SANDIA REPORT

SAND2011-6301
Unlimited Release
Printed Sept 2011

Develop feedback system for intelligent dynamic resource allocation to improve application performance

Jim Brandt, Ann Gentile, David Thompson, Tom Tucker

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

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: <http://www.osti.gov/bridge>

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: <http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online>

Develop feedback system for intelligent dynamic resource allocation to improve application performance

Jim Brandt, Ann Gentile, David Thompson

Sandia National Laboratories

M.S. 9159, P.O. Box 969

Livermore, CA 94551, U.S.A.

`brandt@sandia.gov`, `gentile@sandia.gov`, `dcthomp@sandia.gov`

Tom Tucker

Open Grid Computing, Inc.

4030 West Braker Lane STE130

Austin, Texas 78759

`tom@opengridcomputing.com`

Abstract

This report provides documentation for the completion of the Sandia Level II milestone "Develop feedback system for intelligent dynamic resource allocation to improve application performance". This milestone demonstrates the use of a scalable data collection analysis and feedback system that enables insight into how an application is utilizing the hardware resources of a high performance computing (HPC) platform in a lightweight fashion. Further we demonstrate utilizing the same mechanisms used for transporting data for remote analysis and visualization to provide low latency run-time feedback to applications. The ultimate goal of this body of work is performance optimization in the face of the ever increasing size and complexity of HPC systems.

Acknowledgment

The authors would like to thank:

- John Zepper (SNL), John Noe (SNL), and Jeff Ogden (SNL), Sue Kelly (SNL), and Robert (Bob) Ballance (SNL) for providing root access, time, and support on SNL's Cielo Del Sur (key in enabling us to develop on a platform at our target scale) and on Muzia, both Cray XE6 systems
- Jim Lujan (LANL), Cory Lueninghoener (LANL), Kathleen (Kaki) Kelley (LANL), Cindy Martin (LANL), and Quellyn Snead (LANL) who were all instrumental in our Cielo deployment beginning with small scale testing on their development platforms and culminating in several dedicated application times at our target scale
- Jerry Friesen (SNL), and Adam Supinger (SNL) for providing root access to and allowing us to install hardware and software on SNL's Whitney TLCC cluster for data collection and analysis
- Tuesday Armijo (SNL) for working with SNL's cyber security personnel to accelerate our software approval for deployment in Whitney's security environment
- Karen Devine (SNL)[Zoltan]; Pat Notz (SNL), Greg Wagner (SNL), Stefan Domino (SNL), Alex Brown (SNL), and Robert Baca (SNL) [SIERRA/Aria and SIERRA/Fuego]; Elijah Newren (SNL) and Mike Kurtzer (SNL) [SIERRA platforms]; Jonathan Hu [SIERRA-Trilinos integration]; Nicholas Gentile (LLNL) [MCB]; and Mahesh Rajan (SNL) [cth] for their support and insight into the applications used in the preparation and execution of this milestone
- Joel Stevenson (SNL) for his help and support for running applications on Cielo and Muzia
- Larry Kaplan (Cray) and Jason Schildt (Cray) for providing needed XE6 platform specific information

Finally we would like to thank Sudip Dosanjh (SNL) and Ron Brightwell (SNL) for their funding support on the projects that jointly enabled this milestone.

Contents

1	Executive Summary	9
1.1	Description	9
1.2	Motivation	9
2	Motivation	11
3	Architecture	12
3.1	Design	12
3.1.1	Design Criteria	13
3.2	Implementation	13
3.2.1	Cielo Del Sur	14
3.2.2	Cielo	14
3.2.3	Node-local Data Collection	15
3.2.4	Off-node Transport	16
3.2.5	Remote Storage	18
3.2.6	Analysis and Visualization	19
3.2.7	Node-Local Feedback to Application	26
4	Criteria-Related Results	30
4.1	ASC Relevant Platform	30
4.2	Data Acquisition	30
4.3	Data Analysis and Visualization	34
4.4	Affect Run-time Work to Resource Mapping	39
4.4.1	SIERRA/Fuego	40
4.4.2	SIERRA/Aria	42
5	Conclusion	45
	References	46
	Appendix: Signoff Memo	47

1 Executive Summary

This report provides documentation for the completion of the Sandia Level II milestone "Develop feedback system for intelligent dynamic resource allocation to improve application performance"

1.1 Description

Demonstrate the ability to affect application performance on a Cielo-like architecture or similar-scale platform through appropriate resource allocation (both static and dynamic) based on historic, run-time, or user furnished process resource requirements in conjunction with current platform resource usage and availability. Historic and run-time application resource requirement characteristics as well as platform resource utilization will be derived using a scalable information gathering and run-time analysis system. Numeric hardware related metrics and information acquired from the platform scheduler/resource manager will be used.

Expected deliverable: Demonstration of high-fidelity monitoring with distributed data collection, run-time analysis, response trigger generation, and response at large scale ($> 10,000$ computational units).

1.2 Motivation

Due to the size and complexity of modern supercomputers it is difficult for application writers and users as well as platform system administrators to gain insight into how the resources of individual platform components (socket, core, memory, network, etc.) are being utilized by application and system processes. As part of Sandia's resilience effort, which seeks to gain low-level understanding of resource characteristics for prompt failure diagnosis and prediction, we have developed lightweight and scalable data collection, analysis, and visualization tools. As part of a research project at Sandia aimed at improving application resource utilization through system feedback to applications, we have also been working on using resource utilization information to help guide intelligent applications in their selection of resource to load binding decisions. This milestone combines tools from both areas of research and development to enable both large-scale resource utilization information collection for post run analysis and run-time use of this information for guiding load balancing operations.

Based on the text and intent of the milestone we define the following to be the minimum success criteria for this milestone:

1. The platform from which resource information will be acquired and on which applications will be run will be an ASC relevant architecture.
2. Acquisition and storage of > 20 metrics per processing element over $> 10K$ processing elements at a collection frequency of 6 per minute.

3. Analysis of information pertaining to resource utilization for a particular application run or set of runs spanning 10K processing elements.
 - (a) Post-run analysis on aggregate archived information
 - (b) Run-time analysis and visualization on streaming data
4. Use of a subset of metric values being collected to affect run-time work to resource mapping in an application run and/or ensemble of application runs.
 - (a) Affect of feedback on application performance will be evidenced through a change in load to resource mapping, application run-time or both as compared with the application running concurrent with data acquisition but with no feedback.
 - (b) At least one application run will span $> 10K$ processing elements.

