# *The Evolution of Cray Management Services*

Tara Fly, Alan Mutschelknaus, Andrew Barry and John Navitsky
OS/IO
Cray, Inc.
Seattle, WA USA
e-mail: {tara, alanm, abarry, johnn}@cray.com

***Abstract*— Cray Management Services is quickly evolving to address the changing nature of Cray Systems. NodeKares adds advanced features to support gang scheduling, reservation and application level health checking. Reservation level node health improves node availability by allowing administrators to move tests between application and reservation boundaries. Lightweight Log Manager provides more complete and standardized log collection. Configuration hooks allow administrators to extend Lightweight Log Management for third-party logs, and to aggregate logs to external log hosts. Modular xtdumpsys will provide an extensible framework for system dumping. Administrators can quickly extend plugins, and define plugin sets that allow them to target data gathering and collection based on classes of failures. Resource Utilization Reporting provides a scalable, extensible framework for data collection, including power management, GPU utilization, and application resource utilization data. This paper presents these new features: covering configuration and benefits.**

***Resource Utilization Reporting, Lightweight Log Manager, Node Health Checker, xtdumpsys, SEC***

## I. INTRODUCTION

Today's Cray XE6 and XK7 systems provide petascale performance to customers to support real-world science. System uptimes and resiliency are better than ever. Cray Linux Environment (CLE) now supports shared libraries and Independent Service Vendor (ISV) applications. Scientific users can compile and run application codes without porting. As the availability and number of users increase, the need for improved monitoring, management, accounting and fault diagnosis increases. To address this change, Cray System's Management team has provided a number of foundational features in CLE4. The fundamental architecture of all of these features is the same: scalable data collection and launch mechanisms, centralized data collection, and customer-extensible, plugin-based architectures. Cray can provide a core set of plugins, with easy, online methods for the sites to extend these capabilities.

This paper will review the key infrastructure enhancements in CLE4. Significant infrastructure improvements for diagnosis include: lightweight log manager, modular xtdumpsys, and enhancements to Node Health Checker. It will also present an introduction to the accounting Resource Utilization Reporting (RUR) roadmap and architecture. RUR provides a scalable, plugin architecture for data collection.

## II. FEATURE OVERVIEW

This section provides an overview of new features in Cray Management Services. CMS has adopted an iterative development model, providing critical infrastructure more quickly, and then adding capabilities on a pure release basis. Capabilities leverage scalable framework, core functionality provided through plugins, and hooks to allow customization if desired.

### *A. Lightweight Log Manager*

Lightweight log manager (LLM) provides an easy, reliable and resilient mechanism to gather CLE native system messages (logs) from service nodes and embedded cabinet controllers.  The LLM infrastructure provides filtering (discarding), persistent logging, and optionally forwarding to a site log host. The infrastructure will only queue to persistent storage on the XE if it temporarily cannot forward to the SMW. LLM is built on top of rsyslog, and formats all messages using the RFC-5424 standard. Figure 1 illustrates a redundant LLM implementation including external log forwarding.

The log collection on the SMW places logs in a LSB-compliant name-spaced directory corresponding to the current boot session with the format shown in Figure 2.

