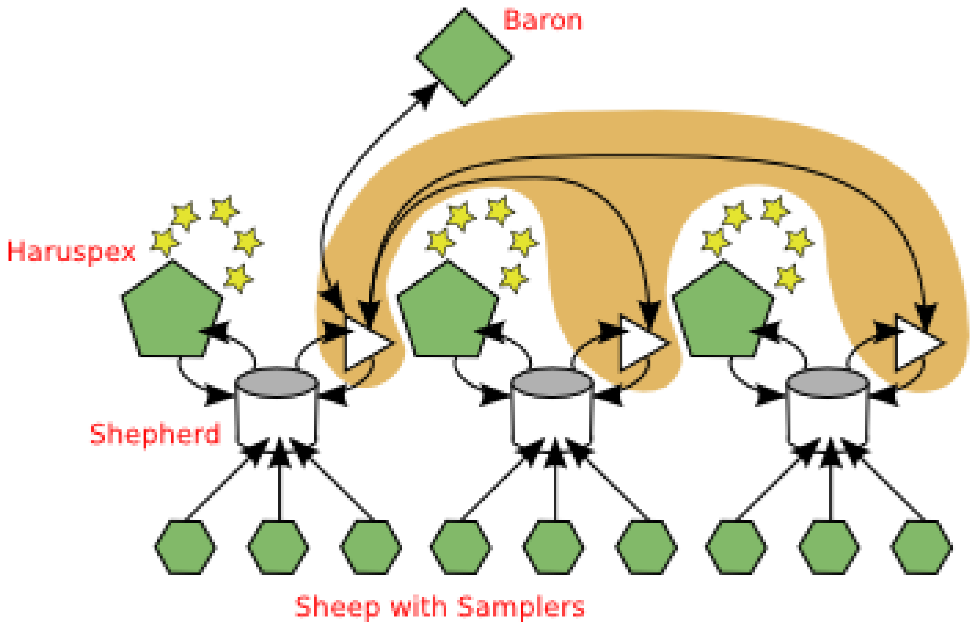


Figure 1: Components of OVIS: Sheep, Shepherds, and Baron.

```
./bin/ovis-db -d -t 16383 -u mysql://ovis@localhost/OVIS_Testone \
 -x /path/to/ovisSrcDir/data/testone.ovdb
```

- Running the Shepherd(s):

  ```
  ./bin/shepherd --name=Testone \
    --database=mysql://ovis@localhost/OVIS_Testone
  ```

  Note that you can only one run one Shepherd on a given node for a given cluster[2]. Multiple simultaneous Shepherds for a given collection/analysis are supported, but they must be run on different nodes. All such Shepherds should then advertise under the same name as indicated by the *name* flag.

- Running the Sheep:

  ```
  ./bin/sheep --name=Testone
  ```

- Running the Baron:

  ```
  ./bin/baron
  ```

Note that you do not always have to start all components. If you are only taking data and not examining it, you only need to start the Sheep and Shepherd. If you are only examining data in an existing database and are not taking any additional data, you only need to start the Shepherd and

[2]You can run multiple Shepherds for *different clusters* on the same node.

the Baron. If you only want to look at the geometry of a cluster and neither want to take data nor perform any analyses, you only need to start the Baron.

Also, note that the cluster name defaults to “Cluster” so if your site has a single cluster you need not specify the `--name` argument. Similarly, `psql://ovis@localhost/OVIS_Cluster` is the default value for the database URL so if this is satisfactory, you need not specify `--database`.

4 Setup

This section describes set up of the configuration files, samplers, and database necessary for running OVIS.

4.1 XML File

The XML file, canonically named `<clustername>.ovdb`, contains information on the cluster components, including additional components such as storage elements; their physical configuration; the metrics to be sampled; and IP addresses of the Sheep, Shepherd, and Baron nodes.

The OVIS 3.2 release comes with an example XML file for the Glory cluster, `gloryrelease.ovdb`. In most cases, it may be easiest to copy this file and edit it for your cluster. Metric data for this example is also included and will be discussed in §9.1.

An additional file, `whitneyterascalademo.ovdb` is included that illustrates use of OVIS for simultaneous monitoring of compute and storage resources. Finally, two small example files, `testone.ovdb` and `testonecpu.ovdb`, are included and discussed in §9.2 for use on your machine. The latter file also illustrates the specification for display of per-core data for multicore processors as discussed in §5.2.

This section describes items of note in the XML file.

Details of the XML File

In the *cluster* tag, the *name* field should be replaced with your cluster name.

The *component_types* section lists information for each type of component you have in your cluster, e.g., node, rack, switch. Each type should be listed separately in its own *component_type* block. Each component type should specify the following:

- type information - including the name by which the component will be identified, e.g., "rack" and whether it is a container or not, e.g., a rack will contain nodes and is therefore a container.
- drawing information that will represent that item's appearance (e.g., `render/Rack.vtp`, where the path is relative to the directory the ovdb file is in)
- size and slot information for drawing and the latter for determining the position of contained components if this component is a container. Units are in millimeters and we take 1 rack unit (1 RU or 1U) to be exactly 42 [*mm*] for convenience.
- sampler information - samplers that will run on this type of component, which metrics they will sample, and their interval. Samplers will be described in more detail in §5.2.

```
<component_types>
  <component_type
    type="rack" category="rack" is_container="1"
    glyph_shape="render/Rack.vtp"
    manufacturer="ASR"
    make="3100"
    model="ASR"
    description="TLCC cabinet"
    slot_origin="57.775,35,19"
    slot_size="500,700,44"
    slots_per_axis="1,1,42"
    slot_frame="1,0,0,  0,1,0,  0,0,1"
    height="1911" width="600" depth="737">
```

Figure 2: Excerpt from the Glory XML file pertaining to the rack.

```
 <component_type
   type="node" category="node" is_container="0"
   glyph_shape="render/RackNode1UFlat.vtp"
   glyph_image="render/Appro1ULight.vti"
   manufacturer="Appro"
   make="Supermicro"
   model="H8QM8"
   description="Glory compute node, Quad-Core AMD Opteron Processor 8354, 32XGB RAM"
   width="550" depth="600" height="44">

    <!-- Metrics -->

   <!-- Sampler for position metrics -->
   <sampler name="ovMetricNodePositionSampler" interval="-1">
     <metric name="PositionX"         units="m"
storage_type="float" frequency="once" constancy="time"/>
     <metric name="PositionY"         units="m"
storage_type="float" frequency="once" constancy="time"/>
     <metric name="PositionZ"         units="m"
storage_type="float" frequency="once" constancy="time"/>
   </sampler>

   <!-- lm sensors sampler -->
   <sampler name="ovMetricLinuxTLCClmsensorsSampler" interval="60">
    <metric name="CPU1_Temp" units="Celsius" storage_type="float" frequency="regular"/>
    <metric name="CPU2_Temp" units="Celsius" storage_type="float" frequency="regular"/>
    <metric name="CPU3_Temp" units="Celsius" storage_type="float" frequency="regular"/>
    <metric name="CPU4_Temp" units="Celsius" storage_type="float" frequency="regular"/>
    <metric name="SYS_Temp" units="Celsius" storage_type="float" frequency="regular"/>
    <metric name="VCoreA" units="V" storage_type="float" frequency="regular"/>
    <metric name="VCoreB" units="V" storage_type="float" frequency="regular"/>
    <metric name="VCoreC" units="V" storage_type="float" frequency="regular"/>
    <metric name="VCoreD" units="V" storage_type="float" frequency="regular"/>
    <metric name="3p3V" units="V" storage_type="float" frequency="regular"/>
    <metric name="5V" units="V" storage_type="float" frequency="regular"/>
    <metric name="12V" units="V" storage_type="float" frequency="regular"/>
    <metric name="5VSB" units="V" storage_type="float" frequency="regular"/>
    <metric name="VBATA" units="V" storage_type="float" frequency="regular"/>
    <metric name="FAN1" units="RPM" storage_type="float" frequency="regular"/>
    <metric name="FAN2" units="RPM" storage_type="float" frequency="regular"/>
    <metric name="FAN3_CPU2" units="RPM" storage_type="float" frequency="regular"/>
    <metric name="FAN4_CPU1" units="RPM" storage_type="float" frequency="regular"/>
    <metric name="FAN5" units="RPM" storage_type="float" frequency="regular"/>
    <metric name="FAN6" units="RPM" storage_type="float" frequency="regular"/>
    <metric name="FAN7_CPU4" units="RPM" storage_type="float" frequency="regular"/>
    <metric name="FAN8_CPU3" units="RPM" storage_type="float" frequency="regular"/>
    <metric name="FAN9" units="RPM" storage_type="float" frequency="regular"/>
   </sampler>
```

Figure 3: Excerpt from the Glory XML file pertaining to the node.

Selections of these blocks are shown in Figure 2 for the rack, which is a container with slots, and in Figure 3 for the node, which has samplers. The Glory cluster consists twelve nine racks with 288 compute nodes. This layout can be see in the OVIS 3.2 physical display in many figures in §9.1.

Include at least one type for Shepherd nodes in this section. Shepherd nodes will be used for interactions with the database when taking data and/or when performing analyses. (You do not always have to do both as described in §3.2.) A Shepherd node may also be a Sheep (i.e., running a sampler) node, so if you plan to use compute or I/O nodes as Shepherds you need not define a separate component type.

The *instances* section lists the number of each type of component. The order in which these occur will determine the overall component identifications – CompId in database tables (discussed in §4.2) – and will be significant in particular for remote samplers (discussed in §5.2).

OVIS 3.2 supports multiple simultaneous Shepherds, each hosting a database into which data is inserted and then used for analysis. Additionally, you can specify multiple Shepherds even if they are not deployed immediately, thus allowing yourself options as to where you will choose to start a Shepherd at a later time. In this example this would take the form of increasing the number of instances of component type "ts" in Figure 4 (top) and assigning different address information to each of those instances.

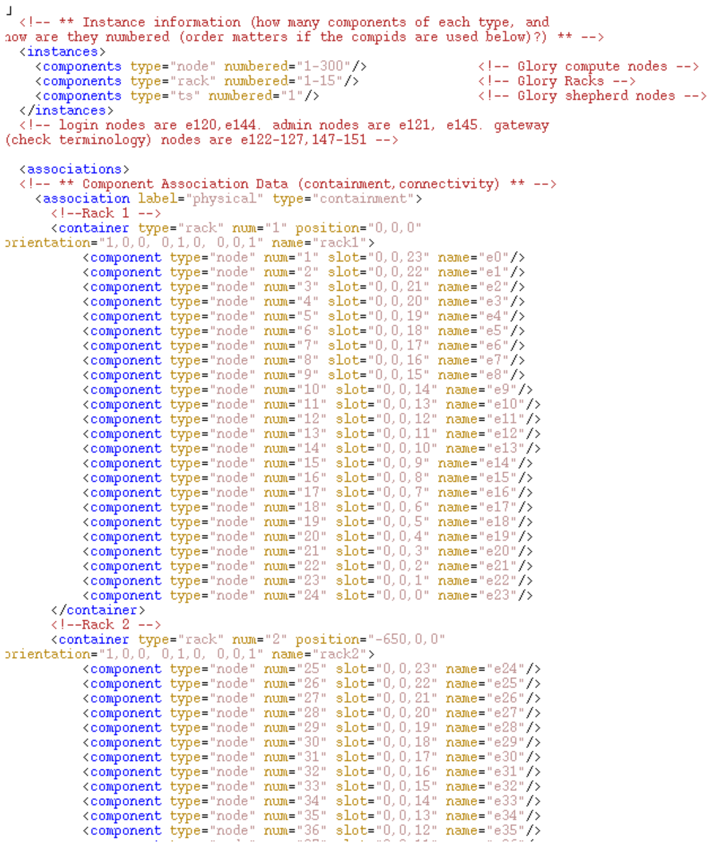
For Shepherds that need not run simultaneously (i.e., failover nodes), you may assign multiple network addresses to the same component instance (i.e., the same CompId). The specification of address information is described below.

The *associations* section specifies the physical layout. Containers and contained components and their relative associations are specified. Containers must specify their type, their instance (by "num"), their overall position and orientation in space, and, optionally a short name by which they can be identified. Contained components must specify their type, their instance, the slot in the container in which they reside (number of slots were specified in the *component_type* block, and an optional short name (e.g., for Glory node 1, "e0").

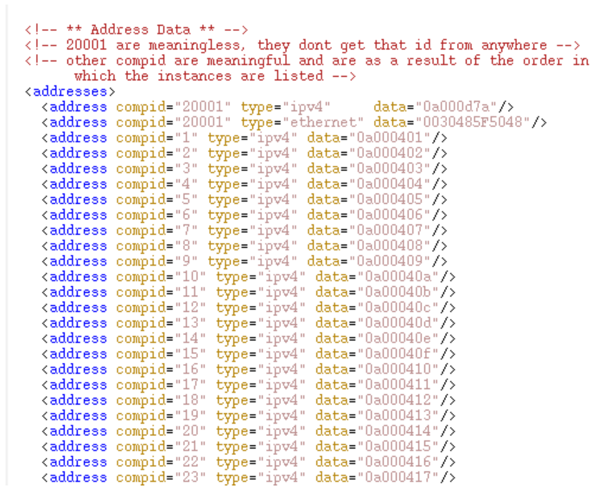
Contained components can also be used to illustrate components of a node – e.g., fans or cores. Figure 5 illustrates the use of contained components for a multi-core display. The sampler for this case is described in §5.2.

Not all components in the *instances* must be in the *associations* section – for instance, you may not want the Shepherd nodes to appear in the 3-D physical view. Examples of these blocks are in Figure 4 (top).

The *addresses* section gives IP and MAC addresses by which Sheep and Shepherds are identified. An excerpt of this section is in Figure 4 (bottom). While multicast DNS (Avahi or Bonjour) is used by Shepherds to advertise their availability, each component must have a unique identifier in order to insert information in the database. These identifiers appear in the CompId column of the **StartupData** table. You may specify addresses explicitly in the *addresses* section or you may use a regular expression to transform values from a column in the **StartupData** table into a component id (CompId) with a *hint*. Explicit address entries look like either of the two options below in the

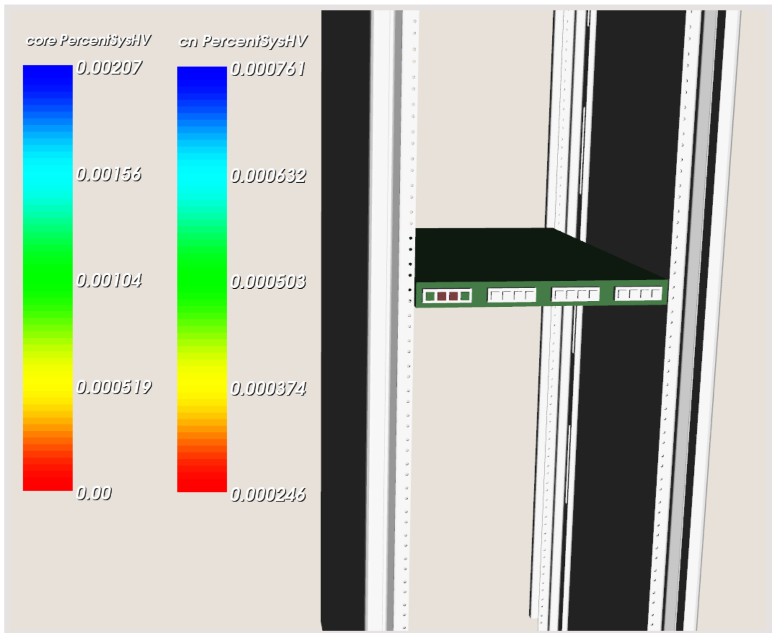


```
┘
  <!-- ** Instance information (how many components of each type, and
ow are they numbered (order matters if the compids are used below)?) ** -->
  <instances>
    <components type="node" numbered="1-300"/>             <!-- Glory compute nodes -->
    <components type="rack" numbered="1-15"/>              <!-- Glory Racks -->
    <components type="ts" numbered="1"/>                    <!-- Glory shepherd nodes -->
  </instances>
  <!-- login nodes are e120,e144. admin nodes are e121, e145. gateway
(check terminology) nodes are e122-127,147-151 -->

  <associations>
  <!-- ** Component Association Data (containment,connectivity) ** -->
    <association label="physical" type="containment">
      <!--Rack 1 -->
      <container type="rack" num="1" position="0,0,0"
rientation="1,0,0, 0,1,0, 0,0,1" name="rack1">
          <component type="node" num="1" slot="0,0,23" name="e0"/>
          <component type="node" num="2" slot="0,0,22" name="e1"/>
          <component type="node" num="3" slot="0,0,21" name="e2"/>
          <component type="node" num="4" slot="0,0,20" name="e3"/>
          <component type="node" num="5" slot="0,0,19" name="e4"/>
          <component type="node" num="6" slot="0,0,18" name="e5"/>
          <component type="node" num="7" slot="0,0,17" name="e6"/>
          <component type="node" num="8" slot="0,0,16" name="e7"/>
          <component type="node" num="9" slot="0,0,15" name="e8"/>
          <component type="node" num="10" slot="0,0,14" name="e9"/>
          <component type="node" num="11" slot="0,0,13" name="e10"/>
          <component type="node" num="12" slot="0,0,12" name="e11"/>
          <component type="node" num="13" slot="0,0,11" name="e12"/>
          <component type="node" num="14" slot="0,0,10" name="e13"/>
          <component type="node" num="15" slot="0,0,9" name="e14"/>
          <component type="node" num="16" slot="0,0,8" name="e15"/>
          <component type="node" num="17" slot="0,0,7" name="e16"/>
          <component type="node" num="18" slot="0,0,6" name="e17"/>
          <component type="node" num="19" slot="0,0,5" name="e18"/>
          <component type="node" num="20" slot="0,0,4" name="e19"/>
          <component type="node" num="21" slot="0,0,3" name="e20"/>
          <component type="node" num="22" slot="0,0,2" name="e21"/>
          <component type="node" num="23" slot="0,0,1" name="e22"/>
          <component type="node" num="24" slot="0,0,0" name="e23"/>
      </container>
      <!--Rack 2 -->
      <container type="rack" num="2" position="-650,0,0"
rientation="1,0,0, 0,1,0, 0,0,1" name="rack2">
          <component type="node" num="25" slot="0,0,23" name="e24"/>
          <component type="node" num="26" slot="0,0,22" name="e25"/>
          <component type="node" num="27" slot="0,0,21" name="e26"/>
          <component type="node" num="28" slot="0,0,20" name="e27"/>
          <component type="node" num="29" slot="0,0,19" name="e28"/>
          <component type="node" num="30" slot="0,0,18" name="e29"/>
          <component type="node" num="31" slot="0,0,17" name="e30"/>
          <component type="node" num="32" slot="0,0,16" name="e31"/>
          <component type="node" num="33" slot="0,0,15" name="e32"/>
          <component type="node" num="34" slot="0,0,14" name="e33"/>
          <component type="node" num="35" slot="0,0,13" name="e34"/>
          <component type="node" num="36" slot="0,0,12" name="e35"/>
```



```
  <!-- ** Address Data ** -->
  <!-- 20001 are meaningless, they dont get that id from anywhere -->
  <!-- other compid are meaningful and are as a result of the order in
        which the instances are listed -->
  <addresses>
    <address compid="20001" type="ipv4"     data="0a000d7a"/>
    <address compid="20001" type="ethernet" data="0030485F5048"/>
    <address compid="1" type="ipv4" data="0a000401"/>
    <address compid="2" type="ipv4" data="0a000402"/>
    <address compid="3" type="ipv4" data="0a000403"/>
    <address compid="4" type="ipv4" data="0a000404"/>
    <address compid="5" type="ipv4" data="0a000405"/>
    <address compid="6" type="ipv4" data="0a000406"/>
    <address compid="7" type="ipv4" data="0a000407"/>
    <address compid="8" type="ipv4" data="0a000408"/>
    <address compid="9" type="ipv4" data="0a000409"/>
    <address compid="10" type="ipv4" data="0a00040a"/>
    <address compid="11" type="ipv4" data="0a00040b"/>
    <address compid="12" type="ipv4" data="0a00040c"/>
    <address compid="13" type="ipv4" data="0a00040d"/>
    <address compid="14" type="ipv4" data="0a00040e"/>
    <address compid="15" type="ipv4" data="0a00040f"/>
    <address compid="16" type="ipv4" data="0a000410"/>
    <address compid="17" type="ipv4" data="0a000411"/>
    <address compid="18" type="ipv4" data="0a000412"/>
    <address compid="19" type="ipv4" data="0a000413"/>
    <address compid="20" type="ipv4" data="0a000414"/>
    <address compid="21" type="ipv4" data="0a000415"/>
    <address compid="22" type="ipv4" data="0a000416"/>
    <address compid="23" type="ipv4" data="0a000417"/>
```

Figure 4: Excerpts from the Glory file pertaining to the instances and associations (top) and addresses (bottom). The relationship between `compid`, `component type`, and `component num` ensures that all information for a given component is properly associated.



```
<association label="physical" type="containment">
  <container type="rack" num="1" position="0,0,0" orientation="1,0,0, 0,1,0, 0,0,1">
    <container type="cn" num="1" slot="0,0,23" name="e0">
      <container type="cpu" num="1" slot="0,0,0" name="e0c0">
        <component type="core" num="1" slot="0,0,0" name="e0c0c0"/>
        <component type="core" num="2" slot="1,0,0" name="e0c0c1"/>
        <component type="core" num="3" slot="2,0,0" name="e0c0c2"/>
        <component type="core" num="4" slot="3,0,0" name="e0c0c3"/>
      </container>
      <container type="cpu" num="2" slot="1,0,0" name="e0c1">
        <component type="core" num="5" slot="0,0,0" name="e0c1c0"/>
        <component type="core" num="6" slot="1,0,0" name="e0c1c1"/>
        <component type="core" num="7" slot="2,0,0" name="e0c1c2"/>
        <component type="core" num="8" slot="3,0,0" name="e0c1c3"/>
      </container>
      <container type="cpu" num="3" slot="2,0,0" name="e0c2">
        <component type="core" num="9" slot="0,0,0" name="e0c2c0"/>
        <component type="core" num="10" slot="1,0,0" name="e0c2c1"/>
        <component type="core" num="11" slot="2,0,0" name="e0c2c2"/>
        <component type="core" num="12" slot="3,0,0" name="e0c2c3"/>
      </container>
      <container type="cpu" num="4" slot="3,0,0" name="e0c3">
        <component type="core" num="13" slot="0,0,0" name="e0c3c0"/>
        <component type="core" num="14" slot="1,0,0" name="e0c3c1"/>
        <component type="core" num="15" slot="2,0,0" name="e0c3c2"/>
        <component type="core" num="16" slot="3,0,0" name="e0c3c3"/>
      </container>
    </container>
  </container>
</association>
```

Figure 5: Display (top) and assocation specification (bottom) of contained components used to illustrate a node with four CPUs, each of which has four cores. The association excerpt is from the `testonecpu.ovdb` file.

XML file:

```
<addresses>
  <address compid="2" type="ipv4"     data="481d53a4"/>
  <address compid="2" type="ethernet" data="0017deadbeef"/>
</addresses>
```

```
<addresses>
  <address comptype="cn" compnum="1" type="ipv4"     data="481d53a4"/>
  <address comptype="cn" compnum="2" type="ipv4"     data="481d53a5"/>
</addresses>
```

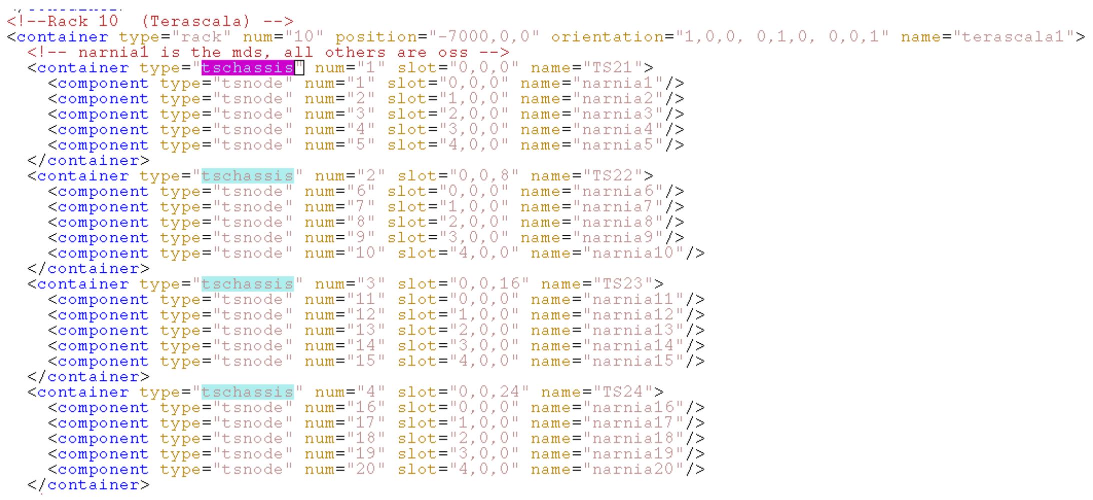
where the type attribute should be either “ipv4” (for TCP/IP addresses) or “ethernet” (for MAC addresses) and the data attribute should be a hexadecimal number that is either 4 or 6 bytes long depending on the address type.

