SAND2016-3305C

# Design and implementation of a scalable monitoring system for Trinity

A. DeConinck, A. Bonnie, K. Kelly, S. Sanchez, C. Martin, and M. Mason
Los Alamos National Laboratory
Los Alamos, NM
Email: {ajdecon,noranzyk,kak, samsanchez,c_martin,mmason}@lanl.gov

J. Brandt, A. Gentile, B. Allan, and A. Agelastos
Sandia National Laboratories
Albuquerque, NM
Email: {brandt,gentile, baallan,amagela}@sandia.gov

M. Davis and M. Berry
Cray Inc.
Email: {u3186,mrberry}@cray.com

***Abstract*—The Trinity XC-40 system at Los Alamos National Laboratory presents unprecedented challenges to our system management capabilities, including increased scale, new and unfamiliar subsystems, and radical changes in the system software stack. These challenges have motivated the development of a next-generation monitoring system with new capabilities for collection and analysis of system and facilities data.**

**This paper presents the design of our new monitoring system, its implementation, and an analysis of impact on system and application performance. This will include the aggregation of diverse data feeds from the compute platform, such as system logs, hardware metrics, power and energy usage, and High Speed Network performance counters; as well as data from supporting systems such as the parallel filesystem, network infrastructure, and facilities. We will also present tools and analyses used to better understand system and application behavior, as well as ongoing development in our monitoring processes based on early experiences with the machine.**

## I. Background

### A. Review of the Trinity system

*Trinity* is the new Cray XC-40 system currently being deployed at Los Alamos National Laboratory (LANL) for the Alliance for Computing at Extreme Scale (ACES), a collaboration between LANL and Sandia National Laboratories (SNL). The first phase of the Trinity deployment, completed in 2015, consists of 9,324 compute nodes with dual-socket Intel Xeon CPU E5-2698 v3 processors, as well as 594 service nodes. The second phase, to be completed in 2016, will include over 9,000 compute nodes with the new generation of Intel Xeon Phi (“Knights Landing”) processors and a few hundred additional service nodes.

In addition to a new processor architecture, Trinity will include several new subsystems which are new or unfamiliar for ACES systems. These include several hundred service nodes with solid-state disks (SSDs), running the Cray DataWarp software, which will act as a fast “burst buffer” storage tier; two large Cray Sonexion Lustre filesystems, each totaling 35 PB of usable space; a new water cooling system; a new power management subsystem; and the Aries [1] network. The Trinity deployment also includes the deployment of the new Cray system software stack (CLE 6.0 / SMW 8.0), which includes a tight integration with the Ansible [2] configuration management tool and is architected in a very different manner from either our previous Cray deployments (CLE 4.x and 5.x), or our commodity cluster systems.

### B. Motivations for monitoring

Given the extensive changes introduced by Trinity relative to past deployments, system monitoring has become a key tool in improving our understanding of how this new system works. Effective monitoring will give us real insights into the actual, rather than expected, behavior and performance. The design of our monitoring system is dictated by several important workflows, each with its own time scale.

First, the monitoring system should generate timely alerts for significant system events. These events are related to the availability of the whole system, a set of specific system components or services, or any security related events. This also includes environmental conditions for the system, such as temperature, power, leak detection, or other facility conditions. Alerts for these events should be generated within seconds or minutes of the event occurring, so that system operators or administrators can take action to correct these issues.

Second, the monitoring system should be able to produce performance data for understanding the behavior of user applications. At the system level, this should take the form of metrics on the utilization and availability of system resources, e.g. free memory, CPU utilization, network counters, etc. This data could be used by users in the hours, days, or weeks following an application run to understand their job’s performance and plan their next run, as well as administrators investigating systemic performance issues.

This work has been authored by an employee of Los Alamos National Security, LLC, operator of the Los Alamos National Laboratory under Contract No. DE-AC52-06NA25396 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting this work for publication, acknowledges that the United States Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce this work, or allow others to do so for United States Government purposes. Los Alamos National Laboratory strongly supports academic freedom and a researcher’s right to publish; however, the Laboratory as an institution does not endorse the viewpoint of a publication or guarantee its technical correctness. This paper is approved for release under LA-UR-XX-XXXXX.

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.

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research.

Third, the monitoring system will be used for longer-term analysis of system metrics in order to understand system behavior over longer time scales. This may include such analyses as studies of hardware failure analysis, periodic checks for degradation in performance or reliability, and analysis of typical user workloads in order to understand how the system is used. These types of analysis will be conducted over the time scale of months or years, including studies conducted past the lifetime of *Trinity* itself, and may be used to help plan future hardware procurements.

### C. Insufficiency of existing monitoring stack

Most HPC systems currently deployed at LANL share a common monitoring stack based primarily around log analysis.[3], [4] (See Figure 7.) Each system has some type of "cluster master" server which collects syslog data from all other nodes in the cluster. (In the case of our Cray systems, the system management workstation (SMW) performs this function.) Each cluster master may also run periodic scripts to collect additional data from the system, such as temperature data or InfiniBand metrics, and inject this data into the syslog feed. The cluster master then forwards the collected logs to a dedicated monitoring server ("mon box") which runs a heavily-customized instance of Zenoss[5], an open-source monitoring system. Zenoss applies a series of "filters" to the incoming data, watching for specific message types, and alerts the operations staff to recognized issues using an email, page, or dashboard visualization. In addition to the Zenoss feed, log data is duplicated and fed into a shared Splunk[6] installation, which can be used to interactively search historical data and produce dashboard visualizations.

While this monitoring stack has proved useful for most of our systems so far, it has a few weaknesses. First, it is designed primarily for alerting on well-characterized issues which produce easy-to-parse messages in our log stream. While this is useful for handling most day-to-day issues, it provides little or no visibility for problems which are not yet well-understood or which cannot be easily caught using system logs. Also, while a few "active" checks exist (such as a script for monitoring the InfiniBand fabric), we mostly rely on the cluster software to log the issues we expect to see. Due to the number of new technologies and software versions deployed with *Trinity*, we expect to add many new types of sanity checks and monitor for many new conditions which may affect the system's reliability and performance.

Second, this stack provides little or no visibility into application issues. For the most part, our users are expected to perform whatever application-level logging or monitoring they need in their own code or job scripts. In many cases, the only data our administrators have with respect to any given job is a basic list of metadata from the cluster scheduler: when the job started, when it ended, which nodes were assigned, and the paths to the input script, standard output and standard error files. This limits the ability of HPC staff to help diagnose many types of issues, such as the always-challenging "my job is slow", especially in the case where poor job performance or behavior is not noticed until after the job is completed and many other jobs may have run on the same hardware. Our new monitoring stack for *Trinity* must provide better insight into the actual behavior of the system with respect to any given job to help us better assist our users.

Third, we do not expect our current monitoring system to be able to scale to *Trinity*, either in terms of providing the necessary data throughput or the flexibility to carry out many different types of analysis. Our largest unclassified commodity cluster resource, the 1,600-node *Mustang*, currently produces approximately 140 MB of monitoring data per day. Even accounting only for system and other logs, the first phase of the *Trinity* deployment produces approximately 5.0 GB per day, approximately 36 times more than *Mustang*. This increase will be further exacerbated by the collection of new types of data in order to address the issues mentioned above with respect to understanding application performance and diagnose system issues. We will also need the flexibility to be able to provide our monitoring data feeds to many different applications, in addition to Zenoss and Splunk, in order to carry new types of analysis and better understand the system.

## II. Architecture and Implementation

### A. Overview

Our new monitoring system for *Trinity* provides a flexible, scalable architecture for ingesting, distributing, and processing many different types of data about the behavior of the system. The monitoring system itself takes the form of a cluster architecture, provisioned and managed by the same tools we use to manage our commodity HPC clusters. Various types of monitoring data are produced by each component of *Trinity*, and are then transported to and collected by the monitoring cluster depending on the source and type of the data. This data is pre-processed and stored by the nodes of the cluster, and then forwarded on to a message bus where it is distributed to additional systems for further analysis, including our existing Zenoss and Splunk infrastructures. The overall flow of data through this system is shown in Figure 1.