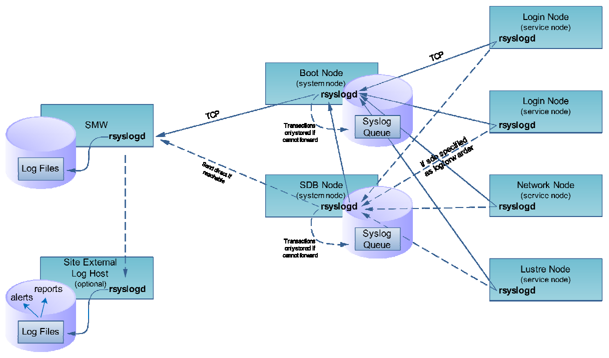

Figure 1. Lightweight Log Management Architecture

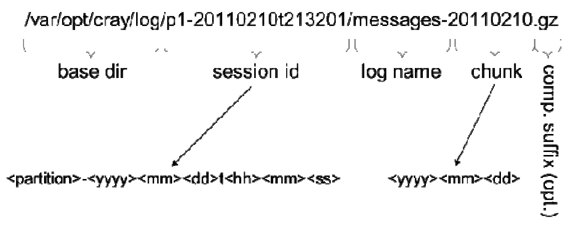
/var/opt/cray/log/p1-20110210t213201/messages-20110210.gz
base dir
session id
log name
chunk
comp. suffix (opt.)
<partition>-<yyyy><mm><dd>t<hh><mm><ss>
<yyyy><mm><dd>

Figure 2. Log message format

Logs are compressed after a configurable time interval, and can be configured to be deleted. By default, compression happens after 30 days, and logs are always retained. By default, LLM allows rotates all logs daily. However, it can also be configured to rotate logs more frequently, to a granularity of one minute location. Different log files can use different rotation frequencies.

LLM utilizes rsyslog as its delivery mechanism. This decision was made because SuSE announced deprecation of syslog-ng, with migration to rsyslog in SLES12. Overall scalability of rsyslog to the existing syslog-ng, RFC-5424 support, and transaction queuing support also played a role in the decision. LLM provides in-order message delivery under normal operating conditions.

### 1) *Controller Log Forwarding*

SMW 7.0UP01 also includes controller log forwarding. Currently log files from the controllers are stored on a tmpfs files system. Because of space limitations with the tmpfs file system, only a limited number of files can be stored. This means that when the number of files exceeds a preset limit, older log data is lost. With syslog files, this often happens, and when a problem condition occurs, information that would be useful in determining the cause of the problem is lost before it can be viewed. Also, because controller log files are stored on a tmpfs files system, these files are lost whenever the controller is rebooted.

LLM consolidates these controller logs into the main log repository. Doing so insures that the logs are available on the SMW in case of a system failure. The base LLM rules for forwarding CLE and SMW log files are provided in a turn-key fashion, which provides ease of deployment to site personnel. Sites can, additionally, implement custom extensions to the LLM system through site local customizations. These extensions include the ability to read/monitor any arbitrary file, including those provided by 3rd-party software, and forward that log to the site log host, or add it to the existing log stream. These logs can then be post-processed, allowing improved fault diagnosis.

### 2) *LLM Scalability*

The default LLM configuration file separates high volume logs into multiple streams. Rsyslog creates a listener for each port, preventing network bottlenecks to a single port. During development, we ran two tests on a 4 socket SMW containing 4-core 2.4 GHz AMD processors, 16GB total ram, 512KB cache, writing to a 7200 RPM SATA drive. Two different loads were generated: 72K messages / second and 200K messages per second. The CPU utilization was compared to a simple program which wrote a RFC-5424 message to a single log file, at the same message rates.

At 72K messages/sec. the CPU is 80% idle and rsyslog accumulated 17 seconds of total CPU time vs 10 seconds for the C program, cloop. Rsyslog seems to handle the load efficiently and without a large impact on the system. Each approach wrote 4GB of data. At 200K messages/sec., the CPU is 54% idle and rsyslog accumulated 46 seconds of total CPU time vs 30 seconds for cloop. Each approach wrote 11GB of data.

In real world situations, LLM has handled and processed data rates of more than 200 GB/day with successful log delivery.

### 3) *LLM Resiliency and Fault Tolerance*

The LLM implementation also provides delivery message delivery, even if the syslog agent becomes unavailable. The LLM libraries first attempt to contact the syslog host. If the host cannot be reached, messages are written to an in-memory message queue, and then to out to disk. An intermediate agent, llmrd, monitors for recovery of rsyslogd, and then delivers the queued messages. Figure 3 illustrates LLM delivery paths including the resiliency agent.