Hints are specified like so:

```
<addresses>
  <hint>
    <key column="NodeName">
      <regex comptype="cn" compnum="%1">cn([0-9]+)</regex>
      <regex comptype="cn" compnum="1">admin</regex>
    </key>
  </hint>
</addresses>
```

Each *hint* tag (there may be several) must always contain exactly one *key* tag. The column attribute of the *key* tag specifies a column in the **StartupData** table. Note that when the Sheep program is run on a component without an entry in **StartupData**, the Sheep will run a special metric sampler that adds columns to **StartupData** containing the node’s name (NodeName in the example above), operating system, kernel version, and other information reported by the `uname` command. This results in many rows in **StartupData** with network addresses and other information specified but no values in the CompId column that assigns a unique id to the component by its network address. You may use *regex* tags in the XML file to assign a component number to any entry in **StartupData** by extracting a numeric substring from the key column’s value with a regular expression. In the example above, any entry in the NodeName column that starts with “cn” and ends with a decimal number will be assigned the CompId corresponding to the “cn” CompType and whose *instance* number matches the number in the hostname. It is also possible to create regular expressions that match only a given hostname (such as the second regular expression in the example) and specify both the component type and number directly. Note that component *numbers* are unique across all components of a given type while component *ids* are unique across all components of all types. The **ComponentTable** contains a mapping between component numbers and ids. If you use hints to specify component numbers (and thus ids), you must allow the Sheep processes to insert their network addresses, host names, and other data into the **StartupData** table and then run `ovis-db` with `-t 512` (c.f. §4.2) to assign values to the CompId column in **StartupData**. Only after you do this will the Sheep be able to insert metric data into the database.

If your machines have multiple addresses, list all of them in this section. There is no way to force OVIS to use a particular interface for communications; the addresses are only used for identification.



```
<!--Rack 10  (Terascala) -->
<container type="rack" num="10" position="-7000,0,0" orientation="1,0,0, 0,1,0, 0,0,1" name="terascala1">
  <!-- narnia1 is the mds, all others are oss -->
  <container type="tschassis" num="1" slot="0,0,0" name="TS21">
    <component type="tsnode" num="1" slot="0,0,0" name="narnia1"/>
    <component type="tsnode" num="2" slot="1,0,0" name="narnia2"/>
    <component type="tsnode" num="3" slot="2,0,0" name="narnia3"/>
    <component type="tsnode" num="4" slot="3,0,0" name="narnia4"/>
    <component type="tsnode" num="5" slot="4,0,0" name="narnia5"/>
  </container>
  <container type="tschassis" num="2" slot="0,0,8" name="TS22">
    <component type="tsnode" num="6" slot="0,0,0" name="narnia6"/>
    <component type="tsnode" num="7" slot="1,0,0" name="narnia7"/>
    <component type="tsnode" num="8" slot="2,0,0" name="narnia8"/>
    <component type="tsnode" num="9" slot="3,0,0" name="narnia9"/>
    <component type="tsnode" num="10" slot="4,0,0" name="narnia10"/>
  </container>
  <container type="tschassis" num="3" slot="0,0,16" name="TS23">
    <component type="tsnode" num="11" slot="0,0,0" name="narnia11"/>
    <component type="tsnode" num="12" slot="1,0,0" name="narnia12"/>
    <component type="tsnode" num="13" slot="2,0,0" name="narnia13"/>
    <component type="tsnode" num="14" slot="3,0,0" name="narnia14"/>
    <component type="tsnode" num="15" slot="4,0,0" name="narnia15"/>
  </container>
  <container type="tschassis" num="4" slot="0,0,24" name="TS24">
    <component type="tsnode" num="16" slot="0,0,0" name="narnia16"/>
    <component type="tsnode" num="17" slot="1,0,0" name="narnia17"/>
    <component type="tsnode" num="18" slot="2,0,0" name="narnia18"/>
    <component type="tsnode" num="19" slot="3,0,0" name="narnia19"/>
    <component type="tsnode" num="20" slot="4,0,0" name="narnia20"/>
  </container>
```

Figure 6: Excerpt from the Whitney-Terascala XML file with the association information for the Terascala storage rack.

Note that OVIS can simultaneously monitor any number and type of resources. The example file `whitneyterascalademo.ovdb` includes specification information for the Whitney cluster and an associated Terascala [12] Storage rack which is a container of 4 chassis each of which contains 5 blades. The *associations* section of this file specifying this rack is shown in Figure 6. The resultant OVIS physical display for the combined resources is shown in Figure 7.

4.2 Database Effector

Once you have a properly formatted ovdb XML file (located, say, at `/path/to/cluster.ovdb`), you should run the `ovis-db` program to populate a database for the Sheep, Shepherds, and Baron to use given the XML file. The `ovis-db` utility also allows you to insert test data (formatted as an XML ovdata file) into a database but that is not covered here.

First, you should create the database on each host that will be running a Shepherd. The name of the database must be OVIS_⟨*ClusterName*⟩ where ⟨*ClusterName*⟩ matches the *name* attribute of the *cluster* tag of your ovdb file. We will assume ⟨*ClusterName*⟩ is "Cluster" for the rest of this section. For MySQL, run this command to create the database

```
echo "CREATE DATABASE OVIS_Cluster;" | mysql -u ovis
```

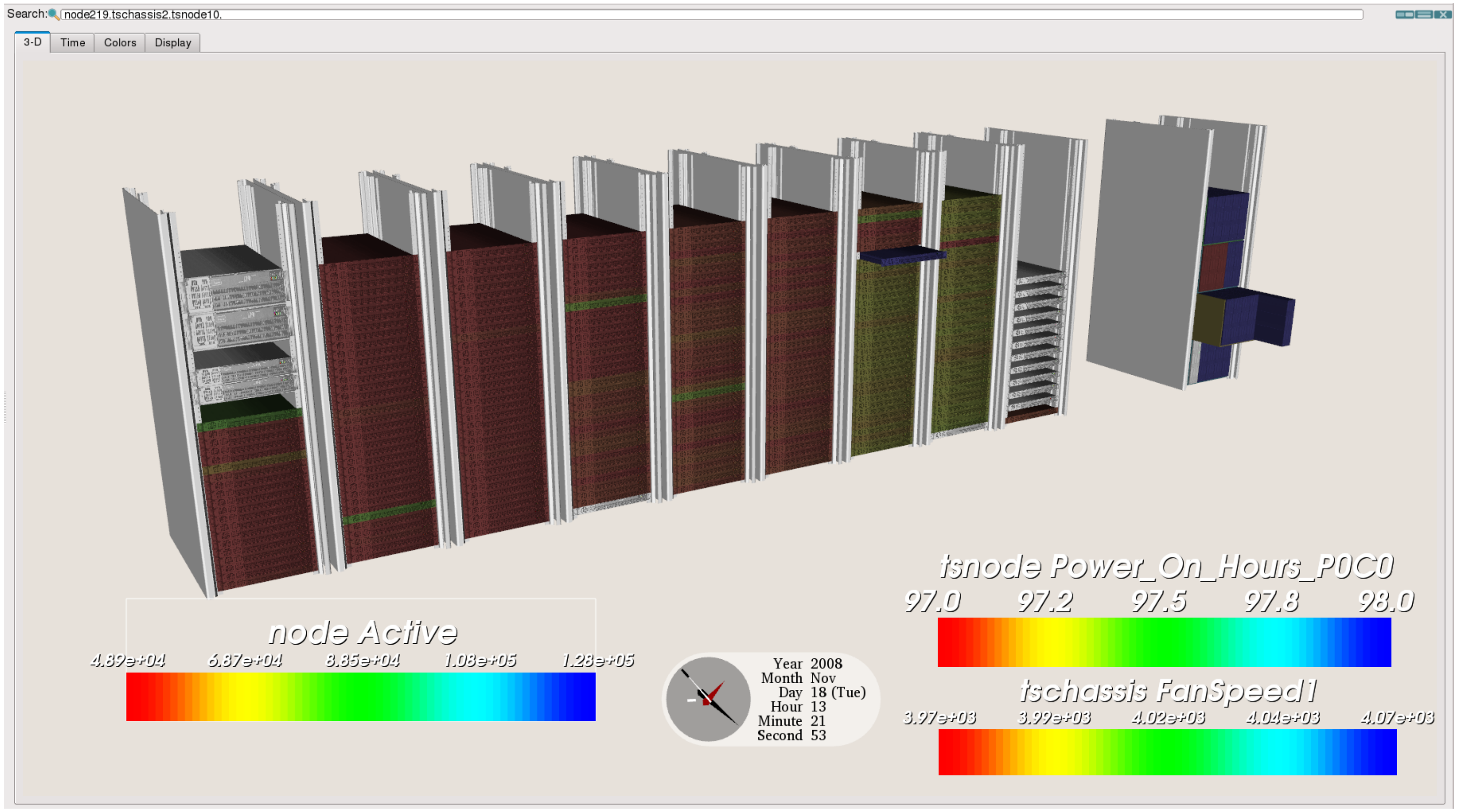


Figure 7: Physical display of the Whitney compute cluster nodes and the Terascala Storage Rack. For the compute nodes, Active Memory is displayed; for the Terascala Chassis, Fan Speed is displayed; for the Terascala Blades, Power On Hours nare displayed. One instance of each of these component types is popped out in the figure. Metric values for the Terascala nodes are being obtained via remote samplers running on the Terascala admin node (not shown).

Constant	Action
1	Create the database
2	Create tables
4	Load dynamic library functions into the database server
8	Populate ComponentTypes table
16	Populate MetricValueTypes table
32	Populate MetricValueTableIndex table
64	Populate Components table
128	Populate MetricCollectionSamplers table
256	Populate RemoteSamplerLookup table
512	Populate StartupData table
1024	Populate ComponentGlyphs table
2048	Prepare trigger actions
4096	Populate metric tables with static data (e.g. component coordinates)

Table 1: Numeric constants that may be summed and passed to the `ovis-db`'s `-t` flag specifying which actions are to be performed.

For PostgreSQL, run this command

```
createdb -U ovis OVIS_Cluster
```

Once the database exists, run `ovis-db` to populate it. For MySQL databases use:

```
ovis-db -d -t 8191 -x /path/to/cluster.ovdb \
  -u mysql://ovis@localhost/OVIS_Cluster
```

For PostgreSQL databases use:

```
ovis-db -d -t 8191 -x /path/to/cluster.ovdb \
  -u psql://ovis@localhost/OVIS_Cluster
```

The `-d` flag tells `ovis-db` to drop any existing rows or tables as necessary. The `-t 8191` option instructs `ovis-db` which actions to perform. The number specified is a bit vector composed by adding numbers from Table 1. The actions are executed in the order they appear in the table. Generally you will only need to run this command once as specified above.

While most actions simply create or populate tables, some perform less trivial tasks and order is important. The dynamic libraries (`libovis-mysql.so` and/or `libovis-psql.so`) must be installed into their proper locations before `ovis-db` is asked to have the SQL server process load them. The SQL server must have loaded these dynamic libraries before the database triggers are prepared since the triggers invoke functions provided by these libraries[3]. If you load the example

[3]Specifically, the function `signal_local_haruspex` is provided to send a UDP message to the local Shepherd when rows of the HaruspexRequests or haruspex subresults tables discussed later are inserted or modified.

Glory mysqldump for your first test case, rather than creating the database tables via the effector as described above, it is recommended that you run the effector with flag `-t 4` in order to ensure that this function is loaded, in case it is not loaded as part of loading the mysqldump.

Within MySQL, you may check the `mysql.func` table to see if the functions have been loaded, and call `show triggers` on your particular database to check to see if the triggers (which call the functions) are present.

Database Tables

Information in the XML file is used to set up the database tables. Of particular interest regarding the cluster specification are:

- **ComponentTypes** - information about the Component Types
- **ComponentTable** - associates a given Component Type and number with the unique CompId and its name.
- **StartupData** - for each component address information and samplers

If you are running multiple Shepherds, each Shepherd's database should contain consistent and complete information about the components – that is, a consistent and complete set of the CompIds. Sheep will discover the possible options for Shepherds via the Shepherds' Avahi advertisements and will randomly select one in which to insert data. Thus, each possible database will contain the data from only a subset of the Sheep. Correct results for analysis and display then rely on CompIds being consistent across the full set of databases.

Database tables for the samplers are described in more detail in §5.2.

5 Getting Data into the Database

The OVIS data tables are principally standalone entities; except for component IDs, they do not reference other tables using foreign keys or other indirect references and are themselves only referenced by other tables that (1) index all data tables; (2) enumerate the storage mode, component type, and other attributes of the data tables; and (3) describe the relationship between samplers, the tables holding the sampled data, the components performing the sampling, and the components which the samples refer to. Each data table contains an integer component ID, the time at which the row was inserted, and the data value.

Because the data value tables are largely self-contained, there are several options for inserting data into the tables. The primary method is run-time insertion of data via the Data Samplers. A second method is via the `Metric Generator` option which allows users to create new data values into new tables; this prinicpally targets the creation of derived data based on existing data, such as accumulating error counts over a job, or time-based averages. The final option, recommended for advanced users only, is direct insertion via some database API into a user-created table.

Data insertion options are discussed in this section.

5.1 Data Tables

There are separate database tables for each combination of component type and metric that are canonically named **'Metric'** ***ComponentType MetricName*** **'Values'**. As an example, compute nodes of type `cn` with a metric named CPUTemp would have metric values stored in a table named **MetricCnCPUTempValues**. Each row corresponds to a unique reporting of a metric value and its associated component.

Each data table contains:

- TableKey - an autoincrementing integer
- CompId - an integer component id
- Time - the time of the data point
- Value - the data value

Each table is listed in the **MetricValueTableIndex**:

- TableId - an autoincrementing integer
- TableName - the name of the metric table, as described above
- CompType - the component type for the data

- ValueType - metadata information about the type the data, which will be decribed in more detail below
- SamplerId - an integer indicative of the `Sampler` which collects this data
- Stride - to be described later

Each ValueType is listed in the **MetricValueTypes**:

- ValueType - an autoincrementing integer
- Name - the Metric Name
- Units - its units
- Storage - the storage type - e.g., float
- Constant - 1 or 0 indicating if the metric is a constant or not

Thus, if a new metric is to be added to the system, a new table for the data must be added, that table must then be added to the **MetricValueTableIndex** and information about the data value must be added to the **MetricValueTableIndex**. After that, data can be continously inserted, by any of several methods into the data table.

Insertion of data via the `Metric Collection Samplers` and the `Metric Generator` handle all the details of creating the data table and inserting the correct information into the **MetricValueTableIndex** and the **MetricValueTypes** tables. If you choose to bypass these methods you should handle these details yourself.

Analysis and display of data relies on data to be inserted increasing in time. It will not be fatal if data points are out of order, but it maybe result in some minor issues as described further in §7.2.

Do not use hyphens in your metric names.

5.2 Metric Collection Samplers

The `Metric Collection Samplers` collect specific data from specific sources and insert that data associated with the correct component and time into the database. Insertion of data by this method results in the data being timestamped by the time of the insertion into the database.

The OVIS 3.2 release comes with several samplers, some of which read from well-known locations on Linux installations and will probably work as is, and some which will have to be altered to work with your system. The former include those that read from `/proc`, such as `ovMetricLinuxProcStatSampler` and `ovMetricLinuxProcMemInfoSampler`. The latter will require some modification of configuration information, such as available metrics from the source or path information as in, for example, `ovMetricLinuxlmsensorsSampler` and `ovMetricExampleLinuxSmartctlSampler`.

Sampler Classes and Functions

The `Metric Collection Samplers` collect specific data from specific sources and insert that data, associated with the correct component and time into the database.

Samplers are required to support the following functions:

- `GetNumberOfMetrics`
- `GetMetricName`
- `GetMetricStorageType`
- `GetMetricFrequencyStyle`
- `SetMetricStride`
- `Sample`

In the `Sample` function, the sampler gets the data from its source and inserts the metric data value into the database via its `RecordXXX` (e.g., `RecordInt`) function that takes the data value and a number identifier that uniquely distinguishes the component and the metric. Note that some metrics lend themselves to reporting as instantaneous values while others as differences from their last value, as for counters. Other required functions in the sampler, such as `GetMetricName` and `GetMetricStorageType`, associate a particular number identifier with information about a particular metric.

`SetMetricStride` is an option, which, when also supported in the `Sample` function, enables the user to dynamically change how often the data which is sampled is written to the database – the `Sample` function can be written to increment a counter which can then be compared against the stride to determine whether or not the sampled value should actually be recorded. This is not always implemented in the samplers that come with OVIS.

Samplers can be defined to be run continously and inserting data continously (or upon certain conditions, for example, a sampler that reads error counts may only sampler an error has occured), or can be set, via the `FrequencyStyle` to read and record only once at Startup, such as `ovMetricPosixUnameSampler`.

Samplers can either be *local* or *remote*. A local sampler is a process running on machine *A* that collects metrics related to machine *A* and then inserts them on behalf of itself into an OVIS database – which is usually some remote machine *C* but could also be *A*. An example of this is `ovMetricLinuxProcNetDevSampler` which runs on the node and inserts data into the database associated with the node. A remote sampler is a process running on machine *A* that collects metrics related to some other machine *B* and then inserts them on behalf of machine *B* into an OVIS database – which is usually some remote machine *C* but could also be *A*, but this would be odd). Examples of this include a) `ovMetricLinuxlmsensorsSampler` which collects data on a node, but may associate that data with a CPU of that node or b) `ovMetricLinuxSNMPExampleSampler` that collects data remotely from another component (e.g. SNMP) and inserts that data such that it is associated with the remote component. In the case of the combined Whitney-Terascala monitoring, all of the Terascala samplers were remote with chassis data being collected via an SNMP sampler, and blade and associated disk data being collected via both an SNMP sampler and a Smartctl sampler. These remote samplers were running on the Terascala admin node.

Because of the potential complexity in handling the remote associations, a variety of sampler base classes exist to ease in the creation of specific samplers.

Base classes for `Samplers` are:

- `ovMetricCollectionSampler` - Base class for all samplers
- `ovMetricCollectionSamplerLocal` - Samples for the local component
- `ovMetricCollectionSamplerRemote` - Samples for remote and/or local components, by default via the `MetricNodeMap`
- `ovMetricCollectionSamplerDynamicKey` - Samples for remote and/or local components; metrics can be added dynamically; known metrics but for different component ids can be added dynamically
- `ovMetricCollectionSamplerAssociation` - Samples for remote and/or local components that are children of the local component

For local samplers, unique distinguishing of the component with which a particular data value should be associated is done innately by the identity of the Sheep. The `RecordXXX` function when called from a subclass of `ovMetricCollectionSamplerLocal` automatically records the data value with the CompId being that of the local component.

The situation is more complex for remote samplers, however. Not only must the remote sampler associate a particular numeric identifier with a particular metric, but the numeric identifier must also be used to distinguish the component. For example, a remote sampler that keeps track of 3 metrics on behalf of 5 other components will use 15 unique identifiers. The `ovMetricCollectionSamplerRemote` class and its derived classes all exist to enable more generic mapping.

The most generic mapping is enabled by `ovMetricCollectionSamplerRemote`, in which the mapping of the samplers numbering to component id is specified via the sampler listing in the XML file and the `metric node map`, as described below. These identifiers are then placed into the **RemoteSamplerLookup** table of the database by the `ovis-db` command and read by the Sheep process running the remote sampler at startup. The author of such a sampler need only keep track that that when recording a data value via the `RecordXXX` function, that the correct metric number (consistent with that specified in the `metric node map` is used.

Use of the `metric node map` requires that the remote component mappings be done at the time the XML file is loaded into the database. In the `MetricCollectionSamplerDynamicKey` class, mappings can be determined at runtime, thus obviating the need to specify a `metric node map` in the XML file. This class creates the associations between the sampler's metric numbering and the intended CompId for a given metric numbering at runtime. This capability is particularly useful for cases where the number of possible associations is large, but the number of actual instances is small or for writing a single sampler to be used in either local or remote mode. If the sampler is to be used as local, then the remote CompId assigned is actually the local CompId.

A common case of this is addressed by the `ovMetricCollectionSamplerAssociation` which handles the case where a particular metric may be collected on behalf of the local component or on behalf of a child (physical) of the local component. For instance, for a node, values of CPU utilization can potentially either be associated with the cores or with the nodes. Arguments to the sampler are then used to determine the child and instance information with which a given metric should be associated. These are described more in Section 5.2. A map is then created which gives the correct metric id that should be used a given metric in the `RecordXXX` function call. The `ovMetricLinuxlmsensorsSampler` is such a sampler and should be consulted as an example. Note that to use this sampler for your own setup, you do not have to change the sampler, but only change the metric names and assocations in your XML file specifying the sampler details. This is described further in Section 5.2.

Note that the table names may have to be different if a sampler is written to sampler for local vs remote components. For example if the core metrics are to be associated with remote components (CPU), then the same metric names can be used for each CPU (e.g., CPUTemp); if the CPU metrics are all to be associated with local components (cn), then the metric names must be different for each CPU so as not to conflict with one another (e.g., CPU0_Temp, CPU1_Temp).

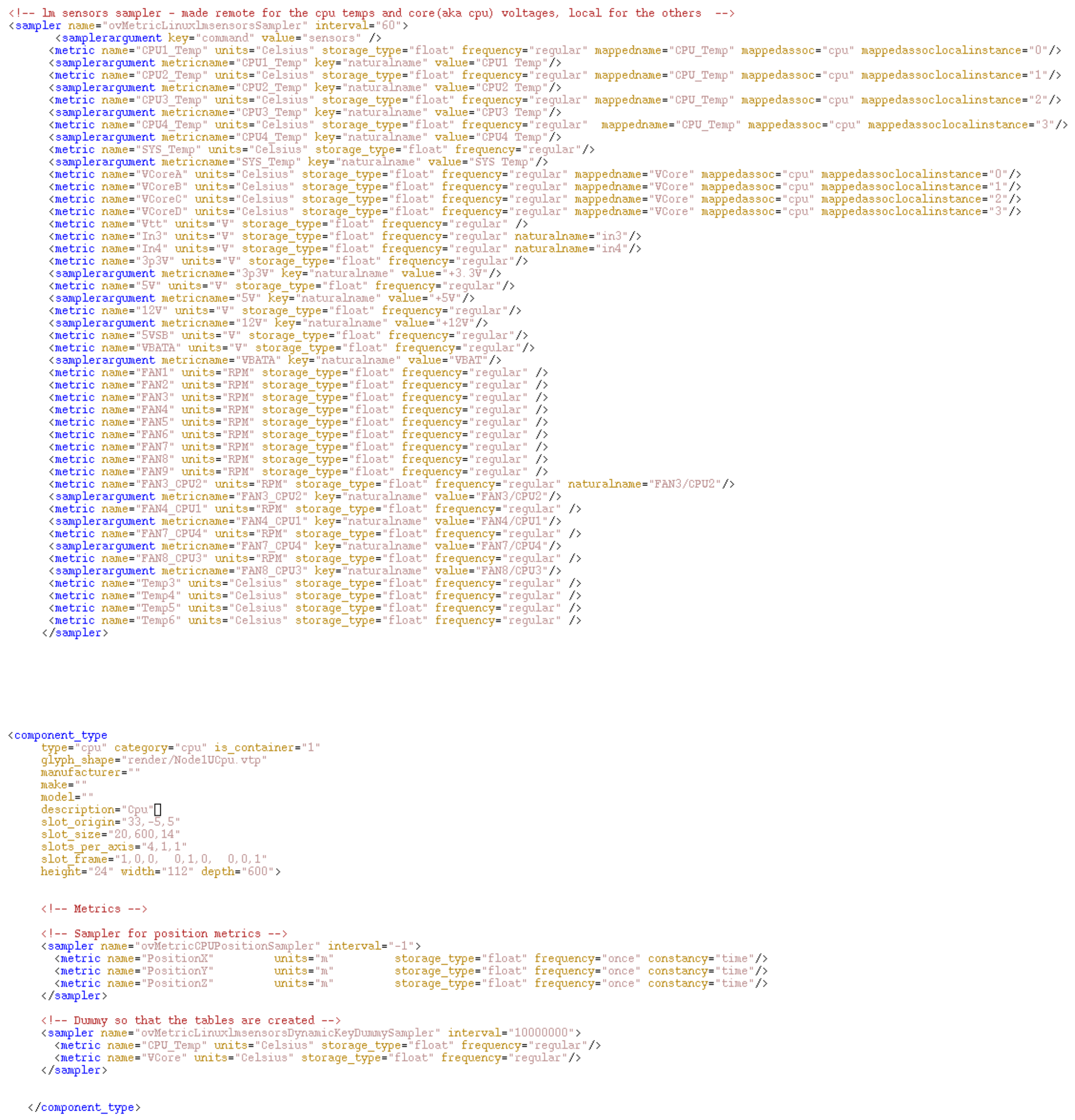
Samplers can also be passed arguments via the XML file that can be read in as described in Section 5.2.

