Version 0.11.0 Release Notes

New Features:

- MCE Log collection in sensor monitoring framework
- Enhancements to ORCM Cluster Configuration File to support 4K nodes
- ORCM Database ability to "purge" the database of out-of-date data.
- Inventory collection Network Interfaces, External PCI Devices and HWLOC new version integration

Open issues:

- ORCM OPAL_PREFIX and LD_LIBRARY_PATH conflicts with openmpi
  Workaround is to not relocate RPM install location
- DB Purge by partition not dropping all the tables
  Workaround is to manually delete tables when needed
- When 2 sensor plugins with use the same BMC errors occur randomly
  Workaround is to not run separate sampling threads targeting the same BMC
- MCE data not being processed when mce log error msg is being read
  Workaround is to limit other kernel messages to syslog

Version 0.11.0 User Guide

1 ORCM

1.1 Background

The Open Resilient Cluster Manager (ORCM) was originally developed as an open-source project (under the Open MPI license) by Cisco Systems, Inc to provide a resilient, 100% uptime run-time environment for enterprise-class routers. Based on the Open Run-Time Environment (ORTE) embedded in Open MPI, the system provided launch and execution support for processes executing within the router itself (e.g., computing routing tables), ensuring that a minimum number of copies of each program were always present. Failed processes were relocated based on the concept of fault groups - i.e., the grouping of nodes with common failure modes. Thus, ORCM attempted to avoid cascade failures by ensuring that processes were not relocated onto nodes with a high probability of failing in the immediate future.

The Cisco implementation naturally required a significant amount of monitoring, and included the notion of fault prediction as a means of taking pre-emptive action to relocate processes prior to their node failing. This was facilitated using an analytics framework that allowed users to chain various analysis modules in the data pipeline so as to perform in-flight data reduction.

Subsequently, ORCM was extended by Greenplum to serve as a scalable monitoring system for Hadoop clusters. While ORCM itself had run on quite a few “nodes” in the Cisco router, and its base ORTE platform has been used for years on very large clusters involving many thousands of nodes, this was the first time the ORCM/ORTE platform had been used solely as a system state-of-health monitor with no responsibility for process launch or monitoring. Instead, ORCM was asked to provide a resilient, scalable monitoring capability that tracked process resource utilization and node state-of-health, collecting all the data in a database for subsequent analysis. Sampling rates were low enough that in-flight data reduction was not required, nor was fault prediction considered to be of value in the Hadoop paradigm.

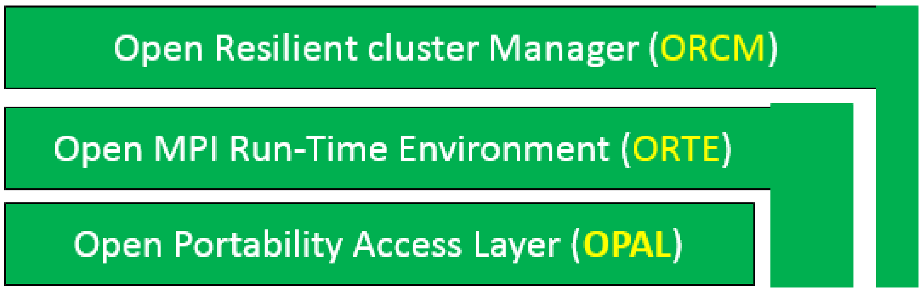
However, data flows did require introduction of an aggregator role. Aggregators absorb the data sent by other nodes and can either store the data in a database, analyze the data, or both. The objective of the aggregator is primarily to concentrate the database operations, thus minimizing the number of active connections to the database itself.

Throughout this time, ORCM has retained ORTE’s ability to perform scalable launch and process monitoring, and ORTE’s support for a variety of scheduling environments. We are now in the process of validating and extending ORCM to provide both monitoring and launch support for exascale environments.

1.2 Overview

1.2.1 Architecture

ORCM (Open Resilient Cluster Manager) is a derivative from Open MPI implementation. ORCM framework consists of the following:


Open Resilient cluster Manager (ORCM)
Open MPI Run-Time Environment (ORTE)
Open Portability Access Layer (OPAL)

1. **ORCM**: Open Resilient Cluster Manager. Provides the following: Resource Management, Scheduler, Job launcher and Resource Monitoring subsystem.

2. **ORTE**: The Open Run-Time Environment (support for different back-end run-time systems). Provides the RM messaging interface, RM error management subsystem, RM routing subsystem and RM resource allocation subsystem.

3. **OPAL**: The Open Portable Access Layer (utility and "glue" code used by ORCM and ORTE). Provides operating system interfaces.

There are strict abstraction barriers in the code between these sections. That is, they are compiled into three separate libraries: liborcm, liborte, and libopal with a strict dependency order: ORCM depends on ORTE and OPAL, and ORTE depends on OPAL.

As such, this code organization more reflects abstractions and software engineering, not a strict hierarchy of functions that must be traversed in order to reach lower layer. For example, ORCM can call OPAL functions directly – it does not have to go through ORTE. Indeed, OPAL has a different set of purposes than ORTE, so it wouldn't even make sense to channel all OPAL access through ORTE. ORCM can also directly call the operating system as necessary.

Here's a list of terms that are frequently used in discussions about the Open MPI code base:

The Modular Component Architecture (MCA) is the foundation upon which the entire Open MPI project is built. It provides all the component architecture services that the rest of the system uses. Although it is the fundamental heart of the system, it's implementation is actually quite small and lightweight – it is nothing like CORBA, COM, JINI, or many other well-known component architectures. It was designed for HPC – meaning that it is small, fast, and reasonably efficient – and therefore offers few services other finding, loading, and unloading components.

1. **Framework**: An MCA framework is a construct that is created for a single, targeted purpose. It provides a public interface that can be used external to framework, but it also provides its own internal services. An MCA framework uses the MCA's services to find and load components at run time – implementations of the framework's interface. An easy example framework to discuss is the MPI framework named "btl", or the Byte Transfer Layer. It is used to sends and receives data on different kinds of networks. Hence, Open MPI has btl components for shared memory, TCP, Infiniband, Myrinetc, etc.

ORCM frameworks

- analytics: Data reduction using user defined workflow
- cfgi: cluster configuration management
- db: Database system
- diag: System Diagnostics
- pvsn: Provisioning system
- scd: Job Scheduler
- Sensor: Sensor data monitoring subsystem
- sst: cluster subsystem initialization

ORTE frameworks

- dfs: Daemon file system operations
- errmgr: Error manager
- ess: Environment specific service
- IOf: I/O forwarding
- filem: Remote file management

- grpcomm: Group communications
- oob: Out-of-band communication
- odls: Daemons local launch subsystem
- plm: Process launch management
- ras: Resource allocation subsystem
- rmaps: Resource mapping subsystem
- routed: Routing table for runtime messaging layer
- rml: Runtime messaging layer (routing of OOB messages)
- rtc: Runtime control
- state: State machine
- snapc: Snapshot coordination interface
- sstore: Distributed stable storage

OPAL frameworks

- allocator: allocate memory
- backtrace: back trace
- btl: Byte transfer layer
- compress: compression framework
- crs: checkpoint and restart
- dstore: database framework for internal storage
- event: event handler
- hwloc: Hardware locality
- if: net if

- installdirs: opal build prefix for install folders
- memchecker: Memory checker
- memcpy: memory copy
- memory: memory hooks
- mpool: memory pool
- pstat: process status
- rcache: Registration cache framework
- shmem: shared memory backing facility
- timer: High-resolution timers
- sec: security authentication/authorization

1. **Component**: An MCA component is an implementation of a framework's interface. Another common word for component is "plugin". It is a standalone collection of code that can be bundled into a plugin that can be inserted into the Open MPI code base, either at run-time and/or compile-time.
2. **Module**: An MCA module is an instance of a component (in the Object Oriented Programming (OOP) sense of the word "instance"; an MCA component is analogous to a "class"). For example, if a node running an Open MPI application has multiple Ethernet NICs, the Open MPI application will contain one TCP btl component, but two TCP btl modules. This difference between components and modules is important because modules have private state; components do not.

Components can be dynamic or static, that is, they can be available as plugins or they may be compiled statically into libraries (e.g., liborcm).

1.2.2 Core Features

- Plugin architecture based on Open MPI's Module Component Architecture (MCA)
  - Sophisticated auto-select algorithms based on system size, available resources
  - Binary proprietary plugin support
  - On-the-fly updates for maintenance*
  - Addition of new plugin capabilities/features without requiring system-wide restart if compatibility requirements are met
- Hardware discovery support
  - Automatic reporting of hardware inventory on startup
  - Automatic updating upon node removal and replacement
- Provisioning support
  - Images provisioned based on user directive prior to launch*
- Scalable overlay network*
  - Supports multiple topologies, including both tree and mesh plugins
  - Automatic route failover and restoration, messages cached pending comm recovery
  - Both in-band and out-of-band transports with auto-failover between them
- Sensors
  - Both push and pull models supported
  - Read as a group at regular intervals according to a specified rate, or individual sensors can be read at their own regular interval, or individual readings of any combination of sensors can be polled upon request

  - Polling requests can return information directly to the caller, or can include the reading in the next database update, as specified by the caller
  - Data collected locally and sent to an aggregator for recording into a database at specified time intervals
  - Environment sensors
    * Processor temperature - on-board sensor for reading processor temperatures when coretemp kernel module loaded
    * Processor frequency - on-board sensor for reading processor frequencies. Requires read access to /sys/devices/system/cpu directory
    * IPMI readings of AC power, cabinet temperature, water and air temperatures, etc.
    * Processor power - reading processor power from on-board MSR. Only supported for Intel SandyBridge, IvyBridge, and Haswell processors
  - Resource utilization
    * Wide range of process and node-level resource utilization, including memory, cpu, network I/O, and disk I/O
  - Process failure
    * Monitors specified file(s) for programmatic access within specified time intervals and/or file size restrictions
  - Heartbeat monitoring
- Analytics*
  - Supports in-flight reduction of sensor data
  - User-defined workflows for data analysis using available plugins connected in user-defined chains
  - Analysis chain outputs can be included in database reporting or sent to requestor at user direction

    - Plugins can support event and alert generation
    - "Tap" plugin directs copied stream of selected data from specified point to remote requestor
    - Chains can be defined at any aggregation point in system
- Diagnostics*
    - Supports system diagnostic capabilities includes memory diagnostic, processor diagnostic and Ethernet diagnostic
    - Launch diagnostic from octl on specific node or a list of nodes
- Database
    - Both raw and processed data can be stored in one or more databases
    - Supports both SQL and non-SQL* databases
        * Multiple instances of either type can be used in parallel*
        * Target database for different data sources and/or types can be specified using MCA parameters during configure and startup, and can be altered by command during operation*
    - Cross-data correlation maintained
        * Relationship between job, sensor, and performance data tracked and linked for easy retrieval*
- Scalable launch
    - Distributed mapping system to minimize data transmission of launch commands*
    - Rapid MPI wireup
        * Endpoint management and support for static endpoints, enabling communication upon init
        * PMIx wireup support for unmanaged environments*
        * Automatic pre-positioning of dynamic libraries*
        * Pre-loading of libraries and data by user directive*
- Fault Tolerance

- Self-healing communication system (see above)
- Non-heartbeat detection of node failures
- Automatic state recovery based on retrieval of state information from peers*
- Support for time-based checkpoint of applications*
- Burst buffer management for rapid checkpoint/restart*

*indicates areas of development

Following are ORCM software tool components:

1. orcmd – An ORCM daemon which runs on all compute nodes with root privileges. This daemon is used for RAS Monitoring data collection (In-Band) sensor data from the compute nodes.
2. orcmd aggregator – Another ORCM daemon (orcmd) to collect OOB and In-Band RAS monitoring data from the compute nodes from a service node (Rack Controller, Row Controller or from a system management server). OOB data collection requires an IPMI access to BMC on the compute nodes. In-Band data collection goes through a management network. The orcmd is same as above with role assigned as aggregator in the ORCM configuration file (orcm-site.xml).
3. orcmsched – A scheduler daemon runs on the SMS node or Head node. This daemon is used for managing the compute resources in a cluster, allocating resources for a user session request and managing a queue for session allocation requests.
4. osub – A command line tool for requesting a resource allocation from the scheduler.
5. orun – Job launch command line tool for requesting allocation and launching user jobs.
6. orcmsd – A session daemon launched by the scheduler for the allocated sessions. These session daemons are launched with user privileges and it stays up for the duration of the allotted time or until job process completes execution and releases the allocation.
7. octl – An administrative command line interface

8. ocli – A user command line interface

1.3 Database

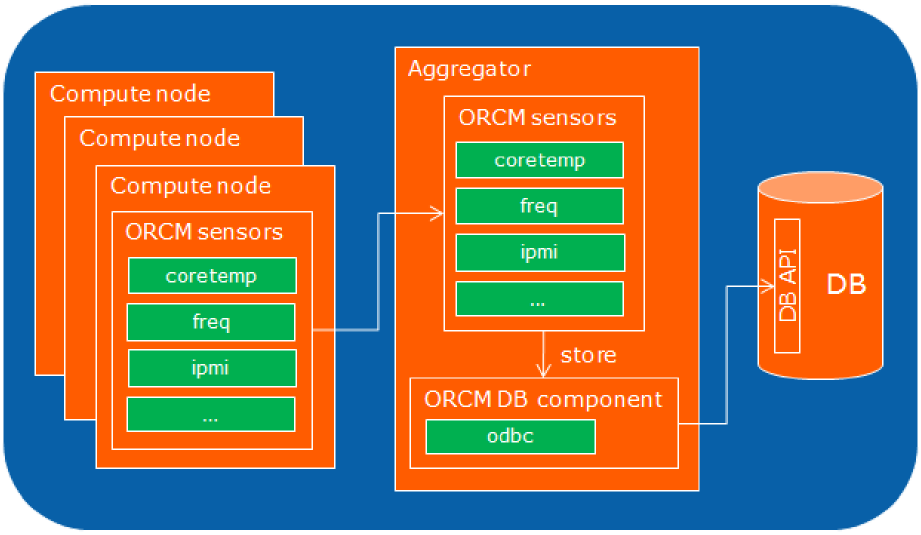
The database is a key component for supporting the various applications of the cluster software stack. So it is just as important to ORCM as it provides support for its system and resource management tasks by storing hardware inventory, system environmental data, RAS events (including error conditions) and job accounting data. Furthermore, the data stored in the database may also be useful to other applications, for example: cluster monitoring, system diagnostics and operator interface.

Finally, the database is not meant to be just a repository for all the data. It should provide an API to abstract some of the low-level data storage details and the schema to provide applications independence from the data layer, thus providing flexibility and making the database and application maintenance easier.

1.4 RAS Monitoring

RAS monitoring encompasses the collection of all metrics & sensor related data from each node. It is primarily implemented under the 'sensor' framework present under the orcm project. It contains several plugins that monitor various metrics related to different features present in each node. These metrics range from sensor related 'tangible' information like Temperature, Voltage, Power Usage, etc. to non-tangible metrics related to OS parameters like, Memory Usage, Disk Usage, Process information, file monitoring, etc.

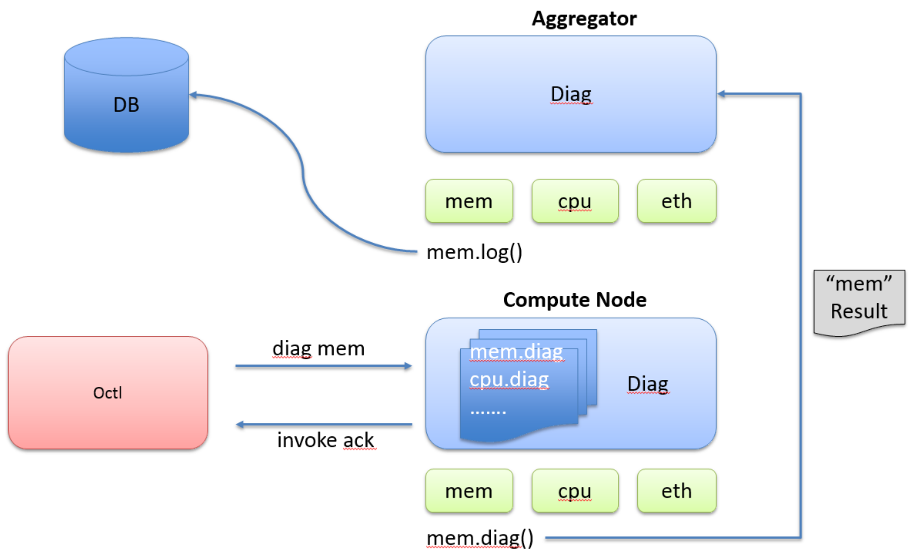
The RAS Monitoring in ORCM relies on the the Aggregator-Compute_Node model. All the sensor components running in each compute node scan the system and record metrics and send it to the aggregator, which acts as a collector and filters the data to be logged onto the database. A single exception to this is the IPMI plugin - contrary to running in the compute nodes and collecting the data, it runs actively only in the aggregator node, and collects the remote BMC's sensor data by issuing IPMI-over-LAN commands.