In this section, we will describe in detail the generation of monitoring data by *Trinity* and related systems; the implementation of our monitoring cluster for collecting and processing this data; and its distribution by LANL's shared monitoring infrastructure for further analysis.

### B. Collection of platform data

*1) Log data:* As on our other platforms, log data represents an important data source for understanding *Trinity's* behavior. The logs are generated by a wide variety of components and software services, including the compute nodes and service nodes which make up "*Trinity* proper", as well as "external" components such as the parallel filesystem, user-facing front-end servers, the workload management server, and the InfiniBand subnet manager (SM). Each of these components is connected, directly or indirectly, to a dedicated "monitoring network" (Figure 2). Logs are collected and forwarded to the nodes of the monitoring cluster using syslog, and combined with the overall stream of monitoring data.

Several services, including the Torque[7] cluster resource manager and the Moab[8] job scheduler, write many of their logs directly to local log files on the nodes where they run, instead of logging to a shared infrastructure. For these services,

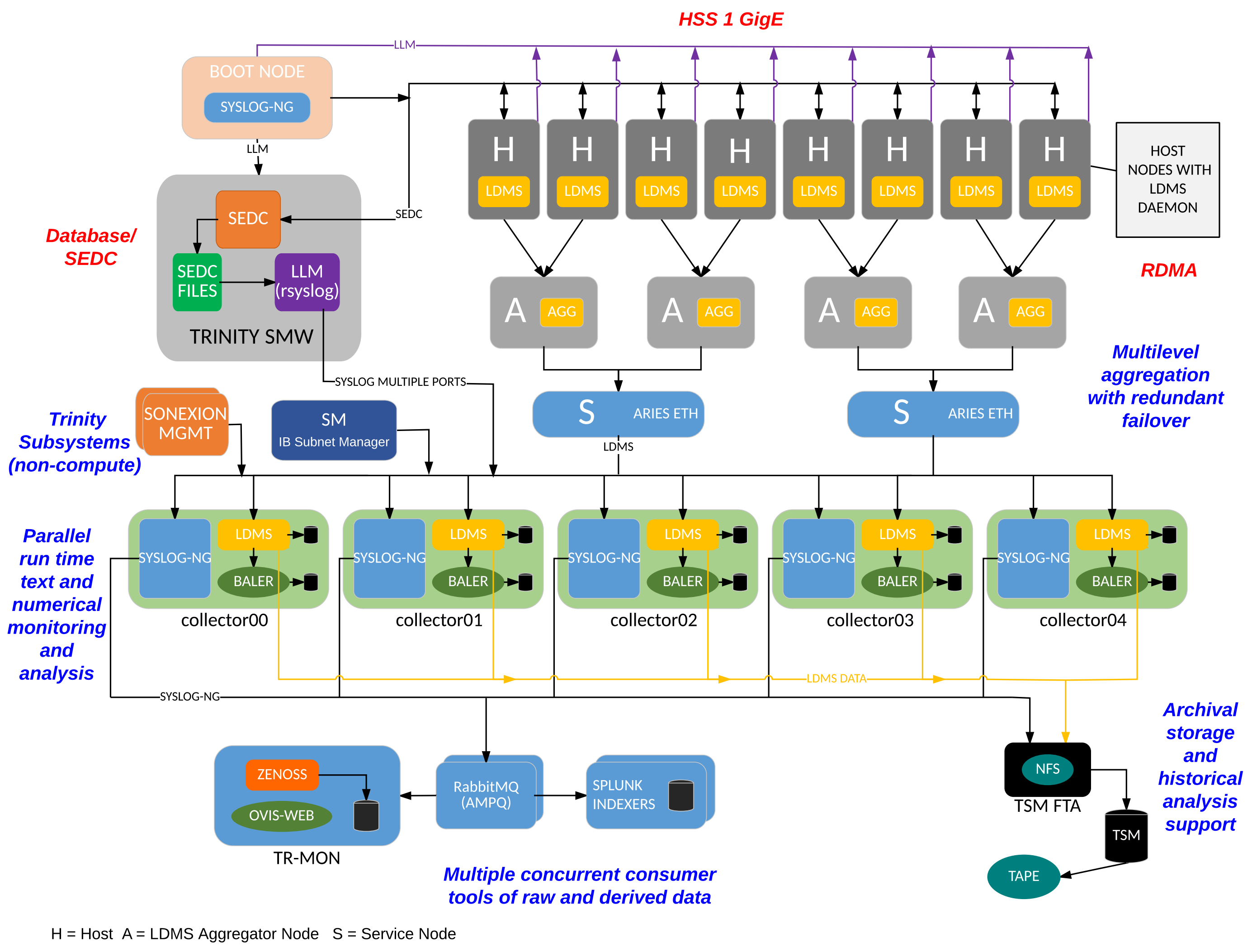


Fig. 1: Data flow paths from Trinity and associated subsystems to the monitoring infrastructure. LDMS aggregator nodes labeled “A” are Cray service nodes or re-purposed compute nodes.

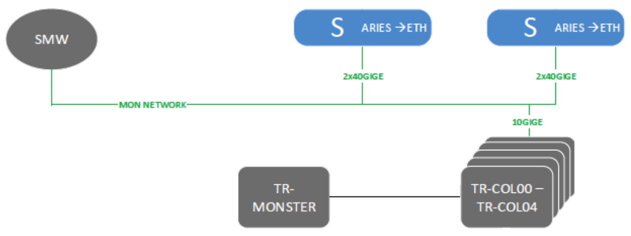


Fig. 2: A dedicated 10GbE “monitoring network” connects the SMW, the monitoring cluster, the level three LDMS aggregators (see Section II-B3), and external services which log data related to *Trinity* (not shown).

we wrote *rsyslog* rules using the *imfile* plugin to ingest these logs into the syslog stream with appropriate tags. Once in syslog, these logs were forwarded on to the rest of the shared monitoring infrastructure.

*2) System environment data collections (SEDC):* SEDC is a Cray-specific service for collecting and reporting environmental data on Cray systems. [9] This data includes information from sensors located on hardware components throughout the system, such as blade and cabinet controllers, processors, memory, fans, and components of the liquid cooling system; and includes such metrics as fan speeds, temperatures, voltages, and valve positions.

SEDC metrics are collected over the HSS network and stored on the SMW by the *sedc_manager* daemon. Unfortunately, while *sedc_manager* provides the option of storing SEDC metrics to either flat files or the power management database (PMDB), it does not provide a native mechanism for forwarding these metrics to a downstream monitoring system. While Cray recommends using the PMDB store for easy queries of SEDC metrics from the SMW, our experiments with this store gave us no easy way to extract metrics and inject

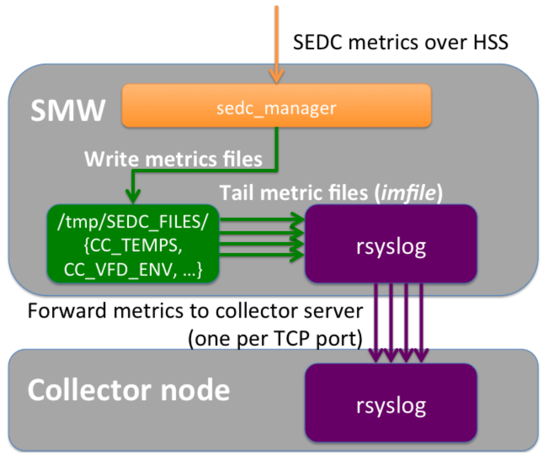

SEDC metrics over HSS
SMW
sedc_manager
Write metrics files
Tail metric files (imfile)
/tmp/SEDC_FILES/ {CC_TEMPS, CC_VFD_ENV, ...}
rsyslog
Forward metrics to collector server (one per TCP port)
Collector node
rsyslog

Fig. 3: SEDC metrics are collected over the HSS by the sedc_manager daemon on the SMW. This daemon writes each metric on a per-device, per-time-step basis to a file named by metric name under `/tmp/SEDC_FILES`. To ingest this data, we have configured the *rsyslog* daemon to tail each of these files using the *imfile* plugin. Each metric feed is then forwarded to one of the collector nodes, with each metric being sent over a separate TCP port so it can be identified at the far end.

them into the larger monitoring system using standard tools.