Samplers and the XML File

Sampler details are specified in the XML file. This includes which samplers will be run on which components, their rate of collection, which metrics will be collected by each sampler, and optional sampler arguments. In each sampler definition, each metric must be identified by its name, the frequency of that metric's insertion into the database (the stride), and the metric value's data type (e.g., `ovis::INT`), as illustrated in Figure 8.

The XML file is read in by the database effector in order to set up the database tables. Each metric table required to exist for a given metric on a given component type must be specified via metric identification information in the `samplers` section for a given component type. Each metric must be associated with a particular sampler. Some slight of hand then is done to enable remote component mappings.

In this example, when this XML file is read tables **MetricCnCPU1_TempValues**, **MetricCnCPU2_TempValues**, **MetricCnSYS_TempValues** etc will be created (with associated information about the samplers specified as described previously) and sampler



```xml
<!-- lm sensors sampler - made remote for the cpu temps and core(aka cpu) voltages, local for the others  -->
<sampler name="ovMetricLinuxlmsensorsSampler" interval="60">
      <samplerargument key="command" value="sensors" />
     <metric name="CPU1_Temp" units="Celsius" storage_type="float" frequency="regular" mappedname="CPU_Temp" mappedassoc="cpu" mappedassoclocalinstance="0"/>
     <samplerargument metricname="CPU1_Temp" key="naturalname" value="CPU1 Temp"/>
     <metric name="CPU2_Temp" units="Celsius" storage_type="float" frequency="regular" mappedname="CPU_Temp" mappedassoc="cpu" mappedassoclocalinstance="1"/>
     <samplerargument metricname="CPU2_Temp" key="naturalname" value="CPU2 Temp"/>
     <metric name="CPU3_Temp" units="Celsius" storage_type="float" frequency="regular" mappedname="CPU_Temp" mappedassoc="cpu" mappedassoclocalinstance="2"/>
     <samplerargument metricname="CPU3_Temp" key="naturalname" value="CPU3 Temp"/>
     <metric name="CPU4_Temp" units="Celsius" storage_type="float" frequency="regular"  mappedname="CPU_Temp" mappedassoc="cpu" mappedassoclocalinstance="3"/>
     <samplerargument metricname="CPU4_Temp" key="naturalname" value="CPU4 Temp"/>
     <metric name="SYS_Temp" units="Celsius" storage_type="float" frequency="regular"/>
     <samplerargument metricname="SYS_Temp" key="naturalname" value="SYS Temp"/>
     <metric name="VCoreA" units="Celsius" storage_type="float" frequency="regular" mappedname="VCore" mappedassoc="cpu" mappedassoclocalinstance="0"/>
     <metric name="VCoreB" units="Celsius" storage_type="float" frequency="regular" mappedname="VCore" mappedassoc="cpu" mappedassoclocalinstance="1"/>
     <metric name="VCoreC" units="Celsius" storage_type="float" frequency="regular" mappedname="VCore" mappedassoc="cpu" mappedassoclocalinstance="2"/>
     <metric name="VCoreD" units="Celsius" storage_type="float" frequency="regular" mappedname="VCore" mappedassoc="cpu" mappedassoclocalinstance="3"/>
     <metric name="Vtt" units="V" storage_type="float" frequency="regular" />
     <metric name="In3" units="V" storage_type="float" frequency="regular" naturalname="in3"/>
     <metric name="In4" units="V" storage_type="float" frequency="regular" naturalname="in4"/>
     <metric name="3p3V" units="V" storage_type="float" frequency="regular"/>
     <samplerargument metricname="3p3V" key="naturalname" value="+3.3V"/>
     <metric name="5V" units="V" storage_type="float" frequency="regular"/>
     <samplerargument metricname="5V" key="naturalname" value="+5V"/>
     <metric name="12V" units="V" storage_type="float" frequency="regular"/>
     <samplerargument metricname="12V" key="naturalname" value="+12V"/>
     <metric name="5VSB" units="V" storage_type="float" frequency="regular"/>
     <metric name="VBATA" units="V" storage_type="float" frequency="regular"/>
     <samplerargument metricname="VBATA" key="naturalname" value="VBAT"/>
     <metric name="FAN1" units="RPM" storage_type="float" frequency="regular" />
     <metric name="FAN2" units="RPM" storage_type="float" frequency="regular" />
     <metric name="FAN3" units="RPM" storage_type="float" frequency="regular" />
     <metric name="FAN4" units="RPM" storage_type="float" frequency="regular" />
     <metric name="FAN5" units="RPM" storage_type="float" frequency="regular" />
     <metric name="FAN6" units="RPM" storage_type="float" frequency="regular" />
     <metric name="FAN7" units="RPM" storage_type="float" frequency="regular" />
     <metric name="FAN8" units="RPM" storage_type="float" frequency="regular" />
     <metric name="FAN9" units="RPM" storage_type="float" frequency="regular" />
     <metric name="FAN3_CPU2" units="RPM" storage_type="float" frequency="regular" naturalname="FAN3/CPU2"/>
     <samplerargument metricname="FAN3_CPU2" key="naturalname" value="FAN3/CPU2"/>
     <metric name="FAN4_CPU1" units="RPM" storage_type="float" frequency="regular" />
     <samplerargument metricname="FAN4_CPU1" key="naturalname" value="FAN4/CPU1"/>
     <metric name="FAN7_CPU4" units="RPM" storage_type="float" frequency="regular" />
     <samplerargument metricname="FAN7_CPU4" key="naturalname" value="FAN7/CPU4"/>
     <metric name="FAN8_CPU3" units="RPM" storage_type="float" frequency="regular" />
     <samplerargument metricname="FAN8_CPU3" key="naturalname" value="FAN8/CPU3"/>
     <metric name="Temp3" units="Celsius" storage_type="float" frequency="regular" />
     <metric name="Temp4" units="Celsius" storage_type="float" frequency="regular" />
     <metric name="Temp5" units="Celsius" storage_type="float" frequency="regular" />
     <metric name="Temp6" units="Celsius" storage_type="float" frequency="regular" />
    </sampler>
```

```xml
<component_type
     type="cpu" category="cpu" is_container="1"
     glyph_shape="render/Node1UCpu.vtp"
     manufacturer=""
     make=""
     model=""
     description="Cpu"
     slot_origin="33,-5,5"
     slot_size="20,600,14"
     slots_per_axis="4,1,1"
     slot_frame="1,0,0,   0,1,0,   0,0,1"
     height="24" width="112" depth="600">

     <!-- Metrics -->

     <!-- Sampler for position metrics -->
     <sampler name="ovMetricCPUPositionSampler" interval="-1">
       <metric name="PositionX"         units="m"         storage_type="float" frequency="once" constancy="time"/>
       <metric name="PositionY"         units="m"         storage_type="float" frequency="once" constancy="time"/>
       <metric name="PositionZ"         units="m"         storage_type="float" frequency="once" constancy="time"/>
     </sampler>

     <!-- Dummy so that the tables are created -->
     <sampler name="ovMetricLinuxlmsensorsDynamicKeyDummySampler" interval="10000000">
       <metric name="CPU_Temp" units="Celsius" storage_type="float" frequency="regular"/>
       <metric name="VCore" units="Celsius" storage_type="float" frequency="regular"/>
     </sampler>

   </component_type>
```

Figure 8: Excerpt from `testonecpu.ovdb` file for the `ovMetricLinuxlmsensorsSampler`, a subclass of `MetricCollectionSamplerAssociation` illustrating (1) specification of metric name, stride, data type etc, (2) specification of sampler arguments, and (3) association information. Top shows the specification in the `cn` component type; bottom shows the specification in the `cpu` type.

ovMetricLinuxlmsensorsSampler will attempt to be started on the nodes. Additionally, tables **MetricCpuCPUTempValues** will be created and sampler ovMetricLinuxlmsensorsDummySampler will attempt to be started on the CPUs. Since no sampler ovMetricLinuxlmsensorsDummySampler exists, that sampler will not be started, however, the necessary remote component data tables will be created. Since sampler ovMetricLinuxlmsensorsSampler does exist, it will be started, but since that sampler is configured (described in more detail below) to sample remotely, the tables **MetricCnCPU1_TempValues** will be created, but will not be used. The ovMetricLinuxlmsensorsSampler will read a value for CPU1_Temp but will insert it, with the proper component id of the remote component into the **MetricCpuCPU_Temp**. Note that local associations require no such subterfuge, with SYS_Temp being collected by this sampler and recorded for the local component.

For the particular case of the ovMetricCollectionSamplerAssociation, some individual metric arguments are defined to enable the determination of remote associations: these are:

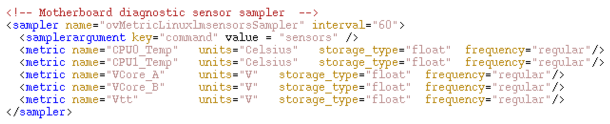
- mappedname - the name of the metric on the remote instance
- mappedassoc - the (child) association of the remote instance.
- mappedassoclocalinstance - an identifier indicating which particular child this value should be associated with

This information is stored in the **RemoteSamplerHints** table.

Currently the type has to be in a child (direct or indirect) relationship as specified in the physical association section of the XML file (e.g., cores are children of CPUs are children of nodes), with the instance being the local instance (e.g., instance 2 means the 2nd child of that type for a particular node).

In the example in Figure 8, when the sampler on a particular node reads the sensors value for CPU1_Temp, it will then actually record that value into the **MetricCpuCPU_TempValues** table, with the CompId corresponding to that node's 0th (first) CPU. This is done via the mapping which is then used in the RecordXXX function.

For this sampler, the metrics will default to local if no mapping information is given as in Figure 9.



```
<!-- Motherboard diagnostic sensor sampler  -->
<sampler name="ovMetricLinuxlmsensorsSampler" interval="60">
  <samplerargument key="command" value = "sensors" />
  <metric name="CPU0_Temp"   units="Celsius"   storage_type="float"  frequency="regular"/>
  <metric name="CPU1_Temp"   units="Celsius"   storage_type="float"  frequency="regular"/>
  <metric name="VCore_A"     units="V"   storage_type="float"  frequency="regular"/>
  <metric name="VCore_B"     units="V"   storage_type="float"  frequency="regular"/>
  <metric name="Vtt"         units="V"   storage_type="float"  frequency="regular"/>
</sampler>
```

Figure 9: Excerpt from testone.ovdb file for the ovMetricLinuxlmsensorsSampler, a subclass of ovMetricCollectionSamplerAssociation when all metrics are associated with the local (cn) component.

A more general mapping specification using the `metric node map` is shown in Figures 10 and 11.

In Figure 10 the sampler specification for the Terascala chassis is a dummy, with the sampler specification on the Terascala admin node being the actual remote sampler that will be instantiated. The metric node map in the *associations* section shown in Figure 11 associates the unique metric numberings of the real sampler running on the Terascala admin node with the particular Terascala component (e.g., Chassis or Blade) to which those metrics pertain. For instance, in Figure 11 metrics 0-6 (the unique identifiers in the remote sampler) in the `ovMetricTerascalaSNMPChassisSampler` running on the first instance of type tsadmin (which is the only admin node) are actually values for the first chassis, tsnode 1 (the component types and number associations are in the *instances* and *associations* sections as described earlier with respect to the Whitney XML file). The order of the metrics (e.g., which metric is 0) is determined as previously described. The information specified in the `metric node map` is stored in the **RemoteSamplerLookup** table.

Key-value pairs for sampler arguments can be specified in the XML file via `sampler argument`. These are stored in the **MetricCollectionSamplersArguments** table and can be extracted by a Sampler at run time via calls such as

```
this->GetSamplerArgument("naturalname")
```

as in the case of Figure 8.

This call cannot be made in the Sampler constructor, but can be made in the `CanSamplerRun()` function (which is only called once) or the `Sample()` function (which is called multiple times).

The samplers are listed in the **MetricCollectionSamplers** table, along with information regarding upon which component type they sample on behalf of and on what interval they sample. The interval is determined at setup time from information in the XML file.

There is currently no mechanism to dynamically add in a sampler once the OVIS database has been instantiated, as the tables for the metric data are set up at that time.

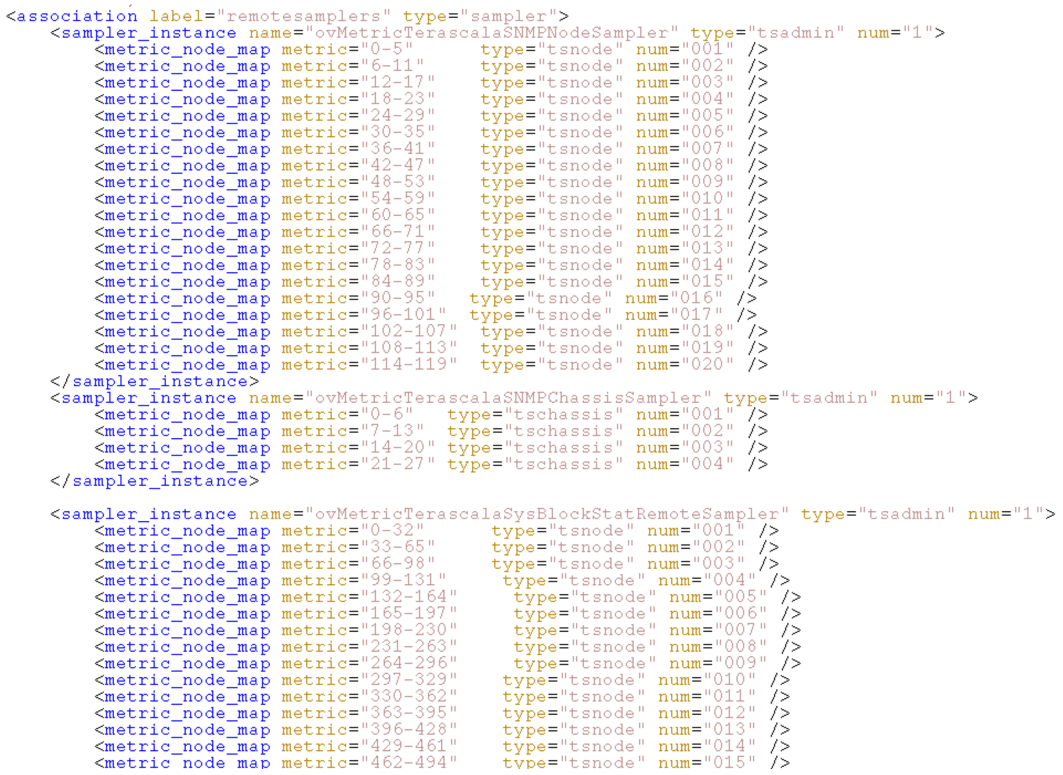
Threads

In samplers that collect many metrics from various sources or that collect on behalf of many remote components, it may be more time efficient to perform the collection using multiple threads. OVIS provides the `ovThreadPool` class to create and manage multiple threads. `ovMetricTalonIPMIToolSampler` is an example sampler illustrating use of the `ovThreadPool`. The Sampler contains a number of `Tasks`, each responsible for sampling some subset of the metrics; The Sampler's `Sample` command calls `Execute` on the `ThreadPool` for a particular `Task`. Each `Task` will then sample data relevant to a given remote node. The `Execute` interface takes the identity of the `Task` and the `Task`'s function to be executed (typically called `Sample`). It also takes a `time to live` for the `Task` after which the task will be killed. In addition, it takes a comparison function that is used to eliminate

```
<component_type
  type="tschassis" category="blade chassis" is_container="1"
  glyph_shape="render/ts_chassis.vtp"
  manufacturer="Terascala"
  make=""
  model=""
  description="Terascala chassis using Force10 chassis description"
  slot_origin="35,-10,11"
  slot_size="88,600,352"
  slots_per_axis="8,1,1"
  slot_frame="1,0,0,  0,1,0,  0,0,1"
  height="352" width="530" depth="700">

  <!-- Dummy sampler for some dummy metrics for these dumb chassis  -->
  <sampler name="ovMetricSwitchInfoSampler" interval="-1">
    <metric name="Position"          units="m"         storage_type="posn3" frequency="once"/>
    <metric name="Orientation"       units="m"         storage_type="vec3"  frequency="once"/>
  </sampler>

  <!-- SNMP Chassis sampler (dummy so that tables are
  created. this is actually running on y node) -->
    <sampler name="ovMetricTerascalaSNMPChassisDummySampler" interval="10000">
      <metric name="FanSpeed1"   units="RPM"   storage_type="int"   frequency="regular"/>
      <metric name="FanSpeed2"   units="RPM"   storage_type="int"   frequency="regular"/>
      <metric name="FanSpeed3"   units="RPM"   storage_type="int"   frequency="regular"/>
      <metric name="FanSpeed4"   units="RPM"   storage_type="int"   frequency="regular"/>
      <metric name="FanSpeed5"   units="RPM"   storage_type="int"   frequency="regular"/>
      <metric name="FanSpeed6"   units="RPM"   storage_type="int"   frequency="regular"/>
      <metric name="Temp"        units="Celsius"   storage_type="int"   frequency="regular"/>
    </sampler>

</component_type>
```

```
<component_type
  type="tsadmin" category="node" is_container="0"
  manufacturer="Dell"
  make="PowerEdge"
  model="1850"
  description="terascala admin node"
  width="550" depth="600" height="44">

<!-- ** Metrics ** -->

  <!-- Just a signal that the machine is alive -->
  <sampler name="ovMetricHeartbeat" interval="10000">
    <metric name="Heartbeat" units="1" storage_type="none" frequency="regular"/>
  </sampler>


  <!-- SNMP Node sampler (remote sampler for the nodes) -->
    <sampler name="ovMetricTerascalaSNMPNodeSampler" interval="30">
      <metric name="CPU1CoreTemp"   units="Celsius"   storage_type="int"   frequency="regular"/>
      <metric name="AmbientTemp"   units="Celsius"   storage_type="int"   frequency="regular"/>
      <metric name="BladeTemp1"   units="Celsius"   storage_type="int"   frequency="regular"/>
      <metric name="BladeTemp2"   units="Celsius"   storage_type="int"   frequency="regular"/>
      <metric name="BladeTemp3"   units="Celsius"   storage_type="int"   frequency="regular"/>
      <metric name="BladeChassisTemp"   units="Celsius"   storage_type="int"   frequency="regular"/>
    </sampler>


  <!-- SNMP Chassis sampler (remote sampler for the chassis) -->
    <sampler name="ovMetricTerascalaSNMPChassisSampler" interval="30">
      <metric name="FanSpeed1"   units="RPM"   storage_type="int"   frequency="regular"/>
      <metric name="FanSpeed2"   units="RPM"   storage_type="int"   frequency="regular"/>
      <metric name="FanSpeed3"   units="RPM"   storage_type="int"   frequency="regular"/>
      <metric name="FanSpeed4"   units="RPM"   storage_type="int"   frequency="regular"/>
      <metric name="FanSpeed5"   units="RPM"   storage_type="int"   frequency="regular"/>
      <metric name="FanSpeed6"   units="RPM"   storage_type="int"   frequency="regular"/>
      <metric name="Temp"        units="Celsius"   storage_type="int"   frequency="regular"/>
    </sampler>


    <!-- SysBlockStat (now remotely for the nodes) -->
  <sampler name="ovMetricTerascalaSysBlockStatRemoteSampler" interval="100000">
    <metric name="sda_READ_IOs"                          units="1"   storage_type="float"      frequency="regular"/>
    <metric name="sda_READ_MERGES"                          units="1"   storage_type="float"      frequency="regular"/>
    <metric name="sda_READ_SECTORS"                          units="1"   storage_type="float"      frequency="regular"/>
    <metric name="sda_READ_TICKS"                          units="1"   storage_type="float"      frequency="regular"/>
    <metric name="sda_WRITE_IOs"                          units="1"   storage_type="float"      frequency="regular"/>
    <metric name="sda_WRITE_MERGES"                          units="1"   storage_type="float"      frequency="regular"/>
    <metric name="sda_WRITE_SECTORS"                          units="1"   storage_type="float"      frequency="regular"/>
    <metric name="sda_WRITE_TICKS"                          units="1"   storage_type="float"      frequency="regular"/>
```

Figure 10: Excerpts from the Whitney-Terascala XML file with partial specification of the remote samplers. The samplers (top) given for the component to which the data pertains, in this case the Chassis, are dummy and are just used to establish the metric tables. The samplers (bottom) for the component that will actually do the database insertion are real.



```
<association label="remotesamplers" type="sampler">
    <sampler_instance name="ovMetricTerascalaSNMPNodeSampler" type="tsadmin" num="1">
        <metric_node_map metric="0-5"      type="tsnode" num="001" />
        <metric_node_map metric="6-11"     type="tsnode" num="002" />
        <metric_node_map metric="12-17"    type="tsnode" num="003" />
        <metric_node_map metric="18-23"    type="tsnode" num="004" />
        <metric_node_map metric="24-29"    type="tsnode" num="005" />
        <metric_node_map metric="30-35"    type="tsnode" num="006" />
        <metric_node_map metric="36-41"    type="tsnode" num="007" />
        <metric_node_map metric="42-47"    type="tsnode" num="008" />
        <metric_node_map metric="48-53"    type="tsnode" num="009" />
        <metric_node_map metric="54-59"    type="tsnode" num="010" />
        <metric_node_map metric="60-65"    type="tsnode" num="011" />
        <metric_node_map metric="66-71"    type="tsnode" num="012" />
        <metric_node_map metric="72-77"    type="tsnode" num="013" />
        <metric_node_map metric="78-83"    type="tsnode" num="014" />
        <metric_node_map metric="84-89"    type="tsnode" num="015" />
        <metric_node_map metric="90-95"   type="tsnode" num="016" />
        <metric_node_map metric="96-101"  type="tsnode" num="017" />
        <metric_node_map metric="102-107"  type="tsnode" num="018" />
        <metric_node_map metric="108-113"  type="tsnode" num="019" />
        <metric_node_map metric="114-119"  type="tsnode" num="020" />
    </sampler_instance>
    <sampler_instance name="ovMetricTerascalaSNMPChassisSampler" type="tsadmin" num="1">
        <metric_node_map metric="0-6"   type="tschassis" num="001" />
        <metric_node_map metric="7-13"  type="tschassis" num="002" />
        <metric_node_map metric="14-20" type="tschassis" num="003" />
        <metric_node_map metric="21-27" type="tschassis" num="004" />
    </sampler_instance>

    <sampler_instance name="ovMetricTerascalaSysBlockStatRemoteSampler" type="tsadmin" num="1">
        <metric_node_map metric="0-32"      type="tsnode" num="001" />
        <metric_node_map metric="33-65"     type="tsnode" num="002" />
        <metric_node_map metric="66-98"     type="tsnode" num="003" />
        <metric_node_map metric="99-131"     type="tsnode" num="004" />
        <metric_node_map metric="132-164"     type="tsnode" num="005" />
        <metric_node_map metric="165-197"     type="tsnode" num="006" />
        <metric_node_map metric="198-230"     type="tsnode" num="007" />
        <metric_node_map metric="231-263"     type="tsnode" num="008" />
        <metric_node_map metric="264-296"     type="tsnode" num="009" />
        <metric_node_map metric="297-329"    type="tsnode" num="010" />
        <metric_node_map metric="330-362"    type="tsnode" num="011" />
        <metric_node_map metric="363-395"    type="tsnode" num="012" />
        <metric_node_map metric="396-428"    type="tsnode" num="013" />
        <metric_node_map metric="429-461"    type="tsnode" num="014" />
        <metric_node_map metric="462-494"    type="tsnode" num="015" />
```

Figure 11: Excerpt from Whitney-Terascala XML file showing the `metric node maps` which associate remote sampler metric numbering and the corresponding components.

duplicate `Tasks` in the `Thread Pool`'s queue of tasks. In this way, a second instance of a `Task` collecting for a resource will not be put into the queue until the previous one is out of the queue. This will keep many multiples of the same `Task` from building up in the queue.

Notes on Specific Samplers

- `ovMetricLinuxlmsensorsSampler`

In the storage case described in a previous section, the use of a remote sampler was required as data could only be obtained from the admin node. However, there are cases where both local and remote samplers are possibilities. This sampler can function either locally or remotely, on a per-metric basis, depending on whether or not mapping information is specified for each metric. This depends, of course on the fidelity of the display, depending on whether only the nodes are drawn or if the cores are drawn as well.

To adapt this sampler for your system, change the metric names to those of your system and component types to those of your system. Also change the `sampler arguments` for `naturalnames` – these are used to match the label name obtained from the sensors call the actual metric name; these are necessary because the label name may have a space which is not allowable for table names.