1.5 Diagnostics

System level diagnostics will detect and report failures of critical resources, including memory, processors, network paths and I/O interfaces. The diagnostic routines will be capable of isolating hardware problems down to the Field Replaceable Unit (FRU) level in both the system and its peripheral equipment. The ORCM diagnostics framework supplies a consistent set of APIs for invoking the desired diagnostics functionality, while hiding the complexity of the implementation.

Normally, diagnostics can be launched by the system administrator via the *octl* tool. The *octl* diagnostics commands can invoke the desired diagnostic tests on a specific node or a list of nodes. The actual diagnostic tests will be executed on the compute node(s) quietly. Once the diagnostic is completed, compute nodes will send their result to the aggregator node, which logs the data into database. The diagnostic result logging feature is still under development.



1.6 Admin RAS Policy

System admins can define what constitutes a RAS event, assign a priority to the event, and what info is to be included in the event report. This is to be given to the notifier framework, and we need to verify that the syslog component in that framework properly writes the result in a machine-readable format into the syslog. Event definitions must support level checks (both high and low) on a specific sensor with leaky-bucket data smoothing algorithm, for example, if M measurements are outside bounds in a time window.

A typical admin RAS event policy includes:

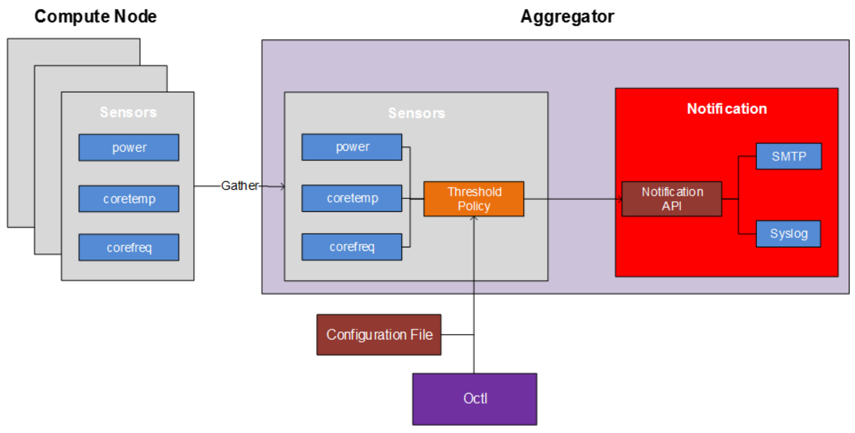
- Sensor name: coretemp and corefreq are supported as for now
- Threshold value: threshold boundary of sensor measurement
- Threshold type: high or low
- Max count: maximum number of measurement are out of bounds in a defined time window
- Time window: time window of the leaky bucket
- Severity: severity/priority of the RAS event that is triggered by the policy; severity level follows RFC 5424 syslog protocol
- Notification mechanism: notification mechanism of the event, syslog or SMTP email

Example of coretemp policy as below:

```
sensor_coretemp_threshold=100:hi:2:60:crit:syslog;70:lo:4:60:warn:syslog
```

Examples of event notification in Syslog as below:

- Mar 18 14:23:35 leo-dev1 OpenRTE Error Report:[22448]: [Wed Mar 18 14:23:35 2015#012][[0,0],1] SET EVENT : host: dev1 core 6 temperature 43.000000 °C, higher than threshold 40.000000 °C for 4 times in 30 seconds
- Mar 18 14:29:31 leo-dev1 OpenRTE Error Report:[22478]: [Wed Mar 18 14:29:31 2015#012][[0,0],1] SET EVENT : host: dev1 core 0 freq 3.326000 GHz, higher than threshold 3.000000 GHz for 4 times in 30 seconds



Users can pass RAS event policies as MCA parameters when ORCM daemon starts as below:

```
orcmd --omca sensor heartbeat,coretemp --omca sensor_coretemp_policy 30:lo:3:60:crit:syslo
```

or

```
orcmd --omca sensor heartbeat,freq --omca sensor_freq_policy 3.0:hi:3:60:crit:syslog
```

Admins can also use octl command line tool to view, add or update RAS event policies at the admin level.

The octl policy command allows setting admin event policies on remote nodes. These commands require a node regex specification for determining which remote nodes to run on. See the 3.3 ORCM Node Regular Expressions section for details on how to construct the regex.

Example 1: show RAS event policies on node001

```
% octl sensor get policy node001
ORCM getting sensor policy from Node:node001
Sensor    Threshold   Hi/Lo   Max_Count/Time_Window   Severity   Action
------------------------------------------------------------------------------
coretemp    30.000      Lo      3 in   60 seconds      WARN      syslog
coretemp   100.000      Hi      2 in   60 seconds     ALERT      syslog
corefreq     3.000      Hi      3 in   60 seconds      CRIT      syslog
```

Example 2: set coretemp high critical admin policy on node001 through node010

```
% octl sensor set policy node[3:1-10] coretemp 80 hi 3 30 crit syslog
Success
```

1.7 ErrorManager Notification

ORCM daemon OOB communication failures and any internal system errors will not terminate the ORCM daemon instead the error are handled by the Error Manager and reported to the notification method specified by the admin user based on the severity.

The Error Manager framework provides a common API for reporting system errors.

```
ORTE_FORCED_TERMINATE(errorno)
```

This API is a macro which is used in places where there are communication errors and internal errors. This macro sets the job error state in the state machine and the Error Manager registers a callback function to handle the error state.
Majority of the internal errors are reported as NULL state errors and this is handled in the Error Manager call back function to report this to the Notifier with critical severity.
Callback function is defined below:

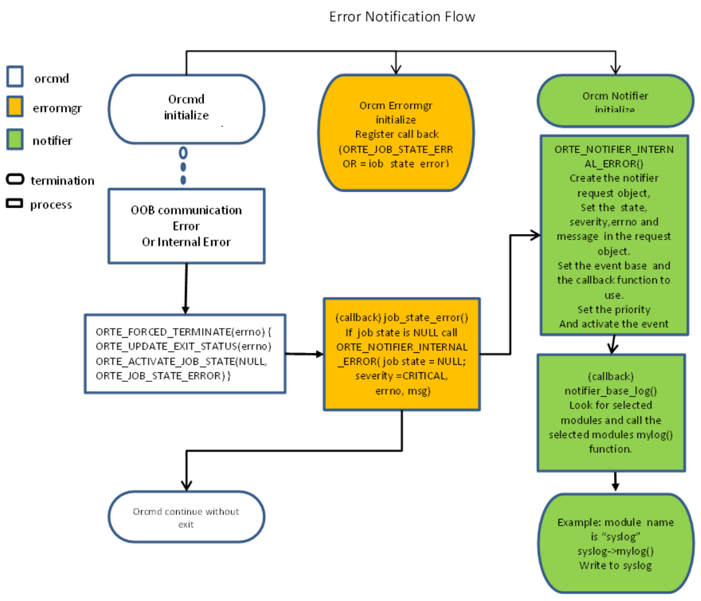
```
orte/mca/errormgr/orcm/errmgr_orcm.c: job_errors()
```

The Notification framework provides APIs for logging errors and events to the user select-able logging mechanism.

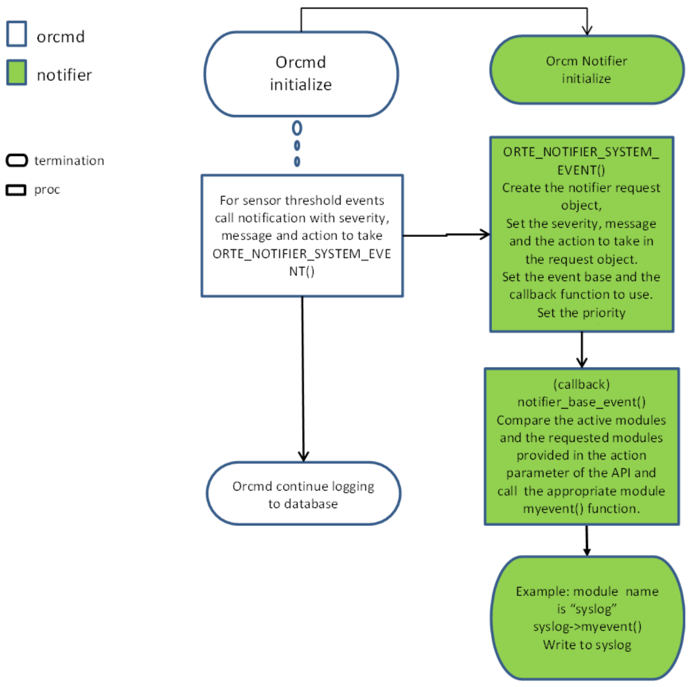
```
ORTE_NOTIFIER_INTERNAL_ERROR(Jdata, jobstate, severity, errorno, message);
jdata -  ORCM internal jdata object;
jobstate  - ORCM job state
severity - critical, emergency, warning, notice, etc.
    errorno - ORCM error code
message -  A descriptive error message to report to syslog.

ORTE_NOTIFIER_SYSTEM_EVENT(severity, message, action);
severity - critical, emergency, warning, notice, etc
message -  A descriptive error message to report to syslog.
action - log options  example syslog,smtp.
```

Error Notification Flow


Error Notification Flow
orcmd
errormgr
notifier
termination
process
Orcmd initialize
Orcm Errormgr initialize Register call back (ORTE_JOB_STATE_ERROR = job_state_error)
Orcm Notifier initialize
OOB communication Error Or Internal Error
ORTE_NOTIFIER_INTERNAL_ERROR() Create the notifier request object, Set the state, severity,errno and message in the request object. Set the event base and the callback function to use. Set the priority And activate the event
ORTE_FORCED_TERMINATE(errno) { ORTE_UPDATE_EXIT_STATUS(errno) ORTE_ACTIVATE_JOB_STATE(NULL, ORTE_JOB_STATE_ERROR) }
(callback) job_state_error() If job state is NULL call ORTE_NOTIFIER_INTERNAL_ERROR(job state = NULL; severity =CRITICAL, errno, msg)
(callback) notifier_base_log() Look for selected modules and call the selected modules mylog() function.
Orcmd continue without exit
Example: module name is "syslog" syslog->mylog() Write to syslog

Event Notification Flow


orcmd
notifier
termination
proc
Orcmd initialize
Orcm Notifier initialize
For sensor threshold events call notification with severity, message and action to take ORTE_NOTIFIER_SYSTEM_EVENT()
ORTE_NOTIFIER_SYSTEM_EVENT() Create the notifier request object, Set the severity, message and the action to take in the request object. Set the event base and the callback function to use. Set the priority
(callback) notifier_base_event() Compare the active modules and the requested modules provided in the action parameter of the API and call the appropriate module myevent() function.
Orcmd continue logging to database
Example: module name is "syslog" syslog->myevent() Write to syslog

Severity Levels

```
ORTE_NOTIFIER_EMERG  -  Emergency (highest level of severity)
```

```
ORTE_NOTIFIER_ALERT  - Alert
ORTE_NOTIFIER_CRIT   -  Critical
ORTE_NOTIFIER_ERROR  - Error
ORTE_NOTIFIER_WARN   -  Warning
ORTE_NOTIFIER_NOTICE - Notice,
ORTE_NOTIFIER_INFO   - Information only,
ORTE_NOTIFIER_DEBUG  - Debug
```

Logging Options

```
syslog  - system log
Other options not yet implemented
smtp - email option
File - user defined file
```

The Notification framework provides MCA parameters to select more than one logging method for error and event notification.
For event notification, the framework provides methods to select notification options by user by work flow.
Admin users can register notification method to log the critical errors to syslog or receive an alert via email to take further actions.

Following are the Notifier framework MCA parameters:

```
NOTIFIER_BASE_USE_PROGRESS_THREAD - To use a dedicated progress thread for notification [Def
NOTIFIER_BASE_SEVERITY_LEVEL - To report all events at or above this severity [Default is err
NOTIFIER_BASE_DEFAULT_ACTION - To report all events to the default action [Default is syslog
```

Optional MCA parameters:

```
NOTIFIER_BASE_EMERG_EVENT_ACTION - To report emergency events to the specified action [examp
NOTIFIER_BASE_ALERT_EVENT_ACTION - To report alert events to the specified action [example:
NOTIFIER_BASE_CRITI_EVENT_ACTION - To report critical events to the specified action [exampl
NOTIFIER_BASE_WARN_EVENT_ACTION - To report warning events to the specified action [example:
NOTIFIER_BASE_NOTICE_EVENT_ACTION - To report notice events to the specified action [example
NOTIFIER_BASE_INFO_EVENT_ACTION - To report Information events to the specified action [exam
NOTIFIER_BASE_DEBUG_EVENT_ACTION - To report debug events to the specified action [example:
NOTIFIER_BASE_ERROR_EVENT_ACTION - To report error events to the specified action [example:
```

2 Build and Installation Guide

2.1 ORCM Build and Installation

2.1.01 Build From SRPMS

Note: See [[2.1.05-Build-Dependencies]] if you have not previously set up your system for running ORCM.

Download the source rpms and run ‘rpmbuild’ command to build binary rpms for your system. For example,

```
shell$ echo "%_topdir $HOME/rpmbuild" > $HOME/.rpmmacros # if you have not already setup a .r
shell$ rpmbuild  --rebuild open-rcm-<version>.src.rpm
```

This will create the following folder in users home directory

```
/home/<user>/rpmbuild
SRPM
RPM
BUILD
BUILDROOT
SPEC
```

Install the rpm:

```
shell$ cd /home/<user>/rpmbuild/RPMS/x86_64/
shell$ sudo rpm – ivh open-rcm-<version>.rpm
```

This will install ORCM into

```
/opt/open-rcm/
```

You may want to set your path:

```
PATH=/opt/open-rcm/bin:$PATH:
export PATH
```

2.1.02 Relocate RPM Install Path

To install rpm and to relocate the binaries to a different folder add the –relocate switch. Run the following command as root.

```
shell$ rpm -ivh openrcm-<version>.rpm --relocate /opt/openrcm=<new location>
```

The following environment variables will need to be set to the new location:

```
export OPAL_PREFIX=<new_location>
export OPAL_LIBDIR=<new_location>/lib64
export OPAL_DATADIR=<new_location>/share
export LD_LIBRARY_PATH=<new_location>/lib64:$LD_LIBRARY_PATH
export PATH=<new_location>/bin:$PATH:
```

2.1.03 Build From Source Tar Files

Please review the sections on "Build Dependencies" and "Pre-build Configurations" before actually starting to build.

The recommended install route is to build and install from the source RPM. However, you can build directly from the source tar file as well.

Download the the source tar files gzip or bzip and use 'tar xf' to extract the files. It is recommended to always specify a prefix when building. Platform files can be used for managing sets of options for targets.

```
shell$ tar xvfz open-rcm-<version>.tar.gz or
shell$ tar xvfj open-rcm-<version>.tar.bz2

shell$ cd open-rcm-<version>
shell$ ./configure [--prefix <install_folder>] \
                   [--with-platform=<platform-file>]
shell$ make all
```

Run as root to install the binaries if current user does not have write permissions to prefix directory

```
shell$ make install
```

By default the tar file based install will go into /usr/local/lib/ and /usr/local/bin/.