2 Motivation

Performance of an application on a particular platform depends not only on the speed and capabilities of the hardware and system software but also on how the application utilizes those resources. Though it is the job of the operating system scheduler to place processes efficiently, it only has insight into what is going on local to a node. Thus the burden of efficiently allocating work across all of the nodes associated with a particular application falls to the application software and user. Further, for performance reasons, the typical practice is to bind processes to a particular core within a node leaving balance within a node to the application/user. Thus tuning an application for performance requires some level of insight into how it will utilize the underlying compute resources both at the system level (e.g. nodes in network topology, storage) and at the node level (e.g. cpu, memory, cache, shared communication bus, network subsystem). Due to the size and complexity of modern supercomputers it is difficult to gain insight into how these resources are being utilized by the application processes. Profiling tools such as OProfile [12], Tuning and Analysis Utilities (TAU) [11] and CrayPat [1], to name a few, allow a user to profile their applications but can incur significant overhead which can impact the behavioral profile of the application. Also many of these tools require building instrumented code or relinking against other libraries. With complex codes the tools for automatically instrumenting an application may fail leaving the user to instrument by hand. As part of Sandia's resilience effort, which seeks to gain low-level understanding of resource behavioral characteristics for the purpose of failure modeling, we have been developing lightweight and scalable data collection, analysis, and visualization tools targeted at automated decision making based on relevant component level (node, core, network, etc.) information. As part of a research project at Sandia we have also been targeting use of resource utilization information to help guide intelligent applications in their selection of resource to load binding decisions.

The applicability of and need for this direction of development is called out in an NNSA Workshop on Exascale Computing Technologies tools working group white paper [14] This document expresses the need for "scalable collection and analysis of performance data", the "ability to feed analysis results back to the application and/or system software", and for "analysis results to be used by applications for run-time optimization".

This milestone combines tools from both areas of research and development to enable both large-scale resource utilization information collection for post run analysis and run-time use of this information for guiding load balancing operations. The main goal of this milestone is to demonstrate the viability of large scale collection and use of this type of information to inform application load balancing decisions through both post-run analysis and run-time node/process local analysis.

3 Architecture

This section describes the architecture of our system for collection of information on the compute nodes of a HPC platform, transport of that information to remote hosts for analysis as well as acquisition of the information by applications for use in load balancing decisions.

3.1 Design

Figure 1. Main functional components of system

The five main functional components in our design are illustrated in Figure 1.

Node local data collection mechanisms acquire data from a node and store it in local user space memory. Off-node transport of data to remote hosts is accomplished via a flexible socket or RDMA based mechanism. Remote storage to either database or flat file is supported. Analysis* and visualization are supported in both streaming and post processing mode Node-local feedback to application is accomplished via the same interface as used for off-node transport

Each of these components is described in more detail Section 3.2

*Outlier detection

3.1.1 Design Criteria

The following guidelines and criteria were used in the execution of this milestone and related work:

- Provide an architecture to gain insight into resource utilization that is separate from the application but can be utilized by the application without significant application change
- No change required in the application (e.g. re-link to additional libraries) in order for the user to get utilization information
- The data collection and transport must not significantly impact the performance of the application
- Lower bound of 1 sec data collection intervals on compute nodes to match the minimum the time stamp resolution of MySQL.
- Upper bound of 10 sec data collection intervals in order to enable post processing analysis of some level of dynamic use of resource in applications with run-times in minutes (6 data points per variable per minute).
- Minimal state retention on the node limited to the current set of data and meta-data
- Transport must support multiple host hops to enable use of collection to remote hosts given architectural and security constraints of our target platform, Cielo
- Support long term retention of data for long term analysis and multi-run comparison, and the ability to provide data sets to application users
- Streaming visualization on a single data variable at the storage data rate required for:
 - Interactive user feedback to application
 - Provide a qualitative understanding at live rates to guide more detailed post-run analysis
- Node-local feedback only due to security constraints

3.2 Implementation

In this work we deployed our monitoring, analysis, and feedback system on two Cray XE6 platforms: Cielo at LANL, and Cielo Del Sur at SNL. These compute platforms were the targets for gathering resource utilization data while running applications and using that information to give feedback to these applications via initial scheduling decisions, static weighting of resource capability, or run-time feedback. Additionally we required infrastructure for transporting information off platform, storing it for post-run analysis, and performing post-run analysis. Diagrams of each whole system deployment are shown for Cielo in Figure 12 and for Cielo Del Sur in Figure 13. In this section we discuss the characteristics, challenges, and solutions for each of these deployments.

3.2.1 Cielo Del Sur

Cielo Del Sur is a six cabinet Cray XE6 with 21 service nodes and 556 compute nodes (8880 processors). Given our goal of demonstration at greater than 10K cores this is a reasonably compatible system. Since there is no connection from either the compute nodes or boot node, which has direct connection to the compute nodes, to the external network we had to take multiple hops as shown in Figure 13. Further, since we could not initiate a socket connection from an external host to cdssmw we had to run ldmsd 3.2.4 in bridge mode on this host and passive on each of the storage and analysis hosts. For our storage and analysis hosts we used four desktop Shuttle XPC PCs each with a single Intel Core i7 3.3 GHz processor and 6GB of memory and running Fedora 14 and a 2.6.35 kernel. We stored information to two MySQL databases per storage host. For database storage we used a single Crucial M4-CT256M4SSD2 solid state drive (SSD) per host which provided almost a factor of ten decreased insert times for our volume of traffic over using the mechanical disk used for general user and system software. Thus our final configuration used 8 databases (2 per SSD) across our four storage hosts. With this configuration we were able to ingest 1012 samples per node per 5 second interval over all compute nodes.

The XE6 platform, like most other production HPC platforms, only allows a user to schedule a single job on a node at a time. Thus in order to run our monitoring software concurrently with an application we run it as a root process from the platform boot node prior to running an application to be monitored. Additionally we have a vmstat kernel module and a supporting ldms kernel module that, in order to be used, must be installed by root. Launching or removing all components and checking for their status over all CDS compute nodes in parallel takes on order of 60 sec making it feasible as an automated launch-time option.

3.2.2 Cielo

Cielo is a 96 cabinet Cray XE6 with 9K compute nodes and 145K cores. To reach our goal of 10K processors we used 7 racks with 646 compute nodes and 10,336 processors. To meet our goal of distributed data collection on the platform we utilized two visualization nodes that had both direct connectivity to the compute nodes and a direct connection to the tri-lab DISCOM network (see Figure 12). We utilized a DISCOM connected node of SNLs Whitney cluster as the next hop with the end points being a diskfull admin node and a diskless compute node. We populated each of these nodes with two Crucial M4-CT256M4SSD2 SSDs for storing the data from Cielo. The Whitney nodes used each have four AMD Barcelona quad core 2.2 GHz processors and 32GB of memory. The admin and compute node were both running Red Hat Enterprise Linux 5 with a 2.6.18 kernel. Due to the poor database performance of these machines we wrote the data initially to flat files and subsequently bulk loaded it into a database on the admin node for analysis. We collected 1012 data values per node at a sample interval of 9 seconds to meet our criteria of 6 data sets per minute per monitored node.

