Doomsday: Predicting Which Node Will Fail When on Supercomputers

Anwesha Das, Frank Mueller
North Carolina State University
{adas4,fmuelle}@ncsu.edu

Paul Hargrove, Eric Roman, Scott Baden
Lawrence Berkeley National Laboratory
{phhargrove,ERoman,baden}@lbl.gov

***Abstract*—Predicting which node will fail and how soon remains a challenge for HPC resilience, yet may pave the way to exploiting proactive remedies before jobs fail. Not only for increasing scalability up to exascale systems but even for contemporary supercomputer architectures does it require substantial efforts to distill anomalous events from noisy raw logs. To this end, we propose a novel phrase extraction mechanism called TBP (time-based phrases) to pin-point node failures, which is unprecedented. Our study, based on real system data and statistical machine learning, demonstrates the feasibility to predict which specific node will fail in Cray systems. TBP achieves no less than 83% recall rates with lead times as high as 2 minutes. This opens up the door for enhancing prediction lead times for supercomputing systems in general, thereby facilitating efficient usage of both computing capacity and power in large scale production systems.**

***Index Terms*—Machine Learning, HPC, Failure Analysis**

I. Introduction

Significant efforts have been made to improve the resilience of HPC systems in recent times. Existing health check monitors and techniques such as root cause diagnosis and failure detection use diverse log sources to combat failures. However, they still fall short of strong means to handle node failures *proactively* in complex, large scale computing systems. First, supercomputing systems are constantly changing due to novel architectures, design, upgraded applications and logging mechanisms. Prior techniques of automated fault diagnosis do not suffice for the evolving changes [1].

Second, existing HPC infrastructures with their increasing component count required for exascale ($\approx 10^6$ nodes) make accurate fault prediction hard. Aborted jobs due to node failures inflict significant energy costs [2]. More than 20% of the compute capacity is wasted in failures and recovery, as reported by DARPA [3]. With increasing number of nodes, the mean time between failures (MTBF) reduces for a node, making fault identification and resolution even more difficult. Increasing complexity in emerging next generation systems obviates the need for adaptive fault aware solutions to address this critical reliability challenge.

To address this challenge, we present a fault-tolerant solution to pin-point potential node failures in HPC systems. Our study on Cray system data with an automated machine learning technique suggests that careful time series analysis of log phrases can be used to predict node failures. Recovery techniques such as checkpoint/restart (CR) and redundancy/replication incur additional costs [4]. Our methodology to identify *which nodes* are likely to fail (location information) prior to actual fail-stop behavior can reduce the overhead of failure recovery.

As of November 2017, 29% of the top 100 and 40% of the top 10 supercomputers (e.g., Titan, Cori, Trinity) [5] were Cray machines. It is important to investigate Cray machines more closely to explore techniques that increase reliability. This paper discusses the inherent challenges of Cray systems and proposes a mechanism to predict node failures.

***Motivation**:* From log data analysis to root cause diagnosis across various levels (hardware, system, application) researchers have studied failure manifestations in HPC systems and devised ways to improve recall rates [6]. In spite of such a large body of work on resilience, further investigation is required for the following reasons:

- Existing work performs prediction and diagnosis without sufficient emphasis on lead time requirements. Pinpointing which nodes will fail well ahead in time to proactively counter performance disruptions still remains a challenge. Optimal learning window interval selection and determining appropriate lead times are important considerations for successful prediction of node failures.
- Most prior studies [7]–[9] use the same training data for future predictions over a long time frame. Gainaru et al. [6] found that correlations determined off-line when are dynamically adapted result in limitations for short training sets applied to a long future time window. This makes prediction impractical on production systems. Dynamic learning and scalable online prediction techniques need to be investigated to improve prediction efficiency.
- Prior resilience work [7]–[12] studied rich BlueGene logs of decommissioned systems. Time sensitive failure prediction in contemporary systems (e.g., Cray) with lower-level Linux-style raw logs need further exploration to better understand the requirements of resilience.

We focus on the 1st and 3rd aspects in this paper.

***Contributions**:* This paper shows a novel way to extract failure messages indicative of compute node failures for Cray systems. First, we provide an analysis of Cray system logs and job logs and show how failure prediction in such systems poses additional challenges compared to systems such as BlueGene.

Second, we discuss what node failure exactly means in the context of Cray systems, what traits govern normal shutdowns

SC18, November 11-16, 2018, Dallas, Texas, USA
978-1-5386-8384-2/18/$31.00 ©2018 IEEE

and abnormal reboots and how we can avoid trivial pitfalls in node failure detection. We provide frequency estimates of compute and service node failures highlighting their potential consequences on systems and user applications.

Finally, we propose a novel prediction scheme, TBP (time-based phrase), to extract relevant log messages indicative of node failures from noisy data. This scheme relies on phrase likelihood estimation considering continuous time-series data to elicit out useful messages. These events help forecast future failures with lead times ranging from 20 secs to 2 minutes.

II. Background

Let us provide a brief overview of the system logs studied highlighting the main components of such logs used for analysis. Table I summarizes the system and job logs collected from three contemporary systems, namely: SC1, SC2 and SC3. Size refers to the log data size and scale indicates the cluster size in terms of the number of compute and service nodes. These systems have been widely deployed and typically run more than 1,400,000 jobs/year.

TABLE I: System Details

System	Duration	Size	Scale	Type
SC1	14 months	573GB	5600 nodes	Cray XC30
SC2	18 months	450GB	6400 nodes	Cray XE6
SC3	8 months	39GB	2100 nodes	Cray XC40

Cray System Architecture: Figure 1 shows a high-level overview of a Cray system. A job scheduler distributes user jobs on the allocated compute nodes. Production jobs are executed on the compute nodes and external clients access the cluster through the login nodes. The parallel file system (e.g. *Lustre*) and a network server (e.g., *Aries* implementation of the generic network interface (GNI)) communicate with the service nodes and compute nodes. The **S**ystem **M**anagement **W**orkstation (SMW) administers and logs various cluster components and monitors resource usage. The **S**ervice **D**ata**b**ase Node (SDB) stores information of all the service nodes. The boot node manages the shared file system with service nodes. Login, boot and SDB are some of the service nodes of the system in addition to syslog, I/O and networking nodes. The **A**pplication **L**evel **P**lacement **S**cheduler (ALPS) processes such as `aprun`, `apbridge`, `apsched`, `apinit` etc. are responsible for user application submission and monitoring and run on both service and compute nodes.

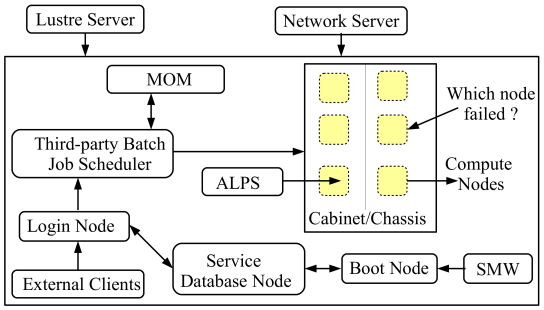

Fig. 1: Overview of a Cray System

Table II indicates the major log sources. From these archived logs, we consulted p0-directories that contain comprehensive logs of the entire system with information pertaining to the internals of compute nodes including system and environment data. We use the acronym *id* to refer to node or job *identifiers*. The files (console/message) in these directories contain timestamped event logs including node ids (cX-cYcCsSnN) per line. We track the node ids per log line (event) while correlating and integrating data. Logs can contain sequential decimal numbering of nodes (nids), which can be converted to the physical id, i.e., cX-cYcCsSnN format to identify a node's physical location such as blade (S), chassis (C), cabinet (XY) and the node (N). Additional references to boot messages and job logs aid the prediction scheme since they provide the status of nodes and jobs over time.

TABLE II: Data Details

Source	Content
p0 directory	Internals of compute nodes
Boot Manager	Boot node messages
Log System	`rsyslog` messages
Power/State Logs	Component power and state information
Event Messages	Event router records
SMW Messages	System Management Workstation messages
HSN Stats	High Speed Network Interconnect logs
Job Logs	Batch job/application scheduler messages

We found that noisy information pertaining to power management, SMW, the log system, network interconnect logs, events, and the state manager did not reveal significant textual indicators related to node failures and, hence, have not been considered. Manuals/system administrators help to understand logs, but even then post-mortem analysis for unsupervised information extraction remains non-trivial from such data.

Technical Challenges: The challenges of data diversity, system complexity, and overwhelming logging volume have been investigated in the context of failure detection [8], [13]. Furthermore, Cray systems specifically have the following additional challenges:

- BlueGene systems have Node Card, Service Card, Link Card and Clock Card components, each of which provide current, voltage, temperature data etc. In Crays, failure needs to be discovered by integrating a distributed set of events in space and time, coming from different system components, some of which are replicated. Feature identification even before dimensionality reduction is hard.
- Binary or numeric values (normalized or mapped) of features (see [7], [14]) do not suffice. Simple absence or presence of an event is not enough. Which event, when did it happen, is it related to the node under consideration and similar factors are also of importance here.
- RAS logs in some HPC systems contain fatal and warning flags indicating the severity levels with the log messages [15]–[17] aiding researchers to segregate between failures and benign events. Critical and warning flags are present in Crays as well pertaining to certain components such as `netwatch`, `pcimon` etc. However, direct classification of log messages based on occasionally appearing flags is ineffective [16] for long-term time sensitive data since several non-critical messages could be a better indicator of failures over time. Besides, seemingly

benign events in one context may lead to fatal events in another, which means past system (BlueGene) logs may result in shorter lead times. Hence, we consider phrases irrespective of flags/severity-levels.
- The unsteadiness in timestamps between service nodes, job schedulers (Slurm/Torque) and compute nodes makes time-based correlation non-trivial. Time-series analysis handles this and allows us to study lead time sensitivity.