LLM does not guarantee that the queued messages will be delivered in order time during a recovery period, as the application thread may be delivery new messages while the llmrd is replaying the messages from disk, resulting in some interspersing of logs. When considering the alternative of dropped message, this was considered a reasonable tradeoff. Lastly, all LLM system processes (rsyslogd, llmrd) are automatically monitored, and the system will restart the services if they are not running.

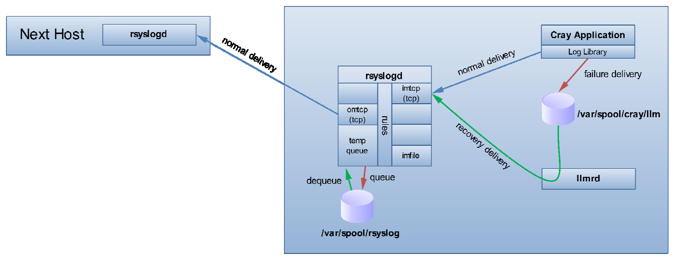

Figure 3. LLM Resiliency Model

## B. *System Dump Enhancements*

Centralized log management provides infrastructure for improved failure analysis in a centralized location on the SMW. Delivery of L0 and L1 controller messages provides much needed data to isolate failures in embedded controller hardware and software components. Systems are growing in overall size, and total system uptime is increasing. As a result, the total amount of log data gathered for any given boot session has grown significantly. Successful and efficient fault analysis requires gathering and providing all data relevant to the fault, and only the data relevant to the fault. The goal is to provide the smallest possible, complete data package to service and engineering personnel.

The existing xtdumpsys data collection tool was developed first for the original XT3 system deliveries. The program included a large core, implemented in TCL. The original xtdumpsys includes some extensibility hook locations. These hook locations occur at the beginning of processing the core logic of the script, after hardware level data and available controller logs are gathered, and after all of the core script logic has executed. These plugins live in a single file. There are a number of tuning options that have been added over the years to improve scalability; however, the result is an increasingly unwieldy invocation line.

Today, the total number of cabinets in the largest XE install is approximately three times as large as the largest XT3 machine. In addition, Cray XE modules are dual-socket, resulting in six times as many processors. With the additional log data provided by LLM, the total aggregate logs size for a single boot session can easily reach 100GB or more on our largest machines.

### 1) *System Dump Architecture*

To provide a framework for the future, Cray has developed a new, modular xtdumpsys architecture, which deploys all plugins relevant to the current execution phase. The core logic is responsible for determining the session context, handling any user specified command line options, actual execution of the system dump, and overall process cleanup. System dump occurs in four stages: (i) initialization, (ii) analyze, (iii) gather and (iv) finalization. Figure 4 depicts xtdumpsys execution flow.

During the initialization phase, the xtdumpsys core determines the overall session context for the dump, processes and validates any user specified options then starts the system dump. System dumping occurs in two phases: an analyze phase and a gather phase. The analyze phase analyzes system logs and current controller health to determine what nodes logs of interest. It then iterates through a series of data collection plugins. The collection plugins are grouped by numerical precedence. All plugins at a given precedence level are eligible to run in parallel.

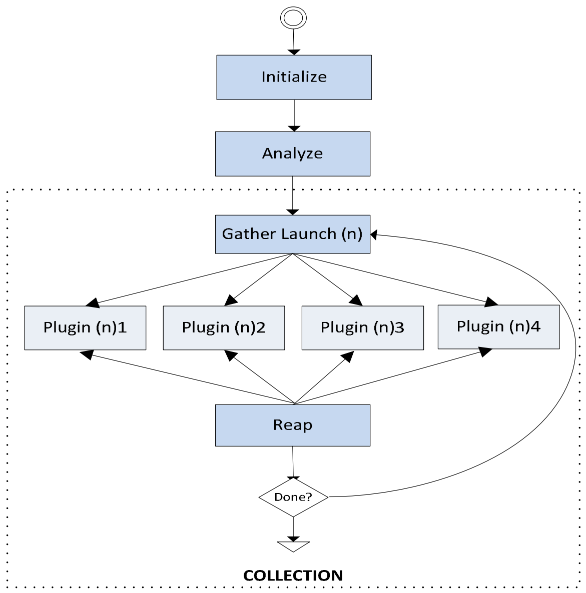

Figure 4. System Dump Execution

This logic repeats until all plugins have been executed, and have completed execution. Finally, xtdumpsys will finalize the system dump, and notify the administrator of its location.

