Resource Utilization Reporting

Gathering and evaluating HPC system usage

Andrew Barry
Cray Inc.
Saint Paul, MN, USA
abarry@cray.com

***Abstract*— Many Cray customers want to evaluate how their systems are being used, across a variety of metrics. Neither previous Cray accounting tools, nor commercial server management software allow the collection of all the desirable statistics with minimal performance impact. Resource Utilization Reporting (RUR) is being developed by Cray, to collect statistics on how systems are used. RUR provides a reliable, high-performance framework into which plugins may be inserted, which will collect data about the usage of a particular resource. RUR is configurable, extensible, and lightweight. Cray will supply plugins to support several sets of collected data, which will be useful to a wide array of Cray customers; customers can implement plugins to collect data uniquely interesting to that system. Plugins also support multiple methods to output collected data. Cray is expecting to release RUR in the second half of 2013.**

I. INTRODUCTION

Since the earliest days of time-sharing computers, administrators have wanted to track what users are doing with system resources. For almost thirty years, Cray systems software has provided administrators with tools for tracking what users are doing. Some of these legacy tools are still available on Cray systems, but are poorly suited to current system circumstances and lacking many desirable features. Over the last year, Cray has collected requirements for, and begun implementing a new feature, tentatively called Resource Utilization Reporting (RUR). This feature will be available to customers in the second half of 2013. RUR will be resilient to failures of nodes within the Cray system, scalable to the largest system sizes, will not contribute meaningful operating system noise to compute nodes during application run-time, and will collect an easily expanded set of utilization data about each user's applications.

II. A LOOK AT PREVIOUS CRAY ACCOUNTING SOFTWARE

Over the decades, Cray Research and Cray Inc. have supported several accounting tools. Each has served its purpose at the time it was released, and several are still available on XE and XC systems. Given the types of systems Cray is now selling, and the diversity of data required by administrators, these tools are no longer sufficient to address the needs of Cray customers.

- COS accounting – Anecdotes from former XMP programmers suggest that the COS batch scheduler allowed for the collection of program statistics. Details about this functionality are not readily available.

- BSD accounting – 4.2BSD unix introduced process accounting in 1983. Most unix varients since that time have included this feature, such as Unicos, Irix, and Linux, the last of which forms the basis for Cray's CLE operating system. BSD-style process accounting generates a file containing records about the cpu usage, memory consumption, and the number of network and filesystem transactions of each process. This accounting is tracked per operating system instance, meaning that a modern Cray system will have hundreds to tens of thousands of such files. The Linux implementation of BSD-style process accounting is incomplete, with some fields accurately tracked, and others always reporting zero.

- Comprehensive System Accounting (originally Cray System Accounting) – Unicos 7 introduced CSA, which tracked a wider range of process characteristics than BSD accounting, grouped together processes based on the batch job, where applicable, and allowed automated billing based on user or group system usage. CSA was supported for many years on the single-system operating system of Cray vector computers. It was ported to support microkernel-base MPP systems, and CC-Numa systems, which ran pseudo Single System Image operating systems. With the release of the Cray X2 vector supercomputer, CSA was ported to Linux, and to the multi-node architecture that is similar to Cray's current CLE operating system. The CLE variant of CSA depends on the Lustre filesystem to aggregate data from all of the nodes in the system. While this was plausible on X2 systems, the scale of current Cray systems presents an order of magnitude more compute nodes. While Lustre allows for this number of nodes to do the metadata transactions, and small file-I/O required for this procedure, it does not perform this well, or easily. It is unreasonable to size, purchase, and tune the

Lustre filesystem based on the needs of the accounting software.

- Mazama Application Completion Reporting – Mazama supports a limited set of application statistics and error codes in the ACR database. This functionality puts a severe strain on the boot raid, which hosts the Mazama database, and which is not generally tuned for database performance. The data collected is also quite limited.

- Application Resource Utilization – ARU is a limited release feature of ALPS, which collects application accounting data in apinit, and aggregates the data up the ALPS fan-out tree. The complexity and overhead of ARU is very low, but it lacks the flexibility to easily expand the data it collects, and the means it uses to report these types of data. ARU is meant to be a stopgap solution, while a more robust and extensible tool can be developed.

