



King's Research Portal

Document Version
Peer reviewed version

[Link to publication record in King's Research Portal](#)

Citation for published version (APA):

Alharthi, K. A., Jhumka, A., Di, S., Gui, L., Cappello, F., & McIntosh-Smith, S. (2023). Time Machine: Generative Real-Time Model For Failure (and Lead Time) Prediction in HPC Systems. In *The 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks* IEEE Press.

Citing this paper

Please note that where the full-text provided on King's Research Portal is the Author Accepted Manuscript or Post-Print version this may differ from the final Published version. If citing, it is advised that you check and use the publisher's definitive version for pagination, volume/issue, and date of publication details. And where the final published version is provided on the Research Portal, if citing you are again advised to check the publisher's website for any subsequent corrections.

General rights

Copyright and moral rights for the publications made accessible in the Research Portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognize and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the Research Portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the Research Portal

Take down policy

If you believe that this document breaches copyright please contact librarypure@kcl.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Time Machine: Generative Real-Time Model For Failure (and Lead Time) Prediction in HPC Systems

Khalid Ayed Alharthi^{1,5,6}, Arshad Jhumka¹, Sheng Di², Lin Gui³, Franck Cappello^{2,7}, Simon McIntosh-Smith⁴

¹University of Warwick, UK; ²Argonne National Laboratory - The University of Chicago, USA; ³ King’s College London;

⁴ Bristol University, UK; ⁵ University of Bisha, KSA; ⁶ The Alan Turing Institute, UK;

⁷ University of Illinois at Urbana-Champaign, USA

Emails: {khalid.alharthi, h.a.Jhumka}@warwick.ac.uk, sdi@anl.gov, cappello@mcs.anl.gov, lin.l.gui@kcl.ac.uk

Abstract—High Performance Computing (HPC) systems generate a large amount of unstructured/alphanumeric log messages that capture the health state of their components. Due to their design complexity, HPC systems often undergo failures that halt applications (e.g., weather prediction, aerodynamics simulation) execution. However, existing failure prediction methods, which typically seek to extract some information theoretic features, fail to scale both in terms of accuracy and prediction speed, limiting their adoption in real-time production systems. In this paper, differently from existing work and inspired by current transformer-based neural networks which have revolutionized the sequential learning in the natural language processing (NLP) tasks, we propose a *novel* scalable log-based, self-supervised model (i.e., no need for manual labels), called *Time Machine*[†], that predicts (i) forthcoming log events (ii) the upcoming failure and its location and (iii) the expected lead time to failure. *Time Machine* is designed by combining two stacks of *transformer-decoders*, each employing the *self-attention mechanism*. The first stack addresses the failure location by predicting the sequence of log events and then identifying if a failure event is part of that sequence. The lead time to predicted failure is addressed by the second stack. We evaluate *Time Machine* on four real-world HPC log datasets and compare it against three state-of-the-art failure prediction approaches. Results show that *Time Machine* significantly outperforms the related works on Bleu, Rouge, MCC, and F1-score in predicting forthcoming events, failure location, failure lead-time, with higher prediction speed.

I. INTRODUCTION

Large-scale HPC systems, such as supercomputers, execute resource-hungry applications such as weather forecasting, flow dynamics simulations among many others. These systems, consisting of sophisticated hardware (HW) and software (SW), often fail due to their scale and their design complexity. The SW components, such as OS and parallel file systems, typically generate log messages that capture the health of various components in the system, such as network and memory, and these log messages are recorded in a central repository [1], which we call a log file. As a result, system administrators find it a very useful source of information when trying to predict system failure because it contains rich information about normal behavior (i.e., informational messages) or abnormal behavior (i.e., error messages) of various system components. As such, failure log analysis of HPC systems is attracting more

and more researchers from academia and industry in order to improve the reliability of such systems.

When errors in the HPC system are not suitably handled, which can occur at specific components (e.g., nodes), then a failure of the system, i.e., more specifically, one or more affected components, may occur. A failure is a special event in the system and results in a special log (e.g., lockup log) to be recorded in the log file. The impact of such failures may be enormous on applications: drastic computational overhead could be introduced, such as through (partial) re-execution, thereby having severe impact on application execution. In an era of exascale computing (i.e., HPC systems executing 10^{18} floating point operations per second), failures are predicted to occur more frequently, exacerbating associated overhead.

To mitigate the impact of failure, efficient failure management strategies are required. Specifically, the development of accurate failure prediction is becoming important, to support the timely deployment of proactive recovery management techniques such as checkpointing/restart or job migration [2], [3]. Unfortunately, the effectiveness of failure prediction tools is still insufficient, thereby necessitating the development of online failure prediction techniques to flag impending failures and their lead-times with high prediction accuracy and speed and with lesser computational overhead. Some works on failure prediction exists, e.g., [4], [5].

In this paper, we address this important problem of failure prediction by developing and applying a *transformer-decoder* model on HPC log data to build a *generative self-supervised model*, which we call **Time Machine**, to predict two important failure parameters: (i) the failure location, i.e., which nodes will crash, and (ii) the lead time to failure, i.e., how long is left before the predicted failure happens. Our designed *Time Machine* works as follows: (i) for the location problem, it first predicts the future sequence of logs (future health state) and then identifying if a failure event is part of the predicted sequence and (ii) for the lead time to failure problem, *Time Machine* reduces the time prediction problem (which is a regression problem) into a self-annotated multi-class classification problem, by predicting the class for the failure lead time. We discuss the motivation of modelling the failure lead time problem in our methodology, section IV-B1. Note that our work introduces a novel method to construct (no need for manual labels) and augment a self-time annotated training dataset on sequential time-based (timestamps) raw data via an

[†]A **Time Machine** allows us to travel into the future to observe the health state of HPC system and report back. Here, we travel into the log extension to report an upcoming failure.

automatic accumulative and iterative process.

The use of generative models for failure prediction is very challenging: (i) erroneous states and failures are rare(r) events, (ii) logs are often incomplete, duplicate and (iii) messages are alphanumeric in nature and generally lack a proper structure [6], which is very different from the context of, say, text prediction application [7]. Nonetheless, there are many state-of-the-art RNN-based failure prediction methods such as Long Short-term Memory (LSTM) [8] (Desh), Bidirectional Long Short Term Memory (Bi-LSTM) [9], and Gated Recurrent Unit (GRU) [10], which however suffer from non-trivial weaknesses: (i) long training time because of the absence of parallelization in recurrence learning, and (ii) the vanishing gradient problem with loss of earlier “memory”, which may cause limited accuracy.

To address the above limitations, we propose using transformer-decoder based model (called *Time Machine*), which can improve failure prediction accuracy and speed regarding both locations and lead times, through (i) self-attention mechanism and (ii) parallelization which are the crux of transformer neural networks. The reason why transformer-decoder model may work well on failure prediction is that it has the ability to capture long-term dependencies in sequential data (i.e., text) and understand the relationship/order between the words in the sequence (i.e., sentence, paragraph, etc.) via attention mechanism and parallelization. Similarly, the health state for HPC systems can be represented by the sequence of log events (i.e., sequential) generated by their components (e.g., compute nodes) based on timestamps associated with each log event. These consecutive log events are usually related to each other and generated one after another like words in the text corpus. For example, failure events usually occur after a sequence of preceding events, including errors/faults that cause those failures. As such, if we consider a log event as a word and a log sequence as a text sequence (e.g., a sentence), we can explore the auto-correlation of event sequences by the transformer-decoder model, which in turn can perform the prediction work.

Additional failure prediction models are developed for HPC systems, however these solutions are mainly based on supervised-learning, requiring extensive data labelling such as [11], [12], [13]. Most unsupervised & self-supervised solutions do not address the problem of predicting the lead-time to failure, such as Clairvoyant [14]. Specifically, Clairvoyant used one stack transformer-decoder to predict failures only. To enable the prediction of failure lead time, there are several key innovative designs proposed in our solution. Our Time Machine framework adopts a two-stack transformer-decoder architecture to predict not only failures but their lead times. The adaptation of the transformer-decoder to predict the failure lead times is based on a novel approach to self-attention: Specifically, the Time Machine framework demonstrates how the self-attention mechanism developed for text prediction is used to predict the failure lead times, by encoding/decoding log events to map each log event onto its timestamp step during the training and prediction phases. This the first work to over-

come these limitations by formulating the time prediction as a self-annotated multi-class classification problem by predicting the class for the failure lead time. Moreover, the Time Machine can construct training instances in real-time because of our novel synthetic minority oversampling design.

We evaluate **Time Machine** [15] on four real-world HPC logs and we compare it against LSTM [8] (Desh), Bi-LSTM [9], and GRU [10]. Results show that Time Machine significantly outperforms the best of them: (I) **Log events prediction**: Time Machine obtains a Bleu and Rouge score of up to 0.79 and 0.77 respectively whereas best of the three techniques only has 0.47 and 0.34. (II) **Failure Location**: Time Machine obtains a MCC and F1-score of up to 0.80 and 0.87, respectively, while the best one of the three techniques only has 0.53 and 0.71 respectively. (III) **Failure lead time**: Time Machine is also the best in class, with MCC and F1-score of up to 0.87 and 0.95, respectively. (IV) **Speed-up of training and prediction**: Time Machine is significantly faster than other approaches in both training (5.4~9.4× speed-up on average) and chain prediction (over 15× faster than the related works), making Time Machine very suitable for online failure prediction in real-time production HPC systems.

II. SYSTEM MODEL AND FAULT MODEL

In this section, we present the system model, fault model, and HPC system component (e.g., node) failure.

A. System Model

We describe the general HPC system model targeted by our research as follows: In the HPC system, there is a set of compute nodes $C = \{C_1, \dots, C_m\}$ provided to execute user-submitted jobs $J = \{J_1, \dots, J_n\}$ (e.g., weather forecasting). We assume the compute nodes to be co-located geographically such that message delays between nodes and a central log repository is bounded. Also, we assume nodes to be time-synchronised, through the use of clock synchronisation, which is important for event timestamping. A job scheduler is used to assign the jobs to different production time-slots $T = \{T_1, \dots, T_p\}$ on specific nodes. As the system operates, a bunch of log messages are generated to capture the health of the system and collected on a central log server or file [14].

