Desh: Deep Learning for System Health Prediction of Lead Times to Failure in HPC

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

Charles Siegel*
Cray Inc.
csiegel@cray.com

Abhinav Vishnu*
Advanced Micro Devices, Inc.
abhinav.vishnu@amd.com

ABSTRACT

Today's large-scale supercomputers encounter faults on a daily basis. Exascale systems are likely to experience even higher fault rates due to increased component count and density. Triggering resilience-mitigating techniques remains a challenge due to the absence of well defined failure indicators. System logs consist of unstructured text that obscures essential system health information contained within. In this context, efficient failure prediction via log mining can enable proactive recovery mechanisms to increase reliability.

This work aims to predict node failures that occur in supercomputing systems via long short-term memory (LSTM) networks that exploit recurrent neural networks (RNNs). Our framework, *Desh*[1] (De̲ep Learning for S̲ystem H̲ealth), diagnoses and predicts failures with short lead times. Desh identifies failure indicators with enhanced training and classification for generic applicability to logs from operating systems and software components without the need to modify any of them. Desh uses a novel three-phase deep learning approach to (1) train to recognize chains of log events leading to a failure, (2) re-train chain recognition of events augmented with expected lead times to failure, and (3) predict lead times during testing/inference deployment to predict which specific node fails in how many minutes. Desh obtains as high as 3 minutes average lead time with no less than 85% recall and 83% accuracy to take proactive actions on the failing nodes, which could be used to migrate computation to healthy nodes.

CCS CONCEPTS

• **Computer systems organization** → **Reliability**; • **Computing methodologies** → *Machine learning approaches*; *Neural networks*; • **General and reference** → Evaluation;

KEYWORDS

LSTM, Failure Prediction, Log Mining, HPC, Node Failures, Lead Times, Anomaly Detection, Deep Learning

*Both the authors contributed to the paper when they were researchers at the Pacific Northwest National Laboratory
[1] *Desh* means *Country or Native Land* in Hindi

ACM Reference Format:
Anwesha Das, Frank Mueller, Charles Siegel, and Abhinav Vishnu. 2018. Desh: Deep Learning for System Health Prediction of Lead Times to Failure in HPC. In *HPDC '18: International Symposium on High-Performance Parallel and Distributed Computing, June 11–15, 2018, Tempe, AZ, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3208040.3208051

1 INTRODUCTION

Significant research efforts [23, 33] have been invested on anomaly detection and failure prediction for enhanced system reliability. With the transitioning from the petascale to the exascale computing era, failures in large-scale HPC systems are anticipated to increase and the MTBF (mean time between failures) will decrease, wasting considerable compute capacity [8]. This obviates research efforts to investigate the trade-offs between power, performance and resilience with alternate solutions. Several failure characterizations [24, 25], and machine learning (ML) solutions [22, 33] for anomaly detection exist for large-scale computing systems. Past work identified faults of gradually failing components with hours of lead time [48], but most faults occur within a much shorter window. The state-of-the-art lacks in two key aspects. First, faults need to be predicted even when lead times are short (in the order of minutes), together with their exact fault location. In other words, pin-pointing the component (e.g., which node) of impending failures and to do so just in time so that proactive recovery actions can be taken (such as job migration [39] or quarantining unhealthy nodes [25]) are equally important. Second, the large component count of extreme-scale HPC presents a challenge to data mining techniques such as support vector machines (SVM) [20] or principal component analysis (PCA) [33] due to their limited scalability since prediction has to be performed in real time, and results have to be available prior to the actual failure. Hence, novel scalable and optimized data mining solutions are required. Moreover, the natural language of unstructured logs produced by the computing systems gives rise to two problems. First, since the data lacks any structure and labels, conventional ML techniques suffer from limitations in processing it, e.g. forming feature vectors or classifiers is non-trivial. Second, it is infeasible to infer intricate patterns from high dimensional data quickly, unless the data is processed and fed with appropriate input representation. Deep learning has made tremendous progress in these aspects recently, particularly in natural language understanding [31]. This motivates the need to explore scalable unsupervised deep learning techniques in the context of node failure prediction. Researchers agree that failure prediction is useful even if imperfect and with limited precision [8]. Suppose 50% of the node failures are correctly predicted and the remaining ones are incorrectly predicted (false positives), we can then prevent half of the expensive checkpoint/restarts that require global coordination with much cheaper process migrations.

HPC systems suffer from various kinds of failures at the hardware, software, and application levels. While some failures are critical and obvious to detect, such as kernel panics, most anomalies are not easy to track. Which component will fail and how it will impact the system is not known ahead of time. The observed symptoms of anomalies in the system may or may not reflect the exact root cause. For example, a kernel panic may be caused by a Lustre file system bug or a hardware machine check exception. However, the unwanted consequences, such as node failures, job abortions, etc., can be mitigated if the anomalous patterns are detected well ahead of time by incorporating fast data mining techniques.

This paper proposes Desh, Deep Learning for System Health, to predict node failures using a recurrent neural network technique called LSTM (long short-term memory). Desh obtains an average lead time of more than 3 minutes, which suffices for commonly known recovery mechanisms (see Section 4.6). While deep learning has been investigated extensively in the areas of vision [17] and speech recognition [28], its efficacy in the context of fault prediction and localization for large-scale systems needs more investigation. While DeepLog [18] detects anomalies using LSTM without any lead time analysis, Desh predicts node failures. We discussed several other differences to prior work in Section 4.5.

Challenges: Recent failure prediction approaches either do not consider lead time to failure, use fault injection [38, 45] and synthetic data for evaluation, modify systems with custom log augmentation [47] relying on the source code format, or do not consider the semantic information of the log entries [36]. In contrast, we use real logs from four supercomputing systems without modifications or augmentation as these logs are vendor controlled, and as are the software layers emitting them, i.e., they cannot be modified by us. The efficacy of the approach is determined by investigating which parts of the log are pertinent for failure prediction, by discarding benign events, and by leveraging expert-labeled ground truth. The main challenge in efficient prediction lies in processing the timestamps between two events across nodes correctly to predict accurate lead times. The data has to retain the location information of the anomaly, keeping track of the time differences between correlated events that are generally not adjacent in the logs. This requires a methodology that considers the anomalous phrases leading to node failures and estimates the time of failure based on prior learning.

Contributions: Desh uses LSTM to estimate lead times for impending node failures. We perform phrase analysis of unlabeled log entries, which may or may not belong to the failure chain. Desh uses a novel three-phase deep learning approach to first train to recognize chains of log events leading to a failure, second re-train chain recognition of events augmented with expected lead times to failure, and third predict lead times during testing/inference deployment to predict which specific node fails in how many minutes. The use of time differences between log entries in a chain requires this second re-training step as failure chains are unknown prior to the first training phase, i.e., it is unknown which events of a log are safe or erroneous until the initial training has formed such chains. Only after chains are known can time differences of earlier events in the chain to a terminal log event indicating a node failure be calculated. Based on time differences and along with the phrases, Desh infers failure chains and ultimately reports specific node ids (identifiers) leading to node failures. Based on trained failures, Desh predicts failures with acceptable lead times before a node stops to respond. Thus, Desh not only helps in flagging failures to take recovery actions, it also gives insights as to what phrases indicate node failures based on this statistical analysis.

2 BACKGROUND

Traditional language modeling uses frequency counts of variable length sequences (n-gram model) in a vocabulary of words considering every word an individual unit. In other words, the probability of a word given a history is based on maximum likelihood estimation (MLE); two histories are similar if the last (n-1) words are the same. N-gram models [7] do not correlate semantically close words since words are indivisible. However, recurrent neural networks (RNNs) have the power to predict future data based on sequences of past data considering the semantic closeness since they exploit a distributed representation of words, i.e., word vectors of real values. Hence, in RNNs semantically similar words can be close together in the vector space. LSTMs (long short-term memory) are RNNs augmented with memory and logic gates, known to retain a long-term memory of short-term data chains that represent events correlated in time and space (see [27] for details). LSTMs contain hidden layers, which strengthen their memory persistence. Supercomputing logs have unstructured textual data with short-term failures from seconds (e.g., kernel crash in 20 seconds) up to minutes (e.g., link control block failure in 5 minutes). Moreover, time-stamped logs have diverse events logged in the granularity of microseconds, and patterns evolve over varying intervals of time that have to be remembered over a long time (days to weeks).