Instead, we chose to configure *Trinity* to use the flat file store. With this store, each SEDC metric is saved to a separate file, with files being periodically rotated within its directory according to configuration at `/opt/cray/hss/default/etc/sedc_srv.ini`. To ingest these metrics into a stream we could forward, we wrote a set of rules for the *rsyslog* daemon on the SMW to tail these files using the *imfile* plugin. Once these metrics were in the syslog stream, forwarding rules were written to send each metric to one of the collector nodes in the monitoring cluster. A separate TCP port was used to forward each metric, as the individual metric lines contain no metadata, but instead just a comma-separated list of numeric values. Corresponding syslog rules were written on the collector side to accept these values and label them appropriately for forwarding on to the rest of the monitoring infrastructure. A schematic of this scheme is shown in Figure 3.

*3) In-band Data Collection and Transport:* We are particularly interested in getting resource utilization data globally in order to understand our workload demands, system state, and where multiple applications contending for shared resources, such as the HSN and file systems, can result in performance issues.

There are a number of data sources which are not made available via the standard Cray monitoring services or else are not made available at the frequencies we would want to resolve features of interest. Where such data is exposed via node-level interfaces, we can collect it on the node and transport it off the system to the monitoring cluster via the HSN. For the collection and transport infrastructure, we use the Lightweight Distributed Metric Service (LDMS) [10], which is used on NCSA's Blue Waters [11] Cray XE/XK for collecting similar information, however there are some variations in the collection that we describe here due to additional exposed data on the XC. In this section, we first describe the data sources and access, followed by the instantiation of the collection, transport, and storage architecture on Trinity.

*In-band Data Sources and Access Methods*

The data sources we will collect in-band include the Aries network counters [12], which are are made continuously available only via node level interfaces. These counters can be used to provide insight on traffic and congestion. While they are used internally for routing and congestion-response decisions, higher-level information of this sort is only transported via the HSS and made available to the logs in problem cases (e.g., congestion and quiescence). There are significantly more counters exposed for the Aries than for the Gemini [13], and therefore, while the Aries network counter information can be made available via the *gpcdr* interface [14] which exposes such data on node via the `/sys` filesystem, as is done on Blue Waters, we instead read the Aries network counter information via Cray's *gpcd* interface which has a lower performance overhead for processing. The general methodology for reading the *gpcd* counters applied to the Gemini be found in [15], with detail specific to our collection of the Aries counters in [16]. This accounts for approximately 850 metrics.

Power data is newly exposed on the XC as opposed to the XE/XK. However, while power data is made available via SEDC, it is reported via SEDC at 1 Hz with higher frequency data only available for limited times and node count [17]. Power data is exposed at 10Hz at the node via the `/sys` filesystem. In our Trinity Application Readiness Testbeds (ART systems), we have seen [18] that the 10Hz data can be collected and resolved across nodes to provide insight into features of interest. Note that while we collect the data at 10Hz, we transport it at a lower frequency; this is described further below. This accounts for 2 metrics.

In addition, we collect information on the shared parallel file system, Lustre, such as opens, closes, reads and writes; current free and active memory; and CPU utilization information. This data is largely from `/proc`. This accounts for approximately 85 metrics.

*Architectural Instantiation*

The LDMS infrastructure provides collection, transport, and storage functionalities for HPC monitoring. LDMS is a plugin based infrastructure. Plugins to and configuration of the same core daemon, *ldmsd*, determine the functionality of a particular instantiation of that daemon. Canonically, daemons with sampling plugins collect data on the nodes into *metric sets*; daemons that collect *metric sets* from other daemons are called *aggregators*; daemons with storage plugins store, and optionally process, their *metric sets*. Only the current *metric set* is retained on a sampler or aggregator node at any time in order to minimize the required memory.

The constitution of a *metric set* is up to the implementer. It may be convenient to gather data from related sources by a single plugin and thus into a single *metric set*. An additional consideration is that a single timestamp is associated with a *metric set*, which can simplify analysis. On *Trinity*, then, we are dividing the in-band data amongst four sampling plugins: one for the 10Hz power data; one for the Aries NIC metric data; one for the Aries Router metric data; and a catch-all for the remaining data. The 10Hz power metric set is roughly a vector of multiple timestamped instances of the 10Hz data which then aggregated at a lower frequency. Since all nodes on a blade have access to the same Aries router data, the same Aries Router metric sampler plugin is run on each node, but with different configurations in order to reduce the amount of duplicate data collected.

The aggregators use a pull model to fetch data from the sampling daemons in order to minimize the impact and complexity of the daemons running on the compute nodes. The pull utilizes the Remote Distributed Memory Access (RDMA) protocol in order to minimize the CPU involvement of the compute nodes in the data transport. Arbitrary connection topologies are supported. Different transport protocols can be used at the different aggregators (e.g., RDMA vs socket).

Typically, then, samplers are run on compute nodes and aggregators are run on nodes not acting as compute nodes but connected by the HSN to the sampler nodes in order to take advantage of RDMA over the HSN. Location of storage daemons is guided by the constraints of the storage. If the storage daemons are off-cluster they can aggregate from aggregators over socket (as opposed to RDMA).

On *Trinity*, we have set a target of one second intervals for collection and transport, with the exception of the power data which is collected at 10Hz and aggregated at 1 Hz, in order to obtain relatively fine-grained data about resource consumption on each node. The constraints of the *Trinity* deployment and possible limits on the scalability of aggregation have led us to employ a three-level aggregation scheme for transporting metrics off the system.

When initially investigating the LDMS fan-in ratios (sampler hosts to aggregator ratio) on XE/XK systems we ran into a limit of about 16,000:1, which Cray told told us was a known bug/limitation of the particular kernel they were running and that newer kernels had a patch for it. We have not yet re-investigated this limit on the current *Trinity* software stack. However, using this as a general guideline, since *Trinity* will eventually comprise more than 19,000 compute nodes and nearly 1,000 service nodes, we would then require a minimum of two aggregators for the compute nodes, with at least four desired for reliability and failover.

However, in order to transport collected metrics off the system, we have to provide external network connectivity to our aggregators from the storage daemons on the monitoring cluster and the *Trinity* deployment only included two unallocated service nodes which could be used to transport this data. Thus, our aggregation scheme, as shown in Figure 1, includes a *level one* tier of four re-purposed compute nodes (indicated by *A* with the *ldms* daemons shown as *AGG*) which pull from the *ldmsd* samplers (shown as *LDMS* running on the hosts indicated by *H*) running on compute and other service nodes via RDMA over the HSN, with failover between pairs of aggregators; a *level two* tier consisting of the two previously-unallocated service nodes (indicated by *S*, with both the Aries and the external Ethernet connections) each of which can act as a failover partner for the others, which pull from the level one aggregators, also via RDMA over the HSN; and a *level three* tier consisting of the the nodes of the monitoring cluster (shown as *LDMS* on the green monitoring cluster nodes), which pull from the level two aggregators over socket.

The third level aggregators also perform *in transit* processing of some of the metrics in order to produce functional forms that are more conducive to immediate analysis, such as rates and ratios of various raw metrics. Both the raw and the derived data are used for limited-term analysis on the monitoring cluster and stored off cluster for historical analysis and archival storage. These aggregators also forward selected metrics on to the shared LANL monitoring infrastructure.

## C. Collection of facilities data

The facilities infrastructure had to change in order to accommodate *Trinity*. Heat exchangers between racks of *Trinity* cool the air flowing from one end of racks to the other. This configuration required additional piping, pumps, and construction to the existing data center and facilities. The new infrastructure specific to *Trinity* is comprised of two cooling loops, a primary and secondary loop. Heat exchangers between the loops remove heat generated by *Trinity*.