3.2.3 Node-local Data Collection

Our lightweight distributed metric service (LDMS) infrastructure was designed to meet the need for both low impact data collection and movement on a compute node and for low overhead transport off a compute node for storage and/or analysis.

LDMS allows for zero CPU overhead when gathering data from compute nodes through the RDMA transport, however, there is still overhead associated with gathering the data at the sampled node. Unfortunately data of interest is kept in many different places on a Linux system. Some data is maintained by user-mode programs, for example, Zoltan, or MPI rank and job data, other data is kept in the kernel, for example scheduler and VM data, and still other data is kept by hardware itself such as hardware performance counters.

This presents a number of challenges; to first order, the data has to be gathered into a Metric Set where it can be fetched by the sampling node. For user-mode data, this can be trivially accomplished by either keeping the data natively as a LDMS Metric Set, or by gathering the data of interest and writing it to an LDMS Metric Set. For kernel data, there is the problem of accessing the data itself. Some data is accessible trivially by kernel modules because the containing data structure is exported, e.g. vmstat data. Other data, however, is 'hidden' by virtue of being declared "static" in C and can only be accessed through a user-mode /proc filesystem interface; e.g. schedstat. Finally Hardware performance counter data can be accessed with the kernel perfctr API. In all cases, however, there is the problem of introducing scheduler jitter when sampling. For kernel data gathering, this can be avoided by using a kernel work-queue and only gathering data during I/O wait or otherwise idle cycles. In user-mode, this can be accomplished by scheduling the sampler thread with very low priority. In both cases, however, the data gathering time should be minimized in order to avoid delaying an I/O wake up that would awaken a compute process.

An optimally low-jitter sampler would:

- Be run on a kernel work-queue thread
- Be run in less than 100ns
- Gather data directly from memory or hardware performance counters
- Avoid filesystem interfaces such as /proc

In order to achieve this in practice, however, requires that the stock kernel be modified in order to access metrics in hidden data structures such as schedstat and the LDMS kernel modules be installed that provide a means for LDMS to map kernel resident data into user mode and have this memory registered for RDMA base transports such as Infiniband, iWARP or GNI.

3.2.4 Off-node Transport

In developing an appropriate transport mechanism we had to take into consideration the following: 1) data sources may have data available at different times and may collect on different time scales, 2) data would have to be relayed through intermediate hosts which may have asymmetric network access privileges between them and the next/previous hop host, 3) latency between initiating a data request and receiving the information, 4) we would like it to incur minimal impact/overhead on the monitored compute nodes, and 5) we need to be able to parallelize this process for scalability reasons. To accommodate for these constraints we developed a transport which is included in our lightweight distributed metric service (LDMS) suite of tools called `ldmsd`.

Because of the asynchronous nature of the data gathering across compute nodes we decided that data movement would be done autonomously and asynchronously by each `ldmsd` on a hop by hop basis from the source to the storage host. A generation number is published with each data set and is checked before insertion into storage to ensure that old data is not stored repeatedly. Further, data sets are comprised of raw values and deltas are taken at the time of insert into long term storage so that if a set is missed due to the asynchronous nature of the collection it only results in loss of fidelity not data. Data sets are generated on a set by set basis on the compute node asynchronously and on a time interval set by the user. When a next hop host `ldmsd` makes a request for data, the `ldmsd` on the compute node sets up a connection and sends both the data set and variable names (meta-data) and data to that host. After the initial connection and meta-data transfer only data is sent on subsequent requests unless the meta-data generation number changes. This next-hop `ldmsd` stores the information locally and queries the compute node for more data according to a user specified time interval which need not have any relationship with the actual collection frequency. This same behavior happens on a hop by hop basis clear to and including the storage host. Finally on the storage host another entity, called `komondor`, fetches information on its own time scale from the `ldmsd` on the storage host. The reason the collection frequency is allowed to vary across `ldmsd` entities is that this leaves it open for a user to set the node based collection frequency based on how often their application may require feedback while letting the long term storage system collect at a rate commensurate with the collection system and fidelity of information they want to store and analyze. Latency for a data fetch is dependent on the data volume, network (bandwidth and latency), and host load.

In traversing multiple hosts from the compute nodes to the final storage hosts in our deployments there were hosts that for security reasons were not allowed to initiate connections with the next-hop host. In order to get around this impediment we developed three different modes of operation for `ldmsd` (active, bridge, passive). An active `ldmsd` can be connected to directly as described above. For the asymmetric case, the host that is allowed to initiate connections runs `ldmsd` in bridge mode and polls for the target `ldmsd` running in passive mode. Once the connection is established the passive `ldmsd` behaves as described above and polls the bridge `ldmsd` periodically for new data. The directions of connection, data request, and data flow are shown in Figure 2.

As is described in Section 3.2.5, the performance bottleneck in this system is on final processing of data by the `komondor` for insert into a database. Though data is collected and transmitted serially by a `ldmsd` as a single data stream from which multiple `komondor` processes can read subsets of the

Figure 2. Directions of connection, data request, and data flow in the ldms chain. Multiple connection operation modes (Bridge, Active, Passive) support constraints on connection initiation.

information there is time involved in serially bundling and transporting this information which can also be affected by how busy the systems processing the data are. Thus we used a single komondor per data stream and sized the stream (in terms of number of data sets) to yield the database write performance we were targeting.

To demonstrate the scalability of utilizing multiple ldmsds in parallel over different paths we ran multiple ldmsd entities on separate first hop hosts (see Figure 12), and then on a common second hop host. We determined the number of compute node data sets per komondor by increasing the number until the database insert rate per component started decreasing and then backing off. This is of course dependent on the frequency of inserts and the amount of data being collected per compute node. While data transport CPU related overhead could be minimized by utilizing the Gemini RDMA transport, due to the proprietary nature of this interconnect we used Cray's socket based transport over the Gemini network. We were not able to evaluate the relative impact as we did not achieve an operational Gemini RDMA based solution during the course of this work.

The network impact of data flowing to one of the four storage hosts shown in Figure 13 can be seen in Figure 3. The traffic on the inbound interface of each storage host was captured at one second intervals though the ldmsd updates were at four second intervals. The reason for the various levels is that we were running six ldmsds per storage host and they were started with a small time delay between to spread out the traffic. The aggregate traffic going to a storage host in this case is 2MB per second or 4Mb per second. Thus the aggregate network traffic for collecting 1012 values for each of the 556 compute nodes on CDS is 16Mb per second. As shown in Figure 13 this data was streamed over the wide area network from SNL/NM to SNL/CA.