Prior Text Mining Techniques: Past solutions based on probabilistic models, PCA/ICA (principal/independent component analysis) [33], and Markov chain and decision trees (e.g., random forests) worked for systems with comparatively more structured logs, where structure aids in feature extraction and offline anomaly detection. They are less efficient when it comes to unstructured text data mining with time constraints. Prevalent approaches such as Support Vector Machines (SVMs) [20] and sequence mining [19] either require complex feature extraction or are unable to capture long-term dependencies making systems intractable with scale. Very recently Coates et al. [13] demonstrated that large-scale training can be done through deep learning on HPC infrastructures with acceptable classification performance and scalable efficiency. LSTM works well for time sensitive data. It can unlearn and relearn over time making it a preferable choice for Desh over other RNNs such as logistic regression and multilayer perceptron (MLP).

Node Failures: This paper uses the term "node failures" frequently. We define node failures as abnormal node shutdowns caused by some system anomaly triggered by software or hardware. These failures differ from intentional maintenance-related massive node shutdowns or periodic service reboots. As any other prediction solution, Desh does not investigate the exact root cause of the anomaly leading to a failure, it instead emphasizes phrase mining to identify impending node failures ahead of time. We have investigated normal service patterns of node shutdowns. Large-scale node reboots clearly indicate service-oriented shutdowns. Desh

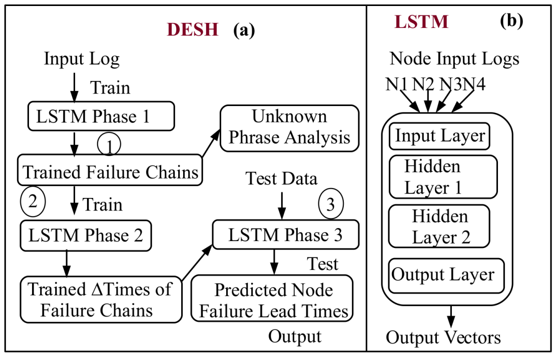

Figure 1: Desh with LSTM

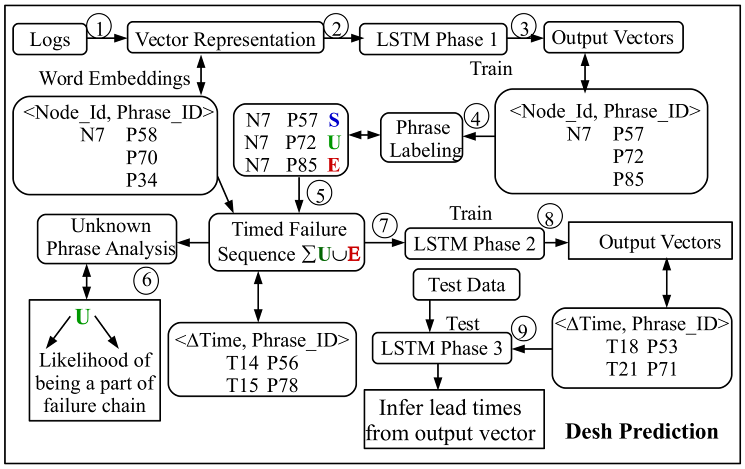

Figure 2: Desh Overview

considers anomalous node failures, that are identifiable by a terminal log message, which is verified in consultation with the system administrators.

3 DESH OVERVIEW

Table 1 provides an overview of the system logs studied, which are collected from four contemporary HPC systems, namely M1, M2, M3 and M4. All of these systems are production level clusters typically running more than 1,200,000 jobs/year and tens of thousands of compute node hours. Duration refers to the time duration of the datasets used for evaluation. Size refers to the log data size and scale indicates the cluster size in terms of the number of nodes. 40% of the top 10 supercomputers belong to the Cray series [2]. Our system logs have been procured from contemporary Cray machines for prediction evaluation.

Table 1: Log Details

System	Duration	Size	Scale	Type
M1	10 months	373GB	5600 nodes	Cray XC30
M2	12 months	150GB	6400 nodes	Cray XE6
M3	8 months	39GB	2100 nodes	Cray XC40
M4	10 months	22GB	1872 nodes	Cray XC40/XC30

These logs are analyzed by our framework called Desh, with its three-phase solution design (Figure 1a). (1) In the first phase, a sequence of phrases leading to node failures is extracted, which trains LSTM phase 1 to learn/recognize such chains based on training data. (2) In the second phase, the formulated chains from phase 1 is fed to LSTM phase 2 to make it aware of the cumulative time differences (Δ times) of just those phrases that belong to a failure chain relative to the terminal phrase in the respective chain. (3) In the inference phase, learned chains from the second phase allow us to estimate the lead times of future failures with an indication of the future location from test data that is disjoint from the training data (Figure 1a). Desh features an RNN with input/output layers along with multiple hidden layers that form a stacked LSTM to facilitate training and prediction on system logs (Figure 1b). Figure 2 elaborately depicts the overall solution design for lead time prediction described in detail below.

3.1 Phase 1: Training

The first phase entails learning failure chains from raw data. The raw logs of Cray/Linux systems contain phrases with anomalies interspersed with considerable amounts of noise and benign events, much in contrast to IBM logs [1]. The phrases with timestamps pertaining to specific nodes are separated. For example, a log message has a timestamp T1, an event phrase P1 and a node N1 (see row 4 in Table 2). Similarly, several such timestamped phrases will correspond to specific nodes.

Tracking the node ids helps to retain the specific failure location in the cabinet/blade/chassis. We train datasets node wise in this phase (see Figure 3a). In other words, logs from each node are concatenated and fed to the same LSTM. This has two advantages. First, there is no overhead of storing the node id and processing it in the vector, which saves memory and computation costs. Second, to learn the patterns of failure chains observed, node identity is of no consequence, what phrases appear in a sequence leading to various node failures is required. We do not predict time in this phase, but sort the log messages by their timestamps for the same reason. The order of the phrases matters here, not their time difference. Time is taken care of in phase 2. As seen in Figure 3, phase 1 uses sequence of phrase ids as the vector.

Each event phrase is then segregated into static and dynamic contents to identify the constant message subphrase separating it from the variable component (e.g., error identifier, IP address) as shown in Table 2. Thus P1, breaks down to its static component and variable component (see Table 2). The dynamic component is discarded. Once the constant messages are extracted they are encoded to a uniquely identifiable number. These phrases are hyphenated multi-word entities. Now, if we have a sequence of encoded phrases, namely, {45, 67, 89, 40} for node N1, LSTM cannot comprehend their semantic or syntactic relevance in this discrete form. Vector space models with distributed representation help establish semantic correlation. These encoded phrases are then vectorized using word embeddings. Embeddings are defined contexts that check what appears before and after a target event phrase in a sequence of events. We use the traditional skip-gram model [34] for word embeddings of TensorFlow [3] to vectorize the data. For all the encoded phrases

Table 2: Phrase Vectors

#	Timestamp (T)	Node Id (N)	Phrase (P)	
			Static	**Dynamic**
1	16:25:48.301744	c1-0c1s1n0	kernel ∗ LNet: hardware quiesce ∗	p0-20141216t162520, All threads awake
2	16:39:59.507009	c4-0c0s0n2	Running ∗ using values from ∗	sysctl, /etc/sysctl.conf
3	00:01:16.704832	c2-0c0s15n2	hwerr ∗ Correctable aer replay timer timeout error∗	[28451]:0x6624, Info1=0x500: Info2=0x18:
4	10:47:39.417963 (T1)	c0-0c0s0n2 (N1)	hwerr *:ssid rsp a status msg protocol err error∗ (P1)	:Info1=0x4c00054064: Info2=0x0: Info3=0x2

Table 3: Phrase Labeling

#	Safe	Unknown	Error
1	Mounting NID specific	LNet: No gnilnd traffic received from	WARNING: Node * is down
2	cpu * apic_timer_irqs	∗ invoked oom killer	Debug NMI detected
3	Setting flag	LNet:∗ gnilnd:kgnilnd reaper dgram check	cb node unavailable
4	Wait4Boot	PCIe Bus Error: severity=Corrected	Kernel panic not syncing
5	Sending ec node info with boot code	ERROR: Type:2; Severity:80;	Stack/Call Trace

in the data, we have a set of phrase embeddings referring to their context such as Lustre, Lnet, Hwerror etc. Using the embeddings and what appears before and after a target, the vector space is formed. Window sizes of 8 and 3 are used, respectively, to consider the number of phrases left and right of a specific target phrase. Desh trains via the stochastic gradient descent optimizer (sgd) with categorical cross-entropy since log analysis is a multi-class problem. Desh predicts the target phrase using the nearby phrases (e.g., for sequence {45, 67, 89, 40, 89, 102}, it provides the probability that 40 is observed when {45, 67, 89} appears before it and {89, 102} appears after it). This conforms to the vector representation (2) in Figure 2. Now, these vectors are fed into the stacked LSTM using two hidden layers (3) to perform 3-step prediction (to predict the next 3 phrases). A larger history size and a higher number of hidden layers increase accuracy, but also the computation time. Experimentation proved 3-step prediction with 2 hidden layers, to have ≈85% accuracy taking ≈0.65 milliseconds in time. More than 1 hidden layer strengthens LSTM's efficacy to remember past phrases to make predictions. This unsupervised LSTM phase 1 training emits the trained sequences of phrase vectors.