III. A LOOK AT POTENTIAL REPLACEMENT UTILIZATION TRACKING SOFTWARE

Many server management tools are available as open source or commercially sold software. These tools are primary targeted at managing scale-out server environments. These collections of servers often have a scale similar to large supercomputers, but are designed to service millions of independent transactions, twenty-four hours a day. While these tools track resource usage over time, they do not track usage per a user's batch job. Time-based accounting at high frequency, can be correlated with a batch reservation record to produce a close approximation of application-level resource tracking. However, high-frequency use of a time-based tool can introduce operating system noise to compute nodes at the least opportune time. Since these server management tools are written for network servers, much of the monitored values are related to TCP networking, which is a small part of what Cray systems typically do.

- Nagios/Shinken/Icinga – This family of related Server management software packages offer a wide array of system management statistics, focused on TCP networks, but offering information relevant to HPC systems. Scaling to Cray-sized systems requires a hierarchy of dedicated management nodes.

- Ganglia – Ganglia is a monitoring tool written for HPC systems. It can be made to scale to Cray system sizes, but not with the sample frequency required to simulate application-based accounting.

- Pandora FMS, OpenNMS, NetXMS – These tools are written primarily to monitor network performance, with some secondary features for system management.

Many Cray customers are interested in time-based system monitoring, and some are using tools of this sort. These tools provide a range of statistics, but do not provide tracking of all system statistics desired by administrators. Moreover, they do not scale to handle the large number of nodes, and large number of samples per-node required to simulate application-based utilization tracking, and introduce noise to running applications. It is clear how a tool which knows the start and end time of an application, can sample intelligently – providing accurate statistics without constantly perturbing the running application, and with minimal overhead in running the monitoring software. Application-based tracking software provides capability uniquely useful to HPC systems. This functionality doesn't map well to the ways most scale-out server environments function; thus it is not surprising that most server management tools are not written to serve this need.

Several varieties of batch scheduling software support the collection of statistics from batch jobs; this is supported on SLURM, PBS Pro, LSF, and Moab/Torque. However, providing this set of data requires running daemons on compute nodes, which is not done on Cray systems. While some Cray customers are planning to run batch scheduler daemons on compute nodes, in the future, this will not be the case for all Cray systems. For Cray systems running in the traditional fashion, with batch scheduler daemons on Cray service nodes, and ALPS sitting between these daemons and compute nodes, the batch scheduler daemon will not be able to collect the raw data needed to keep proper accounting. Furthermore, it is desirable that Cray customers all have access to the same configurable, extensible set of utilization data, independent of which batch scheduling software package they run.

IV. A FRESH START WITH RESOURCE UTILIZATION REPORTING

In 2012, Cray investigated the possibility of extending an existing accounting tool, adopting and extending a third party tool, or writing a new tool. Third party tools, both open source and commercial, are largely written to solve a different problem from those required for application-scale accounting. Cray's most robust existing tool, CSA, requires a new data aggregation method, uses a binary data format with no built-in extensibility, and requires a patched operating system to be run on tracked nodes. Updating CSA would not save significant effort, nor conserve much existing usability or testing infrastructure. Thus a new tool proved the most prudent choice.

In order to meet the requirements of Cray customers, a new tool has been designed: Resource Utilization Reporting. After an application has finished running, RUR collects data elements off of each compute node used by the application, and pulls that data back to the login node from which the application was launched. The login node then reduces the data to a condensed format, and stores this condensed data to a log file or other backing store. RUR differs from prior resource tracking tools by collecting a diverse and expandable mix of data elements, and presenting usage statistics in a highly configurable way. Because RUR runs before and after an application, it introduces virtually no performance noise into the running application.

V. THE RUR PLUGIN ARCHITECTURE

Resource Utilization Reporting is design to allow plugins for collecting arbitrary elements of resource usage data. Past accounting software has not well anticipated the needs of today; there is no reason to believe that one can any better anticipate the needs of tomorrow. RUR is design to allow future requirements for data collection to be quickly implemented, and easily integrated with existing collected data. This is not only intended to allow Cray to more quickly add functionality to the feature, but also to allow Cray customers to generate their own plugins, and integrate those plugins within the RUR framework.

Resource Utilization Reporting comprises four stages of operation, three of which support plugins. Figure 2 shows the components needed for RUR operation.

1. Data Staging – This stage of RUR prepares data on compute nodes, and may include two sub-stages. The pre-application sub-stage collects compute node state information prior to running an application. The post-application sub-stage collects compute node state after an application is run, compares it with pre-application state, where relevant, and writes the data to a staging file on the compute node. The staging file is written to a directory specified in the RUR configuration file. Each RUR plugin must have a data staging plugin; some plugins will not require a pre-application sub-stage, depending on the data collected.