The RPM spec.in file stores the default configure switches that are used when building the source RPM and can be used as a reference when building from the tar file. That file can be found here:

- contrib/dist/linux/open-rcm.spec.in

The recommended minimum settings when running configure on tar file based installs:

```
--prefix=/opt/open-rcm
--with-platform=contrib/platform/intel/hillsboro/orcm-linux
```

These are addtional configure options that also can be used but are not required:

```
--exec-prefix=/opt/open-rcm
--bindir=/opt/open-rcm/bin
```

```
--sbindir=/opt/open-rcm/sbin
--sysconfdir=/opt/open-rcm/etc
--datadir=/opt/open-rcm/share
--includedir=/opt/open-rcm/include
--libdir=/opt/open-rcm/lib64
--libexecdir=/opt/open-rcm/lib
--localstatedir=/var
--sharedstatedir=/opt/open-rcm/com
--mandir=/opt/open-rcm/share/man
--infodir=/opt/open-rcm/share/info
```

2.1.04 Build From GitHub Repo

Please review the sections on “Build Dependencies” and “Pre-build Configurations” before actually starting to build.

The recommended install route is to build and install from the source RPM. However, you can build directly from the GitHub repo as well.

Following GNU build tools minimum versions are requirement for configuring and building ORCM from the GitHub development repo.

- autoconf: 2.69 from https://www.gnu.org/software/autoconf/
- automake: 1.12.2 from http://www.gnu.org/software/automake/
- libtool: 2.4.2 from https://www.gnu.org/software/libtool/

It is recommended to download and install the .tar.gz file of each, untar and run the following:

```
shell$ ./configure [--prefix <install_folder>]
shell$ make
```

Run the following as root user if current user does not have write permissions to install folder

```
shell$ make install
```

Building from repo also requires:

```
flex libnl libtool-ltdl m4 patch xz
```

Following are the steps for building ORCM from the github repo. Before building create an account in the github. Clone the ORCM repo from the github branch.

```
shell$ git clone https://github.com/open-mpi/orcm
-or-
shell$ git clone https://<username>@github.com/open-mpi/orcm

shell$ cd orcm
shell$ ./autogen.pl
shell$ mkdir build
      Note: User can create the build folder where ever they want
```

```
shell$ cd build
shell$ ../configure \
     --with-platform=../contrib/platform/intel/hillsboro/orcm-linux \
        [--prefix=<install_folder>]
```

The file pointed to by –with-platform= holds a list of configure options. See configure –help for the full list of options.

```
shell$ make
```

Run the following as root user if current user does not have write permissions to install folder

```
shell$ make install
```

This will install the files under /usr/local unless the optional (but recommended) prefix is used.

The RPM spec.in file stores the default configure switches that are used when building the source RPM and can be used as a reference when building directly from the GitHub repo. That file can be found here:

- contrib/dist/linux/open-rcm.spec.in

The recommended minimum settings when running configure on GitHub based installs:

```
--prefix=/opt/open-rcm
--with-platform=contrib/platform/intel/hillsboro/orcm-linux
```

These are additional configure options that also can be used but are not required:

```
--exec-prefix=/opt/open-rcm
--bindir=/opt/open-rcm/bin
--sbindir=/opt/open-rcm/sbin
--sysconfdir=/opt/open-rcm/etc
--datadir=/opt/open-rcm/share
--includedir=/opt/open-rcm/include
--libdir=/opt/open-rcm/lib64
--libexecdir=/opt/open-rcm/lib
--localstatedir=/var
--sharedstatedir=/opt/open-rcm/com
--mandir=/opt/open-rcm/share/man
--infodir=/opt/open-rcm/share/info
```

2.1.05 Build Dependencies

2.1.5.1 Sigar and SSL

2.1.5.1.1 CentOS

```
shell$ yum install libtool-ltdl openssl openssl-devel
shell$ yum install sigar sigar-devel
```

2.1.5.2 Dependencies for sensor framework IPMI component IPMI Util – IPMI util library is a dependency for including IPMI plugin in the sensor framework. IPMI Util libraries is for accessing Base board management controller (BMC) on compute nodes and for collecting OOB RAS monitoring data from the compute nodes. Aggregator node will need this installed for OOB access to the compute node BMC's.

You can grab the 2.9.4 version here:

```
wget -nd --reject=*.html* --no-parent -r \
   http://ipmiutil.sourceforge.net/FILES/ipmiutil-2.9.5-1.src.rpm
rpm -hiv ipmiutil-2.9.5-1.src.rpm
## The above step will install the ipmiutil source code and a spec file
## in a "rpmbuild" directory and store it in the home directory.

## Add the configure option "--enable-libsensors" to the configure command
## in the ipmiutil.spec file

rpmbuild -bb SPECS/ipmiutil.spec
rpm -ivh ~/rpmbuild/RPMS/x86_64/ipmiutil-2.9.5-1.el6.x86_64.rpm
rpm -ivh ~/rpmbuild/RPMS/x86_64/ipmiutil-devel-2.9.5-1.el6.x86_64.rpm
```

2.1.5.3 Database Dependencies The database is an optional component for logging RAS data. At the moment, ORCM provides support for the PostgreSQL DBMS. There are two options (components) for enabling database support:

- PostgreSQL native client library
- ODBC

To build ORCM with database support connecting via the PostgreSQL native client library, the following dependencies are required:

- PostgreSQL development package

- PostgreSQL client package (installed as a dependency of the development package)

- PostgreSQL shared libraries (installed as a dependency of the client package)

NOTE: PostgreSQL usually installs to its own directory, so when configuring ORCM make sure to specify the directory prefix for the include files and libraries via the "with-postgres" parameter (whether through the command line or the configuration file).

To build ORCM with database support connecting via the ODBC interface, the following dependencies are required:

- unixODBC development package

- unixODBC (installed as a dependency of the development package)

- PostgreSQL ODBC driver

Installation instructions for these components depends on the Linux distribution. For detailed instructions for the PostgreSQL packages, please refer to: PostgreSQL installation wiki.

For more information on unixODBC, please refer to: unixODBC.

For more details on the database installation itself, please refer to: [[2.2-Database-Installation]].

2.1.06 Pre build Configuration

Following three files are used for configuring ORCM:

1. A platform file with options to include in the ORCM build
   (ex: orcm-linux)

2. A default MCA parameter file for selecting default MCA parameters for ORCM
   (ex: orcm-linux.conf)

3. A ORCM site configuration XML file which configures the cluster nodes, aggregator, scheduler and the compute nodes
   (ex: orcm-linux.xml)

2.1.07 Post build Configuration

Following are the instructions for configuring ORCM in a cluster. This may require root privileges depending on your installation folders.

```
shell$ vi /opt/open-rcm/etc/orcm-site.xml
```

Configure node names for aggregator, scheduler and compute nodes to be included in the ORCM cluster. Example:

```
<?xml version="1.0"?>
 <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>


 <configuration>

 <orcm-aggregators>
   <nodes>
     <value>aggregator-hostname</value>
   </nodes>
   <port>
     <value>55805</value>
   </port>
   <mca-params>
    <value>sensor_base_sample_rate=5,sensor_base_log_samples=1</value>
   </mca-params>
 </orcm-aggregators>

 <orcm-daemons>
   <nodes>
     <value>CN-hostname-01</value>
     <value>CN-hostname-02</value>
     <value>CN-hostname-03</value>
     <value>CN-hostname-04</value>
   </nodes>
   <port>
     <value>55810</value>
   </port>
   <mca-params>
    <value>sensor_base_sample_rate=5,sensor_base_log_samples=1</value>
   </mca-params>
 </orcm-daemons>

 <orcm-schedulers>
  <description>Identify the node that houses the ORCM scheduler. Only
```

```
   one allowed at this time</description>
   <nodes>
     <value>scheduler-hostname</value>
   </nodes>
   <port>
     <value>55820</value>
     <description>Port used by orcm scheduler</description>
   </port>
   <mca-params>
    <description>List of MCA params to be used by scheduler</description>
   </mca-params>
 </orcm-schedulers>

 </configuration>
```

NB: This file must be identical across all nodes of the cluster.

2.1.08 Setting Up pre requisites for RAS Monitoring

We need to enable the Baseboard Management Controller (BMC) and configure the IPMI libraries for some of the functionality under RAS monitoring. Especially to enable the "**ipmi**" plugin. Following are the instructions for setting up BMC in BIOS for compute nodes and service nodes (aggregator):

Enable BMC in BIOS, how to do this is specific to your BIOS. Refer to your server manual for guidance.

These instructions work on our specific development machine:

```
In BIOS settings, go to Server Management tab:
1. Enable Plug & Play BMC Detection.
2. Select BMC Lan Configuration
    * Select IP source as dynamic
    * Scroll down, select user Id, and select root
    * Enable User Status.
    * Select Password, and enter a password
```

Once BMC is enabled, you can optionally configure it within Linux using ipmitool:

```
for module in ipmi_devintf ipmi_si ipmi_msghandler; do
    /sbin/modprobe $module
done

ipmitool lan set 1 ipsrc static
ipmitool lan set 1 ipaddr <IP>
ipmitool lan set 1 netmask <NETMASK>
ipmitool lan set 1 defgw ipaddr <GATEWAY>
```

Check settings with:

```
ipmitool lan print 1
```

Setup user:

```
ipmitool user set name 2 <user>
ipmitool user set password 2 <password>
ipmitool channel setaccess 1 2 link=on ipmi=on callin=on privilege=4
ipmitool user enable 2
```

If users need to use the “coretemp” plug-in they need the following kernel module on all nodes:

```
sudo modprobe coretemp
```

If users need to use the “componentpower” plug-in they need the following for setting up MSR on compute nodes and aggregator nodes:

```
ls -la /dev/cpu/<cpu #>/msr
```

If the “msr” file is not present, please load MSR module by running:

```
sudo modprobe msr
```

2.1.09 Startup Instructions

ORCM Systemd/SysV Init

Systemd starting ORCM starting scheduler

```
% systemd start orcmsched
```

starting aggregator/daemon

```
% systemd start orcmd
```

Systemd stopping ORCM stopping scheduler

```
% systemd stop orcmsched
```

stopping aggregator/daemon

```
% systemd stop orcmd
```

SysV Init starting ORCM starting scheduler

```
% service orcmsched start
```

starting aggregator/daemon

```
% service orcmd start
```

SysV Init stopping ORCM stopping scheduler

```
% service orcmsched stop
```

stopping aggregator/daemon

```
% service orcmd stop
```

Systemd/SysV Init configuration settings Scheduler runtime settings can be changed in

```
/etc/sysconfig/orcmsched
```

Aggregator and Daemon runtime settings can be changed in

```
/etc/sysconfig/orcmd
```

Manual Head Node (HN) and Compute Node (CN) Setup

If you relocated ORCM without using configure switches to specify locations you will need to set environment variables on all nodes. See

- https://github.com/open-mpi/orcm/wiki/2.1.02-Relocate-RPM-Install-Path

Start 'orcmsched' as root daemon on the SMS –
system management server (or on a head node for small scale clusters)

```
shell$ orcmsched
[SMS-linux:96034] Sun Aug  3 23:39:11 2014: ORCM SCHEDULER [[0,0],0] started
```

Start 'orcmd' as a root daemon on the aggregator nodes
(on a small size cluster, aggregator can run on head node alongside scheduler).

```
shell$ orcmd 2>&1 | tee orcmd-log.txt
```

You should see output lines that include both general system information, and other lines that include temperature info; e.g:

```
shell$ orcmd
[AGGREGATOR-linux:96071] Sun Aug  3 23:40:00 2014: ORCM aggregator [[0,0],1] started
2014-08-03 23:40:05-0700,AGGREGATOR-linux,41.000000,39.000000,35.000000, ...
2014-08-03 23:40:10-0700,CN-linux-01,41.000000,40.000000,36.000000, ...
```

Start 'orcmd' as a root daemon on the compute nodes.
On ORCM Compute Node 1:

```
shell$ orcmd
[CN-linux-01:26355]
******************************
Mon Aug  4 00:35:04 2014: ORCM daemon [[0,0],2] started and connected to aggregator [[0,0],1]
My scheduler: [[0,0],0]
My parent: [[0,0],1]
******************************
```

On ORCM Compute Node 2:

```
shell$ orcmd
[CN-linux-02:26355]
******************************
Mon Aug  4 00:35:04 2014: ORCM daemon [[0,0],3] started and connected to aggregator [[0,0],1]
My scheduler: [[0,0],0]
My parent: [[0,0],1]
******************************
```

On ORCM Compute Node 3:

```
shell$ orcmd
[CN-linux-03:26355]
******************************
Mon Aug  4 00:35:04 2014: ORCM daemon [[0,0],3] started and connected to aggregator [[0,0],1]
My scheduler: [[0,0],0]
My parent: [[0,0],1]
******************************
```

SSH Environment Setup for CNs

The ssh client and sshd needs to be setup to pass in the ORCM environment variables.

Here is an example using pexec:

```
vi /etc/ssh/sshd_config
  AcceptEnv OPAL_PREFIX OPAL_LIBDIR LD_LIBRARY_PATH
vi /etc/ssh/ssh_config
  SendEnv OPAL_PREFIX OPAL_LIBDIR LD_LIBRARY_PATH

pexec -Ppm 'node[01-32]' --scp '/etc/ssh/sshd_config' %host%:/etc/ssh/.
pexec -Ppm 'node[01-32]' --ssh 'service sshd reload'
# Copy over the ORCM release
pexec -Ppm 'node[01-32]' --rsync '/opt/open-rcm' %host%:/opt/.

# After starting ORCM on HN, start on the CNs
pexec -Ppm 'node[01-32]' --ssh '/opt/open-rcm/bin/orcmd'
```

2.1.10 Troubleshooting

Problems with Yum: Note: Yum installation requires privileged access.

Do you have an accessible yum repository? Check in /etc/yum.repos.d/rhel-source.repo: If the RedHat repos are unavailable, you can try to add an alternate, e.g.:

```
[centos]
name=Centos
baseurl=http://vault.centos.org/6.4/os/x86_64/
enabled=1
gpgcheck=0
```

Problems running orcmd: If you get an error such as the following after completing the build process:

```
shell$ orcmd
[chris-linux-01:11937] [[0,0],INVALID] ORTE_ERROR_LOG: Not found in file
../../../../orcm/mca/cfgi/base/cfgi_base_fns.c at line 764
```

Check that you have the correct orcm-site.xml in <install_folder>/etc. The 'make install' process overwrites this file with a default version.

Problems with orcmd startup: Add verbosity switches

```
--omca mca_base_verbose 100
--omca oob_base_verbose 100
--omca db_base_verbose 100
--omca sensor_base_verbose 100
```

Duplicate key error when storing sensor data to the database If ORCM is unable to store sensor data because an error message is returned from the database explaining that a duplicate key violates a unique constraint, this most likely means there is an issue with that data that is being collected, possibly due to an incorrect configuration on one or more nodes. For example, check and make sure that all the nodes on the system have unique hostnames. If multiple nodes have the same hostname, when trying to store sensor data from these nodes, it will appear that same node is trying to store multiple values for the same metric at the same time (thus generating a unique key violation in the database).

Unusual errors when trying to store data to the database If ORCM is able to connect to the database, but is unable to store some or any data to the database and returns unusual database errors (e.g. errors that indicate that a certain table or procedure does not exist), this most likely means that there is a version mismatch between ORCM and the schema. Please make sure that you're using the schema that corresponds to a particular ORCM release (included with each release).

2.1.11 Setting MCA Parameters

There are different ways to setup the MCA parameters in ORCM. Following are methods to setup MCA parameter and its order of precedence:

- Setting up using the command line parameters

Each ORCM application command line supports the following command line options:

- `--omca <param_name> <value>` option

The command line options override the default options in ORCM takes precedence over other methods of setting up MCA parameters.

- Setting up using environment variables

Using ORCM MCA environment variables MCA parameter can be setup for all ORCM tools in this node. The environment setup can be overridden using the command line parameter.

```
export ORCM_MCA_<mca_param_name>=<value>
```

- Setting up in orcm-site.xml file

Under each node type setup MCA parameters using the MCA xml tag. These are default settings and takes precedence over other methods of setting up MCA parameters. This is applicable for all tools and daemons using this orcm-site.xml for startup. These are specified in key=value format.

```
<mca-params>
<value>sensor_base_sample_rate=5,sensor_base_log_samples=1</value>
</mca-params>
```

- Setting up using openmpi-mca-params.conf

Use an Openmpi-mca-params.conf file to setup the global default MCA parameters. This method is overridden using the above methods in the order or precedence.

