What Supercomputers Say: A Study of Five System Logs

Adam Oliner[1]
Stanford University
Department of Computer Science
Palo Alto, CA 94305 USA
oliner@cs.stanford.edu

Jon Stearley[2]
Sandia National Laboratories
Albuquerque, NM 87111 USA
jrstear@sandia.gov

Abstract

If we hope to automatically detect and diagnose failures in large-scale computer systems, we must study real deployed systems and the data they generate. Progress has been hampered by the inaccessibility of empirical data. This paper addresses that dearth by examining system logs from five supercomputers, with the aim of providing useful insight and direction for future research into the use of such logs. We present details about the systems, methods of log collection, and how alerts were identified; propose a simpler and more effective filtering algorithm; and define operational context *to encompass the crucial information that we found to be currently missing from most logs. The machines we consider (and the number of processors) are: Blue Gene/L (131072), Red Storm (10880), Thunderbird (9024), Spirit (1028), and Liberty (512). This is the first study of raw system logs from multiple supercomputers.*

1 Introduction

The reliability and performance challenges of supercomputing systems cannot be adequately addressed until the behavior of the machines is better understood. In this paper, we study system logs from five of the world's most powerful supercomputers: Blue Gene/L (BG/L), Thunderbird, Red Storm, Spirit, and Liberty. The analysis encompasses more than 111.67 GB of data containing 178,081,459 alert messages in 77 categories. The system logs are the first place system administrators go when they are alerted to a problem, and are one of the few mechanisms available to them for gaining visibility into the behavior of the machine. Particularly as systems grow in size and complexity, there

is a pressing need for better techniques for processing, understanding, and applying these data.

We define an *alert* to be a message in the system logs that merits the attention of the system administrator, either because immediate action must be taken or because there is an indication of an underlying problem. Many alerts may be symptomatic of the same *failure*. Failures may be anything from a major filesystem malfunction to a transient connection loss that kills a job.

Using results from the analysis, we give recommendations for future research in this area. Most importantly, we discuss the following issues:

- Logs do not currently contain sufficient information to do automatic detection of failures, nor root cause diagnosis, with acceptable confidence. Although identifying candidate alerts is tractable, disambiguation in many cases requires external context that is not available. The most salient missing data is *operational context*, which captures the system's expected behavior.

- There is a chaotic effect in these systems, where small events or changes can dramatically impact the logs. For instance, an OS upgrade on Liberty instantaneously increased the average message traffic. On Spirit, a single node experiencing disk failure produced the majority of all log messages.

- Different categories of failures have different predictive signatures (if any). Event prediction efforts should produce an ensemble of predictors, each specializing in one or more categories.

- Along with the issues above, automatic identification of alerts must deal with: corrupted messages, inconsistent message structure and log formats, asymmetric alert reporting, and the evolution of systems over time.

Section 3 describes the five supercomputers, the log collection paths, and the logs themselves. Section 3.2 explains the alert tagging process and notes what challenges will be

faced by those hoping to do such tagging (or detection) automatically. Section 4 contains graphical and textual examples of the data and a discussion of the implications for filtering and modeling. Finally, Section 5 summarizes the contributions and our recommendations.

The purpose of this paper is not to argue for a particular reliability, availability, and serviceability (RAS) architecture, nor to compare the reliability of the supercomputers. The systems we study are real, and the logs are in the form used by (and familiar to) system administrators. Our intention is to elucidate the practical challenges of log analysis for supercomputers, and to suggest fruitful research directions for work in data mining, filtering, root cause analysis, and critical event prediction. This is the first paper, to our knowledge, that has considered raw logs from multiple supercomputing systems.

2 Related Work

Work on log analysis and large-systems reliability has been hindered by a lack of data about their behavior. Recent work by Schroeder [21] studied failures in a set of cluster systems at Los Alamos National Lab (LANL) using the entries in a *remedy database*. This database was designed to account for all node downtime in these systems, and was populated via a combination of automatic procedures and the extensive effort of a full-time LANL employee, whose job was to account for these failures and to assign them a cause within a short period of time after they happened. Schroeder also examined customer-generated disk replacement databases [22], but there was no investigation into how these replacements were manifested in the system logs. Although similar derived databases exist for the supercomputers considered in this paper, our goal was to describe the behavior of the systems rather than human interpretations.

There is a series of papers on logs collected from Blue Gene/L (BG/L) systems. Liang, et al [10] studied the statistical properties of logs from an 8-rack prototype system, and explored the effects of spatio-temporal filtering algorithms. Subsequently, they studied prediction models [9] for logs collected from BG/L after its deployment at Lawrence Livermore National Labs (LLNL). The logs from that study are a subset of those used in this paper. Furthermore, they identified alerts according to the *severity* field of messages. Although it is true that there exists a correlation between the value of the severity field of the message and the actual severity, we found many messages with low severity that indicate a critical problem and vice versa. Section 3.2 elaborates on this claim and details the more intensive process we employed to identify alerts.

System logs for smaller systems have been studied for decades, focusing on statistical modeling and failure prediction. Tsao developed a *tuple* concept for data organiza-

tion and to deal with multiple reports of single events [26]. Early work at Stanford [13] observed that failures tend to be preceded by an increased rate of non-fatal errors. Using real system data from two DEC VAX-cluster multicomputer systems, Iyer found that alerts tend to be correlated, and that this has a significant impact on the behavior and modeling of these systems [25]. Lee and Iyer [8] presented a study of software faults in systems running the fault-tolerant GUARDIAN90 operating system. The task of automatically discovering alerts in log data has been explored from a pattern-learning perspective [7]. There have also been efforts at applying data mining techniques to discover trends and correlations [12, 23, 27, 28].

