A State-Machine Approach to Disambiguating Supercomputer Event Logs

Jon Stearley, Robert Ballance, Lara Bauman
Sandia National Laboratories[1]
{jrstear,raballa,lebauma}@sandia.gov

Abstract—**Supercomputer components are inherently stateful and interdependent, so accurate assessment of an event on one component often requires knowledge of previous events on that component or others. Administrators who daily monitor and interact with the system generally possess sufficient operational context to accurately interpret events, but researchers with only historical logs are at risk for incorrect conclusions. To address this risk, we present a state-machine approach for tracing context in event logs, a flexible implementation in Splunk, and an example of its use to disambiguate a frequently occurring event type on an extreme-scale supercomputer. Specifically, of 70,126 heartbeat stop events over three months, we identify only 2% as indicating failures.**

I. Introduction and Related Work

System logs are often the first place system administrators turn when trying to determine the cause of problems in supercomputers. They are also often studied by researchers to understand failure modes and rates of existing systems, towards designing methods to overcome resilience concerns of future systems. Concerns are rising due to socket counts and memory size increasing faster than socket reliability and I/O rates. Components generate event logs, some indicating benign events and others malignant ones, such as memory errors, spurious kernel panics, and the like. The reliability significance a kernel panic during system testing versus one during production operation are very different. The need for such context was described in a study of logs from five supercomputers [1], but significant advances on this front are not apparent in published literature. Instead, studies of event logs mention no consideration of the operational status of the hosts that generated them.

It is widely recognized that a single fault may result in many event messages. The most common approach to determining fault rates from event data is to coalesce events based on their proximity in time, particularly when multiple hosts emit warning messages simultaneously. Approaches include fixed-size sliding time windows [2], dynamic window sizing [3], and validity comparisons [4]. Tarate and others [5] calculate reliability metrics based on coalesced logs. Many studies analyze raw logs collected over months or years [6], during which time scheduled maintenance almost certainly occurred, but no mention is made of removing innocuous events (such as reboots during scheduled maintenance) from the data. Our results show that such consideration can dramatically affect event counts, and thus subsequent reliability metrics.

Several researchers have applied state-machine models to the analysis of event logs. Song used three states (up, down, warm) in a Markov model of system availability, and applied it to logs from the "White" supercomputer [7]. A state machine was also applied to Hadoop logs for the purpose of debugging programs based on anomalous time periods between state transitions [8]. These studies use small models for very specific purposes, and do not address operations context such as periods of scheduled maintenance.

It is our opinion that the calculation of reliability metrics based on individual log messages - in the absence of operational context - is prone to error at least, and "absurd" at worst [1]. This paper describes a framework to track and apply relevant context. In the next Section we present a set of states which we believe meaningfully describe the operational states of components in a variety of supercomputers. Then in Sections III and IV we describe state transition rules and a flexible state machine implementation built using the Splunk log analysis software package. Finally in Section V we provide an example of using the machinery to disambiguate an event type on a Cray XE6 system containing over 9,000 hosts.

II. State Machine Definition

The state hierarchy diagram proposed in [1] provides a framework to categorize time throughout a system's lifetime. However, we believe that three of the proposed states cover the majority of time and significance. In discussions among Sandia, Lawrence Livermore, and Los Alamos National Laboratory staff, there was consensus that three states cover the majority of time and are of greatest interest. From production users' perspective, a system is either available for their use or not. Unavailability is either due to scheduled reasons (such as maintenance or upgrade), or unscheduled (such as system failures). We thus target three categories of time: Production Uptime, Scheduled Downtime, and Unscheduled Downtime. Knowledge of this category for all components over time enables accurate calculation of component and aggregate system reliability metrics [9].

While these categories are useful, they are a gross simplification of actual component states in supercomputers. We thus sought to define a minimum number of finer-grained states which are practical for both capacity and capability high

[1] Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000. This document's Sandia identifier is SAND2012-2144C.

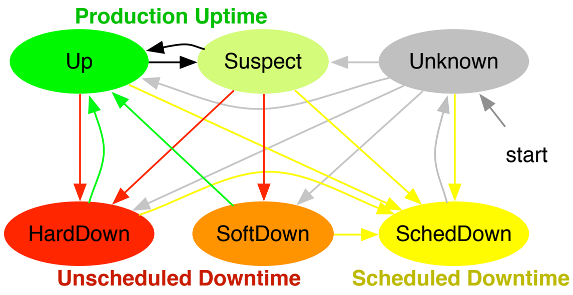

Fig. 1. Each component is in one operational state at any time (indicated by ovals), which group into categories (e.g. Production Uptime). *Unknown* is used when there is insufficient information to assign a different state to components. Red arrows indicate state transitions which we consider to be "failures", and green indicates "repairs".