Figure 3. Network traffic to one storage node over 1000 sec with measurements taken at 1 sec intervals

3.2.5 Remote Storage

Storage of data occurs on the remote hosts as indicated in Figure 1. Motivation for storing data includes the desire to enable post-run and long-term analysis and multi-run comparison. An additional desirable feature is the potential to hand off per-application datasets to the application user post-run.

Constraints on the handling of data at endpoints then include 1) the ability to ingest data at the desired collection rate 2) its representation in some form that supports analysis and visualization and 3) the ability to transform the data into the desired form(s) on necessary time-scales.

Data is read at the endpoint via an ldmsd entity running on the remote host. Another entity, called the komondor, reads LDMS data from the local ldmsd and is responsible for its transformation into an intermediate or final stage. A database is a desirable option for representation of data while it is being used for analysis and visualization of long term data, particularly where a large number of components or many data metrics are possible. Databases additionally support the ability to write out to files or to load in data files, thus supporting storing and re-accessing data. However, the insert performance of the database is a potential bottleneck in supporting data ingestion rates. For this reason, we have written the komondor to optionally 1) write the data directly into a database on a per-component per-metric basis as it is received; 2) submit the data to the shepherd [13, 3], which can both process the data as it is streamed and aggregate the data on a per-metric basis for efficient bulk insertion into a database; or 3) write the data to files which can then be periodically loaded into the database. The first method can potentially reduce time skew (subject to the insert rate supported by the database); the latter two methods support greater ingestion rates at the host but are subject to a delay before a timestamp is added to the measurement. In addition, multiple komondor processes can read all or subsets of the data from a single ldmsd enabling balancing of

the data flow with the database insert performance. Finally, the architecture also supports the user of multiple, non-replicated databases for analysis and visualization, in order to support the data load. Options for the komondor are shown in Figure 4. In this work we utilized SSD's for database and flat file storage as our rotating media had unacceptable performance characteristics.

Figure 4. Handling of the data stream on the remote endpoints. Komondor read data from local ldmsd. They can optionally provide measurements to 1) the shepherd, where recent values are cached and logged for bulk inserts into the database; 2) the database directly; or 3) flat files. Multiple komondor can read from the same ldmsd. The multiple options enable the data storage rate to be commensurate with the data ingestion rate on the node.

Note that as in other parts of the transport system both the local ldmsd and the komondor can set their query rates independently of all other parts of the ldmsd chain. Thus, if desired, data can be collected more frequently at the nodes but transported off the the nodes and/or collected by the komondor less frequently if higher fidelity information is required on node than is desired to be stored. Additional processing steps can occur before storage in case only a subset of the data is wanted for long term storage.

3.2.6 Analysis and Visualization

In order to perform analysis and visualization of the measurements provided by LDMS, the raw data and results of analysis must be aggregated from distributed databases. The next generation of OVIS's [13, 3] baron – a tool for inspecting system metrics – was designed so that the subsets of data required for presentation are small in order to provide interactivity and scalability.

In addition to the demands placed on visualization by the parallel nature of the data, informal input from stakeholders in the development and administration of HPC platforms was used to define common tasks that the baron must address. These are presented in Table 1. Beyond common tasks, user input also suggested several desirable features for visualization and analysis. Because

Table 1. Use cases for analysis and visualization.

Describe the system

Show or enumerate the components that compose the system.

Illustrate the layout of components.

Characterize the system behavior

List the measurements made on each type of component in the system.

Present the range of values taken on by a metric over the entire system.

Identify the set of components associated with a particular application.

Illustrate trends over time in measurements.

Discover simultaneity of events.

Diagnose problems and faults

Determine which components share access to resources (e.g., network, cache, power).

Compute statistics of a set of metrics over a set of components and range of time.

Identify which components take on extreme values of a metric or statistic.

Compare metric measurements taken on different components.

Compare measurements of different metrics taken on the same component.

many administrators manage HPC systems remotely via thin clients, renderings that require fast graphics cards should be avoided in favor of simpler illustrations. Also, while the ability to examine detailed information is important, having a summary view of the entire HPC system is also needed to quickly alert administrators when a problem arises. Finally, users submitting jobs to run on the HPC system should also have access to metric data taken during the time of their run.

From this information, we decided on an architecture that uses an embedded web server – Mongoose [9] – as the primary means of communicating with both data collection and visualization/analysis processes. Mongoose and other embeddable web servers use multiple threads to process hundreds to thousands of requests per second, provide access controls and encryption, and transparently handle socket connection management. The user interface is composed in HTML and JavaScript so that web browsers such as FireFox and Safari can be used on thin clients.

Data is sent to the browser in JavaScript Object Notation (JSON) form[4]. Summary metadata for the cluster is present in every shepherd process’s database and includes the number and type of components and metrics. The browser is instructed to retrieve this data from a single shepherd – usually the one from which the HTML and JavaScript code were also obtained. Other data is partitioned among all of the shepherds’ databases: it resides on only one shepherd. The browser retrieves each shepherd’s subset of the data by sending a request to each shepherd directly. Because web browsers have been optimized to perform layout and rendering concurrently as network traffic is received, subsets of data returned by each shepherd are displayed immediately as they become available; any processing required to aggregate per-shepherd data is performed asynchronously.

Figure 5 shows the baron’s summary view which lists available visualizations. The following figures and paragraphs describe each one in turn.

Figure 5. The baron provides a list of views, each of which embodies a different visualization of OVIS data.

The Namespace View, shown in Figure 6, provides the user with a description of all metrics in the database organized by the type of component the measurement is associated with. For instance information from `/proc/meminfo` is provided on a node basis and is hence associated with the "node" component type while `/proc/stat` provides information at both the node and core level which is then split into "node" metrics and "core" metrics.

Figure 7 shows the baron's Graphical View. It draws graphs (laid out using a subgraph that is a tree) to show how components are related. By default only the top-level components in the tree are displayed. When the user enters the name of a particular component, it is highlighted and drawn with all of its siblings, its parents, and its parents' siblings. This can be used to identify, for instance, which cores are present on a particular node.

Typically, when users are exploring measurements, they will identify related components using the Graphical View and then plot time histories of a few metrics on these components. The Time-history View (Figure 8) presents an interactive plot where users can specify a list of metrics and components whose data should be plotted. Panning and zooming interactions are supported and the data is subsetting so that a roughly constant number of samples are drawn per-component per-metric. This subsetting is achieved by assigning a level-of-detail (LOD) to each sample in the database and then only requesting samples with a LOD greater than $k \log_2 \Delta t$, where Δt is the time range of the view and k is a constant related to the desired number of samples per plot series relative to the nominal sampling frequency of the metric.