Phrase Labeling: From these vectors, we decode the phrases to filter (4) them into three categories: safe, error and unknown as shown in Table 3. *Safe* represents the benign phrases, which are definitely not related to any system anomaly (e.g., *Wait4Boot*). *Error* refers to those phrases, which are definitely indicative of some anomaly (e.g., *Stack Trace*). The *Unknown* tag is given to those phrases that may or may not be indicative of some anomaly (e.g., *PCIe Bus Error: severity=Corrected*). It should be clarified that tagging a phrase as Error does not imply that it will always be a part of node failures, it may or may not be a part of node failure. However, these phrases are either terminal messages (e.g., Stop NMI detected) or major hardware, software malfunctioning (e.g., page faults) seen in Linux logs. We do not consider the log severity levels even if present. This phrase grouping is based on consultation with the system administrators. After categorization, we have phrases with Safe (S), Error (E) or Unknown (U) labels. Safe (S) phrases are eliminated now, since our primary interest is in the error and unknown phrases.

Phrase labeling is deliberately not done before vectorization since training is more robust with noise. Moreover, indicators for erroneous or benign events are not an a priori for unsupervised learning. Desh incorporates labeling to optimize computation costs in phase 2 to determine the likelihood that unknown phrases lead to node failures. A sequence of events leading to a node failure is formed using Unknown (U) and Error (E) tagged phrases after referring to the raw data, since terminal messages indicating a node going down are known (e.g., Shutdown events, cb_node_unavailable). Desh forms trained failure chains (5) learning contextually relevant phrases temporally close as shown in Table 4. At this juncture, we evaluate statistically how certain unknown phrases form a failure chain, while others never appear in any chain. This likelihood estimation unveils insights to log messages eventually leading to node failures. We discuss the analysis of unknown phrases in Section 4.3.

LSTM Phases: Figure 3 depicts a unified view of the three phases. Desh phase 1 and phase 2 are training phases that process the vectors concatenated, i.e., one node after the other (Figures 3a and 3b). Desh learns failure sequences from different nodes sequentially. The difference is the contents of the input vector for each log message pertaining to a node. While phase 1 contains only phrase ids, phase 2 contains the time differences between phrases along with phrase ids (Section 3.2). In phase 3, Desh detects failures based on the previous training applied to the new test data (different from the training set). As seen in Figure 3, the vectors fed to LSTM phase 3 are from a specific node (not concatenated). The vectors contain time differences (ΔT) of phrases along with phrase ids similar to phase 2 (Section 3.3). The idea behind this is to first learn from all nodes and use that assimilated learning to detect failures per individual node during testing and inference.

3.2 Phase 2: Training

The objective of the second phase is to predict lead times based on the learned failure chains. In phase 1, the presence of noise makes ΔT calculation infeasible, since we need to consider anomalous phrases leading to node failures without interspersed Safe phrases. In this phase, we segregate the phrases forming the failure chains from the rest (not part of a failure chain), and compute the time differences between phrases in the failure chain to enable lead time prediction. Desh then trains multiple failure chains to learn the

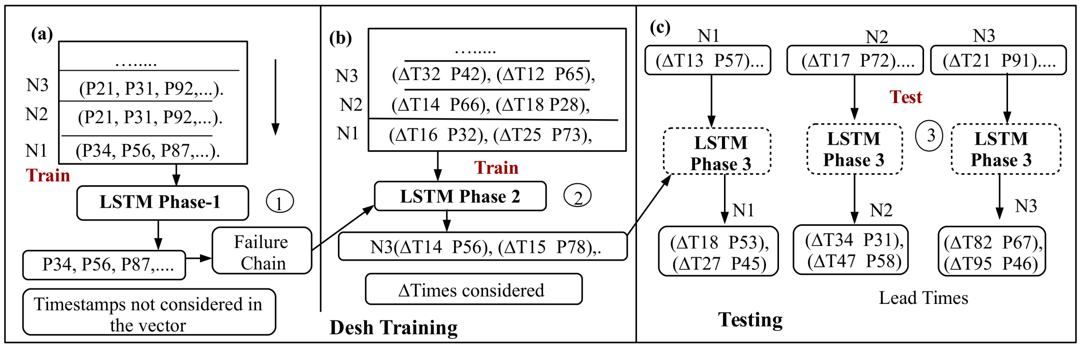

Figure 3: LSTM Phases

Table 4: Example Failure Chain

#	Timestamp	Phrase	Label	Phrase Vector
P1	03:59:58.466 (T1)	CPU *: Machine Check Exception:	U	ΔT1=07.822, P1
P2	03:59:59.543 (T2)	[Hardware Error]: Run the above through 'mcelog –ascii	U	ΔT2=06.745, P2
P3	04:00:00.477 (T3)	[Hardware Error]: RIP !INEXACT! 10:	U	ΔT3=05.811, P3
P4	04:00:01.706 (T4)	Kernel panic - not syncing: Fatal Machine check	E	ΔT4=04.582, P4
P5	04:00:01.731 (T5)	Call Trace:	E	ΔT5=04.557, P5
P6	04:00:06.288 (T6)	cb_node_unavailable	E	ΔT6=00:000, P6

Table 5: LSTM Parameter Specifications

#	Input Vector	Output Vector	#HL	Steps	#HS	Loss Function, Optimizer
Phase-1	(P1, P2..PN)	(P11, P15..PN)	2	3	8	SGD, categorical crossentropy
Phase-2	(Δ T1, P1), (Δ T2, P2,..)	(Δ T11, P11), (Δ T22, P22,..)	2	1	5	MSE, Rmsprop
Phase-3	(Δ T4, P4), (Δ T5, P5,..)	(Δ T15, P15), (Δ T16, P16,..)	2	1	5	MSE, Rmsprop

diverse ΔTs with phrase ids to eventually predict what times are expected in the future. This enables the capability of lead time prediction of Desh.

We know the timestamps and the phrases (either U or E) pertaining to a detected node failure from phase 1. Table 4 demonstrates how the time differences between the phrases in the failure chain are converted to an input vector for LSTM phase 2, for a specific node failure. This failure was caused by hardware processor corruption. Here, a CPU experienced a hardware machine check exception, followed by kernel panic with a call trace, after which the node failed. Table 4 enlists a few phrases for brevity.

ΔTime Calculation: Our target is to predict the lead time to a node failure. Anomalous messages precede terminal messages in a failure sequence. The manifested failure is indicated by the higher order time-series. Hence, we sort the data in descending order of timestamps and calculate ΔTs, which is the cumulative time difference between the current phrase and the last phrase (highest order) in the sequence. The highest timestamped phrase in the sequence is assigned ΔT=0 since there is no phrase left in the sequence to calculate the time difference. For example, in Table 4, ΔT6 is assigned 0. Next, we compute the ΔTs subtracting timestamps of every phrase from T6 (e.g., T6-T5, T6-T4, T6-T3 etc.) in the failure chain, respectively, and obtain the time differences in seconds (7.822, 6.745, 5.811, 4.582, 4.557, 0). The input to LSTM phase 2 is a 2-state vector with the ΔT and the phrase id, as shown in the Phrase Vector column of Table 4. As seen in Figure 3, the vectors are concatenated across node logs, but with added Δ times unlike phase 1. Training on these time differences helps LSTM learn how late the terminal phrase is expected to appear in the sequence based on the previously seen phrases. In phase 2 the LSTM is fed with these vectors with a history size of 5 to perform 1-step prediction for every sequence with 2 hidden layers.

This training performed offline on multivariate time-series expects the predicted value to be close to the actual value seen in the training data, thus the objective function employed is the mean squared error (MSE) loss minimization combined with the RMSprop optimizer.

3.3 Phase 3: Testing

In the third phase, LSTM is used on the test data to validate trained failure chains from Phase 2. The test data is processed to form encoded vectors with time differences and phrases similar to the discussion of Table 4 in Section 3.2. It should be noted that here we need to track the node id to know which node is expected to fail with how much time is left before any failure. Please note in Figure 3c, the vectors are not concatenated across nodes as in phase 1 and 2. Instead, the log of each node is passed to an identical trained LSTM. Sequences of vectors containing ΔTs and phrase ids pertaining to nodes are formed from the test data. We form batches corresponding to distinct nodes with their sequence of phrase vectors. Suppose we have 100 distinct nodes in the test data, then we have 100 batches of size M, i.e., M 2-state vectors in each batch. This represents node ids in a way that saves computation cost and conforms to the input vector format. LSTM uses this test data and the target data obtained from the previously trained failure chains for evaluation. LSTM predicts the next sample from the trained data (failure chain), compares with the vector in the test data and computes the MSE. Notice that this is not a binary classification problem. While validating phrases in a sequence the prediction if a node failure will happen is determined by trying to find a close match to the actual target failure chain.