***Node Failure**:* The boot log reveals several clustered node failures caused by problems ranging from communication failures, network-interconnect and application-based errors, resource-contention, file system or hardware errors. Further study revealed certain patterns in the context of node failures. If nodes shut down in bulk (multiple blades) within a few seconds, the root cause tends to be maintenance. Such shutdowns often are massive (e.g., 98 or 126 nodes going down at once). But even in case of single compute node failures, the culprit could be either external or internal events (see Table III). *Internal events* are compute node specific, either caused by applications running on that node or hardware/software problems related to memory, kernel etc. pertaining to that node. *External events* occur outside a specific compute node such as Lustre server-related errors, a Link Control Block (LCB) failure, or a network interconnect failure causing multiple chassis to shutdown. Cabinet- or blade-controller problems can also manifest as massive node failures.

TABLE III: Node Shutdown Events

Internal Failures	External Failures	Normal Shutdowns
Application Bugs	Blade or Cabinet Controller Issues	Massive shutdown
Node System Bugs	File system or Network Server Issues	Maintenance Reboots
Node Hardware Issues	Router or other Hardware Issues	Periodic Node Reboots

***Node Failure Definition*:** Not every node unavailability indicates an anomaly. Power outages, maintenance and deliberate shutdowns have been eliminated in our study. Cases related to the internal and external events (discussed above) are considered as node failures since they manifest as anomalies.

Periodic service reboots are common in Cray nodes. Nodes are rebooted several times before they come up as part of regular maintenance. Such spurious cases are not counted as node failures since these are not caused by any faults in the system. Counts of node failures do not consider unique nodes since a specific node can fail multiple times at different timestamps. Unresponsive nodes, stress testing, and changing power cooling conditions manifest in log messages and have been considered in our study; a failed heartbeat indicates failure with unknown root cause (network/OS/hardware failures resulting in lost connection) and are indistinguishable from anomalous node failures in terms of manifestation. We have timestamped logs, and based on the time and scale of shutdowns, we segregate failures. In many instances, service node failures impact compute node failures (see Sec. V-A).

It should be noted that this work aims to *correctly predict node failures*. The goal is *not to identify the exact root cause* that provoked fault manifestation. In other words, the methodology is to identify chains of time-based events, which eventually led to a failed node. The actual cause and the location of a root cause may not be indicated by our prediction methodology. E.g., in Table IV column 1 (bit flips), after repeated soft errors a Link Control Block (LCB) went down. The cause of `CRC error messages` may range from hardware (link/NIC) problems to silent data corruption, but this cause is of no concern for our analysis.

***Illustrative Examples*:** Table IV shows three cases of node failures. The first column shows a case of a failure caused by soft errors (bit flips) detected by the LCB. Due to too many errors the LCB went down after which 2 nodes of a blade went down within 12 minutes. The second column shows the case of a hardware error caused by the network interconnect. This triggered Lnet and Lustre errors followed by memory problems, and the node went down within 11 minutes. The third column shows an application (Matlab) failure caused by excessive memory allocation. This killed several tasks, followed by a kernel panic, failing this specific node. In these cases, if prediction happens a few minutes before the actual shutdown, some proactive measures can be taken.

TABLE IV: Examples of Node Failures

bit flips caused failure	hardware caused failure	app. caused failure
`4.25.30 pm` LCB on and Ready	`8.44.12 pm` Hardware Overflow Error	`2:44:49 am` Matlab invoked oomkiller
`4.30.33 pm` Micropacket CRC Error Messages	`8.46.09 pm` Lnet errors Recvd down event	`2:54:14 am` Out of memory: Kill process
`4.35.29 pm` Network chip failed due to too many soft errors	`8.47.45 pm` Lustre Errors Binary changed	`2:58:14 am` Killed process
`4.36.42 pm` Aries LCB operating badly, will be shutdown	`8.48.06 pm` Bad RX packet error	`2:59:40 am` Kernel panic not syncing:
`4.37.31 pm` Failed LCB components	`8.52.37 pm` Out of memory/Killed processes	`3:00:00 am` page_fault+0x1f/0x30
`4.37.39 pm` 2 nodes unavailable	`8.55.13 pm` Node unavailable	`3:00:03 am` Node unavailable
Failed within `12 min.`	Failed within `11 min.`	Failed within `16 min.`

III. Predictor Design

Our study shows that Topics over Time (TOT) [18] (an LDA-style [19] unsupervised topic model) based learning can help identify rare compute node failures.

***Job Logs and Data Integration**:* Figure 2 demonstrates how jobs scheduled on allocated nodes are identified in the ALPS logs inside the p0-directories. The job server (e.g., Torque) provides information about job id, the allocated nodes for that job and the status. These can be referenced in the ALPS logs through a mapping conversion. The job ids map to batch ids. The job information is referenced using `res`, `app` and `pagg` like tags added by ALPS. The timestamp is considered for correlation by checking the amount of time lag in logs from different sources (job, server/compute node). If it is within a given threshold ($\approx$15 seconds), it is considered. This did not cause any correlation errors since the ids were matched correctly over time. In our experience, missing data in log archives and MOM (machine-oriented miniserver) node

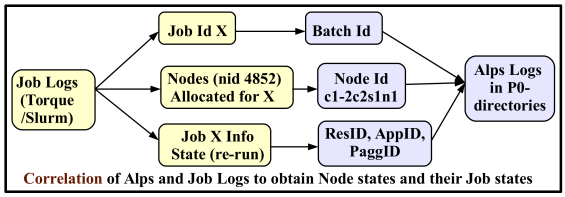

Fig. 2: Correlation with Job Logs

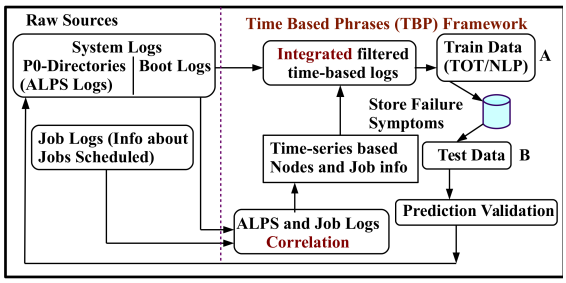

Fig. 3: TBP Framework

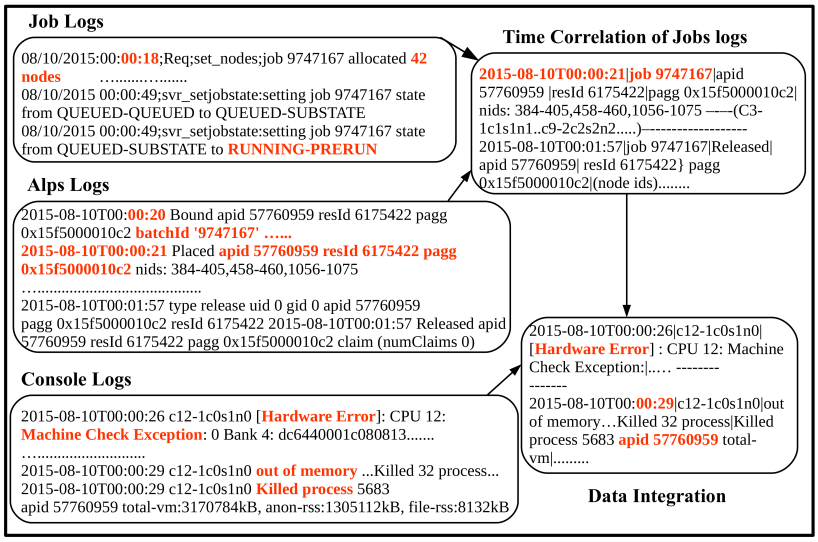

Fig. 4: Time Correlation and Data Integration

failures (see Figure 1) affecting job data complicates this correlation across the available logs. This arises due to stopped daemons/logging, with or without upgrades. Upgrades in job schedulers (ALPS-Torque, ALPS-Slurm) can create inconsistencies in the available data. But this does not hamper the prediction mechanism since the correlation has been confirmed by prior offline manual validation with system administrators, and only complete and consistent data is used for evaluation.

After successful correlation, a text document with timestamps, node ids and filtered log messages is formed to generate a viable input for our model (Figure 4). We do not use any environmental data, such as **S**ystem **E**nvironment **D**ata **C**ollection (SEDC) logs, since such information predominantly does not aid in phrase extraction. Temperature and voltage values may indicate an anomaly but our work intends to discover salient phrases providing symptoms of abnormalities.