Operational State	SLURM	MOAB	CRAY
Up	Up	Up	Up
Suspect	Draining	Down	Suspect
SoftDown	Drained, Drained*	Down	Admindown
HardDown	Draining*, Down	Down	Down
SchedDown	(not tracked as a distinct state)		

TABLE I
A MAPPING OF OUR OPERATIONAL STATE NAMES TO SLURM, MOAB, AND CRAY PERSPECTIVES OF HOST STATE.

performance computing (HPC) systems (e.g. clusters which run many small jobs, versus massively parallel "big-iron" system which run a small number of large jobs). We now describe these states, which are shown in Figure 1, and then address details on transitions among them in Section III. While our long-term intent is to track the state of all component types (such as CPUs, network cables, switches, disks, etc), this paper focuses on individual computers only, which we refer to as "hosts".

A. Production Uptime

When all evidence suggests that a host is working properly, we say it is an *Up* state. It must be powered on, booted, and either running a user job or considered by the scheduler as available to run a job. When there is evidence that the host may be unhealthy, but not enough to conclude that it is unusable, we say it is in a *Suspect* state. It may or may not be running a job. Schedulers do not kill jobs on such hosts, but will not start new jobs on them either. It is appropriate for schedulers to be conservative in this respect - the authors have observed entire queues of jobs be quickly emptied onto failed nodes (job start, fail, iterate). When this occurs, all jobs fail, users clean up and resubmit, and system administrators apologize. On the other hand, incorrectly setting hosts as *Suspect* can lead to inappropriate delays in running jobs (which users definitely take notice of as well).

MOAB and SLURM are widely used software packages which schedule jobs and allocate nodes. Capability-class systems (such as Cray XE6) tend to have additional software for monitoring component states. Each of these software elements have their own notion and terminology for host state, which are compared in Table I. MOAB is a job scheduler, so "Down" from its perspective means that a host is not scheduleable, meaning that a new job should not be started on it. SLURM uses the term "Draining" for hosts already running a job, but are *Suspect* and should not receive new jobs. Our use of *Suspect* matches Cray's use.

B. Scheduled Downtime

Maintenance is inevitable - nothing lasts forever. It is not unusual or unacceptable for HPC hosts and complete systems to periodically be set aside for maintenance, upgrade, testing, and the like. Since system administrators are unavoidably involved in such activities, they interpret event logs during such periods with that in mind. If a new software release is being tested, new and interesting failure modes may occur, but they are easily attributed to the testing. They are simply indicators that the software is not ready for production deployment. Hosts may be powered on and off repeatedly in the process, but again these are insignificant from a system reliability perspective.

A primary problem for those analyzing historical logs is that such periods are not known to them. Monitoring nodes do not know why host operating systems stop, just that they did. Permanent records of *Scheduled Downtime* may exist in archived emails, spreadsheets, text files, or databases, but this is site specific. Furthermore, not all sites keep detailed records. The precision of records also varies - did the *Scheduled Downtime* start at precisely at 6:00 AM, or just sometime in the first half of the 6AM hour?

In order to get such events into the logs, the practice at our site is for administrators to use a MOAB "reservation" upon unitiating *Scheduled Downtime*. The reservation must also follow a consistent naming convention (such as PreventativeMaintenance, other examples appear in Table III). Some sites use a similar mechanism in SLURM. The result is that the exact start and stop times of the downtime are permanently recorded, along with the exact list of hosts which were reserved for the activity. This is a key enabler for the event disambiguation described in Section V.

C. Unscheduled Downtime

"Failure is not optional - it is bundled in the software" (Unknown attribution). The underlying sentiment in this humorous quip is that failure is also inevitable. From the user perspective, it does not matter if a host operating system panic'd, a power supply short-circuited, cooling fan bearing wore out, or an administrator's shutdown command was issued in the incorrect terminal window ("oops, I meant to do that on the test system, not the production one") - resources are unexpectedly unavailable. Event logs detail such events, but require operational context information for accurate interpretation of event significance.