In order to solve the reliability and performance challenges facing supercomputer installations, we must study the machines as artifacts, characterizing and modeling what they *do* rather than what we expect them to do. By number of processor hours ($\sim$774 million), this is the most extensive system log study to date.

3 Supercomputer Logs

The broad range of supercomputers considered in this study are summarized in Table 1. All five systems are ranked on the Top500 Supercomputers List as of June 2006 [2], spanning a range from #1 to #445. They vary by two orders of magnitude in the number of processors and by one order of magnitude in the amount of main memory. The interconnects include Myrinet, Infiniband, GigEthernet, and custom or mixed solutions. The various machines are produced by IBM, Dell, Cray, and HP. All systems are installed at Sandia National Labs (SNL) in Albuquerque, NM, with the exception of BG/L, which is at Lawrence Livermore National Labs (LLNL) in Livermore, California.

3.1 Log Collection

It is standard practice to log messages and events in a supercomputing system; no special instrumentation nor monitoring was added for this study. Table 2 presents an overview of the logs. The remainder of this section focuses on the infrastructure that generated them.

On Thunderbird, Spirit, and Liberty, logs are generated on each local machine by `syslog-ng` and both stored to `/var/log/` and sent to a logging server. The logging servers (`tbird-admin1` on Thunderbird, `sadmin2` on Spirit, and `ladmin2` on Liberty) process the files with `syslog-ng` and place them in a directory structure according to the source node. We collected the logs from that directory. As is standard syslog practice, the UDP protocol is used for transmission, resulting in some messages being lost during network contention.

System	Owner	Vendor	Top500 Rank	Procs	Memory (GB)	Interconnect
Blue Gene/L	LLNL	IBM	1	131072	32768	Custom
Thunderbird	SNL	Dell	6	9024	27072	Infiniband
Red Storm	SNL	Cray	9	10880	32640	Custom
Spirit (ICC2)	SNL	HP	202	1028	1024	GigEthernet
Liberty	SNL	HP	445	512	944	Myrinet

Table 1. System characteristics at the time of collection. External system names are indicated in parentheses. Some information was obtained from the Top500 Supercomputer list [2]. The machines are representative of the design choices and scales seen in current supercomputers.

System	Start Date	Days	Size (GB)	Compressed	Rate (bytes/sec)	Messages	Alerts	Categories
Blue Gene/L	2005-06-03	215	1.207	0.118	64.976	4,747,963	348,460	41
Thunderbird	2005-11-09	244	27.367	5.721	1298.146	211,212,192	3,248,239	10
Red Storm	2006-03-19	104	29.990	1.215	3337.562	219,096,168	1,665,744	12
Spirit (ICC2)	2005-01-01	558	30.289	1.678	628.257	272,298,969	172,816,564	8
Liberty	2004-12-12	315	22.820	0.622	835.824	265,569,231	2,452	6

Table 2. Log characteristics. More alerts does not imply a less reliable system - it also reflects the redundancy of the reporting and the preferences of the system administrators. Alerts were tagged into categories according to the heuristics supplied by the administrators for the respective systems, as described in Section 3.2. Two alerts are in the same category if they were tagged by the same expert rule; the categories column indicates the number of categories that were actually observed in each log. Compression was done using the Unix utility `gzip`**.**

Red Storm has several logging paths [1]. Disk and RAID controller messages in the DDN subsystem pass through a 100 Megabit network to a DDN-specific RAS machine, where they are processed by `syslog-ng` and stored. Similarly, all Linux nodes (login, Lustre I/O, and management nodes) transmit syslog messages to a different `syslog-ng` collector node for storage. All other components (compute nodes, SeaStar NICs, and hierarchical management nodes) generate messages and events which are transmitted through the RAS network (using the reliable TCP protocol) to the System Management Workstation (SMW) for automated response and storage. Our study includes all of these logs.

On BG/L, logging is managed by the Machine Management Control System (MMCS), which runs on the service node, of which there are two per rack [3]. Compute chips store errors locally until they are polled, at which point the messages are collected via the JTAG-mailbox protocol. The polling frequency for our logs was set at around one millisecond. The service node MMCS process then relays the messages to a centralized DB2 database. That RAS database was the source of our data, and includes hardware and software errors at all levels, from chip SRAM parity errors to fan failures. Events in BG/L often set various RAS flags, which appear as separate lines in the log. The time granularity for BG/L logs is down to the microsecond, unlike the one-second granularity of typical syslogs. This study does not include syslogs from BG/L's Lustre I/O cluster and shared disk subsystem.

	Raw		Filtered	
Type	Count	%	Count	%
Hardware	174,586,516	98.04	1,999	18.78
Software	144,899	0.08	6,814	64.01
Indeterminate	3,350,044	1.88	1,832	17.21

Table 3. Hardware was the most common type of alert, but not the most common type of failure (as estimated by the filtered results). Filtering dramatically changes the distribution of alert types.

3.2 Identifying Alerts

For each of the systems, we worked in consultation with the respective system administrators to determine the subset of log entries that they would *tag* as being alerts. Thus, the alerts we identify in the logs are certainly alerts by our definition, but the set is (necessarily) not exhaustive. In all, we identified 178,081,459 alerts across the logs; see Table 2 for the breakdown by system and Table 4 for the alerts, themselves. Alerts were assigned *types* based on their ostensible subsystem of origin (hardware, software, or indeterminate); this is based on each administrator's best understanding of the alert, and may not necessarily be root cause. Table 3 presents the distribution of types both before and after filtering (described in Section 3.3).