- `ovMetricLinuxProcStatUtilSpecialtySampler`

The data source `/proc/stat` contains overall node CPU utilization as well as core level utilization, storing data in tables **MetricCnPercentUserCnViewValues** or **MetricCorePercentUserCnView-Values** respectively. The `CnView` in the name indicates that the value was obtained from reading `/proc/stat` on the node itself.

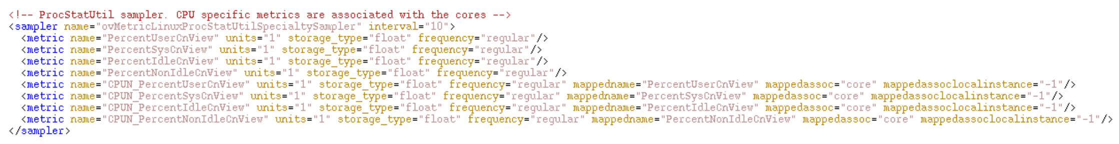
The use of the word *specialty* in the sampler name refers to a particular implementation detail used in this sampler. In this case, when `mappedassoclocalinstance` value -1 is used, the sampler interprets that to mean to record a value of PercentUser for every child of type *core* that exists for this node. The sampler determines that based on the number of return values it gets from its data gathering call. The specification is shown in Figure 12

To adapt this sampler for your system, check the component type names.

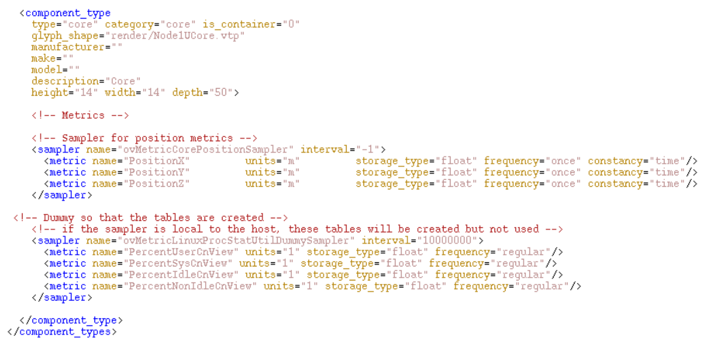
Testing Samplers

You can test samplers (both local and remote) via the following:

```
./bin/ovisTests testSampler -sampler mySampler interval iterations
```



```
<!-- ProcStatUtil sampler. CPU specific metrics are associated with the cores -->
<sampler name="ovMetricLinuxProcStatUtilSpecialtySampler" interval="10">
  <metric name="PercentUserCnView" units="1" storage_type="float" frequency="regular"/>
  <metric name="PercentSysCnView" units="1" storage_type="float" frequency="regular"/>
  <metric name="PercentIdleCnView" units="1" storage_type="float" frequency="regular"/>
  <metric name="PercentNonIdleCnView" units="1" storage_type="float" frequency="regular"/>
  <metric name="CPUN_PercentUserCnView" units="1" storage_type="float" frequency="regular" mappedname="PercentUserCnView" mappedassoc="core" mappedassoclocalinstance="-1"/>
  <metric name="CPUN_PercentSysCnView" units="1" storage_type="float" frequency="regular" mappedname="PercentSysCnView" mappedassoc="core" mappedassoclocalinstance="-1"/>
  <metric name="CPUN_PercentIdleCnView" units="1" storage_type="float" frequency="regular" mappedname="PercentIdleCnView" mappedassoc="core" mappedassoclocalinstance="-1"/>
  <metric name="CPUN_PercentNonIdleCnView" units="1" storage_type="float" frequency="regular" mappedname="PercentNonIdleCnView" mappedassoc="core" mappedassoclocalinstance="-1"/>
</sampler>
```



```
 <component_type
  type="core" category="core" is_container="0"
  glyph_shape="render/Node1UCore.vtp"
  manufacturer=""
  make=""
  model=""
  description="Core"
  height="14" width="14" depth="50">

  <!-- Metrics -->

  <!-- Sampler for position metrics -->
  <sampler name="ovMetricCorePositionSampler" interval="-1">
    <metric name="PositionX"        units="m"        storage_type="float" frequency="once" constancy="time"/>
    <metric name="PositionY"        units="m"        storage_type="float" frequency="once" constancy="time"/>
    <metric name="PositionZ"        units="m"        storage_type="float" frequency="once" constancy="time"/>
  </sampler>

<!-- Dummy so that the tables are created -->
  <!-- if the sampler is local to the host, these tables will be created but not used -->
  <sampler name="ovMetricLinuxProcStatUtilDummySampler" interval="10000000">
    <metric name="PercentUserCnView" units="1" storage_type="float" frequency="regular"/>
    <metric name="PercentSysCnView" units="1" storage_type="float" frequency="regular"/>
    <metric name="PercentIdleCnView" units="1" storage_type="float" frequency="regular"/>
    <metric name="PercentNonIdleCnView" units="1" storage_type="float" frequency="regular"/>
  </sampler>

 </component_type>
</component_types>
```

Figure 12: Excerpt from `testonecpu.ovdb` file for the `ovProcStatUtilSampler`, illustrating the specfication in the `cn` (node) component type (top), and in the `core` type (bottom).

where mySampler is the sampler to be tested and interval is the interval between iterations of the test. The test does not utilize the full machinery of OVIS, but instead uses a test class `ovTestMetricRecorder` in the test directory that writes the values to stdout and not to a database. The `ovTestMetricRecorder` does not have access to a database for reading either, so values for items with may be requested from `ovMetricRecorder` such as `CompIds` or `CompTypes` have either been filled in with fake return values or they will have to be added.

Using the `ovisTests` functionality in order to test samplers that take advantage of this feature, requires altering the `ovTestMetricRecorder` to return the appropriate values, since the `ovTestMetricRecorder` is not associated with an actual database.

5.3 Direct Insertion

Data can be inserted into the OVIS database directly. In this case, you should fill in the metric table metadata as described in Section 5.1. Note that the lack of normal information, such as the relationship of a metric with a particular sampler can result it making it impossible to associate metric information with its source or type and can possibly cause confusion for metrics added via methods such as the `Metric Generator` in post processing.

5.4 Metric Generator

The `Metric Generator` makes it easy to insert data into the database directly via a script. The Metric Generator handles the details of creating the metadata as described in Section 5.1 and invokes a user-created script that reads in the relevatn data, generates new data, and inserts it into the database. Example scripts come with the OVIS release in the `scripts` directory an can be used as examples, particularly for the data gathering and insertion operations.

Currently the Metric Generator works only in post processing mode, where the script is run on an existing set up data, rather than dynamically calculating the derived quantities and inserting them as new raw data comes in.

Use of the `Metric Generator` using the `Baron` will be discussed in Section 8.

6 Job Data

OVIS 3 has a preliminary implementation of an interface for job exploration. It allows one to search for jobs based on attributes such as time, date, components, end state, and user name. It also displays statistics of jobs, both numerically and in an interactive pie chart. Finally, support for transferring lists of components associated with a particular job exists for the physical and analysis views. More information on how to use the job search capability is given in Section 9.1. This purpose of this section is to describe the database requirements of the job search capability.

Currently, OVIS provides support for the SLURM [1] database schema, as implemented on the TLCC clusters at Sandia. It expects a database named `slurm`, with one table of interest, `slurm_job_log`; and a database `slurm_acct_db` with one table of interest, `job_table`. These tables contain information on job ids, start and end times, user names etc. Complete descriptions of these tables can be found by looking at the example data set that comes with the OVIS release.

OVIS creates an additional table called `ComponentTable_slurm_job_log`. in its database that holds raw numerical data This table contains associations of the `job_ids` with the OVIS `Component IDs`. The OVIS release contains an executable `slurm_job_log_importer` which can build this table.

The next version of the job search capability will incorporate a more generic interface to support other schedulers/resource managers and will not require the `ComponentTable_slurm_job_log` table.

7 Haruspices

In Roman practice inherited from the Etruscans, a *haruspex* (plural *haruspices*) was a man trained to practice a form of divination called *haruspicy*, the inspection of the entrails of sacrificed animals, especially the livers of sacrificed Sheep.

In this section, we first provide an overview of the analysis engines, which are called haruspices in OVIS parlance. We subsequently illustrate the utilization of these haruspices by providing an application of the multi-correlative haruspex; for more context about this application example, please refer to [7].

7.1 Overview

Definition 7.1. A OVIS haruspex is a process that:

1. is triggered by the Baron,
2. runs one on or multiple Shepherd nodes,
3. executes an analytical engine.

In particular, haruspices rely on specific tables of the underlying OVIS 3.2 database.

The OVIS 3.2 release supports several haruspecies in learn mode: Descriptive, Bi-variate and Multi-variate Correlative, 2-variable Contingency Statistics, and Time-series. The initial OVIS 3.X release does not support the monitor mode of operation, in which data is compared to the model determined by learn mode, which was present in the OVIS 2.X release; monitor mode will be re-integrated into a later OVIS 3.X release. More on this mode can be found in [6].

More detail on the OVIS 3.2 haruspecies can be found in [8]; the parameters and output values for the learn modes are copied here for convenience.

Descriptive Statistics

The primary model of a descriptive analysis contains the following statistics in a single table:

Cardinality: number of observations in the data set

Variable: name of metric

Minimum: minimum

Maximum: maximum

Mean: sample mean

M2: sum of quadratic deviations from the mean

M3: sum of cubic (absolute) deviations from the mean

M4: sum of quartic deviations from the mean

Correlative Statistics

The primary model of a correlative analysis contains the following statistics in a single table:

Cardinality: number of observations in the data set

Variable X: name of first metric

Variable Y: name of second metric

Mean X: mean of first metric

Mean Y: mean of second metric

M2 X: sum of quadratic deviations from the mean for first metric

M2 Y: sum of quadratic deviations from the mean for second metric

M XY: sum of crossed deviations from their respective means for both metrics

Contingency and Information Statistics

The primary model of a contingency and information analysis contains the following meta information in a first table:

Variable X: Name of first metric

Variable Y: Name of second metric

This meta information tables provides keys (specifically, the row number of a given X,Y pair is the key) into a second table which provides the actual statistics. This is to save storage space (to avoid multiple storages of pairs of string of arbitrary length) and facilitate retrieval in subsequent stages. This second table contains the following statistics:

Key: Key of the X,Y pair

x: Value of first metric

y: Value of second metric

Cardinality: Number of occurrences of observation (x,y) for metric pair with corresponding key

Note that the first row of this second table is also always reserved for use by the derive and assess phases, and is not filled with valid values during a learn phase: to make this explicit, a value of -1 is stored for the cardinality in this first row.

Multi-Correlative Statistics

The primary model of a multi-correlative analysis contains a table with the following columns:

Column1: The name of the first metric (except for the first row, which contains the reserved name "Cardinality")

Column2: The name of the second metric when an entry is present

Value: The stored value, equal to:

- the cardinality of the data set when Column1 is equal to "Cardinality",
- the mean of the metric with name stored in Column1 when it is not equal to "Cardinality" and when no entry is present in Column2,
- the sum of crossed deviations from their respective means for both metrics with names stored in Column1 and Column2, which amounts to the sum of squared deviations from the mean of the metric when the name is the same in both columns.

Multi-Correlative Time Series Statistics

The primary model of a multi-correlative time series analysis contains a table with the following columns:

Frequency: The Fourier mode index

Column1: The name of the first metric (except for the first row, which contains the reserved name "Cardinality")

Column2: The name of the second metric when an entry is present

ReEntries and **ImEntries:** The real and imaginary parts of a stored value, equal to:

- a cardinality measure of the data set, defined as $\#\text{windows} + i\#\text{observations}$, when Column1 is equal to "Cardinality";

- the mean of the metric with name stored in Column1 when it is not equal to "Cardinality" and when no entry is present in Column2;

- the sum of crossed deviations from their respective means for the Fourier transforms of both metrics with names stored in Column1 and (with complex conjugation) Column2, which amounts to the sum of absolute squared deviations from the mean of the Fourier transform of the metric when the name is the same in both columns.

7.2 Haruspex Calculation

Details of the Haruspex tables are beyond the scope of this document; see [9] for information on how analyses are implemented and how to extend OVIS to include your own analysis. However, users should be aware that a Haruspex analysis occurs in several phases: first, the shepherd into whose database a request is inserted copies the request into other shepherds' databases. Then, each shepherd independently summarizes the metric data in its own local database by running an analytical engine. When it completes the summary, it writes the resulting model to its own database and then copies this model to the other shepherds' databases. Each shepherd, upon receipt of a model from another shepherd, aggregates all the local models into a single global model. Each time an aggregation is done, the aggregate result is presented in the Baron, along with the identities of the shepherds contributing to the result. If several shepherds are involved in an analysis, the result presented in the baron may update several times during the course of an analysis, as more shepherds report. In this way, if any shepherds survive, their results are presented to the user regardless of shepherds that fail to complete the analysis.

There is one parameter relevant to the analysis that can be set in ccmake, as described in Section 2: `OVIS_SQL_FP_RTOL`. The `OVIS_SQL_FP_RTOL` is used as the tolerance in floating point comparisons used in SQL queries to fetch chunks of metric data to fetch at a time. By default this is set to a value we have found to work in practice. Should it fail, it is possible that a haruspex thread will hang looking for metric data it cannot find (but this is unlikely).

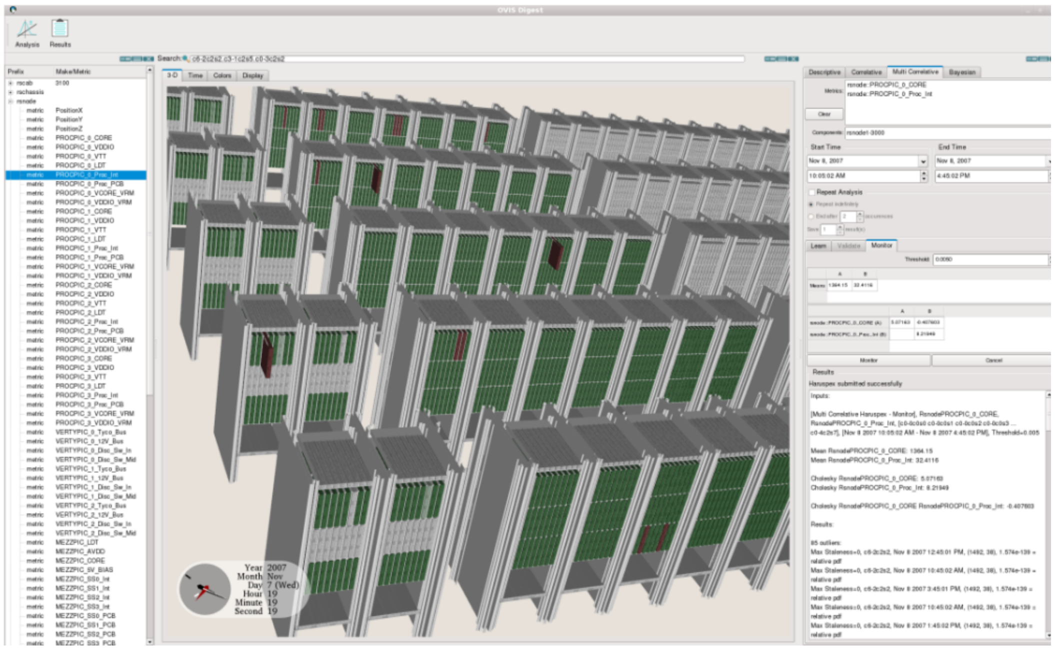
Certain calculations involve determining relationships between variables that occur as a set, for example in the multivariate correlations. In these calculations, if there is some time offset among instantiations of the raw data collected, interpolation of variable values will occur to bring data values and times in alignment. Currently, a default piecewise-constant interpolation is used. Therefore, data values in the database tables must be ordered monotonically increasing in time for the correct interpolation to be done. Minor variations in this will cause values to be dropped from the calculations or sub-optimal interpolations to occur, but a result will still be obtained. Given a generally large number of data values and generally small time between observations, it is expected that such variations will not have substantial effect on the analysis outcome.

7.3 Example: Multi-Correlative Haruspex

We now illustrate the use of OVIS 3.0 haruspices with an application of the multi-correlative haruspex to resource characterization. This example will also cover use of the monitor capability, present in OVIS 2, for context, though it is not yet implemented in OVIS 3.

In this approach, anomalous behaviors are sought by

1. calculating (with "training data") or devising (with "expert knowledge") mean vectors and covariance matrices – and thus, implicitly, a multiple linear regression model – for a set of tuples of variables of interest, and
2. examining how individual observations of these tuples of variables of interest deviate from the aforementioned model; such deviations are characterized in terms of the significance level to which they correspond when the mean vector and covariance matrix are made those of a multivariate Gaussian model. Note that this is directly related to the multivariate Mahalanobis distance computed with the mean vector and covariance matrix.



Metrics: rsnode::PROCPIC_0_CORE
rsnode::PROCPIC_0_Proc_Int

Components: rsnode1-3000

Start Time: Nov 8, 2007 10:05:02 AM
End Time: Nov 8, 2007 4:45:02 PM

Repeat Analysis

Learn | Validate | Monitor

Threshold: 0.0050

	A	B
Means	1364.15	32.4116

	A	B
rsnode::PROCPIC_0_CORE (A)	5.07163	-0.407603
rsnode::PROCPIC_0_Proc_Int (B)		8.21949

Monitor | Cancel

Results

Haruspex submitted successfully

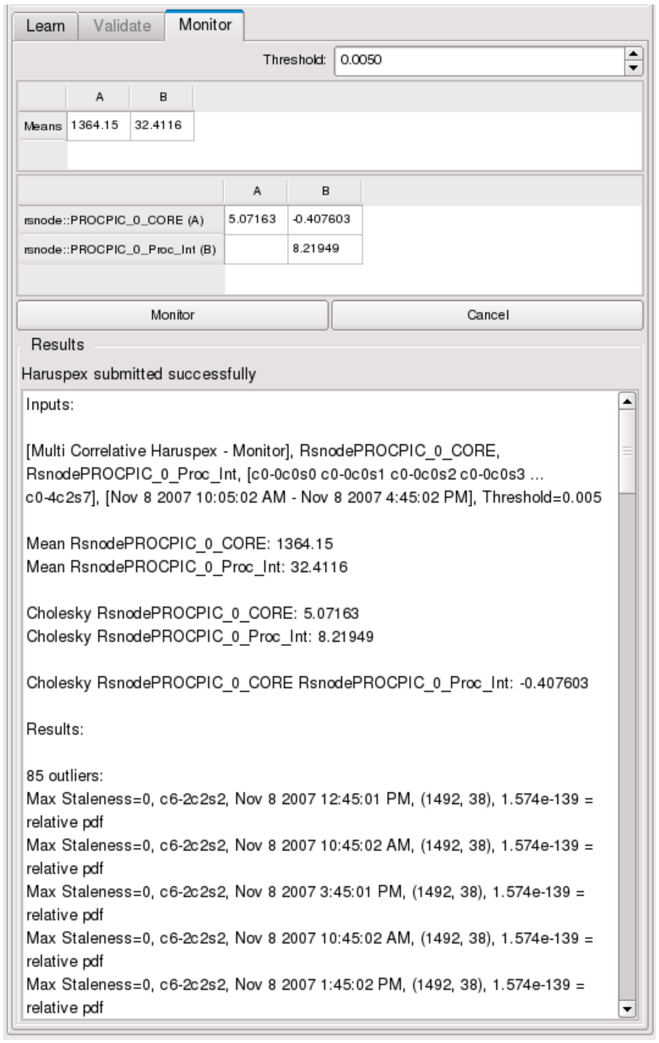
```
Inputs:

[Multi Correlative Haruspex - Monitor], RsnodePROCPIC_0_CORE,
RsnodePROCPIC_0_Proc_Int, [c0-0c0s0 c0-0c0s1 c0-0c0s2 c0-0c0s3 ...
c0-4c2s7], [Nov 8 2007 10:05:02 AM - Nov 8 2007 4:45:02 PM], Threshold=0.005

Mean RsnodePROCPIC_0_CORE: 1364.15
Mean RsnodePROCPIC_0_Proc_Int: 32.4116

Cholesky RsnodePROCPIC_0_CORE: 5.07163
Cholesky RsnodePROCPIC_0_Proc_Int: 8.21949

Cholesky RsnodePROCPIC_0_CORE RsnodePROCPIC_0_Proc_Int: -0.407603

Results:

85 outliers:
Max Staleness=0, c6-2c2s2, Nov 8 2007 12:45:01 PM, (1492, 38), 1.574e-139 = relative pdf
Max Staleness=0, c6-2c2s2, Nov 8 2007 10:45:02 AM, (1492, 38), 1.574e-139 = relative pdf
Max Staleness=0, c6-2c2s2, Nov 8 2007 3:45:01 PM, (1492, 38), 1.574e-139 = relative pdf
Max Staleness=0, c6-2c2s2, Nov 8 2007 10:45:02 AM, (1492, 38), 1.574e-139 = relative pdf
Max Staleness=0, c6-2c2s2, Nov 8 2007 1:45:02 PM, (1492, 38), 1.574e-139 = relative pdf
```

Figure 13: Actual rendering of the Red Storm platform zoomed in on the partition on which data were taken. The nodes are colored red if below the user-defined probabilistic threshold for being too unreliable and green otherwise. Grey indicates there was no data in the display time window for that resource for the metric being displayed.

For instance, Figure 13 displays a simple use case where only one pair of variables is of interest to the analyst, namely PROCPIC_0_CORE and PROCPIC_0_Proc_Int, which we will respectively denote *A* and *B*.

When the first phase of the process described above is meant to be calculated (as opposed to devised, e.g., using expert knowledge), then the “Learn” mode of the haruspex is turned on prior to the execution of the haruspex on a set of training data. This is the case in the example of Figure 13: specifically, all observations (a,b) of (A,B) between the specified start and end times (respectively 10:52:02 a.m. and 4:45:02 p.m. on November 8, 2007) on all components called `rsnode`n, where n varies between 1 and 3000, are used to “Learn” a model. To ensure that the number of observations are the same for each component, the data is interpolated by taking the most recent observed value at even time intervals for each component.

As a result, a mean vector and a covariance matrix are calculated, and are available to the user in the “Learn” tab (not selected in the figure).



Threshold: 0.0050

	A	B
Means	1364.15	32.4116

	A	B
rsnode::PROCPIC_0_CORE (A)	5.07163	-0.407603
rsnode::PROCPIC_0_Proc_Int (B)		8.21949

Haruspex submitted successfully

```
Inputs:

[Multi Correlative Haruspex - Monitor], RsnodePROCPIC_0_CORE,
RsnodePROCPIC_0_Proc_Int, [c0-0c0s0 c0-0c0s1 c0-0c0s2 c0-0c0s3 ...
c0-4c2s7], [Nov 8 2007 10:05:02 AM - Nov 8 2007 4:45:02 PM], Threshold=0.005

Mean RsnodePROCPIC_0_CORE: 1364.15
Mean RsnodePROCPIC_0_Proc_Int: 32.4116

Cholesky RsnodePROCPIC_0_CORE: 5.07163
Cholesky RsnodePROCPIC_0_Proc_Int: 8.21949

Cholesky RsnodePROCPIC_0_CORE RsnodePROCPIC_0_Proc_Int: -0.407603

Results:

85 outliers:
Max Staleness=0, c6-2c2s2, Nov 8 2007 12:45:01 PM, (1492, 38), 1.574e-139 =
relative pdf
Max Staleness=0, c6-2c2s2, Nov 8 2007 10:45:02 AM, (1492, 38), 1.574e-139 =
relative pdf
Max Staleness=0, c6-2c2s2, Nov 8 2007 3:45:01 PM, (1492, 38), 1.574e-139 =
relative pdf
Max Staleness=0, c6-2c2s2, Nov 8 2007 10:45:02 AM, (1492, 38), 1.574e-139 =
relative pdf
Max Staleness=0, c6-2c2s2, Nov 8 2007 1:45:02 PM, (1492, 38), 1.574e-139 =
relative pdf
```

Figure 14: The interface in Figure 13, zoomed in on the analysis output.

Note that the second phase of the analysis process, called “Monitor”, can be performed on either the same data set used to infer a model, or on a different data set. For simplicity, the former option is the case in our running example. Therefore, as illustrated again in Figure 13 and Figure 14, under the “Monitor” tab, one can see the mean vector and Cholesky-decomposed covariance matrix that have been calculated by the haruspex during the “Learn” phase. In particular, the means $\mu_A = 1364.15$ and $\mu_B = 32.4116$ as well as the covariance matrix

$$\Sigma := \operatorname{cov}(A,B) = U^t U,$$

where

$$U = \begin{pmatrix} 5.07163 & -0.407603 \\ \\ 0 & 8.21949 \end{pmatrix},$$

are those of the underlying (bivariate, in this case) linear regression model.