TABLE V: Topic Assignment

#	Event Phrase	Topic
1	Lnet: waiting for hardware..	Lnet
2	Lnet: Quiesce start..	Lnet
3	Debug NMI detected	NMI
4	DVS: uwrite2: returning error	DVSBug
5	Kernel panic/not syncing/Fatal Machine check	Panic
6	MCE threshold of fff..	MCE

Phrase Likelihood Estimation: One primary tendency observed in the available logs is the recurrence of failure messages and changing patterns that continuously evolve over time. A phrase is defined as an event log message corresponding to a specific node at a certain timestamp. This trend of time-based evolution prompted us to leverage a well known machine learning technique called Latent Dirichlit Allocation (LDA) [19], a probabilistic model on discrete data. One important factor for log analysis targeting prediction is time. Hence, continuous time series-based evolution is required to extract patterns from the integrated document. To address this, we utilize the **T**opics **o**ver **T**ime (TOT) [18] algorithm to identify the top `N` topics (i.e., phrases or log messages) over a period of time and track how the topics change over time. TOT employs Gibbs Sampling and is useful for dynamic co-occurrence of patterns when an upsurge and downfall of phrases exists over time. Since TOT models time in conjunction with frequency of phrases, which is analogous to the case in temporal logs, TOT is a good choice for our study in contrast to other existing competitive approaches such as [16], [17], [20]–[22].

On similar grounds, discrete-time **D**ynamic **T**opic **M**odeling (dDTM) [23] is inappropriate since there is a significant variation in occurrence of events (at the granularity of milliseconds) so that several chunks of logs cannot be clustered under a single time instance. Coarse-level time discretization fails to capture short-term time variations. The ability to identify a known delta time difference between two known messages is critical here. The main idea is that every phrase is assigned a topic. Multiple phrases can be assigned the same topic (see Table V). We have finite number of topics for an integrated document. During the training phase, TOT learns top `N` topics referring to phrases. TBP forms sequences of phrases that correspond to failures in the past referring to the data. We use them to forecast future failures when those phrases reappear in the test data. Once we know the significant phrases, we denormalize their timestamps and refer to the nodes associated with them to identify nodes subject to future failures. We name our prediction mechanism "*time-based phrases*" (TBP).

IV. TBP Framework

Figure 3 shows the work-flow diagram of our approach. After job and system data correlation and formation of an integrated document, we obtain the top ranked phrases over time and determine the nodes corresponding to them (Box A). Then, TBP obtains a chain of messages leading to the node failures. We use them on test data to compute recall rates and lead times (Box B). Let us clarify that we do not want to determine the distribution of node failures over time/space or failure characteristics on the susceptibility of a specific node to future failures [24]–[26].

Time Correlation: Figure 4 illustrates the idea of time correlation. The job log indicates that at `00:18` a job with id

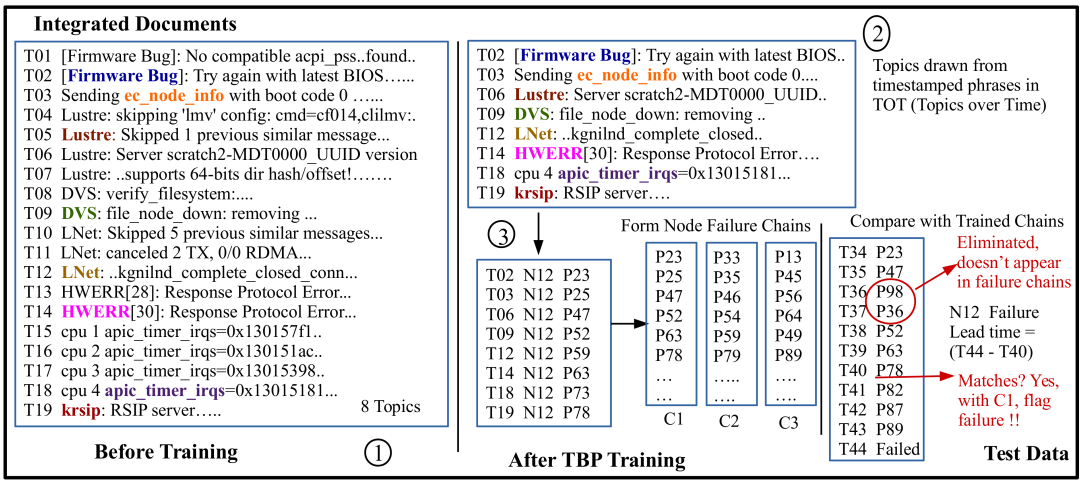

Fig. 5: TBP Prediction: Topic Modeling for Node Failure Prediction

9747167 is allocated 42 nodes. This job is correlated with its corresponding ALPS message apid 57760959, resId 6175422, pagg 0x15f5000010c2 logged at 00:20, just 2 seconds later. Since 2 seconds are within the threshold (15 seconds), these become correlated. Simultaneously, we obtain the ids of nodes allocated to this job by converting from nid (decimal node id) to the cX-cYcCsSnN format. These node ids and job ids can be time correlated to the rest of the logs (e.g., console logs) in a similar fashion as shown in Figure 4.

TBP Learning: TBP uses TOT to learn the failure chains from the training data. Topic assignment assigns a relevant topic to every phrase pertaining to that topic as shown in Table V. We have used more than 100 topics in our training data. Multiple phrases can be assigned the same topic if the content of that phrase is not anomalous or if they have similar system event context (e.g., 1 & 2). A distinct phrase can also be assigned a topic if it indicates a unique event (e.g., 3 & 5). TOT chooses top N topics (i.e., phrases) as follows (also see Figure 5): Phrases chosen are localized in time. As the distribution of phrases changes over a continuous time frame, top phrases evolve since the phrase co-occurrence changes [18]. The 8 topics (firmware bug, ec_node_info, Lustre, DVS, LNet, hwerr, apic_timer_irqs and krsip) shown in Figure 5 (box 1) pertain to phrases containing that topic. TBP uses the top picked phrases over time to formulate the failure chains. Topic-based training helps to extract only the significant phrases relevant to failures (boxes 2+3).

We varied the value of N in different data sets to ensure that we are not missing relevant phrases. In our experiments, N ranges from 50 to 80 based on the amount of data considered. We manually inspected the output of TOT while choosing a subset of N considering time and space constraints. To clarify, N has been varied but it is impractical and inefficient to inspect too large of an N value. TBP has chosen a smaller subset (smaller N) at times to effectively collect indicative phrases.

Node Failure Prediction: Figure 5 illustrates the key idea of prediction. T denotes timestamps, N stands for node ids, and P for phrase ids (for brevity we omitted job ids in the figure). The integrated document (Figure 4) is trained using TOT. We know the terminal node shutdown messages from sysadmins (e.g., System halted, cb_node_unavailable, see Table X, last 10 phrases). TBP forms failure chains linking phrases among the top N with time-stamps and node-ids referring to the data. From the filtered phrases (box 3), TBP obtains the node failure chains as shown by C1, C2 and C3. During testing, no top phrases are generated. TBP compares the incoming phrases with those in the failure chains. If chains with at least 50% similarity in log messages are formed, the corresponding node is likely to fail in the future. E.g., p98 and p36 are rejected since these were not seen in the training data, and the remaining phrases match with C1 till P78. In the test data, before N12 fails, we compare and predict it to be a potential failure. From N12 (i.e., cX-cYcCsSnN) we derive the node's exact location (blade S, chassis C, cabinet XY) to potentially trigger proactive resilience actions before it fails at a future timestamp (T44). About 1 month's data is used for training. The test data is comprised of a moving time window of 3 days to 1 week for generating better lead times.

V. Experimental Evaluation

We have implemented a prototype of the TBP predictor using the factorie [27] library and python. We use TBP on 3 datasets from SC1, SC2 and SC3 supercomputers (Table I) for evaluation. TBP achieves as high as 86% recall rates with acceptable lead times in predictive fault localization of nodes.

A. Average Node Failure Estimation

Figure 6 shows an estimate of service and compute node failures over 4 months of the data in Table I for SC1, SC2 and SC3. SC2 had the highest overall number of failures but SC3 had a slightly higher frequency of failures (failures/month). The compute node failure count is higher than the number of service node failures. Our observation indicates that both service and compute node failures are randomly distributed over time. Hence, this could be misleading in terms of estimating failure rates over longer periods of time. E.g., over a period

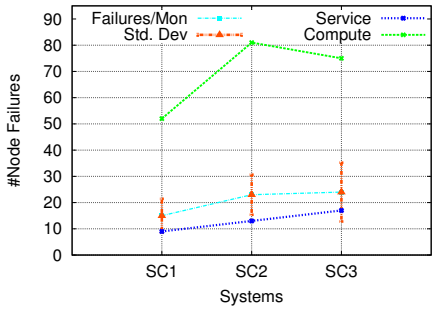

Fig. 6: Estimate of Node Failures

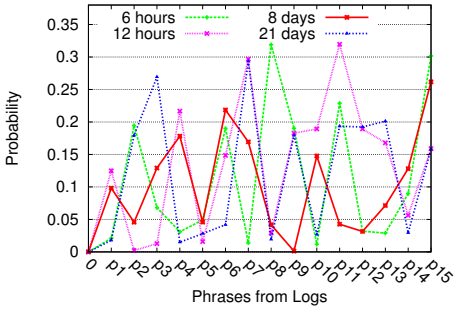

Fig. 7: Phrase Likelihood

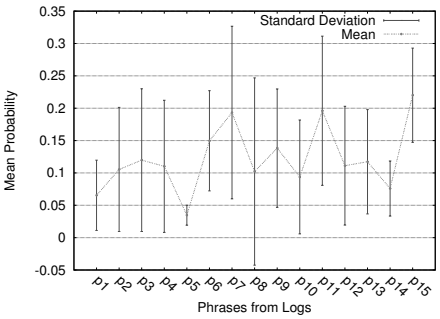

Fig. 8: Mean and Std. Deviation

of 3 weeks, SC3 encountered 10 service node failures, but the subsequent time periods of 9 days and 2 weeks had 0 and 1 service node failures, respectively. There exists sufficient variability of failures over time across the three systems. This explains the standard deviation of the frequency ranging from ± 6 to ± 11. Nonetheless, this gives us an idea of the number of failures encountered in such systems.