Note that many of these alerts were multiply reported by one or more nodes (sometimes millions of times), requiring filtering of the kind discussed in Section 3.3. Furthermore, it means that the number of alerts we report does not neces-

Alert Type/Cat.	Raw	Filtered	Example Message Body (Anonymized)
BG/L	**348,460**	**1202**	
H / KERNDTLB	152,734	37	`data TLB error interrupt`
H / KERNSTOR	63,491	8	`data storage interrupt`
S / APPSEV	49,651	138	`ciod: Error reading message prefix after LOGIN_MESSAGE on CioStream [...]`
S / KERNMNTF	31,531	105	`Lustre mount FAILED : bglio11 : block_id : location`
S / KERNTERM	23,338	99	`rts: kernel terminated for reason 1004rts: bad message header: [...]`
S / KERNREC	6145	9	`Error receiving packet on tree network, expecting type 57 instead of [...]`
S / APPREAD	5983	11	`ciod: failed to read message prefix on control stream [...]`
S / KERNRTSP	3983	260	`rts panic! - stopping execution`
S / APPRES	2370	13	`ciod: Error reading message prefix after LOAD_MESSAGE on CioStream [...]`
I / APPUNAV	2048	3	`ciod: Error creating node map from file [...]`
I / 31 Others	7186	519	`machine check interrupt`
Thunderbird	**3,248,239**	**2088**	
I / VAPI	3,229,194	276	`kernel: [KERNEL_IB][...] (Fatal error (Local Catastrophic Error))`
S / PBS_CON	5318	16	`pbs_mom: Connection refused (111) in open_demux, open_demux: cannot [...]`
I / MPT	4583	157	`kernel: mptscsih: ioc0: attempting task abort! (sc=00000101bddee480)`
H / EXT_FS	4022	778	`kernel: EXT3-fs error (device sda5): [...]Detected aborted journal`
S / CPU	2741	367	`kernel: Losing some ticks... checking if CPU frequency changed.`
H / SCSI	2186	317	`kernel: scsi0 (0:0): rejecting I/O to offline device`
H / ECC	146	143	`Server Administrator: Instrumentation Service EventID: 1404 Memory device [...]`
S / PBS_BFD	28	28	`pbs_mom: Bad file descriptor (9) in tm_request, job [job] not running`
H / CHK_DSK	13	2	`check-disks: [node:time] , Fault Status assert [...]`
I / NMI	8	4	`kernel: Uhhuh. NMI received. Dazed and confused, but trying to continue`
Red Storm	**1,665,744**	**1430**	
H / BUS_PAR	1,550,217	5	`DMT_HINT Warning: Verify Host 2 bus parity error: 0200 Tier:5 LUN:4 [...]`
I / HBEAT	94,784	266	`ec_heartbeat_stop|src:::[node] |svc:::[node]warn|node heartbeat fault|[...]`
I / PTL_EXP	11,047	421	`kernel: LustreError: [...] @@@ timeout (sent at [time], 300s ago) [...]`
H / ADDR_ERR	6763	1	`DMT_102 Address error LUN:0 command:28 address:f000000 length:1 Anonymous [...]`
H / CMD_ABORT	1686	497	`DMT_310 Command Aborted: SCSI cmd:2A LUN 2 DMT_310 Lane:3 T:299 a: [...]`
I / PTL_ERR	631	54	`kernel: LustreError: [...] @@@ type == [...]`
I / TOAST	186	9	`ec_console_log|src:::[node]|svc:::[node]|PANIC_SP WE ARE TOASTED!`
I / EW	163	58	`kernel: Lustre:[...] Expired watchdog for pid[job] disabled after [#]s`
I / WT	107	45	`kernel: Lustre:[...] Watchdog triggered for pid[job]: it was inactive for [#]ms`
I / RBB	105	19	`kernel: LustreError: [...]All mds cray_kern_nal request buffers busy (0us idle)`
H / DSK_FAIL	54	54	`DMT_DINT Failing Disk 2A`
I / OST	1	1	`kernel: LustreError: [...]Failure to commit OST transaction (-5)?`
Spirit	**172,816,564**	**4875**	
H / EXT_CCISS	103,818,910	29	`kernel: cciss: cmd 0000010000a60000 has CHECK CONDITION, sense key = 0x3`
H / EXT_FS	68,986,084	14	`kernel: EXT3-fs error (device[device]) in ext3_reserve_inode_write: IO failure`
S / PBS_CHK	8388	4119	`pbs_mom: task_check, cannot tm_reply to [job] task 1`
S / GM_LANAI	1256	117	`kernel: GM: LANai is not running. Allowing port=0 open for debugging`
S / PBS_CON	817	25	`pbs_mom: Connection refused (111) in open_demux, open_demux: connect [IP:port]`
S / GM_MAP	596	180	`gm_mapper[[#]]: assertion failed. [path]/lx_mapper.c:2112 (m->root)`
S / PBS_BFD	346	296	`pbs_mom: Bad file descriptor (9) in tm_request, job [job] not running`
H / GM_PAR	166	95	`kernel: GM: The NIC ISR is reporting an SRAM parity error.`
Liberty	**2452**	**1050**	
S / PBS_CHK	2231	920	`pbs_mom: task_check, cannot tm_reply to [job] task 1`
S / PBS_BFD	115	94	`pbs_mom: Bad file descriptor (9) in tm_request, job [job] not running`
S / PBS_CON	47	5	`pbs_mom: Connection refused (111) in open_demux, open_demux: connect [IP:port]`
H / GM_PAR	44	19	`kernel: GM: LANAI[0]: PANIC:[path]/gm_parity.c:115:parity_int():firmware`
S / GM_LANAI	13	10	`kernel: GM: LANai is not running. Allowing port=0 open for debugging`
S / GM_MAP	2	2	`gm_mapper[736]: assertion failed. [path]/mi.c:541 (r == GM_SUCCESS)`

Table 4. Example alert messages from the supercomputers. System names are listed with the total number alerts before and after filtering. "Cat." is the alert category. Types are H (Hardware), S (Software), and I (Indeterminate). Indeterminate alerts can originate from both hardware and software, or have unknown cause. Due to space, we list only the most common of the 41 BG/L alert categories. Bracketed text indicates information that is omitted; a bracketed ellipsis indicates sundry text. Alert categories vary among machines as a function of system configurations, logging mechanisms, and what each system's administrators deem important.