- Programming defaults

Some MCA parameters are required and they have programming defaults hard-coded in the code. This default values can be overridden using the above methods.
To get a list of all possible MCA parameters, run the following command: (orcm-info –help for details)

```
shell$ orcm-info  --param  all  all
```

To set the parameter in any orcm program (including orcmd and opal_db) use the following syntax:

```
shell$ <orcmprog> --omca  parameter1-name  parameter1-value  \
                [--omca parameter2_name parameter2_value] ...
```

For example:

```
shell$ orcmd --omca sensor_base_sample_rate=5 \
             --omca sensor_base_log_samples=1
```

2.2 Database Installation

2.2.1 Database Server

This section provides instructions on how to set up the ORCM DB. These instructions are meant to be executed on the server that will contain the database. At the moment, ORCM provides support for PostgreSQL, so the following sections will provide instructions on how to set up the database using this DBMS.

To complete the following steps, the following file is needed: "orcmdb_psql.sql" (found in the ORCM repository in the "contrib/database" directory).

NOTE: the following instructions will use the following settings as reference, but the database administrator may choose to use different settings:

- Database instance name: *orcmdb*
- Database user: *orcmuser*

2.2.1.1 Software Requirements

Package	Version	Req.	Notes
PostgreSQL Server	9.3 or higher	Yes	Required on database server
PostgreSQL Client	9.3 or higher	Yes	Required on database server

2.2.1.1.1 Requirements for PostgreSQL NOTE: Client may be installed on any machine for administrative tasks: testing the database connection, data and schema management, etc.

2.2.1.2 Installation Overview At a high level, installing the database requires the following steps:

1. Installing the DBMS
2. Performing some configuration tasks (e.g. enabling remote access to the database)
3. Creating the database
4. Performing basic DBA tasks: creating users and roles

2.2.1.3 Notes On User Privileges For simplicity, the following steps provide instructions for creating a single database user with all the privileges. However, it's recommended to create roles and set privileges appropriately. It's up to the DBA to decide this and it will depend on the number of users that need to be managed and on organization policies.

General recommendations regarding users and privileges:

- A separate administrative user should be created and it should be used to create the database.

- Roles should be used to manage user privileges. Administrative users should have all privileges on the database while regular users should be restricted (depending on the data they need to access for their tasks).

- The standard ORCM user should have the following privileges:
  - Select, insert, update and delete privileges on all tables
  - Execute privileges on all stored procedures

2.2.1.4 Preparing the Server

2.2.1.4.1 PostgreSQL installation

1. Install the PostgreSQL server and client
   - Please refer to the PostgreSQL documentation for installation instructions
   - PostgreSQL installation wiki

2. Verify the installation by starting the *postgresql* service
   - `service postgresql start`
   - NOTE:
     - Depending on the version, before being able to start the service it may be necessary to execute:
       * `service postgresql initdb`
     - If desired, the service may be configured to start automatically:

      * `chkconfig postgresql on`

    - The actual name of the service may vary (e.g. “postgresql-9.3”)

    - These commands need to be run with administrative privileges

3. Enable external TCP connections to the *postgresql* service

   - Make sure the firewall is configured to allow incoming connections to the *postgresql* service port (5432 by default)

   - Enable client authentication
     - Edit the “pg_hba.conf” configuration file
       * The file location may vary depending on the installation package used

       * For example:
         · “/etc/postgresql/9.3/main”

         · “/var/lib/pgsql/9.3/data/”

     - The file contains detailed instructions on how to add authentication configuration options

     - At the very least, external connections should be allowed to the *orcmdb* database

     - Recommendation: start with basic password authentication and try more secure configurations once this is working

   - Enable networking for PostgreSQL
     - Edit the “postgresql.conf” configuration file

     - Edit the following line to specify what IP addresses to listen on:
       * `listen_addresses = ‘<comma-separated list of addresses>’`

       * NOTE: use `‘*’` to specify all

4. Create *orcmuser*

- Use the *createuser* command as the default *postgres* user:
  - `sudo -u postgres createuser -P orcmuser`
  - NOTE:
    * This command will prompt the user for a password. Please choose a strong password.
    * To verify if the user was created successfully, execute `\du` from a *psql* session.

5. Create the *orcmdb* database
   - NOTE: this requires the database creation script found in ORCM: "orcmdb_psql.sql"
   - Create the database:
     - `sudo -u postgres createdb --owner orcmuser orcmdb`
   - Use the *psql* tool to run the database creation script:
     - `psql --username=orcmuser --dbname=orcmdb --password -f orcmdb_psql.sql`
     - NOTE: depending on the authentication configuration in "pg_hba.conf" for local connections, the *orcmuser* may not be allowed to execute this command. There are two alternatives for handling this:
       * Enable password authentication for local connections (at least temporarily)
       * Execute this command remotely

6. Verify the installation
   - Make sure the database server is listening on a port
     - `netstat -plane |grep postmaster`
   - Connect to the database from a remote machine:
     - `psql --host=<hostname or IP address> --username=orcmuser --dbname=orcmdb --password`

- List the database's tables:
  - `\dt`
  - Here are some of the tables that will be listed
    - data_item
    - data_sample
    - event
    - event_type
    - fru
    - fru_type
    - job
    - job_node
    - maintenance_record
    - node
- See 4.3 ORCM DB Schema for the DB schema diagrams

2.2.2 Database Connectivity

This section provides instructions to configure clients to connect to the ORCM DB. This is required for clients that will need to communicate with the database (e.g. clients that need to run ORCM with the DB component enabled, like an aggregator node).

ORCM provides support for two database connection methods:

- Connecting via the PostgreSQL client library
- Connecting via ODBC

2.2.2.1 Connecting via the PostgreSQL Client Library

Package	Version	Req.	Notes
PostgreSQL Client	9.3 or higher	Yes	Required on database clients
PostgreSQL shared libraries	9.3 or higher	Yes	Installed as a dependency

2.2.2.1.1 Software Requirements The PostgreSQL client library is going to be required for ORCM nodes that need to connect to the database. This is usually for nodes with the aggregator role.

2.2.2.1.2 Configuring the ORCM DB "postgres" Component Once the PostgreSQL client package is installed, clients should be able to link to the PostgreSQL libraries and use the native API to communicate with the database. In the case of ORCM, the following MCA parameters are required to tell the ORCM *postgres* component how to connect to the database:

- db_postgres_uri: the URI for the host where the PostgreSQL server process is running. This can be specified in the following format: ":". The port is optional. If left blank, it will use the PostgreSQL default port.
- db_postgres_database: the name of the database (e.g. "orcmdb").
- db_postgres_user: the user and password in the following format: ":".

2.2.2.2 Connecting via ODBC To complete the following steps, the following files are needed (found in the ORCM repository in the "contrib/database" directory):

- "psql_odbc_driver.ini"
- "orcmdb_psql.ini"

Package	Version	Req.	Notes
unixODBC	2.2.14 or higher	Yes	Required on the database clients
PostgreSQL ODBC driver	09.03.0210 or higher	Yes	Required on the database clients
PostgreSQL Client	9.3 or higher	No	Optional on management machines

2.2.2.2.1 Software Requirements NOTE: The PostgreSQL client may be installed on any machine for administrative tasks: testing the database connection, data and schema management, etc.

These client packages are generally required on nodes with the role of aggregator.

2.2.2.2.2 Installation Overview Configuring the clients for connecting to the database requires:

1. Installing an ODBC driver manager
2. Installing the ODBC driver for the desired DBMS
3. Configuring a DSN

Before installing the DBMS ODBC driver, it's necessary to install the ODBC driver manager: unixODBC. The unixODBC package provides the necessary functionality to allow clients to connect to a database via ODBC. In addition, for developing and building applications that will use ODBC, the unixODBC development package is necessary. So the development package is necessary for building ORCM with ODBC support.

After installing the unixODBC driver manager, execute the following command: `odbcinst -j`. Note where unixODBC installed the configuration files for: drivers, system data sources and user data sources. These files will be needed in the following sections.

Please refer to the unixODBC web page for installation instructions.

NOTE: unixODBC already provides a driver for PostgreSQL. However, it's recommended to install the latest drivers provided by the respective vendors.

2.2.2.2.3 ODBC Installation for PostgreSQL

1. Get ODBC configuration information:
   - Execute: `odbcinst -j`
   - Note where where unixODBC installed the configuration files for: drivers, system data sources and user data sources. These files will be needed in the following steps.

2. Install the PostgresSQL ODBC driver
   - Please refer to the PostgreSQL installation wiki for availability of a package for the ODBC driver.
   - Alternatively, the ODBC driver can be built from source
     - The source code can be downloaded from the PostgresSQL downloads web page
     - Please refer to the installation instructions provided with the source code. Usually, the steps are:
       - `./configure`
       - `make`
       - `sudo make install`
     - After completing the installation, note the directory where the driver (".so" file) was installed

3. Register the PostgreSQL ODBC driver
   - Edit the "psql_odbc_driver.ini" file and fill in the required parameters:
     - Driver: specify the absolute path where the PostgrSQL ODBC driver (".so" file) was installed
     - Execute the following command:
       - `odbcinst -i -d -f psql_odbc_driver.ini`
     - Open the ODBC driver configuration file and verify the driver was configured correctly

4. Configure a DSN to connect to the ORCM DB
   - NOTE: the DSN may be configured at the user level (visible only to the current user) or at the system level (visible to all users that log in to the machine).
   - Edit the "orcmdb_psql.ini" file and fill in the required parameters:
     - Driver: specify the exact name of the driver as configured in the ODBC driver configuration file
     - Server: specify the hostname or IP address of the server where the database was installed
   - Configure the DSN:
     - `odbcinst -i -s -f orcmdb_psql.ini -l`
     - NOTE:
       * This will configure the DSN at the system level (visible to all users)
       * To configure the DSN at the user level (visible only to the current user):
         · `odbcinst -i -s -f orcmdb_psql.ini -h`
   - Open the respective DSN configuration file to verify the DSN was configured correctly

5. Verify the installation
   - Use the *isql* command-line utility provided by unixODBC to connect to the database:
     - `isql orcmdb_psql orcmuser <orcmuser's password>`
   - Try executing an SQL command:
     - `select * from data_sample`
     - The table will most likely be empty, but the query should at least succeed

2.2.2.2.4 Configuring the ORCM DB "odbc" Component The *odbc* component requires the following MCA parameters to be defined:

- db_odbc_dsn: the DSN name configured in the previous section.
- db_odbc_user: the user and password in the following format: ":".
- db_odbc_table: the database table that the DB component operations will use. At the moment this parameter has no effect for RAS monitoring operation. Please set to: "data_sample".

2.2.2.3 Setting MCA Parameters These MCA parameters may be specified in:

- The "openmpi-mca-params.conf" file
- The "orcm-site.xml" file
- The command line via the ORCM "-omca" command-line parameters

For more details on setting MCA paramters, please refer to: [[2.1.11-Setting-MCA-Parameters]].

2.2.3 Enabling Data Purge

This section provides instructions on how to automate data purging when using PostgreSQL. Specifically, on the table with high growing tuple count (e.g. the `data_sample_raw` table).

The same pattern and technique can be translated to another database dialect, but the code is Postgres specific. Also the scripts mentioned in this step are available in the ORCM repository `contrib/database` directory.

2.2.3.1 Background on ORCM Data Purging Requirement The ORCM process is often configured to continuously collect and store data to the database. Depending on the configuration, these data can grow rapidly and unbounded. Hence, there need to be a way to purge out the old data.

Please note, data archive, if interested, should be done before purging data. Once the data is purge, it is gone from the database. The ORCM PostgreSQL schema (orcmdb_psql.sql) provides two ways to pure out data (based on timestamp attribute):

- A stored procedure to delete data after certain time interval from current time
- A trigger based partition with dropping partitions that has exceeded certain time interval from the timestamp of the latest tuple

Each of these will be shown in detail on how to deploy in the following sections.

2.2.3.2 Purging with 'purge_data()' Function The `purge_data()` function provides the simple way to deleting data after the specified time interval from current time. At the high level, this function resolves an SQL DELETE statement:

```
DELETE FROM table_name WHERE timestamp_column < CURRENT_TIMESTAMP - interval '<interval_numb
```

Below is an example of the SQL statement invocation of the `purge_data()` function. This statement would DELETE all rows in the `data_sample_raw` table WHERE `time_stamp` column having value of less than 6 months from the current time.

```
SELECT purge_data('data_sample_raw', 'time_stamp', 6, 'MONTH');
```

The SQL statement example above can be schedule to run on a regular basis using pgAgent. Alternative, the statement can also be executed as a shell script using `psql` tool. It then can easily be scheduled to run on a regular basis using any schedule tool such as `crontab`.

```
psql -U username -d database -c "SELECT purge_data('data_sample_raw', 'time_stamp', 6, 'MONT
```

2.2.3.3 Purging with Dropping Partition Another way to purge data is to DROP the table rather than DELETE tuple from a table as in how the `purge_data()` function works. We do not want to DROP the whole table, but only a section of the table. There is no such thing as DROP a section of a table; However, DROP of a table partition is possible. So for this to work, we will first enable partition on the table and DROP any out dated partition.

Below is the detail instruction on how to enable partition on a given table with and without the helper script.

2.2.3.3.1 Partition without Helper (psql)

The steps in this section involve invoking PostgreSQL functions. If you are not comfortable with SQL, see the next section on how to perform this task with the helper script.

Note, running the `generate_partition_triggers_ddl()` will only output the DDL code to enable partition. It will not make any change to the database. Also a few things to be aware when calling this function:

- Within a GUI application such as pgAdminIII, the result may be truncated, so be sure to set it to allow longer length of characters in the return result.

- From the command line such as `psql` the result is displayed as

Below is an example of the SQL statement invocation on the generated partition code for the `data_sample_raw` table on the `time_stamp` column by DAY and keep the last 180 partitions.

1. Generate the partition DDL (data definition language) code.

   ```
   SELECT generate_partition_triggers_ddl('data_sample_raw', 'time_stamp', 'DAY', 180)';
   ```

2. Review the generated code from the output above. Note, some GUI tool may be set to limit the length of the string in the result. Check the GUI setting or run from command line interface (psql) to get the complete generated code.

3. Turn on auto dropping old partition by uncomment the `-- EXECUTE('DROP TABLE...` statement within the generated code. By default, the generated DROP TABLE statement is commented out.

4. Execute the modified and reviewed version of the generated code

2.2.3.3.2 Partition with Helper Script

The same process can be done using `pg_partition_helper.py`, a python helper script that would perform the step above. This pythons script uses SQLAlchemy to connect to the PostgreSQL database and execute the SQL DML (data manipulation language) and DDL. For this to work, the system needs to have the following python modules installed:

- SQLAlchemy - Database Abstraction Library - `sudo pip install SQLAlchemy`

- psycopg2 - Python-PostgreSQL, also the default PostgreSQL driver used by SQLAlchemy - `sudo pip install psycopg2`

Because this helper script uses SQLAlchemy to connect to the database, the database connection string will need to be in the format of SQLAlchemy database URL syntax

The helper script expects the PostgreSQL database URL to be stored in the environment variable `PG_DB_URL` or passed to it with the `--db-url` option, see example below on how to set it.

```
export PG_DB_URL=dialect+driver://username:password@host:port/database
```

The helper script has two commands: enable and disable partition. Use `-h` or `--help` to get usage syntax. Below is an example of enabling trigger based partition for the `data_sample_raw` table on the `time_stamp` column by DAY and keeping the last 10 partitions (based on the timestamp of the new tuple).

```
export PG_DB_URL=dialect+driver://username:password@host:port/database
python pg_partition_helper.py enable data_sample_raw time_stamp DAY --interval-to-keep 10
```

Example of disable partition for the `data_sample_raw` table.

```
export PG_DB_URL=dialect+driver://username:password@host:port/database
python pg_partition_helper.py disable data_sample_raw
```

2.2.3.3.3 Managing Partitions

Table partitioning in PostgreSQL database is based on table inheritance. Each new partition of a main table is a whole new table that inherits the schema of the main table. In the examples above, `data_sample_raw` is the main table that does not hold any record. All records are stored in its corresponding partition. When querying data from the main table, `data_sample_raw`, PostgreSQL automatic queries the data from the partitioned tables. This can be verified using PostgreSQL `EXPLAIN` statement.