We use a threshold of 0.5 for inferring node failures. In other words, when LSTM obtains MSE≤0.5, we consider those outcomes to check for failure. Based on experimentation, more than 0.5 MSE in the test data emitted chains that are quite dissimilar from those in the trained failure chains. Suppose the prediction is {P1, P3, P7} and the failure chain is {P1, P3, P8}. We may consider it a failure and cross validate with the ground truth data. We can then extract the ΔTs from the vectors and compute the predicted lead times. Let us say we have {(6.2, P4), (2.57 P6), (1.4, P9)}, with time in minutes, then a lead time of 4.8 minutes is predicted before the node fails. Overall, Desh obtains node failures with as high as 87.5% recall and an avg. of 19.43% false positives. Table 5 depicts the major parameters of LSTM training for Desh in phases 1, 2 and 3. The input/output vectors show the details of the vector entries; HL indicates the number of hidden layers used; Steps refers to the number of steps of prediction, i.e., how many samples to predict based on the history provided; and HS refers to the History Size, the window size of samples given during training based on which prediction happens. Lastly, the loss function and optimizer required for LSTM is indicated per phase.

4 EVALUATION

We have built a prototype implementation of Desh using Keras [11] with a TensorFlow [3] backend. Our evaluation uses actual system logs of several supercomputing machines to demonstrate the efficacy of Desh. Desh obtains as high as 3 minutes lead time, with no less than 83.63% accuracy. We split the dataset for all the systems for training and testing. 30% of the data is used for training and the remaining is used for testing. The experiments are performed on the Intel platform. Our experimental evaluation quantifies the prediction efficacy of Desh in terms of the standard performance metrics, performs lead time sensitivity and determines the implications of unknown phrase analysis in the context of system reliability.

4.1 Prediction Accuracy

Table 6 tabulates the metrics used for statistical analysis, namely, recall, precision, accuracy, F1 score, false positive rate and the false negative rate. Their corresponding formulas are indicated in Column 2, where TP = True Positives, FN = False Negatives, TN = True Negatives and FP = False Positives, respectively. Correctly predicted failures are true positives, incorrectly predicted failures are false positives, failures missed by Desh are false negatives, and the sequence of phrases not predicted by Desh as failures, which are actually not failures, are true negatives. Additionally, we have computed the F1 score, which evaluates Desh's failure prediction accuracy considering the weighted average of recall and precision.

Figure 4 illustrates that, both accuracy and F1 score are relatively high for all the systems. While the recall rates do not vary much (85.10% through 87.5%), M4 has comparatively low precision.

Observation 1: *Desh has ≥84% precision, ≥83.6% accuracy and ≥85.7% F1 score along with as high as 87.5% recall rates across all the four systems.*

This implies that the information gained from the learned failure chains aided Desh in making accurate predictions for the overall test data. In other words, new patterns or unknown failures are rare, hence, our model was a good match.

The failure chains were reliable to sustain the model's predictive power over the events encountered during testing. Another reason for Desh's performance is the history window size is 5 to 8 in Desh. More history improves accuracy consuming more time. Reducing the history size to 3 brings down the accuracy by 10% to 14%.

Table 6: Prediction Efficiency

Metrics	Formula
Recall (%)	TP/(TP+FN)
Precision (%)	TP/(TP+FP)
Accuracy (%)	(TP+TN)/(TP+FP+FN+TN)
F1 Score (%)	2*(Recall*Precision)/(Recall+Precision)
FP Rate (%)	FP/(FP+TN)
FN Rate (%)	FN/(TP+FN), (1-Recall)

The false positive and false negative rates for the systems are shown in Figure 5. While the false positive rates range from 16.66% to 25%, the false negative rates do not exceed 15%. They vary between 12.5% to 14.89% indicating that Desh is effective in not missing actual failures. M1 has a higher false positive rate and higher true negatives. For M1, incorrectly flagged failures were higher. Since the overall accuracy and F1 score are high, a 25% false positive rate is acceptable and Desh is capable of predicting most of the node failures.

Table 7: Node Failure Classes

#	Class	Failures	Avg. Lead Times(secs)
1	Job	Job scheduler (Slurm)-based errors, Task/Application related bugs	81.52
2	MCE	H/W Machine Check Exceptions, Page/Memory Faults, Processor Corruptions	160.29
3	FileSystem (FS)	Lustre/DVS Bugs, Packet/Protocol Errors	119.32
4	Traps	Segmentation Faults, Trap invalid opcode	115.74
5	Hardware (H/W)	NMI Faults, critical hardware errors, Node heartbeat errors	124.29
6	Panic	Stack Trace, Kernel Panic	58.87

4.2 Lead Times

Our research goal is to obtain sufficient lead times while retaining the prediction accuracy. We have analyzed lead times across three dimensions, seeking to answer the following questions:

- How does the diversity of node failures affect the lead times?
- How high is the average lead time for each of the systems?
- How sensitive are the lead times w.r.t. the false positive rates?

To this end, we classify node failures considering their predominant context of failures. We have investigated various chains leading to failed nodes and determined the prominent phrases causing anomalous node shutdowns. Table 7 enumerates those classes evaluating the sequence of events. `Job` class refers to the node failures caused by the slurm job scheduler due to slurm controller connectivity problems or resource outage caused by application aborts etc. `MCEs` refer to hardware machine check exceptions frequently encountered by compute nodes causing processor corruptions, memory faults and other hardware interrupts. `FileSystem` bugs are mostly Lustre (parallel file system) errors, mount problems of DVS