It is beyond the scope of this article to delve into too many details about multiple linear regression models and their relationships to multivariate Gaussian distributions; one only has to know that the underlying linear model is mapped into an N-variate (bivariate in our running example) Gaussian model, whose probability density function (PDF) is, by definition,

$$f_X(x) := \frac{1}{(2\pi)^{N/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^t\Sigma^{-1}(x-\mu)\right),$$

where $x^t := (x_1,\ldots,x_N)$ is the observation of an N-tuple of interest. In our bivariate example, this simplifies into

$$f_{(A,B)}(x) = \frac{1}{2\pi|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^t\Sigma^{-1}(x-\mu)\right),$$

where $x^t := (a,b)$ and thus $(x-\mu)^t = (a-\mu_a, b-\mu_b)$. (Note that the inverse covariance matrix Σ^{-1} is computed only once, by means of the Cholesky decomposition.) The argument of the exponential is $-\frac{1}{2}$ times the squared Mahalanobis distance, which is the natural metric associated with the multivariate distribution. The significance level of observation x is defined as the probability (in the Gaussian model) of observing a Mahalanobis distance greater than that of x. This significance level is a natural choice of cumulative distribution function (CDF) for the multivariate Gaussian distribution; it ranges from 1 for a central (mean) observation to 0 for observations infinitely far from the mean vector.

With this in mind, we define an outlier as any observation x of X whose significance level is less than a user-specified threshold τ (typically $\tau \ll 1$). If observations accurately follow the multivariate Gaussian model, then a fraction τ of observations should meet this criterion. Observations may not follow the inferred model, however, either because the data are non-Gaussian or because the data being monitored have a different distribution from the training data. Nevertheless, the computed significance level is useful in assessing the deviance of an observation.

A simpler description of the significance level is possible in the bivariate case. There, the significance level happens to equal the exponential factor in the Gaussian PDF. This factor can then be described as the relative probability (normalized to the maximum of the PDF), and an outlier can alternatively be defined as an observation x with

$$\frac{f_X(x)}{\max_{\mathbf{R}^2} f_X} < \tau.$$

As shown in the "Monitor" tab of Figure 14, a threshold value of $\tau = 0.005$ was chosen, resulting in 85 outliers being reported by the haruspex, and listed in the lower right text window of the user interface. For example, the first of these outliers corresponds to an observed value of $(1492, 38)$, which, in the context of the underlying model, has a significance level (or relative probability) of $\approx 1.574 \cdot 10^{-139} < \tau = 0.005$, making it an outlier according to our definition. (Such a vanishingly small value indicates that the data are non-Gaussian, since an event with actual probability of order 10^{-139} would not realistically occur.) In turn, in the cluster view of Figure 13, all components evincing outlier behavior at the time shown in the view are colored in red, whereas other components appear in green (data were not collected on the grayed-out components during the time interval of interest).

The surface determined by learn and the data comparison in monitor are shown in Figure 15.

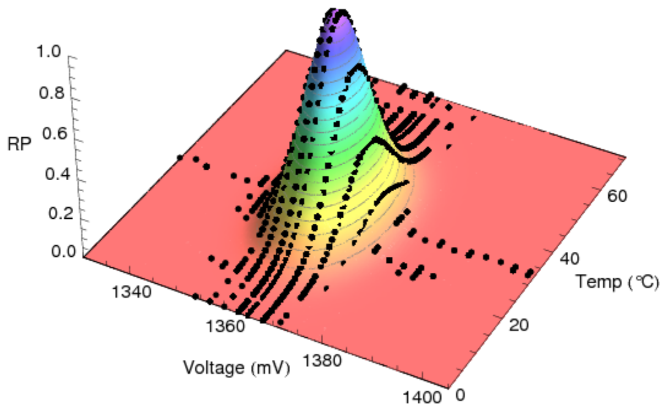


Figure 15: Plot of the surface determined by the Learn phase of the Haruspex. Actual instances are shown on the surface as black dots. Instance values are compared to the surface to determine outliers in the monitor phase.

8 Baron

The Baron is the graphical interface of OVIS which enables exploration of the data. The Baron allows the user to:

1. select a cluster and a relevant database
2. visualize cluster geometry in a physically accurate display
3. create derived variable values
4. visually inspect raw and derived variable values on the physical display
5. create haruspices and inspect the results of their analyses
6. view historical data, browsing through time history both manually and animatedly
7. view live data, updating in real-time, and
8. tune many display parameters.

Features and capability of the Baron are described in this section. Figure 27 is a picture of the Baron with the elements referred to in this section labeled.

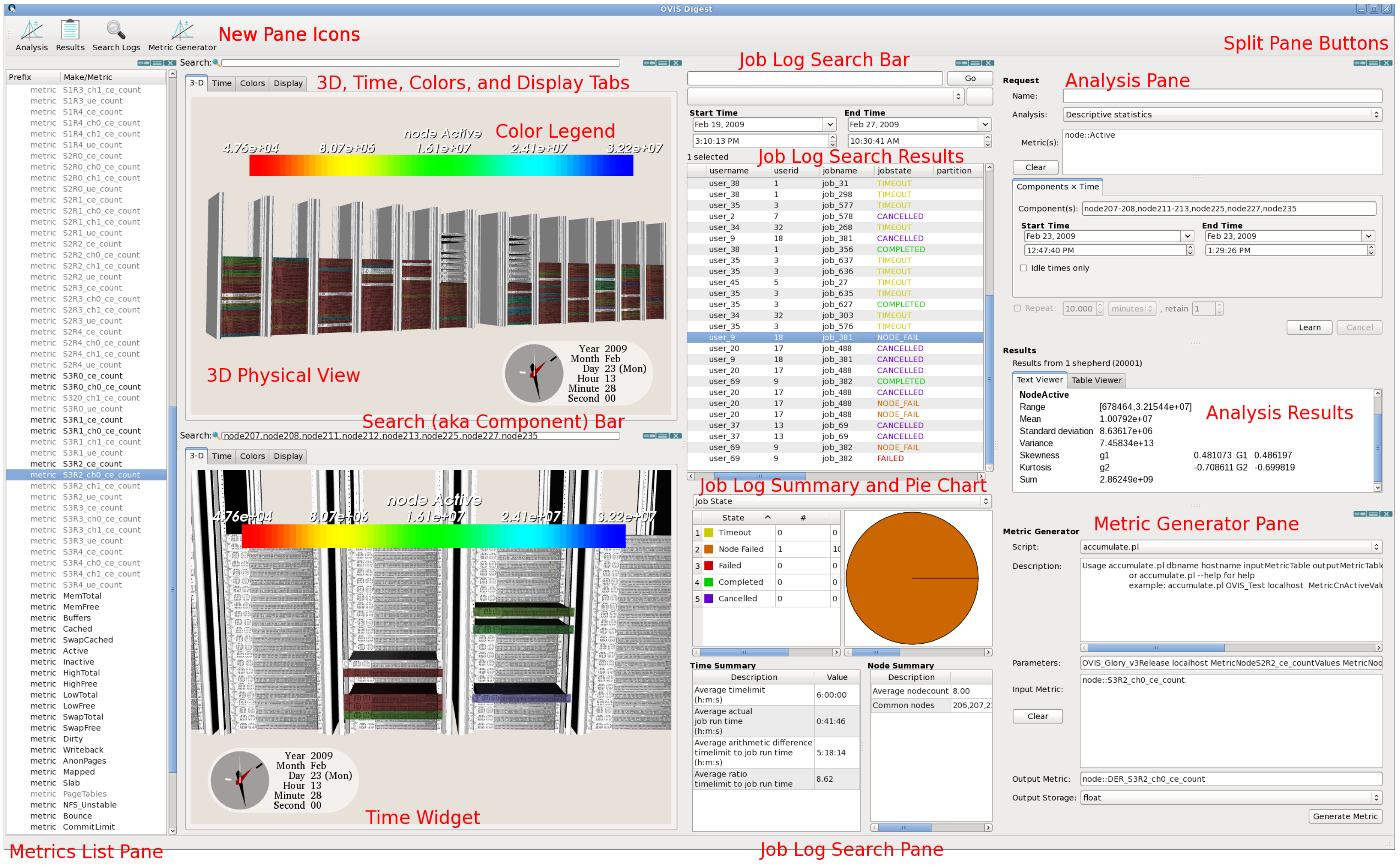
8.1 Cluster and Database Selection

The Bookmark Editor allows you to specify to which database you want to connect. Figure 17 (left) shows selection options filled out for connecting to local database holding the test data set for the Glory cluster. Use of this data set is discussed in §9.1. In addition to selecting a database, you may enter connection parameters (user name and, possibly, a password) in the Server Connection window in shown in Figure 17 (right). **WARNING: In the ServerConnection window, if you choose the option to remember the password for any of the different OVIS_⟨clustername⟩ databases, the user name and password on your database will be stored in plain text in the file** `${HOME}/.config/Sandia/ovis.conf`.

The yellow star icon indicates a user-specified static entry. The blue globe icon indicates an automatically populated entry, representing an available Shepherd as advertised by Avahi.

8.2 Adding Views

Any number of panes may be shown simultaneously. You can create additional panes by either clicking on the New Pane Icons in the upper left corner of Figure 27 or the Split Pane buttons in



Search: node207.node208.node211.node212.node213.node225.node227.node235

Year 2009
Month Feb
Day 23 (Mon)
Hour 13
Minute 28
Second 00

Prefix	Make/Metric
metric	S1R3_ch1_ce_count
metric	S1R3_ue_count
metric	S1R4_ce_count
metric	S1R4_ch0_ce_count
metric	S1R4_ch1_ce_count
metric	S1R4_ue_count
metric	S2R0_ce_count
metric	S2R0_ch0_ce_count
metric	S2R0_ch1_ce_count
metric	S2R0_ue_count
metric	S2R1_ce_count
metric	S2R1_ch0_ce_count
metric	S2R1_ch1_ce_count
metric	S2R1_ue_count
metric	S2R2_ce_count
metric	S2R2_ch0_ce_count
metric	S2R2_ch1_ce_count
metric	S2R2_ue_count
metric	S2R3_ce_count
metric	S2R3_ch0_ce_count
metric	S2R3_ch1_ce_count
metric	S2R3_ue_count
metric	S2R4_ce_count
metric	S2R4_ch0_ce_count
metric	S2R4_ch1_ce_count
metric	S2R4_ue_count
metric	S3R0_ce_count
metric	S3R0_ch0_ce_count
metric	S320_ch1_ce_count
metric	S3R0_ue_count
metric	S3R1_ce_count
metric	S3R1_ch0_ce_count
metric	S3R1_ch1_ce_count
metric	S3R1_ue_count
metric	S3R2_ce_count
metric	S3R2_ch0_ce_count
metric	S3R2_ch1_ce_count
metric	S3R2_ue_count
metric	S3R3_ce_count
metric	S3R3_ch0_ce_count
metric	S3R3_ch1_ce_count
metric	S3R3_ue_count
metric	S3R4_ce_count
metric	S3R4_ch0_ce_count
metric	S3R4_ch1_ce_count
metric	S3R4_ue_count
metric	MemTotal
metric	MemFree
metric	Buffers
metric	Cached
metric	SwapCached
metric	Active
metric	Inactive
metric	HighTotal
metric	HighFree
metric	LowTotal
metric	LowFree
metric	SwapTotal
metric	SwapFree
metric	Dirty
metric	Writeback
metric	AnonPages
metric	Mapped
metric	Slab
metric	PageTables
metric	NFS_Unstable
metric	Bounce
metric	CommitLimit

Start Time: Feb 19, 2009 3:10:13 PM
End Time: Feb 27, 2009 10:30:41 AM

1 selected

username	userid	jobname	jobstate	partition
user_38	1	job_31	TIMEOUT	
user_38	1	job_298	TIMEOUT	
user_35	3	job_577	TIMEOUT	
user_2	7	job_578	CANCELLED	
user_34	32	job_268	TIMEOUT	
user_9	18	job_381	CANCELLED	
user_38	1	job_356	COMPLETED	
user_35	3	job_637	TIMEOUT	
user_35	3	job_636	TIMEOUT	
user_45	5	job_27	TIMEOUT	
user_35	3	job_635	TIMEOUT	
user_35	3	job_627	COMPLETED	
user_34	32	job_303	TIMEOUT	
user_35	3	job_576	TIMEOUT	
user_9	18	job_381	NODE_FAIL	
user_20	17	job_488	CANCELLED	
user_9	18	job_381	CANCELLED	
user_20	17	job_488	CANCELLED	
user_69	9	job_382	COMPLETED	
user_20	17	job_488	CANCELLED	
user_20	17	job_488	NODE_FAIL	
user_20	17	job_488	NODE_FAIL	
user_37	13	job_69	CANCELLED	
user_37	13	job_69	CANCELLED	
user_69	9	job_382	NODE_FAIL	
user_69	9	job_382	FAILED	

Job State

		State	#
1	🟨	Timeout	0
2	🟧	Node Failed	1
3	🟥	Failed	0
4	🟩	Completed	0
5	🟪	Cancelled	0

Time Summary

Description	Value
Average timelimit (h:m:s)	6:00:00
Average actual job run time (h:m:s)	0:41:46
Average arithmetic difference timelimit to job run time (h:m:s)	5:18:14
Average ratio timelimit to job run time	8.62

Node Summary

Description	
Average nodecount	8.00
Common nodes	206,207,2

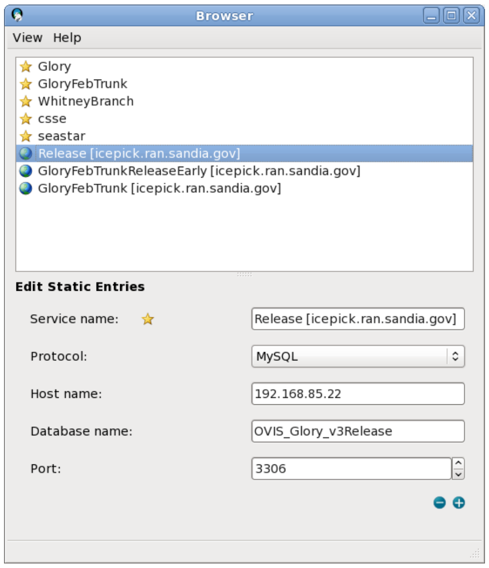
Request
Name:
Analysis: Descriptive statistics
Metric(s): node::Active
Components × Time
Component(s): node207-208,node211-213,node225,node227,node235
Start Time: Feb 23, 2009 12:47:40 PM
End Time: Feb 23, 2009 1:29:26 PM
Idle times only
Repeat: 10.000 minutes, retain 1

Results
Results from 1 shepherd (20001)

NodeActive			
Range	[678464,3.21544e+07]		
Mean	1.00792e+07		
Standard deviation	8.63617e+06		
Variance	7.45834e+13		
Skewness	g1	0.481073 G1	0.486197
Kurtosis	g2	-0.708611 G2	-0.699819
Sum	2.86249e+09		

Metric Generator
Script: accumulate.pl
Description: Usage accumulate.pl dbname hostname inputMetricTable outputMetricTable
or accumulate.pl --help for help
example: accumulate.pl OVIS_Test localhost MetricCnActiveVal
Parameters: OVIS_Glory_v3Release localhost MetricNodeS2R2_ce_countValues MetricNod
Input Metric: node::S3R2_ch0_ce_count
Output Metric: node::DER_S3R2_ch0_ce_count
Output Storage: float

Figure 16: Overview of the elements of the Baron with annotation.



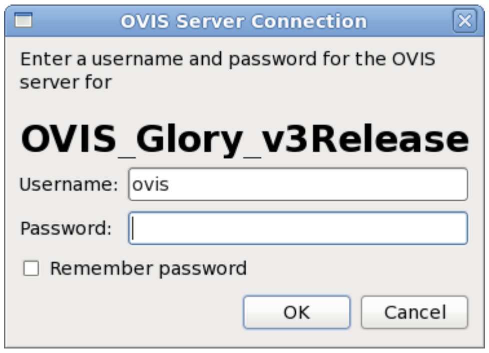
Browser

View Help

Glory
GloryFebTrunk
WhitneyBranch
csse
seastar
Release [icepick.ran.sandia.gov]
GloryFebTrunkReleaseEarly [icepick.ran.sandia.gov]
GloryFebTrunk [icepick.ran.sandia.gov]

Edit Static Entries

Service name:	Release [icepick.ran.sandia.gov]
Protocol:	MySQL
Host name:	192.168.85.22
Database name:	OVIS_Glory_v3Release
Port:	3306



OVIS Server Connection

Enter a username and password for the OVIS server for

OVIS_Glory_v3Release

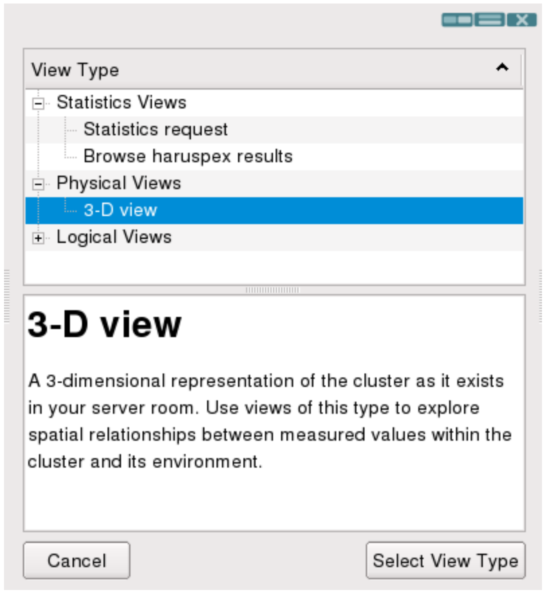
Username: ovis

Password:

☐ Remember password

OK Cancel

Figure 17: Bookmark Editor (left) and Server Connection (right) windows.



View Type

- Statistics Views
  - Statistics request
  - Browse haruspex results
- Physical Views
  - 3-D view
- Logical Views

3-D view

A 3-dimensional representation of the cluster as it exists in your server room. Use views of this type to explore spatial relationships between measured values within the cluster and its environment.

Cancel Select View Type

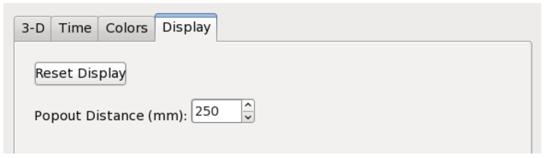
Figure 18: Options for instantiating a new pane.

the upper right corner of any existent pane, also seen in Figure 27 which split the existent pane in the direction indicated in the buttons. The buttons include one by which a pane can be closed.

When a split is selected, options for the new pane are presented and can be selected as in Figure 18.

8.3 Rotating, Panning, and Zooming the 3-D View

The second pane from the the left in Figure 27 is a 3-D interactive physical representation of the cluster of interest. Moving the mouse while pressing the left mouse button will rotate the 3-D view. Moving the mouse while pressing the middle mouse button will pan the 3-D view. Moving the mouse while pressing the right button will zoom the 3-D view. Also, rotating the mouse wheel button will zoom the 3-D view. Selecting the *Display* tab, shown in Figure 19, presents the user with the option to reset the physical view to its original position.



Popout Distance (mm): 250

Figure 19: The display tab, which resets the 3-D display and controls the pop out distance.

8.4 Search Bar

The Search Bar (aka Component Bar) 27 is used to pop out components in the physical display. This is shown in the lower physical pane of Figure 27 where we have popped out nodes involved in a particular job. All component types can be popped out, for example, racks and CPUs where available. Components to pop out can be specified by short name (e.g., cn207) or by component type and number (e.g., rack8). The advantage of the former is that the short name is generally well known. The advantage of the latter is range notation is supported (e.g., “node10-14,node57-65”).

Jobs can be dropped on the search bar, resulting in the job components being popped out, which allows one to more easily see the data for a particular job through time. This is made particularly easy when coupled with the `Color popped out components only` option in the *Colors* tab described further in §8.7. Pop out distance is controlled by the *Display* tab as shown in Figure 19.

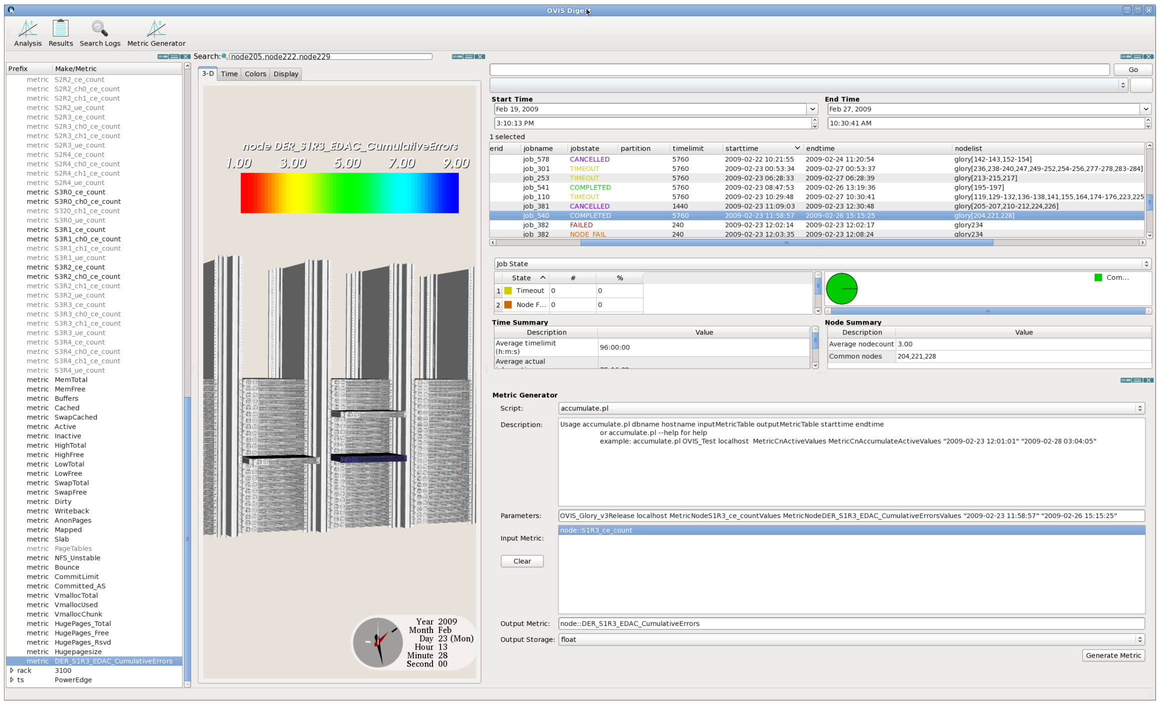
8.5 Metric Drop

On the far left of Figure 27 is a pane which lists the various component types in this data set and the metrics which are taken upon them. Metrics can be selected with the mouse and dragged and dropped upon the physical view (Middle pane). The physical view will then color the elements by the selected metric. A color bar will indicate the color-value mapping. By default, the range has as its max and min values the max and min values for that metric for the time displayed (More on Time in §8.10). The range can be overridden as described in §8.7.

Components having no value at that time (or within the relevant fade period, described in §8.10) will be shown as gray.

8.6 Metric Generator

The Metric Generator pane is shown in the lower right fof Figure 29. It consists of a selection bar from which scripts can be selected. When a script is selected, the result of its `help` output is shown in the window below to indicate the required arguments. These arguments can then be input into the other fields of the display. Once the Metric Generator has returned, the new metric is available for display and analysis as shown in the Metric List in the left of Figure 29.