### 2) *Configurability and Extensibility*

Site administrators can extend the functionality of xtdumpsys with site specific plugins. These are placed in a global directory, /etc/opt/cray/dumpsys/local. These plugins are preserved between upgrades. Administrators can also specify default settings and system parameters such as plugin timeout and concurrency.

### 3) *Performance*

There are three significant performance enhancements in the updated xtdumpsys. The first is that plugins can be run in parallel. The default configuration file specifies that four processes at the given precedence level can run at a single time. Administrators can modify that default number of concurrent plugin processes on a system wide basis, and for a specific dump invocation.

The second major performance feature is the ability to run dump scenarios. Dump scenarios provide the ability to perform a dump that contains only specific data relevant to a given failure mode. These scenarios are simply configuration files listing plugins corresponding to a particular failure mode. The first release provides scenarios for five common failure modes: HSN congestion, admindown node(s), down node(s), panicked node(s), and power and cooling failures.

The final performance enhancement is provided through the ability to specify log windows, which specify gathering only a portion of the current boot logs. There are three related runtime options: log_window, log_window_start and log_window_end. The log_window option indicates how far to look back from the current time when gathering logs. The log_window_start and log_window_end options cause xtdumpsys to gather logs only within the specified period of time. Note though that xtdumpsys correlates logs to a system dump through symbolic links, so it will apply this granularity as best as it can based on the frequency that LLM rotates its logs, and will only reduce sizes if LLM is configured on for the system. Use of these scenarios will greatly reduce the size of system dumps, especially if the system has been up for a long time since the last reboot.

The updated xtdumpsys is implemented in python. Cray Management Services is standardizing on the python for new complex tools development, as it has a broader user base, and is easy to read. At the original time of selection, there were also a number of language deficiencies in the distribution version of TCL/TK which exhibited worse comparative performance to either perl or python. The majority of these have been significantly improved in TCL 8.5, however, diminishing the weight of language performance when choosing the implementation language. The distribution includes a template plugin that executes a shell script, which can be easily modified by administrators who are not familiar with the python programming languages.

### C. NodeKares

With 4.1 UP01, NodeKares (otherwise known as node health checker) introduced support for reservation level node health. Reservation level node health allows administrators to set up a test to run on either a reservation level boundary or a batch job boundary. Certain tests, such as the free memory test are better fit for a job boundary. A given problem may involve multiple application steps. For many ISV applications, each of these job steps leave data in memory or tmpfs, to be used be used in a subsequent job step. Checking available free memory on job step boundaries could result in false Admindown of nodes for certain workloads. As a result, many sites set memory tests to notify only of the state. By moving the test to the reservation level, Node Health Checker (NHC) diminishes false positives, and allows improved hands-off management of the system.

#### 1) Configuration Changes

Figure 5 shows the changes in node health configuration file syntax, including the 'Sets' and 'advanced_features' aspects. The CLE installer will convert existing node health configuration files to the new syntax on an upgrade.

NHC Configuration file has moved to an 'ini-style' formatting on the service node and compute node image. Tests may have a "Sets" option. Sets provide a way of assigning tests to logical groupings.

```
nhcon: on
dumpdon: off
suspectenable: y
Memory: Log 20 30 30 600
```

```
[Options]
advanced_features: on
nhcon: on
dumpdon: off
suspectenable: y

[Memory]
Action: Log
WarnTime: 20
Timeout: 30
RestartDelay: 30
Threshold: 600
Sets: Reservation
```

Figure 5. Node Health Configuration Enhancements (old and new)

Typically the sets will be grouped into the "application" and "reservation" sets. If no "Sets" option is used, NHC will default to the "application" set. New Sets can be created by using them. Tests are now separate binaries on the compute node and can be run manually if needed.