B. Fault Model

A fault model specifies the way a system is expected to be affected by faults. Our designed failure prediction framework can be applied on various types of discrete faults at different levels, such as hardware, system, application level, file system, and at an aggregate supercomputer level. As a fault occurs, the resulting errors will be manifested as error messages in the system log file. Overlooking the error messages will likely result in a system/application failure, which will also be logged. For simplicity of description, we consider node failures [7] without loss of generality, and focus on the prediction of node failure events as well as their failure lead-times through our proposed methods (called *Time Machine*), which can also be applied to failures of other components (such as switches, GPUs).

C. HPC System Node Failure

HPC node failure is a state in which the operating system kernel hang-up, becomes unresponsive, goes stuck, or loops without ends, blocking other processes from executing and ultimately causing the nodes to shutdown. In HPC systems, there could be many factors or different types of preceding errors (from hardware errors to software/application faults) that can result in node failures. The preceding errors that cause node failures are very diverse, including hardware issue (memory, GPU, network), OS process errors, file system errors, application errors, etc [14].

The consequence of these preceding errors may differ a lot. Some errors may induce failures very quickly because of their fast propagation: i.e., the sequence of log events between the first error message and the ending failure event could be very short. On the other hand, some other errors may take a long time before their corresponding failure occurs, corresponding to a lengthy sequence of log events with a relatively high delay between the first error message and the ending failure event. This could be due to, e.g., the system attempting a recovery that eliminates the error (e.g., message loss + re-transmission). However, the error may come back later and if recovery is unsuccessful despite many attempts, failure may ensue, with a long lead time. It is also worth noting that there could be many interleaved & irrelevant events recorded between node failures and their preceding error events, for both short and long sequences, making failure prediction more challenging.

III. PROBLEM FORMULATION

We formulate the research problem as below: Given a log dataset with a sequence of events, our objective is to predict the upcoming sequence of log events and determine if this sequence may contain a failure event; if yes, then predict the lead-time of the failure event as well.

Research challenges: There are two important attributes of failure prediction: (i) **Location:** the component (or node) that would fail/crash should be accurately predicted so that the failure recovery mechanism can be launched at proper “location” and (ii) **Failure lead time:** the time at which the failure log event is predicted to occur should be similar/accurate compared to the one in real-time, otherwise, the failure recovery mechanism would be triggered at wrong time.

We denote the set of log sequences by \mathcal{L}^r , where its length is at most r . Suppose we are given two sets: \mathcal{L}^m and \mathcal{L}^k ($k \leq m$), where the elements in the set \mathcal{L}^k are possible extensions of the elements in \mathcal{L}^m . That is, each element in \mathcal{L}^m can be assigned an element from \mathcal{L}^k as an output. Accordingly, for each $\epsilon_i \in \mathcal{L}^m$, $\epsilon'_j \in \mathcal{L}^k$ indicates the true prediction outcome (i.e., the real sequence of log events following ϵ_i). Our failure prediction research is to model a mapping $\mathcal{M} : \mathcal{L}^m \rightarrow \mathcal{L}^k$, in which $\mathcal{M}(\epsilon_i)$ is the predicted sequence which follows ϵ_i , i.e., $\epsilon_i \cdot \mathcal{M}(\epsilon_i)$ is a predicted upcoming log sequence of length $(k + m)$, i.e., $\epsilon_i \cdot \mathcal{M}(\epsilon_i) \in \mathcal{L}^{m+k}$.

We formulate the two problems as follows:

Definition 1 (Log Events Prediction). *For a sequence of log events $\epsilon_i \in \mathcal{L}^m$, a predictor \mathcal{M} is expected to be with the minimal distance for the log sequences of length $(m+k)$, i.e., $\arg \min_{\mathcal{M}} \mathcal{D}(\epsilon_i \cdot \mathcal{M}(\epsilon_i), \epsilon_i \cdot \epsilon'_j)$, where \cdot indicates ‘sequence concatenation’ and $\mathcal{D} : \mathcal{L}^{m+k} \times \mathcal{L}^{m+k} \rightarrow \mathbb{R}$ is the distance measure. \mathcal{D} is a distance metric on two log sequences; $\mathcal{D}=0$ means two logs are identical to each other. If the distance is 0, we claim “ \mathcal{M} correctly extends ϵ_i ”; or else, we say “ \mathcal{M} approximately extends ϵ_i ”.*

Definition 2 (Failure Prediction). *For a predictor \mathcal{M} on a log sequence $\epsilon_i \in \mathcal{L}^m$ with an extension $\epsilon'_j \in \mathcal{L}^k$, we say “ \mathcal{M} accurately solves the failure prediction” iff $\mathcal{F} \in \epsilon'_j \Leftrightarrow \mathcal{F} \in \mathcal{M}(\epsilon_i)$. We say “ \mathcal{M} approximately solves the failure prediction problem” if $\mathcal{F} \in \mathcal{M}(\epsilon_i) \Rightarrow \mathcal{F} \in \epsilon'_j$.*

When $\mathcal{F} \in \epsilon'_j$, we say that the extension ϵ'_j is a *failure extension* of ϵ_i and when $\mathcal{F} \in \mathcal{M}(\epsilon_i)$, we say that $\mathcal{M}(\epsilon_i)$ is a *predicted failure extension* of ϵ_i . We also say that ϵ_i is a *failure precursor sequence*. Note that $\mathcal{D}(\epsilon_i \cdot \mathcal{M}(\epsilon_i), \epsilon_i \cdot \epsilon'_j)=0$ means that the predicted lead time of failure event is accurate perfectly, i.e., the failure event would occur right at the predicted moment. Also note that a small value of \mathcal{D} indicates that the failure event occurrence moment is approximately correct in the sequence.

For the failure lead time [16], due to the non-determinism at the system level, it is difficult to accurately predict the *exact* failure lead time. To circumvent this challenge, we propose to model the failure lead time prediction as a multi-class classification problem. We propose a general formal definition of failure lead time as follows.

Definition 3 (Lead time to Failure). *Given a log sequence $\epsilon_i \in \mathcal{L}^m$, its extension $\epsilon'_j \in \mathcal{L}^k$ which is a failure extension of ϵ_i , the failure lead time of ϵ_i is the difference between the timestamp of the last event in $\epsilon_i \in \mathcal{L}^m$ and the failure event in ϵ'_j and is equal to $TS(\mathcal{F} \in \epsilon'_j) - TS(\text{last}(\epsilon_i))$, where TS denotes the timestamp function and last function returns the last element of a sequence respectively.*

Let \mathcal{L}^m be the instance space. Every point $\epsilon_i \in \mathcal{L}^m$ is a potential state of the log. Given a pair $\langle \epsilon_i, F(\epsilon_i) \rangle$, where $\epsilon_i \in \mathcal{L}^m$ is a failure precursor sequence, ϵ'_j is a failure extension of ϵ_i and $F(\epsilon_i)$ denotes the failure lead time of ϵ_i , we wish to learn an approximation of the unknown F , denoted by \hat{F} and $\hat{F}(\epsilon_i) = TS(\mathcal{F} \in \mathcal{M}(\epsilon_i)) - TS(\text{last}(\epsilon_i))$, where \mathcal{M} is a predictor that solves the failure prediction problem.

Definition 4 (Predicted Lead time to Failure). *Given a failure precursor log sequence $\epsilon_i \in \mathcal{L}^m$, its failure extension $\epsilon'_j \in \mathcal{L}^k$, a set of non-overlapping p ranges $R = \{R_1, \dots, R_p\}$, $R_i \in \mathbb{Z}^+ \times \mathbb{Z}^+$, $R_i \cap R_j = \emptyset$ and a predictor \mathcal{M} which solves the failure prediction issue approximately, we say that the predicted failure lead time is correct for ϵ_i if $\exists R_i \in R. \hat{F}(\epsilon_i) \in R \Rightarrow F(\epsilon_i) \in R$.*

IV. TIME MACHINE METHODOLOGY

Inspired by recent work in (NLP) tasks, we propose a transformer-decoder based sequential model to predict the

forthcoming events, node failure, and failure lead-time in HPC systems. In general, we take the self-attention based language model as an estimator for the posterior probabilities, in which we consider the log events as input words, a sequence of log events as a sentence, and the probabilities of failure in HPC as a context-based generative probabilities. Furthermore, self-attention is friendly to *parallelization*, such that the training and prediction time can be significantly reduced by leveraging parallel techniques, compared to existing state-of-the-art failure prediction methods, such as RNN-based methods used in [8], [9], [10]. To this end, we develop a novel real-time online approach namely **Time Machine** which is fully self-supervised without the need for labeling by HPC system administrators. As shown in Figure 1, the architecture of our model consists of **two** transformer-decoder neural network components, and both of the two decoders are based on the transformer-decoder variant [17]. The **first** transformer component aims to predict HPC node failures with two major steps: (1) for each node, it predicts the sequence of future events (or future health state); (2) it determines whether a failure is included in the predicted sequence. The **second** transformer component aims to predict *lead times*, based on which one or more proactive fault-tolerant techniques can be accurately selected in time ahead of the failure occurrence.

As for the Time Machine methodology which adopts two stacks of transformer-decoder to predict HPC system failures and their lead-times, its framework design also includes three key innovative points:

- Transformer neural networks are employed originally for NLP tasks such as text classification, text generation, summarisation, while the **Time Machine** method is the first work utilizing transformer architecture to predict the lead-time to failures. This method can be generalized to other domains for time-based prediction tasks.
- Time Machine introduces a novel synthetic minority oversampling technique for online time-based tasks to construct the training instances from failure sequences.
- In the fault tolerance research, our Time Machine method is the first study to reduce/convert the time prediction problem (a regression problem) into a self-annotated multi-class classification problem, by predicting the class for the failure lead time.

We detail our proposed framework in the rest of this section.

A. Node Failure Prediction

1) **Phase I. Log Event Prepossessing**: In the first phase, similar to the tokenization in NLP task, we first convert the log message into an ID based log event sequence: e_1, e_2, \dots, e_m , where m is the length of event sequence, $e_i \in \{t_j | j = 1, 2, \dots, T\}$ represents the i -th event, and t_j stands for all possible event types in the log event prediction. Besides, we let $m < 1024$ in order to make it possible that all input event sequences share similar length for parallel processing, which is distinct from the existing RNN methods. Moreover, the Byte Pair Encoding (BPE) method is utilized to tokenize the input to encode any unusual tokens (IDs of log entries). BPE is a

compression NLP technique that repeatedly replaces the most common pair of adjacent bytes with a new byte that does not exist in the original data [18].