If a host in an *Up* state crashes, loses power, or otherwise suddenly becomes unexpectedly unavailable, we say it has transitioned to a *HardDown* state. If a job was running on the host at the time of transition, it is harshly terminated (e.g. manually or automatically). In contrast, if a graceful transition to unexpectedly unavailable occurs, we say it is in a *SoftDown* state. An example of this would be a host running a job, and

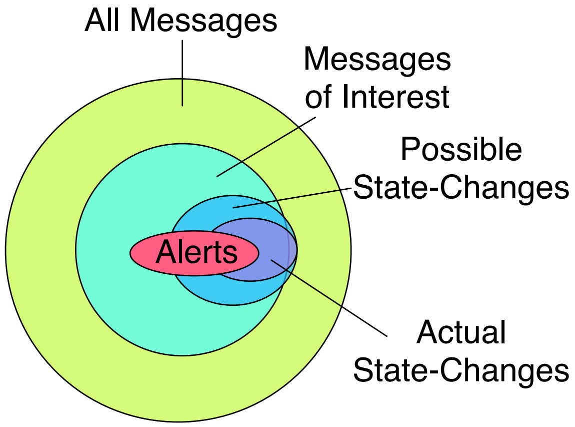

Fig. 2. Of all message types, only a portion of them are of interest to system administrators and regularly reviewed in raw or nightly summaries. The subset matching the rules in Table II indicate possible state changes. Administrators want active notification (e.g. via email or pager) for a very small set of messages, for which we use Splunk's "Alert" mechanism.

either automatically or manually set to *Suspect* due to some symptom. The job is allowed to complete normally (minutes or hours later), and then the host is automatically transitioned to *SoftDown*, awaiting administrator investigation.

Returning to Table I, our *SoftDown* matches Cray's "Admindown" - the host is not running a job nor available for new jobs ("Down" in MOAB). SLURM refers to this state as "Drained". If a host is no longer communicating with SLURM it appends an asterisk "*" to the label. So "Drained*" indicates a host which was draining, then the job ended, and now it is not communicating. If instead the host was draining, and then it suddenly stopped communicating, SLURM will identify it as "Draining*", meaning that a job was killed in the process. If a host was not running a job and stopped communicating, SLURM refers to it as simply "Down".

The meaning of "up" and "down" depends on perspective, be it software or person - a user and administrator can mean quite different things by saying "the host is down." We have attempted to map SLURM, MOAB, and Cray perspectives to a minimal but sufficient set of reasonably intuitive and unambiguous operational state names.

III. State Transition Rules

An integral part of our state machine is the creation of rules that define possible state transistions based on information from administrators and message logs.

A. Messages of Interest

We notionally visualize different classes of log messages in Figure 2. Of the many types of messages, only some are interesting to administrators, which we creatively refer to as "Messages of Interest" (MOI)[1]. MOI examples include login failures, kernel panics, disk failures, memory failures, filesystem errors, etc. Such events are operationally reviewed as they occur, or in daily summary form. A very small set of messages are of sufficient interest that administrators desire active notification via email or pager - we refer to these simply as "Alerts" (matching Splunk's use of that term).

Researchers generally focus on MOI, but non-MOI messages are necessary in order to disambiguate them as actually significant or not. For example, administrators initiating a scheduled downtime with a MOAB reservation command results in a `MOABRSVSTARTSYS` log message. This message is generally not of interest to the administrators, but it is critically important for historical review. Another example is a log message indicating a job start. This happens nearly constantly during the normal operation of the system, so administrators do not generally review them. However, the start of a job may be the first indicator that a host has entered an *Up* state. This explains why the state change regions of Figure 2 are not completely contained in the MOI region.

B. Creating Transistion Rules

Figure 1 shows all the possible transistions in our state machine. *Unknown* is a special state. All hosts begin in *Unknown*. The first matched event type for a host in *Unknown* will trigger a transition to the corresponding new state, thus *Unknown* is fully connected outward.

Note that *SchedDown* is fully connected inward, but has a single outgoing transition to *Unknown*. In actuality, most hosts are brought to a fully functional state before the end of a scheduled downtime. However, some hosts are not, such as those needing additional hardware repairs. Since multiple reboots and test jobs may occur during a scheduled downtime, it is not appropriate to count hosts as entering an *Up* state until after a `MOABRSVENDSYS` event is observed. These correspond to the *Up* group in Table II. Tracking of additional states within *SchedDown* is possible (such as if the host is booted or scheduleable), but we elected to keep the state machine as simple as possible for this initial work. In practice, most hosts quickly begin running jobs after `MOABRSVENDSYS`.