The primary loop provides 45°F inlet water from four cooling towers and three pumps. This water is recirculated taking the heat generated by *Trinity* from the heat exchanger into this loop and cools it down again via the cooling towers. The secondary loop resides on the other side of the heat exchanger and is considered a closed loop to *Trinity*. It flows to remove the heat generated by *Trinity* but is never replaced with different water. This allows for a "clean" inside loop to the cluster which reduces the need for water treatment and prevents buildup on the cluster side.

*1) TRANE-Building Automation System:* The Building Automation System (BAS) both controls and monitors the facility cooling equipment. Through TRANE, software is utilized to provide visual representations of the cooling systems both for the building and cooling for the supercomputers. This also includes all the recently added cooling configuration for *Trinity*, such as pumps and piping.

This system can also alert the facilities team on pre-determined flags. For example, if a pump unexpectedly turns off, or a flow rate drops below a certain point, facilities can be notified to check it out. The system is also configured to provide failover to the systems. If a pump does fail, a failover procedure is in place to bring another pump online and maintain operation.

A screen capture from the TRANE BAS system is show in Figure 4. This system is impletemented in a web interface in which several of the cooling systems can be viewed and configured. The screen capture shows the new pumps specific to *Trinity*.

*2) Snider Electric:* The newest addition to facilities monitoring infrastructure is the ability to monitor power data from the cluster. In the past tools were utilized to measure power but were limited to minimal readings. Tools from Snider Electric allow for more precise and frequent monitoring of power consumption from *Trinity*. Because *Trinity* is capable of using 15MW of power, knowing its use is crucial to working with the power company to prepare for swings in power consumption.

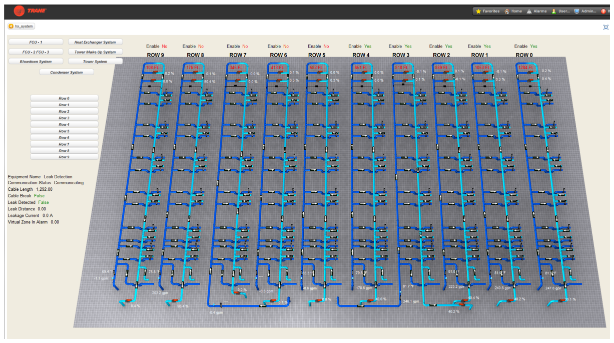


Fig. 4: TRANE BAS screen capture showing the underfloor layout for *Trinity*

*3) Environet:* Environet is a data collection package which utilizes wireless sensors[19] throughout the data center to collect environmental temperatures. The sensors are attached to racks to help determine cooling problems and provide a quick visual reference through temperature maps to locate problems. Traditionally the sensors are attached to both the front and back of racks to show the typical hot aisle and cold aisle differences. With *Trinity's* unique cooling structure there is no longer a hot/cold aisle configuration; however, the sensors still provide good relative room temperatures.

While Environet is not used for alerting, it displays a screen in the operations center that updates in real time to show a heat map of the data center and help operations staff find problems faster. For example if a lot of nodes are reporting hot, and the temperature map shows a hot spot – there could be an issue with cooling that may need immediate attention. The operators can then page the facilities team to resolve the problem.

The data from these sensors is logged as CSV and allows the facilities team to look back at a history of temperatures to see what was normal (or not) over a period of time. It also allows them to look back and determine when an issue may have started as they are able to plot this information over time easily.

Figure 5 shows a screen capture of the Environet web interface. Access to different data centers on site are all accessible from the same web page. The screen capture shows *Trinity* specifically. As clusters are added to the data center, manual reimplementation of the graphics must be completed to provide a visual for the room. While under construction the current shot does not show the heat map that would normally be available. As this graphic is finalized it will show a heat map for the reported sensor temperatures.

## D. Tools and analyses

*1) Baler Log File Analysis:* We are utilizing multiple tools for log file analysis. While tools such as Splunk [6] and Cray's SEC [20] are used for discovering occurrences of log lines matching known patterns of interest, in our operations thus far with Trinity, we have experienced a number of scenarios where our current known patterns are insufficient to capture the full set of scenarios we are interested in. We have added a number of such patterns to our SEC and Splunk rules, for instance with respect to network congestion, thermal throttling, and Lustre issues. We expect this to continue, particularly given the evolving nature of the CLE and Data Warp software.

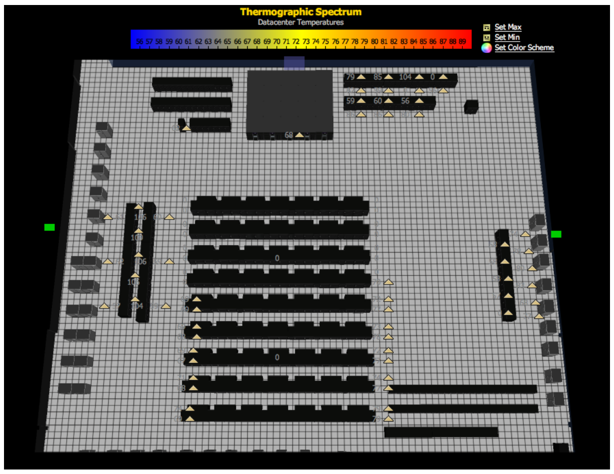


Fig. 5: Environet screen capture showing *Trinity*

In addition, then, we are also deploying the Baler [21] log file analysis tool. The goal of Baler is to extract a greatly reduced number of patterns from the voluminous log data without requiring prior knowledge of the log lines of interest. This is achieved in the following manner: Baler takes as input a "dictionary" of words and creates the pattern from the log line by retaining the dictionary words and treating all other word-like-delimited items as variables. The reduction in data will ease the discovery of new types of events of interest.

Further, by characterizing all log lines as distinct patterns, we can then more easily further investigate associations of numerical resource data from SEDC and LDMS with the log events. For example, on the ART systems, we have used the Baler patterns in conjunction with system state data to examine occurrences of thermal events under different power capping scenarios [18].

Baler runs in parallel on the monitoring cluster, as is shown in Figure 1. The LDMS data from the storage daemons can thus be used in conjunction with the pattern data at this point.

*2) Zenoss:* Zenoss is an open-source event driven monitoring tool capable of real-time notification [5]. Rules and filters are written within Zenoss to utilize syslog from clusters and alert on event. It is well designed for detection of both known problems as well as hardware failures. Zenoss is currently the primary tool for the 24/7 operations staff at LANL for alerting on hardware and other node failures.

Because Zenoss is dependent on syslog feed, some tests for Trinity are injecting lines into syslog to provide PASS/FAIL output for over a dozen 'sanity' checks for the system, including: utilization, number of nodes down, responsiveness of Moab, and several other quick check things. Section IV-A provides more details.

*3) Splunk Log Analysis and Visualizations:* Splunk is both a log collection and log analysis tool. It captures and indexes real-time data in searchable databases [6]. Utilizing multiple indexers a search head can search over a variety of metrics and over multiple sources. Splunk allows correlation of data between clusters and systems allowing cross-platform/system analysis. For example, one could examine I/O bandwidth by observing cluster compute nodes, filesystem servers, and even the network switches between them. It provides a full picture of the entire system by providing access to correlate all the components making up the entire system.

*4) RabbitMQ:* RabbitMQ is a message broker system based on the Advanced Message Queueing Protocol (AMQP)[22]. This producer/consumer model allows system data to be captured in messaging queues which are uniquely keyed, allowing multiple consumers to subscribe to a particular queue for consumption of data. Having multiple instances of RabbitMQ provides failover and redundancy, preventing data loss.

For Trinity, all syslog data will be forwarded to RabbitMQ. The feed will then be dispersed appropriately to Splunk and other desired collectors of the feed.