One might argue, if node failure events are relatively rare compared to the overall scale of anomalies in the system, why do we need to predict them and take proactive actions? In this regard, we have summarized two main takeaways:

- *The number of compute node failures increases dramatically with an increase in service node failures.* On multiple occasions, time periods with high service node failures have affected a large number of compute nodes around the same time due to external failures. We did not investigate further the exact root cause of each failed compute node, but in addition to maintenance related shutdowns, controlled service node failures can definitely prevent compute node failures, which benefits jobs of users running on them.
- *Rescheduling a job after a node failure delays the overall job execution time and utilizes additional resources.* A single job, on average, is generally allocated to many compute nodes (up to tens of thousands for peta-/exascale capability jobs). If blades fail repeatedly, the effect is logged as independent job failures, which are rescheduled. Past work has observed that a significant fraction of applications fail due to system problems apart from user-related errors [2]. If user application disruptions are a concern and system-wide outages are to be reduced, node failure prediction is of paramount importance.

B. Phrase Distribution

Table VI shows a sample snippet of 10 phrases over a time window of 2 weeks with their probability distributions depicting a compute node failure case caused by Lustre server-based errors. The table shows that Lustre, Lnet and filesystem-related messages are produced by TBP with higher probability than other system messages. These phrases are related to filesystem problems impacting the node. We do not care about individual probabilities as long as the top `n` phrases are of interest in the context of node failures.

Observation 1: *Significant phrase variation exists over short time intervals. HPC logs indicate event changes at a high frequency, which calls for continuous-time statistical models that can handle this variability prior to identifying phrase relevance for node resilience.*

Figure 7 and Table VII illustrate the variation in probability of occurrence for the same 15 phrases over four continuous time intervals (6 hours, 12 hours, 8 days and 21 days) for SC3 data. The 4 disjoint time frames have been selected from over 3 months of data and illustrate the lack of a uniform distribution. Table VII depicts frequently occurring system log messages pertaining to Lustre, Lnet etc. Figure 7 shows the fluctuations of phrase probabilities over different time intervals. We provide a quantitative analysis of such variations to signify the lack of discernible features. Pattern extraction with such non-uniform distribution of unstructured log messages but without clearly flagged errors is hard.

Figure 8 quantifies this variance through mean (curve) and standard deviation (indicated by error bars). The standard deviation for most phrases is high (e.g., for `p7`, `p8` and `p11`), except for `p5`. Example: Message `p7` is emitted by Lustre for an unmounted device with a higher probability distribution in the 2nd (12 hours) and 4th (21 days) time intervals, but the device was mounted and the message occurred less in the 1st (6 hours) and 3rd (8 days) time intervals. Similarly, `p11` indicates a Data Virtualization Service (DVS) server failure, which is unmounted. The failover event occurred with a different magnitude over the intervals for multiple nodes. Message `p5` related to the LDAP server had less deviation since the LDAP connection was successful after a few attempts in those time intervals. Discrete-time statistical methods [17] are ineffective under such variability.

C. Prediction and Lead Time Analysis

Phrases detected during the learning phase of around 4 weeks help in node failure prediction on new test data. TBP checks the test data for phrase similarity relative to the training data (Figure 5). If similar, we obtain the node ids corresponding to those phrases and compute recall rates. N-fold cross validation is less effective for time-series data. Our train and test data split respects temporal event ordering, (lower-order time-series training, higher-order testing).

TBP uses the standard evaluation metrics of Recall and Precision to estimate prediction efficacy. Table VIII enumerates

TABLE VI: Phrase Extraction

#	Phrases	Prob.
1	Ensure file system is mounted on the server and then restart DVS	0.0214
2	LNET: waiting for hardware quiesce flag to clear	0.0145
3	nscd: nss_ldap: failed to bind to LDAP server	0.0167
4	LustreError: *:*:.....unable to mount	0.0298
5	startproc: nss_ldap: failed to bind/reconnecting to LDAP server	0.0302
6	Error: No data from cname	0.0178
7	Lustre: skipped * previous similar messages	0.0119
8	Lustre:*:*:vvp_io.c:*: vvp_io_fault_start	0.0176
9	reconnected to LDAP server	0.0247
10	Lnet: * Dropping PUT from *	0.0218

TABLE VII: Recurring Phrases

No.	Phrases
P1	crms_wait_for_linux_boot: nodelist: *
P2	Lnet: Quiesce start: hardware quiesce
P3	Wait4Boot: JUMP:KernelStart *
P4	krsip:RSIP server * not responding
P5	startproc: nss_ldap: failed to bind
P6	checking on pid *
P7	LustreError: *:*:.....can't find the device name
P8	GNII_SMSG_SEND + *
P9	No .....bios_settings file found
P10	Lnet: Added LNI *
P11	DVS: file_node_down: removing *
P12	Lustre: skipped * previous similar messages
P13	Lnet: skipped
P14	<node_health:*> RESID* xtnhc FAILURES
P15	Bad RX packet error

TABLE VIII: Evaluation Metrics

Metric	Formula & Implication
Recall	TP/(TP+FN) # Node failures, TBP *correctly* predicted
Precision	TP/(TP+FP) # Total node failures, TBP predicted
FP Rate	FP/(FP+TN) # False Positive Rate
True Positive (TP)	# Actual node failures, TBP successfully predicted
True Negative (TN)	# Nodes actually didn't fail, TBP did not predict as failure
False Positive (FP)	# Nodes actually didn't fail, TBP predicted as failure
False Negative (FN)	# Actual node failures, TBP failed to predict

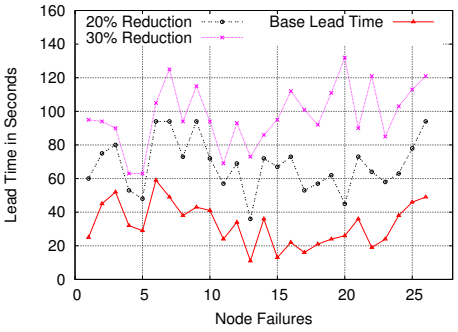

Fig. 9: Sensitivity of Lead Times

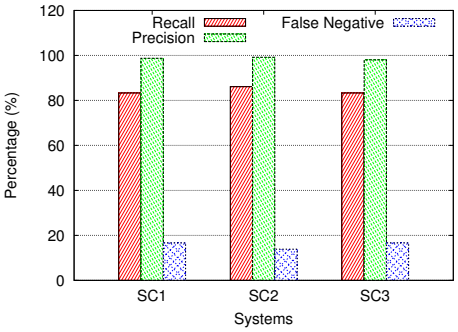

Fig. 10: Recall/Precision/FNR Rates

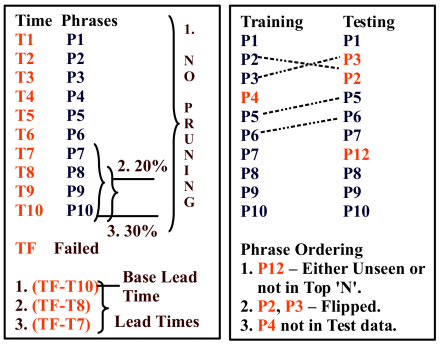

Fig. 11: Phrase Reduction and Order

their formulae and the implications in the context of node failure prediction. The recall rate is defined as the fraction of node failures that are correctly predicted by TBP, and precision rate as the total fraction of node failures predicted by TBP (need not be correct). The FP rate is the false alarm rate, the ratio of actual failures missed by TBP.

Validation is performed by manually checking the logs with the timestamps of actual failures. In the test phase, we consider 3 days' to 1 week's data, and compare the phrases with the obtained trained data. We re-train (4 weeks data) and move the test data time interval with shifts of 1 week to predict impeding failures and procure lead times. The base lead times (without phrase reduction) in Figure 9 are in the range of 20 to 60 seconds. We have optimized TBP's lead time sensitivity further through phrase pruning (see Section V-D).

Observation 2: *TBP achieves > 83% recall and > 98% precision with a modest number of false negatives (16.66%) and as high as 1 minute base lead time.*

Figure 10 shows the recall, precision and false negative rates. We observe a precision rate of up to 99%, which indicates a low rate of false positives. This indicates that the trained failure chains were indeed indicative of node failures. The recall rates are 86% or lower. TBP aims not to miss actual node failures irrespective of the causes and correlations between them. Even if correlated failures are removed, precision and recall exceed 80%. Across all the 3 systems the false negative rate is as high as 16.66%. This is partly because of the new phrases seen while testing and partly due to the corner cases where phrase extraction is difficult. SC2 has a relatively low false negative rate since failure events learned during training were mostly seen during testing with similar failure types (e.g., networking problems). Figure 10 rates correspond to the base lead time (no phrase reduction).