Also note that there is no transition from *SoftDown* to *HardDown*. Once a host has gracefully entered an unscheduled downtime state, it is unavailable to users whether it is rebooted or not. So (for example) it is inappropriate to tally `CRAYHEARTBEATSTOPALERT` as additional failures.

A single state transition may be indicated by a variety of messages, but messages can be corrupted or missing due to system stress or unreliable transimission (e.g. UDP). For this reason, we group multiple event types into state transition rules, and trigger upon the first observed match. This is accomplished by a logical OR of the message types shown in Table II, as described in Section IV.

These rules were created and refined in collaboration with system administrators, towards identifying operationally meaningful state changes.

IV. Implementation in Splunk

This section describes the design and implementation of our state machinery. The implementation is general, allowing

[1] Messages that are squelched at the source therefore become "messages of disinterest." It's not uncommon to discover, when debugging a problem, that the wrong filters are being applied at the source.

New State	Event Type	Source	Interpretation
Up	MOABJOBSTART	Moab	A Job is starting.
	MOABSCHEDULEABLE	Moab	Scheduler deems the host to be available for scheduling.
	CRAYBOOTED	Syslog	A host has completed its boot-up process.
	CRAYAVAIL	Cray	Cray management software deems the host as available for scheduling.
	CRAYHEALTHUP	Syslog	The "node health checker" has set a compute host to the available state, e.g. after the host has entered a Suspect state and extensive checks were run on the system before risking starting additional jobs on it.
Suspect	MOABUNSCHEDULEABLE	Moab	Scheduler deems the host to be unavailable for new jobs.
	CRAYHEARTSTOPWARN	Cray	A host heartbeat has gone silent, indicating that the host might be dead.
	CRAYSUSPECT	Syslog	The "node health checker" has set a compute host to suspect because the host failed a health test.
SoftDown	CRAYADMINDOWN	Syslog	The "node health checker" (or system administrator) has marked a host as warranting additional attention before it should be trusted to run new jobs.
HardDown	CRAYHEARTSTOPALERT	Cray	A host heartbeat has gone silent for long enough that the host is considered to be dead, or there are directly observable signs that the host has died, such as well-known error codes in the heartbeat memory location. Cray management software will automatically kill dependent jobs, halt scheduling, or similar actions depending on the criticality of the declared dead host. See Table III for an example.
	KERNEL_PANIC	Cray	A host kernel has panic'd (died). Most such events come through the Cray logs, there are are some hosts for which kernel panics can only be seen via syslogs.
SchedDown	MOABRSVSTARTSYS	Moab	A system reservation has started. Typically used to designate the start of a system maintenance period. See Table III for an example.
Unknown	MOABRSVENDSYS	Moab	A system reservation is complete.

TABLE II
EVENT TYPES INDICATING POSSIBLE STATE TRANSITIONS.

New State	**Event Type**	**Splunk syntax**	`example message, ... indicates truncation due to space limitation`
HardDown	**CRAYHEARTSTOPALERT**	**sourcetype=cray ec_heartbeat_stop "considered dead"**	`2012-05-29 12:55:45 ec_heartbeat_stop src:::c13-1c2s1 seqnum:0x0 svc:::c13-1c2s1n1(131276)[alert node heartbeat fault] Cause:ec_null Text:node c13-1c2s1n1 considered dead, last heartbeat 00025e67, HT cave indicates a HT link is down, HT lockup criteria met, HT lockup reset is on, socket 0 core 0 bank 0 status 0xb600000000000185 addr 0x0000000000000400 ...`
SchedDown	**MOABRSVSTARTSYS**	**sourcetype=moabstats etype=rsvstart rsvid=PreventMaint.* OR rsvid=system* OR rsvid=*-sys.***	`05:24:50 1338290685:2551224 rsv PreventMaint.2 RSVSTART 1338290660 1338290660 1538290660 8894 8894 0.000000 0.000000 92,93,2,3,9212,...(every nid appears individually, we have truncated 9000+ nids here) - RSV=%=PreventMaint.2=; Other - -`

TABLE III
MORE DETAILS ON TWO EVENT TYPES OF PARTICULAR INTEREST IN THIS PAPER.

arbitrary state machines to be constructed entirely through normal Splunk configuration layers.