2) **Phase II. Log Events Learning and Failure Prediction**: Our proposed approach aims to take the self-attention mechanism to improve the connectivity among the events in log sequences. Accordingly, we utilise transformer-decoder architecture, a stack of decoder blocks preceded by an input layer to embed the real-time log events sequence logged by the HPC system component node, and then followed by linear and softmax layers to predict failures (e.g., node crashes, networks failures) by two steps: predicting the future sequence of events and then identifying if a failure is part of the predicted sequence. More details are described in the following text. We refer the readers to read [17] for detailed background of the *transformer variant* which we will use to build our model.

We summarize the current masked language (failure and lead-time prediction in our case) model as follows:

In a typical transformer block ℓ , assuming the input feature for token e_i in $\ell-1$ -th layer of transformer is $e_i^{\ell-1}$, the information propagation process is given by:

$$\begin{aligned} v_i^{\ell-1} &= \text{Self-Attention}(e_i^{\ell-1} | e_1^{\ell-1}, e_2^{\ell-1}, \dots, e_m^{\ell-1}) & (1) \\ \Phi(v_i^{\ell-1}) &= \varphi(W^{\ell} v_i^{\ell-1} + b^{\ell}) & (2) \\ e_i^{\ell} &= \text{LayerNorm}(\Phi(v_i^{\ell-1}) + e_i^{\ell-1}) & (3) \end{aligned}$$

where e_i^{ℓ} is the learned feature for e_i in ℓ -th layer, v_i^{ℓ} is the corresponding value vector in the regard of the self-attention mechanism according to e_i^{ℓ} , φ is an element-wise nonlinear function applied to a feed-forward layer, whose weight matrix, $W^{\ell} \in \mathbb{R}^{n_{\ell} \times n_{\ell-1}}$, transforms the feature dimension from $n_{\ell-1}$ to n_{ℓ} , $\text{Self-Attention}(e^{\ell-1})$ returns the weighted value vector of all input representations where weights are derived by multiplying the query vector of the current input $e^{\ell-1}$ with the key vectors from other inputs. Between every two transformer blocks, there is a skip-connection and a layer normalisation. The former mechanism bypasses the transformer block ℓ and adds the input $e^{\ell-1}$ directly to the output v^{ℓ} of this block, while the latter normalises the input across the feature dimension.

STEP 1: Transfer Learning Based Sequence Prediction

The main idea in pre-trained language model, such as GPT-2 [17], aims to predict a particular word based on its context by: $P(e_i | e_1, e_2, \dots, e_{i-1}, e_{i+1}, \dots, e_m)$. However, in the prediction of HPC log events, the tokens that follow the expected prediction e_i are unseen by the model. Furthermore, the vocabulary used to represent log event types is much smaller than that used in typical NLP tasks, which may lead to overfitting if we simply train an over-parameterised model.

Therefore, we propose using a pre-trained NLP model (GPT-2) to initialise the transformer-decoder model, fine-tuning it on the HPC dataset to predict future log events. Here, we define the probability of future log event by Softmax $P(e'_{m+i} | e_1, e_2, \dots, e_m) = \text{Softmax}(\text{FNN}(e_i^{\ell} | e_1, e_2, \dots, e_m))$, where the FNN stands for the feedforward neural network which is a linear layer to transform the last decoder outputs L into higher-dimensional logits vectors using a linear transformation and non-linear activation function. Softmax layer is

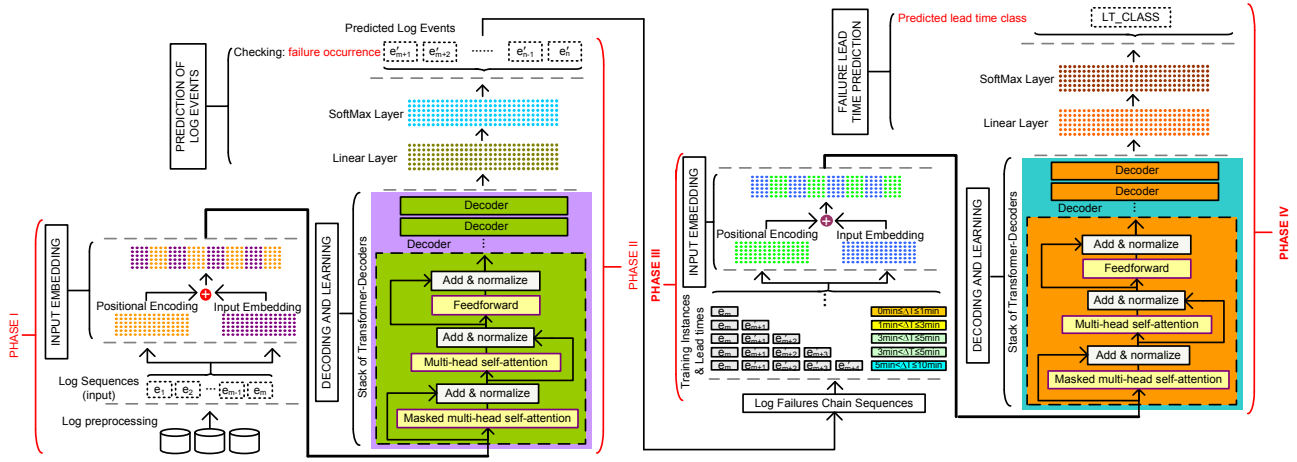


Fig. 1: Illustration of Health State/Failure/Lead Time Prediction Phases

the final layer applying softmax function to the linear layer's output and converts it to a probability distribution. According to definition 1, we use the cross-entropy as the metric to measure the distance between distribution in loss function,

$$\mathcal{L} = \sum_{i=1}^{n-m} P(e'_{m+i}) \cdot \log(P(\hat{e}'_{m+i}|e_1, e_2, \dots, e_m)) \quad (4)$$

where the estimated probabilities of $P(\hat{e}'_{m+i})$ are defined by the Softmax function with the learned vectors along to the last FNN layer, and the $P(e'_{m+i})$ is the true output from training corpus, which is an advantage that, in such learning architecture, we do not require a specific annotation for self-supervised learning. The sequence of log events can be generated from large scale of raw data automatically. In this way, the pre-trained language model can be easily adapted to the log events prediction task in real-time.

STEP 2: Failure Prediction: Based on the prediction of log events, we can generate a log events sequence by a given $\{e_1, e_2, \dots, e_m\}$, marked as: $\{e_1, e_2, \dots, e_m, e'_{m+1}, \dots, e'_n\}$, where $\{e_i | i \leq m\}$ is the given event and $\{e'_j | m+1 \leq j \leq n\}$ is the predicted event. According to definition 2, the failure prediction aims to identify if the e'_j is the failure extension of e_i . Hence, we can convert the generated event e'_j to the unique ID to check if it is the failure. Here, we suppose the whole vocabulary of log events is \mathcal{V} , which contains two subsets, the failure events \mathcal{V}^f , and the normal events \mathcal{V}^n , where $\mathcal{V}^f \cup \mathcal{V}^n = \mathcal{V}$ and $\mathcal{V}^f \cap \mathcal{V}^n = \emptyset$. Then, one can easily quantify that a predicted log event is a failure if it is a member of failure events set ($e'_j \in \mathcal{V}^f$).

B. Predicting Lead Times to The Node Failure

One key novelty that is significantly different from existing transformer based sequence models, is predicting lead times for the failure events such that appropriate proactive methods could be triggered in time, which is handled by the Phase III and Phase IV.

1) **Phase III: Failure Sequences Construction for Lead-Time Prediction:** In order to predict the node failures' lead times, the first and foremost step is establishing and preparing a dataset based on failure chains (i.e., the Phase III as

presented in Figure 1). Our framework can be easily deployed for HPC systems in real-time, because the training/testing datasets from the failure chains and associated labels (i.e., lead times) are created automatically (**no need for manual labelling**) based on log events' timestamps.

We introduces a **novel synthetic minority oversampling technique** for online time-based tasks to construct the training instances in the real-time from failure sequences as follows. In our model, predicting a node's failures ahead is achieved through accurately predicting the forthcoming log events $\{e'_{m+1}, e'_{m+2}, \dots, e'_n\}$. Without loss of generality, we assume that the predicted log events sequence ends with a failure event since the motivation of our proposed method aims to predict the failure. Hence, based on the given log events sequence $\{e_1, e_2, \dots, e_m\}$ and the proposed events/failure prediction methods, we have a failure chain of $\{e'_{m+1}, e'_{m+2}, \dots, e'_n\}$, where $\exists e'_j \in \mathcal{V}^f$.

To predict the lead time for any concrete failure chain, we then use the timestamp $TS(\cdot)$ to estimate the lead time when the failure appears for a given sequence of log events. Intuitively, we only need to predict the $TS(e'_j)$, where $\{e'_j \in \mathcal{V}^f\}$. However, considering that the size of \mathcal{V}^f is limited, it is essential to design a smoothing method to overcome the potential risk caused by over-fitting. Hence, we propose to utilise the transformer-decoder based method to approximate the $TS(\cdot)$ for both $e'_j \in \mathcal{V}^n$ and \mathcal{V}^f , and take advantage of sequential model to guarantee the latent pattern $TS(e'_i) < TS(e'_j) (i < j)$ is true, to achieve both reasonable and stable lead-time prediction. Specifically, to make the trade-off between efficiency and accuracy, we convert the prediction of lead times from a regression problem to a multi-class classification problem which predicts the class for the failure lead time. Such a design is motivated by the fact that there are only a few proactive recovery techniques used in practice (e.g., less than 10 techniques), and each technique requires a specific lead time to launch. Moreover, the correction/proactive actions generally require approximately estimated lead times instead of the exact lead times. Accordingly, we define 6 lead-time classes \hat{y}_j in our study: $\hat{y}_j \in \{[0\text{min}, 1\text{min}], (1\text{min}, 3\text{min}],$

$(3\text{min}, 5\text{min}]$, $(5\text{min}, 10\text{min}]$, $(10\text{min}, 15\text{min}]$, $(15\text{min}, \infty)$. Our model is flexible in increasing/decreasing lead time classes based on the system recovery mechanism.

We use an example to describe how to construct the training instances. Without loss of generality, suppose a failure chain contains 6 log events including the failure event with associated timestamps: $((e_m, 01:00:00)$, $(e'_{m+1}, 01:00:30)$, $(e'_{m+2}, 01:01:00)$, $(e'_{m+3}, 01:03:10)$, $(e'_{m+4}, 01:04:55)$, $(e'_{m+5}, 01:07:16))$, where the $e'_{m+5} \in \mathcal{V}^f$ is the failure event. The training instances and their associated lead time classes are constructed automatically in terms of the failure chain iteratively, as illustrated in Figure 2.