Partition can be un-linked from the main table (no longer a part of the main table), this effectively provides an archive mechanism, or it can be DROPPED from the schema. Below is an example of breaking the inheritance for the `data_sample_raw_20150701_time_stamp` partition from the main table `data_sample_raw`.

```
ALTER TABLE data_sample_raw_20150701_time_stamp NO INHERIT data_sample_raw;
```

2.2.3.4 Recommendation The simple `purge_data()` function ultimately is an SQL DELETE statement. The process of deleting records usually involves some logging as the overhead. In contrast to deleting records, dropping the whole partition of a table is close to instantaneous (i.e. quick format). However, this requires partitioning a table in order to have partition to drop. Partition of a table in turn incurs a cost of executing a trigger to route each new tuple to the respective partition, but it provides many other benefits beside able to drop the whole partition.

The recommendation is to use the `purge_data()` approach as this may already be sufficient for the data purging. The data partitioning approach is better for archive and potentially improve performance on accessing the data since the data is ‘partitioned’ to different tables.

2.2.4 Database Distro Specific Examples

2.2.4.1 CentOS 6.5

2.2.4.1.1 CentOS 6 PostgreSQL Installation Edit `/etc/yum.repos.d/CentOS-Base.repo` and add `exclude=postgresql*` for [base] and [updates] section.

```
yum localinstall http://yum.postgresql.org/9.3/redhat/rhel-6-x86_64/pgdg-centos93-9.3-1.n
yum install postgresql93-server
```

To remove:

```
rm -rf /var/lib/pgsql
yum remove postgresql93-server
```

2.2.4.1.2 PostgreSQL Configuration

```
service postgresql-9.3 initdb
```

Edit `/var/lib/pgsql/9.3/data/pg_hba.conf` and add: `local all orcmuser trust` (must be before other entries).

```
service postgresql-9.3 start
```

Failed to start? Check `/var/lib/pgsql/9.3/pgstartup.log` for details. Make sure an existing *postgres* process is not already running: `/usr/pgsql-9.3/bin/postmaster -p 5432 ...`

2.2.4.1.3 PostgreSQL DB and User Setup

```
su - postgres
dropdb orcmdb
dropuser orcmuser
createuser orcmuser
createdb --owner orcmuser orcmdb
```

NOTE: the "drop" commands were included to remove a preexisting installation (they're not necessary if this is the first installation).

2.2.4.1.4 ORCM-Specific Database Creation Script The scripts can be found in the ORCM repo, in the following directory: `orcm/build/contrib/database`.

```
psql --username=orcmuser --dbname=orcmdb -f orcmdb_psql.sql
```

2.2.4.1.5 Client Access Setup Edit `/var/lib/pgsql/9.3/data/pg_hba.conf` and add `host all all 127.0.0.1/32 trust` (for IPv4) and `host all all ::1/128 trust` for IPv6.

```
service postgresql-9.3 restart
```

To use the PostgreSQL native client library:

```
yum install postgresql93
psql --dbname=orcmdb --username=orcmuser #Try select * from data_samples_view.
```

To use ODBC:

```
yum install postgresql93-odbc
```

Edit `psql_odbc_driver.ini` and specify the path where the PostgreSQL ODBC driver was installed. Use the following command to find it: `rpm -ql postgresql93-odbc | grep psqlodbc.so`.

```
odbcinst -i -d -f psql_odbc_driver.ini
```

Edit `orcmdb_psql.ini` and specify the host where the PostgreSQL service is running and specify the name of the driver configuration (the key name from the ".ini" file from the previous step).

```
odbcinst -i -s -f orcmdb_psql.ini -h
odbcinst -s -q # List Data Source Names (DSNs)
isql -v orcmdb_psql orcmuser # Test ODBC access to the DB.  Try select * from data_samples_vie
```

Look in `/var/lib/pgsql/9.3/data/pg_log/postgresql-Tue.log` for access logs to the DB.

NOTE: for simplicity, these steps are configuring the authentication method for the database as "trust". This is a good approach to start with to make it easier to get everything up and running. However, once the basic setup is completed, it's highly recommended to configure a more secure authentication method.

2.2.4.1.6 Run ORCM To use the ORCM *postgres* component:

```
orcmd --omca sensor heartbeat,coretemp --omca db_base_verbose 100 --omca db_postgres_uri lo
```

To use the ORCM *odbc* component:

```
orcmd --omca sensor heartbeat,coretemp --omca db_base_verbose 100 --omca db_odbc_dsn orcmdb
```

NOTE: because the "trust" authentication method was configured in the previous step, the password here is irrelevant. However, after the basic setup is up and running, it is highly recommended to at least configure the "password" authentication method, in which case the correct password should be used here.

2.2.4.1.7 Query the DB First, initiate a *psql* session: `psql -d orcmdb -U orcmuser [-W]`.

Querying RAS monitoring data:

```
select * from data_samples_view;
```

Deleting the data from RAS monitoring:

```
delete from data_sample_raw;
```

Getting the number of sample data rows from RAS monitoring:

```
select count(*) from data_sample_raw;
```

Querying node feature data (inventory):

```
select * from node_features_view;
```

2.2.4.1.8 Enabling Network Access to the DB Edit `/var/lib/pgsql/9.teuser3/data/postgresql` and set: `listen_addresses = '*'`.

2.2.4.1.9 Adding a password Edit `/var/lib/pgsql/9.3/data/pg_hba.conf` and set the authentication method to `password` instead of `trust`:

From within a *psql* session (logged in as orcmuser):

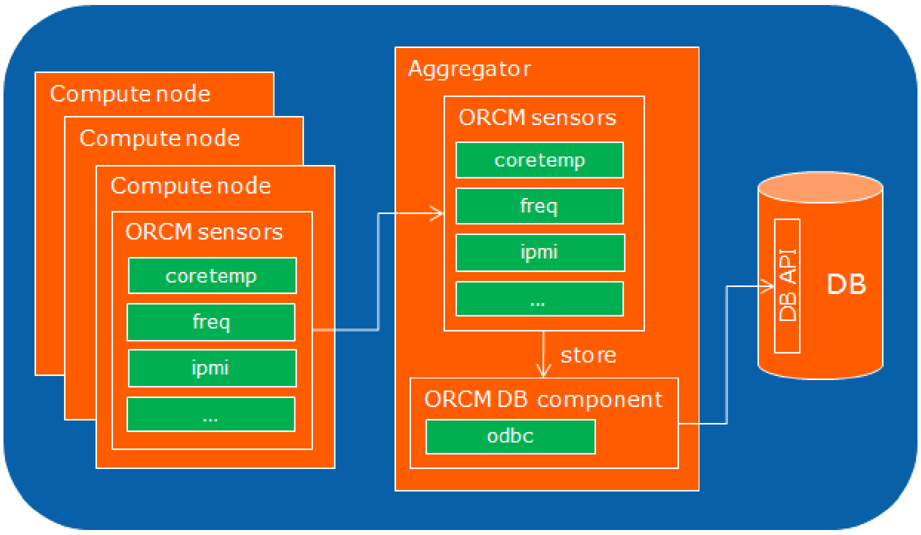
```
alter user orcmuser with password '<choose a password>';
```

2.3 RAS Monitoring

2.3.01 Introduction

RAS monitoring is enabled by default in the ORCM, if it was configured during build time. Alternatively any of the sensor plugins can be selected or deselected by passing specific mca parameters. For example, for enabling only the sensor ipmi, coretemp, we need to pass:

```
orcmd --omca sensor heartbeat,ipmi,coretemp
```



Note that some sensors, such as ipmi, require orcmd to be run as root for access to the underlying metric collection.

Heartbeat is a special plugin under the sensor framework which collects a bucket of data holding all the sampled data metrics by each sensor and sends it to the aggregator, so we need to enable it every time we need collect sensor related data.

If no mca parameters are passed w.r.t. 'sensor' then the framework by default enables all the plugins that were are available and able to run. Some plugins contain special mca parameters that define certain special conditions in their functionality, the are listed under each corresponding plugin where ever applicable.

Apart from RAS monitoring, the sensor framework is also used to collect the inventory details of the compute nodes. The 'dmidata' plugin is a unique plugin which collects only the inventory information but doesn't collect any periodic metrics.

MCA Parameters for sensor framework:

- sensor_base_verbose: Set the verbosity level while launching the orcmd daemon. The amount of debug messages getting logged depends on the verbosity levels.

- sensor_base_sample_rate: Set the sampling rate at which each of the sensor components sample their respective metrics.

- sensor_base_log_samples: Enable/Disable logging the collected metrics into the database.

- sensor_base_collect_metrics: Enable collecton of the metrics.

- sensor_base_collect_inventory: Enable collecton of the inventory details.

- sensor_base_set_dynamic_inventory: N/A

2.3.1 coretemp

This component is used to read the DTS temperature sensor values from each Processor present in each compute node. The coretemp.ko kernel module needs to be loaded for this plugin to function. This can be done by running:

```
sudo modprobe coretemp
```

If this module is not present in the linux distro, then the lm-sensors package needs to be installed. See the instructions here.

MCA parameters:

- sensor_coretemp_test: Enable logging a random test sample by the plugin for testing sensor to database connectivity

- sensor_coretemp_enable_packagetemp: Enable/Disable collecting the package or die temperature. Enabled by default.

- sensor_coretemp_use_progress_thread: Enable coretemp to run on it's own separate thread.

- sensor_coretemp_sample_rate: If coretemp is running in a separate thread, then this parameter is used to configure the sample rate at which the data will be sampled.

2.3.2 file

This component is used by applications to detect stalled programs. It tests to see if the specified file has been modified since the last time we looked, if it hasn't been touched for the specified number of times, then we declare the application to have stalled.

MCA parameters:

- sensor_file_file: The name of the file that needs to be monitored
- sensor_file_check_size: Boolean value to indicate whether the file size needs to be monitored or not.
- sensor_file_check_access: Boolean value to indicate whether the last access time has to be monitored or not.
- sensor_file_check_mod: Boolean value to indicate whether the last modified time has to be monitored or not.
- sensor_file_limit: Integer value indicating the maximum count before which the staleness of the file can be ignored, it is by default set to 3.
- sensor_file_test: Enable logging a random test sample by the plugin for testing sensor to database connectivity
- sensor_file_use_progress_thread: Enable file to run on it's own separate thread.
- sensor_file_sample_rate: If file is running in a separate thread, then this parameter is used to configure the sample rate at which the data will be sampled.

2.3.3 freq

This component is used to read the CPU frequency scaling values from each Processor present in each compute node.

MCA parameters:

- sensor_freq_test: Enable logging a random test sample by the plugin for testing sensor to database connectivity
- sensor_freq_pstate: Enable collection of the intel_pstate related information

- sensor_freq_use_progress_thread: Enable freq to run on it's own separate thread.
- sensor_freq_sample_rate: If freq is running in a separate thread, then this parameter is used to configure the sample rate at which the data will be sampled.

2.3.4 ft_tester

This component is used to test the fault tolerance/resilience of the system. This component will roll a random number generator each sampling time. If the lucky number hits, then the plugin will terminate either the indicated app proc or the daemon itself will kill itself, based on the specified parameter setting.

MCA parameters:

- sensor_ft_tester_fail_prob: Set an upper threshold probability value for killing a child process.
- sensor_ft_tester_multi_allowed: Boolean value to indicate whether multiple child processes should be killed or not.
- sensor_ft_tester_daemon_fail_prob: Set an upper threshold probability value for killing itself.

2.3.5 heartbeat

This component is used to gather all the relevant data sampled by the other sensor components that have been enables and properly send that data to the appropriate log function.

2.3.6 ipmi

This component is used to read IPMIUtil data from the BMC(s) present in each compute node.

MCA parameters:

- sensor_ipmi_bmc_username: Used to set the username of the remote BMC nodes for retrieving the metrics via the IPMI interface
- sensor_ipmi_bmc_password: Used to set the password of the remote BMC nodes for retrieving the metrics via the IPMI interface, for the above configured username

- sensor_ipmi_sensor_list: Used to set the list of BMC monitored sensor names whose value is to be retrieved. Use the sensor names stored in the BMC as defined in section 43.1. For example: “PS1 Power In,Processor 2 Fan,Fan 1”

- sensor_ipmi_sensor_group: Used to set the group of BMC monitored sensor names whose value is to be retrieved, any sensor whose name contains this term will be retrieved. For example: “Fan” (This will filter out all the sensors with the term ‘fan’ in it)

- sensor_ipmi_test: Enable logging a random test sample by the plugin for testing sensor to database connectivity

- sensor_ipmi_use_progress_thread: Enable ipmi to run on it’s own separate thread.

- sensor_ipmi_sample_rate: If ipmi is running in a separate thread, then this parameter is used to configure the sample rate at which the data will be sampled.

2.3.8 sigar

This component reads memory, swap memory, cpu load, disk, and network data for each compute node. This component is mutually exclusive with the resusage component. Which means that only one component can be operational at any given time. The metrics collected by this component are logged as ‘procstat’ or ‘procstat_’ for per child process related metrics.

Please note that the exact list of metrics collected and in some cases the units of measurement could differ between these two components for some parameters.

MCA parameters:

- sensor_sigar_test: Enable logging a random test sample by the plugin for testing sensor to database connectivity

- sensor_sigar_mem: Enable collecting and logging memory usage details.

- sensor_sigar_swap: Enable collecting and logging swap memory usage details.

- sensor_sigar_cpu: Enable collecting and logging cpu usage details.

- sensor_sigar_load: Enable collecting and logging system load details.

- sensor_sigar_disk: Enable collecting and logging disk access details.
- sensor_sigar_network: Enable collecting and logging network usage details.
- sensor_sigar_sys: Enable collecting and logging system usage details.
- sensor_sigar_proc: Enable collecting and logging system wide process details as well as orcmd and child process details. Note that the child process stats will be logged into the database with the primary key - ‘procstat_\. This also means that if orcm is launched from an emulator like valgrind it will be logged with a different primary key than ’procstat_orcm’
- sensor_sigar_use_progress_thread: Enable sigar to run on it’s own separate thread.
- sensor_sigar_sample_rate: If sigar is running in a separate thread, then this parameter is used to configure the sample rate at which the data will be sampled.

2.3.9 resusage

This is an alternative to the sigar component, which collects OS metrics by directly accessing the file system, without using any third party libraries. This component is mutually exclusive with the sigar component. Which means that only one component can be operational at any given time. The metrics collected by this component are logged as ‘procstat’ or ‘procstat_’ for per child process related metrics.

Please note that the exact list of metrics collected and in some cases the units of measurement could differ between these two components for some parameters.

MCA parameters:

- sensor_resusage_sample_rate: N/A
- sensor_resusage_node_memory_limit: N/A
- sensor_resusage_proc_memory_limit: N/A
- sensor_resusage_log_node_stats: Whether the sampled node status information is to be sent to the log function for further processing.

- sensor_resusage_log_process_stats: Whether the sampled process status information is to be sent to the log function for further processing.

- sensor_resusage_test: Enable logging a random test sample by the plugin for testing sensor to database connectivity

- sensor_resusage_use_progress_thread: Enable resusage to run on it's own separate thread.

- sensor_resusage_sample_rate: If resusage is running in a separate thread, then this parameter is used to configure the sample rate at which the data will be sampled.

2.3.10 nodepower

This component is used to read node power using raw commands supported by PSUs. Currently the input power to PSUs is reported.

MCA parameters:

- sensor_cnodepower_use_progress_thread: Enable nodepower to run on it's own separate thread.

- sensor_nodepower_sample_rate: If nodepower is running in a separate thread, then this parameter is used to configure the sample rate at which the data will be sampled.

2.3.11 componentpower

This component is used to read CPU and DDR power using RAPL MSRs.

MCA parameters:

- sensor_componentpower_use_progress_thread: Enable componentpower to run on it's own separate thread.

