SaNSA - the Supercomputer and Node State Architecture

Neil Agarwal*†, Hugh Greenberg*, Sean Blanchard*, and Nathan DeBardeleben*
* Ultrascale Systems Research Center
Los Alamos National Laboratory
High Performance Computing
Los Alamos, New Mexico, 87545
Email: {hng, seanb, ndebard}@lanl.gov
† University of California, Berkeley
Berkeley, California, 94720
Email: neilagarwal@berkeley.edu

***Abstract*—In this work we present SaNSA, the Supercomputer and Node State Architecture, a software infrastructure for historical analysis and anomaly detection. SaNSA consumes data from multiple sources including system logs, the resource manager, scheduler, and job logs. Furthermore, additional context such as scheduled maintenance events or dedicated application run times for specific science teams can be overlaid. We discuss how this contextual information allows for more nuanced analysis. SaNSA allows the user to apply arbitrary attributes, for instance, positional information where nodes are located in a data center. We show how using this information we identify anomalous behavior of one rack of a 1,500 node cluster. We explain the design of SaNSA and then test it on four open compute clusters at LANL. We ingest over 1.1 billion lines of system logs in our study of 190 days in 2018. Using SaNSA, we perform a number of different anomaly detection methods and explain their findings in the context of a production supercomputing data center. For example, we report on instances of misconfigured nodes which receive no scheduled jobs for a period of time as well as examples of correlated rack failures which cause jobs to crash.**

***Index Terms*—system state, node state, health monitoring, anomaly detection, software architecture**

I. Introduction

Roughly two decades ago the "Beowulf Revolution" [1] began to take over the field of supercomputing. While we often refer to these machines today simply as *clusters*, they are extremely similar in nature and are identified by being comprised primarily of commodity hardware technology that typically makes use of open source operating systems, middleware, and tools. The extreme-scale systems which push the bleeding-edge of technology forgo these constraints in several areas to achieve extreme scales.

The move to open source operating systems and middleware has had many benefits but one of the problems is that it can be difficult to standardize. Of interest to our discussion is the notion that capturing the state[1] of the individual nodes of the cluster over time requires a great deal of knowledge that must be acquired from many different sources. While historically this feature would have been provided by a supercomputing vendor, today there exists relatively poor *fine-grained* information about the state of individual nodes of a cluster. Operators and technicians of data centers have dashboards which actively provide real time feedback about coarse-grained information such as whether a node is up or down.

The Slurm Workload Manager, which runs on most of the U.S. Department of Energy's clusters, has one view of the state of the nodes in the system. Slurm uses this to know whether nodes are available for scheduling, jobs are running, and when they complete. However, there are other views such as the node's own view as expressed via system logs it reports to centralized logging infrastructures and administrator views which annotate sets of nodes in states such as being upgraded, serviced, or allocated for special scientific workloads.

Sometimes these views conflict, and different consumers of this data have different needs and place different value judgments on it as well. It would not be unreasonable for one to say that it does not matter that a node is up if it is unreachable and not able to be scheduled by the resource manager. That would be a very *user-centric* view. However, from the perspective of node reliability, it seems incorrect to count it as truly down if the hardware was up the entire time and some other system, perhaps a software timeout, kept it from being accessible.

As system architects interested in understanding cluster health over time, we believe that there does not exist a system for properly capturing the information we need and performing the analyses necessary. Therefore, in this paper, we present SaNSA, the Supercomputer and Node State Architecture, a software architecture and infrastructure we have designed which addresses these needs at Los Alamos National Laboratory (LANL).

In Section II, we first describe the architecture of SaNSA. Then, in Section III, we apply SaNSA to several problems to demonstrate the power that is enabled by the underlying state architecture. We identify anomalous nodes on a large production cluster at LANL and then use topology-aware analysis to identify a rack in that same cluster that is being under-utilized by the scheduler. We then show how SaNSA

[1]State refers generally to what a node or supercomputer is doing such as running a job, booting up, or down for maintenance.

can be used to explore the state transition probabilities of a large open compute cluster to identify the signature for "normal" operation in a typical cluster node. We also calculate the percentage of time this cluster spends in each state and hint at future work in Markov processes. We then use SaNSA to explore correlated failure events that involve hardware failures that ultimately result in job interruptions. In Section IV, we look at some related work and finally conclude with a discussion of our future plans in Section V.

II. SaNSA Architecture

Figure 1 depicts the design of SaNSA which combines data sources (the input), a central repository for intermediate storage, and an interface for a variety of users (the output). Multiple data sources can be used as inputs to make up the node and system state. We discuss the data sources used in our initial experiments with SaNSA in Section II-A. These sources are then aggregated together depending on the view to resolve the state from a given perspective. Although rare, sometimes different source inputs will contradict such as a node's system log stating that it was up yet the resource manager cannot communicate with that node and marks it as a lost connection. Which state is the node in? It is a matter of perspective and different analysts need both pieces of information for different reasons at different times. The different states we have defined at this time are discussed in Section II-B. However, SaNSA is extensible and we expect more states to be added as we expose signals to capture the state. Once the state information is assembled, analytics can be performed. We outline a number of different applications in Section III.

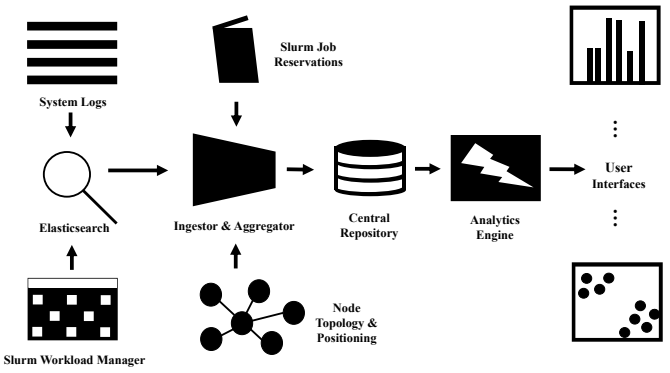

Fig. 1. A visual depiction of the flow of data within SaNSA. The *Ingestor & Aggregator* is capable of pulling data from multiple sources, highlighting SaNSA's capability of easily combining more data sources in the future. Moreover, multiple user interfaces allow analyses to be performed by multiple actors, suggesting that SaNSA can be valuable across teams and management hierarchies.

A. Data Sources

Currently, the system directly ingests data from two components: Elasticsearch [2] and the Slurm Workload Manager [3]. Elasticsearch, a distributed search and analytics engine capable of fast and efficient data level queries, handles system logs. As each compute node produces millions of lines of system logs a day, Elasticsearch enables easy and quick data retrieval. LANL's Elasticsearch infrastructure not only has full system logs from all nodes, but also telemetry as well as LDMS [4] metric data for an increasing number of systems.