Prefix	Make/Metric
metric	S2R2_ce_count
metric	S2R2_ch0_ce_count
metric	S2R2_ch1_ce_count
metric	S2R2_ue_count
metric	S2R3_ce_count
metric	S2R3_ch0_ce_count
metric	S2R3_ch1_ce_count
metric	S2R3_ue_count
metric	S2R4_ce_count
metric	S2R4_ch0_ce_count
metric	S2R4_ch1_ce_count
metric	S2R4_ue_count
metric	S3R0_ce_count
metric	S3R0_ch0_ce_count
metric	S320_ch1_ce_count
metric	S3R0_ue_count
metric	S3R1_ce_count
metric	S3R1_ch0_ce_count
metric	S3R1_ch1_ce_count
metric	S3R1_ue_count
metric	S3R2_ce_count
metric	S3R2_ch0_ce_count
metric	S3R2_ch1_ce_count
metric	S3R2_ue_count
metric	S3R3_ce_count
metric	S3R3_ch0_ce_count
metric	S3R3_ch1_ce_count
metric	S3R3_ue_count
metric	S3R4_ce_count
metric	S3R4_ch0_ce_count
metric	S3R4_ch1_ce_count
metric	S3R4_ue_count
metric	MemTotal
metric	MemFree
metric	Buffers
metric	Cached
metric	SwapCached
metric	Active
metric	Inactive
metric	HighTotal
metric	HighFree
metric	LowTotal
metric	LowFree
metric	SwapTotal
metric	SwapFree
metric	Dirty
metric	Writeback
metric	AnonPages
metric	Mapped
metric	Slab
metric	PageTables
metric	NFS_Unstable
metric	Bounce
metric	CommitLimit
metric	Committed_AS
metric	VmallocTotal
metric	VmallocUsed
metric	VmallocChunk
metric	HugePages_Total
metric	HugePages_Free
metric	HugePages_Rsvd
metric	Hugepagesize
metric	DER_S1R3_EDAC_CumulativeErrors
rack	3100
ts	PowerEdge

Start Time Feb 19, 2009 3:10:13 PM
End Time Feb 27, 2009 10:30:41 AM

1 selected

erid	jobname	jobstate	partition	timelimit	starttime	endtime	nodelist
	job_578	CANCELLED		5760	2009-02-22 10:21:55	2009-02-24 11:20:54	glory[142-143,152-154]
	job_301	TIMEOUT		5760	2009-02-23 00:53:34	2009-02-27 00:53:37	glory[236,238-240,247,249-252,254-256,277-278,283-284]
	job_253	TIMEOUT		5760	2009-02-23 06:28:33	2009-02-27 06:28:39	glory[213-215,217]
	job_541	COMPLETED		5760	2009-02-23 08:47:53	2009-02-26 13:19:36	glory[195-197]
	job_110	TIMEOUT		5760	2009-02-23 10:29:48	2009-02-27 10:30:41	glory[119,129-132,136-138,141,155,164,174-176,223,225
	job_381	CANCELLED		1440	2009-02-23 11:09:03	2009-02-23 12:30:48	glory[205-207,210-212,224,226]
	job_540	COMPLETED		5760	2009-02-23 11:58:57	2009-02-26 15:15:25	glory[204,221,228]
	job_382	FAILED		240	2009-02-23 12:02:14	2009-02-23 12:02:17	glory234
	job_382	NODE_FAIL		240	2009-02-23 12:03:35	2009-02-23 12:08:24	glory234

Job State

	State	#	%
1	🟨 Timeout	0	0
2	🟧 Node F...	0	0

Time Summary

Description	Value
Average timelimit (h:m:s)	96:00:00
Average actual	

Node Summary

Description	Value
Average nodecount	3.00
Common nodes	204,221,228

Metric Generator

Script: accumulate.pl

Description:
```
Usage accumulate.pl dbname hostname inputMetricTable outputMetricTable starttime endtime
        or accumulate.pl --help for help
        example: accumulate.pl OVIS_Test localhost  MetricCnActiveValues MetricCnAccumulateActiveValues "2009-02-23 12:01:01" "2009-02-28 03:04:05"
```

Parameters: OVIS_Glory_v3Release localhost MetricNodeS1R3_ce_countValues MetricNodeDER_S1R3_EDAC_CumulativeErrorsValues "2009-02-23 11:58:57" "2009-02-26 15:15:25"

Input Metric: node::S1R3_ce_count

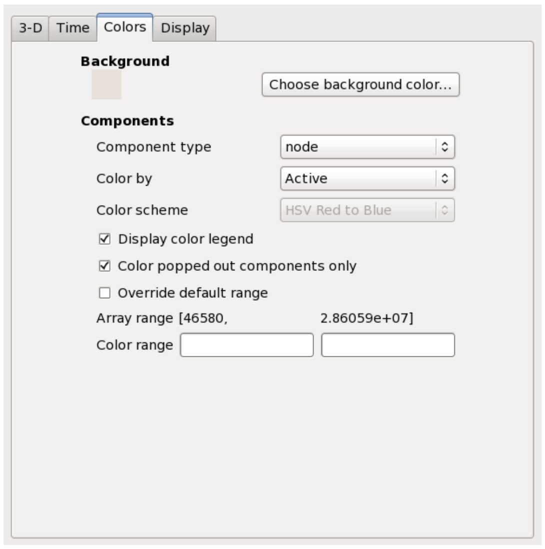
Output Metric: node::DER_S1R3_EDAC_CumulativeErrors

Output Storage: float

Figure 20: Metric Generator Pane and resulting Derived Metric in the Metric List and droppped on the 3-D display.

8.7 Setting the Colors

The Baron provides a default background color, as well as default color scheme, scale and ranges for the component variables. These can be modified through the *Colors* tab – the tab is shown in 27), with the content of the tab illustrated in Figure 21. The Components section of the Color



Background

Components

Component type: node
Color by: Active
Color scheme: HSV Red to Blue
☑ Display color legend
☑ Color popped out components only
☐ Override default range
Array range [46580, 2.86059e+07]
Color range

Figure 21: The color tab, where the color legend and range can be set.

tab allows one to override the default range for the color legend for metrics. This can be done by selecting `Override default range` and specifying the range explicitly. If this is done, the color range for the metrics will be fixed, even as one animates through time (see §8.10).

Setting the range manually allows easy visual comparison of the distribution and locations of regions of interest across time and across data sets. Selecting a subset of the overall possible range allows one to see finer grained detail within a section of interest. For example, if the range for a particular metric ranges from 0-100, one may be primarily interested in details of the values in the upper end of the range, and therefore may set the color range from 80-100, for instance.

The color tab also enables the option to more easily see the behavior of data on components selected in the Search Bar by selecting the `Color popped out components only` option in the *Colors* tab.

All color related options in this section will be retained as described in §8.13.

8.8 Job Log Search

The Baron has a job log search capability as described in §6. The Job Search pane is pictured in Figure 22. The top of the pane features a search bar where jobs can be subselected based on criteria such as userid, job completion state, components participating in the job, etc. Compound searches involving `AND` and `OR` are supported. Jobs can also be subselected based on Time by use of the `Start Time` and `End Time` feature – this will select jobs whose run time overlaps that indicated time range.

Jobs matching the criteria are shown in the Results area of the pane. Some statistics for these jobs are shown in the rest of the pane. This includes an interactive pie chart that can also display the distribution of the selected jobs based on Job Completion State, User Name etc. Thus one can select, for example, jobs that `FAIL` and then use the pie chart to see those jobs distributed by `Username`. The pie chart and its labels can be clicked on to do further subselection. Jobs can be selected in the results area of the pane to also subselect for the pie chart.

Double clicking on a job in the results section of the pane, brings up an Analysis pane, populated with the times and components of the job.

8.9 Haruspices

The Baron provides a graphical interface for creating and obtaining the results of the haruspices (cf. 7).

A typical analysis pane is shown in Figure 23. It contains regions in which to input the metric or metrics of interest, the time range of the calculation, and the components involved in the calculation. The metrics can be populated by dragging and dropping them from the metric list. The components can be specified by component type and number, in which case ranges are supported, or by short name.

learn analyses learn a model as described in §7.1. Clicking the Learn button on the pane starts the analysis. When the results in an analysis window are not current, either because new parameters are being entered into an analysis pane or because the analysis has not yet returned, the buttons and entry text boxes in the Analysis pane are grayed out.

The *Repeat Analysis* button is currently not enabled. When enabled it will automatically recalculate the analysis including any newly collected data. This will ensure that the model is current and will allow the user to note model changes with time. This feature will be enabled in a future release.

8.10 Time Features

One can manually scroll through time or have the Baron automatically animate playing through time, and view the current state in the physical display.

The user interactive time widget 24, shows the current time in the physical display. There are marks/hands for month, day, hours, minutes, and seconds that can be grabbed and pulled forward and backward in time with the current state shown in text as well.

Additional handling of time is done in the *Time* tab – the tab is shown in 27), with the content of the tab illustrated in Figure 25.

The first check-box controls the visibility of the time widget. The box below enables animated playing through time. Setting the `x RealTime` entry to values greater than zero enables automatic playback at the specified rate, where 1 equals real time. Setting the `Frame Rate (target)` entry specifies the maximum number of times per second that the Baron should update and redraw the cluster. High values require more communication with the database but will make color fades and the motion of the clock hands on the time widget appear smoother. Low values require less communication but may result in a “jumpy” looking interface. It is possible to specify a frame rate that the Baron is unable to produce. In this case, it will redraw the cluster as quickly as it can.

After the `x RealTime` and `Frame Rate (target)` entries have been set, the user can press the space bar (on the keyboard) in the 3-D View tab to start and stop the clock. Note that stopping the clock will only freeze the display, and the internal clock will continue to progress; the 3-D View will reflect the data at the frozen time.

The playback time section allows one to manually set the initial time of the clock. The `Earliest` button will set the time to the earliest time in the database; the `Now` button will set the time to the current time on the machine. After adjusting the time, click the `Apply` button for the changes to take place.

Data is recorded in the database at the fidelity of a second. Because of issues such as clock skew, database insert times, etc., it is desirable to see all data not at a given point in time, but within a window of time. For this region a `Fade Period` can be set that will allow components in the physical data to be colored by the data value corresponding for that time nearest in time to that shown on the clock within the specified fade period. For example, for the Glory database, described in §9.1 data is taken on 60 second intervals, so one should set the Fade Period to 120, and preferably greater, so that all components will be colored by a timely value. In order to distinguish the age of values, the component color will also fade out as the data value gets increasingly distant from the clock time. There is no correct value of the fade period – this is determined by the frequency of data collection/inserts and the user’s desire regarding the fading effect. For example, if the user finds the fading distracting, a longer time may be desirable; if the user is specifically trying to investigate when components cease to report a smaller fade time is more appropriate. Note also that longer fade periods require more data to be sifted through so that longer fade periods result in decreasing performance.

Note that currently the fade period is a global variable pertaining to all physical views in the Baron.

All time related options in this section will be retained as described in §8.13.

8.11 Haruspex Requests View

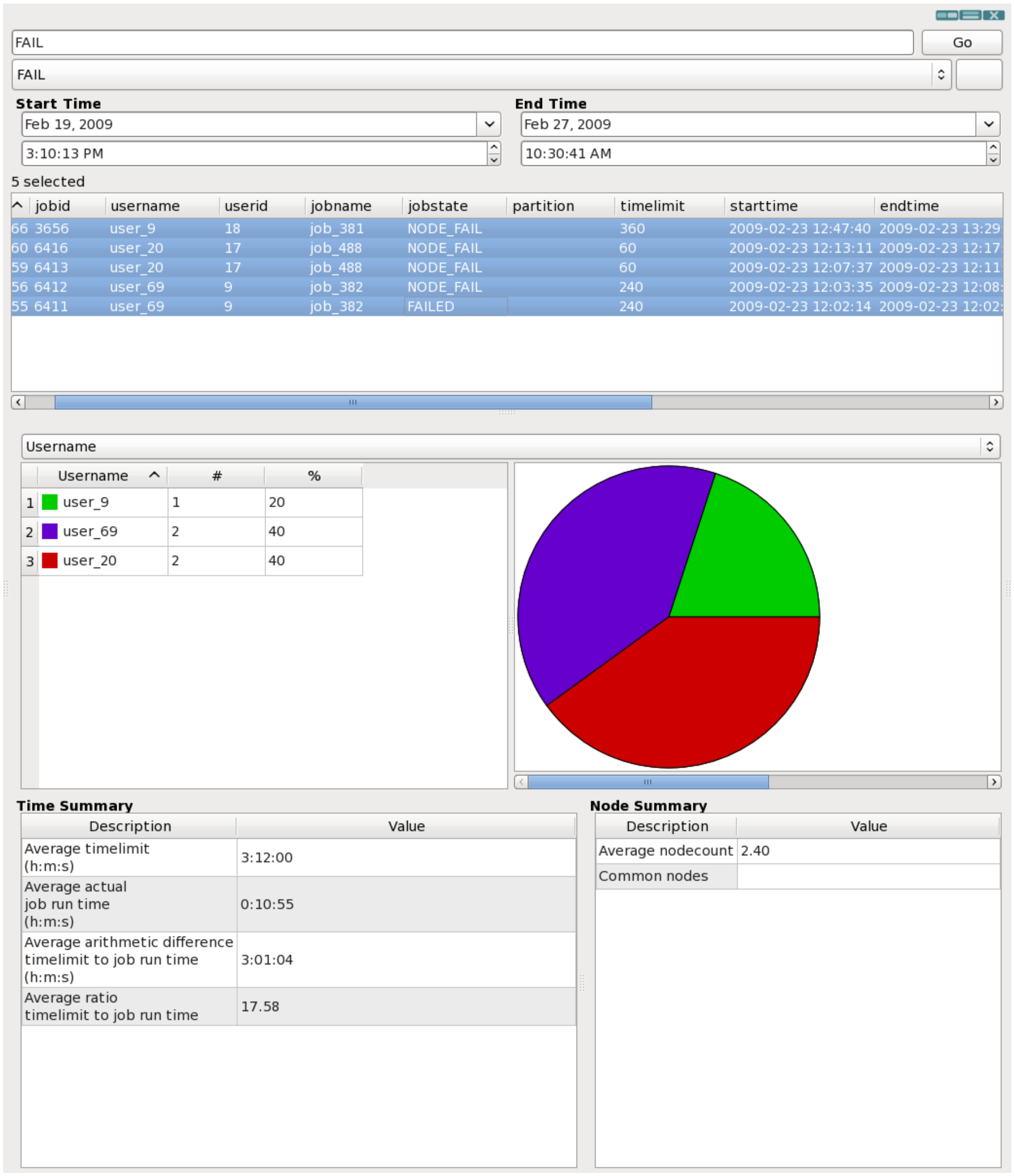
The Haruspex Requests view 26 can be generated via the New Pane icons in 27. This is a table that lists the previously requested analyses, displaying the RequestId and some parameters of the request. Single clicking an entry brings up the results. Double clicking an entry instantiates the corresponding Analysis view, filled out with the request and results. This allows the user to view the results without having to redo the analysis. The analysis pane generated in this way is similar to other generated panes and can be dropped on the the physical display and Search Bar, as usual.

8.12 Multiple Shepherds

If multiple Shepherds are being used, the Baron will connect to one, via the choice in the Bookmark Editor. When an analysis is invokved via the Baron, it will take place in parallel across all the databases involved, with the aggregate result reported in the Baron. Currently, however, the visualization presents only the data available in the database to which the Baron is connected. Visualization of distributed data will be implemented in a future release.

8.13 Saving State

State, including color bar ranges, time ranges for analyses, time shown in the physical pane, etc., in plain text in a file `${HOME}/.config/Sandia/ovis.conf`.



FAIL — Go

FAIL

Start Time

Feb 19, 2009

3:10:13 PM

End Time

Feb 27, 2009

10:30:41 AM

5 selected

	jobid	username	userid	jobname	jobstate	partition	timelimit	starttime	endtime
66	3656	user_9	18	job_381	NODE_FAIL		360	2009-02-23 12:47:40	2009-02-23 13:29
60	6416	user_20	17	job_488	NODE_FAIL		60	2009-02-23 12:13:11	2009-02-23 12:17
59	6413	user_20	17	job_488	NODE_FAIL		60	2009-02-23 12:07:37	2009-02-23 12:11
56	6412	user_69	9	job_382	NODE_FAIL		240	2009-02-23 12:03:35	2009-02-23 12:08:
55	6411	user_69	9	job_382	FAILED		240	2009-02-23 12:02:14	2009-02-23 12:02:

Username

	Username	#	%
1	🟩 user_9	1	20
2	🟪 user_69	2	40
3	🟥 user_20	2	40

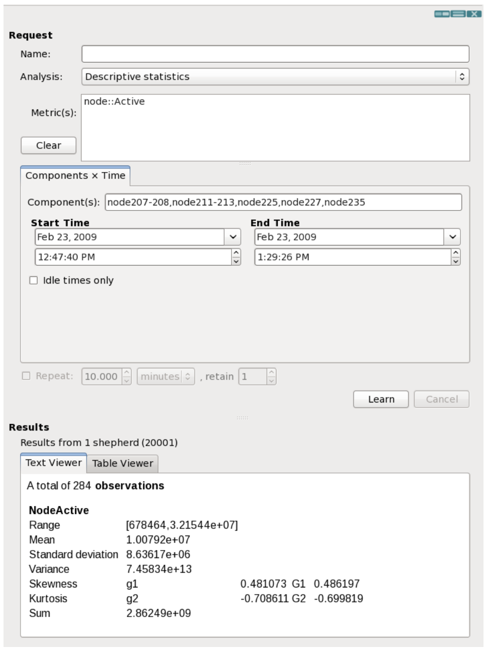
Time Summary

Description	Value
Average timelimit (h:m:s)	3:12:00
Average actual job run time (h:m:s)	0:10:55
Average arithmetic difference timelimit to job run time (h:m:s)	3:01:04
Average ratio timelimit to job run time	17.58

Node Summary

Description	Value
Average nodecount	2.40
Common nodes	

Figure 22: The job log pane, with search and results display, including drill down pie chart.



Request

Name:

Analysis: Descriptive statistics

Metric(s): node::Active

Components × Time

Component(s): node207-208,node211-213,node225,node227,node235

Start Time: Feb 23, 2009 12:47:40 PM

End Time: Feb 23, 2009 1:29:26 PM

☐ Idle times only

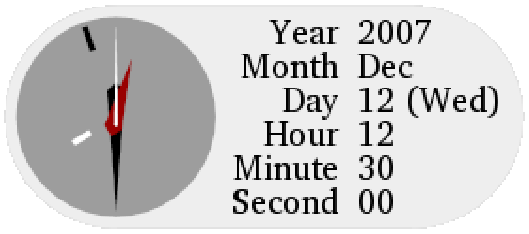
☐ Repeat: 10.000 minutes , retain 1

Results

Results from 1 shepherd (20001)

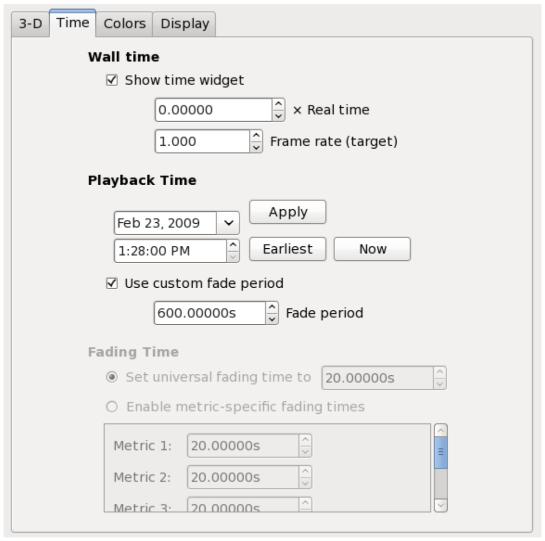
```
A total of 284 observations

NodeActive
Range               [678464,3.21544e+07]
Mean                1.00792e+07
Standard deviation  8.63617e+06
Variance            7.45834e+13
Skewness            g1                    0.481073  G1   0.486197
Kurtosis            g2                    -0.708611 G2   -0.699819
Sum                 2.86249e+09
```

Figure 23: Descriptive learn Analysis pane where the metric, components, and time range for analysis are specified



```
  Year  2007
 Month  Dec
   Day  12 (Wed)
  Hour  12
Minute  30
Second  00
```

Figure 24: The user interactive time widget allows the user to scroll through time in the physical display.



Wall time

☑ Show time widget

0.00000 × Real time

1.000 Frame rate (target)

Playback Time

Feb 23, 2009 — Apply

1:28:00 PM — Earliest — Now

☑ Use custom fade period

600.00000s Fade period

Fading Time

● Set universal fading time to 20.00000s

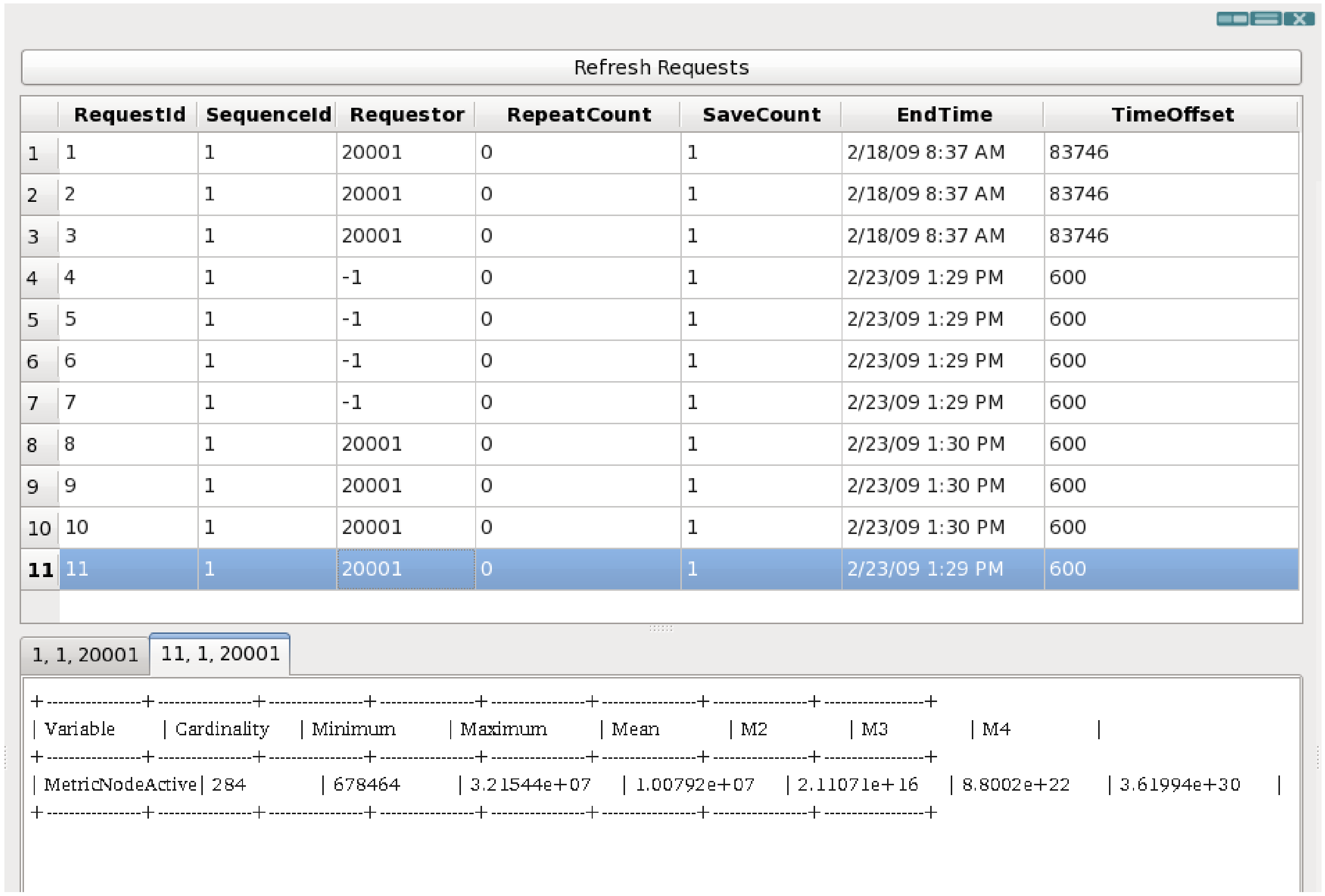
○ Enable metric-specific fading times

Metric 1: 20.00000s

Metric 2: 20.00000s

Metric 3: 20.00000s

Figure 25: The Time tab, which allows the user to set the time; choose to play through time; and set the fade period.



Refresh Requests

	RequestId	SequenceId	Requestor	RepeatCount	SaveCount	EndTime	TimeOffset
1	1	1	20001	0	1	2/18/09 8:37 AM	83746
2	2	1	20001	0	1	2/18/09 8:37 AM	83746
3	3	1	20001	0	1	2/18/09 8:37 AM	83746
4	4	1	-1	0	1	2/23/09 1:29 PM	600
5	5	1	-1	0	1	2/23/09 1:29 PM	600
6	6	1	-1	0	1	2/23/09 1:29 PM	600
7	7	1	-1	0	1	2/23/09 1:29 PM	600
8	8	1	20001	0	1	2/23/09 1:30 PM	600
9	9	1	20001	0	1	2/23/09 1:30 PM	600
10	10	1	20001	0	1	2/23/09 1:30 PM	600
11	11	1	20001	0	1	2/23/09 1:29 PM	600

```
+---------------+---------------+---------------+---------------+---------------+---------------+---------------+---------------+
| Variable      | Cardinality   | Minimum       | Maximum       | Mean          | M2            | M3            | M4            |
+---------------+---------------+---------------+---------------+---------------+---------------+---------------+---------------+
| MetricNodeActive| 284         | 678464        | 3.21544e+07   | 1.00792e+07   | 2.11071e+16   | 8.8002e+22    | 3.61994e+30   |
+---------------+---------------+---------------+---------------+---------------+---------------+---------------+---------------+
```

Figure 26: The Haruspex Requests view which allows one to examine previous analyses. The upper left is an interactive table for selecting previous analyses. Single clicking an entry brings up the results in the lower left. Double clicking an entry instantiates the corresponding Analysis view.

9 Examples

The OVIS 3.2 release comes with three example cases. The first includes a set-up file and example data from a cluster. The example data is released as a separate tarball. The second and third are examples from XML files, that with minor modifications will take and display data from your local machine.

See §2 for info on the MySQL settings and general system settings before beginning.

9.1 Glory Example Data

This case involves exploration of some test data already gathered from a cluster. It illustrates use of the Baron, building the display in Figure 27.

