Runtime collection and analysis of system metrics for production monitoring of *Trinity Phase II*

A. DeConinck, H. Nam, D. Morton, A. Bonnie, and C. Lueninghoener
Los Alamos National Laboratory, Los Alamos, NM

J. Brandt, A. Gentile, K. Pedretti, A. Agelastos, C. Vaughan, S. Hammond, and B. Allan
Sandia National Laboratories, Albuquerque, NM

M. Davis and J. Repik
Cray, Inc., Albuquerque, NM

***Abstract*—We present the holistic approach taken by the ACES team in the design and implementation of a monitoring system tailored to the new Cray XC40 KNL based *Trinity Phase II* system currently being deployed in an Open Science campaign. We have created a unique dataset from controlled experiments to which we apply various numerical analyses and visualizations in order to determine actionable monitoring data combinations that we can associate with performance impact and system issues. Our ultimate goal is to perform run-time analysis of such data combinations and apply runtime feedback to users and system software in order to improve application performance and system efficiency.**

I. Introduction

In previous HPC community meetings (e.g., SC15, SC16, CUG 2016, Cray System Monitoring Working Group), the need for monitoring versus the current state of monitoring has been discussed. The community generally agreed that the single biggest question asked by users is *Why does my application performance vary so much?* The foremost needs from monitoring in support of answering this question were seen to be understanding of the performance, utilization, and congestion characteristics of the High Speed Network (HSN) and IO. However, many sites stated that they were not monitoring because they didn't know what they would look for in the data or even how to approach figuring it out.

In this work, we have created a unique dataset from controlled experiments to which we apply various numerical analyses and visualizations in order to determine actionable metrics that we can associate with performance impact and system issues. While this is still work in progress, we present the current state of our research in order to provide information to the community on what we are trying to learn, what we are analyzing and how we are analyzing it, how we are enabling analysis, and what else we need to make further progress.

ACES (LANL/SNL) Trinity Phase II is a $9,984$ node, $678,912$ core Cray XC40 platform with Intel Xeon Phi 7250 processors (code-named "Knights Landing", or KNL), as well as 346 service nodes performing system functions, and a Cray Aries high-speed network (HSN) as interconnect. TR2 is currently deployed in a stand-alone configuration for "Open Science", a deployment phase in which selected high-impact projects can be given dedicated runtime on the full machine before it is moved to its production environment. When the *Trinity* deployment is complete, this machine will be combined with *Trinity Phase I* (TR1), which consists of ∼9,000 Intel Xeon E5-2698 v3 ("Haswell") processors and additional service nodes, into a single system.

Open Science also provides an opportunity to better understand the behavior of a large KNL machine undergoing high utilization, as well as developing tools and processes for running this type of machine in production. In addition to the more general exploration for understanding application performance variation, we have some specific circumstances motivating our monitoring. TR2 is one of the first large scale deployments of the KNL processor architecture for regular HPC workloads. With the introduction of multiple cluster and memory modes available on the KNL, there are new complexities to be considered in determining the best configuration of the KNL for a particular application. In addition, the scale of TR2 and the eventual full *Trinity* deployment provides a strong motivation for measuring energy usage. Correlation of energy usage to system conditions, such as the application mix or network congestion, could lead to useful decisions regarding the management of the system. Measurement of energy usage and its variation during real workloads will enable us to evaluate the potential benefits of incorporating power data into job scheduling decisions.

We are developing and deploying an architecture for whole-system monitoring. Data sources consist of log, out-of-band, and in-band node level data. Analysis and storage targets include a monitoring and analysis cluster hosting a short-term active working dataset and storage with long-term data stored in an archive that supports retrieval. Additional downstream entities will also be consumers of the analysis results.

To reduce the time to understanding, we support streaming analysis at the monitoring cluster before insertion into a database and other storage. Currently we target functional forms of data, such as rates, aggregations, and ratios. We can integrate numeric out-of-band, numeric in-band, and log data with our analysis tools. We are working toward enabling streaming computations and transformations of numeric data at arbitrary points within the monitoring infrastructures. We use third party tools for time-series numerical and log analysis and visualization and we augment these with domain-specific

This document is approved for release under LA-UR-17-24148.

Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

analysis and visualizations.

This paper is structured as follows. In Section II we describe the implementation of our monitoring system. In Section III we present controlled experiments and resulting observations. In Section IV we demonstrate some of our current approaches to analysis of the controlled experiments (still work in progress). In Section V we present some assessments of in-band monitoring impacts on application performance. Finally, we conclude in Section VI with some observations of outstanding questions in HPC monitoring.

II. Implementation of monitoring system on TR2

A. Overview

In [1], we described a scalable monitoring system developed for gathering and analyzing monitoring data from TR1 during its Open Science phase. This monitoring system combined the collection of standard, Cray-provided system logs and environmental metrics ("System Environment Data Collection", or SEDC) with additional metrics collected from each compute node using an on-node daemon provided by the Lightweight Distributed Metric Service (LDMS) [2]. LDMS data was transported off the TR1 system via a set of dedicated aggregator nodes, and from there to a dedicated monitoring cluster which was used for runtime analysis. The regular syslog and SEDC streams were transported to the same monitoring cluster for analysis and correlation with the LDMS data, as well as being forwarded to shared site-level monitoring tools which provide alerting and search capabilities for multiple HPC systems.

The TR2 monitoring system for the Open Science campaign, shown in Figure 1, has similarities to the system implemented for TR1 [1], but includes several notable changes. Additional capabilities for power monitoring have been added, and some components of the system have been simplified to take into account both past production experience and the limited duration of the Open Science campaign. In this section, we describe in detail the implementation of several components of the TR2 monitoring system, including the hardware and software deployed, its configuration, and the flow of data through the system. Where a component is not described in detail, it should be assumed to be the same as in [1].

B. Power and environmental data

Cray's out-of-band system monitoring infrastructure measures per-node and per-cabinet power usage and collects this information at 1 Hz granularity to a Power Management Database (PMDB), which is also configured to store SEDC information. For this data, we have deployed a dedicated server which is referred to as the "PMDB node". This server has a connection to Cray Hardware Supervisory System (HSS) network and runs an Event Router Daemon (ERD) endpoint, allowing it to collect out-of-band monitoring data from TR2 independently of the SMW. This functionality is new in our architecture and was introduced in CLE 6.0 UP01. The PMDB node also runs its own instance of the Power Management Database software (hence the name), and collects both power management data and SEDC data via the ERD endpoint.

The PMDB node is a Dell PowerEdge R720 server, with a single-socket Intel Xeon E5-2650 v2 processor (8 cores) and 256GB RAM. Approximately 1TB of local storage is provided (in RAID 1) for storage of monitoring data. The PMDB node runs Cray ERD software which allows it to monitor data which is broadcast on the HSS network. It polls for and stores power management data and SEDC data to a local Postgres database. Due to the limited storage on this node, it rolls this data off at a configurable interval to make room as new data arrives. Under typical usage on TR2, the PMDB holds a window of approximately 11 hours for power and SEDC data provided by the blade controllers, and approximately 2 days for data provided by cabinet controllers. ALPS job data is also stored on the PMDB, but this data represents an extremely low storage burden, such that we have configured this data to be rolled off less frequency than once a month.

C. Node Level Monitoring: LDMS

The Lightweight Distributed Metric Service (LDMS) [2] is used to collect data in-band from system components. The configuration for LDMS is similar to that investigated for Trinity Phase 1 [1]. LDMS daemons are run on, and collect data from, both compute and service nodes with only the most recent data being held on the node in set memory. Data is pulled via RDMA over the Aries network to aggregation hosts in the system. Daemons running on the monitoring cluster hosts pull the data from the aggregation nodes over socket. This is shown in Figure 1.

Collection can occur for any exposed data. We collected the same data for this work as for Trinity phase 1. The Aries network counters [3] are exposed via the *gpcd* [4] interface and account for approximately 850 metrics. Power data is exposed via the `/sys` filesystem; this accounts for 2 metrics. Note that while we collect the power data at 10Hz, we transport it at the same 1Hz frequency as we do for the rest of the data. We additionally collect, within a single LDMS data set, current free and active memory; CPU utilization information; and client side information about accesses to the shared parallel Lustre file system such as opens, closes, reads, and writes. This data is collected largely from *procfs* and accounts for approximately 85 metrics. One could opt to minimize contention for node level resources on the KNL of LDMS with some applications (e.g., those using powers of two node cores) by binding the LDMS processes to an unused core. This option was not pursued during our testing as some of the applications would not fall into this category and we did not want to add another dimension to the configuration permutations that were evaluated.

The major change in LDMS configuration in this work, over our previous configuration [5], is improved redundancy for collection of the Aries router counters. Since all Aries router counters are exposed on all nodes sharing the Aries routers, we collect with two-way redundancy from compute nodes, which are four to a router, and full redundancy from service nodes, which are two to a router. We have augmented our network analysis to include assessments of backpressure at the NIC, of injection into the HSN, and of indirect indications of queue depths in the network.