TABLE I
DATA SOURCES FROM CLUSTERS AT LANL INGESTED INTO SANSA. SYSTEM SIZES INCLUDE SOME NON-COMPUTE NODES SUCH AS ADMINISTRATIVE NODES, IO NODES, ETC.

Cluster Name	Program	Processor	Total Nodes
Grizzly	CTS-1	Intel Xeon Broadwell	~1500
Wolf	TLCC2	Intel Xeon Sandybridge	~640
Snow	CTS-1	Intel Xeon Broadwell	~380
Woodchuck	N/A	Intel Xeon Haswell	~200

SaNSA also ingests data from the Slurm Workload Manager. The supercomputers at LANL use this manager for handling the queuing and distribution of user jobs across all the compute nodes. The software continuously produces data about job completion status, logging various end states. A portion of this data is fed to Elasticsearch which relays the data to our system. The rest of the data, such as job reservations for system maintenance / upgrades as well as dedicated user reservations is ingested directly by SaNSA.

SaNSA also allows for non-time-series data to be ingested, such as attributes about the nodes. In the study presented here, the attributes used are the node topology and placement within the data center. This allows us to determine in three dimensional space where each node is physically located. Given the volume of data that we will ingest, using machine learning techniques to find correlations on data center positions could prove useful and may lead to insights.

Table I lists the clusters used in this study. These represent easy-to-access machines that execute unclassified computing jobs. While these are also some of the smaller systems, we are in the process of moving SaNSA to LANL's larger supercomputers. In Table I, CTS, or Commodity Technology Systems, are the NNSA's standardization approach across many laboratories for clusters that are relatively standard technology and well characterized. TLCC are TriLab Linux Capacity Clusters and represent a similar design philosophy but simply older generation technology. Woodchuck is a one-off system for data analytics and, as such, does not fall into a CTS or TLCC category.

The time period under study was from February 20, 2018 to August 31, 2018 (190 days). From the perspective of system logs, we ingested over 1.1 billion individual lines (462 million from Grizzly, 116 million from Wolf, 549 million from Snow, and 23 million from Woodchuck).

B. Node States

Node states make up the core of SaNSA and through the compilation and aggregation of data from these sources, we create an internal view of the state of not just the individual nodes, but also as an aggregation of components that form a single view of the overall system as a function of time. SaNSA's flexible state architecture allows for the definition of

new states easily based on conditions specified by configuration parameters. Based on the current data, we have identified 8 different states:

1) DOWN - A node becomes unavailable and is followed by a reboot [2].
2) CONNECTION_LOSS - The Slurm Workload Manager loses contact with the node but the node is still active.
3) BOOTING - The node is booting, starting services, loading modules / drivers, etc.
4) IDLE - The node is up but not currently running a job. Nodes will appear as IDLE as Slurm builds large allocations for a pending job that it has selected for future execution, or for a job that has a specified advance reservation. At this time we have not yet identified a "Slurm is idling this node for a big upcoming job" signal (work in progress).
5) JOB_RUNNING - A user-defined Slurm job is running on this particular node.
6) ADMIN_RESV - The node is in a reserved state for HPC administrative maintenance and improvement work.
7) RESERVED_IDLE - The node is in reserved state for a user. This is usually done when groups of nodes are allocated to certain projects to accomplish required work deliverables for mission tasks so that the resources are dedicated to only those users.
8) UNSCHEDULED_RESV - *Unscheduled* outages such as power outages (e.g. lightning strikes) and emergency maintenance. We would expect these to be rare events.

C. State Priorities, Compilation & Aggregation

SaNSA builds the state for each node over time based on information gathered from different sources in a "patchwork" way. The patchwork analogy is particularly true in the sense that with SaNSA, states can have priorities. For instance, for the states we defined in Section II-B, we set UNSCHEDULED_RESV and ADMIN_RESV as the highest priorities, respectively. What this means, is that if we record a node (or nodes) as DOWN or any state, these priority states "overlay" on top of the other states effectively shadowing them. This is important because when we calculate statistics such as downtime, we do not want to include events that occurred during dedicated administrative windows when many, often repeated reboots of nodes occur that greatly skew statistics. While calculating those statistics themselves may be interesting, they are outside the scope of what we are discussing in this paper. They are, however, available within SaNSA under the shadowed overlay.

Slurm's *view* of a node's state is probably the most important view as it represents schedulability of resources and ultimately impacts the utilization of the system resources for productive computation. To determine loss of connection, we search for messages from the process *slurmctld* containing the phrase `not responding, setting DOWN`. This indicates that Slurm has identified that a node is not responding and marking it as "down." It is important to note that this is *Slurm's* notion of down and we term this a CONNECTION_LOSS instead. Later we will discuss the node DOWN state. The following is a single record of a sample response:

```
Time: 6/9/2018, 3:29:14 AM, Host:
gr-master, Ident: slurmctld
[2018-06-09T03:29:13.803], "error:
Nodes gr[0084,0097] not responding,
setting DOWN"
```

This record explains that at 3:29:14 AM on 6/09/18, Slurm lost connectivity with nodes 84 and 97 of Grizzly. This allows us to categorize the state of those nodes of Grizzly as CONNECTION_LOSS starting at that time. Next, we must figure out when the nodes come back.

To do this, we then check when the nodes become responsive again. The query searches for log messages from *slurmctld* containing the phrase `now responding`. The following is a single record of a sample response:

```
Time: 6/13/2018, 10:07:09 AM,
Host: gr-master, Ident: slurmctld
"[2018-06-13T10:07:00.379] Node gr0084
now responding"
```

This record explains that on 6/13/2018, Slurm reestablished communication with node 84 of Grizzly at 10:07:09 AM. We have now established the start and stop times for the CONNECTION_LOSS state.

In SaNSA, the difference between CONNECTION_LOSS and DOWN is determined by whether the node ends up booting after a lost connection. Properly differentiating these two states is clearly important.