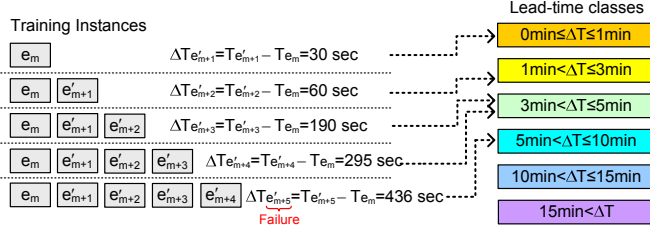


Fig. 2: Lead Time Training Instances Construction

As all log sequences training instances are created from the failure chains as described above, the lead-times have been associated/mapped to the corresponding lead-time classes. Note that this process is conducted during runtime model deployment, and all log event instances are assigned the same tokens as discussed in IV-A2 before being embedded/fed into the second transformer-decoder stack.

2) Phase IV: Lead-Time Learning and Prediction:

Based on the prediction of log events, we can generate a log events sequence by a given $\{e_1, e_2, \dots, e_m\}$, marked as: $\{e_1, e_2, \dots, e_m, e'_{m+1}, \dots, e'_n\}$, where e_i is the given event and e'_j is the predicted event. According to sequence generation in section IV-B1, the lead-time prediction aims to identify the label of y_j for the failure extension of e_i with length j , a.k.a $\{e_1, e_2, \dots, e_m, e_1, \dots, e_j\}$. Hence, we convert this task into sequence classification, in which we employ the fine-tuned transformer-decoder-based model to extract the last representation R_{e_j} of the token e_j , to approximate the posterior probability according to the failure label in real-world datasets, by a Softmax probability:

$$P(y_j) = \text{Softmax}(\text{FNN}(\text{decoder}(R[e'_j] | e_1, \dots, e_m, e'_{m+1}, \dots, e'_j))) \quad (5)$$

In general, any loss function or pre-trained language model can be deployed for the approximation. Without loss of generality, we choose the cross-entropy as loss function and the GPT-2 as decoder in our implementation. In summary, \forall log events sequence e_1, e_2, \dots, e_m , we can predict the failure extension of e'_{m+1}, \dots, e'_n by minimising the loss function defined by equation.4. According to equation.5, we can then predict the lead time to failure of $TS(e'_j) - TS(e_m)$. This proposed framework requires no annotation or supervised signal but facilitates optimising the process which can select the lowest computation cost correction/recovery mechanisms

to correct and fix HPC system errors before the failures occur.

C. Featuring Real-Time in Time Machine

Deploying the *Time Machine* online in real-time requires fine-tuning the model parameters in case new log sequences and failure patterns are encountered. Thus, the teacher forcing technique [19] is proposed to complement *Time Machine* in real-time. The integration between the *Time Machine* and teacher forcing approach enables online training, learning, and prediction by using ground truth input (i.e., the real log events generated by the HPC system in real-time) instead of our model output (the log events predicted by *Time Machine*) from a previous time step as an input. This integration can cope with any new types of log sequences and emerging failure patterns because of various cases, such as upgrade of the HPC system components (i.e., software, hardware, services). The new jobs (e.g., applications) running on HPC systems can also induce new log patterns that have not been met before. Moreover, the teacher-forcing technique forces the real-time log event learning/prediction under the *Time Machine* to be conducted on correct log events (i.e., the correct log sequences generated by the system) rather than log sequences predicted ahead by *Time Machine* (which may be incorrect prediction).

End-to-end use: Once the model is trained, it is deployed onto a supercomputer to predict its failures in real-time. As the jobs are launched on the allocated compute nodes, log messages will be generated on system components and mapped to associated log IDs by our log parser. Then, our model simultaneously begins to predict a sequence of log events for each component (e.g., compute node) using the first stack of transformer-decoder. The predicted log sequence will be updated as a new event is generated on each component. If the predicted sequence contains a failure event, our model immediately calculates its predicted lead-time class to the actual occurrence based on the log sequence, via the second stack of transformer-decoder in our framework. This can inform a failure handling mechanism, e.g., checkpointing. Depending on the predicted lead-time to the failure, a low-cost recovery mechanism is chosen to trigger at an appropriate time.

V. SYSTEM, DATASETS, AND EVALUATION METRICS

A. Systems and Log Data

Table I shows the the four **unlabeled** log datasets and their supercomputers characteristics which used in our study. The four data logs are generated from three different real-world supercomputers clusters. Specifically, these system are of various scales (from 200 nodes to 5600 nodes), various interconnects (Infiniband and Aries Dragonfly), different file systems (Luster, MarFS, etc.), different processors, and different logging mechanism. The log datasets are (i) Syslogs, (ii) Rationalized Logs (abbreviated as RatLogs). Both Syslogs and RatLogs are collected form Ranger supercomputer(operated by Texas Advanced Computing Center (TACC)) [20] at different time and have been widely used for failure analysis [21], [22], [7], [23]. Rationalized Logs is upgraded logs based on a new logging framework called Rationalized logging framework for

Ranger supercomputer which replaced Sys logging framework. Unlike Syslogs, RatLogs has a few additional fields to record more information, for example, job-ID to identify each running job. (iii) Cray XC30 logs [16] generated by Cray XC30 supercomputer, and (iv) Cray XC40 logs [24] generated from Cray XC40 (Mutrino) supercomputer which was managed under a joint effort between Los Alamos National Laboratory (LANL) and Sandia National Laboratories (SNL). Mutrino, sited at SNL. Both Cray XC30 logs and Cray XC40 logs consist of two different types of logs (console and message).

B. Evaluation Metrics

Time Machine predicts (i) forthcoming log events (the entire health state of each node in the system) (ii) the node failure, and (iii) the expected lead time to failure. Therefore, our model is evaluated in three aspects. (1) We evaluate the accuracy by comparing the log events predicted by *Time Machine* versus the actual log events generated in reality by the four HPC systems using two text generation metrics (*Bleu* and *Rouge*). *Bleu* and *Rouge* metrics can complement each other for the NLP text generation (upcoming log events prediction in our case) tasks evaluation. Specifically, they correspond to the precision measure and recall measure, respectively. (2) We evaluate the prediction accuracy of our model regarding the nodes' failure events using several standard metrics including recall, precision, F1_score, Matthew's correlation coefficient (MCC), false-positive rate, and false-negative rate. We removed the log events predicted by our model (candidate) and the actual log events generated by the HPC system (reference) except failures in order to employ these metrics. (3) We evaluate the prediction accuracy of our model regarding the lead-time to node failures based on standard metrics: recall, precision, F1_score, and Matthew's correlation coefficient (MCC).

1) **Log Events Prediction Evaluation Metrics:** *Bleu* and *Rouge* are used to evaluate the prediction accuracy of the log events predicted by our model versus the actual log events generated by the four HPC systems in real-time as follows:

(i) **Bleu** (Bilingual Evaluation Understudy [25])

Bleu is an important indicator to measure the percentage of log events predicted by our model correctly (candidate) compared with the real log events as recorded by the HPC system (reference). A *Bleu* score is in the range of [0,1], where 0 indicates a mismatch and 1 means a perfect match.

The *Bleu* metric is defined in Equation 6 [25]:

$$Bleu = BP \times e^{(\sum_{n=1}^N w_n \log p_n)}$$

$$where \quad BP = \begin{cases} 1, & c > r \\ e^{1-r/c}, & c \leq r \end{cases} \quad (6)$$

$$p_n = \frac{\sum_{C \in \{Candidates\}} \sum_{n-gram \in C} Count_{clip}(n-gram)}{\sum_{C \in \{Candidates\}} \sum_{n-gram \in C} Count(n-gram)}$$

where *BP* refers to *brevity penalty* which is an adjusting factor that penalises the generated candidates that are shorter than the references, *r* refers to the length of the reference event sequence in the HPC system, *c* is the length of the candidate log sequence predicted by our model, *N* refers to the length

of ngrams (ngram denotes the connected sequence of n log events from a given candidate or reference), $w_n = \frac{1}{N}$ means the positive weights. As for the formula of p_n in Equation 6, $Count(ngram)$ and $Count_{clip}(ngram)$ refer to the number of ngrams for the candidate in the test set and the number of clipped ngrams for the candidate log sequence, respectively.

(ii) **Rouge** (Recall-Oriented Understudy for Gisting Evaluation N-gram Co-Occurrence Statistics) [26])

We use *Rouge* to measure the recall – the percentage of the real log events (reference) overlapped with the log events predicted by our model (candidate). The *rouge* score is always in the range of 0 to 1, where 0 indicates a mismatch and 1 means a perfect match. We present its definition in Equation 7 [26]:

$$Rouge = \frac{\sum_{S \in \{Reference\}} \sum_{(gram_n) \in S} Count_{match}(gram_n)}{\sum_{S \in \{Reference\}} \sum_{(gram_n) \in S} Count(gram_n)} \quad (7)$$

where *n* refers to the number of ngrams, $Count(gram_n)$ is the number of ngrams in the reference, and $Count_{match}(gram_n)$ means the maximum number of ngrams included by both reference set and candidate set.

2) **Node Failure and Lead-Time Evaluation Metrics:** We use some other well-known metrics to evaluate node failure prediction, which are summarized in Equations (8) to (11), where TP, FP, FN, and TN refer to True Positives (failure events are predicted correctly), False Positives (failure events are predicted incorrectly), False Negatives (failure events are missed by our model) and True Negatives (normal events are predicted correctly by our model), respectively. Precision and recall are two typical widely measures for failure prediction accuracy. F1_score is an aggregated metric by merging precision and recall. Matthew's correlation coefficient (MCC) is another aggregated metric which returns a high score if and only if the model performs well in all the four categories (TP, FP, FN, and TN).

$$Precision = \frac{TP}{TP+FP}, \quad Recall = \frac{TP}{TP+FN} \quad (8)$$

$$F1 \text{ Score} = 2 \frac{Recall \cdot Precision}{Recall + Precision} \quad (9)$$

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \quad (10)$$

$$FPRate = \frac{FP}{FP+TN}, \quad FNRate = \frac{FN}{TP+FN} \quad (11)$$

As for the evaluation of lead-time prediction, we also use the above standard metrics including recall, precision, F1_score, and Matthew's correlation coefficient (MCC).