Observation 3: *Hardware errors, MCEs and kernel panics often cause node failures. Minor causes are failed components in the network interconnect, bit errors, filesystem caused errors and application based errors.*

Figure 12 shows the proportion of different types of failures observed in the data, namely Kernel panic, MCE (Machine Check Exceptions), FS (filesystem errors), SWERR (Software errors), Soft (bit/packet/protocol related soft errors), App (application errors), and HWERR (hardware node heartbeat faults), respectively. Table IX lists the major classes of anomalies manifesting in node disruption. While 15 to 20% of the nodes fail due to H/W errors, MCEs and kernel panics, bit errors and App caused errors are minor contributors. Our observations conform to failures reported by Gupta et al. [28].

We predict node failures on average a minute in advance; this is an improvement over scenarios where nodes fail within 20-30 seconds of the occurrence of the first reported event that can be linked to a later failure by system administrators. However, certain failures such as NMI faults cause instant failures without a chance to communicate anymore. It is impossible to take proactive actions in those cases.

D. Lead Time Improvements

Starting from the last phrase considered from prior learning, we prune backwards to increase lead times and assess the impact on false positives. Figure 11 (left) depicts backward

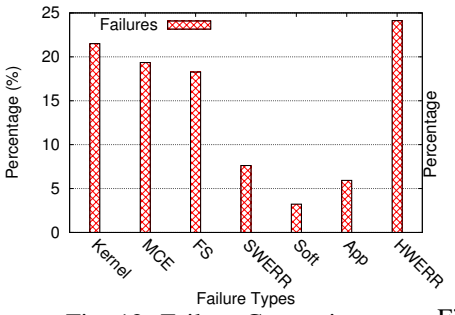

Fig. 12: Failure Categories

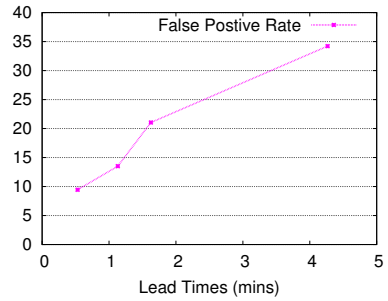

Fig. 13: Lead Times+False Positives

TABLE IX: Major Failure Categories

#	Failure Type	Class
1	Trap invalid opcode/Segfaults, File System Bugs	Software
2	cb_hw_error: failed_component, Firmware Bugs, NMI Faults	Hardware
3	[hwerr]: Machine Check Exception (MCEs), Memory faults	Hardware
4	RCU CPU Stalls/Hangs, Kernel Panic/Fatal Exception, Stack Trace	Software
5	Bit/Packet Errors (dla_overflow error. Msg protocol error)	Soft Errors
6	Machine Check Events (Node heartbeat faults)	Hardware
7	Job Server/Task related errors	Application

pruning as an example. Suppose 10 phrases are considered named P1, P2...P10 with increasing timestamps T1, T2,..T10. A node failure occurred at TF. When 20% phrases are pruned, the last 2 phrases with ordered timestamps are removed from consideration, i.e., those phrases are not checked in the test data to qualify as a failure chain. The percentage of reduction is increased to gain longer lead times. The earlier correct failures are flagged, the higher the lead time will be. Lead times improve from (TF-T10) to (TF-T7) with 30% phrase reduction as shown in Figure 11 (left). In reality, the number of phrases considered are higher than 10 (30-50). Figure 11 (right) depicts cases of phrase mismatches. Observed phrases in the test data may not be in the same order as the trained phrase set under consideration. E.g., `P2` & `P3` or unseen phrases (e.g., `P12`) may be present or a phrase seen in the trained chain earlier is missing (e.g., `P4`) in the test data interval. In such cases, TBP ensures a similarity of 50% or higher and otherwise discards phrases as unmatched.

Figure 9 illustrates the increased lead times with 20% and 30% phrase reduction for each of the 26 node failures across different machines. (The rest of the failures have similar lead times.) With phrase pruning lead times increase. Corresponding to a specific average lead time, we calculate the false positive rate from the data set.

Observation 4: *In general, lead times are as high as 2 minutes, with most of them higher than 1 minute after a 20% reduction, not exceeding a 23% false positive rate.*

With a 30% reduction, a few lead times exceed 2 minutes. After phrase reduction, few sequences of phrases were incorrectly identified as failures. Figure 13 illustrates the rise in average false positive rate as average lead times prior to node failures increase. The average lead times of 0.5, 1.1, 1.6, 4.2 minutes are calculated for the cases of no phrase reduction, 20%, 30% and 40% phrase reduction, respectively. Figure 14 shows an increase in the false positive rate in the test data as the amount of phrases reduced is increased from 20% to 40%. Since the false positive rate is more than 30% with 40% tail reduction, even though we could procure lead times as high as 4 minutes, we restricted our experiments to 30% tail reduction. Most indicative logs appear just prior to the failure, and backward pruning increases the false positives by 5%.

Table X shows a partial example where 3 min. 11 sec. lead time could be procured before the node failed at 21:23:26. This is because none of the previous messages were indicative of a failure w.r.t. the learned failure chains. If we aggressively prune more than 30% phrases, TBP can procure as high as 10 minutes lead time in this case.

TABLE X: Lead Time Improvement

Timestamp	Event Phrase	Lead Time
21:13:01.60	slurm_load_partitions: Unable to contact	10min 25secs
21:14:30.83	Lustre:(ptlrpc _expire_one_request...	8min 56secs
21:17:10.37	Unloading nic compatibility device	≈6min
21:19:10.37	bpmcd_exit: No local access to power statistics..	4min 16secs
21:20:15.48	Invoking ..slurm stop ..stopping slurmd:	3 min 11secs
21:21:19.01	slurmd is stopped..	2 min 7secs
21:21:17.01	Unmounting /dsl/.shared/..	
21:22:18.06	Shutting down...	1 min 17secs
21:23:20.10	rpcbind: cannot save any registration..	6 seconds
21:23:21.63	Shutting down DBus daemon..	
21:23:21.63	Removing... SLURM.Failed..	
21:23:21.63	umount /dsl/var/run 1 ...	
21:23:21.63	Unloading XPMEM driver..	
21:23:23.15	Stopping DVS service:	
21:23:23.15	Could not unmount /dsl...	
21:23:26.22	System halted	Node Down
21:23:35.47	cb_node_unavailable	
21:23:35.49	RCADSVCS : shutdown	

E. Feasible Proactive Measures in 2 minutes

We have found that lead times ranging from 20 seconds to 2 minutes can be obtained with an average lead time of more than 1 minute. Live job migration, process cloning, lazy checkpointing [26], and quarantine technique are some relevant actions that can be taken based on node failure information. Wang et al. [29] report that proactive live migration of jobs can range between 0.29 to 24.4 seconds. Rezaei et al. [30] demonstrate node cloning duration within 200 seconds to aid redundant execution during failures. Gupta et al. [24] quantify that 5% to 9% of future failures can be avoided if blades/cabinets are quarantined (no jobs scheduled on the nodes) for some node-hours after a failure is observed on them.

Such proactive recovery based prior work shows that 2 minutes often suffice. Incorrect predictions (FPs/FNs) require a combination of suitable proactive (migrations) and reactive actions (CR) after careful analysis of cost trade-offs [31]. If the unhealthy nodes are known ahead of time, jobs can be delegated on the existing healthy nodes (spare nodes are helpful but not necessary). Future job scheduling can be delayed in case no nodes are available. Additional root cause diagnosis could further increase the lead time confidence. However, a low false positive rate is also a necessity. Further details of proactive actions are beyond the scope of this paper.

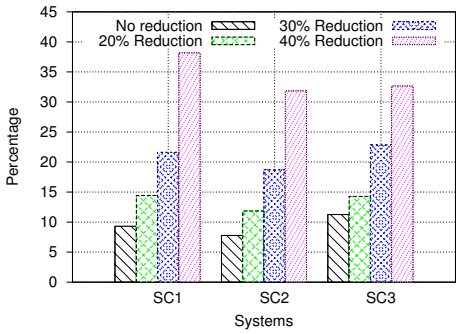

Fig. 14: False Positive Rate

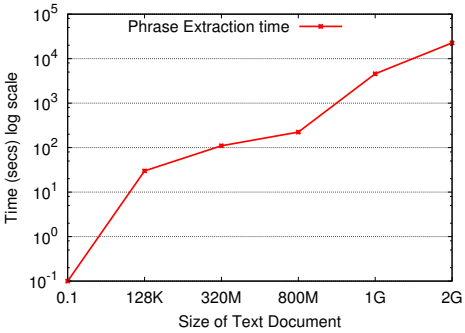

Fig. 15: TBP Scalability

TABLE XI: TBP Impact Assessment

Features	HPC Sys.			Distr. Sys.		
	TBP	[9]	[21]	[22]	[32]	[33]
No Source Code	✓	✓	✓	✓	×	✓
Lead Time	✓	×	✓	×	×	×
Location	✓	×	✓	×	×	×
Prediction	✓	✓	✓	×	×	×
Scalability	✓	×	×	✓	×	✓
Unsupervised	✓	✓	×	✓	✓	✓

F. Case Studies

We assess a case of node failure to highlight TBP's strength and show rare instances where correlation extraction is hard.