To establish booting, we search for system logs from the process *systemd* and containing the phrase `Startup finished`. The following is a single record of a sample response:

```
Time: 6/13/2018, 10:07:09 AM, Host:
gr0084, Ident: systemd, "Startup
finished in 19.988s (kernel) + 10min
57.968s (userspace) = 11min 17.957s."
```

This record explains that at 10:07:09 AM on 6/13/18, node 84 of Grizzly was booting for the last 11 minutes and 17.957 seconds. This allows us to categorize the state of node 84 of Grizzly as BOOTING for that time period. However, notice that we previously categorized the time up to 10:07:09 AM as CONNECTION_LOSS. This appears odd, as now with the BOOTING state information, those 11 minutes would seem as having 2 states. But in this case, BOOTING takes a higher priority than CONNECTION_LOSS and therefore, the last 11 minutes of the CONNECTION_LOSS period would be recategorized as BOOTING, as depicted in Figure 2. Although the node still appears down (recall, Slurm's terminology) to the job manager, the node is actually in a BOOTING state

[2]Note this is an example of different *views* (perspective) of DOWN. Compare this with CONNECTION_LOSS which describes the node as unavailable but active. This difference can be important depending on the type of analysis being done.

and therefore is labeled as such. SaNSA also replaces the CONNECTION_LOSS states with DOWN at this point.

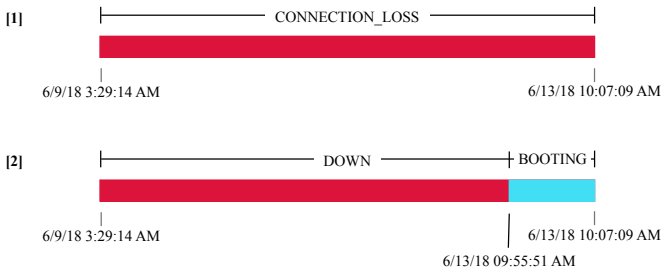

Fig. 2. In pass 1, the period of time from 6/9/18 3:29:14 AM to 6/13/18 10:07:09 AM has been categorized as CONNECTION_LOSS. In pass 2, more information about the BOOTING state has been discovered and part of the time initially categorized as CONNECTION_LOSS is now categorized as BOOTING. This also allows SaNSA to give finer-grained information to the connection loss and recategorize it as a DOWN event. The Slurm Job Manager only offers the perspective of availability, whether it can access the node or not; by incorporating further system log information, we develop a more holistic *system* view, indicating that part of the *unavailable* time is actually during the BOOTING phase of node 84.

The Slurm job logs reveal the times a job starts, ends, and what nodes it runs on as well as the exit code. This exit state can be interesting for analysis separately as has been demonstrated [5] and we save this in SaNSA for future analysis. The following is a single record of a sample response:

```
Time: 3/19/2018, 12:45:10 PM,
Host: gr-master, Ident: logger,
JOBCOMP:"NULL", "grizzly",
"15075159", "JOBCOMP", "2018-03-15
12:32:49", "2018-03-19 01:21:45",
"2018-03-19 01:21:45", "2018-03-19
12:45:10", "1641512160", "userA",
"userA", "standard", "standard",
"40032", "gr[0003-0164, 0166-0376,
0417-0425, 0427-0454, 0540-0547,
0578-0717, 0758-0760, 0833-0885,
0891-1021, 1023-1266, 1268-1297,
1299-1359, 1361-1387, 1486-1490]",
"job_name", "0:0", "SLURM",,,,,,,,,
"COMPLETED", "1112",, "script_name",,
"0:0",,"1:0","", "1641512160","",
"","57600", "0"
```

This record explains that 1112 nodes of Grizzly were running a job from 01:21:45 on 3/19/18 to 12:45:10 on 3/19/18. This allows us to categorize the state of those 1112 nodes of Grizzly as JOB_RUNNING for that time period.

At this point we have "patches" of states in time for many (likely most or all) of the nodes of the cluster(s) under study. We know when the nodes had connection loss to the resource manager and were down, booting, and running jobs. This allows us to draw some very reasonable assumptions about the node being up and, therefore IDLE, during the intervening periods. As such, we fill in the gaps with this time. Data can be cleaned up here by passing these IDLE periods through the system logs for the nodes in question and ensuring that the node was logging information during that time. As is likely obvious, this is challenging and an approximation as HPC systems are tuned to reduce logging and jitter so this step can be a decent estimate and check but there exists no absolute truth. Fundamentally, that is the problem we are facing and efforts like SaNSA and analytics based on them are somewhat fuzzy in nature.

The final phase involves reservation shadowing discussed earlier. SaNSA uses as input a separate reservation data source which defines classes of reservations, the time window, and their priorities. At this time, we have defined these reservations: UNSCHEDULED_RESV, ADMIN_RESV, and USER_RESV. The first two were defined in Section II-B when discussing node states. The USER_RESV reservation state is a special state when portions of a system are dedicated to a groups of users exclusively. This effectively makes a subset of a cluster "private" to them and is used to force particular work through a system at high priority. We rename the state here, RESERVED_IDLE to indicate that the state of the *node* is idle during this time even though it is reserved. Once a user runs a job in this reservation, it transitions into JOB_RUNNING. Effectively one would expect to see a very small amount of time spent in RESERVED_IDLE because, presumably, the users allocated to those cycles would instead be running jobs (JOB_RUNNING). If, instead, the time spent in RESERVED_IDLE was large, it might suggest the dedication to this group was wasted. Having tools like SaNSA to easily calculate this is valuable for a production data center[3].

We now turn from describing the design and architecture of SaNSA to some applications of it on real data at LANL.

III. Applications

The system described previously provides the interface for a number of forms of analysis. We outline some existing uses and the insights that have been provided so far. By first building a core system, SaNSA, that gathers the system and node states over time, we are then empowered to query SaNSA and present different analyses. We show below several different applications and the insights gleaned from them.

A. Anomaly Detection

All the clusters that we have analyzed at this time are homogeneous and that is usually the case for contemporary production data centers. Identifying nodes within a cluster that are deemed anomalous is therefore valuable as it can lead to identifying nodes that may need attention or servicing. Here we look at the Grizzly cluster (see Table I) which runs the TOSS [6] Linux operating system.

For each node i, we compute the value $x_{i,j}$ for each state j where the value $x_{i,j}$ is the percent of time node i spends in state j. We apply a multivariate Gaussian distribution to the data, creating an array of probabilities corresponding to each

[3]We calculated the time wasted in this state and found it to be so low that it was not worth discussing and, therefore, it is not shown as an application of SaNSA in Section III.

node in Grizzly. The lower the probability, the more anomalous the node. We define a value ϵ and say if the probability $p < \epsilon$, then that node is anomalous. This value for ϵ is variable and chosen so that the number of anomalous nodes is roughly 5. It is, however, important to note that anomalous is not equivalent to 'bad' but rather an indication of something rare or unusual. In this case, by setting ϵ to 10^{-61}, we end up with the following anomalous nodes: 7, 426, 1200, 1311, 1319. Figure 3 shows the state transitions for the 5 most anomalous nodes for the ϵ chosen.

