Version 0.5.0 Release Notes

V0.5 release is focused on Database and RAS Monitoring capabilities.

Base Operating System for v0.5 Release Validation: Centos 6.5

Outstanding issues known for this release:

- Issue 53 *IPMI from ipmiutil-2.9.4-1.src.rpm spews extra log messages*
- Issue 70 *Orcmd will complain if a compute node connects and no orcmsched is running*
- Issue 71 *Componentpower plugin always logs zeros for the initial samples*
- Issue 72 *Componentpower plugin does not log units*
- Issue 77 *Database scripts are not included in tar ball*

Database files for release available at:
https://github.com/open-mpi/orcm/tree/v0.5.0/contrib/database

Non-validated binaries included but not supported in this release:

- orun
- osub
- octl
- oqueue

1 ORCM

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.

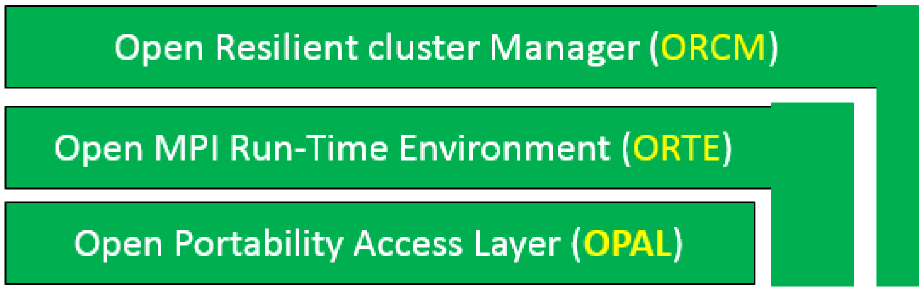
Throughout this time, ORCM has retained ORTE's ability to perform scalable launch and process monitoring, and ORTE's support for a variety of scheduling environments. We are now in the process of validating and extending ORCM to provide both monitoring and launch support for exascale environments.

1.2 Overview

- Architecture
- Core Features

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

- cfgi: cluster configuration management
- db: Database 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
- 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. oqueue – A command line tool to display the resource states, scheduler maintained session queues and the current running session.

1.3 Database

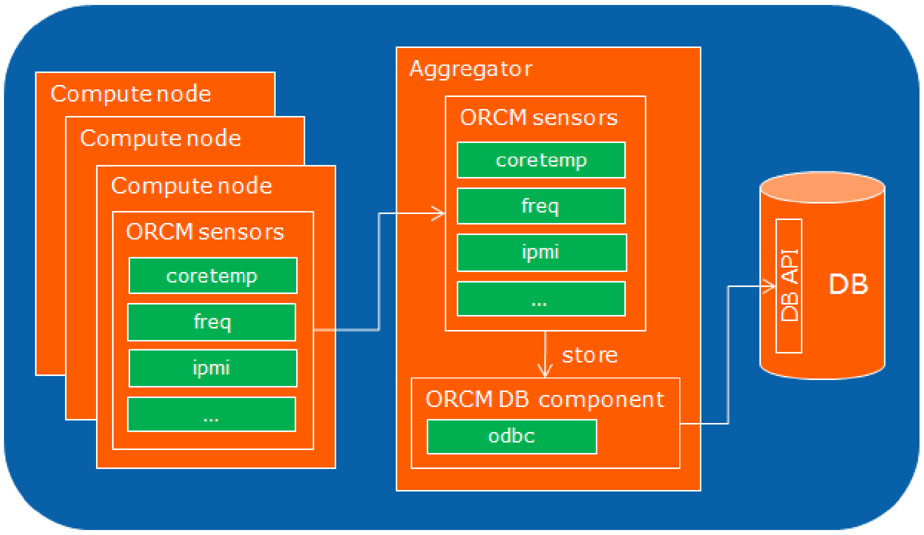
The database is a key component for supporting the various applications of the cluster software stack. So it is just as important to ORCM as it provides support for its system and resource management tasks by storing hardware inventory, system environmental data, RAS events (including error conditions) 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.