Splunk-specific terms are *italicized* so interested readers can efficiently find more information in Splunk documentation. We chose to implement state tracking within the Splunk log analysis platform, because our system administrators already use it for capture, storage, and analysis of all site HPC logs. It has become our site tool of choice due to its range of functionality and extensibility. Interested readers are welcome to download our "HPC" *app* from http://splunkbase.splunk.com.

Figure 3 provides an overview of the *app* design as an layered stack, colored to correspond with message classes in Figure 2. Messages originate from many different sources and are collected at a single point of observation where data reduction and event correlation can occur. Downstream from that observation point lies data analysis. Upstream lie the various components, each configured to emit various messages to be sent out for capture, storage, and analysis. Some messages are about the component where the message originates, while others are about remotely observed components (e.g. from monitoring devices). This distinction often results in components being referred to in different name spaces. At the bottom layer of Figure 3, all messages are stored in Splunk *index*es. Our convention is to use one *index* per HPC system, where each system may contain thousands of hosts. *Eventtype*s are used to define regular expressions which identify messages of interest, such as Out of Memory conditions (OOM) or Machine Check Errors (MCE). *Eventtype*s can also be logical combinations of other *eventtypes*, and we use this to group the set of message types which indicate possible state transitions. As a result, a simple search for *eventtype*=*cos_** yields all such messages within the search time window. *Lookup*s and

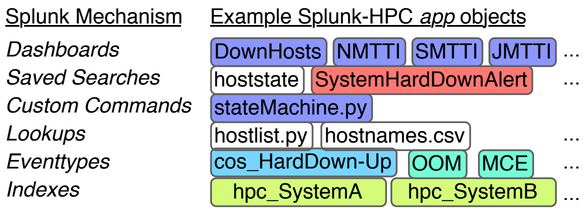

Fig. 3. An overview of Splunk-HPC's design, depicted as a layered analysis stack, with example objects at each layer. Colors match those in Figure 2, indicating the class of information in example objects. The horizontal position of objects is insignificant.

custom commands will be described below. *Search*es extract messages from the indexes and optionally perform a variety of analysis operations, and can be stored as *saved search*es for repeated use. Finally, *dashboards* collect multiple searches onto interactive web pages. For example, our *app* includes dashboards to explore various reliability metrics, or list hosts in a *HardDown* or *SoftDown* state. On Upon clicking a host name, all event logs pertaining to that host within a user-selectable time window appear in a new browser window.

A. Host Namespace Translation via *Lookups*

As mentioned in Section III, different log messages refer to hosts using different namespaces. For instance Cray logs typically refer to hosts by a physical address such as `c6-1c4s5n3`, indicating the host in cabinet 6, row 1, chassis 4, slot 5, node 3 (node and host are generally synonyms in HPC culture). However, when that host emits a syslog message, it identifies itself as `nid09163`. MOAB refers to hosts in nid space (node id), but writes them in "hostlist" format. An example of this is `[1-2342,4356-5542]*8`, which indicates that 8 processes were started on nids `1-2342` and `4345-5542`. The Lustre filesystem uses a yet different namespace for hosts, with names changing upon every mount request (details are omitted due to space constraints). Splunk *lookup*s provide a mechanism to map between these, such that a consistent namespace is automatically available to higher layers in our analysis stack. It is hard to overstate the usefulness of lookups for this and other purposes.

B. State Machinery via a *Custom Command*

We implemented a state machine in 190 lines of Python, which is Splunk's supported scripting language. A *search* is executed resulting in all possible state transition messages, Splunk adds the appropriate *eventtype* to each, which we have formatted as `cos_oldState-newState`, ensures common namespace via *lookups*, and passes them up the analysis stack to the Python script. The script then parses each `oldState` and `newState`, interpreting the corresponding events as indicating a possible state transition of each message's host. By simply defining new *eventtype*s (e.g. via regular expressions matching relevant messages) with `cos_` as their prefix, arbitrary state machines can be constructed.