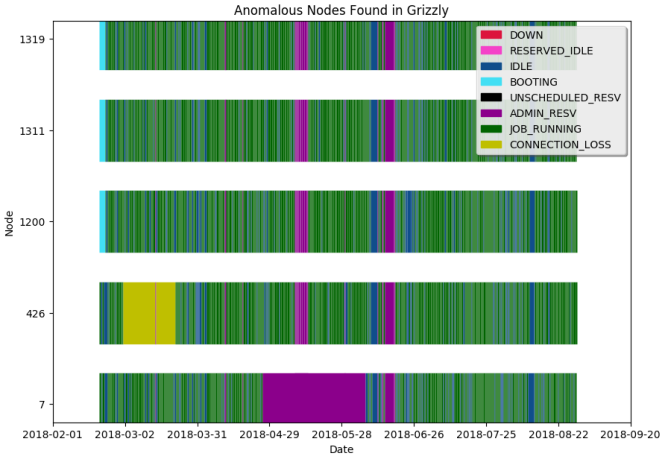

Fig. 3. The 5 most anomalous nodes identified on the Grizzly cluster over the 190 day period using an ϵ of 10^{-61}

Node 426 is identified as anomalous due to the amount of time that is spent in the `CONNECTION_LOSS` state. For the first half of March, 2018 this node appears to be having a connection problem and could not be reached by the Slurm Workload Manager. Figure 4 presents a boxplot of the percent of time Grizzly nodes are in the `CONNECTION_LOSS` state and identifying node 426 as spending over 10% of the time in 190 days in this state. From this, when compared with the other nodes of Grizzly, it is clear why this node was identified as anomalous.

Nodes 1200, 1311, and 1319 are identified as anomalies as well; this can be confirmed in Figure 5 where they appear to spend an relatively high percent of time in a `BOOTING` state. One way to look at these results, however, are that even though this amount of time spent booting is unusual compared to other nodes of Grizzly, they do not in the end account for a particularly large amount of *total* time spent in this state. One might consider that node 426 spending 10% of its time in `CONNECTION_LOSS` is more impactful than these three nodes spending 1.5% of their time `BOOTING`.

Node 7 conspicuously stands out as an anomaly, spending a large amount of time in `ADMIN_RESV`. This can be confirmed in Figure 6 where node 7 is identified as an outlier, over 16 standard deviations above the mean. It would appear that the administrators of Grizzly had node 7 removed from operation for roughly 27% of the time during the 190 days under study.

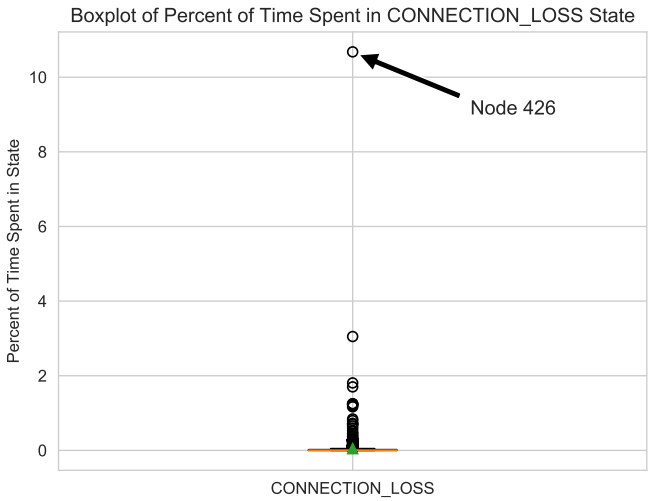

Fig. 4. Boxplot of the percent of time all Grizzly nodes are in a `CONNECTION_LOSS` state. Node 426 is the outlier averaging about 10% of the time in this state while most other nodes spend almost no time in this state (also shown later in Table II).

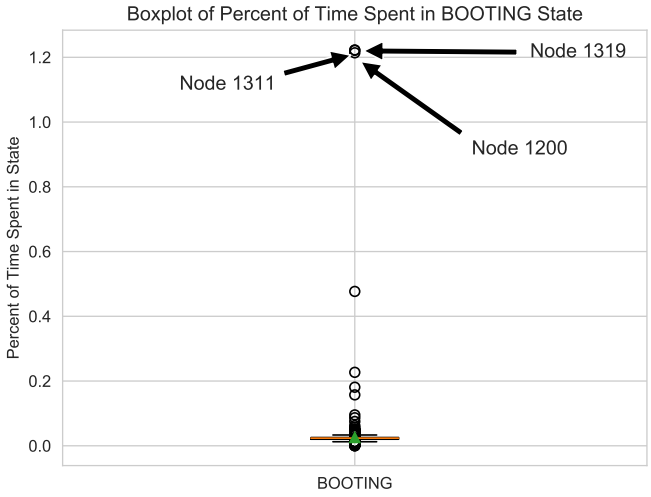

Fig. 5. Boxplot of percent of time Grizzly nodes are in a `BOOTING` state. While three nodes are identified as outliers, the total percent of time these nodes spend `BOOTING` does not account for much.

These examples demonstrate the value of SaNSA in that it allows us to explore other analyses of node and system state to both identify and explain anomalies.

B. Topology and Node Placement in the Data Center

Positional effects in supercomputer data centers can be impactful and meaningful as studies have shown. For instance, in [7], Sridharan shows increased fault rates correlated with the vertical position of nodes in the rack of a large supercomputer. In SaNSA, all nodes can be enhanced with arbitrary attributes and we utilized this capability with the Grizzly cluster to encode a node's data center *row number*, *rack number*, *scalable unit (SU) number*, and its location vertically and horizontally

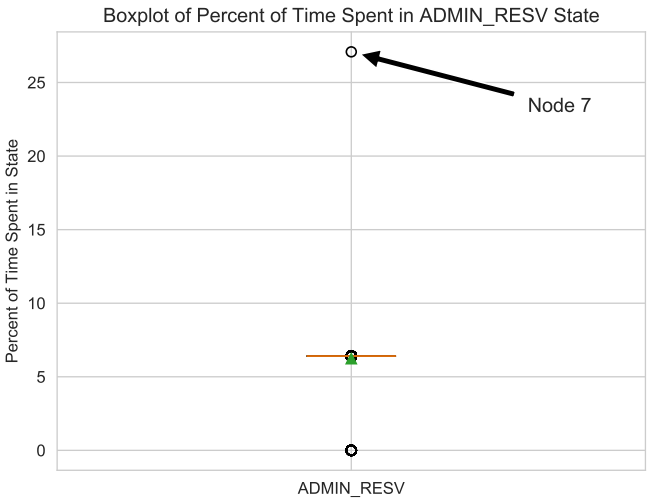

Fig. 6. Boxplot of the percent of time Grizzly nodes are in an ADMIN_RESV state. As identified through the anomaly detection algorithm as an anomaly, Node 7 is over 16 standard deviations above the mean.

within the rack. This allows a positional analysis of the cluster in case this extra information reveals something interesting.