1) A True Positive Case: TBP captured an excerpt of messages, two of which are *out of memory* and *killed process* errors. These indicate a node failure correlated to memory errors. This node failure was caused by a CPU group running out of memory. On further investigation we found that the Slurm node had caused the memory crisis. Subsequently, all the processes sharing that cgroup were killed. This caused a machine hang followed by a few Lustre errors. The compute node went down and was rebooted later. In this case, the co-located phrases in the test data were less frequent than the relevant phrases indicative of the failure, thus TBP identified them. Other compute node failures in the cabinet followed due to this Slurm caused hang. This is a true positive case.

2) Difficult Correlation Extraction: TBP missed certain node failures, which do not conform to past seen failures. In those cases, phrase extraction to indicate future failures is hard. They possess similar probabilities for errors that are not fatal. TBP detected some hardware errors (e.g., `correctable aer_bad_tlp`) failed to predict failures related to messages in Table XII. The job-related errors are less frequent but certain errors have probabilities similar to benign messages. We verified one case where the `correctable MCEs` was ranked low so that any $n \leq 150$ excluded it, i.e., TBP discarded it. More research is needed to address these cases.

TABLE XII: Difficult Correlation Extraction

#	Error	Description
1	Console	interrupt took X ns
2	Console	DVS: lnet_mapuvm: page count mismatch
3	Job	Node id has a different configuration
4	Console	logged... correctable MCEs

G. TBP Performance

TBP takes more time to train as the size of the document increases. This is normally the case for any data mining solution. One reason for this is that our logs were not preprocessed, which has the potential to reduce the size of data slightly. Another aspect is the unsupervised learning model of TBP, which operates on unlabeled data. However, text processing (looking for topics in the phrases) takes longer than analysis of just numerical RAS logs of environment data or other features.

Observation 5: *TBP is scalable, taking ≈ 50 msecs to flag a node failure, and below 2 mins to process 320 MB data.*

Figure 15 shows that the phrase extraction time (y-axis, on a logarithmic scale) grows with increasing data sizes. This is due to the complexity of TBP but does not present a problem in practice as TBP would eventually be deployed to process log events within a time-limited window below 320 MB representing 6 or more hours, i.e., its cost would be below 2 minutes. TBP takes ≈ 30 to 50 msecs to detect a node failure during testing. In real-time, even if data gets large too quickly, parallel message filtering approaches can aid rapid failure prediction, which is subject to future work.

H. TBP Comparison

To the best of our knowledge, this work is the first of its kind for node failure prediction on Crays. Table XIII shows some of the proposed failure prediction techniques. TBP achieves as high as 4 minutes lead time with higher false positive and lower recall rate (≈80%). Hence, we chose to restrict ourselves to 30% phrase reduction. Hora [34] and Zheng et al. [21] procure 10 minutes lead time with much lower recall rates. Li et al. [7] and hPrefects [9] achieve prediction accuracy greater than 90% and 70% respectively. Nakka et al. [20] and Hacker et al. [17] predict node failures without lead time analysis using less scalable methods.

TABLE XIII: TBP Comparison

Solutions	Method	Lead Time	Recall	System	Location
Hora [34]	Bayesian Networks	10 mins	18%	Dist. RSS Feed Reader	Component specific
Zheng+ [21]	Genetic Algorithms	10 mins	60%	Blue Gene/P	Rack-level
Li+ [7]	SVM, KMeans	10 mins	N/A	Blue Gene/P, Glory	Component with sensor
hPrefects [9]	Stochastic Model, Clustering	N/A	N/A	256 node HPC cluster	H/W, S/W components
[20]	Decision Tree	N/A	80%	HPC (LANL)	Node-level
[17]	Neural-gas [35]	N/A	N/A	Blue Gene	Node-level
TBP	Topic Modeling	2 mins	86%	Cray	Node-level

Observation 6: *TBP is effective in pin-pointing potential node failures with acceptable lead times and, in contrast to other state-of-the art approaches, without fault injection or source-code reference.*

We further assess the overall impact of TBP in Table XI by highlighting its traits amongst *log mining solutions* proposed in both HPC systems and distributed systems. We observe that failure diagnosis in distributed systems may refer to the source code [32], and even though some solutions are scalable and unsupervised, they lack location and timing

information of anomalies for practical usage. Also, logging practices in distributed systems and software engineering often modify or even add log messages in the source code, and their performance diagnosis is very application- and problem focused. In contrast, we do not augment logging messages as such modifications can only be performed by the vendor at the runtime/operating system layers of HPC systems. Even without such augmentation, we show that predicting failure locations is feasible, in practice.

Discussion: How generic is TBP? TBP has been evaluated on Cray systems, but we believe TBP to be applicable for most contemporary HPC systems. Before developing TBP, we went through BlueGene logs of two different systems. These logs, as researched in the past, are comparatively more structured, have easier to detect failure indicators such that the existing approaches suffice. In contrast, TBP handles more complicated logs in an unsupervised manner. With appropriate integration and pre-processing, TBP can certainly be adapted to non-Cray systems with simpler logs or generic Linux logs (as Cray logs are mostly a superset of those). The recall rates and accuracy will depend on the quality of data and the failure diversity.

VI. Related Work

Several facets of resilience have been studied in the recent past. We present them categorically and subsequently mention how TBP enhances or compliments them.

Log mining tools and failure characterization: [16], [36]–[40] propose useful log mining tools such as HELO, ELSA, LogDiver, LogMine, and LogAider. These aid in studying event pattern-based correlations, spatial/temporal event analysis, and application-level resilience [2] on HPC logs. [41] characterize node failures through temporal and spatial correlations with no anomaly detection. TBP, unlike any event-correlation framework, exhibits efficient failure prediction.

[7], [10], [42] propose data preprocessing mechanisms to extract salient features and show symmetry in system and job logs, to improve fault prediction. [25], [43] study environmental effects on HPC nodes and, analyze memory errors showing their unpredictability and instability in face of scale change. [24] find that the application mean time to interruption (MTTI) can be improved exploiting both spatial and temporal locality by quarantining locations on the Titan supercomputer (Cray XK7). [44] propose a dynamic checkpointing scheme adapting to regime changes based on fault events. [45] show that hardware contributes to only 23% of system downtime, software being the main culprit (74% contribution) of system-wide outages on Cray systems. [28] discuss insights to non-uniform spatial distribution of failures and the fact that temporal recurrence is significantly different for diverse failure types but similar across systems. These are field-data based failure characterization studies and do not perform timely prediction like TBP. [1] study the Trinity platform (Cray XC40) and highlights how understanding system behavioral characteristics is a challenge because of unclear logging and new system features. Hence, even if prior work has shown solutions to similar goals in different systems, they may be invalid on new systems. This reinforces our motivation.

Anomaly detection and prediction: [46] propose a taxonomy of prevalent failure prediction approaches for system reliability. [6], [47], [48] discuss the impact of fault prediction strategies and propose an approach to mathematically model the optimal value of check-pointing. [12], [13], [49] propose techniques to find abnormal nodes employing dynamic training to find changing patterns that improve prediction accuracy. [9], [11], [50] propose online/offline failure prediction frameworks using ideas of principal component analysis (PCA), covariance modeling and void search (indicating scarce faults). [17], [20], [51] apply the gossip protocol, neural-gas method and decision tree classifiers in the context of node failures. These either deal with more structured logs or lack timing analysis, combined with less scalable solutions, unlike TBP. DeepLog [52] uses stacked LSTM for anomaly detection without any lead time analysis, and flags any log key not in top g as an anomaly. In contrast, TBP identifies anomalies in failure chains. [53] design a proactive memory management system analyzing memory logs, successfully preventing 63% of the memory-driven failures. Hora [34] models architecture dependency for component failure prediction through fault injection, unlike TBP. Moreover, TBP's recall rates (83%) are higher than Hora's (73%). [54] study node soft lockups using supervised classifiers, leveraging the node lock/unlock information from the batch history, unlike TBP. They obtain average lead times of 17 and 22 minutes without discussing false positive rates. We also cover more general node failures.

Recovery methods and log mining in distributed systems: [4], [26], [30], [31] propose efficient methods of checkpointing. [29], [55] leverage techniques of process migration for predictable failures. [8], [56] perform root cause diagnosis. These either focus on improving known recovery techniques like migration and checkpoint/restart or have limitations in root cause diagnosis (e.g., assuming that non-fatal events may not manifest as faults in the future) and are complementary to TBP. [22], [32], [33], [57], [58] mine logs in cloud-based distributed systems (e.g., Hadoop, OpenStack) using application console logs adhering to data center concerns. They either refer to the application source code or lack lead time analysis. Moreover, these studies target smaller systems and, unlike TBP, do not address the problem of predicting node failures.

VII. Conclusion

A novel, time-based phrase (TBP) model for node failure prediction is developed leveraging topic modeling for text mining. Our scheme achieves no less than 83% recall rates, 98% precision and as much as 2 minutes of lead time. The paper shows valuable insights to Cray data revealing that service and compute node failures affect user applications considerably. Node failure classification segregates abnormal failures to address significant variations observed in phrases over short time intervals. Continuous time series-based temporal phrase mining approaches are effective in handling unstructured Cray logs and obtain lead times suitable for proactive actions.

ACKNOWLEDGMENT

The authors thank the shepherd and reviewers for suggested improvements. This work was supported in part by subcontracts from Lawrence Berkeley and Sandia National Laboratories as well as NSF grant 1217748. This work used resources of NERSC and was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

This manuscript has three authors of Lawrence Berkeley National Laboratory under Contract No. DE-AC02-05CH11231 with the U.S. Department of Energy. The U.S. Government retains, and the publisher, by accepting the article for publication, acknowledges, that the U.S. Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for U.S. Government purposes.

Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of NSF, DOE or any other national lab.

REFERENCES

[1] J. Brandt, A. Gentile, C. Martin, J. Repik, and N. Taerat, "New systems, new behaviors, new patterns: Monitoring insights from system standup," in *IEEE CLUSTER*, 2015, pp. 658–665.

[2] C. D. Martino, W. Kramer, Z. Kalbarczyk, and R. K. Iyer, "Measuring and understanding extreme-scale application resilience: A field study of 5, 000, 000 HPC application runs," in *IEEE/IFIP DSN*, 2015, pp. 25–36.

[3] E. Elnozahy, R. Bianchini, T. El-Ghazawi, A. Fox, F. Godfrey, A. Hoisie, K. McKinley, R. Melhem, J. Plank, P. Ranganathan *et al.*, "System resilience at extreme scale," *Defense Advanced Research Project Agency, Tech. Rep*, 2008.

[4] J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. B. Ferreira, and C. Engelmann, "Combining partial redundancy and checkpointing for HPC," in *IEEE ICDCS*, 2012, pp. 615–626.

[5] *Top 500 List*. [Online]. Available: https://www.top500.org/list/2017/11/

[6] A. Gainaru, F. Cappello, M. Snir, and W. Kramer, "Failure prediction for hpc systems and applications current situation and open issues," *IJHPCA*, vol. 27, no. 3, pp. 273–282, 2013.

[7] L. Yu, Z. Zheng, Z. Lan, T. Jones, J. M. Brandt, and A. C. Gentile, "Filtering log data: Finding the needles in the haystack," in *IEEE/IFIP DSN*, 2012, pp. 1–12.

[8] Z. Zheng, L. Yu, Z. Lan, and T. Jones, "3-dimensional root cause diagnosis via co-analysis," in *ACM ICAC*, 2012, pp. 181–190.

[9] S. Fu and C. Xu, "Exploring event correlation for failure prediction in coalitions of clusters," in *ACM/IEEE SC*, 2007, p. 41.

[10] X. Fu, R. Ren, S. A. McKee, J. Zhan, and N. Sun, "Digging deeper into cluster system logs for failure prediction and root cause diagnosis," in *IEEE CLUSTER*, 2014, pp. 103–112.

[11] E. Berrocal, L. Yu, S. Wallace, M. E. Papka, and Z. Lan, "Exploring void search for fault detection on extreme scale systems," in *IEEE CLUSTER*, 2014, pp. 1–9.

[12] Z. Lan, J. Gu, Z. Zheng, R. Thakur, and S. Coghlan, "A study of dynamic meta-learning for failure prediction in large-scale systems," *J. Parallel Distrib. Comput.*, vol. 70, no. 6, pp. 630–643, 2010.

[13] Z. Lan, Z. Zheng, and Y. Li, "Toward automated anomaly identification in large-scale systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 2, pp. 174–187, 2010.

[14] L. Yu, Z. Zheng, Z. Lan, and S. Coghlan, "Practical online failure prediction for blue gene/p: Period-based vs event-driven," in *IEEE/IFIP DSN- Workshop*, 2011, pp. 259–264.

[15] Z. Zheng, L. Yu, W. Tang, Z. Lan, R. Gupta, N. Desai, S. Coghlan, and D. Buettner, "Co-analysis of RAS log and job log on blue gene/p," in *IEEE IPDPS*, 2011, pp. 840–851.

[16] Z. Zheng, Z. Lan, B. H. Park, and A. Geist, "System log pre-processing to improve failure prediction," in *IEEE/IFIP DSN*, 2009, pp. 572–577.

[17] T. J. Hacker, F. Romero, and C. D. Carothers, "An analysis of clustered failures on large supercomputing systems," *J. Parallel Distrib. Comput.*, vol. 69, no. 7, pp. 652–665, 2009.

[18] X. Wang and A. McCallum, "Topics over time: a non-markov continuous-time model of topical trends," in *ACM SIGKDD*, 2006, pp. 424–433.

[19] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," in *NIPS*, 2001, pp. 601–608.

[20] N. Nakka, A. Agrawal, and A. N. Choudhary, "Predicting node failure in high performance computing systems from failure and usage logs," in *IEEE IPDPS Workshop*, 2011, pp. 1557–1566.

[21] Z. Zheng, Z. Lan, R. Gupta, S. Coghlan, and P. Beckman, "A practical failure prediction with location and lead time for blue gene/p," in *IEEE/IFIP DSN-Workshop*, 2010, pp. 15–22.

[22] X. Yu, P. Joshi, J. Xu, G. Jin, H. Zhang, and G. Jiang, "Cloudseer: Workflow monitoring of cloud infrastructures via interleaved logs," in *ACM ASPLOS*, 2016, pp. 489–502.

[23] D. M. Blei and J. D. Lafferty, "Dynamic topic models," in *ICML*, 2006, pp. 113–120.

[24] S. Gupta, D. Tiwari, C. Jantzi, J. H. Rogers, and D. Maxwell, "Understanding and exploiting spatial properties of system failures on extreme-scale HPC systems," in *IEEE/IFIP DSN*, 2015, pp. 37–44.

[25] N. El-Sayed and B. Schroeder, "Reading between the lines of failure logs: Understanding how HPC systems fail," in *IEEE/IFIP DSN*, 2013, pp. 1–12.

[26] D. Tiwari, S. Gupta, and S. S. Vazhkudai, "Lazy checkpointing: Exploiting temporal locality in failures to mitigate checkpointing overheads on extreme-scale systems," in *IEEE/IFIP DSN*, 2014, pp. 25–36.

[27] A. McCallum, K. Schultz, and S. Singh, "Factorie: Probabilistic programming via imperatively defined factor graphs," in *NIPS*, 2009, pp. 1249–1257.

[28] S. Gupta, T. Patel, C. Engelmann, and D. Tiwari, "Failures in large scale systems: long-term measurement, analysis, and implications," in *ACM/IEEE SC*, 2017, pp. 44:1–44:12.

[29] C. Wang, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive process-level live migration in HPC environments," in *ACM/IEEE SC*, 2008, p. 43.

[30] A. Rezaei and F. Mueller, "Dino: Divergent node cloning for sustained redundancy in hpc," in *IEEE CLUSTER*, 2015, pp. 180–183.

[31] M. Bouguerra, A. Gainaru, L. A. Bautista-Gomez, F. Cappello, S. Matsuoka, and N. Maruyama, "Improving the computing efficiency of HPC systems using a combination of proactive and preventive checkpointing," in *IEEE IPDPS*, 2013, pp. 501–512.

[32] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *ACM SOSP*, 2009, pp. 117–132.

[33] X. Zhao, K. Rodrigues, Y. Luo, D. Yuan, and M. Stumm, "Non-intrusive performance profiling for entire software stacks based on the flow reconstruction principle." in *Usenix OSDI*, 2016, pp. 603–618.

[34] T. Pitakrat, D. Okanovic, A. van Hoorn, and L. Grunske, "Hora: Architecture-aware online failure prediction," *Journal of Systems and Software*, vol. 137, pp. 669–685, 2018.

[35] T. Martinetz, S. G. Berkovich, and K. Schulten, "'neural-gas' network for vector quantization and its application to time-series prediction," *IEEE Trans. Neural Networks*, vol. 4, no. 4, pp. 558–569, 1993.

[36] A. Gainaru, F. Cappello, S. Trausan-Matu, and B. Kramer, "Event log mining tool for large scale hpc systems," in *Euro-Par*, 2011, pp. 52–64.

[37] A. Gainaru, F. Cappello, and W. Kramer, "Taming of the shrew: Modeling the normal and faulty behaviour of large-scale HPC systems," in *IEEE IPDPS*, 2012, pp. 1168–1179.

[38] 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 *Workshop on Fault Tolerance for HPC at eXtreme Scale, FTXS*, 2015, pp. 11–18.

[39] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, and A. Mueen, "Logmine: Fast pattern recognition for log analytics," in *ACM CIKM*, 2016, pp. 1573–1582.

[40] S. Di, R. Gupta, M. Snir, E. Pershey, and F. Cappello, "Logaider: A tool for mining potential correlations of hpc log events," in *IEEE/ACM CCGRID*, 2017, pp. 442–451.

[41] S. Ghiasvand, F. M. Ciorba, R. Tschüter, and W. E. Nagel, "Lessons learned from spatial and temporal correlation of node failures in high performance computers," in *Euromicro PDP*, 2016, pp. 377–381.