D. Off-platform data aggregation

All data (e.g., system, SEDC, power, log) can be integrated and analyzed at runtime on the monitoring-and-analysis cluster.

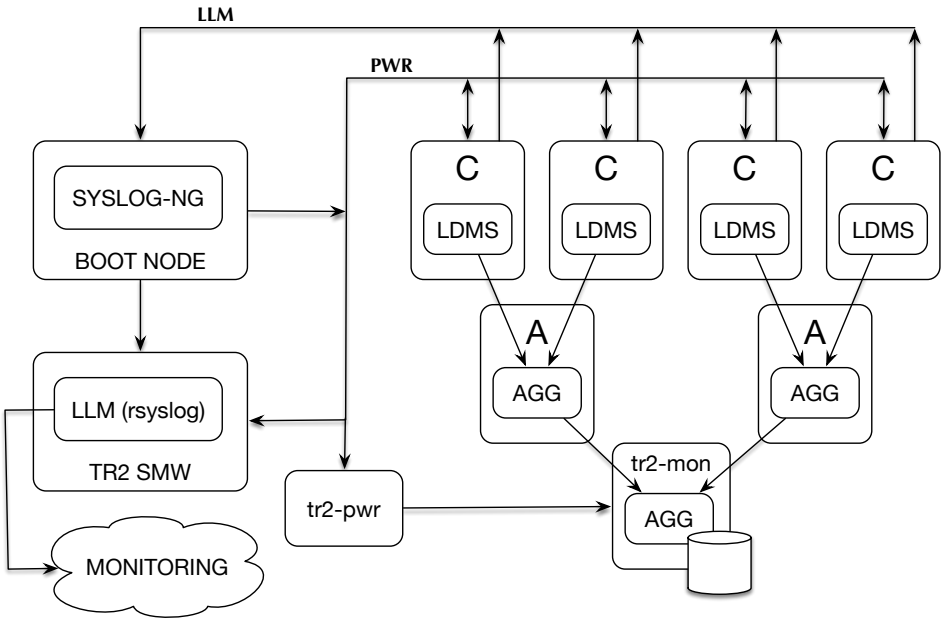


Fig. 1. Flow of monitoring data from TR2. Logging data (LLM) is transmitted from the compute and service nodes over the HSN to the boot node, and from there to the SMW. Power and SEDC data are transmitted via the HSS network to the PMDB node (here labeled tr2-pwr), and then synced to the monitoring resources, tr2-mon (see Section II-D). LDMS data is produced on-node by the sampler daemons, collected by aggregator daemons on two dedicated service nodes, and then aggregated externally on tr2-mon.

During Open Science this consists of one node, but after the Phase 1 and Phase 2 integration this will be expanded to be a small cluster.

To allow the PMDB node to maintain a consistent rate of data ingestion from a large HPC system, we limit the workload on that machine only to performing data ingestion. To enable queries and analysis of power data while the machine is running, we have deployed an additional monitoring node to which we mirror this data, but which is not responsible for ingesting data directly using an ERD endpoint. Throughout this paper, we will generally refer to this as the "monitoring node".

Like the PMDB node, the monitoring node is a Dell PowerEdge R720 server, with a single-socket Intel Xeon E5-2650 v2 processor (8 cores) and 256GB RAM, with similar local storage. However, it has additional storage in the form of a 113TB JBOD attached via fibre channel and configured as a ZFS pool, for storage of monitoring data as it arrives. It also has a connection to a dedicated 113TB Sonexion Lustre filesystem, should additional higher-performance storage be needed.

As power and SEDC data are ingested by the PMDB node, they are saved to several tables in a Postgres database. These tables are partitioned according to the time of ingestion, with the size of each partition being limited to a configurable number of records. As each partition reaches its size limit, a new table partition is started for new data, and the previous partition is dumped to local disk as a file. In addition to the raw Postgres table dumps, we also export this data to CSV files. These files are compressed and transferred, using rsync and an hourly cron job, to the monitoring node, where they are stored to the JBOD storage. Once on the monitoring node, these files can be analyzed in several ways. The Postgres table dumps can be imported into another instance of Postgres for querying in a similar manner to doing so live on the PMDB node; or the CSV files can be parsed and analyzed using a variety of tools.

To aggregate LDMS data, we run an instance of the *ldmsd* daemon on the monitoring node which is configured to pull from the two aggregator daemons on the Cray service nodes. This data is pulled once per second, and stored to the JBOD array in CSV format.

E. Off-platform data analysis

On the off-platform monitoring resources, we can perform data analysis. This includes streaming analysis of the LDMS data before inserting it into the stores, such as computation of functional forms used for the network analysis; analysis of the LDMS data for presentation in directly consumable form (e.g. rates rather than raw counters); and analysis of the log data, either in isolation or in conjunction with the numerical data. Log Data analysis is performed using Baler [6]. Examples of all such analyses applied to a dataset are presented in Section IV.

F. Integration into LANL site monitoring infrastructure

TR2 is deployed in a dedicated network enclave to allow for testing and reconfiguration with minimal impact on the larger production HPC infrastructure. It has been deployed with a dedicated Lustre scratch filesystem, a dedicated NFS appliance for home directories and project files, and dedicated network switches for interconnections between "TR2 proper" (i.e., the Cray service nodes and SMW) and supporting external nodes such as the front-ends, PMDB, and the monitoring node.

Connections to the larger site network have been limited as much as possible, to include only the necessary connections for user access, file transfers, and system administration. Because of these limitations, the integration of TR2 into our larger production monitoring infrastructure has likewise been limited. The bulk of the monitoring data produced by the system, i.e. LDMS metrics, SEDC metrics, and PMDB data, is therefore only analyzed on the dedicated TR2 monitoring node called *"tr2-mon"*.

However, we do forward syslog data, via the SMW, to our site monitoring infrastructure. This allows us to integrate basic system data, which is identical or similar to the same data for our commodity system deployments – e.g., error logs, tests for running services, and security logs – into the same tools we use for managing similar data from our production systems. Currently, our primary tool for managing this data is Splunk [7], which allows us to provide a number of alerts and dashboard visualizations for common system issues. Splunk also provides an easy-to-use search interface, which enables us to easily find and correlate historical data in the course of troubleshooting production issues.

When the TR2 Open Science campaign is completed, the larger volume of monitoring data will be archived and transferred in bulk to our production infrastructure for long term analysis.

III. Controlled Experiments

In support of this work, we obtained a full-system Dedicated Application Time (DAT) on Trinity Phase II. During the DAT, we had exclusive access to the entire system. We ran two controlled workloads, both with and without LDMS monitoring. The goals of the DAT were: 1) to assess performance impact of LDMS on significant ACES applications and 2) to provide us with a controlled experimental dataset from which we could determine actionable indicators to be used for monitoring that would give insight into application performance.

The applications, workloads, and timing results are presented in this section.

A. Experiment Descriptions and Timing Results

Two different workloads were run. Workload 1 consisted of: 4 CTH runs of 1024 nodes each on quad/cache, 2 SPARC runs of 1024 nodes each on quad/flat, and 1 SPARC run of 2048 nodes on quad/flat. Workload 2 consisted of the same set of CTH runs with 4 partisn runs of 1024 nodes on quad/flat. The applications are described in Section III-B.

The workload details and timing results are shown in Figure 3. The workloads were run in the following order in direct succession: 2x Workload 1 with monitoring, 2x Workload 2 with monitoring, 2x Workload 1 without monitoring, and 2x Workload 2 without monitoring.

For this work, the application-to-resource mapping and underlying machine configuration was set up to minimize potential variation due to network contention in order to better assess performance variation due to the monitoring and to more easily identify application interference. Thus, electrical groups were configured entirely as quad/cache or quad/flat and applications were placed to minimize the number of groups and cabinets spanned per application. Applications were run in reservations in order to ensure as similar allocations from run-to-run as possible.

Trinity Phase 2 consists of 50 cabinets and thus 25 electrical groups, with 8 connections between groups. Thus our layout consisted of 12 groups each in quad/cache and quad/flat, with 1 group unallocated. Note that the traffic is still subject to adaptive routing [8], which may result in non-minimal routing and thus traffic in applications may still interfere. The routing rules are the system defaults.

Several nodes were rebooted in the system during the first run of the first workload. As a result, one node in each of CTH1, SPARC1, and SPARC3, and three nodes in SPARC2 were different between the first runs and all subsequent runs. Otherwise, there were no differences in the job-node allocations. All applications in a workload were released simultaneously to the resource manager (moab); with run times planned to largely coincide (15-20 minutes), however, minor variations in the launch time did occur. Adaptive routing (described in Section IV-B1) could result in performance variations in even identically launched workloads.

The system was rebooted earlier in the day, with some intervening runs between that time and the start of the runs presented here. As a result, it is possible that use of the cache may affect the performance of quad/cache workloads. There is no exposed counter that can give us an indication of fragmentation of the cache, so we cannot quantitatively assess the state of the cache. However, the run time were *not* monotonically increasing over time, so overall such an effect was not overwhelming.