2 Build and Installation Guide

2.1 ORCM Build and Installation

2.1.01 Build From SRPMS

Note: See [[2.1.5-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$ 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
```

To install from the binary rpm created by rpmbuild run the following command as root

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

This will install ORCM into /opt/open-rcm/<version>/. For this example, the 0.5.0 ORCM binaries would be installed into

```
/opt/open-rcm/0.5.0/bin
```

The following environment variables will need to be set:

```
LD_LIBRARY_PATH=/opt/open-rcm/0.5.0/lib64:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
PATH=/opt/open-rcm/0.5.0/bin:$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 open-rcm-<version>.rpm \
           --relocate /opt/open-rcm=<new location>
```

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

```
OPAL_PREFIX=<new location>
OPAL_LIBDIR=<new_location>/lib64
LD_LIBRARY_PATH=<new_location>/lib64:$LD_LIBRARY_PATH
export OPAL_PREFIX OPAL_LIBDIR LD_LIBRARY_PATH
PATH=<new location>/bin:$PATH:
export PATH
```

2.1.03 Build From Source Tar Files

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/. It is recommended to use –prefix /opt/open-rcm/<version> when running configure on tar file based installs.

2.1.04 Build From GitHub Repo

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
```

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.

2.1.05 Build Dependencies

2.1.5.1 Sigar, ODBC, and SSL Install SIGAR and ODBC drivers.

2.1.5.1.1 CentOS

```
shell$ yum install libtool-ltdl openssl openssl-devel
shell$ yum install sigar sigar-devel
shell$ yum install unixODBC unixODBC-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-2.9.4-1.src.rpm
rpmbuild --rebuild ipmiutil-2.9.4-1.src.rpm
rpm -ivh ~/rpmbuild/RPMS/x86_64/ipmiutil-2.9.4-1.el6.x86_64.rpm
rpm -ivh ~/rpmbuild/RPMS/x86_64/ipmiutil-devel-2.9.4-1.el6.x86_64.rpm
```

The following optional github repo contains a version of IPMI Util library with patches to resolve an issue found in the 2.9.4 IPMU Util library that causes extra lan connection messages: https://github.com/vpedabal/ipmiutil_orcm.git

```
git clone https://github.com/vpedabal/ipmiutil_orcm.git
cd ipmiutil_orcm
./beforeconf.sh
./configure --libdir=/usr/lib/x86_64-linux-gnu
make -j 8
make install
```

2.1.5.3 Database Dependencies ODBC driver and database schema are optional for logging the RAS monitoring data to the database (mysql, postgres).

mysql or postgres – optional database servers for data logging.

See [[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: https://github.com/open-mpi/orcm/blob/v0.5.0/contrib/platform/intel/hillsboro/orcm-linux)
2. A default MCA parameter file for selecting default MCA parameters for ORCM
   (ex: https://github.com/open-mpi/orcm/blob/v0.5.0/contrib/platform/intel/hillsboro/orcm-linux.conf)
3. A ORCM site configuration XML file which configures the cluster nodes, aggregator, scheduler and the compute nodes
   (ex: https://github.com/open-mpi/orcm/blob/v0.5.0/contrib/platform/intel/hillsboro/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/<version>/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 BMC and IPMI Libraries

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
```

2.1.09 Startup Instructions

Manual Head Node (HN) and Compute Node (CN) Setup Set up your shared library path and executable path to point to the ORCM install. This will be needed on all nodes.

```
LD_LIBRARY_PATH=/opt/open-rcm/<version>/lib64:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
PATH=/opt/open-rcm/<version>/bin:$PATH:
export PATH
```

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

ORCM Scheduler

```
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],
  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/0.5.0/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.

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:

- `-mca <param_name> <value>` option and
- `-gmca <param_name> <value>`

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 set OPENMPI_<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 --mca  parameter1-name  parameter1-value  \
                [--[mca parameter2_name parameter2_value] ...
```

For example:

```
shell$ opal_db --mca db_base_verbose 6 \
              --mca db_hash_store_priority 2 \
              --mca db_hash_fetch_priority 2 \
              --mca db_postgres_store_priority 5 \
              --mca db_postgres_fetch_priority 5
```

2.2 Database Installation

- Database Server
- Database Connectivity
- ORCM Configuration

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 seprate 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 PostreSQL 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.

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 temporarirly)
       * Execute this command remotely

6. Verify the installation
   - 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`
  - The following tables should be listed:
    * data_item
    * data_sample
    * event
    * event_type
    * fru
    * fru_type
    * job
    * job_node
    * maintenance_record
    * node

2.2.2 Database Connectivity

This section provides instructions to configure clients (e.g. an aggregator node) 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).

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”

2.2.2.1 Software Requirements

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

2.2.2.1.1 Requirements for PostgreSQL NOTE: Client may be installed on any machine for administrative tasks: testing the database connection, data and schema management, etc.
Database client packages are generally required on nodes with the role of aggregator

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

2.2.2.3 Configuring the Clients for ODBC Connectivity 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.

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.

2.2.2.4 PostgreSQL Installation

1. 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
2. Register the PostgreSQL ODBC driver
   - Edit the "psql_odbc_driver.ini" file and fill in the required prameters:
     - 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
3. 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

4. Verify the installation

    - Use the *isql* command-line utility provided by unixODBC to connect to the database:
        - `isql <name of the DSN that was configured> 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

2.2.3 ORCM Configuration

This section provides instructions to configure the ORCM DB component to connect to the ORCM DB. This is required for clients that will be running ORCM and need to communicate with the database (ORCM with the DB component enabled).

2.2.3.1 Configuring the ORCM DB Component Specifically, the DB component within the ORCM DB framework that needs to be configured, is the 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: `<user>:<password>`.
- 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`.

These MCA parameters may be specified in:

- The “openmpi-mca-params.conf” file
- The “orcm-site.xml” file
- The command line via the ORCM “-mca” command-line parameter

2.2.4 Database Distro specific examples Centos 6.5