VI. PERFORMANCE EVALUATION

To show the efficacy and applicability of our failure&lead-time prediction method, we carefully evaluate the performance of our model on four real-world supercomputer system logs: (i) SysLogs, (ii) Rationalized logs, (iii) Cray XC30 logs and (iv) Cray XC40 logs. They were logged by three different supercomputers and four different logging mechanisms at different operational times, which are all **unlabeled**. We

compare our approach to three state-of-the-art deep learning prediction techniques (a.k.a., baselines in the following text): Desh (LSTM) [8], Bi-LSTM [9], and GRU [10]. These three related works employ LSTM, Bi-LSTM, and GRU neural networks to predict HPC failures, respectively, and they have been verified as the best in class. We do not compare our work to traditional machine learning (ML) (e.g., Random Forest, SVM, DT, KNN) for two reasons. First, our model is a self-supervised learning that does not need labels whereas ML methods depend on labeled data (i.e., supervised learning-based techniques). Second, even ML algorithms can be utilized for classification (e.g., anomaly detection) and regression tasks, however, they are not designed to resolve text generation (prediction) tasks which is our research problem. In what follows, we show and discuss the major evaluation results.

A. Log Data Preprocessing

We preprocess the log data by sorting the log events according to timestamps, cleaning raw messages, and removing the duplicate messages in terms of the spatial and temporal correlations. Consequently, these log messages are converted to log sequences regarding their associated nodes, which corresponds to the phase I of our methodology. As shown in Table I, the quantities of the datasets' log messages are reduced significantly after the preprocessing step. Specifically, a total of 83087, 25272, 127161, and 49391 log sequences are constructed from Syslogs, Rationalized logs, Cray XC 30 logs, and Cray XC 40 logs, respectively. Each of the four logs is divided into training part and testing part. The training part accounts for 80% of the logs' data, while the testing part accounts for the remaining 20%.

TABLE I: Data Logs before and after the Preprocessing Phase

Log Name	# nodes	processors	Duration	# raw logs	# filtered logs
SysLogs	4,084	AMD Opteron	5 mon	43.6 m	2.3 m
RatLogs	4,084	AMD Opteron	6 mon	361 m	8.1 m
Cray XC30	5,600	IvyBridge	1 mon	133 m	15.3 m
Cray XC40	200	Haswell/KNL	16 mon	237 m	5.9 m

B. Training and Prediction Time Performance

Time Machine remarkably decreases the overall training time compared to the three state-of-the-art prediction approaches (LSTM (Desh), BiLSTM, and GRU). The overall training time includes two parts: (i) the training time in the regard of the prediction of the log event patterns and (ii) the training time for the prediction of the lead time to node failures. For the 4 HPC Systems Data Logs, *Time Machine* takes only 3.53 hours for the overall training on average, while other related works (LSTM, BiLSTM, and GRU) require 14.54 hours, 25.53 hours, and 13.22 hours, respectively. Also, our model is 15× faster over all baseline solutions in predicting the forthcoming log sequence of events. The training and prediction time speed-up results are detailed as follows.

1) **Log Events Training Time Performance:** The training time for learning to predict the log sequences and identifying

failure patterns, which is addressed by the first transformer-decoder stack, is drastically reduced 5.4 ~ 9.4× compared to three other state-of-the-art methods on average as shown in Table II. *Time Machine* requires only 0.7~3.83 hours in training, while LSTM, BiLSTM, and GRU require 3.79~20.75 hours, 6.69~36.66 hours, and 5.74~18.88 hours, respectively.

TABLE II: Log Events Training Time Performance in Hours

	<i>Time Machine</i>	LSTM	Bi-LSTM	GRU
SysLogs	1.60	8.86	15.30	7.89
Rationalized Logs	0.70	3.79	6.69	3.66
Cray XC 30	3.83	20.75	36.66	18.88
Cray XC 40	1.17	6.31	10.33	5.74
Average	1.83	9.93	17.25	9.04

2) Lead Time Prediction Training Time Performance:

The training time for learning to predict the lead time to node failures, which is addressed by the second transformer-decoder stack, also drastically decreased 3.74 ~ 7.23× compared to three other state-of-the-art strategies, as illustrated in Table III. Based on all the 4 HPC Systems Data Logs, *Time Machine* requires only 1.71 hours of training on average, while the other related works (LSTM, BiLSTM, and GRU) require 4.61 hours, 8.29 hours, and 4.17 hours, respectively.

TABLE III: Lead Time Training Time Performance in Hours

	<i>Time Machine</i>	LSTM	Bi-LSTM	GRU
SysLogs	1.2	3.17	5.91	2.84
Rationalized Logs	0.54	1.49	2.69	1.34
Cray XC 30	3.86	10.42	18.32	9.44
Cray XC 40	1.24	3.35	6.24	3.09
Average	1.71	4.61	8.29	4.17

3) Log Events Prediction Time Speed-up Performance:

Our model has the highest speed on the prediction of forthcoming log sequence of events (chain lengths), as shown in Figure 3. As evaluated in our experiments, the speedup of our model in predicting the forthcoming log sequence of events is 15× faster over all baseline solutions (LSTM (Desh), Bi-LSTM and GRU). The Figure 3 shows that only 5.78 secs are needed to predict a log sequence with the chain length of 1024, whereas state-of-the-art methods require 96~167 secs.

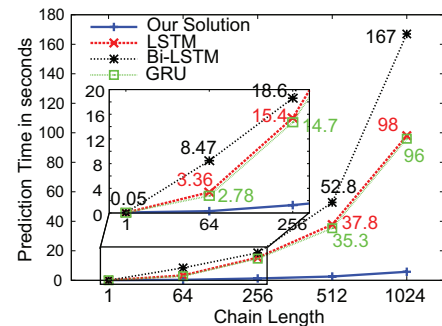


Fig. 3: Prediction Time of Chain Lengths

The low training time and high prediction performance of our model are attributed to the transformer-decoder mechanism's parallelization capability and positional encoding. More

specifically, it takes considerably less time than the RNN models (baselines) because the RNN model lacks parallel training and requires sequential learning. On the one hand, optimization of training time in HPC systems is significant because deploying the model online in real-time requires multiple fine-tunings of the model parameters over time in case of new log sequences and failure patterns appear. HPC system operators frequently elevate system components (software/hardware) and services to add new components (i.e., hardware or software) to improve high-performance computing demands. The increased number of high-resources-hungry jobs (e.g., applications) scheduled day-to-day on HPC systems also induces the logging management systems to generate new log patterns that have not been learned. On the other hand, Time Machine is particularly suitable for real-time failure prediction because of the high-speed prediction of forthcoming log sequence of events (chain lengths). It is noted that the growth of the number of events in the log chain does not come up with higher prediction times using *Time Machine*, whereas the baselines consume a long time to predict the same log sequence, and the speedup turns more and more obvious with increasing log chain lengths. So, our model is suitable for the real-time use-case with vast amount of logs generated in a short time (seconds), especially when the HPC components face erroneous behaviors that may lead to component crashes. Consequently, the high-speed prediction achieved by our model can boost the selection of the most suitable failure recovery action.

C. Overall Learning & Log Events Prediction Performance

We also evaluate the overall accuracy of our model in predicting the forthcoming log events (e.g., normal, errors, or failures) before the actual events occur, with respect to predict the entire system health state.

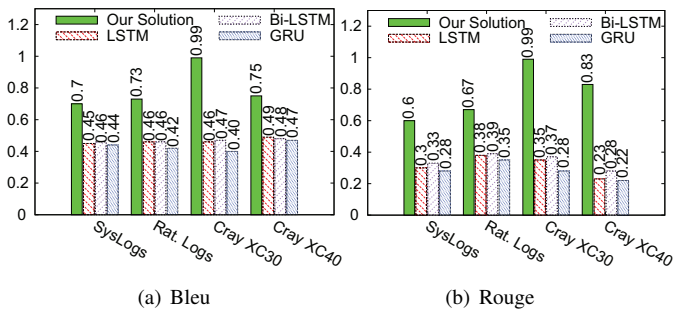


Fig. 4: Entire Health State Prediction Accuracy

Figure 4 (a) and (b) show the Bleu measure and Rouge measure of the entire health state prediction. Bleu and Rouge measure the degree of similarity (overlapping) between the candidate solution (predicted log events by our model or the baselines (LSTM, Bi-LSTM and GRU)) and the reference (log events generated by the supercomputer system in realtime). As shown in the figure, our transformer decoder-based approach achieves a Bleu score of 0.70~0.99 in predicting forthcoming log events based on the four system logs. In contrast, the other three state-of-the-arts have much lower Bleu scores (in

the range of only 0.4~0.5). On average, 79% of log events predicted by our model appeared in the events generated by the HPC system (the reference), compared to just 47% by Bi-LSTM (the best score from among the three baselines).

Also, our solution has a significantly higher Rouge score than the other three methods as depicted in Figure 4 (b). *Time Machine* obtains Rouge scores of 0.60, 0.67, 0.99 and 0.83 on the four system logs (SysLogs, Rationalized logs, Cray XC30 logs, and Cray XC40 logs), respectively. The Bi-LSTM baseline, however, obtains the Bleu scores of only 0.33, 0.39, 0.37 and 0.28, respectively. Similarly, both LSTM (Desh) and GRU based prediction solutions also have fairly low Rouge scores, which are in the range of 0.22 ~ 0.39. On average, $\approx 77\%$ of events generated by the supercomputer systems in real-time (the reference) appear in the log events predicted by *Time Machine* (candidate), compared to just $\approx 34\%$ by the best state-of-the-art prediction solution (i.e., Bi-LSTM).

We note that all the baseline solutions can hardly capture long-range dependencies/correlations between the events of long sequences, because they are all RNN-based models, which suffer from the memory loss issue for earlier events because of the vanishing gradients. By comparison, our solution is able to predict the upcoming log sequence as long as it correlates to the preceding events, as manifested by a high match between the forthcoming log events under our prediction model, and the events generated on the real system. In particular, the self-attention mechanism can efficiently identify the log entries of important events while moving the focus away from irrelevant ones and capturing long-range dependencies/correlations between events in long sequences.

D. Node Failure Prediction Performance Evaluation

Figure 5 shows the prediction accuracy of failure events under our model and baselines. We apply six measurements (Recall, Precision, MCC Score, F1-score FP-Rate, and FN-Rate) to evaluate our candidate solution and reference, based on logs with removed non-failure events.