Severity	Messages		Alerts	
	Count	%	Count	%
FATAL	855,501	18.02	348,398	99.98
FAILURE	1714	0.03	62	0.02
SEVERE	19,213	0.41	0	0
ERROR	112,355	2.37	0	0
WARNING	23,357	0.49	0	0
INFO	3,735,823	78.68	0	0

Table 5. The distribution of severity fields for BG/L among all messages and among our expert-tagged alerts. Tagging all FATAL/FAILURE severity messages as alerts would have yielded a 59% false positive rate.

Severity	Messages		Alerts	
	Count	%	Count	%
EMERG	3	0.00	0	0
ALERT	654	0.00	45	0.00
CRIT	1,552,910	6.09	1,550,217	98.69
ERR	2,027,598	7.95	11,784	0.75
WARNING	2,154,944	8.45	270	0.02
NOTICE	3,759,620	14.74	0	0
INFO	15,722,695	61.63	8,450	0.54
DEBUG	291,764	1.14	0	0

Table 6. The distribution of severity fields for Red Storm syslogs among all messages and among our expert-tagged alerts. These syslog alerts were dominated by disk failure messages with CRIT severity. Except for this failure case, these data suggest that syslog severity is not a reliable failure indicator.

sarily relate to the reliability of the systems in any meaningful way. The heuristics provided by the administrators were often in the form of regular expressions amenable for consumption by the `logsurfer` utility [18]. We performed the tagging through a combination of regular expression matching and manual intervention. The administrators with whom we consulted were responsible for their respective systems throughout the period of log collection and the publication of this work. Examples of alert-identifying rules using `awk` syntax include (from Spirit, Red Storm, and BG/L, respectively) include the following:

```
/kernel: EXT3-fs error/
/PANIC_SP WE ARE TOASTED!/
($5 ~ /KERNEL/ && /kernel panic/)
```

Previous work on BG/L log analysis used simple alert identification schemes such as the *severity* field of messages [9, 10, 20] or an external source of information [21, 25]. Because our objective was not to suggest an alert detection scheme, but rather to accurately characterize the content of the logs, we instead used the time-consuming manual process described above. We discovered, furthermore, that administrators for these machines do not use the severity field

as the singular way to detect alerts, and that many systems (Thunderbird, Spirit, and Liberty) did not even record this information.

Table 5 shows the distribution of severity fields among messages and among unfiltered alerts. If we had used the severity field instead of the expert rules to tag alerts on BG/L, tagging any message with a severity of FATAL or FAILURE as an alert, we would have a false negative rate of 0% but a false positive rate of 59.34%. Of the Sandia systems, only Red Storm is configured to store the severity of syslog messages (the Red Storm TCP log path is not syslog and has no severity analog). Table 6 gives the severity distribution, which suggests that syslog severity is of dubious value as a failure indicator. The use of message severity levels as a criterion for identifying failures be done only with considerable caution.

3.2.1 Alert Identification Challenges

Automatically identifying alerts in system logs is an open problem. To facilitate others in tackling this challenge, we offer the following account of issues we observed while manually tagging the logs that must be addressed by an automated scheme:

Insufficient Context. Many log messages are ambiguous without external context. The most salient piece of missing information was what we call *operational context*, which helps to account for the human and other external factors that influence the semantics of log messages. For example, consider the following ambiguous example message from BG/L (anonymized):

```
YY-MM-DD-HH:MM:SS NULL RAS BGLMASTER FAILURE
    ciodb exited normally with exit code 0
```

This message has a very high severity (FAILURE), but the message body suggests that the program exited cleanly. If the system administrator was doing maintenance on the machine at the time, this message is a harmless artifact of his actions. On the other hand, if it was generated during normal machine operation, this message indicates that all running jobs on the supercomputer were (undesirably) killed. The disparity between these two interpretations is tremendous. Only with additional information supplied by the system administrator could we conclude that this message was likely innocuous. In our experience, operational context is one of the most vital, but often absent, factors in deciphering system logs.

As seen in Figure 1, operational context may indicate whether a system is in engineering or production time. Sandia, Los Alamos, and Livermore National Laboratories are currently working together to define exactly what information is needed, and how to use it to quantify RAS performance [24]. It may be sufficient to record only a few bytes
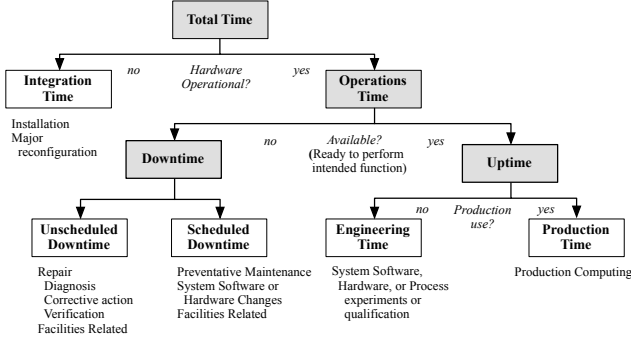
Figure 1. Operational context example. Event significance can be disambiguated if the expected state of components is known. This diagram is the current basis of Red Storm RAS metrics, and is being developed by LANL, LLNL, and SNL towards establishing standardized RAS performance metrics.
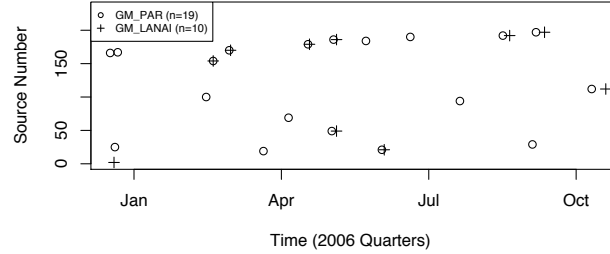