```
## CentOS 6 postgres install

vi /etc/yum.repos.d/CentOS-Base.repo

# 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.noarch.rpm

yum install postgresql93-server

## To remove

# rm -rf /var/lib/pgsql

# yum remove postgresql93-server

# yum remove postgresql93-odbc
```

```
## InitDB and adding an orcmuser

service postgresql-9.3 initdb

vi /var/lib/pgsql/9.3/data/pg_hba.conf

# Must be before other entries; add local all orcmuser trust

# if changing these after the postgres service is started

# service postgresql-9.3 restart

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 ...
```

```
## Postgres user DB and DB user setup; terminal #2

su - postgres

dropdb orcmdb

dropuser orcmuser

createuser orcmuser

createdb --owner orcmuser orcmdb
```

```
## ORCM specific database creation script

# Get scripts from https://github.com/open-mpi/orcm/tree/v0.5.0/contrib/database

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


## Client access setup

yum install postgresql93-odbc


rpm -ql postgresql93-odbc | grep psqlodbc.so

vi psql_odbc_driver.ini # Add psqlodbc.so with path

odbcinst -i -d -f psql_odbc_driver.ini

vi orcmdb_psql.ini # Add server localhost and psqlodbc.so with path

odbcinst -i -s -f orcmdb_psql.ini -h

odbcinst -s -q # List Data Source Names (DSNs)

vi /var/lib/pgsql/9.3/data/pg_hba.conf

# Add "host all all 127.0.0.1/32 trust" and "host all all ::1/128 trust" for IPv4 and IPv6

# Restart!!

service postgresql-9.3 restart


isql -v orcmdb_psql orcmuser # Test ODBC access to the DB.  Try select * from data_sample.

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


## Run ORCM

orcmd -mca sensor heartbeat,coretemp \

        -mca db_base_verbose 100 \

        -mca db_odbc_dsn orcmdb_psql \

        -mca db_odbc_user orcmuser:orc \

        -mca db_odbc_table data_sample


## Query the DB

psql -d orcmdb -U orcmuser [-W]

# Query sample data from RAS monitoring
```

```
select node.hostname,
       data_item.name,
       data_sample.value_num,
       data_sample.units,
       data_sample.value_str
from data_sample
       inner join node on
          node.node_id = data_sample.node_id
       inner join data_item on
          data_item.data_item_id = data_sample.data_item_id;