Data is not gathered, but rather is loaded from a mysqldump of previously gathered data. Relevant files are

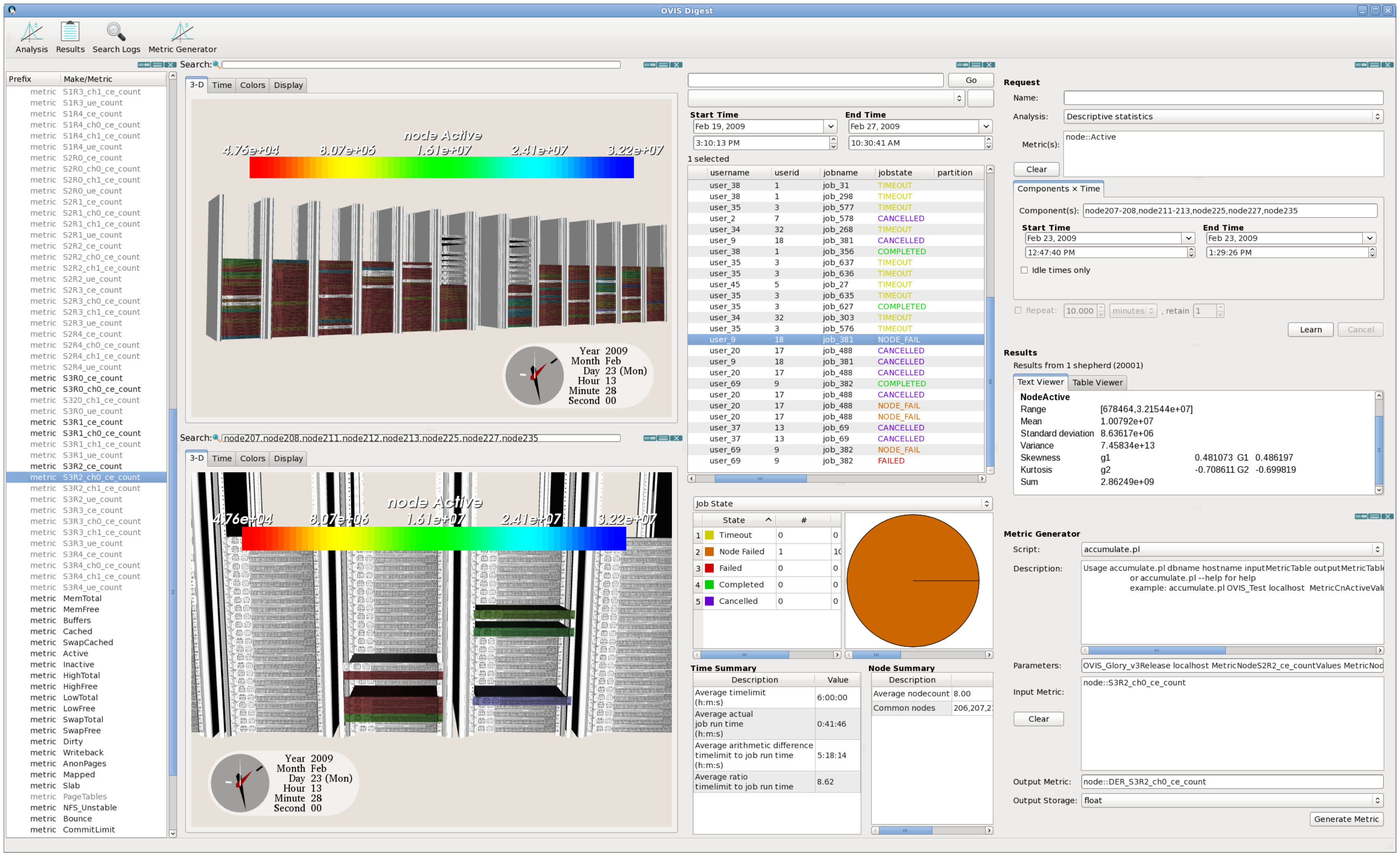
- `gloryrelease.ovdb` in the OVIS data directory which is the set-up data file which was used in the actual data collection and established the cluster display arrangement and
- `mysqldump.OVIS_Glory_v3Release.sql.tgz` which is the mysqldump of the database
- `mysqldump.slurm_mapped.sql` and `mysqldump.slurm_acct_db_mapped.sql` which are mysqldumps of the SLURM databases.
- `mysqldump.ComponentTable_slurm_job_log.sql` which is a mysqldump of a single table that contains component identification to job mappings

Getting Started

1. First load the mysqldump files. This requires creating the empty database in MySQL and then decompressing and loading the various files.

```
mysql> create database OVIS_Glory;
mysql -u ovis OVIS_Glory < mysqldump.OVIS_Glory_v3Release.sql
mysql -u ovis OVIS_Glory < mysqldump.ComponentTable_slurm_job_log.sql
mysql> create database slurm;
mysql -u ovis slurm < mysqldump.slurm_mapped.sql
mysql> create database slurm_acct_db;
mysql -u ovis slurm_acct_db < mysqldump.slurm_acct_db_mapped.sql
```

2. Enable user `ovis` permissions on the databases as described in §2.



Prefix	Make/Metric
metric	S1R3_ch1_ce_count
metric	S1R3_ue_count
metric	S1R4_ce_count
metric	S1R4_ch0_ce_count
metric	S1R4_ch1_ce_count
metric	S1R4_ue_count
metric	S2R0_ce_count
metric	S2R0_ch0_ce_count
metric	S2R0_ch1_ce_count
metric	S2R0_ue_count
metric	S2R1_ce_count
metric	S2R1_ch0_ce_count
metric	S2R1_ch1_ce_count
metric	S2R1_ue_count
metric	S2R2_ce_count
metric	S2R2_ch0_ce_count
metric	S2R2_ch1_ce_count
metric	S2R2_ue_count
metric	S2R3_ce_count
metric	S2R3_ch0_ce_count
metric	S2R3_ch1_ce_count
metric	S2R3_ue_count
metric	S2R4_ce_count
metric	S2R4_ch0_ce_count
metric	S2R4_ch1_ce_count
metric	S2R4_ue_count
metric	S3R0_ce_count
metric	S3R0_ch0_ce_count
metric	S320_ch1_ce_count
metric	S3R0_ue_count
metric	S3R1_ce_count
metric	S3R1_ch0_ce_count
metric	S3R1_ch1_ce_count
metric	S3R1_ue_count
metric	S3R2_ce_count
metric	S3R2_ch0_ce_count
metric	S3R2_ch1_ce_count
metric	S3R2_ue_count
metric	S3R3_ce_count
metric	S3R3_ch0_ce_count
metric	S3R3_ch1_ce_count
metric	S3R3_ue_count
metric	S3R4_ce_count
metric	S3R4_ch0_ce_count
metric	S3R4_ch1_ce_count
metric	S3R4_ue_count
metric	MemTotal
metric	MemFree
metric	Buffers
metric	Cached
metric	SwapCached
metric	Active
metric	Inactive
metric	HighTotal
metric	HighFree
metric	LowTotal
metric	LowFree
metric	SwapTotal
metric	SwapFree
metric	Dirty
metric	Writeback
metric	AnonPages
metric	Mapped
metric	Slab
metric	PageTables
metric	NFS_Unstable
metric	Bounce
metric	CommitLimit

Search: node207,node208,node211,node212,node213,node225,node227,node235

Start Time Feb 19, 2009 3:10:13 PM
End Time Feb 27, 2009 10:30:41 AM

1 selected

username	userid	jobname	jobstate	partition
user_38	1	job_31	TIMEOUT	
user_38	1	job_298	TIMEOUT	
user_35	3	job_577	TIMEOUT	
user_2	7	job_578	CANCELLED	
user_34	32	job_268	TIMEOUT	
user_9	18	job_381	CANCELLED	
user_38	1	job_356	COMPLETED	
user_35	3	job_637	TIMEOUT	
user_35	3	job_636	TIMEOUT	
user_45	5	job_27	TIMEOUT	
user_35	3	job_635	TIMEOUT	
user_35	3	job_627	COMPLETED	
user_34	32	job_303	TIMEOUT	
user_35	3	job_576	TIMEOUT	
user_9	18	job_381	NODE_FAIL	
user_20	17	job_488	CANCELLED	
user_9	18	job_381	CANCELLED	
user_20	17	job_488	CANCELLED	
user_69	9	job_382	COMPLETED	
user_20	17	job_488	CANCELLED	
user_20	17	job_488	NODE_FAIL	
user_20	17	job_488	NODE_FAIL	
user_37	13	job_69	CANCELLED	
user_37	13	job_69	CANCELLED	
user_69	9	job_382	NODE_FAIL	
user_69	9	job_382	FAILED	

Job State

	State	#	
1	🟨 Timeout	0	0
2	🟧 Node Failed	1	1(
3	🟥 Failed	0	0
4	🟩 Completed	0	0
5	🟪 Cancelled	0	0

Time Summary

Description	Value
Average timelimit (h:m:s)	6:00:00
Average actual job run time (h:m:s)	0:41:46
Average arithmetic difference timelimit to job run time (h:m:s)	5:18:14
Average ratio timelimit to job run time	8.62

Node Summary

Description	
Average nodecount	8.00
Common nodes	206,207,2

Request

Name:
Analysis: Descriptive statistics
Metric(s): node::Active

Component(s): node207-208,node211-213,node225,node227,node235
Start Time Feb 23, 2009 12:47:40 PM
End Time Feb 23, 2009 1:29:26 PM
Idle times only
Repeat: 10.000 minutes , retain 1

Results

Results from 1 shepherd (20001)

NodeActive

Range	[678464,3.21544e+07]		
Mean	1.00792e+07		
Standard deviation	8.63617e+06		
Variance	7.45834e+13		
Skewness	g1	0.481073 G1	0.486197
Kurtosis	g2	-0.708611 G2	-0.699819
Sum	2.86249e+09		

Metric Generator

Script: accumulate.pl
Description: Usage accumulate.pl dbname hostname inputMetricTable outputMetricTabl
or accumulate.pl --help for help
example: accumulate.pl OVIS_Test localhost MetricCnActiveVal
Parameters: OVIS_Glory_v3Release localhost MetricNodeS2R2_ce_countValues MetricNod
Input Metric: node::S3R2_ch0_ce_count
Output Metric: node::DER_S3R2_ch0_ce_count
Output Storage: float

Figure 27: Using the Baron for data exploration.

3. If this is your first use of the MySQL database with OVIS it is recommended that you run the database effector to load the user defined function for initiating analyses as described in §4.2.

4. Edit the **StartupData** table so that OVIS will recognize the Shepherd on *your* machine for performing analyses on the example dataset. Note: had the collection actually been performed on your machine, your Shepherd information would already be in the database.

   ```
   mysql> use OVIS_GloryRelease;
   ```

   ```
   mysql> select * from ComponentTypes;
   ```

   #this will show you that the Shepherd has CompType 3

   ```
   mysql> select * from ComponentTable where CompType=3;
   ```

   #this will show you that the Shepherd has CompId 316

   ```
   mysql> select * from StartupData where CompId=316;
   ```

   # this will show you the current allowable Shepherds).

   ```
   mysql> insert into StartupData values(2,316,"XXX",NULL,NULL,NULL)
   ```

   # replace XXX with the hex version of your IP address. Be sure to add *all* the addresses for your Shepherd.

   ```
   mysql> insert into StartupData values(0,316,"XXX",NULL,NULL,NULL)
   ```

   # replace XXX with the hex version of your MAC address. Be sure to add *all* the addresses for your Shepherd.

   ```
   mysql> select * from StartupData where CompId=316;
   ```

   # you should now see your machine listed

5. Start the Shepherd:

   ```
   cd /path/to/ovisBuildDir
   ```

   ```
   ./bin/shepherd --name=Release \
         --database=mysql://ovis@localhost/OVIS_Glory
   ```

   If you get a warning at this point like

   ```
   Shepherd was unable to determine its own component ID.
      This will cause haruspex calculations to be unreliable
      at a minimum. Set a component ID in the StartupData table.
   ```

   then you have not added your Shepherd properly.

6. Start the Baron:

   in a different window but the same directory run:

   ```
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path/to/ovisBuildDir/lib
   ```

   ```
   ./bin/baron
   ```

7. Choose the Glory database in the Baron (The Bookmark Editor and ServerConnection windows are described in §8.1):

(a) When the Browser Window comes up select: *View→Show Bookmark Editor*

(b) Fill in the following fields:

- Service name = Release
- Protocol = MySQL
- Hostname = localhost
- Database name = OVIS_Glory

(c) Click on the plus sign

(d) Select "Release" in window

(e) When the OVIS Server Connection window comes up you should see "ovis" as the user name and nothing for the password. You can use the default, if you have enabled user ovis the appropriate permissions on the OVIS_Glory database; otherwise you can use the user name and password of your choice. Note that these will be stored (as described in §2) in plain text on your machine. Click "OK".

(f) At this point the OVIS_Digest (Baron window) should appear with the cluster displayed.

Using the Baron: Displaying Raw Metric Values

Before displaying data, set the Fade Period in the Time Tab to 600 seconds. The Fade Period and Time tab are discussed in §8.10. Determination of the correct value is based on the innate frequency of the data collection (in this case 60 second intervals) and the preference of the user as to the fading effect. If this is the first time running the Baron, the date/time may be set to January 1, 1970, GMT. To change the date/time to the earliest time at which there is data for this cluster, click on the "Earliest" button in the Time Tab of the 3-D View.

To display raw metric values in the physical display:

1. Click on "node" in the left hand menu to see the available metrics for display for the node.
2. Drag and drop a metric onto the display, such as Active.

The nodes should become colored by value as shown in Figure 28. The range of the values in the color bar is determined by the min and max metric values exhibited at the time in the display, unless overridden by settings in the Color tab (described in §8.7).

Using the Baron: Job Exploration 1

This section illustrates some of the features available for job-centric exploration.

1. First bring up the *Job Search* Pane, via the *Search Logs* icons.

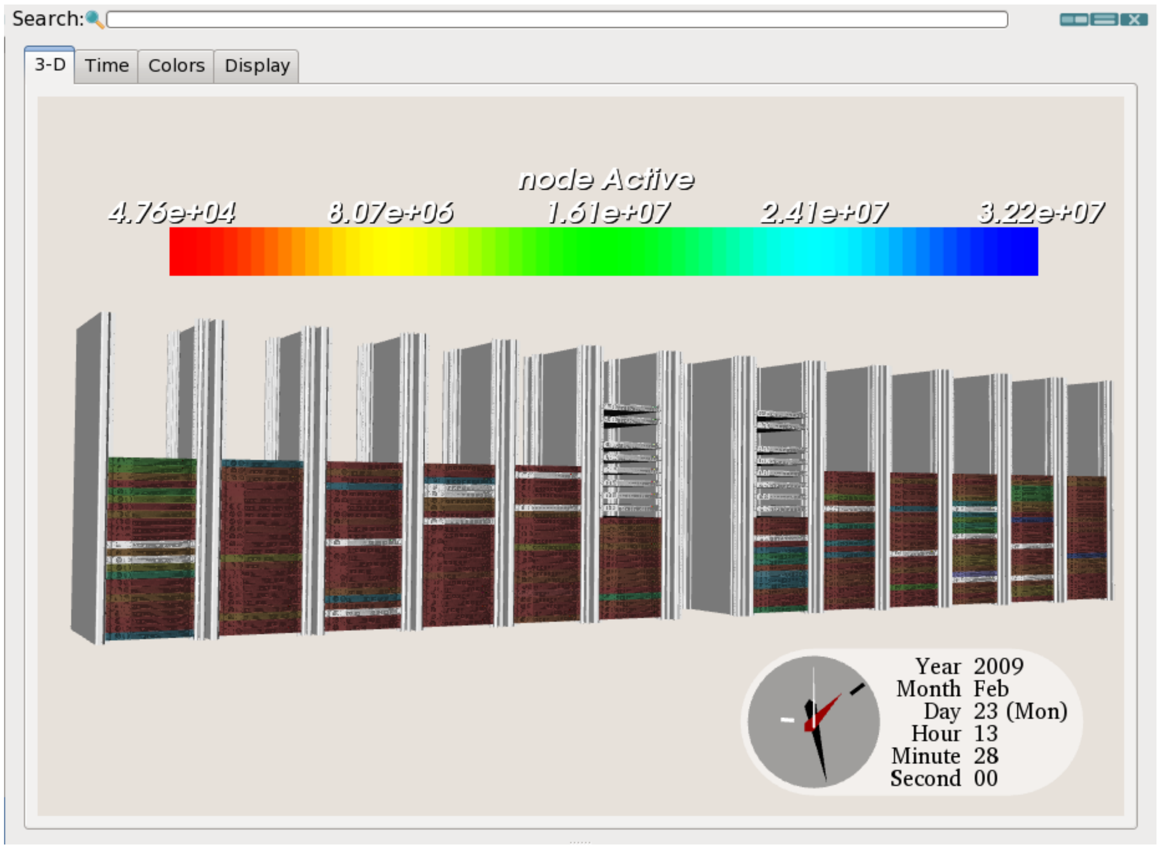


Figure 28: Raw metric values on the physical display pane.

2. Click on the *Go* button. All the jobs whose run time overlap the times given in the time range will be listed in the results panel.

3. Note the pie chart which shows the *Job State* distribution. Selecting other display options, such as *User Name*, will show the new distribution in the pie chart.

4. Inserting *FAIL* into the Search Bar and clicking on the *Go* button will return only those jobs whose information includes the word `FAIL`. This will include both jobs whose ending state is `FAIL` and `NODE FAIL`.

5. From this list, double click on the the job with id 16466. This will bring up an Analysis Pane already populated with the correct components and time range for this job.

6. Perform a *Descriptive Statistics* Analysis of Active Memory, by dragging and dropping that metric from the *Metric Pane*. Note that the distribution is unbalanced.

7. Bring up a second *Physical* display, by clicking on the *Split Pane* buttons and drop the Active Memory Metric on it.

8. Select the *Color* tab and select *Color popped out components only*.

9. Drop the job components from the *Analysis Pane* onto the *Physical* display. This should result in the nodes involved in the job being popped out in the display and in those nodes being the only nodes that are colored.

10. Popping out and coloring only the node involved in the job makes it easier to examine the behavior of data values that pertain only to the job. Manually move the clock hands, noticing that over the lifetime of the job that one node has a markedly higher Active Memory value than the other nodes participating in the job.

The Baron also supports automatic playing data through time. Details can be found in §8.10. For this example, in the Time tab shown in Figure 25:

1. set the Playback time to be the initial time given in the analysis window

2. click on Show time widget

3. set 1xRealtime

In the physical display you should see the time animation of the data. Colors shown indicate the data value at the time currently shown on the clock and the clock should be changing with time.

Using the Baron: Job Exploration 2

In this section, we use the *Metric Generator* to explore a job.

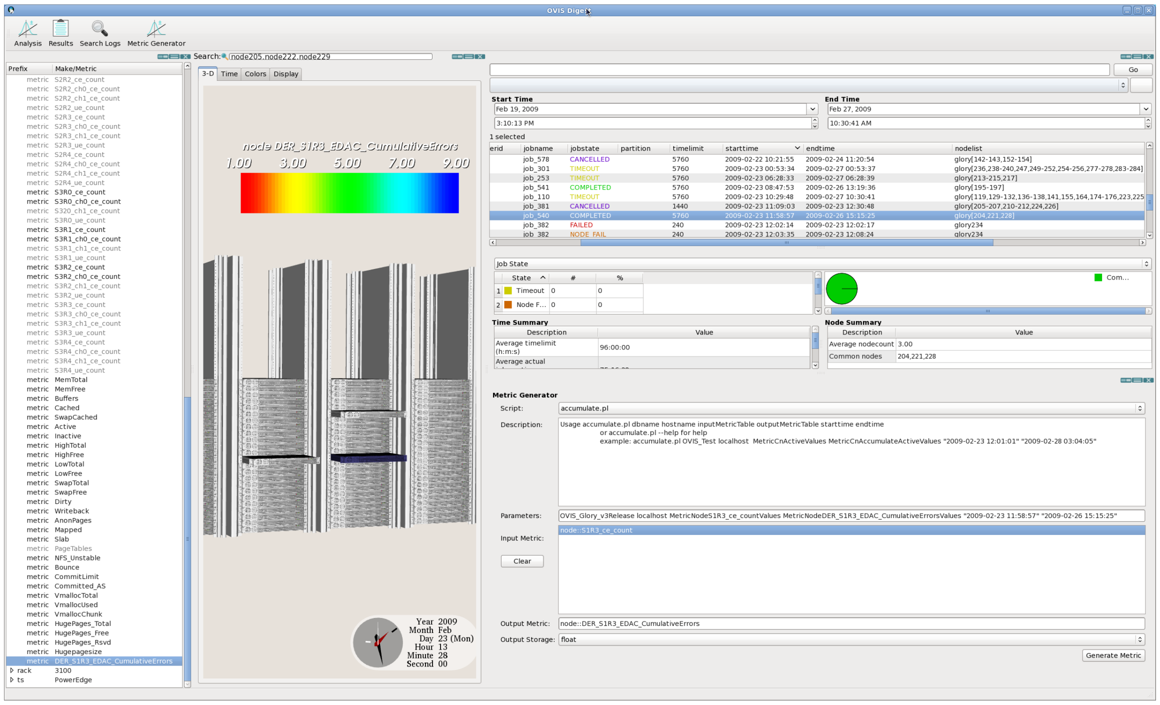
1. In the *Job Search* Pane, select Job 17317.

2. Bring up the *Metric Generator* Pane via the *Metric Generator* icon.

3. Select the *accumulate* script and populate the arguments as indicated in Figure 29. This script generates a new metric which is the accumulated val of the input metric over the time range entered.

4. Click on *Generate Metric*. When the generation is complete, the *Metric Pane* will have refreshed. Expand it again and note that the output metric of the Metric Generator is now in the list.

5. The new metric is available for analysis and display, just as any other metric. This is shown in a job component drop on the *Physical* display in the figure, where only one of the components actually has non-zero values for the generated metric. Note that in this example, that since in the input metric is error counts, which only are recorded as they occur, rather than at regular intervals, so too is the output metric. Thus, the fade period for this case should be extended so that the color does not fade out between data value occurences.



Prefix	Make/Metric
metric	S2R2_ce_count
metric	S2R2_ch0_ce_count
metric	S2R2_ch1_ce_count
metric	S2R2_ue_count
metric	S2R3_ce_count
metric	S2R3_ch0_ce_count
metric	S2R3_ch1_ce_count
metric	S2R3_ue_count
metric	S2R4_ce_count
metric	S2R4_ch0_ce_count
metric	S2R4_ch1_ce_count
metric	S2R4_ue_count
metric	S3R0_ce_count
metric	S3R0_ch0_ce_count
metric	S320_ch1_ce_count
metric	S3R0_ue_count
metric	S3R1_ce_count
metric	S3R1_ch0_ce_count
metric	S3R1_ch1_ce_count
metric	S3R1_ue_count
metric	S3R2_ce_count
metric	S3R2_ch0_ce_count
metric	S3R2_ch1_ce_count
metric	S3R2_ue_count
metric	S3R3_ce_count
metric	S3R3_ch0_ce_count
metric	S3R3_ch1_ce_count
metric	S3R3_ue_count
metric	S3R4_ce_count
metric	S3R4_ch0_ce_count
metric	S3R4_ch1_ce_count
metric	S3R4_ue_count
metric	MemTotal
metric	MemFree
metric	Buffers
metric	Cached
metric	SwapCached
metric	Active
metric	Inactive
metric	HighTotal
metric	HighFree
metric	LowTotal
metric	LowFree
metric	SwapTotal
metric	SwapFree
metric	Dirty
metric	Writeback
metric	AnonPages
metric	Mapped
metric	Slab
metric	PageTables
metric	NFS_Unstable
metric	Bounce
metric	CommitLimit
metric	Committed_AS
metric	VmallocTotal
metric	VmallocUsed
metric	VmallocChunk
metric	HugePages_Total
metric	HugePages_Free
metric	HugePages_Rsvd
metric	Hugepagesize
metric	DER_S1R3_EDAC_CumulativeErrors
rack	3100
ts	PowerEdge

Start Time
Feb 19, 2009
3:10:13 PM

End Time
Feb 27, 2009
10:30:41 AM

1 selected

erid	jobname	jobstate	partition	timelimit	starttime	endtime	nodelist
	job_578	CANCELLED		5760	2009-02-22 10:21:55	2009-02-24 11:20:54	glory[142-143,152-154]
	job_301	TIMEOUT		5760	2009-02-23 00:53:34	2009-02-27 00:53:37	glory[236,238-240,247,249-252,254-256,277-278,283-284]
	job_253	TIMEOUT		5760	2009-02-23 06:28:33	2009-02-27 06:28:39	glory[213-215,217]
	job_541	COMPLETED		5760	2009-02-23 08:47:53	2009-02-26 13:19:36	glory[195-197]
	job_110	TIMEOUT		5760	2009-02-23 10:29:48	2009-02-27 10:30:41	glory[119,129-132,136-138,141,155,164,174-176,223,225
	job_381	CANCELLED		1440	2009-02-23 11:09:03	2009-02-23 12:30:48	glory[205-207,210-212,224,226]
	job_540	COMPLETED		5760	2009-02-23 11:58:57	2009-02-26 15:15:25	glory[204,221,228]
	job_382	FAILED		240	2009-02-23 12:02:14	2009-02-23 12:02:17	glory234
	job_382	NODE_FAIL		240	2009-02-23 12:03:35	2009-02-23 12:08:24	glory234

Job State

	State	#	%
1	🟨 Timeout	0	0
2	🟧 Node F...	0	0

Time Summary

Description	Value
Average timelimit (h:m:s)	96:00:00
Average actual	

Node Summary

Description	Value
Average nodecount	3.00
Common nodes	204,221,228

Metric Generator

Script: accumulate.pl

Description:
```
Usage accumulate.pl dbname hostname inputMetricTable outputMetricTable starttime endtime
          or accumulate.pl --help for help
          example: accumulate.pl OVIS_Test localhost  MetricCnActiveValues MetricCnAccumulateActiveValues "2009-02-23 12:01:01" "2009-02-28 03:04:05"
```

Parameters: OVIS_Glory_v3Release localhost MetricNodeS1R3_ce_countValues MetricNodeDER_S1R3_EDAC_CumulativeErrorsValues "2009-02-23 11:58:57" "2009-02-26 15:15:25"

Input Metric: node::S1R3_ce_count

Output Metric: node::DER_S1R3_EDAC_CumulativeErrors

Output Storage: float

Figure 29: Metric Generator Pane and resulting Derived Metric in the Metric List and droppped on the 3-D display.