Figure 3. Two related classes of alerts from Liberty. Notice that GM_LANAI messages do not always follow GM_PAR messages, nor vice versa. However, the correlation is clear. Current tagging and filtering techniques do not adequately address this situation.

of data: the time and cause of system state changes. For example, the commencement of an OS upgrade would be accompanied by a message indicating that at time t the system entered *scheduled downtime* for a system software installation. A similar message would accompany the system's return to *production time*.

The lack of context has also affected the study of parallel workloads. Feitelson proposed removing non-production jobs from workload traces (such as workload flurries attributable to system testing [5]). Analogously, some alerts may be ignored during a scheduled downtime that would be significant during production time.

Asymmetric Reporting. Some failures leave no evidence in the logs, and the logs are fraught with messages that indicate nothing useful at all. More insidiously, even single failure types may produce varying alert signatures in the log. For example, the Red Storm DDN system generates a great variety of alert patterns that all mean "disk failure". Nodes also generate differing logs according to their function. Figure 2(b) shows the number of messages broken down by source. The chatty sources tended to be the administrative nodes or those with persistent problems, while the reticent sources were either misconfigured or improperly attributed (the result of corrupted messages).

System Evolution. Log analysis is a moving target. Over the course of a system's lifetime, anything from software upgrades to minor configuration changes can drastically alter the meaning or character of the logs. Figure 2(a) shows dramatic shifts in behavior over time. This makes machine learning difficult: learned patterns and behaviors may not be

applicable for very long. The ability to detect phase shifts in behavior would be a valuable tool for triggering relearning or for knowing which existing behavioral model to apply.

Implicit Correlation. Groups of messages are sometimes fundamentally related, but there is no explicit indication of this. See Figures 3 and 4. A common such correlation results from cascading failures.

Inconsistent Structure. Despite the noble efforts of the BSD syslog standard and others, log messages vary greatly both within and across systems. BG/L and Red Storm use custom databases and formats, and commodity syslog-based systems do not even record fields such as *severity* by default. Ultimately, understanding the entries may require parsing the unstructured message bodies, thereby reducing the problem to natural language processing on the shorthand of multiple programmers (consider Table 4). Log anonymization is also troublesome, because sensitive information like usernames is not relegated to distinct fields [6]. Our log data are not available for public study primarily because we cannot remove all sensitive information with sufficient confidence. We are working to overcome this challenge and to release the logs.

Corruption. Even on supercomputers with highly engineered RAS systems, like BG/L and Red Storm, log entries can be corrupted. We saw messages truncated, partially overwritten, and incorrectly timestamped. For example, we found many corrupted variants of the following message on Thunderbird (only the message bodies are shown):

```
kernel: VIPKL(1): [create_mr] MM_bld_hh_mr
    failed (-253:VAPI_EAGAIN)
```
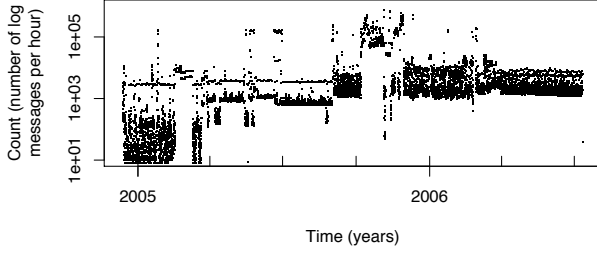
(a) The number of messages, bucketed by hour. We are told that the first major shift (end of first quarter, 2005) corresponded to an upgrade in the operating system after the machine was put into production use. The causes of the other shifts are not well understood at this time.
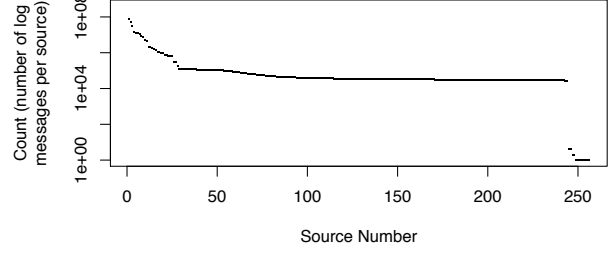
(b) The number of messages by message source, sorted by decreasing quantity. The most prolific sources were administrative nodes or those with significant problems. The cluster at the bottom is from the set of messages whose source field was corrupted, thwarting attribution.

Figure 2. The number of messages generated by Liberty.

Some corrupted versions of that line include:

```
kernel:  VIPKL(1): [create_mr] MM_bld_hh_mr
    failed (-253:VAPI_EAure = no
kernel:  VIPKL(1): [create_mr] MM_bld_hh_mr
    failed (-253:VAPI_EAGAI
kernel:  VIPKL(1): [create_mr] MM_bld_hh_mr
    failed (-253:VAPI_EAGsys/mosal_iobuf.c
    [126]: dump iobuf at 0000010188ee7880 :
```

3.3   Filtering

A single failure may generate alerts across many nodes or many alerts on a single node. Filtering is used to reduce a related set of alerts to a single initial alert per failure; that is, to make the ratio of alerts to failures nearly one. This section motivates the need for effective filtering and then describes our algorithm, which is based on previous work [9, 10] with some incremental optimizations. Briefly, the filtering removes an alert if any source had generated that category of alert within the last T seconds, for a given threshold T. Two alerts are in the same category if they were both tagged by the same expert rule.