From the experiments, we made the following observations:

- No significant in-band monitoring overhead was observed.
- partisn runtimes were highly variable and runtime was not correlated with monitoring on/off.
- CTH4 had significantly longer runtime than any other CTH run.
- Runtimes of CTH were slightly longer in Workload 1 than in Workload 2.

Our goal was to investigate the relationships between our measured indicators and the application run times. In particular, the variations seen suggest possible root causes to be part-to-part variations and impacts of concurrent applications. In this work, we present the current state of our investigations as information and motivation of the kinds of monitoring and analysis we perform. Our analysis is still on-going and no results presented here should be regarded as complete or conclusive.

B. Details of the Applications

a) CTH: CTH [9] is a massively parallel Eulerian finite volume code for solving large deformation and strong shock problems, and is widely used throughout the DOE and DoD. It has models for multi-phase, elastic viscoplastic, porous and explosive materials. CTH uses a structured Cartesian mesh

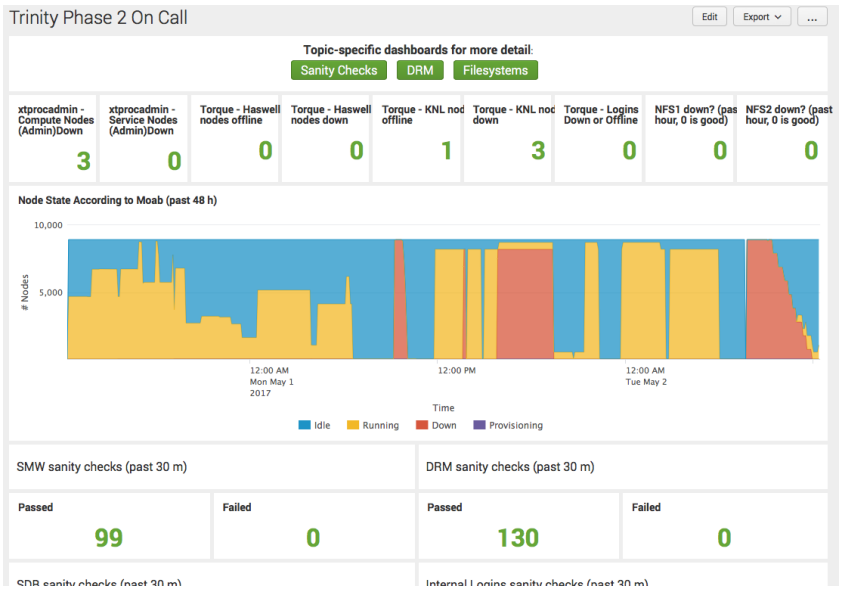


Fig. 2. Screenshot of our standard "on-call" dashboard for monitoring TR2 with Splunk [7]. This dashboard is generated by a combination of regular syslog messages from standard services, as well as metrics which are generated by node-local monitoring scripts and added to the syslog stream.

WK1	Mode	Group	W/Mon 1	W/Mon 2	Baseline 1	Baseline 2
CTH1 (1024)	Quad cache	31 - 33	1282	1275	1297	1285
CTH2 (1024)	Quad cache	34 - 36	1298	1319	1326	1339
CTH3 (1024)	Quad cache	37 - 39	1270	1277	1291	1291
CTH4 (1024)	Quad cache	40 - 42	1452	1456	1465	1439
SPARC1 (1024)	Quad flat	43 - 45	1355	1359	1346	1347
SPARC2 (1024)	Quad flat	46 - 48	1354	1360	1347	1348
SPARC3 (2048)	Quad flat	49 - 54	1482	1485	1485	1487
WK2	**Mode**	**Group**	**W/Mon 1**	**W/Mon 2**	**Baseline 1**	**Baseline 2**
CTH1 (1024)	Quad cache	31 - 33	1206	1205	1223	1209
CTH2 (1024)	Quad cache	34 - 36	1249	1255	1236	1240
CTH3 (1024)	Quad cache	37 - 39	1203	1204	1183	1204
CTH4 (1024)	Quad cache	40 - 42	1403	1417	1413	1437
PARTISN1 (1024)	Quad flat	43 - 45	947	1027	1080	1033
PARTISN2 (1024)	Quad flat	46 - 48	998	888	978	1015
PARTISN3 (1024)	Quad flat	49 - 51	1216	978	1000	992
PARTISN4 (1024)	Quad flat	52 - 54	960	1033	1303	1025

Fig. 3. Workloads and runtime variation used in the controlled experiments.

and has block based Adaptive Mesh Refinement (AMR). The parallelization is done through a standard halo cell exchange method using MPI or memory copies within the same MPI task. Profiling of the code has shown that it is memory bandwidth bound, and the algorithms used have a nearest neighbor's communication pattern.

The version of CTH utilized for this study is version 11.2 with some local modifications to support the Trinity Phase II Open Science period; the latest date of the local modifications is Thursday, February 16th, 2017, at 16:33:47 MST. The CTH used in this study was built with Intel version 17.0.1 compilers and Cray MPICH version 7.4.2.

The CTH test problem used is a 3D elastic-plastic Riemann problem. A 3D domain of 160 cm^3 is divided into 8 quadrants with each quadrant consisting of a different material and initialized to a different pressure, as shown in Fig. 4. The domain is discretized into 2,048 cells with symmetry boundary conditions used for all boundaries. This "flat mesh" setup was crafted to strive for a consistent amount of work for each time step and to better enable weak- and strong-scaling studies.

The CTH test problem was run in quad/cache mode on 1,024 nodes with 64 MPI ranks per node (65,536 total ranks) and 4 cores per node dedicated to core specialization. This test problem was set to perform 241 time steps; this time step limit corresponds to $\sim$20 min. run time at this node and rank count.

This test problem and run time configuration was run thrice

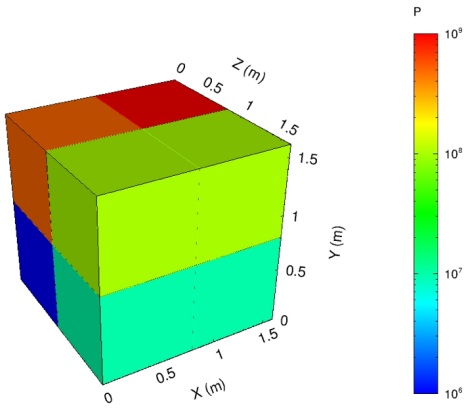


Fig. 4. Initial state of CTH test problem.

TABLE I. CTH PRE-DAT TIMING INFORMATION.

Run/*Quantity*	Time (sec.)
pre-DAT 1	1,238.48
pre-DAT 2	1,174.40
pre-DAT 3	1,260.14
Mean	1,224.34

prior to the DAT. Table I contains these run times and their mean. This gave us an idea of targetting run time and potential runtime variation.

b) partisn: Version 8_23 of Partisn [10] was used for this study. The code was built with version 2.7 of Python. The following source code change was made to `solver/tim3d/timsc3d.f`: write out, as part of `write(601)`, a wall-clock time stamp to the output file on every cycle. This enabled us to track variations in progress through time.

Every Partisn run used 1024 KNL nodes in quad/flat mode, with 64 ranks per node and 1 thread per rank. The input deck was a standard `sntiming.inp` deck, generated with the "make-scaling-decks4" tool. The parameters supplied to the tool were: 720 (zonespcore), 1024 (nodes), and 1 (threads). The `sntiming.inp` file generated by the tool was then edited to use 16 rather than 40 for "nchunk" and 192e-3 rather than 5e-4 for "ts". The application was run with `MPICH_RANK_REORDER_METHOD=3`, and the `MPICH_RANK_ORDER` file was generated using the following `grid_order` command: `grid_order -R -Z -m 65536 -n 64 -g 32x32 -c 4,4`.

Other aprun options used to launch the application include:

- core specialization to segregate OS activities to a dedicated core (`-r1`)
- the `numactl --membind=1` precommand to specify that all application data structures be placed exclusively in MCDRAM.

c) SPARC: Sandia's Parallel Aerosciences Research Code, SPARC, provides advanced simulation of computational fluid dynamics (CFD) calculations. The code is mutli-node parallel, three-dimensional, using traditional MPI but features on-node parallelism provided by OpenMP directives and, for some of the significant computational kernels, implementations using Sandias Kokkos C++ parallel pattern abstractions [11]. SPARC utilizes cell-centered finite volume methods for CFD calculations and Galerkin finite element methods for ablation and thermal analysis [12]. Several packages from the Trilinos framework [13] are also used during execution to provide solutions to complex systems of equations.

IV. MONITORING-ENABLED ANALYSIS

In this section we discuss our approaches to determine actionable metrics that can be associated with the performance issues identified in Section III.