2. Data Collection – RUR uses a scalable, resilient fan-out tree to collect the staged data files from all compute nodes running an application. The collected data is stored on a login or MOM node, in a temporary working file. The fan-out tree is derived from the tree used by the Cray Node Health Checker. This tree is resilient to node failures, and scales to the largest of Cray systems.

3. Post Processing – The RUR post-processing stage may, depending on the RUR configuration file, condense the data gathered from compute nodes. The most common use-case will reduce the data from all compute nodes into a small set of numbers, to be reported to the administrator. Common operations include a mathematical sum, mean, maximum, or minimum for all values reported by all compute nodes. RUR post processing plugins may be very simple, or arbitrarily complex. Future releases of RUR will include libraries to make the creation of simple post-processing plugins easy. A simple case, though one that is likely to be used infrequently, is for the post-processor stage to do nothing, and to simply enumerate data from all compute nodes.

4. Output – The output stage of RUR provides the reporting mechanism and backing storage for all RUR data. Post-processed data is sent to one or more output plugins based on the RUR configuration file. The basic output plugin is a Lightweight Log Manager log stream. This places RUR data in a human readable log file on the System Management Workstation. Other plugins might write data to a database to drive a report generator, or system visualization tool. Output plugins may also be used to connect RUR data to the accounting functions within the workload manager used to schedule the Cray. Output plugins are optional, and many administrators will opt to simply use the default LLM log stream.

VI. RUR AND PROCESS ACCOUNTING STATISTICS

The most basic use case for Resource Utilization Reporting is to replace legacy accounting tools. While these tools vary slightly in the statistics they collect, they all generally collect: CPU time, memory use, and the amount of filesystem traffic an application generates. RUR uses the Linux taskstats interface to collect statistics about processes running on the compute node. The pre-application staging plugin phase for process accounting starts a daemon which listens on the taskstats socket. When a process exits, which is in the process container for the ALPS apid of the running application, the kernel sends relevant process information across the taskstats socket. The blocking wait on the socket ensures that the listening daemon creates very little operating system noise. What little noise that is generated, occurs when processes exit, which is typically not associated with the most sensitive regions of an application's runtime. The post-application phase of the plugin stops the taskstats listener, and aggregates the data for all processes on the compute node.

.

```
Apid: 2000, Jobid: 26400, uid: 3417, stime: 120,
utime: 4811, highwater_rss: 52000, highwater_vm:
56512, read_char: 4096, write_char: 8192
```