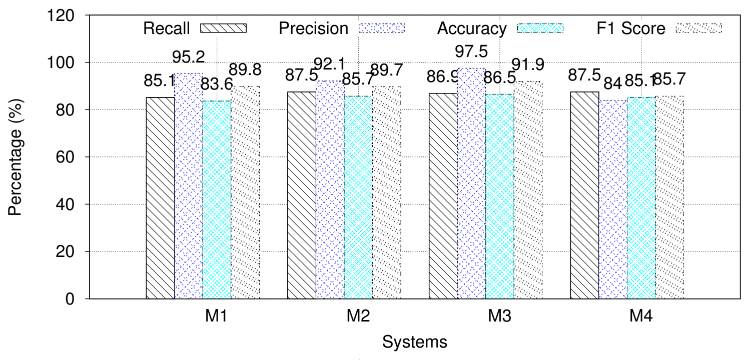

Figure 4: Prediction Rates

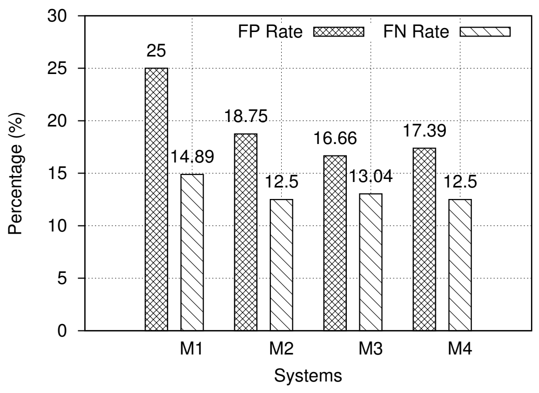

Figure 5: FP Rate and FN Rate

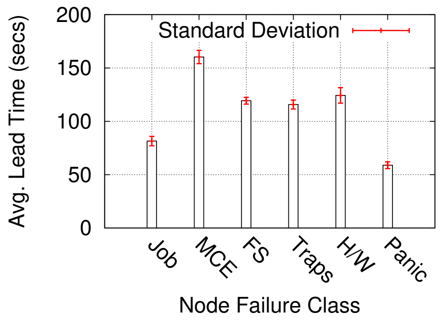

Figure 6: Lead Times+Failure Classes

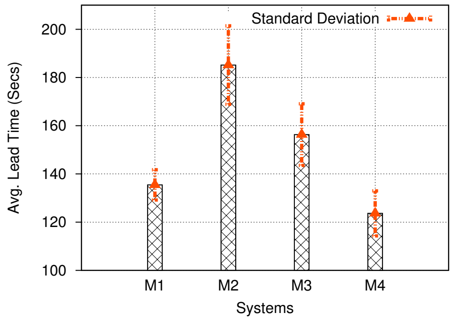

Figure 7: Avg. Lead Times of Systems

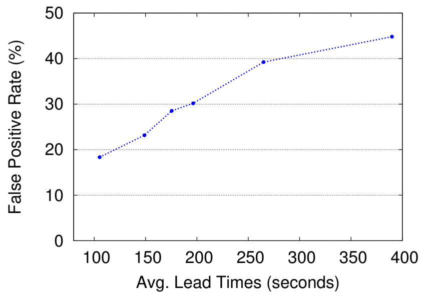

Figure 8: Lead Times and FP Rate

(data virtualization server) etc. Several bit errors, packet and protocol errors are also considered in this class, which are common in many failure chains. While `Traps` are typically software interrupts, exceptions, segmentation faults and invalid opcode errors, the `Hardware` errors are node heartbeat fault messages, NMI faults, interconnect failures etc. Lastly, a kernel `panic` followed by a stack trace can be triggered by both hardware faults and software exceptions. Kernel panics commonly cause nodes to fail. Figure 6 helps to answer the first question. The standard deviation as seen is low. It is intuitive that kernel panics do not have high lead times since the anomalies happen just prior to the failure, without enough lead time for proactive job migration. Their avg. lead time is ≈1 minute. `MCE`s and `FileSystem` failures have comparatively higher lead times, job scheduler-based failures are rare with ≈82 seconds lead time.

Observation 2: *Based on the class of failures, procured lead times differ. LSTM training is efficient when trained data contains major failure classes with correlated log messages preceding an actual terminal message where node stops responding.*

Figure 7 shows the average lead times of each of the 4 systems with their standard deviations. M2 has higher lead times than the rest since M2 features more node failures caused by `Hardware` and `Filesystem` classes and fewer `kernel panics`. All the systems have more than 2 minutes average lead time. The standard deviations of lead times of a specific failure class is lower than it is in a specific system. On investigation we found that, a system has higher variations of failures with diverse failure class, hence the obtained lead times vary. For all the node failures of the same failure class, the deviation in lead times are not that high as seen in Figure 6. Figure 8 shows the lead time sensitivity. Let us recall the cumulative ΔT calculation Desh performs in phase 2 training, as described in Section 3.2. Suppose, we have a sequence of events in the test data, {(4, P1), (3.1, P2), (2.5, P3), (0, P4)} with ΔTs (in minutes) and phrases in each sample vector. Now, if Desh predicts a failure while testing P4, we obtain 0 lead time (P4 is a terminal message), if a failure is flagged after checking P3 we get 2.5 minutes lead time. In other words, the earlier we flag the longer the lead time. The caveat is that there are several other sequence of events similar to a target failure chain not leading to a failed node. In those cases, if failure is flagged after checking P2 or P1, we obtain 4 minutes lead time at the expense of an increasing false positive rate. This makes the sensitivity study important. We aim at longer lead times, yet need to limit the false positive rate. Figure 8 indicates that a false positive rate in the range of 18% to 30% results in a lead time of 105 seconds to 196 seconds. As the lead time increases to more than 4 minutes, false positives rise to 39%, finally reaching 44% with ≥6 minutes lead time.

Observation 3: *Desh obtains more than 2 minutes lead time with acceptable false positive rate (16.66% to 25%) and false negative rate (12.5% to 14.89%) across all the 4 systems.*

Observation 4: *The standard deviation of lead times to node failures of a specific failure class is lower than the standard deviation across all node failures in a system indicating that different failure classes have unique and reproducible lead times to failure.*

Table 8: Unknown Tagged Phrases

#	Phrase	(%)
P1	LustreError *	56
P2	Out of Memory/Killed Process	15
P3	Lnet: Critical H/W error	36
P4	Slurm load partitions error: Unable to contact slurm controller	42
P5	hwerr[*]: Correctable AER_BAD_TLP Error *	12
P6	Sent shutdown to llmrd at process *	17
P7	AER: Multiple corrected error recvd *	21
P8	Trap invalid code * Error *	8
P9	modprobe: Fatal: Module * not found *	27
P10	<node_health> * Warning: program * returned with exit code *	29
P11	DVS: Verify Filesystem	60
P12	BUG: unable to handle kernel NULL pointer dereference	25

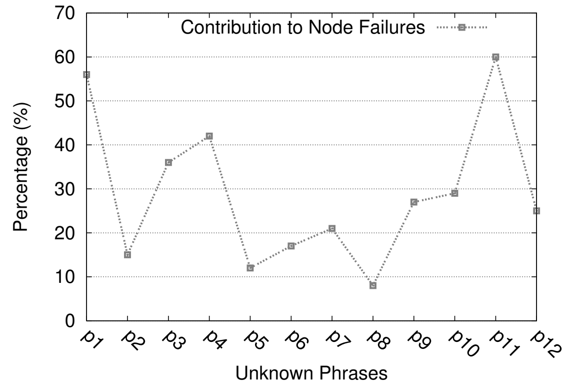

Figure 9: Unknown Phrase Analysis

Table 9: Unknown Phrases with and without Node Failures

#	Failure 1	Failure 2	Not Failure 1	Not Failure 2
1	H/W Error: MCE Logged	LustreError *	nscd: nss_ldap reconnected	LustreError: Skipped
2	Corrected Memory Errors on Page *	DVS: Verify Filesystem: *	<node_health> program * returned with exit code *	nscd: nss_ldap reconnected
3	<node_health> program * returned with exit code *	DVS: * no servers functioning properly	Trap Invalid Code	Hw Error: MCE Logged
4	mce_notify_irq: *	Startproc: nss_ldap: failed..	Killed process *	Corrected DIMM Memory Errors
5	Lnet: critical hardware error: *	Stop NMI Detected	Out of memory *	MCE_notify_IRQ
6	[Gsockets] debug [0]: critical h/w error	Slurm load partitions error: Unable to contact slurm controller	Lustre: * binary skipped *	Lnet: H/W Quiesce
7	Stop NMI Detected	Slurmd Stopped	hwerr[*]: RSP A_status_msg_protocol_error*	Corrected Memory Errors on Page *
8	<node_health> warning: * node is down	System: halted	<node_health> * failures: The following tests * failed	Lustre: * connected to *

4.3 Unknown Phrase Analysis

Table 8 depicts a subset of commonly encountered phrases in Cray logs that are not individually benign or erroneous by itself but can manifest as failures based on other system events. Their percentage of appearance in node failure chains is indicated in Column 3. Some phrases are seen in many abnormal node shutdowns such as LustreError (P1) and DVS: Verify Filesystem (P11). It is interesting to note that an appearance of anomalous messages such as software trap (P8) or network critical hardware error (P3) do not necessarily result in *failing nodes*. We have observed these phrases in sequences of events pertaining to nodes without unusual node shutdowns during those time frames. We distinguish between anomaly-based node failure versus intended node shutdowns such as maintenance service or periodic reboots. Those have simpler patterns in manifestation. Figure 9 demonstrates that while hardware correctable errors (P5) and out of memory/killed process errors contribute less to failures, Lustre filesystem bugs are quite common. This is because several hardware faults trigger software faults that in turn trigger other software errors, even if the root cause may be a hardware error (conforms to the observation by Gainaru et al. [21]). This observation is subject to the events during the time frame considered. Even if the statistics vary based on the target systems and datasets used, this estimate drives home an important point, namely that the presence of software traps or critical hardware errors does not always lead to node failures, if their cause can be corrected eventually. Our aim is not to perform root cause diagnosis here. In contrast, erroneous phases such as kernel panics/stack trace, NMI faults, CPU stalls and several hardware machine check exceptions (MCEs) do result in failed nodes.

Table 9 depicts 4 sample sequences of events. The first two are node failures caused by anomalies, the last two are not node failures. In fact, node shutdown messages were absent during those time frames. We pick snippets of important phrases to highlight cases where such unknown phrases may or may not lead to an anomaly. The first node failure was caused by too many hardware machine check exceptions causing the CPUs to get corrupted. The non-failure case 1 (3rd column) shows a sequence of messages that did not eventually cause any node failure, although the node encountered traps, followed by jobs getting killed and hardware protocol errors. Note that the 10th phrase (program * returned with exit code) from Table 8 is present in both a failure case (1st column) and a non-failure (3rd column). The 2nd node failure was caused by filesystem bugs and a slurm controller connectivity error. The non-failure case 2 (4th column) shows Lustre errors in the beginning, yet the node endured several MCEs and DIMM memory errors without failing immediately. In fact, hardware MCEs and corrected memory errors were logged in both failure (column 1) and non-failure (column 4) cases, as are Lustre errors (column 2+4). Overall, it is clear that phrase mining-based anomaly detection is non-trivial and appearance of anomalous phrases neither indicates root causes nor certainty of eventual node shutdowns. We have provided an estimate of how much the unknown phrases contribute to a node failure. Terminal messages such as *Stop NMI detected* usually appear whenever a node goes down (either normal or anomalous), *debug NMI detected* is more often seen with anomalies. What are the implications of these insights? Most vendors and HPC administrators expect accurate failure indicators to save resources, what messages lead to failures may not be anyone's concern. However, such characterization implies the following observations:

Observation 5: *A log message with a given phrase may be benign in one context while it is part of a failure chain in another one, or it may lead to a fault that is later corrected so that the fault is masked resulting in no failure.*

Major hardware bugs and software panics are known to cause failure [24] in HPC systems. But in-spite of the appearance of similar messages (traps, critical hardware errors), nodes do not always fail. Events external to a specific node (e.g., interconnect bugs, temperature conditions) can cause such bugs and there exists conditions under which these faults do not cause failures. This is different from the known spatial correlation [25] that node failure correlation is higher within the same cabinet than a blade. It will be interesting to determine what faults under what condition do not cause failures.