- sensor_componentpower_sample_rate: If componentpower is running in a separate thread, then this parameter is used to configure the sample rate at which the data will be sampled.

2.3.12 dmidata

This component is used to collect the inventory information from the node

MCA parameters:

- sensor_dmidata_test: Enable logging a random test sample by the plugin for testing sensor to database connectivity
- sensor_dmidata_ntw_dev: Enable collecting the network interface details
- sensor_dmidata_blk_dev: Enable collecting the block/hard drive details
- sensor_dmidata_mem_dev: Enable collecting memory module details.
- sensor_dmidata_pci_dev: Enable collecting PCI devices details (Needed for collecting ntw.blk,mem device details)
- sensor_dmidata_freq_steps: Enable collection of the available frequency steps at which the CPU can be configured to run.

2.3.13 mcedata

This component is used to collect the Machine Check Errors (MCE) that have been encountered by the OS. The open source 'mcelog' daemon has to run in the background. It also needs to run in the 'raw' mode, for mcedata plugin to extract the data accurately.

Note: Due to a kernel bug in machinecheck handling, the edac driver has to be disabled in order for mcelog to collect all the memory module related errors. This bug is fixed in v4.0-rc1 kernel release.

The following types of errors are currently read and decoded:

- Cache Errors
- Memory Controller Errors
- Bus & Interconnect Errors

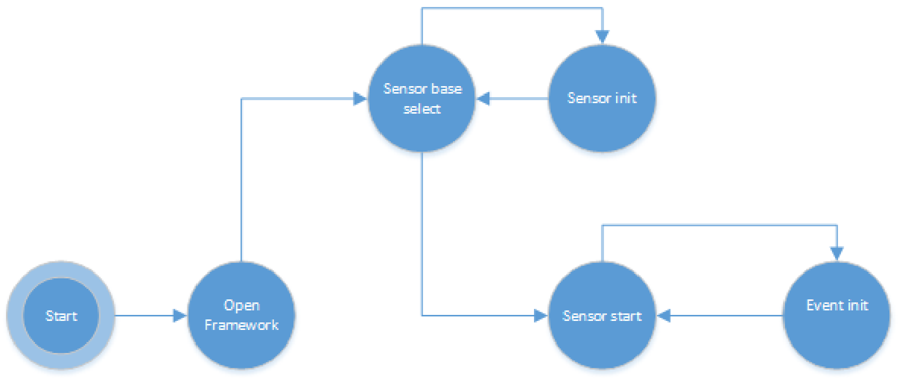
MCA parameters:

- sensor_mcedata_logfile: Pass the path to the logfile into which mcelog stores the recorded errors. It must match the logfile configured in mcelog.conf
- sensor_mcedata_use_progress_thread: Enable mcedata to run on it's own separate thread.

- sensor_mcedata_sample_rate: If mcedata is running in a separate thread, then this parameter is used to configure the sample rate at which the Machine Check Event occurences will be sampled.

2.3.02 State Flow

Sensors Initialization & Start

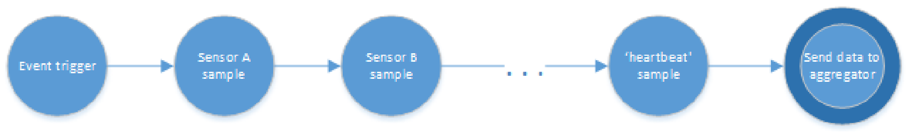


Before sampling data using any of the underlying components, the framework and in effect each component has to be initialized. This is required for detecting the available plugins and verifying their underlying dependencies are met. Initialization stage also provides a safe state to allocate memory/space that's required for the component's functioning.

The base framework also detects the priority assigned for each component and selects the plugin with the highest priority in case of conflicting "component name". This step also calls the init functions of each component in effect instructing them to initialize their structures/memories and check for any particular dependencies. If anything is not in order and the init sequence fails, then the sensors framework removes the concerned plugin from it's 'bucket' list and in turn from sampling that particular metric(s).

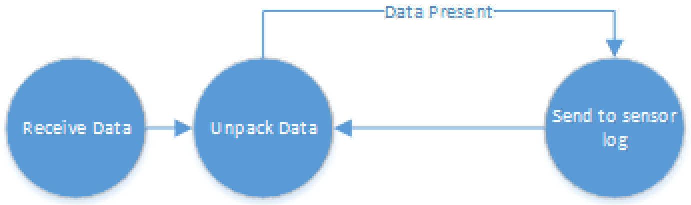
The sensors framework's start call in turn sets up an event loop with a user defined time interval, with a specific callback function - "manually_sample", which gets invoked at after each trigger.

Sensors Sample



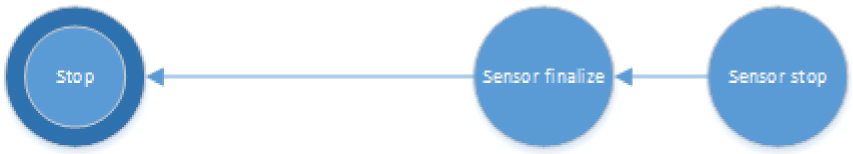
The sampling stage of the sensors is pretty simple. The event loop (setup during the initialization stage) triggers after every sampling period and invokes the sample function of call of each active module. The order of selecting the sample function is based on the priority of each component and is assigned by the developer. Each components ‘samples’ their respective metrics and packs it into a large bucket, preceded with a string containing the plugin’s name. The plugin with the lowest priority is the heartbeat plugin and it gets invoked the last. This is a special plugin in the sense that it it takes the packed buffer and sends it the aggregator, via the RML layer’s send call by using a dedicated tag.

Sensors Log



This state is valid only for the aggregator nodes, since the packed RAS metric data sent by the heartbeat plugin is ‘always’ directed at the aggregator node. Once the aggregator node received the buffer, it unpacks it and directs the contents to the log function of the respective component. This is possible since each component is expected to pack it’s name preceding the metric contents.

Sensors Stop & Finalize



Once the user application decides to stop sampling the metric data, it can invoke the finalize function call of the base framework which in turn removes the event from the event loop.

NOTE: All the above explanation is valid for a single thread sampling. Per-thread sampling is slightly different from the above process. TBD

3 ORCM User Guide

3.1 ORCM Tools

3.1.1 octl

Admin-focused tool for interacting with ORCM. This tool has the ability to run as an interactive shell or as a single one-shot command. Currently the tool provides information about configured resources, sessions, and queues. The tool is also for managing sessions and power budget at the admin level.

The octl command itself takes the following options:

```
octl [OPTIONS]
  Open Resilient Cluster Manager "octl" Tool

   -am <arg0>            Aggregate MCA parameter set file list
   -gomca|--gomca <arg0> <arg1>
                  Pass global MCA parameters that are applicable to
                  all contexts (arg0 is the parameter name; arg1 is
                         the parameter value)
-h|--help                This help message
   -omca|--omca <arg0> <arg1>
                    Pass context-specific MCA parameters; they are
                  considered global if --gomca is not used and only
                   one context is specified (arg0 is the parameter
                         name; arg1 is the parameter value)
-v|--verbose             Be Verbose
-V|--version             Show version information
```

The subcommands have the option to take arguments specific to that command as well.

3.1.1.1 Interactive CLI The interactive mode of the CLI is invoked by running the command without any subcommands. Optional arguments such as MCA parameters can be specified as well.

```
% octl
*** WELCOME TO OCTL ***
 Possible commands:
   resource             Resource Information
   queue                Queue Information
   session              Session Management
   diag                 Diagnostics
   power                Power Budget
octl>
```

Once in the interactive shell, the `<tab>` key can be used to either autocomplete unambiguous partial commands or list possible completions for ambiguous partial commands. The `<?>` key will display more information about all of the commands at the current hierarchy.

For example, pressing `<tab>` after entering `res` will autocomplete the `resource` command, and pressing `<tab>` after the `resource` command is fully entered, will show:

```
octl> resource
    status add remove drain
octl> resource
```

As another example, pressing `<?>` after entering `queue`, will show:

```
octl> queue
Possible commands:
   status               Queue Status
   policy               Queue Policies
   define               Queue Definintion
   add                  Add Resource to Queue
   remove               Remove Resource from Queue
   acl                  Queue Access Control
   priority             Queue Priority Modification

octl> queue
```

Notice how in both cases, control is returned to the user to complete the command as desired.

To exit interactive mode, press `<ctrl>`+`<c>`. Entering an empty command will also exit interactive mode.

In the following sections, examples for each command will be given using normal one-shot execution mode. However, they can also be executed in interactive mode.

3.1.1.2 Resource The resource command set is used to display information about the resources (nodes) configured in the system. Currently resource modification is not supported administratively, but once support for that is added, the functionality will be invoked with this command set. The current implementation displays the node connection state: either up(U), down(D), or unknown(?) and the job state: allocated or unallocated. The node specification is an ORCM node regex.

Example:

```
% octl resource status
TOTAL NODES : 10
NODES                : STATE  SCHED_STATE
-----------------------------------------
node001              : U      UNALLOCATED
node[3:2-10]         : ?            UNDEF
```

3.1.1.3 Queue The queue command set displays information about the currently configured queues as well as the sessions within each queue. The current scheduler defines 3 queues: running, hold, and default. All sessions get placed initially in the default queue. As the scheduler launches the sessions they will be placed on the running queue. And a session that tries to allocate invalid resource definitions will be placed on the hold queue with a string describing the error.

The session information displays:

<session-id> <userid|grouid> <number of nodes requested> <EXclusve or SHared> <Batch or Interactive>

Example:

```
% octl queue status
********
QUEUES
********
running (1 sessions)
----------
1   502|20  1   EX  B
hold (0 sessions)
----------
default (2 sessions)
----------
2   502|20  5   SH  B
3   502|20  2   EX  B
```

3.1.1.4 Session The session command set is used to modify submitted sessions. Currently the only modification supported is canceling the session. If the session is in the default queue waiting to run, canceling the session will remove it from the queue. If the session is running, then canceling the session will terminate the running jobs and end the session. The session cancel command takes a session-id as a required argument.

Example:

```
% octl session cancel 1
```

```
Success
% octl queue status
********
QUEUES
********
running (0 sessions)
----------
hold (0 sessions)
----------
default (2 sessions)
----------
2   502|20  5   SH  B
3   502|20  2   EX  B
```

3.1.1.5 Diag The diagnostic command set allows running diagnostics on remote ORCM daemons. These commands require a node regex specification for determining which remote daemons to run on. See the 3.3 ORCM Node Regular Expressions section for details on how
to construct the regex.

Example 1: run cpu diagnostics on node001 through node010

```
% octl diag cpu node[3:1-10]
Success
```

Example 1: run ethernet diagnostics on node001 through node010

```
% octl diag eth node[3:1-10]
Success
```

Example 1: run memory diagnostics on node001 through node010

```
% octl diag mem node[3:1-10]
Success
```

3.1.1.6 Power The power budget command set allows setting and retrieving a cluster-wide power budget. This will only be enforced if the appropriate plugin or subsystems are in place to honor this request. The power *set* command takes a required argument of a numeric power budget (in Watts) for the system. The power *get* command displays the currently set budget.

Example 1: setting the power budget to 10000 Watts

```
% octl power set 10000
Success
```

Example 2: getting the current power budget

```
% octl power get
Current cluster power budget: 10000
```

3.1.2 ocli

User-focused tool for interacting with ORCM. This tool has the ability to run as an interactive shell or as a single one-shot command. Currently the tools provides information about configured resources, sessions, and queues. The tool is also for managing sessions at the user level.

The ocli command itself takes the following options:

```
ocli [OPTIONS]
  Open Resilient Cluster Manager "ocli" Tool

   -am <arg0>              Aggregate MCA parameter set file list
   -gomca|--gomca <arg0> <arg1>
                  Pass global MCA parameters that are applicable to
                  all contexts (arg0 is the parameter name; arg1 is
                         the parameter value)
-h|--help                This help message
   -omca|--omca <arg0> <arg1>
                    Pass context-specific MCA parameters; they are
                  considered global if --gomca is not used and only
                   one context is specified (arg0 is the parameter
                         name; arg1 is the parameter value)
-v|--verbose             Be Verbose
-V|--version             Show version information
```

The subcommands have the option to take arguments specific to that command as well.

3.1.2.1 Interactive CLI The interactive mode of the CLI is invoked by running the command without any subcommands. Optional arguments such as MCA parameters can be specified as well.

```
% ocli
*** WELCOME TO OCLI ***
 Possible commands:
   resource             Resource Information
   queue                Queue Information
   session              Session Management
ocli>
```

Once in the interactive shell, the `<tab>` key can be used to either autocomplete unambiguous partial commands or list possible completions for ambiguous partial commands. The `<?>` key will display more information about all of the commands at the current hierarchy.

For example, pressing `<tab>` after entering `res` will autocomplete the `resource` command, and pressing `<tab>` after the `resource` command is fully entered, will show:

```
ocli> resource
    status availability
ocli> resource
```

As another example, pressing `<?>` after entering `queue`, will show:

```
ocli> queue
Possible commands:
   status              Queue Status
   policy              Queue Policies

ocli> queue
```

Notice how in both cases, control is returned to the user to complete the command as desired.

To exit interactive mode, press `<ctrl>`+`<c>`. Entering an empty command will also exit interactive mode.

In the following sections, examples for each command will be given using normal one-shot execution mode. However, they can also be executed in interactive mode.

3.1.2.2 Resource The resource command set is used to display information about the resources (nodes) configured in the system. The current implementation displays the node connection state: either up(U), down(D), or unknown(?) and the job state: allocated or unallocated. The node specification is an ORCM node regex.

Example:

```
% ocli resource status
TOTAL NODES : 10
NODES                : STATE  SCHED_STATE
-----------------------------------------
node001              : U      UNALLOCATED
node[3:2-10]         : ?            UNDEF
```

3.1.2.3 Queue The queue command set displays information about the currently configured queues as well as the sessions within each queue. The current scheduler defines 3 queues: running, hold, and default. All sessions get placed initially in the default queue. As the scheduler launches the sessions they will be placed on the running queue. And a session that tries to allocate invalid resource definitions will be placed on the hold queue with a string describing the error.

The session information displays:

<session-id> <userid|grouid> <number of nodes requested> <EXclusve or SHared> <Batch or Interactive>

Example:

```
% ocli queue status
********
QUEUES
********
running (1 sessions)
----------
1   502|20  1   EX  B
hold (0 sessions)
----------
default (2 sessions)
----------
2   502|20  5   SH  B
3   502|20  2   EX  B
```

3.1.2.4 Session The session command set is used to modify submitted sessions. Currently the only modification supported is canceling the session. If the sessions is in the default queue waiting to run, canceling the session will remove it from the queue. If the session is running, then canceling the session will terminate the running jobs and end the session. The session cancel command takes a session-id as a required argument.