Figure 1. Process Accounting Data

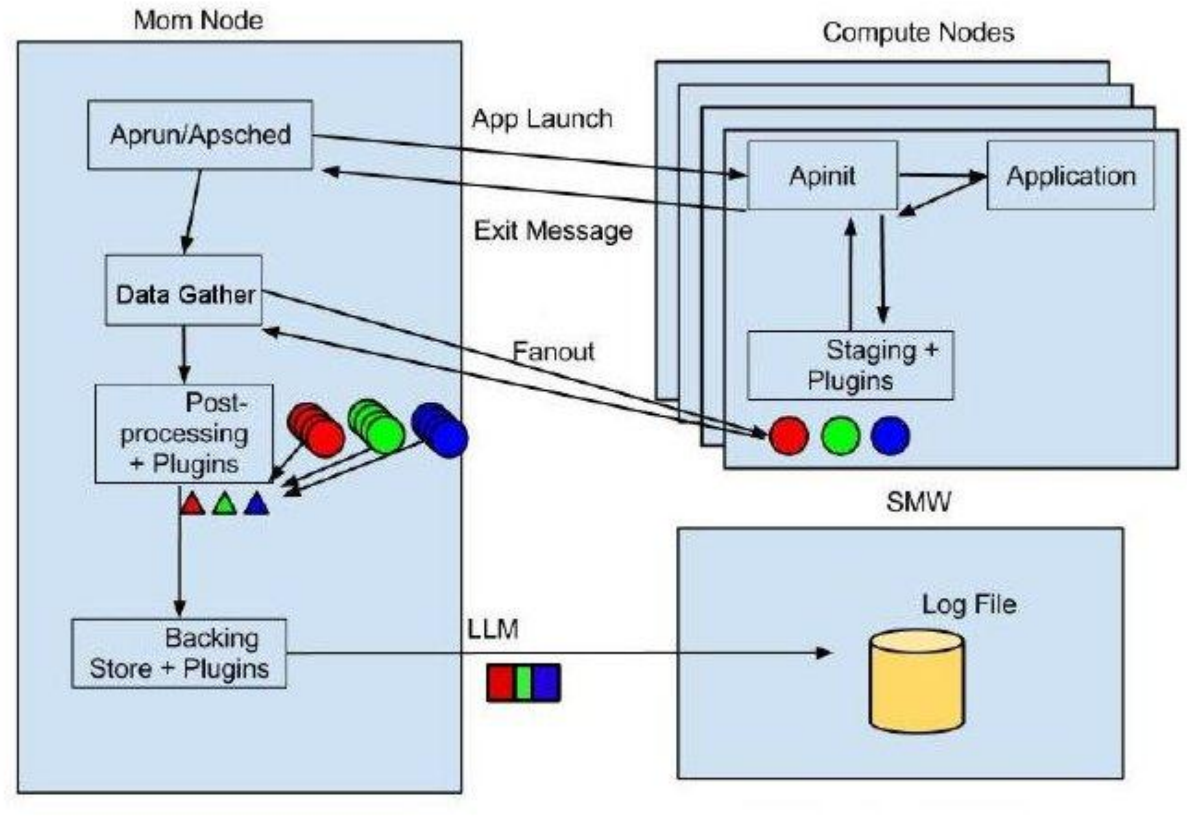

Figure 2. Diagram of RUR Operation

Once the process accounting data is staged on the compute nodes, the RUR data gathering stage collects these staged files from compute nodes, combining them into a single working file on the login node. This file is then provided to the process accounting post-processing plugin. This plugin, at the option of the RUR configuration file, reduces the N records from N compute nodes, to a single record, including the sum of the processor times, the maximum of the high-water levels, and the maximum of the bytes read and written. This single, combined record is passed to the RUR output stage, which, by default, records it as a human-readable record in the LLM log stream for RUR. This record can be inspected on the SMW by the administrator

VII. RUR AND GPU ACCOUNTING

Customers running Cray systems with GPU accelerator nodes may wish to identify which users make use of the GPU, or simply use the CPU on the compute nodes. It may also be interesting to see to what extent users tax the GPU. To this end, RUR will support a plugin to gather GPU statistics. The pre-application phase of this plugin calls the CLE GPU-Accounting utility to clear its buffer of GPU statistics. Once the application has finished, the post-application phase of the GPU plugin again calls the CLE GPU-Accounting utility, which queries the GPU driver for accounting data. This data is then written to the compute node staging file. This requires no run-time daemon to track accounting data, as that functionality is provided by the GPU driver.

```
Apid: 2010, Jobid: 26410, uid: 3417, GPU-time: 3100 GPU-utilization: 5100, highwater-memory: 58
```

Figure 3. GPU Accounting Data

Once the GPU accounting data is staged on the compute nodes, the RUR data gathering stage collects these staged files from compute nodes, combining them into a single working file on the login node. The GPU post-processing plugin sums the processing time, and computes the maximum of the memory high-water measures. In the default configuration, this combined record is then written to the LLM log stream for RUR. If the RUR configuration file is configured to track both process accounting and GPU accounting, for example, the GPU accounting data would appear on separate lines, though both will contain the same apid and jobid entries.

VIII. RUR AND POWER ACCOUNTING

System total power usage is increasingly becoming a limit on HPC system purchases, and power usage is a major cost for Cray customers. Tracking who is using that power, and how it is being used, is becoming increasingly interesting. RUR supports a plugin to gather power usage statistics for applications. Similar to the GPU plugin, the pre-application plugin phase for power accounting resets a counter in a CLE utility for tracking power usage. The post-application phase of the plugin again calls the CLE utility, and writes the node's power-usage metric to the staging file.

```
Apid: 2019, Jobid: 26416, uid: 3417, Joules used: 34028, Power Cap: 310, Changed: no, Consistent: yes
```

Figure 4. Power Accounting Data

Once the data is staged, the RUR data gathering stage collects the staged files from the compute nodes. The power accounting post-processing plugin sums the used power for all compute nodes used by the application, and the combined record is then written to the default LLM log stream for RUR, or other output plugin, according to the RUR configuration file. This data may be used to study the power efficiency of various users, applications, compilers of libraries. It may even be used to influence billing.