A. Power Analysis

As previously stated, run-to-run performance variation was higher than expected in several cases. For example, in Workload 1 (Table II) the final CTH run was 14% slower than the fastest CTH run, even though each of the four CTH runs were configured identically. More alarmingly, the Baseline1 run of the final PARTISN run, Workload 2 (Table III), was 32% slower than the others, with no obvious explanation. To investigate further, we analyzed the power and energy information collected during each run to look for possible explanations. We focused on comparing the fastest and slowest PARTISN runs in Workload 2, "PARTISN 2" and "PARTISN 4" respectively, as these had the largest run-to-run variation observed. Note that this instantiation of Workload 2 was a Baseline and hence without LDMS monitoring so our available data was limited to that available via the Cray-provided monitoring mechanisms.

As can be seen in Table III, there was not a significant difference in average power per node between PARTISN 2 (201.90 W) and PARTISN 4 (202.75 W). The average power was calculated by dividing the total energy used by the job, as recorded by Cray's RUR tool, by the run's total execution time and then dividing by the number of nodes (1024). RUR additionally breaks down the total energy into CPU and memory components. At this level, there is a more significant 12% difference in memory power between the two jobs. However, PARTISN was configured to run exclusively out of on-package MCDRAM, which is counted in the CPU energy measurement rather than memory energy (external DIMM slots). This suggests the 12% difference may be more a result of part-to-part differences in idle external memory power for the different set of nodes used by each run.

The aggregated job-wide energy usage values reported by RUR obscure the individual node-level details. It could be the case that one node out of the 1024 nodes has a very different power usage behavior than the others, possibly suggesting a "slow node". To probe further, we plotted the 1 Hz power samples recorded for each individual node, shown in Figure 5. The plot includes 1024 separate curves, but they largely overlap making it difficult to see the fine detail. There are roughly 100 spikes evident in each plot, which likely correspond to the 100 cycles that PARTISN was configured to run. In the PARTISN 4 run, the spikes are more spread out than in PARTISN 2, indicating that the slow down in PARTISN 4 is spread out over the entire run rather than centralized to a

single time period. If there were slowdowns due to network or I/O contention, we would expect there to be large dips in power usage during each cycle. This is either not the case or the dips are obscured by the overlapping waveforms for the 1024 nodes. As analyzing each of the waveforms by hand is not practical, it might be useful to apply a clustering algorithm or some other automated technique to look for outliers. This is a possible area for future work.

As a final effort to understand the run-to-run variation, we plotted histograms of the per-node average power usage for each run. The histograms count the number of nodes that had the average power usage shown on the x-axis, with 50 bins used. Separate distributions for total node power, CPU power, and memory power are plotted. In general, the histograms for the two runs look very similar. The primary difference seems to be in memory power, with the slower PARTISN 4 run having a narrower range of values, except for one outlier node that is barely visible with an average memory power of 27 W. This is further confirmed by plotting the 1 Hz memory power samples, shown in Figure 6. We have not yet been able to confirm the reason for this outlier node, but it could be due to a miscalibrated power sensor or the node's memory mode somehow being misconfigured (e.g., set to quad-cache instead of quad-flat). A second run of Workload 2 produced the same outlier behavior, but in the second run the PARTISN 4 performance was as expected (i.e., not significantly slower than the other runs). Hence, we do not believe the high memory power outlier node is the reason for the slower PARTISN 4 run, but we continue to investigate.

B. HSN

Identification and determination of network congestion are generally of interest. While the Aries network performance counters [3] can be used to determine traffic and congestion related metrics of interest, quantification of actionable metrics with respect to performance impact is still unresolved. In this work, since the run times of CTH in Workload 1 are longer than those of Workload 2, though not greatly, we potentially have a dataset which would facilitate the determination performance-impacting indicators.

In this section we present some initial analyses and visualizations of the HSN network data. We first present background on the Aries Network.

1) Aries Background: This section provides a brief description of the Aries network necessary to understand this work. The reader is encouraged to check Faanes et al [8] for more information. Highlights and figures in this subsection are from that work.

The Aries router connectivity is shown in Figure 8. Connectivity within an electrical group (2 cabinets) is shown in the top figure. Aries routers (light blue) are connected to other Aries routers within the same chassis via green links and to symmetrically placed Aries routers in all other chassis in the group via black links. Four nodes (dark blue) share an Aries router ASIC. Connectivity between electrical groups is shown in the bottom figure. All groups are connected via multiple links to all other groups.

[1]Times in this table are derived from the RUR data, which may be slightly different than that of the alps data used in Figure 3.

The Aries network utilizes *adaptive routing*. Packets are generally routed adaptively along either minimal or non-minimal paths, but can also be routed deterministically when the runtime requires in order packet delivery. There are configuration parameters which can influence the routes that can be taken. However, no configurations beyond the standard configurations have been applied to Trinity. The following is a high-level description of adaptive and minimal routing.

With adaptive routing, four possible routes are chosen at random. Two of these are minimal and two are non-minimal. The load on each of the 4 paths is compared and the path with the lightest load is chosen. Minimal routing within an electrical group will take at most 2 hops: one green and one black, in either order. Minimal routing between groups will take 1 blue hop, and minimal routes in the source and destination electrical groups. Non-minimal routing is used to avoid congestion and to spread traffic over the available links in the system. Non-minimal routing within an electrical group can take up to 2 green and 2 black hops. Global non-minimal paths can take up to 10 hops. They route to an intermediate Aries router and then to a destination along a minimal path.

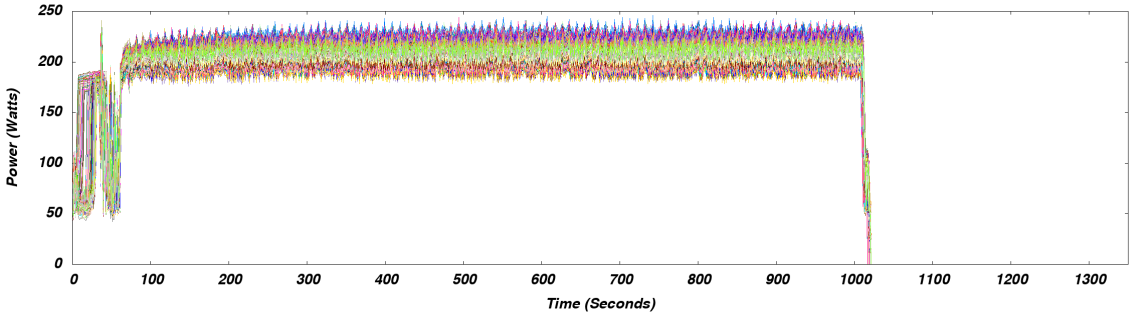
2) Inter-Electrical Group Traffic Analysis: We first seek to assess when an application's traffic is potentially affecting other allocations. To this end, we are investigating visualizations that will enable us to understand inter-group traffic characteristics.

The basis for this visualization is shown in Figure IV-B2. In this case we represent the communications between two electrical groups. Traffic can flow from 0 to 1 or from 1 to 0 across blue links; traffic cannot flow across blue links for any traffic within a group. Thus, we can represent the communications within a 2x2 grid, colored by some attribute value of the link. Traffic incoming into group 1 from group 0 is represented by the yellow color in row 1 column 0. Traffic incoming into group 0 from group 1 is represented by the green color in row 0 column 1. The diagonal elements are not possible and are always colored black.

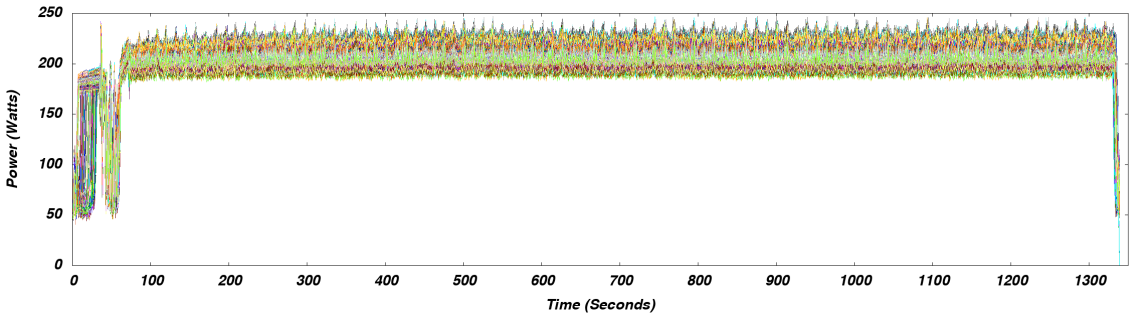
Figure 10 uses this representation for the 24 electrical groups of Trinity Phase 2 The cells are colored by the maximum incoming flits (summed over the Virtual Channels) per sec over a collection interval (1 sec, in this case) over any of the blue links between the two relevant groups during the runtime of the workloads; it is *not* a snapshot in time. The relevant network counter is `AR_RTR_r_c_INQ_PRF_INCOMING_FLIT_VCv`. Details of determination of relevant network counters in the context of the network connectivity can be found in our previous Aries HSN analysis work [5]. The upper figure addresses Workload 1 and the bottom, Workload 2. Green indicates lower values and red indicates higher values. The color-scale is not meant to signify that any particular value is indicative of congestion or "high" load.