#### 2) Node Health Check Operational Modes

This section describes the various NHC invocation modes, with the introduction of reservation level test support.

##### a) Application NHC: By ALPS after application exits

Application NHC can be configured based on the application's exit code. Typically set to non-zero exit codes, such that NHC will run tests if the program encounters any errors or exits because of a signal. Alternatively, NHC can be set to always run after application exit. During this run of NHC a set of tests are performed relevant to post-application health checking. Note that tests may be configured differently depending on machine type, software release, etc. Typically tests run during Application NHC include:

- Application test
- Filesystem test
- Alps test
- Accelerator test
- ugni test

##### b) Reservation NHC: By ALPS at the end of a reservation

Reservation mode will always run regardless of the exit code from the application. Reservation NHC operates at the end of the reservation so that NHC can run health checks and ensure the next time the node is used in a reservation it is healthy and cleaned-up.

Default tests run in reservation mode are:

- Memory test
- Reservation test

### c) *Recent Test Additions*

Recent additions to the node health test suite are described in this section.

#### UGNI TEST

Tests the ugni interface on compute nodes, and by extension the proper operation of parts of the nic. Currently ugni_nhc_plugins only performs one test, where a test datagram packet is sent out to the node's nic and back again.

#### RESERVATION TEST

The reservation test checks that the reservation has been cleaned-up. If it is still present, NHC will attempt to end the reservation by invoking the kernel's Compute node cleanup utilities. Note that in most cases, Alps is able to request kernel cleanup first and NHC only checks that cleanup occurred.

### d) *NHC on Boot NHC on Boot: By init.d scripts*

A local instance of NHC to the compute node, xtnhc, is launched to perform basic checkups. This unique invocation of NHC does not follow the usual paradigm of NHC anatomy. There is no fanout tree and no xtcheckhealth is used for coordination. If the tests are successful, NHC will send an event that alerts xtdbsyncd to set the node to the UP state. It is important to note that this version of NHC uses a configuration file local to the compute node, located at /etc/opt/cray/nodehealth/nodehealth.conf. Changes made to the shared root copy of the NHC configuration file will not pertain to NHC on boot.

It is important to note that when the last Application of a reservation occurs, ALPS will launch an Application NHC as well as a Reservation NHC. In some cases, simultaneous instances of NHC may occur on the same node. Support was built into NHC to facilitate the coordination of multiple instances of NHC. This support is labeled as 'advanced_features' in the configuration file.

## D. *Node Health Checker and Compute Node Cleanup*

In certain cases, NHC must attempt to perform compute node cleanup (CNCU). Typically ALPS can request CNCU and NHC simply checks that cleanup occurred correctly. However, if an Application NHC sets the node to a non-UP state, ALPS will not schedule jobs on the node, nor will it perform CNCU. It is then up to Reservation NHC to detect that CNCU did not occur and then NHC will request the kernel to perform CNCU.

The following diagrams illustrate how NHC can guarantee CNCU on all nodes in a reservation. Assume that the final application in a reservation has finished on NID1-3. ALPS will then call an Application NHC first, followed by a Reservation NHC. Also assume that NID01 is unhealthy and will fail NHC tests. Because it is unhealthy, in this example NHC is configured to reboot nodes that fail Application NHC and dump nodes that fail Reservation NHC. The state is communicated to nhcdbd. The nhcbd daemon resides on the sdb and is used to coordinate actions between separate instances of NHC.

In Figure 6, application node health check is initiated at the end of a job due to non-zero exit status of the application.