In addition to plotting time-histories of components, it is sometimes useful to see the entire state of the system at once. Time-series plots do not allow this for more than a few components. However, hundreds of thousands of components can be represented by pixels in an image on modern computer monitors. The Operating State View of Figure 9 shows every component's value of

Name	Units	Datatype	Constancy	Interval [s]	Min. Delta T [s]
LO_T_SLOT_NODE_PROC_CPU_TEMP	degC	int	none	none	1
LO_T_SLOT_NODE_PROC_VRM_TEMP	degC	int	none	none	1
LO_V_FSIO_NODE_PROC_12V	mV	int	none	none	1
LO_V_FSIO_NODE_PROC_VDDIO_A	mV	int	none	none	1
LO_V_FSIO_NODE_PROC_VDDNB_A	mV	int	none	none	1
LO_V_FSIO_NODE_PROC_VDD_A	mV	int	none	none	1
LO_V_FSIO_NODE_PROC_VTT_A	mV	int	none	none	1
LO_V_G34_NODE_PROC_CORE	mV	int	none	none	1
LO_V_G34_NODE_PROC_VDDNB	mV	int	none	none	1
LO_V_SLOT_NODE_PROC_VDDIO	mV	int	none	none	1
LO_V_SLOT_NODE_PROC_VLDT	mV	int	none	none	1
LO_V_SLOT_NODE_PROC_VTT	mV	int	none	none	1
MachineCheckException	1	int	none	none	1
PositionX	m	float	time	none	1
PositionY	m	float	time	none	1
PositionZ	m	float	time	none	1

Figure 6. The Namespace View displays information about metrics collected for each type of component (or more generally, each *namespace*, since measurements may eventually include jobs and other non-physical entities).

Figure 7. The Graphical View displays relationships between components by connecting them in a graph.

Figure 8. The Time-history View plots how metrics vary over time for subsets of components over which they are collected.

Figure 9. The Operating State View displays the current value of a metric for every component it has been measured on.

metric as a colored pixel in a regular grid with the first component in the bottom left and increasing from left-to-right and then bottom-to-top. These images are generated by each shepherd with transparent pixels where no information is available so that by superimposing images from all the shepherds, a picture of the entire cluster’s state is produced. The shepherds use information cached during metric insertion to generate the images rather than performing SQL queries so that interactive framerates can be achieved. However, in order for this information to be available, the shepherd must be used to insert measurements rather than allowing other processes direct access to the database.

After examining the system state, users may wish to quantify trends and test hypotheses regarding the distribution of metric values. The Haruspex View of Figure 10 allows users to enter parameters for an analysis in the top panes by selecting the type of analysis and the range of components and time over which to sample. Once the user clicks learn, the results are displayed below. Clicking on a colored rectangle summarizing analysis results allows the user to add or view free-form text annotations for the analysis. Clicking on a sequence number in the result pops up a window where users may visualize how well the inferred model matches observations.

Sometimes it is useful to know which components take on a given value. When a metric is constrained to take on discrete values and the number of values is small, the Entities By Value View (Figure 11) can be used to identify which components take on the given value at any time in their entire history. This was used during the milestone runs to determine which cores were assigned to a particular run of a particular application.

Besides its analysis and visualization functions the shepherd’s web server also provides an efficient mechanism for inserting data into the database. The komondor submits a URL encoded with the metric table, component identifier, and measured value to the web server where it is logged and eventually inserted into a persistent store. This is described in greater detail in Section 3.2.5.

3.2.7 Node-Local Feedback to Application

In addition to enabling streaming data off the platform, the LDMS service also supports on-node queries of the data. Thus an application or intermediate code can also query LDMS for resource state and use that information to invoke resource-aware run-time response.

In order to illustrate this capability, we targeted applications that already have capabilities for dynamic reconfiguration and we augmented that process to include dynamic resource-state information into the reconfiguration process. In particular, we targeted Aria [10] and Fuego [7, 6] in the Sierra [8] suite, which, under certain conditions, repeatedly rebalance during run-time using the Zoltan [5] partitioner.

Zoltan determines balanced partitionings taking into account specified *object weight(s)* and *partition size(s)*. Examples of the former include number of elements or particles. Aria and Fuego support some user selectable options for weight, imbalance thresholds to invoke rebalance, and rebalance frequency rates. While in practice in these codes the partition size is typically uniform, we have modified Zoltan within Sierra to call the node-local ldmsd, either directly or indirectly

Analysis Type

☒ Descriptive
 ☐ Correlative
 ☐ Multicorrelative

☐ Principal Components
 ☐ Multicorrelative Time Series

☐ Order
 ☐ Contingency

Domain

Namespace:
 ☐ rack
 ☒ cn
 ☐ link

Metrics:

Select all
 Clear all

Active
 CPU0idle
 CPU0percentutil
 CPU0sys
 CPU0user
 CPU1idle
 CPU1percentutil
 CPU1sys
 CPU1user
 CPU2idle
 CPU2percentutil
 CPU2sys
 CPU2user
 CPU3idle
 CPU3percentutil
 CPU3sys
 CPU3user
 CPUidle
 CPUpercentutil
 CPUsys
 CPUuser
 Cached
 Dummy2Temp
 DummyA
 DummyB
 DummyC
 DummyD
 DummyE
 DummyPositionX
 DummyTemp
 Eth0RxBytes
 Eth0RxPackets
 Eth0TxBytes
 Eth0TxPackets
 Heartbeat
 IPAddress0
 Inactive
 LoadAvg15min
 LoadAvg1min
 LoadAvg5min
 Machine
 MaintenanceAction
 MemFree
 MemTotal
 NodeName
 PositionX
 PositionY
 PositionZ
 PwrAddress
 RackID
 RackSlot
 Release
 RunningProcesses
 SerConAddress
 SwapCached
 SwapFree
 SwapTotal
 SysName
 TotalProcesses
 UniformRandomness
 Version

Entities: 1-2
 3 entities in the range 1-3

Time window:
 ☒ Start 1969-12-31 16:00:00
 ☐ End
 ☒ Duration 500

Parameters

No parameters for descriptive models.