Outlined boxes in the figures indicate the placement of the applications. As depicted in Figure 3 CTH 1 is located within groups 31-33, CTH 2 within 24-37, etc. In both figures, the blue boxes correspond to the CTH locations. In the upper figure, the pink boxes indicate the SPARC placements. In the lower figure the purple boxes indicate the partisn placements.

Note that traffic on the system is not limited to application

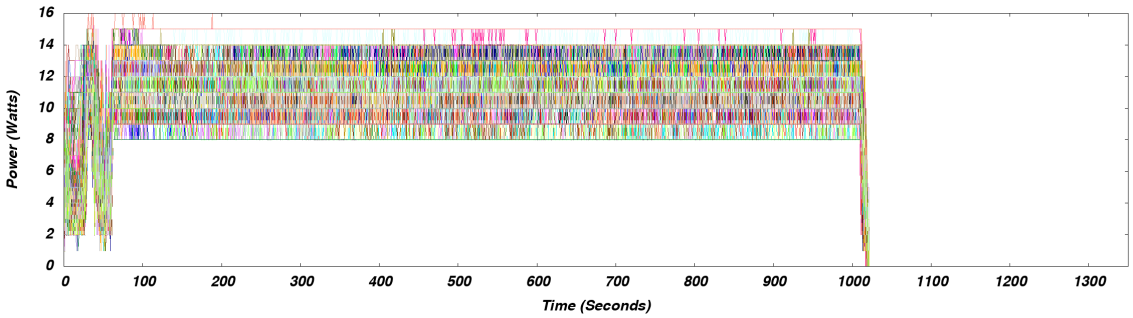


(a) PARTISN 2

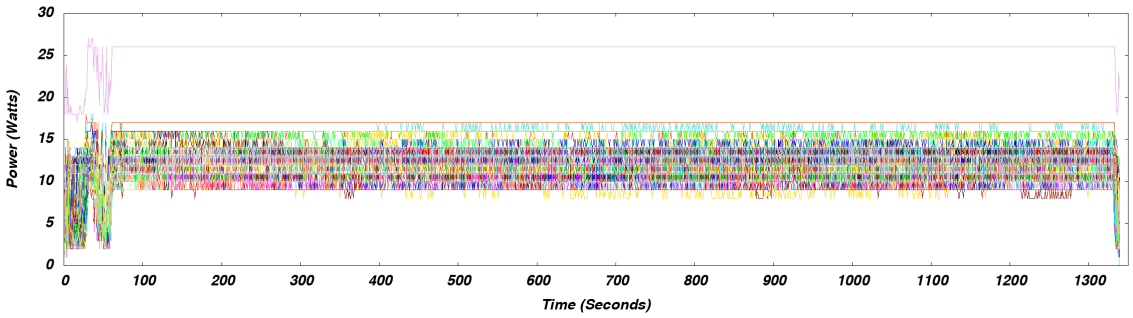


(b) PARTISN 4 (runtime +32%)

Fig. 5. Node-level power over time for two of the PARTISN runs in workload 2. Spikes likely correspond to application cycles indicating that any slowdown in PARTISN 4 is not a large localized event.



(a) PARTISN 2



(b) PARTISN 4 (runtime +32%)

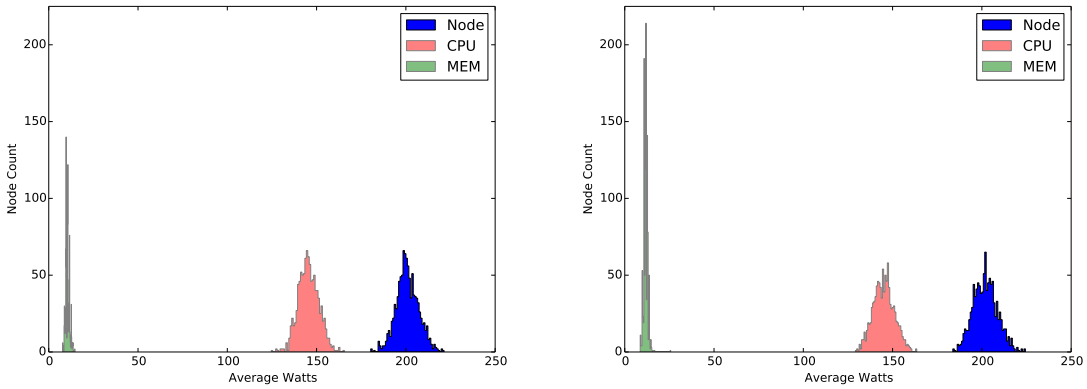
Fig. 6. Memory power over time (external DIMMS, does not include MCDRAM) for two of the PARTISN runs in workload 2.

TABLE II. WORKLOAD 1 POWER AND ENERGY USAGE

	Nodes	Runtime[1] (s)	Total Energy (J)	Avg Power Per Node (W)	Avg CPU Power Per Node (W)	Avg Mem Power Per Node (W)	Number of Nodes Throttled
CTH 1	1024	1343	278268114	207.28	148.84	12.39	0
CTH 2	1024	1339	281224066	205.10	147.28	12.03	3
CTH 3	1024	1304	274096754	205.27	148.24	12.31	0
CTH 4	1024	**1485**	310102646	203.93	146.73	12.20	0
SPARC 1	1024	1371	312268221	222.43	145.78	28.54	0
SPARC 2	1024	1369	306805951	218.86	145.02	26.58	0
SPARC 3	2048	1512	685787490	221.47	145.82	27.89	2

TABLE III. WORKLOAD 2 POWER AND ENERGY USAGE

	Nodes	Runtime[1] (s)	Total Energy (J)	Avg Power Per Node (W)	Avg CPU Power Per Node (W)	Avg Mem Power Per Node (W)	Number of Nodes Throttled
CTH 1	1024	1236	264790220	209.21	149.87	12.30	1
CTH 2	1024	1249	264785934	207.03	148.33	11.95	0
CTH 3	1024	1196	254463798	207.78	149.72	12.23	0
CTH 4	1024	**1424**	299234211	205.21	147.28	12.12	1
PARTISN 1	1024	1120	233431163	203.54	146.73	11.57	5
PARTISN 2	1024	1019	210674495	201.90	146.32	10.53	2
PARTISN 3	1024	1039	215450107	202.50	146.94	10.87	3
PARTISN 4	1024	**1343**	278823301	202.75	145.69	11.81	4



(a) PARTISN 2

(b) PARTISN 4 (runtime +32%))

Fig. 7. Histogram of node-level average power for two of the PARTISN runs in workload 2. The histograms include average power for each of the 1024 nodes in each run, calculated from the per-node 1 Hz power samples recorded during each run. There is an outlier node value in the PARTISN 4 allocation.

traffic alone, although system traffic should not be large. Also, while adaptive routing should contribute to the traffic between groups not sharing an allocation, applications may communicate with nodes outside of the job allocation, for example for IO operations. Since there is no way to perform traffic attribution, we cannot determine the origin an ultimate destination of the traffic. However, we seek to examine visualizations such as these to provide insight into network utilization and to help tie it to application performance. For example we see that there is traffic from all electrical groups into all other electrical groups, despite the job layout, and indication that at least the maximum traffic in Workload 1 is greater than that in Workload 2, which is in alignment with the observation that runtimes of CTH in Workload 1 were greater than those of Workload 2. Four snapshots in time (starting at an arbitrary time) of the traffic for Workload 1 are shown in Figure 11. Each snapshot represents a 1 second interval and the time between snapshots is 30 seconds. This sequence shows temporal variations with fluctuation in intensity of communications.

3) Backpressure Analysis: In assessing congestion, Stall to flit ratios [3] can be indicative of backpressure at the various interfaces. Stall counters increment when a flit which is ready to forward is prevented from dong so by backpressure. If the stalls and flit increment at equal rates, e.g., ratio of 1, then flits are crossing the interface at half the rate they would be if there were no stalls.

Figure 12 shows the percentage of links between electrical groups with stall to flit ratios equal to or greater than various threshold values. These are shown throughout the times of the workloads. The plots include *all* links, not just the maximum value as in the previous subsection. The relevant stall counter to be used with the flit counter above is `AR_RTR_x_y_INQ_PRF_ROWBUS_STALL_CNT`. From discussions with Cray engineers [14], 0.25 is a normal value for this quantity in the network.