As presented in Figure 5, *Time Machine* predicts upcoming node failures with high average precision score (0.91) on the four HPC system logs. In comparison, the best baselines are LSTM and Bi-LSTM, whose average scores (0.79) are lower than our model. For example, 82% of Ranger SysLogs node failures predicted by *Time Machine* indeed appear in the actual events generated by Ranger HPC system, compared to only 66% by Bi-LSTM (the best score from among the three baselines). Also, the results show that our technique obtains a better recall accuracy with an average score of 0.87 on the four HPC system logs. In comparison, the best baseline (both LSTM and Bi-LSTM) obtains the average score of only 0.74. *Time Machine* achieves a recall score of 0.74 on Ranger SysLogs; Bi-LSTM (best-baseline score) obtains a recall score only 0.59. This means on average, 74% of actual node failures generated by Ranger appear in SysLogs can be predicted by *Time Machine*, compared to only 59% by Bi-LSTM.

According to Figure 5, our solution has much higher MCC scores and F1-scores than all other baselines. Specifically, our

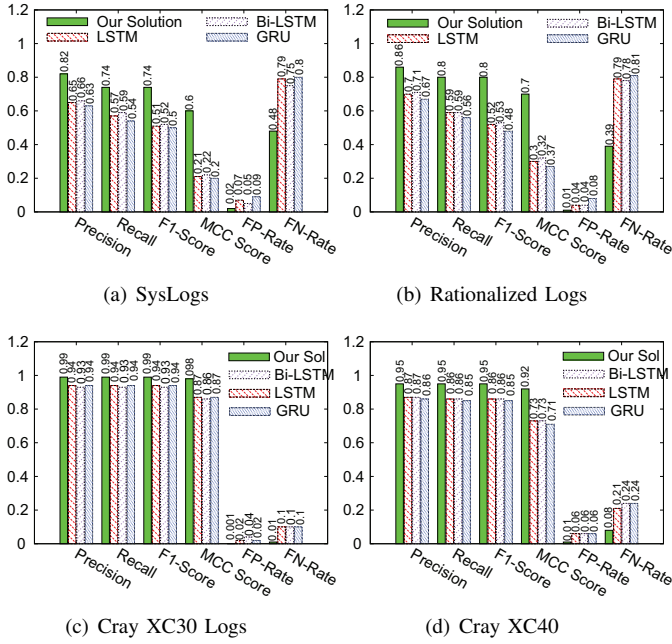


Fig. 5: Failure Prediction Performance

model achieves better prediction on the four system logs with MCC scores of 0.6~0.92, and the f-scores reach 0.74~0.99, which are both much higher than that of other baselines. For example, for SysLogs and Rationalized Logs, the MCC scores of our model can reach 0.6 and 0.7, respectively, which are much higher than the baselines’ (0.2~0.22 and 0.3~0.37).

Furthermore, the significant improvement of our model over baselines is also manifested by false positive rate (FP-rate) and false negative rate (FN-rate) (see Figure 5). For example, our model incurs only 1% FP-rate and only 24% FN-rate on average for all the four system logs, indicating that incorrect recovery actions are seldom triggered. However, the three baselines face higher FP-rate (5%) and FN-rate (47%). FN-rate are high in Syslogs and Rationalized Logs because the four models are not exposed to enough failure patterns for the prediction. Also, the interleaved logs are not removed before predicting failures. Moreover, experts agree that the prediction model’s usefulness remains highly valuable despite its limited accuracy. Even if only half of the failures are accurately predicted, half of the expensive global coordination needed for checkpoints can be avoided [8].

We explain the significant performance gain of our model over baseline models: Different log sequences are observed between the failure events and their associated errors, (e.g., kernel OS process, memory errors and network errors etc). Both failure relevant and irrelevant logs will interleave in such sequences, resulting in lengthy log sequences (e.g., there are still 3000+ events after the logs filtering phase). As such, it may take hours for failure to ensue following an error message. In contrast, our transformer-decoder-based model leverages multi-head masked attention layers and the positional encoding technique, which can completely avoid recursion, processing log sentences as a whole and understands associations between

TABLE IV: Lead Time To Failure Prediction Performance Evaluation on 4 HPC Systems Data Logs

		Time Machine			
		LSTM	Bi-LSTM	GRU	
SysLogs	Precision	0.92	0.91	0.87	0.91
	Recall	0.93	0.93	0.91	0.93
	F1-Score	0.92	0.91	0.89	0.91
	MCC Score	0.76	0.74	0.68	0.74
Rationalized Logs	Precision	0.92	0.89	0.89	0.89
	Recall	0.92	0.90	0.91	0.91
	F1-Score	0.92	0.89	0.89	0.89
	MCC Score	0.83	0.77	0.78	0.77
Cray XC30 Logs	Precision	0.99	0.99	0.99	0.99
	Recall	0.99	0.99	0.99	0.99
	F1-Score	0.99	0.99	0.99	0.99
	MCC Score	0.95	0.91	0.92	0.92
Cray XC40 Logs	Precision	0.97	0.95	0.95	0.95
	Recall	0.97	0.95	0.95	0.95
	F1-Score	0.97	0.95	0.95	0.95
	MCC Score	0.94	0.91	0.92	0.91

log events, leading to higher prediction capability.

E. Lead Time Prediction Performance Evaluation

This section details the performance of the lead-time prediction to failure events. As mentioned previously, a second key goal is to predict not only node failures but their lead times. This is important as appropriate recovery techniques can be triggered in timely manner, based on the failure lead-time.

Two key points need to be clarified first as follows:

- The failure prediction techniques (Bi-LSTM [9] and GRU [10]) do not support lead-time prediction originally. We implemented the lead-time component for each of them based on Bi-LSTM and GRU neural networks, respectively, so that lead-time classes can be predicted through a similar self-annotated multi-class classification problem LSTM [8], Bi-LSTM [9], and GRU [10].
- We define 6 lead-time classes; however our model is flexible in increasing/decreasing lead time classes based on the HPC system recovery mechanisms requirements.

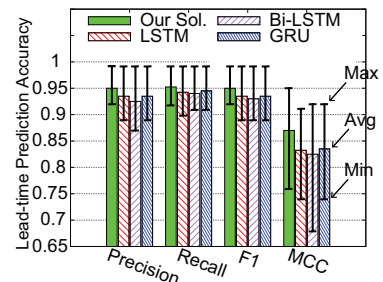


Fig. 6: Lead Time Prediction Performance

Table IV and Figure 6 show the failure lead-time prediction results with two critical observations. On the one hand, our model always has the highest accuracy from among all the four solutions. In absolute terms, the MCC and F1-score under our model can reach up to 0.87 and 0.95, respectively, which are higher than the scores resulted from the three state-of-the-arts (0.87 and 0.94, respectively). In particular, for SysLogs and Rationalized Logs, the MCC scores of Time Machine can get up to 0.76 and 0.83, respectively, while the baselines’

MCC scores are 0.74 and 0.78. On the other hand, all the four prediction methods have relatively high and similar accuracy in predicting failure lead-times on the four real-world system logs. Such a high and similar accuracy is primarily attributed to two reasons (i) modelling the lead-time prediction as a classification task rather than a regression task, (ii) our novel oversampling technique constructs massive, sufficient, and self-supervised training datasets. That is, the four prediction techniques are trained efficiently to learn each individual log event weight to estimate the class for the failure lead time. The reason we can model the lead-time prediction as a classification task is that there are only a few correction/recovery actions in total, and each action requires approximate lead times rather than exact lead times. The proactive recovery approaches may have largely different triggering/recovery costs. The typical proactive fault tolerance methods include job migration, checkpointing, process cloning, node quarantine, error correction code (ECC), and so on. The generic live process migration technology, for example, may require a prior notice of less than 10 seconds according to the experiments conducted by [27], while similar OS virtualization technologies may call for much longer lead times (a warning of 13-24 seconds in general). To assist redundant execution during failures, Rezaei et al. [28] showed that node cloning requires less than 200 seconds. Gupta et al. [29] demonstrated 5-9% of future failures may be prevented when quarantining the blades/cabinets by stopping scheduling jobs on the nodes for a few hours after a failure is manifested.

We elaborate on four key points about lead-time analysis:

- Our model and baselines can all accurately forecast a variety of the lead times to different types of their associated failures. For example, the lead times (60 seconds, 80 seconds, 120 seconds, 160 seconds) of Cray supercomputer failures due to different errors of OS kernel panic, job scheduler Slurm, a hardware non-maskable interrupt (NMI), and Machine Check Exceptions, respectively. It is essential to predict lead times to failures accurately in HPC systems. As the lead time accuracy increases, recovery solutions can be scheduled at the optimal time (not sooner or later), minimizing downtime and reducing costs. For instance, if the lead-time predicted is too early, unnecessary recovery actions may be executed, resulting in unnecessary costs. On the other hand, if the lead-time prediction is too late, the system component (e.g., nodes) may fail before proactive measures are finished, resulting in job failure, unplanned downtime and lost productivity.
- Our model predicts some failures of Ranger and Cray HPC systems that occurred with a very short lead time (only 5 seconds) after the occurrence of their associated errors. Some types of segmentation faults and memory corruption failures are examples of this class. Thus, In these cases, the HPC system management control should first quarantine the corresponding failure-prone nodes for a couple of hours to avoid waste of the compute resources. Second, the system should use a recovery action that takes less time (e.g., the generic live process migration) or avoid triggering any correction/recovery actions as most failures have already occurred in this case.
- Our model predicts accurately many failures with relatively long lead times. For instance, the lead times to different node soft lockups failures in the TACC Ranger supercomputer are ≈ 100 minutes, ≈ 125 minutes, ≈ 300 minutes of their associated errors (general protection, page fault, loss of service connection by via Network Interface Device (NID)). In this case, our technique (Time Machine) can help the HPC system’s administrator choose the best suitable lightweight error correction approach instead of an expensive solution. Also, it is practical to postpone triggering the recovery mechanism technique until a certain short period before the actual failure occurrence because most of these kinds of failures can be corrected themselves automatically.