3.3.1   Motivation for Filtering

During the first quarter of 2006, Liberty saw 2231 job-fatal alerts that were caused by a troublesome software bug in the Portable Batch System (PBS). The alerts, which read pbs_mom: task_check, cannot tm_reply, indicated that the MPI rank 0 mom died. Jobs afflicted by this bug could not complete and were eventually killed, but not before generating the task_check message up to 74 times. We estimate that this bug killed as many as 1336 jobs before it was tracked down and fixed (see Figure 4).

Between November 10, 2005 and July 10, 2006, Thunderbird experienced 3,229,194 so-called "Local Catastrophic Errors" related to VAPI (the exact nature of many of these alerts is not well-understood by our experts). A
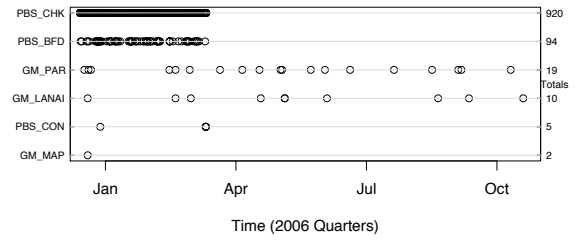

Figure 4. Categorized filtered alerts on Liberty over time. The horizontal clusters of PBS_CHK and PBS_BFD messages are not evidence of poor filtering; they are actually instances of individual failures. Specifically, they are the manifestation of the PBS bug described in Section 3.3.1. These two tags are a particularly outstanding example of correlated alerts relegated to different categories.

single node was responsible for 643,925 of them, of which filtering removes all but 246.

The Spirit logs were largest, despite the system being the second smallest. This was due almost entirely to disk-related alert messages which were repeated millions of times. For example, over a six-day period between February 28 and March 5, there was a disk problem that triggered a total of 56,793,797 alerts. These were heavily concentrated among a handful of problematic nodes. Over the complete observation period, node id sn373 logged 89,632,571 such messages, which was more than half of all Spirit alerts.

3.3.2   Filtering Algorithm

A temporal filter coalesces alerts within T seconds of each other on a given source into a single alert. For example, if a node reports a particular alert every T seconds for a week, the temporal filter keeps only the first. Similarly, a spatial filter removes an alert if some other source had pre-

viously reported that alert within T seconds. For example, if k nodes report the same alert in a round-robin fashion, each message within T seconds of the last, then only the first is kept. Previous work applied these filters serially [9, 10].

Our filtering algorithm, however, performs both temporal and spatial filtering simultaneously; an alert message generated by source s is considered redundant (and removed) if *any source*, including s, had reported that alert category within T seconds. This change reduces computational costs (16% faster on the Spirit logs), and increases conceptual simplicity. We applied this filter to the logs from the five supercomputers using $T = 5$ seconds in correspondence with previous work [4, 9, 10]. The algorithm in pseudocode is given below, where A is the sequence of N unfiltered alerts. Alert a_i happens at time t_i and has category c_i. The sequence is sorted by increasing time. The table X is used to store the last time at which a particular category of alert was reported.

Algorithm 3.1: LOGFILTER(A)

$$l \leftarrow 0$$
$$\textbf{for } i \leftarrow 1 \textbf{ to } N$$
$$\textbf{do} \begin{cases} \textbf{if } t_i - l > T \\ \quad \textbf{then } \text{clear}(X) \\ l \leftarrow t_i \\ \textbf{if } c_i \in X \textbf{ and } t_i - X[c_i] < T \\ \quad \textbf{then } X[c_i] \leftarrow t_i \\ \quad \textbf{else } \begin{cases} X[c_i] \leftarrow t_i \\ \textbf{output } (a_i) \end{cases} \end{cases}$$

This filter may remove independent alerts of the same category that, by coincidence, happen near the same time on different nodes. For example, node sn373 on Spirit experienced disk problems and output tens of millions of alerts over the course of several days. Coincidentally, another node (sn325) had an independent disk failure during this time. Our filter removed the symptomatic alert, erroneously.

In some cases, serial filtering fails to remove alerts that share a root cause, and which a human would consider to be redundant. The problem arises when the temporal filter removes messages that the spatial filter would have used as cues that the failure had already been reported by another source. Alerts removed by our filter that would be left by serial filters tend to indicate failures in shared resources that were previously noticed by another node. The most common such errors in Liberty, Spirit, and Thunderbird were related to the PBS system.

At most one true positive was removed on any single machine, whereas sometimes dozens of false positives were removed by using our filter instead of the serial algorithm. Limiting false positives to an operationally-acceptable rate tends to be the critical factor in fault and intrusion detection systems, so we consider this trade-off to be justified.
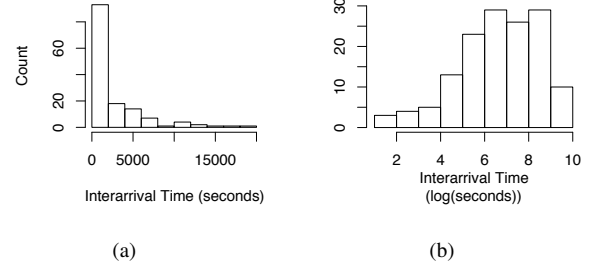

Figure 5. Critical ECC memory alerts on Thunderbird. These data are filtered, but that had little effect on the distribution. Both (a) and (b) are the same data, viewed in different ways. We conclude that these low-level failures are basically independent.