[42] J. Stearley and A. J. Oliner, "Bad words: Finding faults in spirit's syslogs," in *IEEE CCGrid*, 2008, pp. 765–770.
[43] V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi, "Memory errors in modern systems: The good, the bad, and the ugly," in *ASPLOS*, 2015, pp. 297–310.
[44] L. A. Bautista-Gomez, A. Gainaru, S. Perarnau, D. Tiwari, S. Gupta, C. Engelmann, F. Cappello, and M. Snir, "Reducing waste in extreme scale systems through introspective analysis," in *IEEE IPDPS*, 2016, pp. 212–221.
[45] C. D. Martino, Z. T. 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 *IEEE/IFIP DSN*, 2014, pp. 610–621.
[46] F. Salfner, M. Lenk, and M. Malek, "A survey of online failure prediction methods," *ACM Computing Surveys (CSUR)*, vol. 42, no. 3, p. 10, 2010.
[47] A. Gainaru, F. Cappello, M. Snir, and W. Kramer, "Fault prediction under the microscope: a closer look into HPC systems," in *ACM/IEEE SC*, 2012, p. 77.
[48] G. Aupy, Y. Robert, F. Vivien, and D. Zaidouni, "Impact of fault prediction on checkpointing strategies," *CoRR*, vol. abs/1207.6936, 2012. [Online]. Available: http://arxiv.org/abs/1207.6936
[49] L. Yu and Z. Lan, "A scalable, non-parametric method for detecting performance anomaly in large scale computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 7, pp. 1902–1914, 2016.
[50] R. Jia, S. Abdelwahed, and A. Erradi, "Towards proactive fault management of enterprise systems," in *IEEE ICCAC*, 2015, pp. 21–32.
[51] G. Bosilca, A. Bouteiller, A. Guermouche, T. Hérault, Y. Robert, P. Sens, and J. J. Dongarra, "Failure detection and propagation in HPC systems," in *ACM/IEEE SC*, 2016, pp. 312–322.
[52] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *ACM CCS*, 2017, pp. 1285–1298.
[53] C. H. A. Costa, Y. Park, B. S. Rosenburg, C. Cher, and K. D. Ryu, "A system software approach to proactive memory-error avoidance," in *ACM/IEEE SC*, 2014, pp. 707–718.
[54] J. Klinkenberg, C. Terboven, S. Lankes, and M. S. Müller, "Data mining-based analysis of HPC center operations," in *IEEE CLUSTER*, 2017, pp. 766–773.
[55] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive fault tolerance for HPC with xen virtualization," in *ACM ICS*, 2007, pp. 23–32.
[56] E. Chuah, A. Jhumka, S. Narasimhamurthy, J. Hammond, J. C. Browne, and B. Barth, "Linking resource usage anomalies with system failures from cluster log data," in *IEEE SRDS*, 2013, pp. 111–120.
[57] Z. Chothia, J. Liagouris, D. Dimitrova, and T. Roscoe, "Online reconstruction of structural information from datacenter logs," in *EuroSys*, 2017, pp. 344–358.
[58] K. Nagaraj, C. Killian, and J. Neville, "Structured comparative analysis of systems logs to diagnose performance problems," in *Usenix NSDI*, 2012, pp. 26–26.

VIII. Artifact Description Appendix: [Doomsday: Predicting Which Node Will Fail When on Supercomputers]

A. Abstract

Let us describe the major steps required to run the TBP framework and conduct experiments with it. We illustrate how the data is processed, integrated, topics are assigned, node failures chains are learned through training, and data is tested to calculate lead times. The step by step process of generating results clarify the predictor design described in the paper.

B. Description

The artifact description enumerates major packages, software components and language used to formulate the methodology of predicting node failures. In addition, we describe the inputs and outputs of each step.

1) Check-list (artifact meta information):

- **Algorithm: Time, threshold, node identifier based data integration, Time Based Phrase (TBP) Predictor**
- **Program: Factorie, Python, Bash Scripts**
- **Compilation: Maven, No special compilation required**
- **Transformations: Used Topics Over Time API**
- **Binary: Jar executable**
- **Data set: Production logs obtained from real HPC Clusters**
- **Run-time environment: Linux 4.10.13-1.el7.elrepo.x86_64**
- **Hardware: Intel processors**
- **Run-time state: Unused compute nodes**
- **Execution: Command line, Bash Shell**
- **Output: Timestamped events (from the input) pertaining to a node indicating if a failure is probable**
- **Experiment workflow: Job and node data correlation, integration with system logs, selection of topics, training, formulation of failure chains, testing, phrase pruning**
- **Experiment customization: Selection of topics, varying the count of top `N` topics, training with one set of data and testing with another set**
- **Publicly available: Framework upon request, Anonymized excerpts can be accessed from http://moss.csc.ncsu.edu/~mueller/cray/Anonymized_Excerpts/.**

2) How software can be obtained (if available): If accepted, the code can be made available on request via email.

3) Hardware dependencies: Any Intel platform

4) Software dependencies: Python $\geq$3.4.5, Java $\geq$1.7, Apache Maven $\geq$3.0, Scala, Maven should install the correct prerequisite dependencies for Factorie to run.

5) Datasets: Real production datasets, job logs and system logs, were obtained from well used Cray Supercomputing clusters. Representative samples can be seen here.

C. Installation

Install Factorie, python, and run the modules with the correct format of data. Use `mvn compile` to install factorie on which `timebasedphrase` is built.

D. Experiment workflow

In Section III we describe the predictor design. The major steps are as follows:

1) Consider the training data. Enlist all the distinct node ids found in the data, name it "Nodes.txt"

TABLE XIV: Event Phrases

#	Event Phrases
1	DVS: file node down: *
2	mce_notify_irq:*
3	[Hardware Error]: Machine check events logged
4	Corrected memory errors *
5	SOCKET:1 CHANNEL:1 DIMM:
6	HWERR* HAL Completion with Invalid Tag Error
7	LNet: Quiesce start: critical hardware error
8	[gsockets] debug: critical hardware error: *
9	cpu * apic_timer_irqs=*
10	Stop NMI detected on CPU 0

2) From job logs of the same time frame, run:
   `sh correlate.sh Nodes.txt`
   Input: Job logs and Alps logs
   Output: Correlated document of node ids and the job ids scheduled on them
3) Integrate the correlated document with rest of the console logs. Run:
   `sh integrate.sh`
   Input: Output of previous step, console/messages in p0-directories
   Output: Timestamped integrated document of node ids and event phrases
4) Go over the data and select the list of topics. Name it Topics.txt
5) Train the data, run:
   `java -classpath factorie.jar TimeBasedPhrase training_data`
   Input: Topics.txt, output of previous step
   Output: Top N topics referring to event phrases
6) Formulate failure chain, referring to the data:
   `python failure_chain.py`
   Input: Output of previous step, training data, terminal messages leading to node failures
   Output: Node Failure chains
7) Consider the test data. Detect failures in the test data. Run:
   `python detect.py`
   Input: Test data
   Output: Whether a sequence is a failure or not
8) To experiment with different tail reduction lengths of failure chains, choose the % reduction (say A) and run:
   `python pruning.py A`
   Input: Node Failure Chains
   Output: Reduced Failure Chains
9) Repeat Step 7, failure detection.
10) Compute lead time, recall and precision. The test data contains the terminal messages, if any. This helps the lead time calculation.

E. Evaluation and expected result

TBP has been experimented with log sets whose results are explained in the evaluation section V.

1) Table XIV depicts the sequence of events pertaining to a node before forming the failure chain (Section VIII-D Step 3).

TABLE XV: Failure Chains

#	Node Failure Chain 1	Node Failure Chain 2
1	[Firmware Bug]: powernow-k8: No compatible ACPI_PSS objects found	mce_notify_irq:*
2	DVS: verify_filesystem: file system magic value *	[Hardware Error]: Machine check events logged
3	DVS: file_node_down: removing * from list of available servers	Corrected memory errors *
4	Lustre: 29204:0:*	HWERR* HAL Completion with Invalid Tag Error
5	krsip: RSIP server * not responding; *	LNet: Quiesce start: critical hardware error
6	cb_node_unavailable	Stop NMI detected on CPU 0

2) The top topics selected are MCE, HWERR, MEM ERRORS, Lnet, and NMI.
3) After training, the failure chain looks like Table XV, *Node Failure Chain 2* (Section VIII-D Step 6).
4) As seen from both the tables, phrases #1, #5, #8 and #9 got eliminated.
5) For failure detection, these chains are matched with the test data and compared as described in Figure 5 and Table X.
6) The lead time calculation is performed as per the description in the paper (Section V, Lead Time Improvements).

F. Experiment customization

1) Vary the list of topics. The top selected topics over time will change. Accordingly, the accuracy of failure chains changes.
2) Vary `N`, for the top phrases. The number of phrases referred for formulating failure chains will get affected. This will affect the false positives and, hence, the precision. It is observed that most indicative anomalous phrases are within 10 minutes of the failure. If we consider phrases too far ahead of the failure manifestation, it does not help in obtaining better lead times, especially in phrase pruning.
3) The % tail reduction of phrases during phrase pruning to increase lead times can be varied. This affects the false positive rate as mentioned in Section V. Even if more lead times can be procured by phrase pruning, the amount of false positives possible in the considered test data needs to be considered for choosing the percentage of reduction.

G. Notes

Prior to the prediction, the normal node shutdowns are segregated using bash scripts from those that are not anomalous node shutdowns. Run:

```
sh normal_failures.sh
```

The timestamps in the document can be normalized while training. They can be denormalized after obtaining the top N Phrases. While formulating failure chains, we perform per node analysis. E.g., consider node `c0-0c2s0n2`. We refer to the integrated document with phrases pertaining to `c0-0c2s0n2`. Next, we select those phrases from this document which match the top N topics. This aids in forming a timestamped chain of failures. We discard the phrases that did not show up in top N. The terminal messages are known a priori. The failure chains are then formed. Table XV shows failure chains of two distinct node failures. Failure Chain 1 is caused by a firmware bug while Failure Chain 2 is caused by hardware errors and MCEs (machine check exceptions).