Learn

Name	Shep Req	Duration (hh:mm:ss)	Type	Parameters
Zorkmid	2 1	10:00:00	Descriptive	DummyC cn1-3
1	1969-12-31 16:00:00	mean	variance	skewness kurtosis cardinality
	1970-01-01 02:00:00	20.82	3789	1.959 2.729 2000
Fishy	2 2	10:00:00	Correlative	DummyA DummyC cn1-3
1	1969-12-31 16:00:00	x mean	y mean	y variance covariance r cardinality
	1970-01-01 02:00:00	0.2562	13.03 0.6175	831.2 -2.891 -0.128 10000
Cheesy	2 3	10:00:00	Multicorrelative	DummyA DummyC cn1-3
1	1969-12-31 16:00:00	mean	variable	covariance values
	1970-01-01 02:00:00	0.2562	DummyA 0.6175 -2.891	
		13.03	DummyC 831.2	
	2 83	0:16:40	Descriptive	DummyA cn2
1	1969-12-31 16:00:00	mean	variance	skewness kurtosis cardinality
	1969-12-31 16:16:40	-0.01133	1.001	0.06495 0.1946 1001
	2 84	0:16:40	Correlative	DummyA DummyC cn2
1	1969-12-31 16:00:00	x mean	y mean	y variance covariance r cardinality
	1969-12-31 16:16:40	-0.01133	41.14 1.001	6739 -1.078 -0.0131 1001
	2 85	0:16:40	Multicorrelative	DummyA DummyC cn2
1	1969-12-31 16:00:00	mean	variable	covariance values
	1969-12-31 16:16:40	-0.01133	DummyA 1.001 -1.078	
		41.14	DummyC 6739	

Annotations

Name: Zorkmid
 Notes:

You are in a maze of twisty passages, all alike.

Update

Figure 10. The Haruspex View displays results of analyses and allows submission of new analyses of metric data.

Figure 11. The Entities By Value View displays all of the distinct values a metric takes on and allows users to request the set of components taking on a given value.

through intermediate code, to get system state information on demand and evaluate it in order to determine relative partition size targets at the core level. In this work, we address providing feedback to an application to enable its reconfiguration at run-time; follow-on work involves using the system developed here to determine what data is relevant, what functional forms should be used, and what weighting across the partitions should be used. However, demonstration usage of the feedback mechanism is provided as part of this work.

Note that the LDMS update frequency is independent of the timescales of the Zoltan partitionings. Zoltan will receive partition sizes based on the most current system state information held by LDMS. In this way there is minimal delay in obtaining the system state information, with the tradeoff of possibly using out-dated system information. Note however, that ultimately one does not necessarily need to achieve the *best* partitioning, but rather only a *better* partitioning.

4 Criteria-Related Results

Based on the text and intent of the milestone we define the following to be the minimum success criteria for this milestone:

1. The platform from which resource information will be acquired and on which applications will be run will be an ASC relevant architecture.
2. Acquisition and storage of > 20 metrics per processing element over $> 10K$ processing elements at a collection frequency of 6 per minute.
3. Analysis of information pertaining to resource utilization for a particular application run or set of runs spanning $10K$ processing elements.
 - (a) Post-run analysis on aggregate archived information
 - (b) Run-time analysis and visualization on streaming data
4. Use of a subset of metric values being collected to affect run-time work to resource mapping in an application run and/or ensemble of application runs.
 - (a) Affect of feedback on application performance will be evidenced through a change in load to resource mapping, application run-time or both as compared with the application running concurrent with data acquisition but with no feedback.
 - (b) At least one application run will span $> 10K$ processing elements.

4.1 ASC Relevant Platform

This section provides evidence related to our platform relevance criteria described in Section 4 item 1.

In this work we utilized two systems: The Cielo platform at LANL whose resource utilization data was moved across the wide area DISCOM network to Whitney, a TLCC system at SNL/CA, for remote analysis and the Cielo Del Sur System at SNL/NM whose data was moved across the wide area to desktop machines at SNL/CA for remote analysis. Both Cielo and Cielo Del Sur are Cray XE6 platforms and hence relevant ASC platforms as Cielo is the latest large scale tri-lab ASC HPC platform acquisition. The instantiation of our collection and feedback system on these systems are shown in Figures 12 and Figure 13.

4.2 Data Acquisition

This section provides evidence related to our data acquisition criteria described in Section 4 item 2.

Figure 12. System instantiation to address Cielo

Figure 13. System instantiation to address Cielo Del Sur

Figure 14. Fractional overhead for LDMS processes on a per core basis running on a particular compute node

Figure 15. Fractional overhead for LDMS processes on a per core basis averaged over all nodes involved in a 10,112 processor Aria run

Figure 16. Fractional overhead for LDMS processes on a per core basis, including high and low (0), over all nodes involved in a 10,112 processor Aria run

Data sources used in this work were the following files from the proc filesystem: vmstat, meminfo, stat, kgnlnd, interrupts, and PID. We successfully collected 1012 data values per node over 646 nodes (10336 cores), 720 of which were core related (45/core) with a collection period of 9 seconds. The overall overhead on a per node basis for using our monitoring software was 0.02%. Figure 14 shows the overhead variation, on a per core basis, for all LDMS processes running on a particular node. The average over all nodes, on a per core basis, is shown in Figure 15 and again in Figure 16 with high and low bars (note that for all cores the low is zero).

4.3 Data Analysis and Visualization

This section provides evidence related to our data analysis and visualization criteria described in Section 4 item 3. In this work we utilized run-time visualization over various parameters to determine some metrics of interest which we then used in post-run analysis. Both types of analysis are described in this section along with some results and screenshots.

Our first step in identifying metrics of interest was to utilize our run-time display to visualize the distribution of nonvoluntary context switches on a per-user-process basis. Since each user process is bound to a particular core this is equivalent to looking at these same metrics on a per-core basis where the core to rank mapping is acquired from the ALPS scheduler at run-time. The reason for this particular metric is that their occurrence means that the running application process was swapped out by the scheduler in order to run another process which will obviously affect the performance of that process and the application. Figure 17 shows that not only are nonvoluntary context switches not spread uniformly across cores of a each node but that across many nodes (556 in this case) they happen preferentially on certain cores (5, 9, and 13 in this case). Thus we made use of nonvoluntary context switches as a metric for providing feedback to the application as described in Section 4.4.

Looking at where system processes are running (Figure 18) yields insight into what is driving these context switches. It is obvious from this information that the system processes are what is driving the context switches and visualizing this over time shows that though there is some temporal migration of these processes among cores of a node they seem to be scheduled relatively statically on these sets of cores. It is interesting to note that running an application with core-specialization [2] which puts all non application processes on a particular core (default is 15 in this case) causes them to generally be preferentially scheduled on core 15 afterwards presumably due to cache. Figure 19 shows data from a 1024 core (64 node) run where the green plots (system time spent on a core - system time for the application process on that core) summed over the run and the red plots system time spent on our monitoring processes on each core. This figure shows the preferential scheduling of non-application processes on cores 5, 9, and 13. The x axis in this figure is core number and the y axis is clock ticks. The anomaly for a process on a core 11 appears to show a process which had at least some of its system time scheduled on other cores.