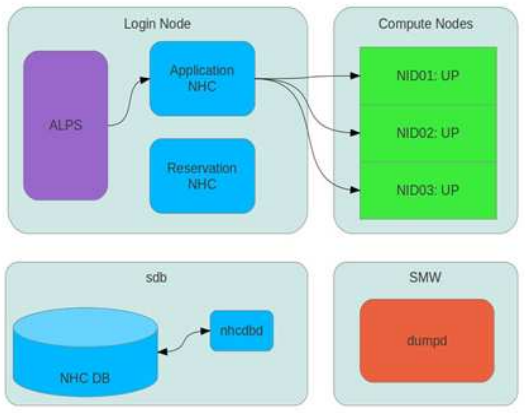

Figure 6. Application NHC Invoked on NID01-NID03

Figure 7 shows Application NHC sets NID1's state to suspect because it failed a health check. Because this node gets set to a non-Up state, ALPS will not request CNCU for this node. NID2-3 pass health checks and have CNCU performed on them via an ALPS call to the kernel. Application NHC stores a reboot request for NID1 in a NHC database for nhcdbd to track.

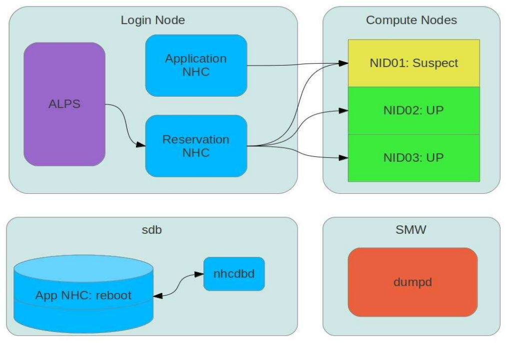

Figure 7. Application NHC Invoked on NID01-NID03

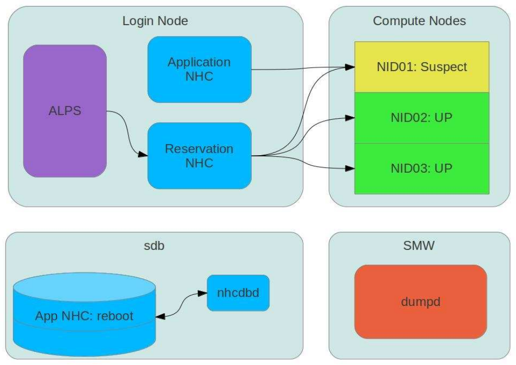

Figure 8. Application NHC Invoked on NID01-NID03

In Figure 8, Reservation NHC is called on NID1-3. It detects that CNCU has not occurred on NID1 and it asks the kernel to cleanup the reservation. NID2-3 pass Reservation NHC. Assume that even though the reservation is cleaned-up on NID1 it still fails a health check and Reservation NHC would like to dump the node.

Figure 9 illustrates Application NHC has marked the node Unavail in preparation to be rebooted. Reservation NHC adds a NHC database entry requesting a dump.

Finally, in Figure 10 nhcdbd reads both entries in the database and sees that no instances of NHC are running on NID1. A single request for a dump and reboot are sent to dumpd on the SMW. The node is then dumped and rebooted.