F. Impact of Time Machine on Checkpoint-based Execution

We show how Time Machine can significantly reduce the execution overhead for the checkpointing-facilitated jobs, by leveraging a discrete event driven simulator [15]. Since the mean time between failures (MTBF) could be hourly or even less than a hour according to [30], we set the MTBF in our experiment to be 1 hour. The failure events are generated following a Poisson process with an average failure interval of 1 hour. The checkpointing and recovery time overhead are set to 60 seconds and 30 seconds, respectively. We evaluate two cases with different job workload lengths (6 hours and 12 hours). We compare four solutions as listed below:

- Ckpt (Young): This is a baseline solution, which adopts optimal checkpointing intervals calculated by Young’s formula [31]. Whenever a failure occurs, the job would roll back to the latest checkpoint to restart the execution. According to our experimental setting, the optimal checkpointing interval is 657.3 seconds.
- Ckpt (2000): We evaluate the execution with a sub-optimal checkpoint interval (2k seconds), considering the practical situation with potentially inaccurate MTBF.
- TimeMachine+Ckpt(Young): We combine our failure lead time predictor (Time Machine) and Young’s checkpoint intervals[31]. The lead times of the failures are predicted with a precision of 92% according to the worst case shown in Figure 6. For any failure predicted by Time Machine, we set the checkpoint ahead of the failure occurrence proactively. Since 92% failure events can be successfully predicted in time, the essential MTBF to the job execution is $10 \times \text{MTBF}$, which leads to a different optimal checkpointing interval (2078 seconds).
- TimeMachine+Ckpt(4000): We evaluate TimeMachine+Ckpt in the case with inaccurate MTBF, which has an inferior setting of checkpoint interval (4k secs).

Figure 7 shows the probability density function (PDF) of the job execution overhead ($= \frac{\text{wallclock} - \text{job_workload}}{\text{job_workload}} \times 100\%$), based on 1000 tests with different series of random failure events for each solution. We observe that combining

TimeMachine with checkpointing can significantly reduce the execution overhead from $\sim 20\%$ down to $\sim 7\%$ in the execution environment. It is also worth noting that the TimeMachine+checkpointing is immune to inaccurate MTBF estimation (see red curve vs. blue curve), while the traditional checkpointing suffers from substantially higher overhead when the MTBF is inaccurate (see black curve vs. green curve). The key reason for the above result is that the checkpoints can be set more efficiently with TimeMachine predictor, leading to significantly lower roll-back loss (abbreviated as *RB Loss*) during the execution, as demonstrated in Table V.

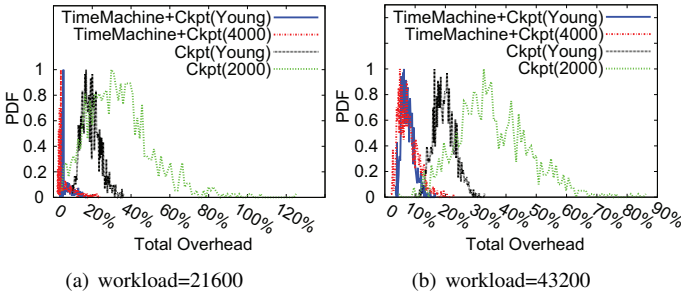


Fig. 7: Distribution of Total Execution Overhead

TABLE V: Performance Breakdown of Simulation (job workload = 6 hours) based on 1000 tests (in seconds)

Solution	stat	Load	Ckpt	Recv	RB loss	overhead
TimeMachine +	min	21600	660	30	2	3.2%
	avg	21600	992	154	432	7.3%
	max	21600	1642	162	2092	16.9%
Ckpt(Young)	min	21600	1920	30	48	9.3%
	avg	21600	1930	192	2007	19.1%
	max	21600	2120	450	5434	36.1%

VII. RELATED WORK

We analyze related works regarding log preprocessing/filtering, anomaly detection, and failure prediction.

First, multiple machine learning and deep learning techniques have been employed for log parsing/filtering. For example, **LogAider**[32], **Logram** [33], **DIP** [34], **LSWE**[35]. Recently, different parsers that rely on transformer variants have also been proposed, e.g., [36], [37].

Second, various machine and deep learning approaches (e.g., SVM, Random Forest, LSTM) are utilized for anomaly detection: classifying the log events or sequences as normal or anomalous (similar to anomaly classification in NLP). For instance, [7], **Logclass** [38], **Deeplog** [39], [40], **SiaLog** [41], **PLELog** [42], and CNN-based studies (e.g., [43], [44], [45]). Some studies [46], [47], [48] proposed anomaly detection methods based on Generative Adversarial Networks (GANs). Several studies have proposed to use self-attention with different transformer variants to detect HPC anomalies. These approaches rely on self-attention-based transformer-decoder [49], self-attention with different transformer-encoder variants [50], [51]. Compared to these approaches, our Time Machine is a real-time generative model for predicting log events, components (e.g., node) failures, and the lead time to the

predicted failures in HPC systems via utilizing two stacks of self-supervised transformer-decoders.

In order to ensure the mitigation techniques effective upon failures, the corresponding failures need to be predicted early enough compared with their occurrence moment. That is, it is critical to predict not only failure locations but also the lead times to these failures accurately, in order to trigger appropriate recovery and correction approaches in time. This makes this problem more complex than the previous three log analysis tasks (log parsing, anomaly detection, and failure diagnosis). Several failure prediction methods for HPC systems are proposed, including rule-based method and mathematics/analytics-based method (e.g., [52], [53], [54], [55], [56], [57], [58]). In comparison, our solution focuses on machine and deep learning categories. Many studies [59], [60], [13], [61], [62], [63], [64], [65], [66], [67], [68] leveraged ML techniques to predict the largescale systems failures based on calculating correlation association scores between the log events and failures, however extensive data engineering efforts and manual labelling are required to transform log messages to the numerical & statistical data before applying these solutions, and they are lack of predicting the failures' lead-time. Moreover, machine learning-based approaches generally lead to lower prediction accuracy compared to the deep learning methods which are the best related works. Different models have been proposed to predict failures in HPC systems based on deep learning algorithms. Similar to the ML-based methods, most of them do not perform inference leadtime analysis such as [69], [70], [9], [10], [14], and few frameworks include sub-models to predict the leadtime of failures such as **Desh** [8] and its extension **Aarohi** [5]. In comparison with deep learning methods, Time Machine is a more efficient failure and lead time prediction technique, which outperforms all RNN-based methods in all evaluation metrics based on our experiments. The work closest to ours, Clairvoyant [14], uses one transformer-decoder (12 decoders) to predict the soft lockup failures only. On the other hand, Time Machine uses **two** stacks of 24 decoders to predict possible failures and the failure lead-time accurately, as detailed in the introduction.

VIII. CONCLUSION

In this paper, we employ a self-supervised technique based on the transformer-decoder neural networks to build *Time Machine*, a novel real-time model that predicts failures and their lead-times in HPC systems. Time Machine and three state-of-the-art techniques are evaluated on four HPC logs. Experiments show that our model significantly outperforms the other techniques in both accuracy and speed. Time Machine can trigger proactive action at the right place and time at a substantially reduced cost. This paper is also the first to formulate the lead time prediction (a regression problem) as a self-annotated multi-class classification problem: predicting the class of the failure lead times. As future work, we will explore other generative models for the prediction of HPC failures and lead-times.

ACKNOWLEDGMENTS

This research was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (ASCR), under contract DE-AC02-06CH11357. We would like to warmly thank: **Dr. Anwesa Das** (Stanford University) for answering our questions about the Cray supercomputer system, **Dr. Edward Chuah** (University of Aberdeen) for answering our questions about Ranger system, **Texas Advanced Computing Center at UT-Austin** for providing the Ranger system logs, **Sandia National Laboratories (SNL) and Los Alamos National Laboratory (LANL)** for providing the Cray XC40 system logs, and the five anonymous reviewers for their valuable feedback.