## *E. Implementation of monitoring cluster*

The *Trinity* monitoring cluster will be expected to scale to collect a large amount of data, both during the initial Phase One deployment, as well as during the Phase Two deployment when the system will roughly double in size. In addition, we cannot know at this time what new types or feeds of data we will find useful for monitoring throughout the lifetime of the system, and there is a strong possibility that our requirements for collecting and processing monitoring data will change drastically over the next few years. We also expect to add additional real-time consumers of our monitoring data, other than the existing Zenoss and Splunk consumers, and will need to be able to easily deploy and manage servers for those consumers.

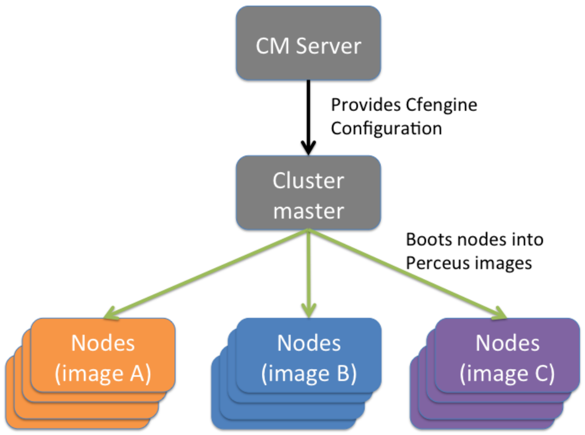


Fig. 6: The CM server provides the Cfengine configuration information and scripts to the cluster master, which builds boot images and boots the nodes using Perceus

Based on these needs, we decided to deploy as much of our monitoring infrastructure as possible using the same cluster deployment tools we use to manage our commodity HPC platforms. In particular, our stack uses a heavily-customized version of Perceus [23] as a provisioning tool, with tight integration into our Cfengine [24] configuration management system. All Perceus images are generated directly from our source OS package repositories and files managed by Cfengine. This stack provides a flexible way for us to provision OS images across the monitoring cluster, supporting images of multiple types and potentially scaling as high as hundreds of nodes. It also gives us a familiar stack which is very well-understood at LANL, minimizing the additional complexity we might have to add to the system.

As currently deployed, the monitoring cluster includes the following components:

- The CM server, which acts as the "source of truth" for the Cfengine configuration, which is synced periodically to the cluster master
- The cluster master, which provisions OS images to the other nodes and provides basic services such as DNS and NTP
- The collector nodes, which accept data feeds from *Trinity* and its external services, and perform some processing of the data
- The RabbitMQ nodes, which provide a message bus for distributing monitoring data to several types of consumer
- The "mon box", which runs Zenoss and acts as the primary monitoring point for our operations staff
- The Splunk indexers, which provide a searchable interface to monitoring data for HPC staff

Each set of nodes (except the master) can be easily scaled through deploying additional hardware running the same Perceus image. Additional components, such as new data consumers, can be deployed using the same infrastructure by building a new Perceus image with the necessary software included.

## *F. LANL HPC shared monitoring infrastructure*

The existing monitoring infrastructure at LANL utilizes both Splunk and Zenoss, with integration of RabbitMQ to reduce the number of feeds each cluster must send out. The current design is based on a per cluster basis outlined in Figure 7. Each cluster has its own designated monitoring box, referred to as a "mon-box". The mon-box runs an instance of Zenoss and an instance of Splunk. The cluster sends a direct syslog feed to the Zenoss collector on the mon-box and another directly to a designated logger box, which collects all raw syslog data from additional clusters on the network. A final third stream from the cluster sends to a RabbitMQ which feeds the Splunk indexer on the clusters associated mon-box. The following subsections detail the current use of these components.

*1) Zenoss:* Zenoss is currently the primary tool for the 24/7 operations staff at LANL for altering on hardware and node failures. Since the data from Zenoss is live, operators can see failed nodes, over temped CPUs and network issues as soon as the occur. Figure 8 shows parts of the Zenoss "GRID", where operators can get a first glance at cluster health.

The "GRID" uses a color scheme to show the state of a cluster as well as a percentage for cluster utilization. In Figure

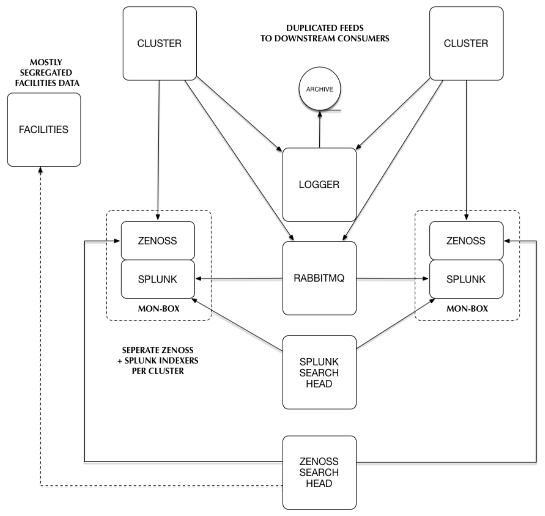
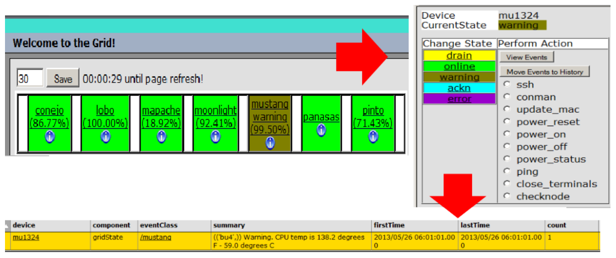


Fig. 7: Current Monitoring Infrastructure

8 it is observed that the cluster *Mustang* is at 99.5% utilization and has nodes in the "warning" state. Additional investigation into the cluster provides more detail to the "warning" as being a CPU temperature reported above the defined thermal threshold. This warning results from filtering within Zenoss that pulled the CPU temperature from the clusters syslog. It has the ability to record multiple occurrences of this event and provide a count for the number of occurrences of the event as well as both the time and date of the first occurrence, and most recent occurrence of the event [5]. This helps provided correlation between events to help track down the root cause.



| device | component | eventClass | summary | firstTime | lastTime | count |
|---|---|---|---|---|---|---|
| mu1324 | gridState | /mustang | (('bu4',)) Warning. CPU temp is 138.2 degrees F - 59.0 degrees C | 2013/05/26 06:01:01.000 | 2013/05/26 06:01:01.000 | 1 |

Fig. 8: Zenoss GRID flow

This sort of event driven alerting allows operators to react immediately and determine root cause and prevent total cluster failure due to something such as facility problems without direct facility data input.

*2) Splunk:* Each "mon-box" instance of Splunk acts as a Splunk indexer. A main Splunk search head provides access to each of these indexers as a collective search over all indexers. The particular search can be limited to a specific cluster, or expansive to compare data between clusters.

Splunk allows for more of a continuous monitoring mechanism with visual support through dashboards [6]. Several dashboards have been developed between the different teams (filesystems, storage, architecture) to monitor changes over time. This data can often show changes in performance for a cluster after a change has been made (for the better or worse) as well as show differences from the norm.

While Splunk has real-time data and can provide alerts based on searches it is primarily used for correlating data and creating reports and visualizations to better understand a failure or change.

Figure 9 shows part of a dashboard for tracking the state of Moab and Torque on Trinity. This dashboard provides an at-a-glance view of the state of the system from the perspective of job scheduling and resource management. The current count of compute nodes in each Torque state are shown, as well as area plots of node and job state from the view of both Torque and Moab.

*3) RabbitMQ:* RabbitMQ is currently being utilized to aggregate cluster data and send it to each clusters respective "mon-box" Splunk indexer. It allows multiple feeds to co-locate and be pushed to appropriate consumers.

RabbitMQ clustering [25] is being tested to reduce the needed feeds from each cluster as well as provide a more centralized location for data collection and redirection. The growth of clusters and their abilities to provide more measurable values, including new metrics from Trinity, are pushing for a redesign of the monitoring infrastructure, besides just the "mon-box".