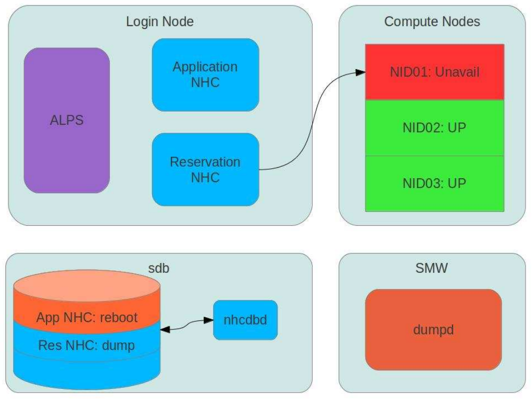

Figure 9. Reservation NHC Marking Node Unavailable

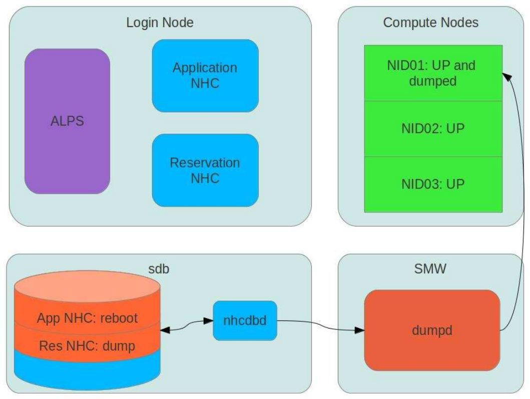

Figure 10. Dump and Reboot of Node

### *E. Resource Utilization Reporting*

Cray is currently in the development phases of a new accounting framework, Resource Resource Utilization Reporting (RUR). The most robust tool available today is Comprehensive System Accounting (CSA). CSA collects and provides job-based data sufficient for per user and per group customer billing. The implementation has several limitations: it is not easily extensible, kernel-based, and it does frequent small file I/O. It is necessary to path the compute kernel image to deliver new functionality with a kernel based solution, requiring system interruption. Frequent small file I/O to Lustre can cause degraded I/O performance, impacting other jobs on the system. Vendors are developing libraries to provide GPU utilization statistics. Utilization metrics are critical in developing efficient codes using new programming models for GPUs. Resource Utilization Reporting addresses this need, by providing a framework and roadmap for system accounting.

#### *1) Goals*

Resource Utilization Reporting provides a scalable framework for data collection. RUR will provide a core set of utilities and a plugin infrastructure for site-extensible customizations. Administrators can add and configure additional plugins on running systems, as well as patch existing functionality without a system reboot. Cray will continue to add functionality for both data collection and the core infrastructure using an iterative roadmap model. Longer term roadmap will include providing multiple data stream formats to facilitate post-processing.

#### 2) *Architecture*

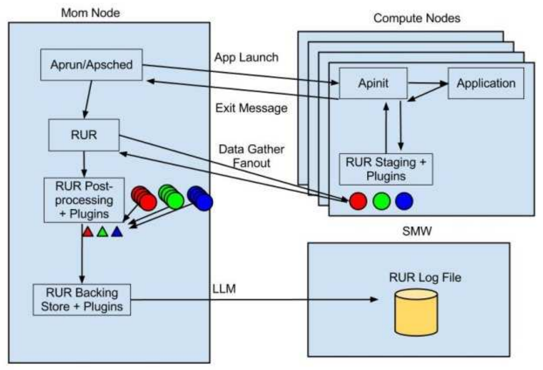

Figure 11 illustrates the overall software architecture. RUR will perform multi-stage data collection. After application run, data is aggregated locally on the compute node. Once the application fully terminates, an agent will pull data off of compute nodes, onto the MOM node. Finally, the agent will post process the data and push it into a backing store such as a syslog stream or specified log file. Post process capabilities will include, but are not limited to:

- providing a single number for each data type such as maximum value
- providing a data histogram (e.g.. mean, standard deviation, maximum outlier)
- presenting all data

### F. *Simple Event Correlation*

Cray is currently working on providing a reference implementation of Simple Event Correlation (SEC). SEC is an administrative tool for correlates events in the domain of log analysis, system monitoring, network management and system security. SEC is an active Sourceforge project, developed by Risto Vaarandi, and is currently widely used by a number of sites. Cray Service will provide installation instructions and a reference set of rules. SEC can be configured on the SMW, where it will watch specific log files, and trigger rules through pattern matching, which then can notify administrators of important systems issues, such as downed nodes. The reference rules support with both LLM and non-LLM based logs formats. Cray Service is working on a plan to provide updates to these reference rules for customers.

## III. CONCLUSIONS

Cray is just completing development on infrastructure foundations for flexible, extensible systems management. These foundations provide the benefit both the administrative and user communities. The framework plus plug-in models, sites can easily extend to handle their use cases. These models allow facilitate iterative roadmaps, where features can be delivered as incremental functionality.

Standardization of data format, and centralized data collection allow better failure diagnosis, and improved system monitoring. Improving fault analysis ultimately results in increased ability to resolve customer issues more quickly.