REFERENCES

- [1] E. Chuah, A. Jhumka, S. Alt, J. J. Villalobos, J. Fryman, W. Barth, and M. Parashar, "Using resource use data and system logs for HPC system error propagation and recovery diagnosis," in *2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. IEEE, 2019, pp. 458–467.
- [2] A. Gainaru, F. Cappello, M. Snir, and W. Kramer, "Fault prediction under the microscope: A closer look into hpc systems," in *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–11.
- [3] —, "Failure prediction for hpc systems and applications: Current situation and open issues," *The International journal of high performance computing applications*, vol. 27, no. 3, pp. 273–282, 2013.
- [4] —, "Fault prediction under the microscope: A closer look into hpc systems," in *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012, pp. 1–11.
- [5] A. Das, F. Mueller, and B. Rountree, "Aarohi: Making real-time node failure prediction feasible," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2020, pp. 1092–1101.
- [6] E. Chuah, A. Jhumka, S. Alt, D. Balouek-Thomert, J. C. Browne, and M. Parashar, "Towards comprehensive dependability-driven resource use and message log-analysis for HPC systems diagnosis," *Journal of Parallel and Distributed Computing*, vol. 132, pp. 95–112, 2019.
- [7] K. A. Alharthi, A. Jhumka, S. Di, F. Cappello, and E. Chuah, "Sentiment analysis based error detection for large-scale systems," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2021, pp. 237–249.
- [8] A. Das, F. Mueller, C. Siegel, and A. Vishnu, "Desh: deep learning for system health prediction of lead times to failure in HPC," in *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, 2018, pp. 40–51.
- [9] J. Gao, H. Wang, and H. Shen, "Task failure prediction in cloud data centers using deep learning," *IEEE Transactions on Services Computing*, 2020.
- [10] M. m. S. Islam and A. Miranskyy, "Anomaly detection in cloud components," in *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. IEEE, 2020, pp. 1–3.
- [11] 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 *2010 International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2010, pp. 15–22.
- [12] T. Pitakrat, D. Okanović, A. van Hoorn, and L. Grunske, "Hora: Architecture-aware online failure prediction," *Journal of Systems and Software*, vol. 137, pp. 669–685, 2018.
- [13] J. Klinkenberg, C. Terboven, S. Lankes, and M. S. Müller, "Data mining-based analysis of hpc center operations," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, 2017, pp. 766–773.
- [14] K. A. Alharthi, A. Jhumka, S. Di, and F. Cappello, "Clairvoyant: a log-based transformer-decoder for failure prediction in large-scale systems," in *Proceedings of the 36th ACM International Conference on Supercomputing*, 2022, pp. 1–14.
- [15] "khalid8alharthi/time-machine: Code and datasets for a research paper accepted at dsn'23," <https://github.com/khalid8alharthi/Time-Machine>, (Accessed on 04/10/2023).
- [16] A. Das, F. Mueller, and B. Rountree, "Systemic assessment of node failures in hpc production platforms," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2021, pp. 267–276.
- [17] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.
- [18] P. Gage, "A new algorithm for data compression," *C Users Journal*, vol. 12, no. 2, pp. 23–38, 1994.
- [19] R. J. Williams and D. Zipsper, "A learning algorithm for continually running fully recurrent neural networks," *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.
- [20] "Ranger supercomputer begins new life - latest news - texas advanced computing center," <https://www.tacc.utexas.edu/-/ranger-supercomputer-begins-new-life>, (Accessed on 11/30/2022).
- [21] N. A. Simakov, J. P. White, R. L. DeLeon, S. M. Gallo, M. D. Jones, J. T. Palmer, B. Plessinger, and T. R. Furlani, "A workload analysis of nsf's innovative hpc resources using xdmoc," 2018.
- [22] J. Dongarra, T. Herault, and Y. Robert, *Fault Tolerance Techniques for High-Performance Computing*, 05 2015.
- [23] N. Gurumdimma, G. D. Bibu, D. B. Bisandu, and M. T. Alams, "Identifying recovery patterns from resource usage data of cluster systems," *Science World Journal*, vol. 13, no. 4, pp. 87–94, 2018.
- [24] mutrino, "The mutrino 2/15-6/16 dataset (12/16 release)," <https://portal.nersc.gov/project/m888/resilience/datasets/mutrino/about-mutrino1yr-v122016.pdf>, 2016.
- [25] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.
- [26] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, 2004, pp. 74–81.
- [27] C. Wang, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive process-level live migration in hpc environments," in *SC'08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE, 2008, pp. 1–12.
- [28] A. Rezaei, F. Mueller, P. Hargrove, and E. Roman, "Dino: Divergent node cloning for sustained redundancy in hpc," *Journal of Parallel and Distributed Computing*, vol. 109, pp. 350–362, 2017.
- [29] S. Gupta, D. Tiwari, C. Jantzi, J. Rogers, and D. Maxwell, "Understanding and exploiting spatial properties of system failures on extreme-scale hpc systems," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2015, pp. 37–44.
- [30] R.-T. Liu and Z.-N. Chen, "A large-scale study of failures on petascale supercomputers," *Journal of Computer Science and Technology*, vol. 33, no. 1, pp. 24–41, 2018.
- [31] J. W. Young, "A first order approximation to the optimum checkpoint interval," *Commun. ACM*, vol. 17, no. 9, p. 530–531, sep 1974. [Online]. Available: <https://doi.org/10.1145/361147.361115>
- [32] S. Di, R. Gupta, M. Snir, E. Pershey, and F. Cappello, "Logaidler: A tool for mining potential correlations of HPC log events," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2017, pp. 442–451.
- [33] H. Dai, H. Li, C. S. Chen, W. Shang, and T.-H. Chen, "Logram: Efficient log parsing using n-gram dictionaries," *IEEE Transactions on Software Engineering*, 2020.
- [34] D. Plaisted and M. Xie, "Dip: a log parser based on" disagreement index token" conditions," in *Proceedings of the 2022 ACM Southeast Conference*, 2022, pp. 113–122.
- [35] W. Meng, Y. Liu, Y. Huang, S. Zhang, F. Zaiter, B. Chen, and D. Pei, "A semantic-aware representation framework for online log analysis," in *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2020, pp. 1–7.
- [36] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "Self-supervised log parsing," *arXiv preprint arXiv:2003.07905*, 2020.
- [37] F. Setianto, E. Tsani, F. Sadiq, G. Domalis, D. Tsakalidis, and P. Kostakos, "Gpt-2c: A gpt-2 parser for cowrie honeypot logs," *arXiv preprint arXiv:2109.06595*, 2021.
- [38] W. Meng, Y. Liu, S. Zhang, F. Zaiter, Y. Zhang, Y. Huang, Z. Yu, Y. Zhang, L. Song, M. Zhang *et al.*, "Logclass: Anomalous log identification and classification with partial labels," *IEEE Transactions on Network and Service Management*, 2021.

- [39] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1285–1298.
- [40] A. Joulain, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "Fasttext. zip: Compressing text classification models," *arXiv preprint arXiv:1612.03651*, 2016.
- [41] S. Hashemi and M. Mäntylä, "Sialog: detecting anomalies in software execution logs using the siamese network," *Automated Software Engineering*, vol. 29, no. 2, pp. 1–28, 2022.
- [42] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. Zhang, "Semi-supervised log-based anomaly detection via probabilistic label estimation," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1448–1460.
- [43] S. Lu, X. Wei, Y. Li, and L. Wang, "Detecting anomaly in big data system logs using convolutional neural network," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 2018, pp. 151–158.
- [44] P. Cheansunan and P. Phunchongharn, "Detecting anomalous events on distributed systems using convolutional neural networks," in *2019 IEEE 10th International Conference on Awareness Science and Technology (ICAST)*. IEEE, 2019, pp. 1–5.
- [45] Z. Wang, J. Tian, H. Fang, L. Chen, and J. Qin, "Lightlog: A lightweight temporal convolutional network for log anomaly detection on the edge," *Computer Networks*, vol. 203, p. 108616, 2022.
- [46] Z. Zhao, W. Niu, X. Zhang, R. Zhang, Z. Yu, and C. Huang, "Trine: Syslog anomaly detection with three transformer encoders in one generative adversarial network," *Applied Intelligence*, pp. 1–10, 2021.
- [47] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2014.
- [48] B. Xia, Y. Bai, J. Yin, Y. Li, and J. Xu, "Loggan: A log-level generative adversarial network for anomaly detection using permutation event modeling," *Information Systems Frontiers*, vol. 23, no. 2, pp. 285–298, 2021.
- [49] Y. Guo, Y. Wen, C. Jiang, Y. Lian, and Y. Wan, "Detecting log anomalies with multi-head attention (lama)," *arXiv preprint arXiv:2101.02392*, 2021.
- [50] Y. Lee, J. Kim, and P. Kang, "Lanobert: System log anomaly detection based on bert masked language model," *arXiv preprint arXiv:2111.09564*, 2021.
- [51] V.-H. Le and H. Zhang, "Log-based anomaly detection without log parsing," *arXiv preprint arXiv:2108.01955*, 2021.
- [52] A. Das, F. Mueller, P. Hargrove, E. Roman, and S. Baden, "Doomsday: Predicting which node will fail when on supercomputers," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2018, pp. 108–121.
- [53] L. Guo, D. Li, I. Laguna, and M. Schulz, "Fliptracker: Understanding natural error resilience in hpc applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '18. IEEE Press, 2018.
- [54] A. Ma, F. Douglass, G. Lu, D. Sawyer, S. Chandra, and W. Hsu, "Raidshield: Characterizing, monitoring, and proactively protecting against disk failures," in *Proceedings of the 13th USENIX Conference on File and Storage Technologies*, ser. FAST'15. USA: USENIX Association, 2015, p. 241–256.
- [55] Y. Watanabe and Y. Matsumoto, "Online failure prediction in cloud datacenters," *Fujitsu scientific & technical journal*, vol. 50, no. 1, pp. 67–71, 2014.
- [56] C. H. Costa, Y. Park, B. S. Rosenburg, C.-Y. Cher, and K. D. Ryu, "A system software approach to proactive memory-error avoidance," in *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014, pp. 707–718.
- [57] A. Gainaru, M.-S. Bouguerra, F. Cappello, M. Snir, and W. T. C. Kramer, "Navigating the blue waters : Online failure prediction in the petascale era," 2013.
- [58] 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 *2014 IEEE International Conference on Cluster Computing (CLUSTER)*, 2014, pp. 103–112.
- [59] S. Ganguly, A. Consul, A. Khan, B. Bussone, J. Richards, and A. Miguel, "A practical approach to hard disk failure prediction in cloud platforms: Big data model for failure management in datacenters," in *2016 IEEE Second International Conference on Big Data Computing Service and Applications (BigDataService)*, 2016, pp. 105–116.
- [60] C. A. C. Rincon, J.-F. Pâris, R. Vilalta, A. M. K. Cheng, and D. D. E. Long, "Disk failure prediction in heterogeneous environments," in *2017 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, 2017, pp. 1–7.
- [61] M. Soualhia, F. Khomh, and S. Tahar, "Predicting scheduling failures in the cloud: A case study with google clusters and hadoop on amazon emr," in *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, 2015, pp. 58–65.
- [62] N. El-Sayed, H. Zhu, and B. Schroeder, "Learning from failure across multiple clusters: A trace-driven approach to understanding, predicting, and mitigating job terminations," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 1333–1344.
- [63] T. Chalermarawong, T. Achalakul, and S. C. W. See, "Failure prediction of data centers using time series and fault tree analysis," in *2012 IEEE 18th International Conference on Parallel and Distributed Systems*, 2012, pp. 794–799.
- [64] Q. Guan, Z. Zhang, and S. Fu, "Proactive failure management by integrated unsupervised and semi-supervised learning for dependable cloud systems," in *2011 Sixth International Conference on Availability, Reliability and Security*, 2011, pp. 83–90.
- [65] A. Sirbu and Ö. Babaoglu, "Towards operator-less data centers through data-driven, predictive, proactive autonomies," *CoRR*, vol. abs/1606.04456, 2016. [Online]. Available: <http://arxiv.org/abs/1606.04456>
- [66] X. LU, H. qiang WANG, R. jie ZHOU, and B. yu GE, "Autonomic failure prediction based on manifold learning for large-scale distributed systems," *The Journal of China Universities of Posts and Telecommunications*, vol. 17, no. 4, pp. 116–124, 2010. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1005888509604970>
- [67] B. Zhu, G. Wang, X. Liu, D. Hu, S. Lin, and J. Ma, "Proactive drive failure prediction for large scale storage systems," in *2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*, 2013, pp. 1–5.
- [68] A. Pelaez, A. Quiroz, J. C. Browne, E. Chuah, and M. Parashar, "Online failure prediction for hpc resources using decentralized clustering," in *2014 21st International Conference on High Performance Computing (HiPC)*, 2014, pp. 1–9.
- [69] T. Islam and D. Manivannan, "Predicting application failure in cloud: A machine learning approach," in *2017 IEEE International Conference on Cognitive Computing (ICCC)*, 2017, pp. 24–31.
- [70] S. Lu, B. Luo, T. Patel, Y. Yao, D. Tiwari, and W. Shi, "Making disk failure predictions smarter!" in *18th {USENIX} Conference on File and Storage Technologies ({FAST} 20)*, 2020, pp. 151–167.