Example:

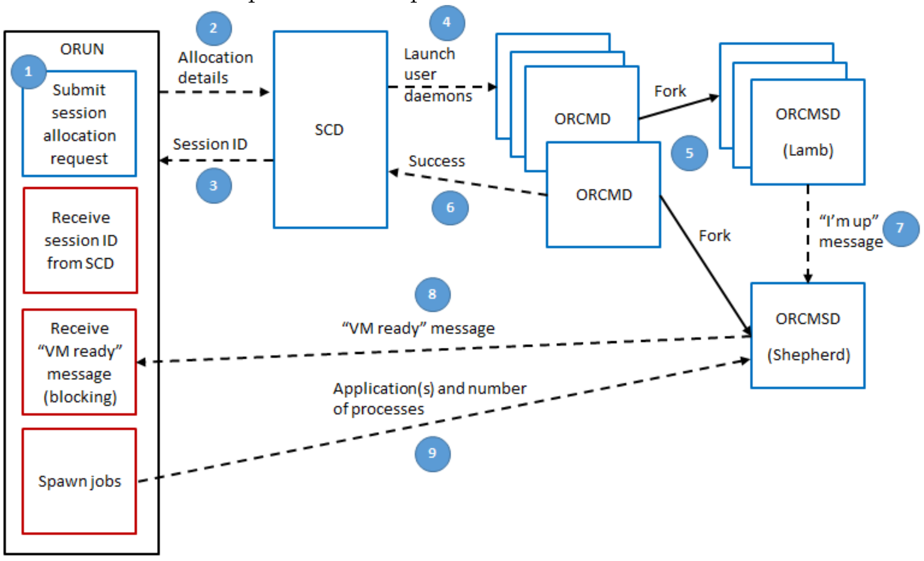
```
% ocli session cancel 1
Success
% ocli queue status
********
QUEUES
********
running (0 sessions)
----------
hold (0 sessions)
```

```
----------
default (2 sessions)
----------
2   502|20  5   SH  B
3   502|20  2   EX  B
```

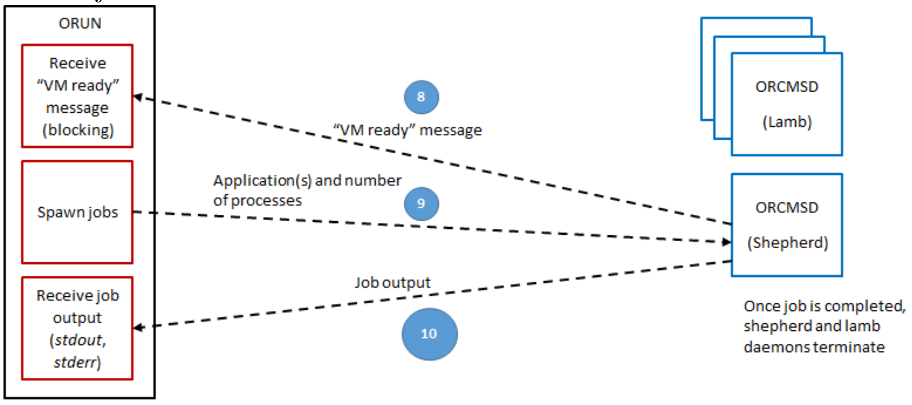
3.1.3 orun

The *orun* tool provides the user an interface to launch jobs, either on an existing allocation or a new allocation (in which case the allocation request is sent to the scheduler together with the job launch request). The latter becomes a convenience utility for session requests with a single job definition. When called with an allocation, *orun* will invoke the next job to be run within the session (this can be done when *orun* is invoked separately or as part of a batch job). When requesting a new allocation, the tool performs a blocking request to the scheduler, and upon obtaining a successful allocation, it launches the job in one shot. The following diagrams depict this scenario.

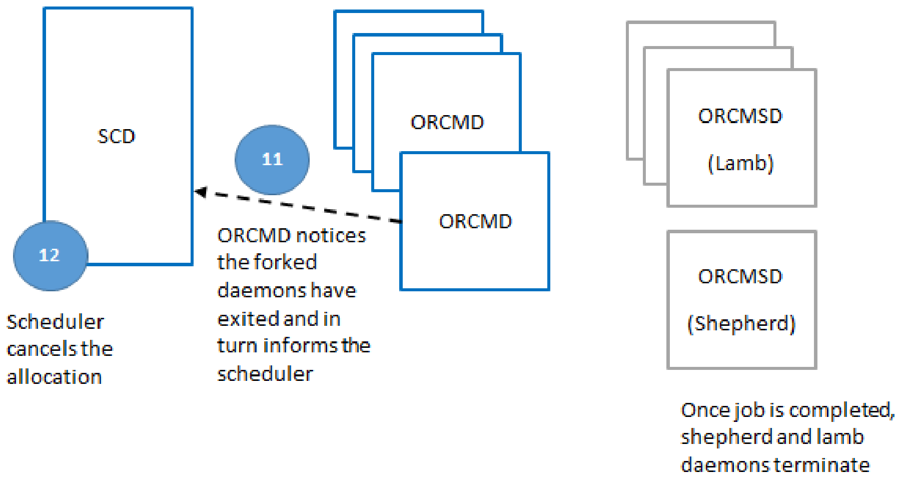
Part 1: allocation request and setup



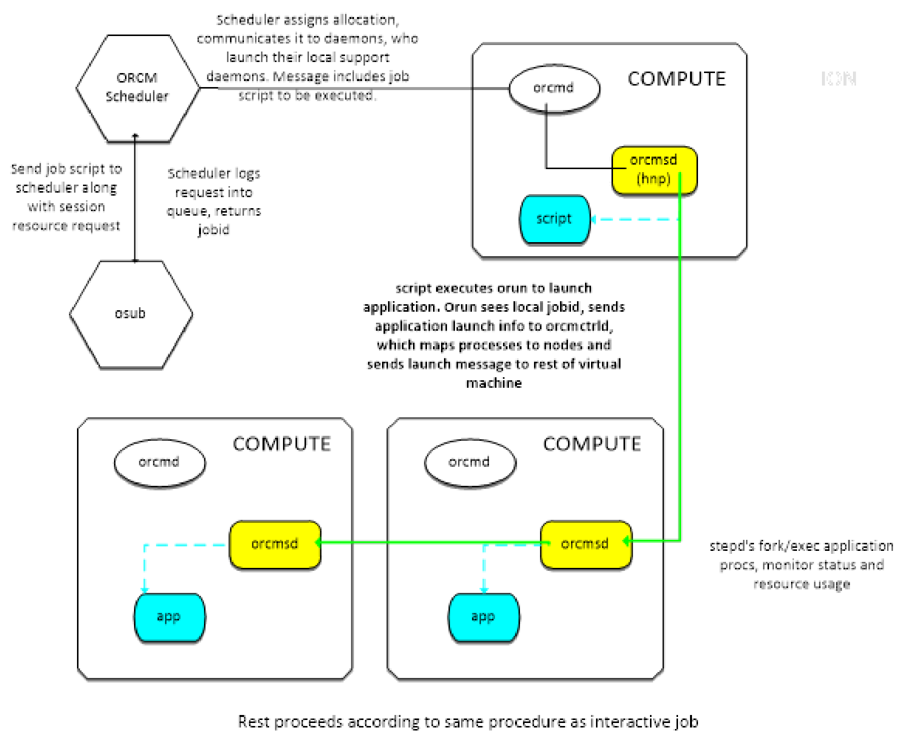
Part 2: job execution


ORUN
Receive "VM ready" message (blocking)
Spawn jobs
Receive job output (stdout, stderr)
8
"VM ready" message
Application(s) and number of processes
9
Job output
10
ORCMSD (Lamb)
ORCMSD (Shepherd)
Once job is completed, shepherd and lamb daemons terminate

Part 3: teardown


SCD
12
Scheduler cancels the allocation
11
ORCMD notices the forked daemons have exited and in turn informs the scheduler
ORCMD
ORCMD
ORCMSD
(Lamb)
ORCMSD
(Shepherd)
Once job is completed, shepherd and lamb daemons terminate

As mentioned previously, *orun* can be invoked within a batch job. For batch jobs, the *osub* tool may be used by providing it a batch script. This tool submits the batch script along with the session allocation request to the scheduler, and upon successful allocation, the scheduler sends the batch script to the session "shepherd" daemon, which then executes the script. The batch script should contain the embedded *orun* calls for launching applications on the allocated session.


Scheduler assigns allocation, communicates it to daemons, who launch their local support daemons. Message includes job script to be executed.
ORCM Scheduler
COMPUTE
orcmd
orcmsd (hnp)
script
Send job script to scheduler along with session resource request
Scheduler logs request into queue, returns jobid
osub
script executes orun to launch application. Orun sees local jobid, sends application launch info to orcmctrld, which maps processes to nodes and sends launch message to rest of virtual machine
COMPUTE
orcmd
orcmsd
app
COMPUTE
orcmd
orcmsd
app
stepd's fork/exec application procs, monitor status and resource usage
Rest proceeds according to same procedure as interactive job

The usage for the *orun* tool is as follows:

```
orun [OPTION]...  [PROGRAM]...
```

To run a user application with a new allocation request, use the following syntax:

```
orun [--np <X>] [--node <N>] <program>
```

Example:

```
% orun --np 4 --node 4 hostname
```

This will allocate 4 nodes with a blocking request to the scheduler (as a new allocation will be requested), and will run 4 copies of *hostname* in the newly created allocation. The **np** option will default to running all copies of the program in a round-robin fashion by allocated nodes. The *orun* tool will send the name of the directory where it was invoked on the local node to each of the remote nodes and attempt to change to that directory.

To run a user application with an already allocated session, use the following syntax:

```
orun [--np <X>] [--hnp-uri <URI>] <program>
```

Example:

```
% osub --node <N> -i
orcmshell% env |grep -i hnp
        ORCM_MCA_HNP_URI=65536.0;tcp://<hnp-ipaddrs>:12345
% orun --np 4 --hnp-uri  "65536.0;tcp://<hnp-ipaddrs> :12345" hostname
```

This will use the existing allocation that was specified and spawn the job by sending it to the session "shepherd" daemon. The session "shepherd" daemon will in turn map the application process to the allocated nodes.

The following are the options for *orun*:

- `--am <file list>`: Specify an aggregate MCA parameter set file list.
- `--app <app. file>`: Provide an application file that contains a list of applications to be executed. Entries are provided one per line and they should include any command-line parameters that need to be passed to the executable. This will ignore all other command-line options.
- `-d | --debug`: Enable debugging of Open RTE.
- `--gomca <name> <value>`: Pass global MCA parameters that will apply to all contexts, where `<name>` is the parameter name and `<value>` is the parameter value.
- `-h | --help`: Print the *orun* help text.
- `--hnp-uri <URI>`: Provide the URI for the "shepherd" node.
- `--map-by <policy>`: Specify the mapping policy to use for allocating processes. The policy may be: `slot`, `hwthread`, `core`, `socket` (default), `numa`, `node`.
- `-n | --np <n>`: Specify the number of processes to run.
- `--omca <name> <value>`: Pass context-specific MCA parameters, where `<name>` is the parameter name and `<value>` is the parameter value. They are considered global if `--gomca` is not used and only one context is specified.
- `--path <path>`: Provide a path for looking for executables to start processes.

- `-V | --version`: Print the tool version.
- `--xml`: Request output in XML format.

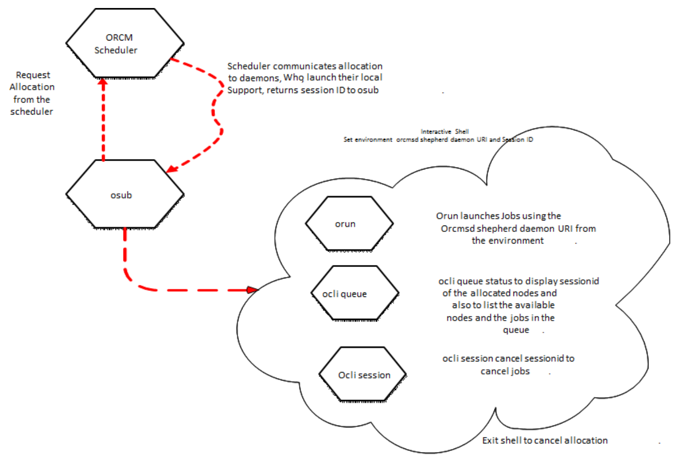
The following options are related to allocation requests sent to the scheduler:

- `--account <name>`: Provide an account to be charged.
- `--gid <ID>`: Provide a group ID to associate to the session.
- `--max-node <n>`: Specify the maximum number of nodes allowed in the allocation.
- `--min-node <n>`: Specify the minimum number of nodes required for the allocation.
- `--node <n>`: Same as `--min-node`.
- `--project <name>`: Provide a user-assigned project name.

3.1.4 osub

The *osub* tool provides a user interface for requesting new sessions to the scheduler. This tool can also be used for submitting batch jobs along with the session request. Upon a successful submission, a session ID will be returned.

The following diagram shows the tool's flow for the interactive option (`osub -i`):


ORCM Scheduler
Request Allocation from the scheduler
Scheduler communicates allocation to daemons, Whq launch their local Support, returns session ID to osub
osub
Interactive Shell
Set environment orcmsd shepherd daemon URI and Session ID
orun
Orun launches Jobs using the Orcmsd shepherd daemon URI from the environment
ocli queue
ocli queue status to display sessionid of the allocated nodes and also to list the available nodes and the jobs in the queue
Ocli session
ocli session cancel sessionid to cancel jobs
Exit shell to cancel allocation

- Users submit an allocation request using the *osub* tool with the interactive option.

- The *osub* tool will submit the request to the scheduler for an allocation and will receive a session ID.

- The *osub* interactive mode option will wait until the nodes get allocated.

- Session daemons are launched and a "VM ready" message is received from the "shepherd" daemon.

The *osub* tool also opens a shell prompt for the user to launch jobs with the allocated nodes (the "shepherd" daemon contact information is already set in the environment). The user can launch their jobs in this shell using the *orun* tool:

Step 1: the job launch tool reads the "shepherd" daemon's contact information from the environment and will continues to launch job processes by sending the job information to the session "shepherd" daemon. The job information includes: application name, application args, number of processes to execute and other environment settings for the application process.

Step 2: Upon job completion the session daemons will report the status back to the session "shepherd" daemon and wait for the command to terminate or launch additional processes.

Step 3: When the user exits the shell prompt, a cancel command is sent to the session "shepherd" daemon, which in turn will send a final "job complete and terminate" command to all the session daemons and exit.

The usage for the *osub* tool is as follows:

```
osub [OPTION]...
```

To submit a new allocation request, use the following syntax:

```
osub [--node <N>]  [-i]
```

Example:

```
% osub --node 4
```

This will send an allocation request (non-blocking) to the scheduler for 4 nodes and report the session ID back to the caller.

Example:

```
% osub --node 4  -i
```

This will send an allocation request (blocking) to the scheduler for 4 nodes and wait for the allocation. Upon allocation, it will open an interactive shell to allow the user to launch jobs using *orun*. The allocation will be released upon exiting the shell.

The following are the options for *osub*:

- `--account <name>`: Provide an account to be charged.
- `-b | --batch-run <script>`: Provide a path to a batch script file containing the list of applications to run.
- `-c | --constraints <constraints>` (TBD): Specify resource constraints to be applied. At the moment, this feature is not supported and will be supported in a future release.

- `--non-exclusive`: Used to specify that allocated nodes should not be shared with other sessions.

- `-f | --nodefile <file>` (TBD): Specify the path to a file containing the list of names of the candidate nodes. At the moment, this feature is not supported and will be supported in a future release.

- `--gid <ID>`: Provide a group ID to associate to the session.

- `-h | --help`: Print the *orun* help text.

- `-i | --interactive`: Request a interactive mode.

- `--omca <name> <value>`: MCA parameters, where `<name>` is the parameter name and `<value>` is the parameter value.

- `-P | --max-pe <n>`: Specify the maximum number of processing elements (PEs) allowed in the allocation.

- `--max-node <n>`: Specify the maximum number of nodes allowed in the allocation.

- `--node <n>`: Specify the minimum number of nodes required for the allocation.

- `-p | --pe <n>`: Specify the minimum number of processing elements (PEs) required for the allocation.

- `--project <name>`: Provide a user-assigned project name.

- `-s | --start <time>` (TBD): Specify the desired date/time for the allocation. At the moment this feature is not supported and the start time defaults to "now". This feature will be supported in a future release.

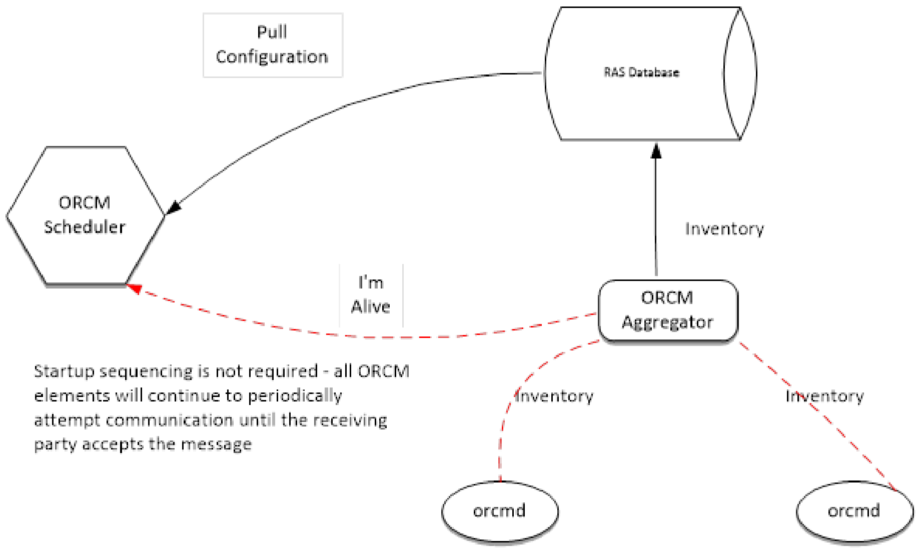
- `-v | --verbose`: Request verbose output.