4 Analysis

Modeling the timing of failure events is a common endeavor in systems research; these models are then used to study the effects of failures on other aspects of the system, such as job scheduling or checkpointing performance. Frequently, for mathematical convenience and reference to basic physical phenomena, failures are modeled as occurring independently (exponential interarrival times). For low-level failures triggered by such physical phenomena, these models are appropriate; we found that ECC failures (memory errors that were critical, rather than single bit errors) behaved as expected. Figure 5 shows these filtered alert distributions on Thunderbird, where the distribution appears exponential and is roughly log normal with a heavy left tail.

For most other kinds of failures, however, this independence is not an appropriate assumption. Failure prediction based on time *interdependence* of events has been the subject of much research [9, 11, 13, 19], and it has been shown that such prediction can be a potent resource for improving job scheduling [17], QoS [16], and checkpointing [14, 15].

We expected CPU clocking alerts, for instance, to be similar to ECC alerts: driven by a basic physical process. We were surprised to observe clear spatial correlations, and discovered that a bug in the Linux SMP kernel sped up the system clock under heavy network load. Thus, whenever a set of nodes was running a communication-intensive job, they would collectively be more prone to encountering this bug. We investigated this message only after noticing that its occurrence was spatially correlated across nodes.

Through our attempts to model failure distributions, we are convinced that supercomputer failure types are diverse in their properties. Some clearly appear to be lognormal (Figure 5(a)), most clearly do not (Figures 6(a) and 5(b)). In even the best visual fit cases, heavy tails result in very poor statistical goodness-of-fit metrics. While the tempta-
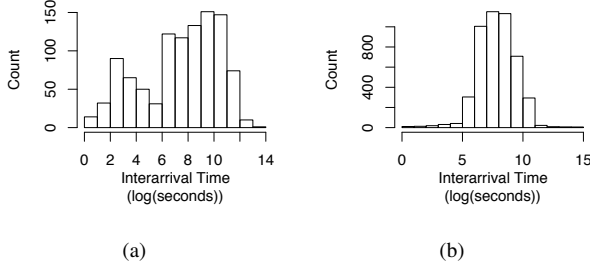
Figure 6. The log distribution of interarrival times after filtering suggests correlated alerts on BG/L (a) and largely independent categories on Spirit (b). This illustrates two weaknesses in current filtering algorithms: (1) message tags must represent independent sets of alerts to avoid timing correlations and (2) a single filtering threshold is not appropriate for all kinds of messages.

tion to select and publish best-fit models and parameters is strong, the most important observation we can make is that such modeling of this data is misguided. The mechanisms and interdependencies of failures must be better understood before statistical models of their distributions will be of significant use. The merit of a model is dependent on the context in which it is applied; one size does not fit all.

Moreover, whereas the failures in this study have widely varying signatures, previous prediction approaches focused on single features for detecting all failure types (e.g. severity levels or message bursts). Future research should consider ensembles of predictors based on multiple features, with failure categories being predicted according to their respective behavior.

Current filtering algorithms, including ours, suffer from two significant weaknesses. First, they require a mechanism for determining whether two alerts from different sources at different times are "the same" in some meaningful way. We are not aware of any method that is able to confidently state whether two messages that are labeled as different are actually correlated with one another. The second major weakness is that a filtering threshold must be selected in advance and is then applied across all kinds of alerts. In reality, each alert category may require a different threshold, which may change over time. The bimodal distribution visible in Figure 6(a) is believed to be a consequence of these shortcomings. One of the modes (the first peak) is attributed to unfiltered redundancy. Figure 3 shows an example of inter-tag correlation. On Spirit, the problems enumerated above were not as prevalent after filtering, and the result was the unimodal distribution in Figure 6(b).

5 Recommendations

In order to accurately detect, attribute, quantify, and predict failures in supercomputers, we must understand the behavior of systems, including the logs they produce. This paper presents the results of the broadest system log study to date (nearly one billion messages from five production supercomputers). We consider logs from the BG/L, Thunderbird, Red Storm, Spirit, and Liberty supercomputers (Section 3), and we identify 178,081,459 alert messages in 77 categories (Table 4). In conclusion, we describe how people want to use supercomputer logs, what obstacles they face, and our recommendations for overcoming those challenges.

Detect Faults We want to identify failures quickly. Most failures are evidenced in logs by a signature (presence or absence of certain messages), while others leave no sign. We believe such silent failures are rare. Accurate detection and disambiguation requires external information like operational context (Figure 1). We suggest logging transitions among operational states (Section 3.2.1).

Attribute Root Causes We want to respond to failures effectively, which requires knowing what failed and why. Logging mechanisms themselves may fail, resulting in corrupted or missing messages. Redundant and asymmetric alert reporting necessitates filtering (Section 3.3); we advise that future work investigate filters that are aware of correlations among messages and characteristics of different failure classes, rather than a catch-all threshold (Section 4).

Quantify RAS We want to model and improve RAS metrics. Despite the temptation to calculate values like MTTF from the system logs, doing so can be inaccurate and misleading. The content of the logs is a strong function of the specific system and logging configuration; using logs to compare machines is absurd. Even on a single system, the logs change over time, making them an unreliable measure of progress. We recommend calculating RAS metrics based on quantities of direct interest, such as the amount of useful work lost due to failures.

Predict Failures We want to predict failures in order to minimize their impact. The mapping from failures to message signatures is many-to-many. Prediction efforts must account for significant shifts in system behavior (Section 3.2.1). Just as filtering would benefit from catering to specific classes of failures, predictors should specialize in sets of failures with similar predictive behaviors (Section 4).

System logs are a rich, ubiquitous resource worth exploiting. They present many analysis challenges, however, and should not be taken lightly. Pursuing the recommendations in this paper will lead us closer to our ultimate goal: reliable computing for production users.