For example, imagine that the first message seen about host 5 is `CRAYBOOTED`, which matches eventtypes `cos_SoftDown-Up` and `cos_HardDown-Up`. Since the host was in an *Unknown* state, the script treats the messages as indicating a transition from *Unknown* to *Up*, and passes that message to higher layers. Next, imagine that a `MOABJOBSTART` message is observed, having *eventtypes* `cos_HardDown-Up` and `cos_SoftDown-Up`. Since the host is already in an *Up* state, the message does not indicate a state transition (there is no matching *eventtype* with `Up` as the `oldState`). It is therefore not passed to higher analysis layers. The `stateMachine.py` custom command acts as a filter: many messages come in as input, but only those that actually indicate state transitions are output. The custom command performs the set reduction from possible state-change messages to actual state-change messages (Figure 2).

For ongoing monitoring of host states, we have configured the `hoststate` *saved search* to run automatically every 5 minutes. This first pulls the last known state of hosts from a *summary index* dedicated to this purpose, which seeds the host states. Next it applies the state machinery to messages occuring since the last write to the *summary index*, and then writes the actual state transitions (with augmented information) into the *summary index*. The contents of that index thus indicate actual state transitions, for evaluation or use in other searches, e.g. "show MOI of Up hosts" or the analysis in Section V. The saved search can also be invoked manually, such as in the aforementioned `DownHosts` *dashboard* ("which hosts are currently Down?").

C. Seeking Validation by Domain Knowledge Experts

No one understands the failure modes and states of a system like the administrators who take care of it on a daily basis. If administrators depend on our *app* to determine host states, the state record will be explicitly validated. We state this a goal, not an accomplishment. Administrators already had multiple tools to determine host state (e.g. SLURM, MOAB, etc), so the machinery described herein is primarily of research rather than operational use as of this writing. This is initial work, development and validation is ongoing.

We aim to provide multiple compelling features [10], such that the *app* becomes the preferred tool to determine host state and determine root causes of downtimes. Evidence of progress on this front appears in our search logs: after a year of production use, we average about 3,000 searches total per month, by 10+ administrators across six HPC systems.

V. Event Disambiguation

We now apply our state machine to the disambiguation of `CRAYHEARTSTOPALERT` events in a 9,000+ host Cray XE6 supercomputer, over several months of 2012. These are visualized in Figure 4. A `CRAYHEARTSTOPALERT` event pertaining to host in a production uptime state indicates that the software kernel on the host stopped incrementing a well-known memory location (which is monitored by another host). Causes for this include loss of power and kernel panics which immediately kill any job on the host, and may appropriately considered a "failure". However, the event can also occur as a result of a normal reboot by system

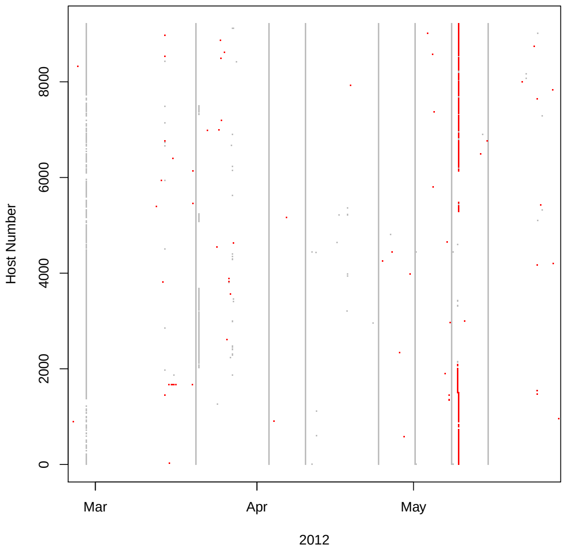

Fig. 4. We visualize each CRAYHEARTSTOPALERT event with a single dot, indicating the time of the event and the host it pertains to. Red dots indicate HardDown state transitions, but grey dots do not - they indicate only innocuous events during a SchedDown state (e.g. administrator-initiated reboots). Regularly spaced vertical grey lines correspond to scheduled downtimes; the red line at May 9 indicates a bad day.

administrators. Scheduled Downtimes often, but not always, involve reboots. In the absence of MOABRSVSTARTSYS and MOABRSVENDSYS events to indicate the start and end of Scheduled Downtimes, researchers might look at the total count of CRAYHEARTSTOPALERT events within small time windows. If the count is high and if these occur at regular intervals, the simple implication is that these correspond to scheduled downtimes. While this is accurate to first order, wide spread failures might be miscategorized. Such an event occurred on May 9, visible as the vertical red line in Figure 4. In contrast, on March 20 there was a secondary reboot of many hosts during a scheduled downtime, visible as a partial vertical grey line to the right of the longer grey line. The tracking of operational context decreases the ambiguity as to which events are failures and which are not.