Observation 6: *In general, tags such as warning or critical with a log message should not be uniquely associated with a log event as the context of correlated events in time and space in a failure chain is indicative of anomalies, not a single event by itself.*

In the past, researchers have heavily relied on *fatal* severity level to formulate detection schemes. With contemporary system logs, understanding the sequence of events enhances machine learning solutions.

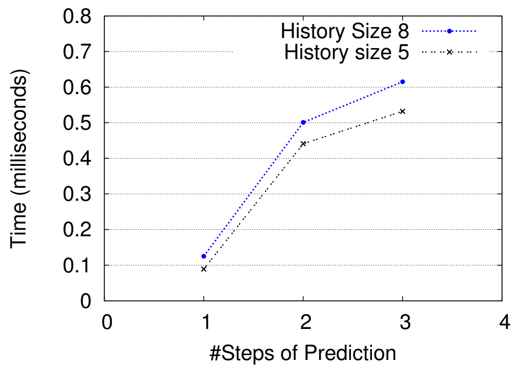

Figure 10: Cost Analysis

4.4 Cost Analysis

Desh operates in three phases as discussed earlier (Section 3). The training phases 1 and 2 are performed offline and have no consequence to the prediction performance. However, we report the time taken to perform predictions based on the history size. As shown in Figure 10, with a larger history window, the time taken is slightly longer. As expected, 3-step predictions take longer than 1-step prediction. The times taken by 2-step predictions are comparable for both the history sizes of 8 and 5. The size of the vector entries and computing platform used to run LSTM also determine the cost. Optimizing detection speedup is not Desh's goal. Nevertheless, the prediction time can be insightful to provide further optimizations in log mining solutions.

4.5 Desh Comparison

The closest work related to Desh is DeepLog [18]. DeepLog uses stacked LSTM on HDFS, Openstack and BlueGene logs to detect anomalies. However, there are fundamental conceptual differences that makes Desh's application of LSTM unique. We discuss them briefly here:

• DeepLog injects anomalies for cloud systems (e.g., during VM creation). Desh considers failures extracted from real field data, with no synthetic injection performed. Their anomaly detection with BlueGene/L RAS logs is similar to our Cray system log analysis. However, their log labeling and inference of anomaly differs substantially. Their normal and abnormal labeling relies on normal execution patterns to detect deviations for abnormal messages, which is not our work. Moreover, these logs are more structured than Cray logs, which are unstructured because of diverse log sources. For example, in Table 12, the first two phrases are considered abnormal and the last two normal based on the a priori information about BlueGene/L system event logging. Desh does not consider an individual message as abnormal unless it contributes to a node failure chain. We also explicitly demonstrate that severity labels are insufficient indicators of system malfunctioning (Section 4.3). Although DeepLog's workflow construction is analogous to our failure chain formation, they consider performance anomaly on a per-log entry level, i.e., if the actual value does not appear in the top g predicted keys, it is considered an anomaly. Our failure definition is more refined, it is based on a sequence of event vectors observed. An individual phrase by itself does not determine an anomaly, since the node failures are flagged based on the trained failure chains.

• We design LSTM to predict lead times and track the node ids to pin-point failure location. In other words, Desh can warn, `In 2.5 minutes, node X located in Y is expected to fail`. The node id (e.g., cA-cBcCsSnN) contains the exact location information (cabinet: AB, chassis: C, blade: S, number: #N). This indication can prevent further scheduling of jobs on node X, existing applications can be migrated from that node to another healthy node. Such proactive actions can mitigate service disruptions and future job failures. DeepLog is not designed to predict lead times, it can detect performance anomalies in the system to aid diagnosis. Desh aims to strike a good balance between lead times and false positives. Increasing lead times hurts the false positive rate. Instead, acceptable lead times with low false positive rates are desirable.

• Apart from the differences in anomaly definition (per log entry level vs. sequence level) and research goals (performance anomaly detection vs. node failure prediction), HDFS and Openstack logs are at a higher software layer (atop native filesystems and JIT engines) than low-level Linux-style Cray logs. Hence, performance anomalies recurring in such systems are different from the hardware critical errors appearing in HPC systems. Moreover, Desh exploits word embedding for vectorization along with cumulative Δ time calculation, differing from DeepLog's solution paradigm.

We have further enumerated the major capability differences between DeepLog and Desh in Table 11. DeepLog has performed online model updates to improve false positive rates and does not

Table 10: Desh Comparison

Solutions	Method	Lead Time	Recall	Precision	Anomaly Injection	System	Location
Hora [38]	Bayesian Networks	10 mins	83.3%	41.9%	✓	Dist. RSS Feed Reader	component specific
Gainaru et al. [21]	Signal Analysis	N/A	60%	85%	×	Blue Waters	N/A
Islam et al. [29]	Deep Learning	N/A	85%	89%	×	Google Cluster	Job-level
UBL [14]	Self-Organizing Map (SOM)	50 secs	N/A	N/A	✓	RUBiS, Hadoop, System S	N/A
CloudSeer [45]	Automatons, FSMs	N/A	90%	83.08%	✓	OpenStack	N/A
Desh	Deep Learning	3 mins	86%	92.2%	×	Cray	node-level

Table 11: Desh vs. DeepLog

#	Features	Desh	DLog
1	No Source-Code	✓	✓
2	Lead Time	✓	×
3	Component location	✓	×
4	Sequence-level Anomaly	✓	×
5	Injected Failures	×	✓
6	Node Failures	✓	×
7	Cloud+HPC	×	✓
8	False Positive Rate	✓	×

Table 12: BlueGene/L Log

#	Log Message	Label
1	kernel Info total of 2 ddr error(s) detected and corrected	Abnormal
2	kernel Info CE sym 9, at *, mask *	Abnormal
3	App fatal ciod: Error creating node map	Normal
4	kernel fatal MailboxMonitor::serviceMailboxes	Normal

aim at tracking the component location. We have performed lead time characterization w.r.t. false positive rates.

Apart from DeepLog, there exist several state-of-the-art failure prediction solutions for system resilience. We have compared Desh with a few of those solutions in Table 10. CloudSeer performs automaton-based workflow monitoring on OpenStack, and UBL [14] predicts performance anomalies exploiting self-organizing maps, both using anomaly injection. UBL (Unsupervised Behavior Learning) calculates lead time as the difference between the points in time of actual SLO (service level objective) violation and of anomaly detection for Hadoop, RUBiS and System S using application logs. In contrast, Desh predicts lead times of node failures using LSTM for low-level system logs. Hora uses fault injection to perform prediction in the Netflix RSS feed reader, which is architecturally different from HPC systems. Islam et al. [29] uses LSTM like Desh, obtains high precision and recall but targets the job failure problem rather than node failure. Gainaru et al. [21] integrates their older research tool, ELSA [22], to perform online anomaly detection on Blue Waters. They obtain comparatively lower recall (60%) rates. Overall, Desh demonstrates a novel way of predicting node failures with low false positives and high accuracy.

4.6 Discussion

With increase in scale of production computing systems, the mean time between failures (MTBF) is expected to decrease. Failures are expected to occur in shorter intervals of time [40]. Failure lead time prediction can decrease the failure rate in large-scale systems. Desh incorporates stacked LSTM efficiently to have high prediction accuracy (85.71%), acceptable false positive rates (19%) and 3 minutes lead time warnings. Unknown phrase analysis further reveals the prospects of understanding system characteristics when a set of events leads to a failure versus conditions when the same set of events does not lead to a failure. In other words, the failure manifestation condition has several correlated system parameters indicating non-deterministic symptoms. Hence, further investigations of phrase mining-based failure prediction look promising.