6 Acknowledgments

The authors would like to thank the following people for their time, expertise, and data: Sue Kelly, Bob Ballance, Ruth Klundt, Dick Dimock, Michael Davis, Jason Repik, Victor Kuhns, Matt Bohnsack, Jerry Smith, and Josh England of Sandia National Labs; Kim Cupps, Adam Bertsch, and Mike Miller of Livermore National Labs; and Ramendra Sahoo of IBM. We would also like to thank our paper shepherd, Ravishankar Iyer, for his guidance.

References

[1] Cray red storm architecture documents. http://www.cray.com/products/xt3/index.html, 2006.

[2] Top500 supercomputing sites. http://www.top500.org/, June 2006.

[3] N. R. Adiga and The BlueGene/L Team. An overview of the bluegene/l supercomputer. In *Proceedings of ACM Supercomputing*, 2002.

[4] M. F. Buckley and D. P. Siewiorek. A comparative analysis of event tupling schemes. In *FTCS-26, Intl. Symp. on Fault Tolerant Computing*, pages 294–303, June 1996.

[5] D. G. Feitelson and D. Tsafrir. Workload sanitation for performance evaluation. In *IEEE Intl. Symp. Performance Analysis Syst. & Software (ISPASS)*, pages 221–230, Mar 2006.

[6] U. Flegel. Pseudonymizing unix log files. In *Proceedings of the Infrastructure Security Conference (InfraSec)*, 2002.

[7] J. L. Hellerstein, S. Ma, and C. Perng. Discovering actionable patterns in event data. *IBM Systems Journal*, 41(3), 2002.

[8] I. Lee and R. K. Iyer. Faults, symptoms, and software fault tolerance in the tandem guardian90 operating system. In *Fault-Tolerant Computing. FTCS-23. Digest of Papers., The Twenty-Third International Symposium on*, pages 20–29, 1993.

[9] 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)*, pages 425–434, 2006.

[10] Y. Liang, Y. Zhang, A. Sivasubramaniam, R. K. Sahoo, J. Moreira, and M. Gupta. Filtering failure logs for a bluegene/l prototype. In *Proceedings of the Intl. Conf. on Dependable Systems and Networks (DSN)*, pages 476–485, June 2005.

[11] T. T. Y. Lin and D. P. Siewiorek. Error log analysis: statistical modeling and heuristic trend analysis. *Reliability, IEEE Transactions on*, 39(4):419–432, 1990.

[12] S. Ma and J. Hellerstein. Mining partially periodic event patterns with unknown periods. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 409–416, 2001.

[13] F. A. Nassar and D. M. Andrews. A methodology for analysis of failure prediction data. In *Real-Time Systems Symposium*, pages 160–166, December 1985.

[14] A. Oliner, L. Rudolph, and R. Sahoo. Cooperative checkpointing theory. In *Proceedings of the 20th Intl. Parallel and Distributed Processing Symposium (IPDPS)*, 2006.

[15] A. Oliner, L. Rudolph, and R. K. Sahoo. Cooperative checkpointing: A robust approach to large-scale systems reliability. In *Proceedings of the 20th Intl. Conf. on Supercomputing (ICS)*, Cairns, Australia, June 2006.

[16] A. J. Oliner, L. Rudolph, R. K. Sahoo, J. E. Moreira, and M. Gupta. Probabilistic qos guarantees for supercomputing systems. In *Proceedings of the Intl. Conf. on Dependable Systems and Networks (DSN)*, pages 634–643, 2005.

[17] A. J. Oliner, R. K. Sahoo, J. E. Moreira, M. Gupta, and A. Sivasubramaniam. Fault-aware job scheduling for bluegene/l systems. In *Proceedings of the 18th Intl. Parallel and Distributed Processing Symposium (IPDPS)*, pages 64+, 2004.

[18] J. Prewett. Analyzing cluster log files using logsurfer. In *Proceedings of the 4th Annual Conference on Linux Clusters*, 2003.

[19] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam. Critical event prediction for proactive management in large-scale computer clusters. In *Proceedings of the 9th ACM SIGKDD, International Conference on Knowledge Discovery and Data Mining*, pages 426–435. ACM Press, 2003.

[20] R. K. Sahoo, A. Sivasubramaniam, M. S. Squillante, and Y. Zhang. Failure data analysis of a large-scale heterogeneous server environment. In *Proceedings of the Intl. Conf. on Dependable Systems and Networks (DSN)*, pages 772–781, June 2004.

[21] B. Schroeder and G. Gibson. A large-scale study of failures in high-performance-computing systems. In *Proceedings of the Intl. Conf. on Dependable Systems and Networks (DSN)*, Philadelphia, PA, June 2006.

[22] B. Schroeder and G. Gibson. Disk failures in the real world: What does an mttf of 1,000,000 hours mean to you? In *5th Usenix Conference on File and Storage Technologies (FAST 2007)*, 2007.

[23] J. Stearley. Towards informatic analysis of syslogs. In *IEEE International Conference on Cluster Computing*, pages 309–318, 2004.

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

[25] D. Tang and R. K. Iyer. Analysis and modeling of correlated failures in multicomputer systems. *Computers, IEEE Transactions on*, 41(5):567–577, 1992.

[26] M. M. Tsao. *Trend Analysis and Fault Prediction*. PhD dissertation, Carnegie-Mellon University, May 1983.

[27] R. Vaarandi. A breadth-first algorithm for mining frequent patterns from event logs. In *Proceedings of the 2004 IFIP International Conference on Intelligence in Communication Systems*, volume 3283, pages 293–308, 2004.

[28] K. Yamanishi and Y. Maruyama. Dynamic syslog mining for network failure monitoring. In *Proceedings of the 11th ACM SIGKDD, International Conference on Knowledge Discovery and Data Mining*, pages 499–508, New York, NY, USA, 2005. ACM Press.