- `-w | --walltime <time>`: Specify the maximum execution time for the job.

3.2 ORCM System Daemons

3.2.1 orcmd

ORCM runtime daemons are root level resource manager daemons launched in all compute nodes and aggregator nodes during cluster boot up process. These daemons collect RAS monitoring data from the compute nodes and logs in to the database.


Pull Configuration
RAS Database
ORCM Scheduler
Inventory
I'm Alive
ORCM Aggregator
Startup sequencing is not required - all ORCM elements will continue to periodically attempt communication until the receiving party accepts the message
Inventory
Inventory
orcmd
orcmd

At startup, each node boots its own ORCMD. The daemon detects the local node inventory and reports it to the aggregator in an initial “I’m alive” message. The aggregator enters the inventory in the database, and forwards the “I’m alive” message to the scheduler so it can construct an in-memory map of the cluster (nodes + inventory).

The orcmd daemon running on each node collects the RAS monitoring data using the sensor framework and its components, these components are configurable by using MCA parameters during build time and runtime.

The monitoring data collected from the compute nodes goes through a analytics framework for data reduction before gets stored in the database.

The ‘orcmd’ tool command line options are described below:

```
Usage: orcmd [OPTIONS]
  Open Resilient Cluster Manager Daemon

   -am <arg0>             Aggregate MCA parameter set file list
   -gomca|--gomca <arg0> <arg1>
                       Pass global MCA parameters that are applicable to
                       all contexts (arg0 is the parameter name; arg1 is
                             the parameter value)
```

```
-h|--help                This help message
   -omca|--omca <arg0> <arg1>
                    Pass context-specific MCA parameters; they are
                   considered global if --gomca is not used and only
                    one context is specified (arg0 is the parameter
                         name; arg1 is the parameter value)
-s|-site-file|--site-file <arg0>
                       Site configuration file for this orcm chain
-v|--verbose             Be verbose
-V|--version             Print version and exit
   -validate-config|--validate-config
                         Validate site file and exit

Note: To get the list of MCA parameters
  'orcm-info --param <arg0> <arg1>'
                         The first parameter is the
                      framework (or the keyword "all"); the second
                   parameter is the specific component name (or the
                         keyword "all").
```

Following is an example to start the orcmd daemon using a configurable mca parameter:

```
shell$ orcmd --omca sensor heartbeat,cpufreq,pwr,ipmi
```

The above will configure orcmd to select the following components for the sensor data collection: heartbeat, cpufreq, pwr and ipmi. The heartbeat module is method to send the periodically collected data to the aggregator, the time interval for data collection and reporting to the aggregator is another configurable parameter in the orcm configuration file.

3.2.2 orcmsched

The ORCM scheduler daemon is a root level resource manager daemon which runs on the SMS (system management server) node (or Head Node).

The ORCM scheduler is responsible for accepting submission requests, prioritizing the requests, and allocating resources once available to the next appropriate request. The scheduler framework will allow for different scheduler plugins to be loaded for required functionality.

The scheduler framework must take requests from session submission utilities and prioritize resource allocation appropriately. The scheduler framework is primarily used to prioritize access to resources. This framework will need to interface with the resource management framework to get updated resource states and session completion information.

The SCD operation can be split into 5 stages:

- Session initialization
- Session schedule
- Session allocated
- Session active
- Session terminated

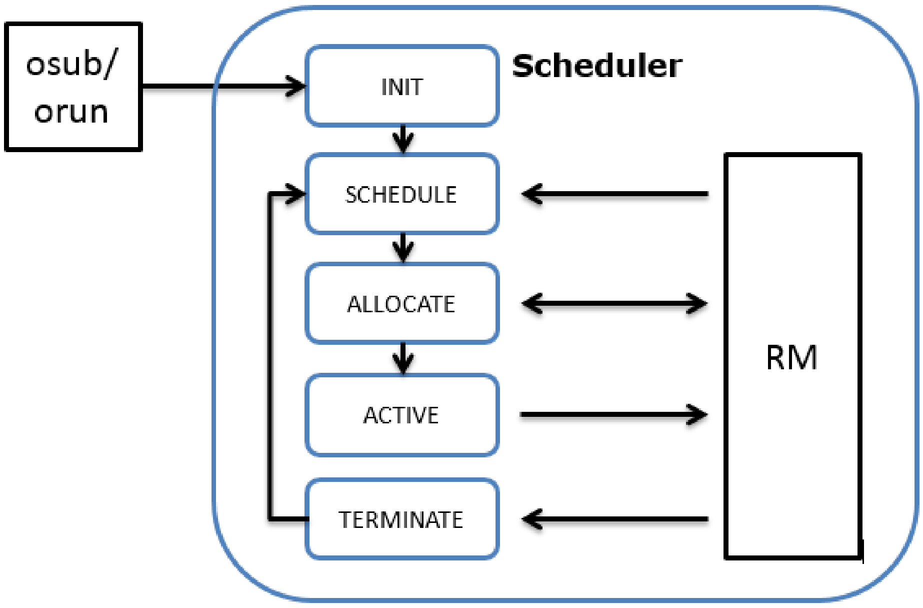
The initialization is called for all new requests to assign session id and queue the request based on priority and requested resources.

The schedule is called whenever there is a resource state change or a request queued. These state changes open the possibility for the next session to be launched.

The session allocated routine is invoked when the scheduler finds a session that can be run on the available resources. This allocated routine is responsible for informing the resource manager framework that the resources are allocated, and to make sure that the current session can get all required resources.

The session active routine is called once all resources are allocated to the given session. The active routine is responsible for launching the initial job steps across the allocation.

The resource manager framework will notify the scheduler framework upon job completion/termination with the terminated routine.



The 'orcmsched' tool command line options are described below:

```
orcmsched  [OPTION]...
-h|--help                    This help message
-s|--spin                Have the scheduler spin until we can connect a
                             debugger to it
-V|--version                 Print version and exit
--daemonize                Daemonize the scheduler into the background
--omca <arg0> <arg1>
                         Pass context-specific MCA parameters (arg0 is
                       the parameter name; arg1 is the parameter value)
```

Following is an example to start the orcmsched daemon using a configurable mca parameter:

```
shell$ orcmsched --omca scd_base_verbose 10
```

The above will configure orcmsched to increase verbosity in the scheduler framework.

orcmsched options:

- `--omca <arg0> <arg1>`: Pass context-specific MCA parameters (`arg0` is the parameter name; `arg1` is the parameter value)

3.3 ORCM Regex

The Open RCM node regex is specified when a set of nodes have a similar name prefix. The prefix is stripped and the numbering that follows is turned into a padding specification and hyphenated range, comma separated list, or combination of both. The padding specification is the total number of digits including 0 padded digits. The regex for a single node is the full nodename. For nodenames that have different padding, the full regex is a comma separated list of regex for each similarly padded node range.

For example:

```
node001,node002 : node[3:1-2]

node1,node2 : node[1:1-2]

node1,node2,node3,node4 : node[1:1-4]

node009,node010 : node[3:9-10]

node9,node10 : node[1:9],node[2:10]

node001,node002,node003,abc001,abc002 : node[3:1-3],abc[3:1-2]
```

3.4 ORCM CFGI User Guide

Introduction

Purpose of this document

This document hopes to instruct the reader on how to create and maintain an ORCM CFGI file.
This file is used to tell the ORCM software the configuration of the cluster.
This document is intended for ORCM developers and system administrators in charge
of installing ORCM and/or maintaining an ORCM installation.

This document assumes the reader is familiar with ORCM and its architecture and with the system they are trying to configure. This document is not a user guide or manual for ORCM.

Reference documents

The reader may find these reference documents helpful in understanding this user guide.

- Open MPI code internals http://www.open-mpi.org/video/?category=internals
- ORCM Developer Information https://svn.open-mpi.org/trac/orcm
- Backus-Naur Notation http://en.wikipedia.org/wiki/Extended_Backus%E2%80%93Naur_Form
- ORCM Regex https://github.com/open-mpi/orcm/wiki/3.3-ORCM-Regex

ORCM CFGI file syntax

The CFGI file in a plain text ASCII file written in XML format. The grammar in the file is defined by the Backus-Naur desciption shown hereafter. Details for each entry will be given afterwards.

- **configuration** ::= **version creation role junction** [**scheduler**]
- **naming** ::= **name**

- **junction** ::= **type naming** [**controller**] **junction**
- **controller** ::= **host** [**port**] [**cores**] [**log**] [**envar**] [**aggregator**]**mca-params***
- **scheduler** ::= **shost port** [**mca-params**] [**aggregator**]

In order to simplify the grammar specification, the basic data types are as follows:

- **string** ::= a sequence of ASCII characters with value in [33, 126]
- **regex** ::= a **string** where the characters, [ ] :, have special meaning.
  See the above link on ORCM Regex for details
- **uint** ::= a **string** of an unsigned 4-bytes integer as defined in the C language
- **ushort** ::= a **string** of an unsigned 2-bytes integer as defined in the C language
- **float** ::= a **string** of a floating point number
- **csList** ::= a **string** containing different fields separated by commas
  A **string** does not need to be delimited with double quotes if it does not contain whitespace characters. Otherwise, it needs to be delimited with double quotes, if the white spaces are to be preserved.
  –>At this time, single quotes cannot be used to delimit **string**.

Because the CFGI is written in XML, the above tokens will have an appropriate representation. For example, the command **configuration**, in XML, is represented by the pair of tags < configuration > ... < /configuration > where the ellipsis here will hold the rest of the configuration.

Details of the CFGI grammar

Each command line with be repeated here, and followed by details. Items with a full description, like , will be addressed when their full description are presented.

configuration ::= **version creation role junction** [**scheduler**]

- **version** ::= A **float** indicating the version and subversion numbers. The format is as follows "X.Y" where X and Y are any integer; Y is the sub-version.

- **creation** ::= A to identify time, date and who build this configuration files. At this time, the format is left to the user's discretion.

- **role** ::= RECORD . Only one key word at this time: RECORD. It is not case sensitive. The key word RECORD refers to the creation of a configuration record. There can be only one configuration with the modifier RECORD.

naming ::= **name**

- **name** ::= A **regex** expanding to one or more names. A special character, '@', is reserved to indicate that the name of the parent junction in the hierarchy will be used to replace the '@' character. For example, if the parent junction is named "rack1" and its child junction as for ="@_node", then the child junction's name string is "rack1_node", where "rack1" of the parent replaced the '@' character.

junction ::= **type naming** [**controller**] **junction**

- **type** ::= ("cluster"|"row"|"rack"|"node") . "cluster" is always the root of the hierarchy. There can only be one junction of type cluster. It must always be mentioned.
  _"row" & "rack" are used in a 4-tiers hierarchy: cluster, row, rack, node. "cluster" is always the root of the hierarchy; and there can be only one cluster. "node" are always leaf points in the hierarchy. With "cluster" as root and "node" as leaf, the hierarchy always has a unique start point and well defined ending points. Each junction must have a name. Furthermore the name of siblings must be different. For example, if a cluster contains two rows, these rows must not have the same name. Currently, at most 4-tiers of hierarchy are supported and exactly the following order: cluster, row, rack, node. If a row or a rack is omitted, a fictitious equivalent will be automatically inserted.

controller ::= **host port** [**cores**] [**log**] [**envar**] **mca-params***

- **host** ::= A **regex** of an actual IP address or an IP resolvable name, of the machine hosting the controller. It can include the hierarchical operator '@' as explained in . If it does, this '@' operator will refer to the immediate junction hosting this controller. There is a strong relationship between the hosting junction name and its controller host value.
  If the hosting junction's name is a **regex**, one must use '@' as the controller host value.
  If the hosting junction's name is a **cstring** and not a **regex**, then the controller host value can also be a **cstring**.

EXCEPTION: For node junction, if the hosting junction's name is a **cstring** and not a **regex**, then its controller host value must have be exactly that node junction's name.
NOTE: When in doubt about a controller host name, use '@'.

- **port** ::= an **ushort** for the ID of the listened port.
- **cores** ::= a **regex** specifying the ID of the cores to be used
- **log** ::= **string** storing the output log file path and name.
- **envar** ::= **cslist** of **string** of environment variable and their values.
- **mca-params** ::= **cslist** of **string**, where each **string** contains a single tag-value pairs, with the "=" as separator. Use multiple statements for multiple specification. All multiple statements must be grouped together in the XML file.
- **aggregator** ::= (yes|no) . If yes, than that particular junction or scheduler will designated as an accumulator.

scheduler ::= **shost port** [**mca-params**] [**aggregator**] **queues***

- **shost** ::= The same as **host** but specific to the **scheduler**

XML example

Typically the file is written in the directory $PATH2ORCM/orcm/etc/orcm-site.xml. ORCM will look for this file by name.

The ORCM XML parser only parse a simplified XML format. The simplification are as follows:

- XML attributes are not supported
- Quoted strings can only use double quotes ".

A prototype CFGI file written in XML is provided hereafter. It presents a cluster with the following configuration:

- A 4-tier hierarchy: cluster, row, rack, junction
- 1 Scheduler

- 1 row named row1 without a controller
- 4 racks in the single row, locally called “agg01”, “agg02”, “agg03”, “agg04”
- 1024 nodes equally distributed among the racks, locally called “node0000”, ..., “node1023”
- Each rack and node has a controller

The example has in-lined comments which provides further details.

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>
    <!-- Version is fixed to 3.0 -->
    <version>3.0</version>
    <!-- We need a single RECORD -->
    <role>RECORD</role>
    <junction>
        <!-- We need a single root for the hierarchy -->
        <type>cluster</type>
        <name>master3</name>
        <junction>
            <type>row</type>
            <name>row1</name>
            <junction>
                <type>rack</type>
                <name>agg01</name>
                <controller>
                    <host>agg01</host>
                    <port>55805</port>
                    <aggregator>yes</aggregator>
                </controller>
                <junction>
                    <type>node</type>
                    <name>node[4:0-255]</name>
                    <controller>
                <!-- This controller takes its host name from its row’s name -->
                   <!-- The @ operator does the unique selection -->
                        <host>@</host>
                        <port>55805</port>
                        <aggregator>no</aggregator>
                    </controller>
                </junction>
            </junction>
            <junction>
                <type>rack</type>
                <name>agg02</name>
                <controller>
                    <host>agg02</host>
                    <port>55805</port>
                    <aggregator>yes</aggregator>
                </controller>
                <junction>
                    <type>node</type>
                    <name>node[4:256-511]</name>
                    <controller>
                        <host>@</host>
                        <port>55805</port>
                        <aggregator>no</aggregator>
                    </controller>
                </junction>
            </junction>
            <junction>
                <type>rack</type>
                <name>agg03</name>
                <controller>
                    <host>agg03</host>
                    <port>55805</port>
                    <aggregator>yes</aggregator>
                </controller>
                <junction>
                    <type>node</type>
                    <name>node[4:512-767]</name>
                    <controller>
                        <host>@</host>
                        <port>55805</port>
                        <aggregator>no</aggregator>
                    </controller>
                </junction>
            </junction>
            <junction>
                <type>rack</type>
                <name>agg04</name>
                <controller>
                    <host>agg04</host>
                    <port>55805</port>
                    <aggregator>yes</aggregator>
                </controller>
                <junction>
                    <type>node</type>
                    <name>node[4:768-1023]</name>
```

```
                    <controller>
                        <host>@</host>
                        <port>55805</port>
                        <aggregator>no</aggregator>
                    </controller>
                </junction>
            </junction>
        </junction>
    </junction>
    <scheduler>
     <!-shost identifies the node that houses the ORCM scheduler. Only one allowed -->
        <shost>master01</shost>
        <port>55820</port>
    </scheduler>
</configuration>
```