Another metric of interest to us in this work was what fraction of each core's available cycles were being used by the application. In order to determine this we used the ratio of the sum of the application processes user and system time to the sum of the application processes user time,

Metric State

Figure 17. Nonvoluntary context switches for one collection period (9 sec) on a per core basis over 556 nodes (8896 cores) of CDS (Note that red is 0 and blue is 1000)

Metric State

Figure 18. System time for one collection period (9 sec) on a per core basis over 556 nodes (8896 cores) of CDS (Note that red is 0 and blue is 3)

Figure 19. Plot, for a 1024 processor Aria run, of the sum of system time (measured in clock ticks), on a per core basis, dedicated to non-application tasks for that core (red-LDMS, green - other). Note that the negative spike for core 9 appears to be due to an application process having system time on its behalf spent on a different core

Figure 20. Fractional CPU utilization for the Aria application on a per core basis running on a particular compute node on Cielo

Figure 21. Fractional CPU utilization for the Aria application on a per core basis running on a particular compute node on CDS

application process system time, and core idle time. Figure 20 shows this for a particular node aggregated over the run-time of the Aria application run over 10,112 cores on Cielo while Figure 21 shows the same plot for Aria run on CDS over 8310 cores (note that y axes of the graphs do not start at 0). This fluctuation on a per node basis led us to utilize this imbalance on a per node basis as another metric for application feedback as described in Section 4.4.

Figure 22. Fractional utilization of core cycles for the Aria application on a 10,112 core run averaged over the course of the run on a per core basis.

We also analyzed the fractional utilization across all 632 nodes of an Aria run on 10,112 cores. The results are shown in Figure 22 including high and low water marks. Note that this is the aggregate over the whole run.

4.4 Affect Run-time Work to Resource Mapping

This section provides evidence related to feedback criteria described in Section 4 item 4.

In this work we demonstrate the capability to use system information at run-time; detailed analysis to determine which data is most relevant to performance tuning and the functional forms to be used in transforming such information into partition sizes is longer-term follow on work. Note that similarly the partitioning frequencies have not been tuned.

We implemented run-time feedback to the Zoltan partitioner in two applications: SIERRA/Aria and SIERRA/Fuego.

4.4.1 SIERRA/Fuego

SIERRA/Fuego is a particle code. In the problem evolution particles move, are injected, can split, and can be removed. Because of this, the computational load will vary in the physical space through time, and hence the workload per processor will vary through time. Fuego can rebalance, using Zoltan, when the particle imbalance across processors exceeds a user defined threshold.

The partitioning in the physical space for an problem across 32 processors (32 chosen for visual clarity) is shown in Figure 23. Particle number drives the partitioning of the space. At this time the partitions are unevenly divided in space, seeking to balancing the particle density which is greatest at the bottom left of the figure. The regions of high particle number vary with time, as the particle locations and evolve inward and up and thus partition sizes and locations will change with time.

Figure 23. Mapping of physical space and particles to partitions in a Fuego problem. Particle number drives the partitioning of the space. The regions of high particle number vary with time, starting at the lower left of the figure and evolving inward and up. Thus partition sizes and locations will change with time.

The effect of the particle evolution on the partitioning is shown in Figure 24. The left column shows partition distributions based on particle number with a uniform partition target size for the case above but on 64 processors. The upper left is the distribution after a rebalancing. It is seen that within the physical constraints of the problem and the algorithm that a completely uniform distribution of particles per partition is not achieved in practice. As particles move in time the particle-partition mapping changes and the distribution is spread out thus triggering a rebalancing. The distribution change after a number of timesteps is shown in the middle left figure and the resultant rebalancing is shown in the middle bottom.

While this example problem is insufficiently large to necessitate running on larger processor counts, it represents the type of dynamic application needs that we seek to address with this work. We illustrate the ability to feed back information to the application to affect the work-load to resource balance in Figure 24. In this case the ratio of idle cycles to total cycles utilized since the last partitioning was used to determine the target partition sizes with the intent that processors that had exhibited larger idle time could take on more load. (In practice we expect that a function that weights more heavily more recent load will be a better choice.) The resultant partition distributions are generally a bit wider as a result of variations in the target sizes(right column).

Figure 24. Particle Distributions in the Partitioning Evolution in a Fuego problem (a) after a rebalance (top) (b) after a few timesteps before the next rebalance (middle) (c) after the next rebalance (bottom) (d) without feedback (left column) (e) with feedback (right column). Rebalancing in general seeks to provide an even distribution but is not in practice completely uniform (top and bottom). As the problem evolves and particles move the distribution widens triggering the need for rebalancing (middle).

Figure 25 shows that for this simple demonstration there is a resultant improvement in computational cycles dedicated to the application execution across all processors involved. Again, detailed determination of data and the feedback functional form for this case were beyond the scope of this work and thus this should not be taken as a general indicator of potential results.

Figure 25. Simple demonstration shows improvement in computational cycles across all processors involved.

4.4.2 SIERRA/Aria

SIERRA/Aria is a thermal code. In a general sense the dynamics of the simulation is not as closely tied to the computational load as in the particle case since, for instance, a hot spot in the calculation does not change the size of a matrix, but rather the values within it. However the problem may still be subject to imbalance and hence Aria provides the ability to rebalance using Zoltan when element imbalance or assembly time imbalance exceed user-defined thresholds.

We ran an Aria problem which has been used in scaling tests and would support the 10K processor target. Analysis on several runs indicated that non-voluntary context switches and interrupts occurred non-uniformly. Since these impact the time dedicated to the application on a resource, we chose to assign partition size based upon the number of occurrences of non-voluntary context switches and interrupts since the previous rebalancing. Extensive analysis of the impact of the occurrences on the amount of time taken from an application was beyond the scope of this current work.

Results of this feedback are shown in Figure 26. In this case the distribution of partition sizes resulting was very broad resulting in a broad distribution of elements per partition as compared to the initial distribution. Non-voluntary context switches and interrupts occurred preferentially on Cores 0 and 9 during this run and as a result, the smaller partitions did occur preferentially on those cores. Overall, while this did demonstrate the ability to enact run-time feedback and mapping at the target scale, more work is required to determine a weighting of these quantities that might be advantageous for overall application run time.

We applied the idle cycle criteria previously used in the Fuego problem to this problem on a 8310 core run on CDS. We targeted a relatively comparable size, but still with less than the full complement of cores at our disposal, since this enabled us to also test the problem with Cray's core specialization [2] option. Results of the partitioning with and without feedback are shown in Figure 27 for a set of selected timesteps. The non-feedback distribution changes only slightly in time, as expected. This is also true of the spatial placement (unshown) which we examined on

Figure 26. Feedback Distributions for 10,112 processor run of an Aria problem on Cielo. Processors exhibiting more context switches and interrupts are assigned a smaller target partition size. In this case the distribution of partition sizes is broad (top) resulting in a broad distribution of elements per partition as compared to the initial distribution (middle). Non-voluntary context switches and interrupts occurred preferentially on Cores 0 and 9 across all nodes and, as a result, the smaller partitions occur preferentially on those cores.

smaller scale runs. In the feedback case an early distribution (green) is much broader than the non-feedback distribution, probably to a larger degree than desired given the general computational balance of the problem. In this case the use of a feedback function that includes computational load drives the target partitioning and thus the distribution to a more balanced distribution.