We explored the total time Grizzly nodes spent running jobs as compared to the positioning of the nodes in the data center. The results are shown in three dimensions in Figure 7. One particular rack, number 13, can be clearly identified as receiving about half as many total job hours scheduled than all the other racks on Grizzly.

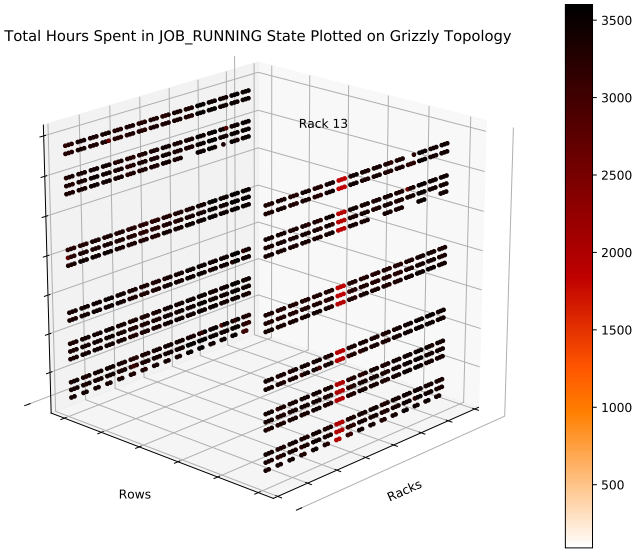

Fig. 7. The 2 rows of Grizzly, comprised of 8 scalable units (SUs), each containing 4 racks for a total of 32 racks. Racks contain between 44 and 47 nodes with blanks clearly observable and by design (not missing data). The plot colors the amount of time each node spent in the JOB_RUNNING state. The orange shows rack 13 has spent considerably less time than other nodes running jobs.

Using SaNSA, we explored this more deeply in Figure 8. Here, rack 13 is singled out as a single line while the other 31 racks are averaged together into another separate line (for clarity). One can clearly see that rack 13 had a period of time where it behaved differently than the other racks but, around May 10th, began receiving approximately similar job workloads. It might be tempting to make assumptions about the state of rack 13's nodes during the anomalous time period however, due to the states we have captured we know precisely the states of those nodes over that time period. Further exploration revealed that the nodes were on for almost that entire time but were IDLE (it seems unlikely Slurm reserved the nodes for such a long period for a pending large job so this is ruled out). HPC administrators were able to confirm merely that rack 13 had cooling problems but were not aware it was not receiving an equal share of jobs. Upon further root cause analysis of specific logs of the nodes in rack 13, we determined that the Slurm configuration file had not been properly propagated to the nodes in the rack. The important result is that SaNSA identified an issue that the administrators were not previously aware of. This is precisely the type of situation where a tool like SaNSA will bring value by drawing attention to anomalies for further investigation by system administrators.

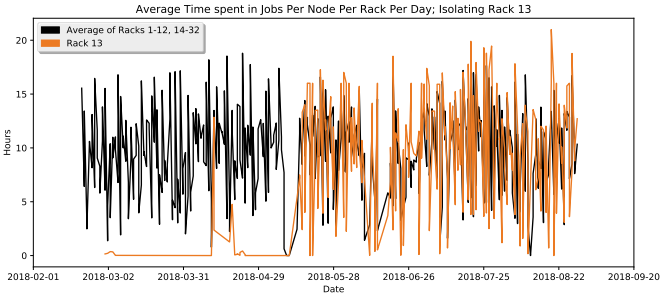

Fig. 8. Grizzly's rack 13 singled out as compared to all other 31 racks of grizzly averaged together into a single line showing the time spent in JOB_RUNNING. Separate analysis showed that rack 13 was *not* DOWN but was IDLE during the period of inactivity where it was receiving a lower amount of jobs.

C. State Transitions

The node states captured inside of SaNSA provide the capability to calculate state transition probabilities. This can be useful for getting a glimpse at system behavior as a whole. As mentioned earlier, we have the capability of overlaying states on top of other states, effectively shadowing and masking the underlying one. We use this for ADMIN_RESV primarily because, in general, we do not want to count events that occur during these administrative outages.

In Figure 9 we show the state transition probabilities for Grizzly without the ADMIN_RESV overlay. The diagram looks very normal and seems to be what one would expect as the nodes transition with high probability between the IDLE and JOB_RUNNING states. Running jobs can be seen to very rarely result in a lost connection or transition into a DOWN node. We look at a few examples of these rare jobs that end in crashed nodes in Section III-E.

TABLE II
PERCENTAGE OF TIME GRIZZLY NODES SPEND IN DIFFERENT STATES OVER 190 DAY OBSERVATION PERIOD. DATA PRESENTED WITH AND WITHOUT ADMIN_RESV OVERLAY STATE.

State	% Time w/o Overlay	% Time w/ Overlay
DOWN	0.46%	0.22%
BOOTING	0.04%	0.03%
IDLE	26.11%	22.80%
JOB_RUNNING	71.11%	70.68%
ADMIN_RESV	*NA*	6.23%
CONNECTION_LOSS	2.28%	0.04%

Missing from this data is the amount of *time* spent in these states but, certainly, this data is contained in SaNSA and this would expand this work to more complicated analysis of Markov processes. This is work in progress and beyond the scope of this paper but we do present a summary of the data in Table II.

We show in Figure 10 how the state transition diagram changes when the ADMIN_RESV overlay is added. The picture of the system becomes more complex as nodes are booted about 5% of the time *into* an ADMIN_RESV state when they need to be serviced. One can also see nodes returned to IDLE when admin reservations end. Coupling Figure 10 with the percentages of times spent in each state shown in Table II (% Time w/ Overlay column) makes it more apparent that while the transition diagram is more complex, most of the time spent by nodes is spent running jobs, idle, or in maintenance by system administrators. This seems like a healthy observation.

Table II allows us to observe that while we see the DOWN state on the diagrams, the nodes of Grizzly are down for between 0.46% and 0.22% of the time period measured - a very small amount. We remind the reader that the IDLE time is misleading as Slurm uses this notion of idle to build large job reservations for jobs that have been waiting in the queue for a long time. However, it does indicate an opportunity for smaller preemptible backfill jobs that could run while Slurm was building these allocations. The time spent in DOWN and CONNECTION_LOSS are further discussed in Section III-D.