Also visible in Figure 4 are non-periodic events, such as the sequence of failures on host 1833 in mid March (short horizontal red line), and apparently randomly distributed failures across various hosts and times. Of the 70,126 total CRAYHEARTSTOPALERT events during this time period, only 1,784 indicated host failures (2%), and 1,708 of those were on May 9 which could reasonably be considered a single catastrophic system failure. This leaves 76 single host failure events.

Plugging these differing counts into mean time to failure equations or model-fitting analyses would yield dramatically different results. Thorough disambiguation of logs is a prerequisite for the development of accurate failure metrics and models based on historical logs.

This section provides a proof of concept of our approach, in future publications we plan to examine additional event types on XE6 and other architectures, and explicit validation by administrators. Future papers will also emphasize analysis results, including reliability metrics and failure models, rather than the approach machinery which is the focus of this paper.

VI. Conclusions

We have presented a flexible framework for tracing operations context in event logs. The framework is not specific to the state machine in this paper, nor supercomputer logs. We have demonstrated the utility of using a state machine approach to analyzing logs, specifically for the disambiguation of event significance. In our example case, the small effort by system administrators to demark scheduled downtimes with MOAB reservations is essential, as it creates permanent and precise records enabling disabmiguation. It is our hope that other sites and researchers will see value in this practice and our state machine approach.

Acknowledgments

The authors would like to thank the following people. Sandia HPC cluster administrators Jerry Smith, Aron Warren, Chris Beggio, Ken Lord, Sophia Corwell, for data and discussion of how to interpret it, and Cray systems architect Greg Woodman for discussion of applying these techniques to the XE6 platform.

References

[1] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *Proceedings of the Intl. Conf. on Dependable Systems and Networks (DSN)*, 2007.

[2] Y. Liang, Y. Zhang, M. Jette, A. Sivasubramaniam, and R. K. Sahoo, "Blue gene/l failure analysis and prediction models," in *Proceedings of the Intl. Conf. on Dependable Systems and Networks (DSN)*, 2006. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/DSN.2006.18

[3] A. Pecchia, D. Cotroneo, Z. Kalbarczyk, and R. K. Iyer, "Improving log-based field failure data analysis of multi-node computing systems," in *Proceedings of the Intl. Conf. on Dependable Systems and Networks (DSN)*. Los Alamitos, CA, USA: IEEE Computer Society, 2011.

[4] C. D. Martino, M. Cinque, and D. Cotroneo, "Assessing time coalescence techniques for the analysis of supercomputer logs," in *Proceedings of the Intl. Conf. on Dependable Systems and Networks (DSN)*, 2012.

[5] N. Taerat, N. Naksinehaboon, C. Chandler, J. Elliott, C. Leangsuksun, G. Ostrouchov, S. Scott, and C. Engelmann, "Blue gene/l log analysis and time to interrupt estimation," in *Availability, Reliability and Security, 2009. ARES '09. International Conference on*, 2009.

[6] B. Schroeder and G. Gibson, "The computer failure data repository," 2007, online at http://cfdr.usenix.org/.

[7] H. Song, C. Leangsuksun, and R. Nassar, "Availability modeling and analysis on high performance cluster computing systems," in *Proceedings of the First International Conference on Availability, Reliability and Security*, ser. ARES '06. Washington, DC, USA: IEEE Computer Society, 2006. [Online]. Available: http://dx.doi.org/10.1109/ARES.2006.37

[8] J. Tan, X. Pan, S. Kavulya, R. G, and P. Narasimhan, "Salsa: Analyzing logs as state machines," in *Proceedings of the First USENIX Workshop on the Analysis of System Logs*, 2008.

[9] J. Stearley, "Defining and measuring supercomputer Reliability, Availability, and Serviceability (RAS)," in *Proceedings of the 2005 Linux Clusters Institute Conference*, 2005, see http://www.cs.sandia.gov/~jrstear/ras.

[10] J. Stearley, S. Corwell, and K. Lord, "Bridging the gaps: joining information sources with splunk," in *Proceedings of the 2010 workshop on Managing systems via log analysis and machine learning techniques*, ser. SLAML'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 8–8. [Online]. Available: http://dl.acm.org/citation.cfm?id=1928991.1929003