Consistent with the indications of the previous subsection, we see that a higher percentage of the links experience greater values of backpressure in Workload 1 (top) than in Workload 2 (bottom). While we seek to assess the relationship of observed

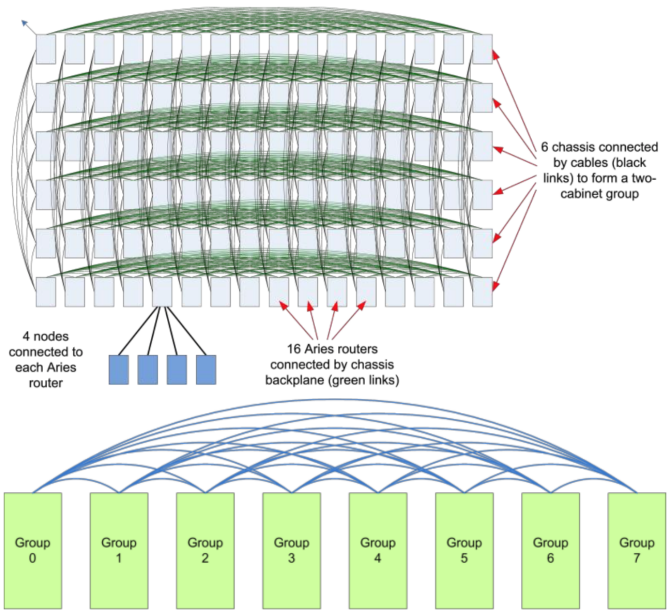


Fig. 8. Aries network connectivity. Within an electrical group (2 cabinets) (top): Aries routers (light blue) are connected to other Aries routers within the same chassis via green links and to symmetrically placed Aries routers in all other chassis in the group via black links. Four nodes (dark blue) share an Aries router ASIC. Between electrical groups (bottom): All groups are connected via multiple links to all other groups. Figures from Faanes et al [8].

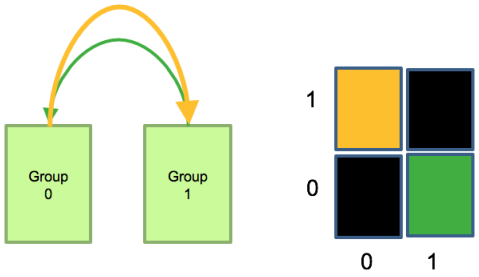


Fig. 9. Explanation of the layout of subsequent figures. Communications between electrical groups can be shown on a colored grid. The convention used is incoming traffic into group 1 from group 0 is shown in row 1 column 0. Communications within a group will not utilize the blue links and hence all diagonal elements are colored black.

values on application performance, our guidance can give us some indication of the possible significance of the values. For example, Workload 1 generally has a stall to flit ratio greater than 1, in about 50 percent of the links while Workload 2 has values that high in only 20 percent of the links.

The stall to flit ratio is relevant to all interfaces. For example, backpressure at the node is a function of stall to flit ratios between the PTILES and the NIC's. As per guidance from Cray Engineers [14] we have been assessing a functional form of the ratio of `AR_NL_PRF_REQ_PTILES_TO_NIC_n_STALLED` to `AR_NL_PRF_PTILES_TO_NIC_n_FLITS` as an assessment of backpressure at the node (details are beyond the scope of this paper). Histograms of the percentage of NICs with maximum backpressure exceeding chosen thresholds over the run time of the workloads is shown in Figure 13. All

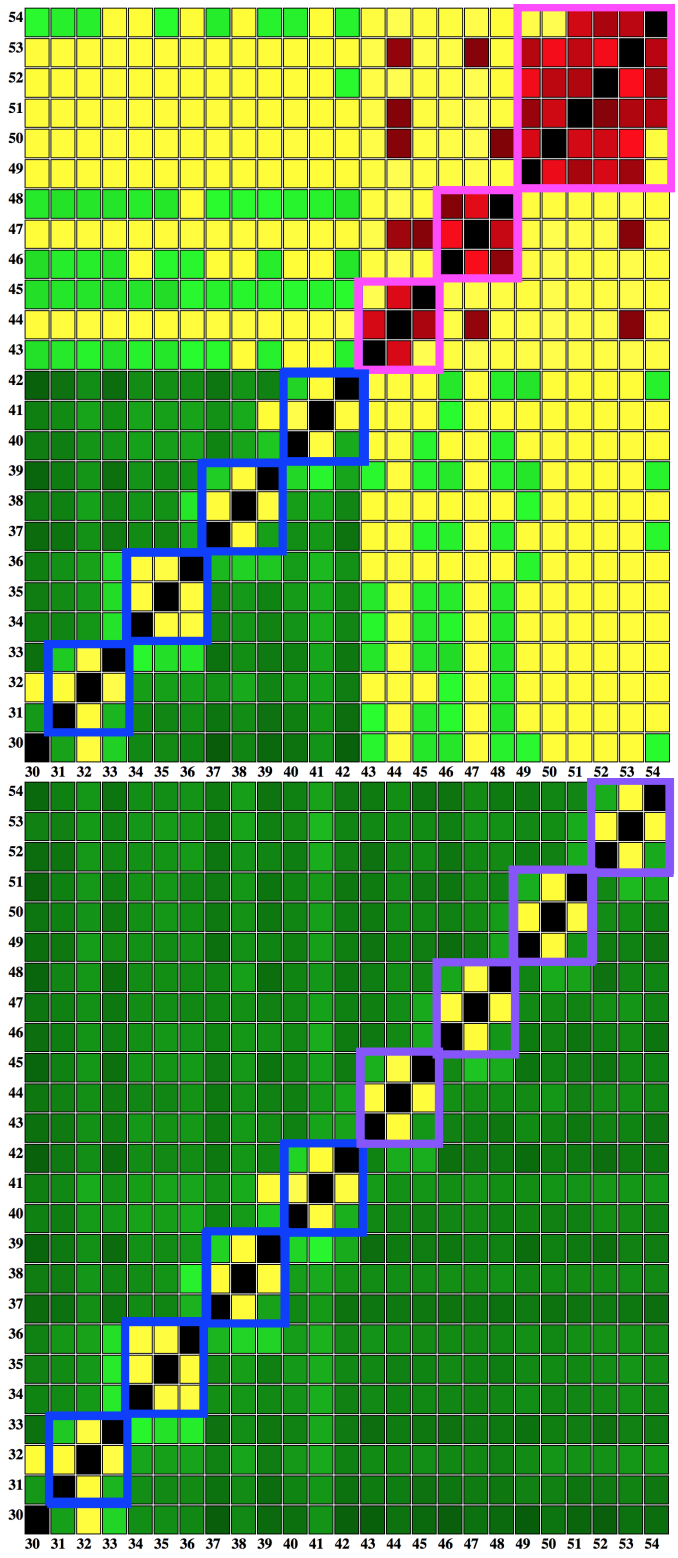


Fig. 10. Traffic (Flits/s) between electrical groups. Values shown are the highest value over the course the workload for any link of those between the two groups. Workload 1 (top) and Workload 2 (bottom).

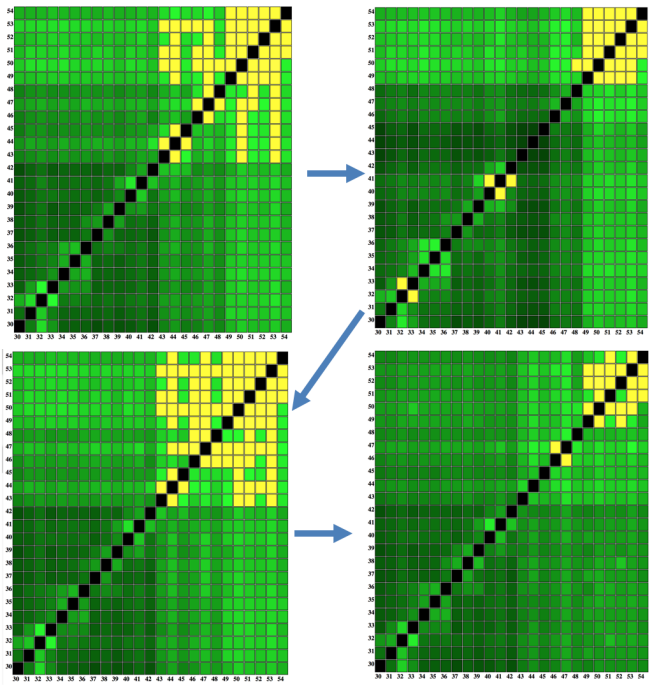


Fig. 11. Timeseries of traffic in Workload 1. Each panel is a snapshot colored by the highest value for any link between those two groups over a 1 second interval. Time of the data snapshot is 1 second. Time between panels is 30 seconds; arrows indicate the progression of time. Fluctuations in the intensity of communications is seen.