These transition probabilities could be used to generate a model of a "normal" node of a system. At this time, we combined all nodes of Grizzly into a single model but we intend to look for clusters of state transitions. We suspect this will be useful in identifying node behaviors quickly.

While it is tempting to make assumptions about Markov processes from these results, this work is not presented here as it is out of scope of this paper. Indeed, SaNSA has the required components to allow for this analysis including the holding time in a state before transitioning. We are exploring continuous and discrete time Markov chain approaches and leave that for future work.

D. Time Between Down and Connection Loss States

It is common to report on the mean time between failures (MTBF) of HPC systems and use this as a measure of system health. Others [8] have shown that this metric may not be the best and may require more nuance such as sliding observation

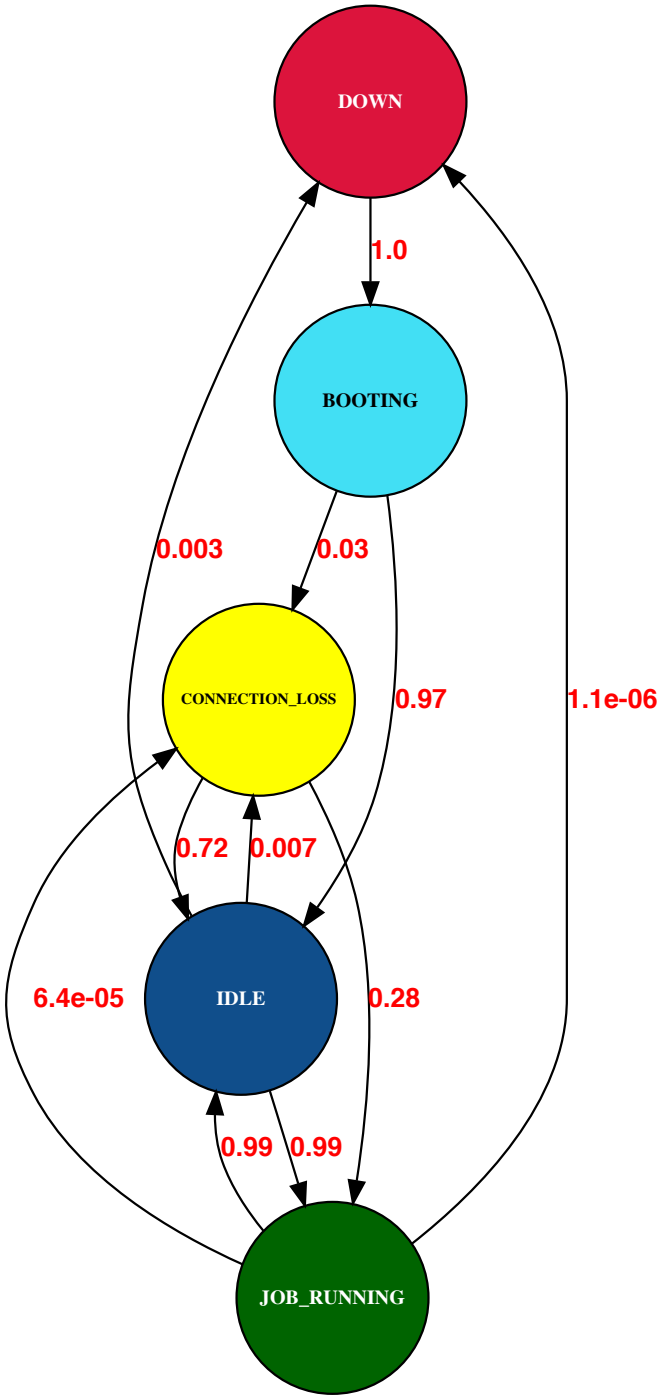

Fig. 9. State transition probabilities of the Grizzly cluster at LANL before the ADMIN_RESV overlay is added. This shows at a glance what would be expected behavior and high probability of transitioning between IDLE and JOB_RUNNING states.

windows. At this time, with SaNSA, we do not have a state specifically outlined as failure but we do have a down state. Furthermore, we have already extracted administrative reboots so there is at least some reason to believe these down events are failure-related. This will be addressed in future work.

In Table III we present the time between DOWN and

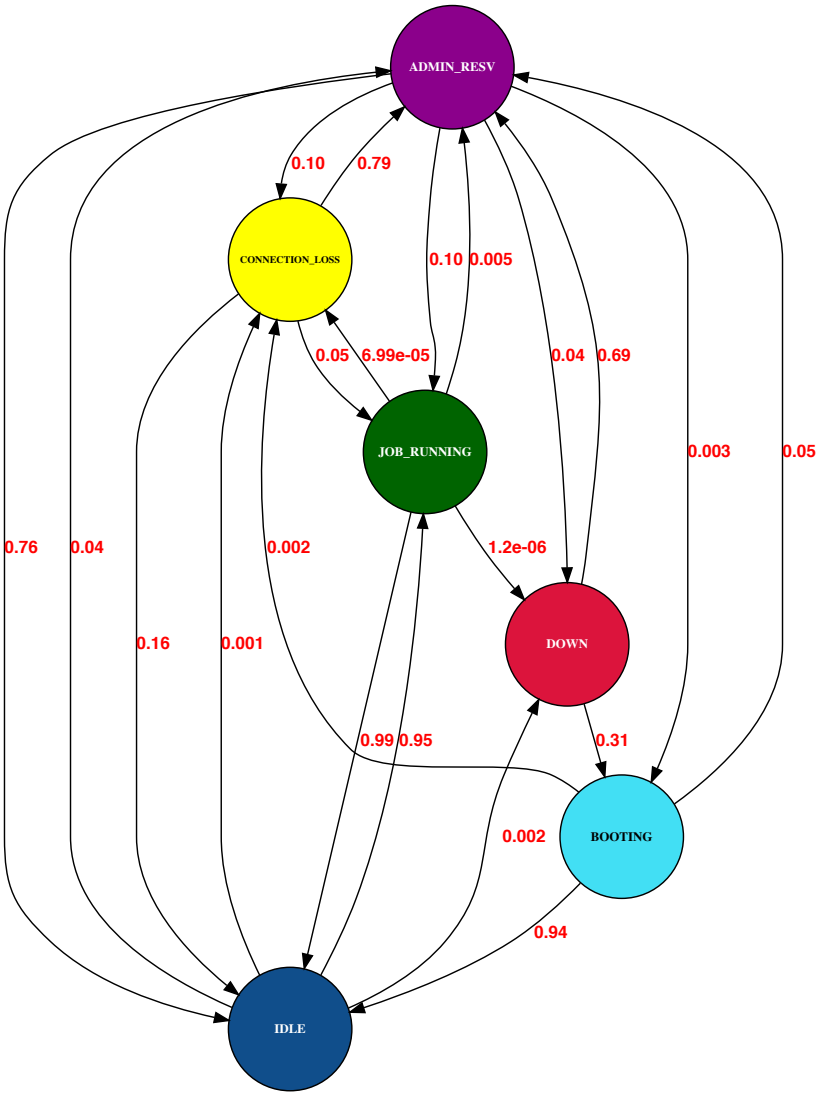

Fig. 10. State transition probabilities of the Grizzly cluster after the ADMIN_RESV overlay is added. Previous states are masked as they now occur during admin reservation windows.