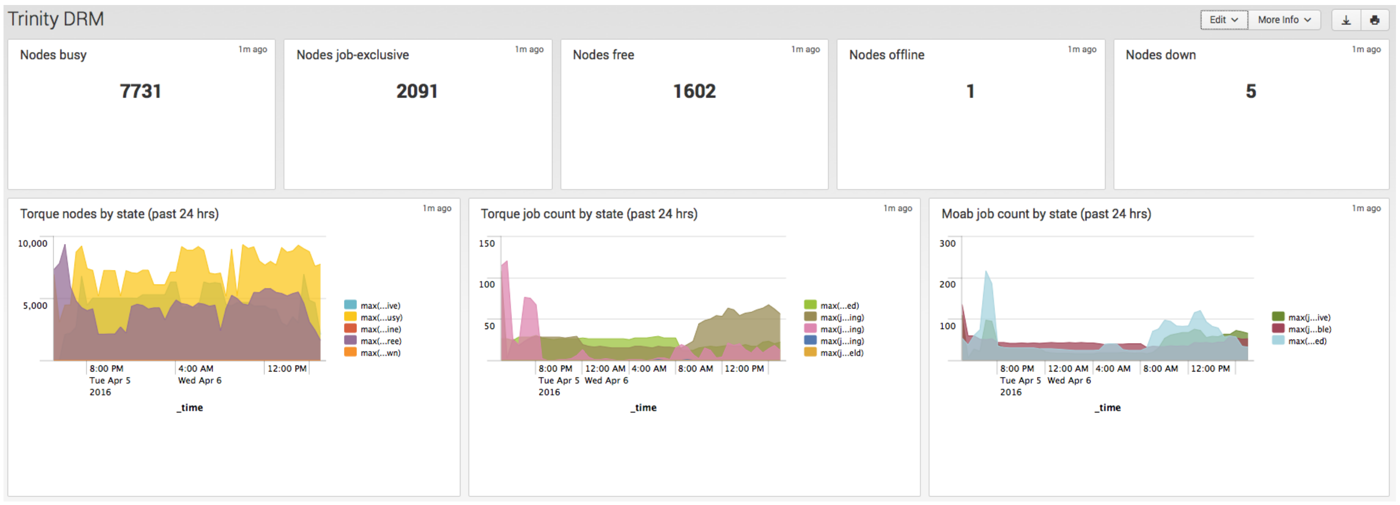
Trinity will be one of the first clusters to utilize the RabbitMQ configuration in the sense that single syslog feed will go to RabbitMQ rather than to RabbitMQ, logger, and Zenoss. Then the message broker can provide the appropriate feeds to the consumers wanting that feed (Splunk, Zenoss, other). This reduces efforts from sending multiple feeds to multiple locations and facilitates the addition and removal of consumers as needed without affecting other traffic.

## III. Overhead of In-Band Monitoring

In order to assess the overhead of the in-band monitoring, Trinity testers ran the following codes at large scale, with and without monitoring:

*High Performance Conjugate Gradient: HPCG* HPCG [26] consists of operations such as sparse matrix-vector products. It is expected to stress the memory subsystem and network communications. Three 10 minute runs of 9293 nodes, with 18568 processes with 16 threads each were run, both in baseline and with 1 second monitoring conditions. The metrics of comparison were the Benchmark Total Time and the All Reduce times. Results are shown in Table I. No significant variation was found. Benchmark average times with monitoring are actually 0.7% lower than the baseline average and the All Reduce times are comparable.

*PARTISN* PARTISN [27] (PARallel, TIme-dependent SN) is a deterministic neutral particle transport code and is important code in the LANL workload. Seven runs of a PARTISN problem on 8192 nodes with 32 ranks per node were run: 3 runs were with monitoring at 1 sec and 4 runs were under baseline conditions. Each run had 5 cycles. The metric used for comparison was cycle times. Results are shown in Table II. No significant impact was found. The percent difference in the


Trinity DRM

Nodes busy 7731

Nodes job-exclusive 2091

Nodes free 1602

Nodes offline 1

Nodes down 5

Fig. 9: Splunk Trinity job scheduling dashboard

| Condition | Run | Benchmark Time (sec) | All Reduce (AR) Min | AR Max | AR Avg |
|---|---|---|---|---|---|
| Baseline | 1 | 603.81 | 19.63 | 79.12 | 74.65 |
| Baseline | 2 | 595.58 | 19.55 | 78.73 | 73.87 |
| Baseline | 3 | 603.59 | 20.64 | 75.25 | 70.85 |
| w/Monitoring | 1 | 598.55 | 22.17 | 78.01 | 72.86 |
| w/Monitoring | 2 | 594.84 | 19.35 | 77.24 | 72.84 |
| w/Monitoring | 3 | 596.51 | 20.16 | 75.84 | 71.07 |

TABLE I: HPCG impact test results. No significant variation.

average run time per cycle between the baseline and LDMS was 0.08% or less.

| Condition | Cycle | Average Time (sec) | Min Time | Max Time |
|---|---|---|---|---|
| Baseline | 1 | 50.53 | 50.41 | 50.66 |
| Baseline | 2 | 38.44 | 38.37 | 38.48 |
| Baseline | 3 | 38.26 | 38.22 | 38.32 |
| Baseline | 4 | 37.28 | 37.22 | 37.34 |
| Baseline | 5 | 37.33 | 37.30 | 37.36 |
| w/Monitoring | 1 | 50.42 | 50.29 | 50.49 |
| w/Monitoring | 2 | 38.47 | 38.45 | 38.49 |
| w/Monitoring | 3 | 38.29 | 38.26 | 38.31 |
| w/Monitoring | 4 | 37.28 | 37.26 | 37.32 |
| w/Monitoring | 5 | 37.36 | 37.34 | 37.38 |

TABLE II: PARTISN impact test results. No significant variation.

*Sierra Low Mach Module: Nalu* Nalu [28] is an adaptive-mesh, variable-density, acoustically incompressible, unstructured fluid dynamics code that supports energy applications of interest. Nalu is built atop Sandia National Laboratories' Sierra Toolkit and Trilinos solver's Tpetra/Epetra stack. For this test, under both baseline and monitoring conditions, we ran 3 concurrent instances of a 65k-core, time-accurate simulation initialized from checkpoint/restart files; this simulation was also performed as part of an Open Science campaign on Trinity. Due to time limitations, however, we ran a limited set of timesteps. The metric used was the code-reported wall time. These metrics are provided in Table III and exhibit no significant variation. The average run time with monitoring had a 0.3% increase, however all monitoring run times were within the variability exhibited from the baseline cases, i.e., the minimum and maximum baseline run times.

| Condition | Run | Walltime (min) |
|---|---|---|
| Baseline | 1 | 8.34 |
| Baseline | 2 | 8.26 |
| Baseline | 3 | 8.40 |
| w/Monitoring | 1 | 8.37 |
| w/Monitoring | 2 | 8.33 |
| w/Monitoring | 3 | 8.35 |

TABLE III: Nalu impact test results. No significant variation.