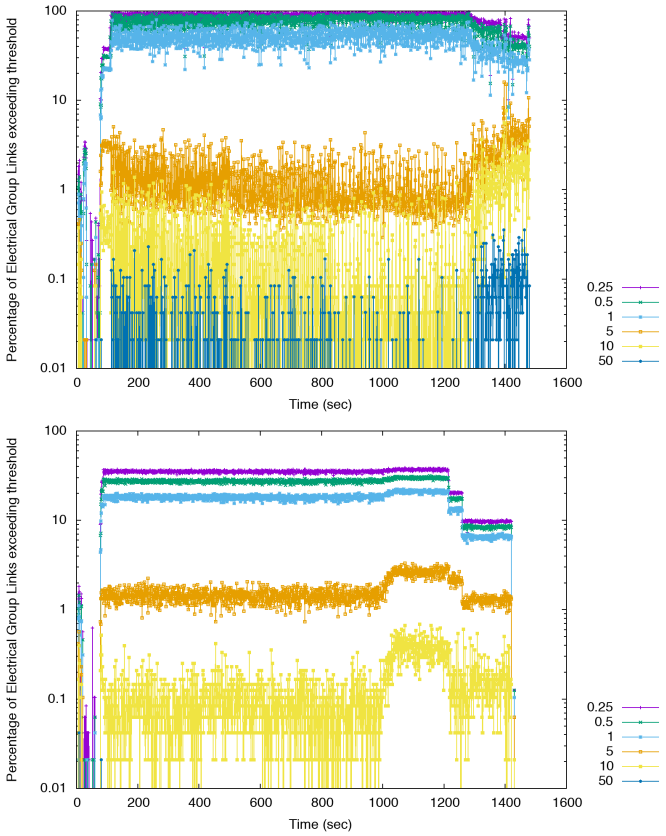


Fig. 12. Percentage of links between electrical groups exceeding Stall/Flit thresholds through time. Workload 1 (top) and Workload 2 (bottom). Larger values indicate higher backpressure. A higher percentage of the links experience greater values of backpressure in Workload 1.

(100 percent) of the NICS have maximum values greater than 0 in both workloads. In general, more links have higher backpressure in Workload 1 than in Workload 2.

Of particular interest is that there is one NIC (0.01 percent of the total NICs) that has higher backpressure than others. A time history examination (unshown) indicates that this NIC is continuously higher through time. This NIC is `c0-7c0s14 NIC 3` is in node allocation of the long runtime CTH4 case in both workloads. We continue to investigate the cause of the backpressure and how to determine if this backpressure is the cause of the long runtime.

C. Log Analysis

With new system hardware and software comes changes in the events of interest and in the logging of such. Thus, we cannot know apriori the log message indicators of events. Brute-force examination of the logs to determine events of interest is not feasible. Five months of the TR2 logs, including pre-production time, contain over 4.5 billion log lines (not including the job related data).

One tool we use for log analysis is Baler [6]. Baler generates *patterns* from log lines requiring no user domain knowledge for determining log lines of interest. The user supplies a dictionary of words; words in the log lines are retained and all other items in the log lines are turned into variables, indicated by `*`. Patterns from log lines are shown in Figure 14. Many log lines reduce to a few patterns, thus easing search and the ability to identify events of interest, including log events that are similar in periods of time or across certain components. An interface exists for querying and searching patterns through time and across components. We have used Baler in previous Trinity-related work to determine significant patterns for the Phase 1 architecture [15], [16].

We have supplemented the Baler English dictionary with a domain-specific dictionary of approximately 100 words, such as `Lustre`, `DIMM`, and `aprun`. Baler pattern analysis and meta-pattern grouping reduces the 4.5 billion log lines to 11,000 patterns. While this is a substantial improvement, we have applied a further heuristic to more favorably highlight the patterns of interest. We have additionally weighted the patterns by the presence of approximately 50 words of interest such as `critical` or `congestion`. Occurrences of more words results in a higher overall pattern weight. After the weighting, this reduces to 1350 meta-patterns. Examples of top-weighted meta patterns are shown in Figure 15.

Numerical data can be turned into patterns as well, by defining a data source and numerical value range as its own pattern. In this way, Baler can be fed numerical data, such as backpressure calculations, in order to enable association of numerical data and log events.

We searched the Baler patterns during the DAT in order to extract significant events, particularly as they might relate to the observations of the runs. One possible pattern of interest was `* HWERR[*-*][*]:*:The pcie had * link width change or * speed change (*, *,`

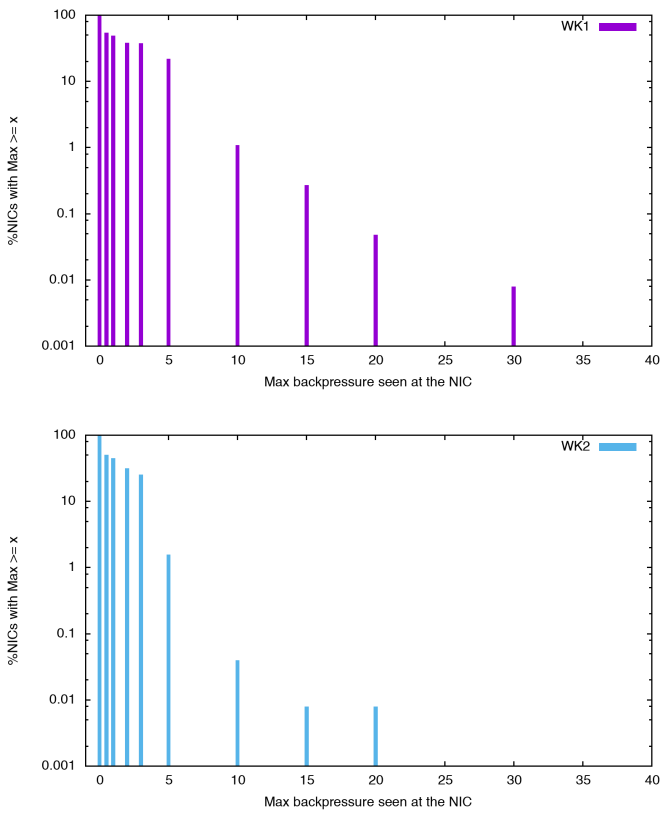


Fig. 13. Percentage of NICs with Max backpressure greater than or equal to threshold value. Larger values indicate higher backpressure. Workload 1 (top) and Workload 2 (bottom). Backpressure at one NIC (0.01 percent of the total number) is significantly higher than all other NIC's values.

Patterns:

* - - Node * interrupt *=*, *=*, *=* *[*]: * * * Processor Hot

* * - - Node * power budget exceeded! Power=*, Limit=*, * Correction Time=*

Loglines:

bcsysd 2080 - - Node 2 interrupt IREQ=0x20000, USRA=0x0, USRB=0x80 USRB[7]: C0_PROCHOT CPU 0 Processor Hot

bcpmd 2140 - - Node 2 power budget exceeded! Power=340, Limit=322, Max Correction Time=6

Fig. 14. Baler extracts patterns from log data without requiring active user input.

`or * speed)`. This pattern occurred on one component at the beginning of the DAT when we performed a reboot at the beginning of the day and again when the machine was rebooted before returning the machine to the users. The component in question was that with the NIC of the continuously high node backpressure in run CTH4, `c0-7c0s14a0n3`. This is not a common pattern; over the lifetime of Trinity Phase 2 this pattern has occurred 6 times on this component, 6 times on `c11-5c1s2a0n1` and 1 time on each of 3 other components. We are investigating if this event could be related to the high back pressure at the same NIC and/or to the performance of CTH4.

TABLE IV. CTH IN WORKLOAD 1 (W/ SPARC) TIMING INFORMATION.

Run/ ***Quantity***	**Mean Time w/o LDMS (sec.)**	**Mean Time w/ LDMS (sec.)**	**Increase "from LDMS"**
CTH1 (1024)	1,290.60	1,278.73	-0.92%
CTH2 (1024)	1,332.30	1,308.48	-1.79%
CTH3 (1024)	1,290.52	1,273.54	-1.32%
CTH4 (1024)	1,451.98	1,454.03	+0.14%
Mean	1,341.35	1,328.69	-0.94%

TABLE V. CTH IN WORKLOAD 2 (W/ PARTISN) TIMING INFORMATION.

Run/ ***Quantity***	**Mean Time w/o LDMS (sec.)**	**Mean Time w/ LDMS (sec.)**	**Increase "from LDMS"**
CTH1 (1024)	1,215.61	1,205.58	-0.82%
CTH2 (1024)	1,237.72	1,252.19	+1.17%
CTH3 (1024)	1,193.72	1,203.64	+0.83%
CTH4 (1024)	1,424.89	1,410.48	-1.01%
Mean	1,267.98	1,267.97	0.00%

TABLE VI. MEAN "LOADAVG" OVER ALL 1,024 NIDS OF EACH JOB.