CONNECTION_LOSS states on the four clusters we studied. This data is over the 190 day period previously outlined. Figures 11 and 12 present the down and loss of connection states as boxplots. This is an example of using SaNSA to get a supercomputer perspective based on assembling the individual states of the nodes that make up that supercomputer. The time between the start of a state (such as DOWN) then is the time between the start of that state across all nodes of that supercomputer.

In the state transition analysis previously presented we showed that the DOWN and CONNECTION_LOSS states do not constitute much of the total time of nodes of Grizzly. However, we can see in Table III that the events do *occur* at an interval which would interrupt jobs that spanned the entire cluster. The nodes of Wolf have no DOWN events because the way that system is operated the ADMIN_RESV state overlays the down events. That is, when nodes fail they are placed into an admin reservation queue where system support diagnoses the problem before returning the node to service.

TABLE III
TIME BETWEEN DOWN AND CONNECTION_LOSS ON THE FOUR CLUSTERS STUDIED AT LANL.

Cluster Name	Time Between (hours)			
	DOWN		CONNECTION_LOSS	
	mean	median	mean	median
Grizzly	18.92	5.75	25.80	5.13
Wolf	NA	NA	38.53	3.23
Snow	33.98	5.23	66.50	19.80
Woodchuck	121.34	55.73	344.29	16.41

Woodchuck is a small system which is dedicated to data analytics jobs. The outliers shown in the figures are reasonable given the system sizes and we see that as the systems grow in size, the time between events become shorter.

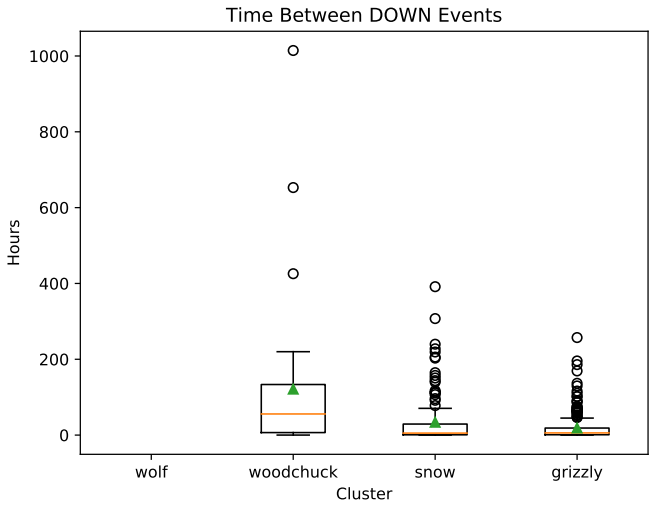

Fig. 11. The mean (triangle) and median (line) between DOWN events for the Woodchuck, Snow, and Grizzly clusters. Wolf has no down events due to the ADMIN_RESV overlay effectively "shadowing" these events as nodes are placed into a special administrative state for error diagnosis.

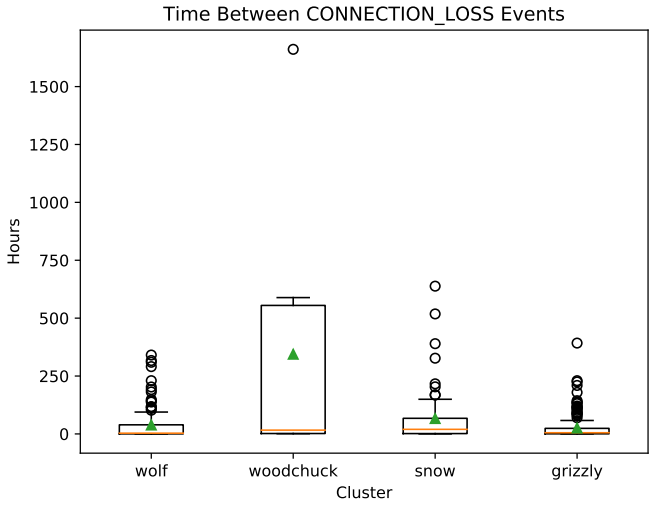

Fig. 12. The mean (triangle) and median (line) between CONNECTION_LOSS events for the Wolf, Woodchuck, Snow, and Grizzly clusters. Slurm appears to lose connectivity with the Grizzly nodes about as frequently as nodes go down. These are distinct and separate states.

E. Correlated Failures

Studies [9] have shown that node failures dominate supercomputer failures in production. While sometimes nodes fail at the same time, or "close" in time (for some fuzzy definition of closeness), these events are thankfully more rare. We consider that these require some sort of shared resources such as network, power backplane, or potentially (in extremely rare circumstances) all running the same job, and call this class of events *correlated failures*.

With SaNSA we can easily look for these events due to the structured nature of the node state data. One event SaNSA found was an instance of an entire rack, number 37 in scalable unit (SU) 8, that went down at the same instant. The 47 nodes in the rack were offline for 6 hours before coming back into operation. Assuming this was some sort of administrative action, we looked through the system logs by hand on these 47 nodes and spoke to the admins and saw no record of this. We do note that the nodes are DOWN and not CONNECTION_LOSS. This indicates that it was not a loss of connectivity to Slurm through some sort of shared networking resource in the rack. We confirmed this in the logs as well. More advanced telemetry such as those present on LANL's Trinity supercomputer in the secure, where SaNSA will be deployed in the coming months, would likely have helped in root cause analysis.

Another event SaNSA discovered was one where 3 racks crashed at the same time. Interestingly, while two of the racks were next to each other (20 and 21 in SU5), the third rack involved was rack 17 in SU4. The way these racks are oriented we are not aware of any shared power resources that could cause this event. Not surprisingly, jobs were running on Grizzly at this time which were impacted by a failure of 18% of the cluster's nodes. In this case, an 80 node job and a 192 node job were killed which stretched across many more nodes of Grizzly. What we then see is an event where (presumably) hardware node failures due to some shared resource impact job failures causing jobs to abort on nodes that remain healthy because the underlying parallel runtime library is not resilient. This would be a case of failure cascading and causing increased harm.