*PSNAP* PSNAP [29] is used to measure OS jitter. 3 runs were performed under baseline and monitoring conditions. We ran 100000 loops of 1000 microsecs with and without a barrier every 100 loops. The runs were performed on 9216 cores, 32 cores per node. The metric of comparison is the change in slowdown, where slowdown is the actual loop times as compared to the ideal loop time. Results are shown in Table IV. With 1 sec monitoring, the slowdown increased by $< 0.02\%$ above the baseline slowdown, which we deemed acceptable.

| Condition | Run | Per node avg slowdown | min slowdown | max slowdown |
|---|---|---|---|---|
| Baseline no barrier | 1 | 0.217 +/- 0.012 | 0.200 | 0.243 |
| Baseline no barrier | 2 | 0.217 +/- 0.011 | 0.199 | 0.245 |
| Baseline no barrier | 3 | 0.217 +/- 0.011 | 0.199 | 0.247 |
| w/Monitoring no barrier | 1 | 0.237 +/- 0.009 | 0.209 | 0.253 |
| w/Monitoring no barrier | 2 | 0.237 +/- 0.009 | 0.205 | 0.254 |
| w/Monitoring no barrier | 3 | 0.238 +/- 0.009 | 0.209 | 0.253 |
| Baseline w/barrier | 1 | 0.226 +/- 0.012 | 0.204 | 0.249 |
| Baseline w/barrier | 2 | 0.226 +/- 0.012 | 0.202 | 0.251 |
| Baseline w/barrier | 3 | 0.223 +/- 0.011 | 0.204 | 0.248 |
| w/Monitoring w/barrier | 1 | 0.238 +/- 0.011 | 0.207 | 0.259 |
| w/Monitoring w/barrier | 2 | 0.238 +/- 0.011 | 0.212 | 0.259 |
| w/Monitoring w/barrier | 3 | 0.231 +/- 0.011 | 0.209 | 0.271 |

TABLE IV: PSNAP impact test results. With 1 sec monitoring, the slowdown increased by $< 0.02\%$ above the baseline slowdown, which we deemed acceptable.

## IV. Development of Monitoring in Production

The *Trinity* deployment is still in its early stages, with Phase One of the machine completing its initial "Open Science" campaign of user jobs as this paper was being completed. While the overall design and implementation of the monitoring cluster and its supporting infrastructure is reasonably well-determined, our day-to-day tools and processes for keeping track of the system's state are still in active development. In this section, we will present the evolving state of tools and practices used by our production teams in monitoring the system.

## A. Health checks for service components

```
PASS: kernel version matches expected value
PASS: hostname matches expected value
PASS: autofs active and enabled
PASS: rsyslog active and enabled
PASS: sshd active and enabled
PASS: df -h works on /lustre/scratch5
PASS: lfs df returns within five seconds
PASS: local filesystems are mounted
PASS: lustre mounted
PASS: ls -l works on /users/testuser
PASS: ls -l works on /usr/projects/testproject
PASS: pbsnodes returns
FAIL: pbsnodes has 9337 nodes in trinity
FAIL: mdiag shows tr-drm as the moab server
FAIL: showq –blocking returns
FAIL: showq returns
FAIL: showres returns
FAIL: showstate returns
12 TESTS SUCCEEDED and 6 TESTS FAILED
```

Fig. 10: Example output from a "health check" run on a *Trinity* front-end (eLogin) node.

While all of the HPC platforms at LANL employ health checks to verify compute node functionality at the beginning or end of each job, we make little use of active checks on internal or external service nodes which provide supporting infrastructure for compute jobs. On our existing platforms, this has not represented a major gap in monitoring, as these systems are considered well-understood and most issues can be easily caught by filters on logs generated by the relevant software services. However, in our limited time running *Trinity* in production, we have already encountered several issues with new functionality which are not easy to catch through simple log filters, including issues relating to the new Ansible configuration management system and the Cray DataWarp burst buffers.

For this reason, we are working to deploy a comprehensive set of health checks to report on the status of various hardware and software services. Our health checks are intended to complement the Cray Node Health Check (NHC)[30] scripts which run on the compute nodes, and are designed to run from *cron* on any node where user-facing performance considerations do not preclude a periodic check. Our most frequently-run tier of checks, which will run every ten minutes, are designed to execute as quickly as possible while still checking for the most common or most problematic issues we have seen in production. Checks in this level include using *systemctl* to check the status of required services; verifying that user-facing commands which are expected to complete quickly will do so (e.g., Moab's *showq*); and verifying that filesystem mounts are present. On some nodes, such as our user-facing front-end nodes, we also plan to deploy a set of heavier weight checks which would run less frequently, i.e. on an hourly or even daily basis. Checks in this level may include verifying that job submission works as expected or verifying that simple software builds complete successfully.

In our initial implementation, each health check script takes the form of a Python program which defines a set of tests, and uses the *unittest* module as a framework for test execution. The status of each test is logged individually in syslog as a "pass", "warn", or "fail" so that these tests can be parsed by monitoring tools such as Zenoss or Splunk and used to generate alerts or populate a dashboard. An example of logged test statuses for a front-end node is shown in Figure 10.

## B. Dashboards and visualization

We are developing several different tiers of visualization to assist us in monitoring the system in production. These visualizations will be used by several different HPC teams at LANL, including our operations staff, Platforms system administration, networking, and filesystems teams, as well as our Cray site personnel.

Our primary operational visualization for *Trinity*, as for our other systems, will be a Zenoss "GRID" similar to Figure 8. This dashboard will provide a quick "at-a-glance" view of the state of the system, helping to identify operational issues. Problems that would cause dashboard indicators to turn red include critical infrastructure hardware going offline (such as the SMW), failure of compute nodes or redundant services over a certain threshold, or the failure of some of the health checks from Section IV-A.

In addition to the Zenoss GRID, we plan to construct several more specialized dashboards using Splunk to visualize specific types of data. While Splunk dashboards update more slowly than the GRID, as they must re-run searches of the monitoring data to update each panel, they provide richer visualization options and can be more easily altered on the fly to suit the needs of day-to-day monitoring. Specific teams will construct dashboards most relevant to their own specific tasks, such as dashboards specific to monitoring networking, filesystem state, or the state of the resource manager and job scheduler. An example dashboard for viewing node and job status with respect to job scheduling is shown in Figure 9.

## C. Data Driven Operations

Our monitoring design seeks to enable easier run-time integration of data sources in order to improve operational decisions. We are working on supporting both numerical and text data sources in our tools to enable correlations of events with numerical state data. Of particular interest is determining relationships of Aries traffic related data and Lustre data with performance issues and failure data in the logs.

We are working on architectural design to enable user visualization of the in-band data. This will allow users to have greater insight into their application resource demands in order to detect imbalance and sub-optimal use of resources. We have seen that system monitoring data at the time resolutions we are are targeting here can give insight into application resource utilization under production conditions [31]. This is as opposed to typical performance analysis tools which provide higher fidelity information, but with potential limits of scale and without insight into the entire system state.

Ultimately, we seek to use increased understanding of the system workload and system characteristics to improve resource utilization within Trinity and within the data center as a whole. Our initial work in monitoring the ART systems [18], [32] has shown thermal distributions within the system and possible performance distributions among components that perhaps should be taken into consideration in application placement, such as placing the most computationally intensive jobs on processors at the coolest locations or that are the more highly performing. Greater understanding of network congestion in combination with understanding of applications' communication patterns can potentially be used to enable better application placement via the scheduler. Finally, the data center as whole is subject to power and cooling constraints that affect all platforms housed within. We seek to gain greater understanding of the platform's power demands under significant applications and workloads so that we can more optimally schedule load and power cap to respond to these constraints.

## V. Conclusions

In this paper, we have described our motivation in designing and deploying a next-generation monitoring system for *Trinity*, both due to its scale and the added complexity of this system relative to previous HPC platforms deployed at LANL. We present the architecture and implementation of a monitoring cluster which collects and aggregates several different types of data feed, and distributes it for further processing and analysis to a larger monitoring infrastructure shared with the other HPC platforms and infrastructure at LANL. We also present an early analysis of the overhead and application impact imposed by our monitoring.

## VI. Future work

While we have accomplished our initial deployment of a next-generation monitoring system for *Trinity*, there are several areas in which our system still needs improvement for functionality and resilience.

First, while we have architected for redundancy in several components of our monitoring infrastructure, the SMW still stands out as a single point of failure for the system. The vast majority of logging data, as well as SEDC metrics, are collected only at the SMW and must be forwarded from there to our monitoring cluster. The SMW has limited network connectivity, with most network ports already dedicated to specific functionality required to run the system, which has constrained us to forwarding monitoring data using the gigabit Ethernet link used for the site network. Any new logging or monitoring which uses syslog adds additional load to the SMW, which already performs a large number of functions for the HPC platform, and constrains the available network bandwidth. To mitigate this, we are working with Cray to explore the idea of adding an additional log host to the system which could take over SEDC data collection, collection of system log data, or both.