How generic is Desh? Desh has been evaluated on Cray Logs. How generic is it for other system logs? We have thoroughly investigated prior HPC logs such as BlueGene that are comparatively easier for feature classification. Several solutions exist in this context [32, 49]. Hence, prior researched techniques suffice. Several studies [29, 30, 42, 46] have employed LSTM for system anomaly detection in cloud computing systems and service-oriented architectures, making Desh's solution paradigm highly generic. How efficient and computationally inexpensive is the question. That depends on the problem goal and system characteristics. Cray systems contain dense unstructured logging from multiple log sources, which makes log mining non-trivial. Desh can certainly be adapted to other large-scale production systems with a different logging paradigm with some customizations. Our approach in itself is unsupervised and remains unperturbed by the chasms of diverse computing infrastructures.

How much lead time is sufficient? What actions are feasible? Desh reiterates the requirement for sufficient lead times. So how much lead time is good enough? Several proactive recovery mechanisms have been investigated such as job migration, process cloning, lazy checkpointing and similar techniques that are successful in mitigating job failures. Traditional reactive checkpoint/restarts are expensive. Process-level job migrations [41] take 13 to 24 seconds, skip/lazy checkpointing [40], or quarantining nodes [25] by preventing future job scheduling on them are all feasible proactive actions that can be taken in practice, if Desh can indicate impending node failures with lead times longer than these mitigation actions require. Dino [39] proposes node cloning service in 90 seconds. Three minutes lead time suffices for the discussed recovery options. With a reduced false positive rate, even lower lead times (≈1.5 minutes) can be helpful for proactive resilience actions. Further discussion about failure recovery is beyond the scope of this paper.

5 RELATED WORK

Data mining solutions for enhanced system reliability are being investigated both in cloud computing and HPC systems. Understanding performance and resilience trade-offs is important for failure prediction in large-scale systems. We categorically highlight features that are distinct in Desh.

LSTM Applications: Recent works such as [29, 30, 42, 46] have leveraged LSTM for failure prediction. Wang et al. [42] uses LSTM to improve the quality of service in service-oriented architectures. Researchers [30, 46] have utilized binary classification for predicting failures without any lead time discussion. Islam et al. [29] perform failure prediction on cloud computing systems focusing on resource usage and jobs/tasks termination and have obtained 87% accuracy. Their high-level goals are similar to ours. However, Desh formulates failure chains and focuses on predicting lead times from real data. The obtained lead times are promising for taking proactive recovery actions in practice. DeepLog [18] is the closest work to Desh. Apart from the target logs such as Openstack, which are very different from Cray logs, DeepLog's definition and detection of anomaly differs from Desh. They predict a single phrase as an anomaly, while Desh evaluates a chain of events to flag an anomaly.

Failure Prediction: Nie et al. [35] study GPU errors using neural networks. Nodeinfo [36] proposes an unsupervised alert detection system using ideas of information entropy and binary scoring. Bouguerra et al. [6] find that the failure distribution correlates with the false negative distribution and that the temporal correlations of failure needs to be understood. Xie et al. [43] assess the Lustre file system of the Titan Supercomputer to propose a statistical regression approach, which predicts output performance in petascale file systems. Chilimbi et al. [10] propose Adam, a scalable deep learning training system. Bautista-Gomez et al. [5] discuss the spatial and temporal analysis of DRAM memory errors in HPC systems. However, their objectives differ as they do not aim at node failures in computing systems. Hora [38] formulates a Bayesian model for component failure prediction using component dependencies unlike Desh, which is unaware of such system dependencies and solely relies on the data for inference. Gainaru et al. [21] discuss an online failure prediction model using feedback simulation with their toolkit ELSA [22]. In contrast, Desh uses deep learning to predict lead times to failures.

Log Analysis: Event block detection by Baseman et al. [4] focuses on tokenizing unstructured text. Pecchia et al. [37] find that grouping events based on predetermined time thresholds performs badly. Additional consideration of likelihood of entries improves field data analysis. Di Martino [15] uses the MTW (multiple time windows) heuristic to group supercomputing error logs. Prior log analysis techniques have studied various event correlation methods [22], time coalition techniques [16] and log parsing methods [26]. Our work uses word embeddings to vectorize the data. While processing the log input is similar to other unstructured text parsing, Desh exploits the relevant context (word embeddings, discussed in Section 3.1) for node failure prediction. Tiwari et al. [40] discuss lazy checkpointing to reduce overhead and describe temporal characteristics of failures in multiple HPC systems. Failure characterization [24, 25] has been studied to understand the requirements of exascale computing, which provides important statistics for system characterization to help understand the challenges before embarking on log mining-based resilience studies.

Anomaly Detection in Distributed Systems: Log mining-based performance diagnosis on cloud computing systems [44, 45, 47] has garnered considerable attention recently. The solutions do have some similarities with the HPC resilience frameworks (e.g., log parsing, applied ML techniques). One concern with these solutions making them incompatible with HPC systems are fault injections [14, 18, 38, 45] and source code referenced log statement parsing or augmentation [44]. Moreover, logs from distributed systems such as HDFS and OpenStack are at a much higher level in the software stack than HPC architectures. Hence, failure prediction gets complex with diverse log sources compared to per-log level anomaly identification. Data center log analytics based on workflow monitoring [12, 45] and system tracing [9] differ in their log analysis objective (e.g., lack of lead time sensitivity study, retaining component location information), distinguishing them from Desh.

In summary, failure prediction is considerably researched. Our novel contribution is an efficient way of predicting lead times in contemporary HPC systems, which may pave way for further analysis of deep learning-based failure prediction.

6 CONCLUSION

Desh provides a powerful technique to process HPC logs using LSTM for efficient failure prediction. Desh uses a novel three-phase deep learning approach to first train to recognize chains of log events leading to a failure, second re-train chain recognition of events augmented with expected lead times to failure, and third predict lead times during testing/inference deployment to predict which specific node fails in how many minutes. Desh obtains more than 2 minutes average lead times with an F1 score as high as 89.88%. Our lead time sensitivity study and its correlation with diverse failure classes can aid system designers comprehend what is required for faster prediction in the upcoming exascale era. We use actual field data from supercomputing sites without any failure injection for anomalies. Our insights can have implications on real-time approaches required for quick anomaly detection online, on the significance of novel phrase mining paradigms befitting for contemporary large-scale computing systems and on new statistical paradigms to leverage unknown log phrases in system anomaly detection.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for their valuable comments and helpful suggestions. This work was funded in part by subcontracts from Lawrence Berkeley and Sandia National Laboratories, Air Force Office of Scientific Research grant FA9550-12-1-0442, and NSF grants 1217748 and 0958311. Any views expressed in this paper are those of the authors and do not necessarily reflect the views of AMD, Cray, NSF, or any other national labs.

REFERENCES

[1] [n. d.]. CFDR Data. https://www.usenix.org/cfdr-data
[2] [n. d.]. Top 500. https://www.top500.org/lists/2017/11/
[3] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.* 265–283.
[4] Elisabeth Baseman, Sean Blanchard, Zongze Li, and Song Fu. 2016. Relational Synthesis of Text and Numeric Data for Anomaly Detection on Computing System Logs. In *15th IEEE International Conference on Machine Learning and Applications, ICMLA 2016, Anaheim, CA, USA, December 18-20, 2016.* 882–885.

[5] Leonardo Bautista-Gomez, Ferad Zyulkyarov, Osman Unsal, and Simon McIntosh-Smith. 2016. Unprotected computing: a large-scale study of DRAM raw error rate on a supercomputer. In *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for.* IEEE, 645–655.

[6] Mohamed Slim Bouguerra, Ana Gainaru, and Franck Cappello. 2013. Failure prediction: what to do with unpredicted failures. In *28th IEEE international parallel and distributed processing symposium*, Vol. 2.

[7] Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. 1992. Class-based n-gram models of natural language. *Computational linguistics* 18, 4 (1992), 467–479.

[8] MD Catonsville, Bob Adolf, Shekhar Borkar, Nathan DeBardeleben, Mike Heroux, Dave Rogers, Vivek Sarkar, Martin Schulz, Mark Snir, Bob Woodward UMN, et al. 2012. Inter-Agency Workshop on HPC Resilience at Extreme Scale. (2012).

[9] Mike Y. Chen, Emre Kiciman, Eugene Fratkin, Armando Fox, and Eric A. Brewer. 2002. Pinpoint: Problem Determination in Large, Dynamic Internet Services. In *2002 International Conference on Dependable Systems and Networks (DSN 2002), 23-26 June 2002, Bethesda, MD, USA, Proceedings.* IEEE, 595–604.

[10] Trishul M. Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. 2014. Project Adam: Building an Efficient and Scalable Deep Learning Training System. In *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014.* 571–582.