Figure 27. Partition Distributions for selected timesteps for an 8310 processor run of an Aria problem on Cielo. Processors exhibiting larger ratio of idle cycles to total cycles utilized since the last partitioning are assigned a higher partition size. Uniform partition sizes (top) results in tighter distributions than those without feedback. In the feedback case, when too broad partitioning occurs early (bottom, green), the partitioning feedback criteria will adjust the distribution.

5 Conclusion

As required by the milestone we have developed a feedback system for intelligent dynamic resource allocation, used it acquire and analyze resource utilization data, and affected run-time to work resource mapping on an ASC relevant architecture at the target scale.

The infrastructure demonstrated in execution of this milestone has potential to provide insight to developers and researchers alike and provides a foundation for further development in this area. While this milestone specifically targeted run-time feedback we are also interested in using the information gained to more appropriately place application processes for applications that do not support runtime adjustment. The data storage, visualization, and analysis capabilities developed and enhanced during the execution of this milestone can be used both at run-time and for post-processing. The latter would be of interest to analysts who could play back through their run.

This work has inspired collaboration and follow on LDRD work with Zoltan to develop run-time resource and machine aware-partitioning and hierarchical partitioning. This infrastructure will provide information that will feed into algorithm development and determination of what machine data is of value and how to properly weight resources in the machine model. Hierarchical partitioning developed in Zoltan will enable a more targeted mapping of work-load to resource, particularly when considering issues such as shared memory and repartitioning within and across nodes.

References

- [1] Using Cray Performance Analysis Tools. Cray Doc S-2376-52.
- [2] Workload Management and Application Placement for the Cray Linux Environment. Cray Doc S-2496-31.
- [3] J. Brandt, B. Debusschere, A. Gentile, J. Mayo, P. Pebay, D. Thompson, and M. Wong. OVIS-2: A robust Distributed Architecture for Scalable RAS. In *22nd IEEE International Parallel and Distributed Processing Symposium, 4th Workshop of System Management Techniques, Processes, and Services*, 2008.
- [4] Douglas Crockford. RFC 4627: The application/json media type for JavaScript Object Notation (JSON). RFC 4627, July 2006.
- [5] K. Devine, E. Bowman, R. Heaphy, B. Henrickson, and C. Vaughan. Zoltan Data Management Services for Parallel Dynamic Applications. In *Computing in Science and Engineering*, volume 4, pages 90–97, 2002.
- [6] S. Domino, G. Wagner, A. Luketa-Hanlin, A. Black, and J. Sutherland. Verification for Multi-Mechanics Applications. In *48th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference (AIAA Paper 2007-1933)*, 2007.
- [7] S. P. Domino, C. D. Moen, S. P. Burns, and G. H. Evans. SIERRA/Fuego A Multi-Mechanics Fire Environment Simulation Tool. In *41st AIAA Aerospace Sciences Meeting (AIAA Paper 2003-0149)*, 2003.
- [8] H. C. Edwards. Sierra Framework Version 3: Core Services Theory and Design, 2002.
- [9] Sergey Lyubka. Mongoose. <http://code.google.com/p/mongoose/>.
- [10] P. Notz, S. Subia, M. Hopkins, H. Moffat, and D. Noble. Aria 1.5 User Manual. Sandia National Laboratories Report SAND2007-2734, 2007.
- [11] University of Oregon. Tuning and Analysis Utilities:TAU. <http://www.cs.uoregon.edu/Research/tau/home/php>.
- [12] OProfile. OProfile. <http://oprofile.sourceforge.net/news>.
- [13] Sandia National Laboratories. OVIS. <http://ovis.ca.sandia.gov>.
- [14] M. Schulz, A. Arowojolu, J. Blanchard, J. Brandt, S. Futral, J. Mellor-Crummey, B. Miller, D. Montoya, M. Rajan, K. Roche, and Zosel. M. Tools and Tool Support for the Exascale Era, 2011.

FY11 ASC/CSSE L2 Milestone #3968 Review

Doerfler, Douglas W

To: [Dosanjh, Sudip S](#)

Cc: [Brandt, James M](#); [Gentile, Ann](#); [Thompson, David C](#); [alumbaugh2@linl.gov](#); [Larry Kaplan \[kaplan@cray.com\]](#); [Stephen Scott \[sscott@tntech.edu\]](#)

Wednesday, September 07, 2011 5:06 PM

Sudip,

A review of FY11 L2 milestone #3968, Develop Feedback System for Intelligent Dynamic Resource Allocation, was held on September 7th, 2011. The review was led by Jim Brandt and included team participants Ann Gentile and David Thompson. The review team included myself, TJ Alumbaugh (LLNL), Larry Kaplan (Cray) and Stephen Scott (ORNL).

The milestone team defined 4 major completion criteria:

- A. The platform from which resource information will be acquired and on which applications will be run will be an ASC relevant architecture.
- B. Acquisition and storage of > 20 metrics per processing element over > 10K processing elements at a collection frequency of 6 per minute.
- C. Analysis of information pertaining to resource utilization for a particular application run or set of runs spanning ~10K processing elements
 - a) Post-run analysis on aggregate archived information
 - b) Run-time analysis and visualization on streaming data
- D. Use a subset of metric values being collected to affect run-time work to resource mapping in an application run and/or ensemble of application runs
 - a) Affect of feedback on application performance will be evidenced through a change in load to resource mapping, application run-time, or both as compared with the application running concurrent with data acquisition but with no feedback.
 - b) At least one application run will span > 10K processing elements.

The milestone team presented evidence in support of the above criteria that included details on the method used, the impact of their method on resource utilization, visualization results demonstrating the ability to view selected metrics over a period of runtime, and the impact of using the method to affect the partitioning capabilities of Zoltan with the later being demonstrated using the applications Sierra/Fuego and Sierra/Aria.

The review team felt that the evidence provided in the review demonstrated that the above criteria were indeed met and the milestone was fully completed.

Doug Doerfler

DISTRIBUTION:

1	MS 9159	Jim Brandt, 8953
1	MS 9152	Ann Gentile, 8953
1	MS 9159	David Thompson, 8953
1	MS 0899	Technical Library, 9536(electronic copy)