Second, while syslog is a convenient transport for collecting most forms of data, it does not provide any good built-in mechanisms for high availability. The naive way to provide "HA" for syslog is simply to forward the same data stream to more than one log server; but the constraints on network bandwidth at the SMW make us hesitant to double or triple the load of log forwarding by pointing it at more than one member of the monitoring cluster. We are currently exploring tools such as *pacemaker* in order to provide HA on a network level, by forwarding logs from the SMW to a single IP which can be failed over between multiple members of the monitoring cluster.

Finally, while we have built an infrastructure for collecting and distributing log data through the shared RabbitMQ infrastructure, we have yet to deploy tools for processing or analyzing this data stream apart from our legacy Zenoss and Splunk infrastructure. We still need to take advantage of this message bus architecture to add additional analysis tools, both for our own systems-level analysis, and for making selected data available to our users in a secure fashion. This will hopefully include an automated way to make LDMS data available on a per-job basis to the owners of those jobs.

## VII. Acknowledgments

The process of designing and deploying the *Trinity* monitoring system has been deeply entwined with the process of deploying *Trinity* itself. This work could not have been accomplished without the assistance and participation of most of the members of the HPC division at LANL, the developers of LDMS at SNL and Open Grid Computing, our Cray support personnel, and many of the users of Trinity during its Open Science campaign. In particular, we would like to thank Vaughn Clinton, Susan Coulter, Gregory Geller, Daryl Grunau, David Morton, Paul Peltz, Conor Robinson, Quellyn Snead, Kevin Stroup, Graham Van Huele, and Jim Williams for their help and their patience as we deployed, tested, and debugged the systems described in this paper. We would also like to thank Scott Pakin and Hai Ah Nam for useful discussions on overhead testing.

## References

[1] B. Alverson, E. Froese, L. Kaplan, and D. Roweth, "Cray XC Series Network," WP-Aries01-1112, 2012. [Online]. Available: http://www.cray.com/sites/default/files/resources/CrayXCNetwork.pdf

[2] Ansible, Inc. (2015) Ansible (Version 1.9.2). [Online]. Available: https://www.ansible.com

[3] A. DeConinck and K. Kelly, "Evolution of Monitoring over the Lifetime of a High Performance Computing Cluster," in *Wrk. on Monitoring and Analysis for High Performance Computing Systems Plus Applications (HPCMASPA) Proc. IEEE Int'l Conf. on Cluster Computing (CLUSTER)*, 2015.

[4] S. Sanchez, A. Bonnie, G. Van Huele, C. Robinson, A. DeConinck, K. Kelly, Q. Snead, and J. Brandt, "Design and Implementation of a Scalable HPC Monitoring System," in *Wrk. on Monitoring and Analysis for High Performance Computing Systems Plus Applications (HPCMASPA) Proc. IEEE Int'l Conf. on Cluster Computing (CLUSTER)*, 2016.

[5] (2016) Zenoss (Version 2.1.3, with LANL customizations). [Online]. Available: http://www.zenoss.org/

[6] Splunk. Inc. (2014) Splunk (Version 5.0.12). [Online]. Available: https://www.splunk.com

[7] Adaptive Computing. (2016) TORQUE. [Online]. Available: http://www.adaptivecomputing.com/products/open-source/torque/

[8] ——. (2016) Moab HPC Suite (Version 8.1.2). [Online]. Available: http://www.adaptivecomputing.com/products/hpc-products/moab-hpc-suite-enterprise-edition/

[9] Cray Inc., "Using and Configuring System Environment Data Collections (SEDC)," Cray Doc S-2491-7001, 2012.

[10] A. Agelastos, B. Allan, J. Brandt, P. Cassella, J. Enos, J. Fullop, A. Gentile, S. Monk, N. Naksinehaboon, J. Ogden, M. Rajan, M. Showerman, J. Stevenson, N. Taerat, and T. Tucker, "Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications," in *Proc. Int'l Conf. for High Performance Storage, Networking, and Analysis (SC)*, 2014.

[11] "Blue Waters." [Online]. Available: https://bluewaters.ncsa.illinois.edu

[12] Cray Inc., "Aries Hardware Counters," Cray Doc S-0045-20, 2015.

[13] ——, "Using the Cray Gemini Hardware Counters," Cray Doc S-0025-10, 2010.

[14] ——, "Managing System Software for the Cray Linux Environment," Cray Doc S-2393-5202axx, 2014.

[15] K. Pedretti, C. Vaughan, R. Barrett, K. Devine, and S. Hemmert, "Using the Cray Gemini Performance Counters," in *Proc. Cray User's Group*, 2013.

[16] J. Brandt *et al.*, "Network Performance Counter Monitoring and Analysis on the Cray XC Platform," in *Proc. Cray User's Group*, 2016.

[17] Cray Inc., "Monitoring and Managing Power Consumption on the Cray XC System," Cray Doc S-0043-7204, 2015.

[18] J. Brandt, D. DeBonis, A. Gentile, J. Lujan, C. Martin, D. Martinez, S. Olivier, K. Pedretti, N. Taerat, and R. Velarde, "Enabling Advanced Operational Analysis Through Multi-Subsystem Data Integration on Trinity," in *Proc. Cray User's Group*, 2015.

[19] GEIST. (2016) Wireless Sensors (ENV-DIT). [Online]. Available: http://www.geistglobal.com/productsmonitor/wireless-sensors

[20] Cray Inc., "Configure Cray SEC Software," Cray Doc S-2542-7204, 2015.

[21] N. Taerat, J. Brandt, A. Gentile, M. Wong, and C. Leangsuksun, "Baler: deterministic, lossless log message clustering tool," *Computer Science - Research and Development*, vol. 26, no. 3-4, pp. 285–295, 2011.

[22] RabbitMQ, Pivotal. (2016) What is RabbitMQ? [Online]. Available: https://www.rabbitmq.com/

[23] Infiscale. PERCEUS: Provision Enterprise Resources and Clusters Enabling Uniform Systems. [Online]. Available: https://web.archive.org/web/20151119185014/http://perceus.org/

[24] CFEngine. cfengine (Version 2.2.10). [Online]. Available: https://cfengine.com

[25] RabbitMQ, Pivotal. (2016) Clustering Guide. [Online]. Available: https://www.rabbitmq.com/clustring.html

[26] "HPCG." [Online]. Available: http://www.hpcg-benchmark.org

[27] R. E. Alcouffe, R. S. Baker, J. A. Dahl, S. A. Turner, and R. Ward, "Partisn: A time-dependent, parallel neutral particle transport code system," *Los Alamos National Laboratory, LA-UR-05-3925 (May 2005)*, 2005.

[28] Sandia National Laboratories, "spdomin/Nalu," https://github.com/spdomin/Nalu.

[29] "PAL System Noise Activity Program," Los Alamos National Laboratory Performance and Architecture Laboratory (PAL), March 2014. [Online]. Available: http://www.c3.lanl.gov/pal/

[30] Cray Inc., "Manage System Software for Cray Linux Environment," Cray Doc S-2393-5202, 2014.

[31] A. Agelastos, B. Allan, J. Brandt, A. Gentile, S. Lefantzi, S. Monk, J. Ogden, M. Rajan, and J. Stevenson, "Toward Rapid Understanding of Production HPC Applications and Systems," in *Proc. IEEE Int'l Conf. on Cluster Computing (CLUSTER)*, 2015.

[32] J. Brandt, A. Gentile, C. Martin, J. Repik, and N. Taerat, "New Systems, New Behaviors, New Patterns: Monitoring Insights from System Standup," in *Wrk. on Monitoring and Analysis for High Performance Computing Systems Plus Applications (HPCMASPA) Proc. IEEE Int'l Conf. on Cluster Computing (CLUSTER)*, 2015.