[11] François Chollet et al. 2015. Keras. https://github.com/keras-team/keras.

[12] Zaheer Chothia, John Liagouris, Desislava Dimitrova, and Timothy Roscoe. 2017. Online Reconstruction of Structural Information from Datacenter Logs. In *Proceedings of the Twelfth European Conference on Computer Systems.* ACM, 344–358.

[13] Adam Coates, Brody Huval, Tao Wang, David Wu, Bryan Catanzaro, and Ng Andrew. 2013. Deep learning with COTS HPC systems. In *International Conference on Machine Learning.* 1337–1345.

[14] Daniel Joseph Dean, Hiep Nguyen, and Xiaohui Gu. 2012. Ubl: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems. In *Proceedings of the 9th international conference on Autonomic computing.* ACM, 191–200.

[15] Catello Di Martino. 2013. One size does not fit all: Clustering supercomputer failures using a multiple time window approach. In *International Supercomputing Conference.* Springer, 302–316.

[16] Catello Di Martino, Marcello Cinque, and Domenico Cotroneo. 2012. Assessing time coalescence techniques for the analysis of supercomputer logs. In *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on.* IEEE, 1–12.

[17] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. 2015. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2625–2634.

[18] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security.* ACM, 1285–1298.

[19] Xiaoyu Fu, Rui Ren, Sally A McKee, Jianfeng Zhan, and Ninghui Sun. 2014. Digging deeper into cluster system logs for failure prediction and root cause diagnosis. In *Cluster Computing (CLUSTER), 2014 IEEE International Conference on.* IEEE, 103–112.

[20] Errin W Fulp, Glenn A Fink, and Jereme N Haack. 2008. Predicting Computer System Failures Using Support Vector Machines. *WASL* 8 (2008), 5–5.

[21] Ana Gainaru, Mohamed Slim Bouguerra, Franck Cappello, Marc Snir, and William Kramer. 2014. *Navigating the blue waters: online failure prediction in the petascale era. Argonne National Laboratory Technical Report.* Technical Report. ANL/MCS-P5219-1014.

[22] Ana Gainaru, Franck Cappello, Marc Snir, and William Kramer. 2012. Fault prediction under the microscope: a closer look into HPC systems. In *SC Conference on High Performance Computing Networking, Storage and Analysis, SC '12, Salt Lake City, UT, USA - November 11 - 15, 2012.* 77.

[23] Ana Gainaru, Franck Cappello, Marc Snir, and William Kramer. 2013. Failure prediction for HPC systems and applications Current situation and open issues. *International Journal of High Performance Computing Applications* 27, 3 (2013), 273–282.

[24] Saurabh Gupta, Tirthak Patel, Christian Engelmann, and Devesh Tiwari. 2017. Failures in large scale systems: long-term measurement, analysis, and implications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2017, Denver, CO, USA, November 12 - 17, 2017.* 44:1–44:12.

[25] Saurabh Gupta, Devesh Tiwari, Christopher Jantzi, James H. Rogers, and Don Maxwell. 2015. Understanding and Exploiting Spatial Properties of System Failures on Extreme-Scale HPC Systems. In *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2015, Rio de Janeiro, Brazil, June 22-25, 2015.* 37–44.

[26] Pinjia He, Jieming Zhu, Shilin He, Jian Li, and Michael R Lyu. 2016. An evaluation study on log parsing and its use in log mining. In *Dependable Systems and Networks (DSN), 2016 46th Annual IEEE/IFIP International Conference on.* IEEE, 654–661.

[27] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[28] Kun-Yi Huang, Chung-Hsien Wu, Ming-Hsiang Su, and Hsiang-Chi Fu. 2017. Mood detection from daily conversational speech using denoising autoencoder and LSTM. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on.* IEEE, 5125–5129.

[29] Tariqul Islam and Dakshnamoorthy Manivannan. 2017. Predicting Application Failure in Cloud: A Machine Learning Approach. In *Cognitive Computing (ICCC), 2017 IEEE International Conference on.* IEEE, 24–31.

[30] XU Jianwu, Ke Zhang, Hui Zhang, Renqiang Min, and Guofei Jiang. 2017. System failure prediction using long short-term memory neural networks. US Patent App. 15/478,714.

[31] Rie Johnson and Tong Zhang. 2016. Supervised and semi-supervised text categorization using LSTM for region embeddings. In *International Conference on Machine Learning.* 526–534.

[32] Zhiling Lan, Jiexing Gu, Ziming Zheng, Rajeev Thakur, and Susan Coghlan. 2010. A study of dynamic meta-learning for failure prediction in large-scale systems. *J. Parallel and Distrib. Comput.* 70, 6 (2010), 630–643.

[33] Zhiling Lan, Ziming Zheng, and Yawei Li. 2010. Toward automated anomaly identification in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems* 21, 2 (2010), 174–187.

[34] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[35] Bin Nie, Ji Xue, Saurabh Gupta, Christian Engelmann, Evgenia Smirni, and Devesh Tiwari. 2017. Characterizing Temperature, Power, and Soft-Error Behaviors in Data Center Systems: Insights, Challenges, and Opportunities. In *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), 2017 IEEE 25th International Symposium on.* IEEE, 22–31.

[36] Adam J Oliner, Alex Aiken, and Jon Stearley. 2008. Alert detection in system logs. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on.* IEEE, 959–964.

[37] Antonio Pecchia, Domenico Cotroneo, Zbigniew Kalbarczyk, and Ravishankar K Iyer. 2011. Improving log-based field failure data analysis of multi-node computing systems. In *Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on.* IEEE, 97–108.

[38] Teerat Pitakrat, Dušan Okanović, André van Hoorn, and Lars Grunske. 2017. Hora: Architecture-aware online failure prediction. *Journal of Systems and Software* (2017).

[39] Arash Rezaei and Frank Mueller. 2015. DINO: Divergent Node Cloning for Sustained Redundancy in HPC. In *Cluster Computing (CLUSTER), 2015 IEEE International Conference on.* IEEE, 180–183.

[40] Devesh Tiwari, Saurabh Gupta, and Sudharshan S Vazhkudai. 2014. Lazy checkpointing: Exploiting temporal locality in failures to mitigate checkpointing overheads on extreme-scale systems. In *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on.* IEEE, 25–36.

[41] Chao Wang, Frank Mueller, Christian Engelmann, and Stephen L Scott. 2008. Proactive process-level live migration in HPC environments. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing.* IEEE Press, 43.

[42] Hongbing Wang, Zhengping Yang, and Qi Yu. 2017. Online Reliability Prediction via Long Short Term Memory for Service-Oriented Systems. In *Web Services (ICWS), 2017 IEEE International Conference on.* IEEE, 81–88.

[43] Bing Xie, Yezhou Huang, Jefrey S Chase, Jong Youl Choi, Scott Klasky, Jay Lofstead, and Sarp Oral. 2017. Predicting Output Performance of a Petascale Supercomputer. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing.* ACM, 181–192.

[44] Wei Xu, Ling Huang, Armando Fox, David A Patterson, and Michael I Jordan. 2010. Detecting large-scale system problems by mining console logs. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10).* Citeseer, 37–46.

[45] Xiao Yu, Pallavi Joshi, Jianwu Xu, Guoliang Jin, Hui Zhang, and Guofei Jiang. 2016. Cloudseer: Workflow monitoring of cloud infrastructures via interleaved logs. *ACM SIGOPS Operating Systems Review* 50, 2 (2016), 489–502.

[46] Ke Zhang, Jianwu Xu, Martin Renqiang Min, Guofei Jiang, Konstantinos Pelechrinis, and Hui Zhang. 2016. Automated IT system failure prediction: A deep learning approach. In *2016 IEEE International Conference on Big Data, BigData 2016, Washington DC, USA, December 5-8, 2016.* 1291–1300.

[47] Xu Zhao, Kirk Rodrigues, Yu Luo, Ding Yuan, and Michael Stumm. 2016. Non-Intrusive Performance Profiling for Entire Software Stacks Based on the Flow Reconstruction Principle. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.* 603–618.

[48] Ziming Zheng, Li Yu, Zhiling Lan, and Terry Jones. 2012. 3-Dimensional root cause diagnosis via co-analysis. In *9th International Conference on Autonomic Computing, ICAC'12, San Jose, CA, USA, September 16 - 20, 2012.* 181–190.

[49] Ziming Zheng, Li Yu, Wei Tang, Zhiling Lan, Rinku Gupta, Narayan Desai, Susan Coghlan, and Daniel Buettner. 2011. Co-analysis of RAS log and job log on Blue Gene/P. In *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International.* IEEE, 840–851.