	JIDs	Mean Over NIDs
1st CTH1 (w/ SPARC)	92115	6,085.7
1st CTH2 (w/ SPARC)	92116	6,095.1
1st CTH3 (w/ SPARC)	92117	6,077.8
1st CTH4 (w/ SPARC)	92118	*6,119.0*
2nd CTH1 (w/ SPARC)	92119	6,063.1
2nd CTH2 (w/ SPARC)	92120	6,088.2
2nd CTH3 (w/ SPARC)	92121	6,065.9
2nd CTH4 (w/ SPARC)	92122	*6,103.7*
1st CTH1 (w/ PARTISN)	92135	6,042.3
1st CTH2 (w/ PARTISN)	92136	6,072.1
1st CTH3 (w/ PARTISN)	92137	6,044.1
1st CTH4 (w/ PARTISN)	92138	*6,101.2*
2nd CTH1 (w/ PARTISN)	92139	6,045.0
2nd CTH2 (w/ PARTISN)	92140	6,063.0
2nd CTH3 (w/ PARTISN)	92141	6,059.0
2nd CTH4 (w/ PARTISN)	92142	*6,151.3*

V. MONITORING IMPACT

In this section we use the run times to assess the impact of the in-band monitoring. Since the partisn runtimes were highly variable across all runs, including runs for the same placement, we cannot use this application for assessing impact.

A. CTH

Each workload was performed twice with and without LDMS monitoring present during the DAT. Table IV provides the data from Workload 1 (i.e., CTH SPARC) and Table V provides the data from Workload 2 (i.e., CTH with PARTISN).

These tables contain the mean run times with and without LDMS for each of CTH's 4 reservations in addition to the mean of all reservations. The differences between the data with and without LDMS running contains no more than $\sim 2\%$ variation, which is within the observed run-to-run variations present from the DAT and pre-DAT data (Table I). Moreover, if the mean across all 4 reservations is taken into account, the data with LDMS on had either an overall reduced mean (see Table IV) or practically no impact whatsoever (see Table V).

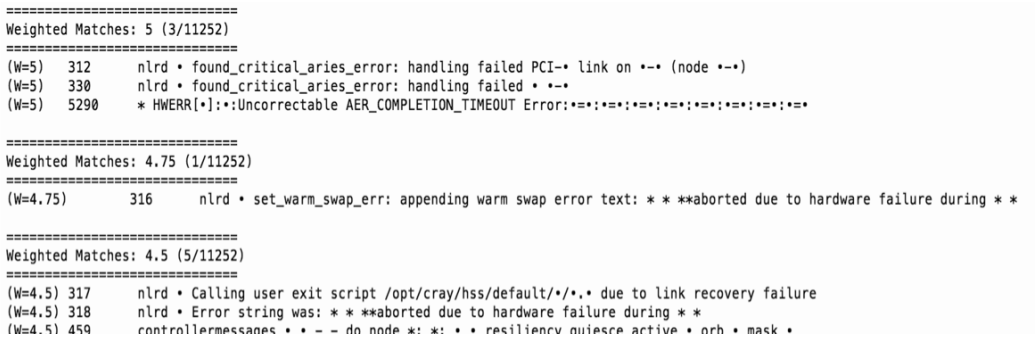


Fig. 15. Selection of top weighted Baler meta-patterns. Weights (W) and number of occurrences are listed.

TABLE VII. SPARC IN WORKLOAD 1 (W/ CTH) TIMING INFORMATION.

Run/ *Quantity*	Mean Time w/o LDMS (sec.)	Mean Time w/ LDMS (sec.)	Increase "from LDMS"
SPARC1 (1024)	1,346.5	1,357.0	0.78%
SPARC2 (1024)	1,347.5	1,357.0	0.71%
Mean	1,47.0	1,357.0	0.74%

TABLE VIII. WORKLOAD 1 (SPARC+CTH) TIMING INFORMATION.

Run/ *Quantity*	Mean Time w/o LDMS (sec.)	Mean Time w/ LDMS (sec.)	Increase "from LDMS"
SPARC3 (2048)	1,486.0	1,483.5	-0.17%

LDMS has no noticeable impact on CTH run times on Trinity Phase II.

Tables IV and V indicate that "CTH 4" consistently had a larger run time than the other three CTH groupings. Average load from LDMS was analyzed for these allocations across each NID during these simulations. Table VI contains the mean load average for each of these jobs. The load average for "CTH 4" is consistently higher than the other three CTH groupings as well. This mean is not the result of an outlier; all of the NIDs for "CTH 4" have this higher average load. Further investigations are ongoing to better understand the cause for this increased load.

B. SPARC

During the DAT, each workload with SPARC was performed on the same nodes, twice with and without LDMS running concurrently. Tables VII and VIII provide the SPARC data from Workload 1 (SPARC was not present in Workload 2).

These tables contain the mean run times, labeled "Mean Time", with and without LDMS running for each of SPARC's 3 reservations in addition to the mean of all reservations. Across all twelve SPARC runs the differences in run times, with and without LDMS running, exhibit less than $\sim 1\%$ variation. In the SPARC (1024) runs there appears to be a clear, though small, impact of $\sim 0.8\%$ that, unlike the CTH case, does not fall within the observed $\sim 0.15\%$ run-to-run variations present in the baseline cases. Also, unlike the CTH case, we had no pre-DAT run data with which to gauge our DAT data. Interestingly, in the SPARC (2048) case, shown in Table VIII we observe an impact of $\sim -0.17\%$ when we would have expected an even greater impact, than for the SPARC (1024) case, in run-time due to the presence of LDMS because of the 2X increase in application scale. We conclude that, while there may be cases where there is an impact greater than the natural run-to-run variation for SPARC, the impact is minimal and worth the information access it can provide.

C. Additional Planned Testing

We will be augmenting this data with additional runs on Mutrino, the Trinity testbed sited at SNL. This will include impact on smaller benchmark runs. We will also run PSNAP [17], which measures OS jitter. Since we run this on a per-node basis [1]), we can get representative results on the smaller system.

VI. CONCLUSIONS AND FUTURE WORK

In this work we have presented our system for large-scale metric collection and analysis of system metrics, platform environmental data, energy consumption and logs. Future work includes solidifying the Trinity monitoring data paths after the system integration in June. We have paths forward for integration of application-provided data, such as phase of an application, with system data and for better support of streaming analysis, including on-node options.

From our DAT, we have obtained a dataset which is a unique resource for determining if there are actionable metrics that we can associate with performance impact and system issues. We have only just started analysis of the dataset, with some initial investigations presented here.

More globally, outstanding questions in enabling meaningful analysis of HPC monitoring datasets remain. How do we best present and analyze large numbers and large dimensions of data for exploration? Often one attempts to reduce the dimensions of the problem, for example, by reducing the number of variables explored, or by doing aggregations of data across multiple components or across time, as we have done here. However it is pertainent to ask: what dimensional reductions still result in meaningful results?

Some of our analyses require significant understanding of architectural issues of the system. The Cray monitoring community has sought to determine how we can better work with Cray in order to get the domain knowledge that we need to guide analysis and to distinguish significant event associations

from coincidences. Machine learning has been suggested and investigated for years in the hopes of automatically extracting information from high dimensional data, however in various domains many "unsupervised" learning procedures return meaningless results.

Community forums, such as the Cray System Monitoring Working Group, and collaboration can help to address such questions.

REFERENCES

[1] A. DeConinck, A. Bonnie, K. Kelly, S. Sanchez, C. Martin, M. Mason, J. Brandt, B. Allan, A. Agelastos, M. Davis, and M. Berry, "Design and implementation of a scalable monitoring system for Trinity," in *Proc. Cray User's Group*, 2016.

[2] 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.

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

[4] ——, "Cray Linux Environment (CLE) 4.0 Software Release," Cray Doc S-2425-40, 2010.

[5] J. Brandt, E. Froese, A. Gentile, L. Kaplan, B. Allan, and E. Walsh, "Network Performance Counter Monitoring and Analysis on the Cray XC Platform," in *Proc. Cray User's Group*, 2016.

[6] 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.

[7] "Splunk: Operational Intelligence, Log Management, Application Management, Enterprise Security, and Compliance." [Online]. Available: http://www.splunk.com

[8] G. Faanes, A. Bataineh, D. Roweth, T. Court, E. Froese, B. Alverson, T. Johnson, J. Kopnick, M. Higgins, and J. Reinhard, "Cray Cascade: A Scalable HPC System Based on a Dragonfly Network," in *Proc. Int'l Conf. on High Performance Computing, Networking, Storage and Analysis (SC12)*, 2012.

[9] J. M. McGlaun and S. L. Thompson, "CTH: A Three-Dimensional Shock Wave Physics Code," *International Journal of Impact Engineering*, vol. 10, pp. 351–360, 1990.

[10] 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.

[11] H. C. Edwards, D. Sunderland, V. Porter, C. Amsler, and S. Mish, "Manycore performance-portability: Kokkos multidimensional array library," *Scientific Programming*, vol. 20, no. 2, pp. 89–114, 2012.

[12] M. A. Howard, "A multi-dimensional finite element based solver for decomposing and nondecomposing thermal protection systems," in *Proc. of the AIAA Conference on Aviation 2015*, 2015.

[13] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, "An overview of the trilinos project," *ACM Trans. Math. Softw.*, vol. 31, no. 3, pp. 397–423, Sep. 2005. [Online]. Available: http://doi.acm.org/10.1145/1089014.1089021

[14] L. Kaplan, D. Roweth, and E. Froese, private communication(s), Cray, Inc.

[15] 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.

[16] 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.

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