# Delete sample data from table
delete from data_sample;


# Get number of sample data rows from table
select count(*) from data_sample;


#### Cut here ####


## Enabling network access to the DB
vi /var/lib/pgsql/9.teuser3/data/postgresql.conf # Add listen_addresses = '*'


## Adding a password
vi /var/lib/pgsql/9.3/data/pg_hba.conf # Use "md5" instead of "trust"
createuser -P orcmuser # Add -P if you want to create with a password
```

2.3 RAS Monitoring

- Enabling RAS Monitoring

2.3.1 Enabling RAS Monitoring

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 -mca 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.

2.3.1.1 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

2.3.1.2 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

2.3.1.3 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.

2.3.1.4 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

2.3.1.5 sigar This component reads memory, swap memory, cpu load, disk, and network data for each compute node.

MCA parameters:

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

2.3.1.6 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.

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.

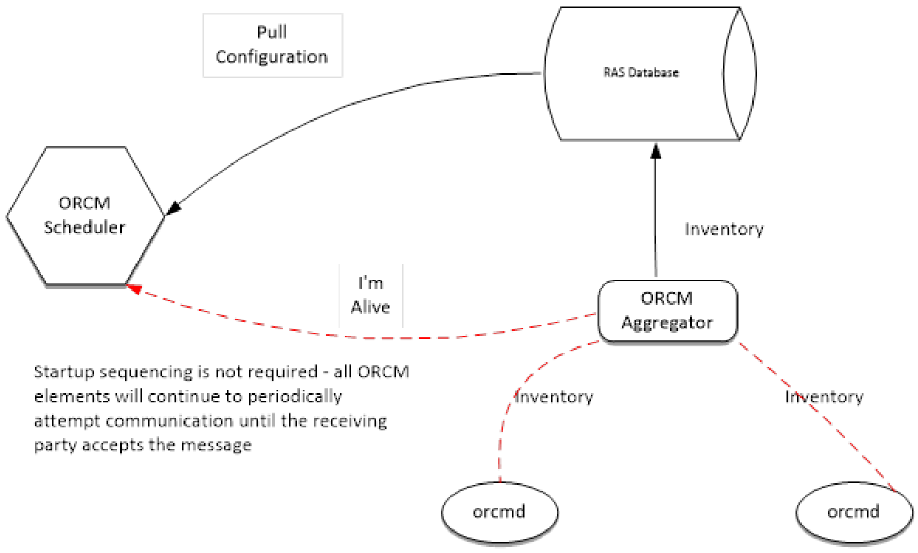
- sensor_resusage_log_process_stats: Whether the sampled process status information is to be sent to the log function for further processing.
- sensor_resusage_test: Enable logging a random test sample by the plugin for testing sensor to database connectivity

3 ORCM Tools User Guide

- orcmd
- orcmsched

3.1 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:

```
orcmd [OPTION]...
```

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

```
shell$ orcmd -mca 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.

orcmd options:

- `-h, --help`: Display help for this command
- `-v, --verbose`: Be verbose
- `-V, --version`: Print version number. If no other arguments are given, this will also cause orun to exit.
- `-gmca, --gmca <arg0> <arg1>`: Pass global MCA parameters that are applicable to all contexts (`arg0` is the parameter name; `arg1` is the parameter value)
- `-mca, --mca <arg0> <arg1>`: Pass context-specific MCA parameters; they are considered global if only one context is specified (`arg0` is the parameter name; `arg1` is the parameter value)

3.2 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

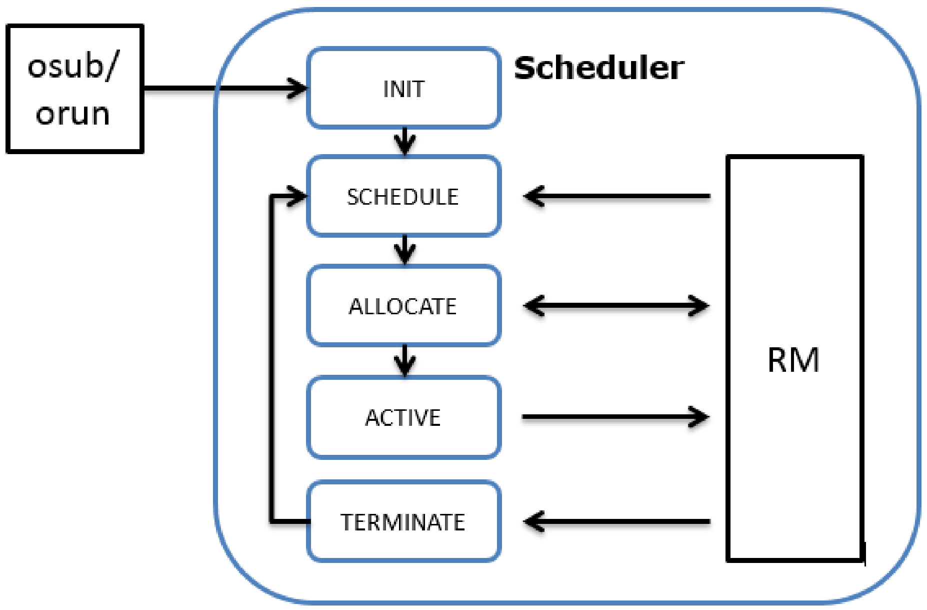
The initialization is called for all new requests to assign session id and queue the request based on priority and requested resources.

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]...
```

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

```
shell$ orcmsched -mca db odbc
```

The above will configure orcmsched to use the mysql database backend.

orcmsched options:

- `-gmca, --gmca <arg0> <arg1>`: Pass global MCA parameters that are applicable to all contexts (`arg0` is the parameter name; `arg1` is the parameter value)
- `-mca, --mca <arg0> <arg1>`: Pass context-specific MCA parameters; they are considered global if only one context is specified (`arg0` is the parameter name; `arg1` is the parameter value)