IV. Related Works

Stearley [10] developed a method for evaluating Cray supercomputers using node states. Much like our work, these events were based on specific messages in log files. However, it was specific to Cray supercomputers only at the time, though it is likely it could be extended. They call out the need for administrators to mark scheduled downtime events which is something we address in SaNSA with reservation overlays. While similar on some levels, this related work did not *use* the state machine that was built to detect problems and lead to solutions.

In [11], a study of supercomputer logs is performed which can be used to categorize log messages into a number of categories. Anomaly detection of log messages for supercomputers is of interest to the HPC community [12]–[14]. However, existing approaches mostly focus on looking for anomalies in log files themselves rather than our approach with SaNSA of detecting anomalies via the state of the nodes. While these efforts have similar end goals, we believe our approach is unique and allows for different discoveries.

Schroeder [15] studied failures on almost two dozen high-performance computing systems. They present work on root cause analysis, time between failures, and repair times. Their studies use job workload data and failure event logs. We present here with SaNSA a tool that could easily be used to conduct such a study and have shown similar applications.

With SaNSA, node state can be assembled through a configurable number of data sources instead of just already processed event logs. The events are determined based on specifications of patterns.

In [9] and [8] the Blue Waters and Titan supercomputers are studied respectively. Both analyses focus on failure events and look to do root cause analysis. In [9] link (network) failures are discussed as well as failures that do not cause job loss. This is likely comparable to the `CONNECTION_LOSS` states which are identified in our work with SaNSA.

Production HPC data centers use different tools for monitoring including LDMS [4] for node metrics, RabbitMQ [16] for log message transport, Splunk [17] or Zenoss [18] for log collection and displays, and Baler [19] for analysis. With SaNSA, we consume many of the data sources which are provided by such a monitoring infrastructure. Unlike approaches like Baler, our focus is on the state of the nodes and clusters over time rather than on detecting anomalies in the raw data used as input.

V. Conclusions and Future Work

In this paper we have discussed SaNSA, a software architecture for studying supercomputer and node state over time. The states that SaNSA can capture are defined by the user and can be assigned priorities depending on the perspective the analyst is interested in studying.

We used SaNSA to ingest over 1.1 billion system log lines from 4 production clusters at LANL over 190 days in 2018. These systems are relatively small as most of LANL's production computing resources are in the secure. The Elasticsearch infrastructure used by SaNSA has only recently been fully deployed in this environment and data collection is ongoing. We intend to use SaNSA on LANL's large production CTS systems as well as the Trinity supercomputer (nearly 20,000 nodes) which will provide for both larger data sources as well as larger production resources to analyze.

We have shown that SaNSA can be used to establish node state transition probabilities which may be useful for characterizing a representative node of a cluster. Next, we are looking at both continuous and discrete time Markov chains for use with SaNSA and how they might be applicable for further analysis.

It is our hope that through better understanding of node and supercomputer state over time, not just through real time health monitoring, we will gain an improved understanding of supercomputer reliability and where additional emphasis needs to be placed. In future work we will look at how user views of node state differ from syslog views and how this impacts overall metrics such as utilization.

VI. Acknowledgments

This work was performed at the Ultrascale Systems Research Center (USRC) at Los Alamos National Laboratory, supported by the U.S. Department of Energy contract DE-FC02-06ER25750. The publication has been assigned the LANL identifier LA-UR-18-28612.

References

[1] T. Sterling, D. Becker, M. Warren, T. Cwik, J. Salmon, and B. Nitzberg, "An assessment of beowulf-class computing for nasa requirements: initial findings from the first nasa workshop on beowulf-class clustered computing," in *1998 IEEE Aerospace Conference Proceedings (Cat. No.98TH8339)*, vol. 4, Mar 1998, pp. 367–381 vol.4.

[2] Elasticsearch, August 2018, `https://www.elastic.co/`.

[3] Slurm Workload Manager, August 2018, `https://slurm.schedmd.com/`.

[4] 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, "The lightweight distributed metric service: A scalable infrastructure for continuous monitoring of large scale computing systems and applications," in *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2014, pp. 154–165.

[5] G. Amvrosiadis, J. W. Park, G. R. Ganger, G. A. Gibson, E. Baseman, and N. DeBardeleben, "On the diversity of cluster workloads and its impact on research results," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, 2018, pp. 533–546.

[6] N. Bass, "TOSS - A RHEL-based Operating System for HPC Clusters," U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC - LLNL-PRES-741473. [Online]. Available: www.redhat.com/cms/managed-files/SC17-Red_Hat-TOSS-Final.pdf

[7] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi, "Feng shui of supercomputer memory positional effects in dram and sram faults," in *2013 SC - International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Nov 2013, pp. 1–11.

[8] S. Gupta, T. Patel, C. Engelmann, and D. Tiwari, "Failures in large scale systems: Long-term measurement, analysis, and implications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: ACM, 2017, pp. 44:1–44:12.

[9] C. D. Martino, Z. Kalbarczyk, R. K. Iyer, F. Baccanico, J. Fullop, and W. Kramer, "Lessons learned from the analysis of system failures at petascale: The case of blue waters," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2014, pp. 610–621.

[10] J. Stearley, R. Ballance, and L. Bauman, "A state-machine approach to disambiguating supercomputer event logs," in *2012 Workshop on Managing Systems Automatically and Dynamically, MAD'12, Hollywood, CA, USA, October 7, 2012*, 2012.

[11] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, June 2007, pp. 575–584.

[12] A. Gainaru, F. Cappello, and W. Kramer, "Taming of the shrew: Modeling the normal and faulty behaviour of large-scale hpc systems," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, May 2012, pp. 1168–1179.

[13] S. Di, R. Gupta, M. Snir, E. Pershey, and F. Cappello, "Logaider: A tool for mining potential correlations of hpc log events," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, May 2017, pp. 442–451.

[14] C. D. Martino, S. Jha, W. Kramer, Z. Kalbarczyk, and R. K. Iyer, "Logdiver: A tool for measuring resilience of extreme-scale systems and applications," in *Proceedings of the 5th Workshop on Fault Tolerance for HPC at eXtreme Scale*, ser. FTXS '15. New York, NY, USA: ACM, 2015, pp. 11–18.

[15] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 4, pp. 337–350, Oct 2010.

[16] RabbitMQ, August 2018, `https://www.rabbitmq.com/`.

[17] A. DeConinck and K. Kelly, "Evolution of monitoring over the lifetime of a high performance computing cluster," in *2015 IEEE International Conference on Cluster Computing*, Sept 2015, pp. 710–713.

[18] Zenoss, August 2018, `https://www.zenoss.com/`.

[19] 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, p. 285, Apr 2011.