Using the Baron: Analyses

Analyses are done as in the example in §9.1. In general, Click on Analysis in the upper left of the window and an Analysis pane should appear. The default time range is the entire time range in the database. Do the following:

1. Select some analysis type.
2. Drag and drop some metric(s) onto the analysis metric window .
3. Fill in some components, e.g., "node1-300". Components can be specified by either their canonical names or their component type and number (as they are in this case). If they are specified by canonical names, note that ranges are not supported as canonical names can have hyphens within them.
4. Fill in the relevant times.
5. Click on Learn

These steps applied to the Contingency and Multi-variable correlative statistics result in the panes shown in Figures 30 and 31.

9.2 Localhost Demo Files

This section describes two example cases which can be used on your local machine. They further illustrate use of local and remote samplers for core utilization information and `lmsensor` information, as described in §5.2.

Node Level display

This case collects data from your local machine for analysis and display. In the OVIS source data directory is the file `testone.ovdb` which is an OVIS configuration file that can be modified to test collecting and displaying data with OVIS. For example purposes the display shows 3 racks, 2 of which have two nodes each. Only one of the nodes will be used in this example and will collect data from your local machine. The physical display is shown in Figure 32 (top) with the real component upon which data will be displayed popped out in the figure.

To use the localhost demo example:

1. Review the samplers to ensure that the metrics are correct for your system
2. Edit the addresses section of `testone.ovdb` where it is indicated to replace the data in the lines with that corresponding to your local IP address and MAC address
3. `mysql> create database OVIS_Testone`
4. `cd /path/to/ovisBuildDir`

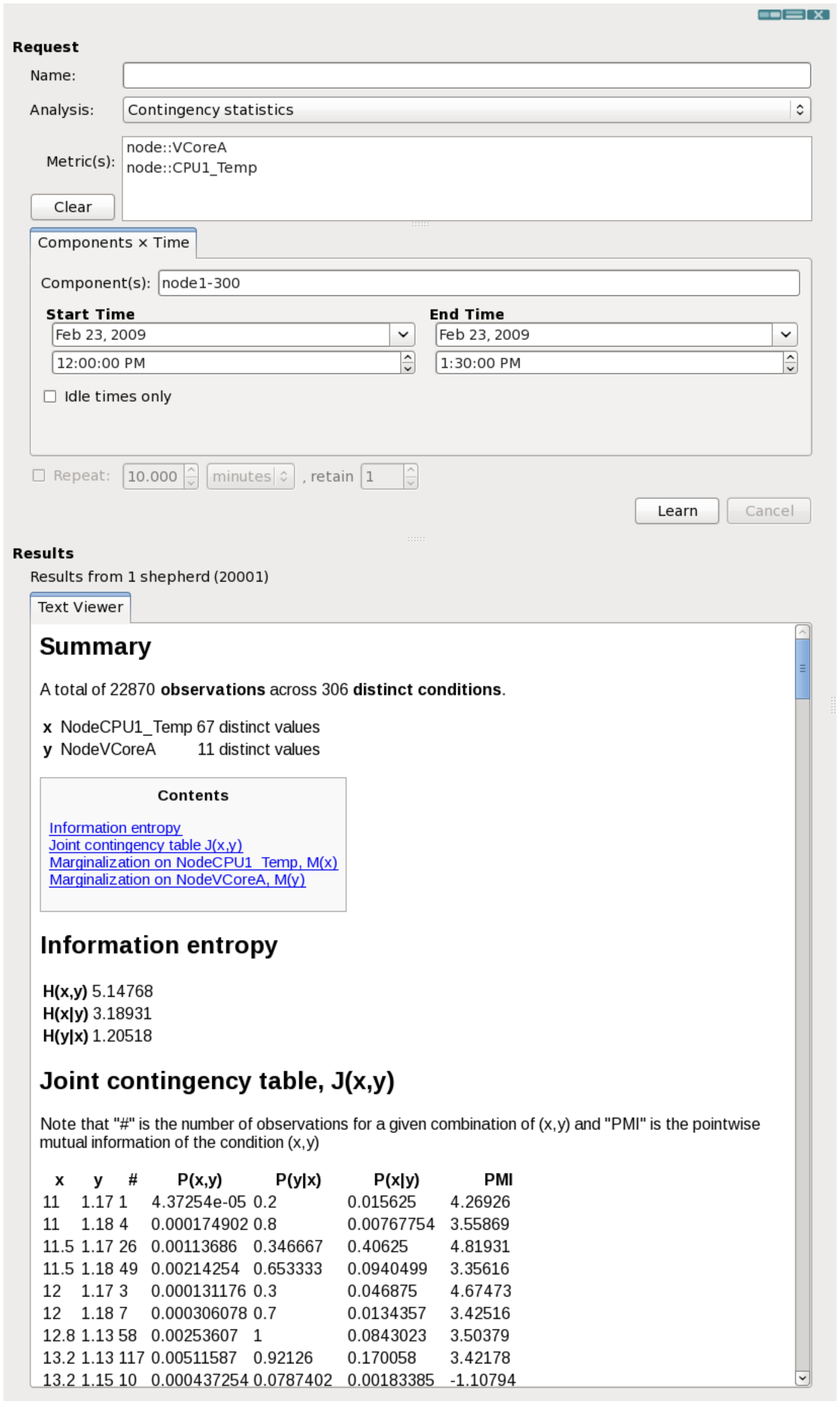
5.
   ```
   ./bin/ovis-db -d -t 16383 -u mysql://ovis@localhost/OVIS_Testone
    -x /path/to/ovisSrcDir/data/testone.ovdb
   ```
   (this will set up the correct tables in your database)
6. In a different window but the same directory run
   ```
   ./bin/shepherd --name=Testone \
     --database=mysql://ovis@localhost/OVIS_Testone
   ```
7. In a different window but the same directory run
   ```
   ./bin/sheep --name=Testone
   ```
8. In a different window but the same directory run
   ```
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path/to/ovisBuildDir/lib
   ./bin/baron
   ```

Core Level display

This case has 1 rack with one node which contains 4 CPUs each of which has 4 cores as shown in Figure 32 (bottom). This illustrates both containment of components and the use of remote samplers. You can use this file in a manner similar to the previous example, but it will only be meaningful if you have a multicore machine.

This case illustrates use of the `ovMetricLinuxProcStatUtilSpecialtySampler` which runs on the node and collecting data either to be displayed on the node (e.g., the overall utilization) or on the cores (e.g., the per core utilization). This is in contrast to the previous example where the `ovMetricLinuxProcStatUtilSpecialtySampler` was used to collect the same information to be drawn to the node level only. Note that in the Node Level display example the per core metrics must all have unique names per core (e.g., CPU0UserPercUtil, CPU1UserPercUtil), where as on the Core Level display, they do not, as they are uniquely identified by the core.

More information on the specification of local and remote association for these examples is given in §5.2.



Summary

A total of 22870 **observations** across 306 **distinct conditions**.

x	NodeCPU1_Temp	67 distinct values
y	NodeVCoreA	11 distinct values

Contents

Information entropy
Joint contingency table J(x,y)
Marginalization on NodeCPU1_Temp, M(x)
Marginalization on NodeVCoreA, M(y)

Information entropy

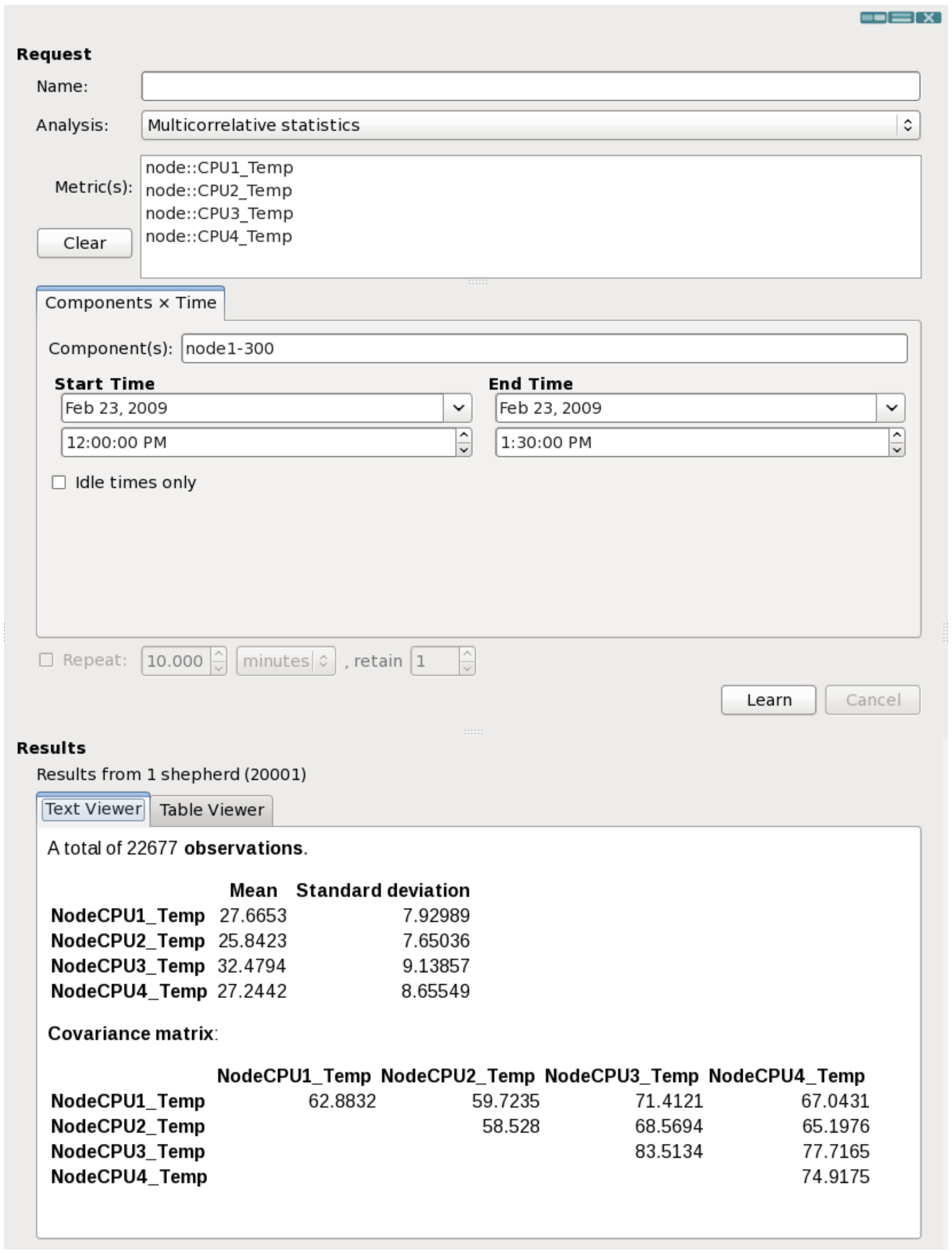
H(x,y) 5.14768
H(x|y) 3.18931
H(y|x) 1.20518

Joint contingency table, J(x,y)

Note that "#" is the number of observations for a given combination of (x,y) and "PMI" is the pointwise mutual information of the condition (x,y)

x	y	#	P(x,y)	P(y|x)	P(x|y)	PMI
11	1.17	1	4.37254e-05	0.2	0.015625	4.26926
11	1.18	4	0.000174902	0.8	0.00767754	3.55869
11.5	1.17	26	0.00113686	0.346667	0.40625	4.81931
11.5	1.18	49	0.00214254	0.653333	0.0940499	3.35616
12	1.17	3	0.000131176	0.3	0.046875	4.67473
12	1.18	7	0.000306078	0.7	0.0134357	3.42516
12.8	1.13	58	0.00253607	1	0.0843023	3.50379
13.2	1.13	117	0.00511587	0.92126	0.170058	3.42178
13.2	1.15	10	0.000437254	0.0787402	0.00183385	-1.10794

Figure 30: Analysis Pane with Contingency Statistics.



Analysis: Multicorrelative statistics

Metric(s): node::CPU1_Temp, node::CPU2_Temp, node::CPU3_Temp, node::CPU4_Temp

Component(s): node1-300

Start Time: Feb 23, 2009 12:00:00 PM

End Time: Feb 23, 2009 1:30:00 PM

Repeat: 10.000 minutes, retain 1

Results from 1 shepherd (20001)

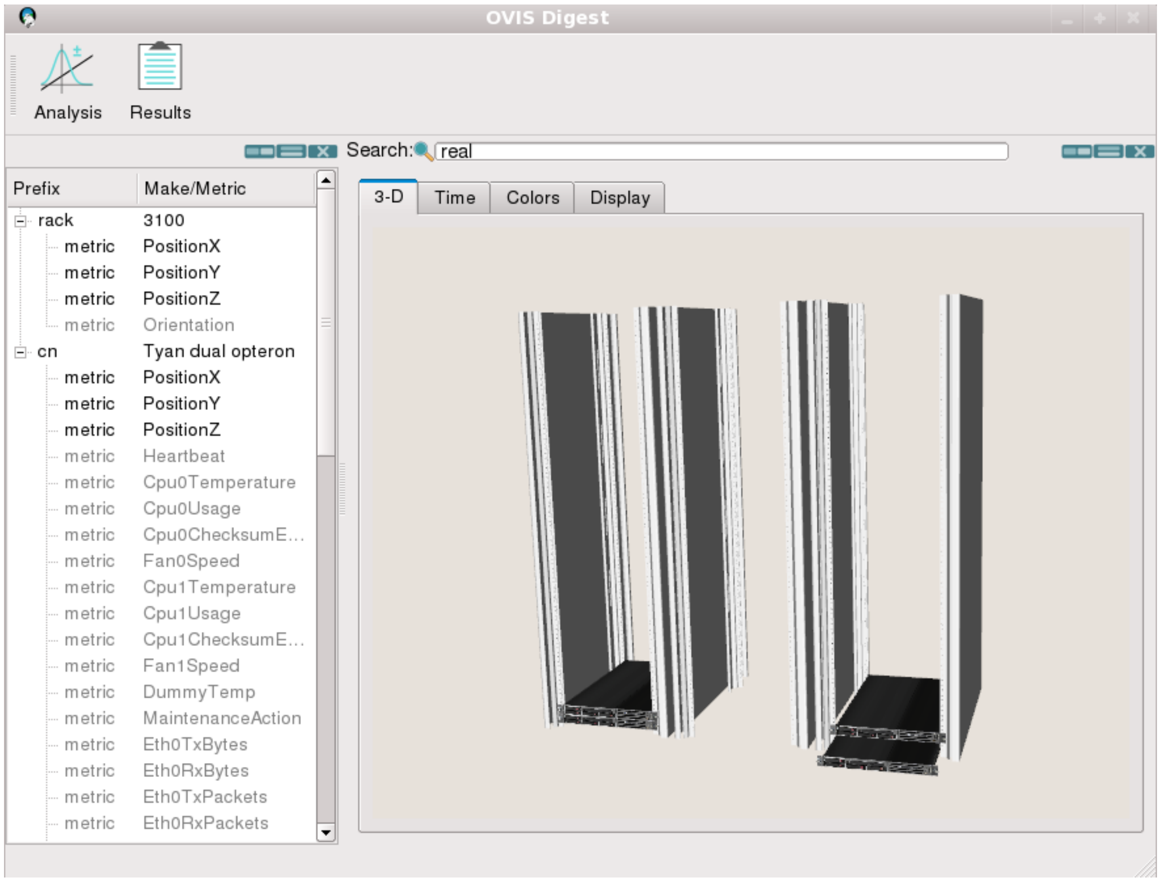
A total of 22677 **observations**.

	Mean	**Standard deviation**
NodeCPU1_Temp	27.6653	7.92989
NodeCPU2_Temp	25.8423	7.65036
NodeCPU3_Temp	32.4794	9.13857
NodeCPU4_Temp	27.2442	8.65549

Covariance matrix:

	NodeCPU1_Temp	**NodeCPU2_Temp**	**NodeCPU3_Temp**	**NodeCPU4_Temp**
NodeCPU1_Temp	62.8832	59.7235	71.4121	67.0431
NodeCPU2_Temp		58.528	68.5694	65.1976
NodeCPU3_Temp			83.5134	77.7165
NodeCPU4_Temp				74.9175

Figure 31: Analysis Pane with Multivariate correlation analysis.



real

Prefix	Make/Metric
rack	3100
metric	PositionX
metric	PositionY
metric	PositionZ
metric	Orientation
cn	Tyan dual opteron
metric	PositionX
metric	PositionY
metric	PositionZ
metric	Heartbeat
metric	Cpu0Temperature
metric	Cpu0Usage
metric	Cpu0ChecksumE...
metric	Fan0Speed
metric	Cpu1Temperature
metric	Cpu1Usage
metric	Cpu1ChecksumE...
metric	Fan1Speed
metric	DummyTemp
metric	MaintenanceAction
metric	Eth0TxBytes
metric	Eth0RxBytes
metric	Eth0TxPackets
metric	Eth0RxPackets

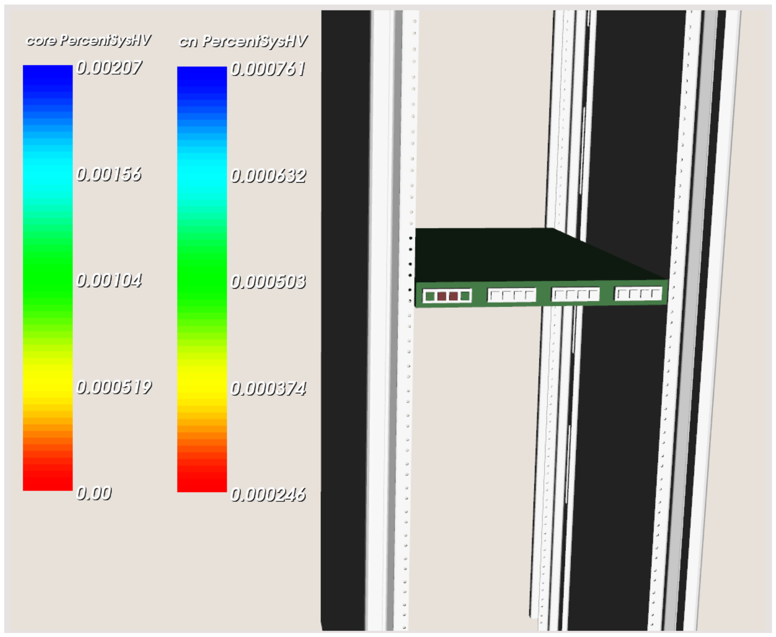


Figure 32: Node Level metric display (Local Association) from the `testone.ovdb` file (top); Core Level metric display (Remote Association) from the `testonecpu.ovdb` file (bottom).

10 Additional Notes and Future Work

This section presents some additional notes and some planned enhancements.

10.1 Analysis

Outlier detection and model drop, which existed in the OVIS 2.x release is temporarily absent in the OVIS 3.2 release. A demo version of the model drop is enabled for Descriptive Statistics only and uses an outlier probabilistic threshold of 2.0 standard deviations. Outlier detection and model drop will be reincorporated in a future OVIS 3.x release. More on these capabilities can be found in [6].

Time series analyses will be included in a future release.

The Repeat Analysis capability (see §8.9) will be included in a future release.

Graphing/plotting capabilities will be included in a future release.

The Bayesian Modeling Analysis, referred to at `ovis.ca.sandia.gov` is not part of the release, and a patent application has been granted on this part of the OVIS work.

10.2 Multiple Shepherds

OVIS 3.2 enables distributed (non-replicated) database back-ends for insertion and analysis. Visualization of the data on the 3-D display currently only works for the database (Shepherd) to which the Baron is connected. Concurrent visualization of data from all Shepherds is in work and will be part of a future 3.X release.

10.3 Data Collection and Insertion

A lighter-weight data collection and insertion methodlogy, which takes the database client off the Sheep has been developed and is in the process of being incorporated into a future OVIS 3.x release.

10.4 Job Search

A more generic database interface will be implemented in a future OVIS release that will enable more generic scheduler/resource manager support.

10.5 Miscellany

OVIS is released open source under BSD license, which allows for the development of platform specific samplers or enhancements to OVIS to be kept for private usage. Please contact `ovis-help@sandia.gov` for more information.

References

[1] SLURM. https://computing.llnl.gov/linux/slurm.

[2] J. Brandt, F. Chen, V. DeSapio, A. Gentile, J. Mayo, P. Pébay, D. Roe, D. Thompson, and M. Wong. Combining virtualization, resource characterization, and resource management to enable efficient high performnce compute platforms through intelligent dynamic resoruce allocation. In *Proc. 24rd IEEE International Parallel & Distributed Processing Symposium (6th Workshop on System Management Techniques, Processes, and Services)*, Atlanta, GA, April 2010.

[3] J. Brandt, F. Chen, V. DeSapio, A. Gentile, J. Mayo, P. Pébay, D. Roe, D. Thompson, and M. Wong. Using cloud constructs and predictive analysis to enable pre-failure process migration in HPC systems. In *Proc. 10th IEEE International Symposium on Cluster, Cloud, and Grid Computing (Workshop on Resiliency in High-Performance Computing in Clusters, Clouds, and Grids)*, Melbourne, Australia, May 2010.

[4] J. Brandt, B. Debusschere, A. Gentile, J. Mayo, P. Pébay, D. Thompson, and M. Wong. OVIS 2: A robust distributed architecture for scalable RAS. In *Proc. 22nd IEEE International Parallel & Distributed Processing Symposium (4th Workshop on System Management Techniques, Processes, and Services)*, Miami, FL, April 2008.

[5] J. Brandt, B. Debusschere, A. Gentile, J. Mayo, P. Pébay, D. Thompson, and M. Wong. Using probabilistic characterization to reduce runtime faults on HPC systems. In *Workshop on Resiliency in High-Performance Computing*, Lyon, France, May 2008.

[6] J. Brandt, A. Gentile, J. Mayo, P. Pebay, D. Roe, D. Thompson, and M. Wong. OVIS 2 user's guide. Sandia Report SAND2009-2329, Sandia National Laboratories, 2009.

[7] J. Brandt, A. Gentile, J. Mayo, P. Pébay, D. Roe, D. Thompson, and M. Wong. Resource monitoring and management with OVIS to enable HPC in cloud computing. In *Proc. 23rd IEEE International Parallel & Distributed Processing Symposium (5th Workshop on System Management Techniques, Processes, and Services)*, Rome, Italy, May 2009.

[8] J. Brandt, A. Gentile, J. Mayo, P. Pebay, D. Thompson, and M. Wong. The Python command line interface to the OVIS analysis functionality. Sandia Report SAND2010-4289, Sandia National Laboratories, 2009.

[9] J. Brandt, V. De Sapio, A. Gentile, J. Mayo, P. Pebay, D. Roe, D. Thompson, and M. Wong. The OVIS analysis architecture. Sandia Report SAND2010-5107, Sandia National Laboratories, 2010.

[10] Kitware, Inc. Visualization Tool Kit (VTK). www.vtk.org.

[11] Sandia National Laboratories. OVIS. ovis.ca.sandia.gov.

[12] TERASCALA. Terascala performance notes. www.terascala.com/pdf/Terascala Performance Notes.pdf. last accessed 2009-04-13.

DISTRIBUTION:

1	MS 0899	Technical Library, 8944 (electronic copy)

Sandia National Laboratories