IX. POSSIBLE FUTURE RUR PLUGINS

Many CLE components are capable of providing performance data or availability statistics. Some of these metrics might be a simple flag indicating normal operation, or a problem state. While severe errors are generally reported by way of the console log, and may provoke Node Health Checker actions, minor errors and general performance data are often not reported in the console log. The data which is reported, has a timestamp, but is not automatically indexed by the application running on the node. Such indexing, and greater volume of performance data can be provided by RUR plugins.

Lustre filesystem components, running on the compute nodes, may collect metrics on delays in processing metadata transactions, or in sending data to storage targets. DVS components may report delays. CLE modifications may make it possible to report filesystem statistics for each mount point. All of these proposals would provide a greater volume and granularity of statistics about file system behavior. Most filesystem issues are likely to be system-wide, rather than isolated to the nodes running an application; Thus these statistics are generally more suited to console error logs, or time-scale server management tools. However, some users or applications may uniquely make extensive use of a filesystem, which may be best tracked with a RUR plugin.

The Cray Aries interconnect chip contains hundreds of performance counters relevant to network behavior. A future RUR plugin may be implemented to track relevant network performance counters for each application. The shear volume of available counters, and the complex meaning contained within each, may make interpreting the collected values a complex task.

Future RUR plugins may also be created to service the needs of future coprocessors, as they become available in Cray systems. The initial RUR release will support the Nvidia Graphics Processors sold in the XK7 system. Cray has announced that it will support accelerator products from Nvidia and Intel in XC supercomputers, and other accelerators may be supported. If useful statistics can be gotten from these devices, a RUR plugin could be implemented to compile the data for each application.

An output plugin may be developed to report application utilization statistics to the user running the application. This would provide users immediate feedback on how the application behaved. Other output plugins might be developed to insert utilization data into a database, and to report trends in utilization to the system administrator, or directly to users. An interface plugin may also be made, which would translate RUR output into a format understood by workload-manager software.

X. SITE CUSTOM RUR PLUGINS

Cray customers may wish to implement site-custom plugins. Several factors may motivate this decision. RUR plugins may be needed to investigate a bug, or performance optimizations that are transient, and may have a very brief lifetime. Plugins may also be desired to measure a hardware, filesystem, or software feature that is unique to a particular site. Customers may wish to interface RUR with an existing database, or system visualization tool.

Given the desirability of allowing Cray customers to write their own RUR plugins, RUR is structured to make this simple. Simple data collection plugins will have to support a compute node staging component, and a post-processing component. Most RUR plugins will not support custom logging or data storage components, though this is possible. RUR will include python libraries for easily doing the most common form of post-processing operations, such as sums, min, max, mean and simple histograms of data elements. RUR plugins do not need to be written in python.

As an example, one can imagine a Cray customer who makes the “widget” software package available to all compute nodes on a DVS filesystem. The admin of this system wants to track the total number of widgets invoked, and the maximum time a compute node spends running the widget software. A wrapper script causes the widget software to write debug information to a file on the compute node. When the application is finished, the RUR post-application plugins are run. The widget plugin reads up the debug file, written by the widget software, and writes the number of widgets invoked, and time the compute node spent running widget software, to the widget staging file, and deletes the debug file. The RUR data gather stage collects the widget staging file. The RUR post-processing stage invokes the widget post-processor. The widget post processor uses the include sum function to find the sum of the all widgets invoked, across all compute nodes, and the include max function to find the maximum time, across all compute nodes, spent running widget software. This is then written to the RUR log stream by the default RUR output module.

XI. SUMMARY

Resource Utilization Reporting is an attempt to address the shortcomings of previous resource tracking tools for Cray systems. A survey of third party solutions shows that server management software packages and work load managers provide a limited set of statistics, and do so under operating assumptions in conflict with maximum performance of the Cray system. For real-time system monitoring, existing server management tools provide a great deal of functionality, but they do not address the needs for a scalable, configurable, extensible, and lightweight utilization tracking tool.

The RUR design is a major improvement on past tools, due in large part to the plugin architecture; this design allows the collection of a diverse and evolving set of statistics, using a single configuration and data collection engine, derived from mature technologies. RUR is highly configurable, with the ability to change the data collected, the sort of post-processing done, and the output format and location. RUR has very little performance impact on the running application, and scales to the largest of Cray systems. RUR will be available to Cray customers in the second half of 2013.

ACKNOWLEDGMENT

Dave Henseler, Tara Fly, John Hesterberg and Dean Roe have contributed greatly to the technical design of the RUR feature. I thank them for their efforts.