Clairvoyant: A Log-Based Transformer-Decoder for Failure Prediction in Large-Scale Systems

Anonymous Author(s)

ABSTRACT

System failures are expected to be frequent in the exascale era such as current Petascale systems. The health of such systems is usually determined from challenging analysis of large amounts of unstructured & redundant log data. In this paper, we leverage log data and propose *Clairvoyant*, a novel self-supervised (i.e., no labels needed) model to *predict node failures in HPC systems* based on a recent deep learning approach called transformer-decoder and the self-attention mechanism. Clairvoyant predicts node failures by (i) predicting a sequence of log events and then (ii) identifying if a failure is a part of that sequence. We carefully evaluate Clairvoyant and another state-of-the-art failure prediction approach – Desh, based on two real-world system log datasets. Experiments show that Clairvoyant is *significantly* better: e.g., it can predict node failures with an average Bleu, Rouge, and MCC scores of 0.90, 0.78, and 0.65 respectively while Desh scores only 0.58, 0.58, and 0.25. More importantly, this improvement is achieved with faster training and prediction time, with Clairvoyant being about 25× and 15× faster than Desh respectively.

KEYWORDS

Transformer-decoder; LSTM; failure prediction; logs; HPC systems; deep learning

1 INTRODUCTION

High-end scientific applications, such as weather forecasting, are typically executed on high-performance computer (HPC) systems such as supercomputers. These systems comprise sophisticated hardware (HW) and software (SW) components (e.g., OS, parallel file systems) to support the resource-hungry applications. Node failures[1] in HPC systems can occur as a result of the scale and design complexity of the systems or due to faults occurring elsewhere in the system. Such failures typically lead to a significant computational overhead which, in turn, may have severe impact of system throughput.

In HPC systems, popular proactive failure management techniques are used such as task migration and checkpointing/restart. However, both techniques are expensive procedures and need to be used only when required, e.g., the computational overhead associated with these techniques may be exacerbated if they are wrongly triggered due to wrong failure prediction. Thus, it is important to develop efficient failure prediction techniques so that the overhead can be kept tractable. Effectiveness of current failure prediction approaches show a true positive rate of 50% in terms of actual failure identification and a false positive rate of less than 10%, meaning that the overhead of proactive techniques can be bounded [18, 31].

The SW of these HPC systems, such as OS and parallel file systems, typically generate a large volume of valuable log messages[2] that are recorded in a centralised log file. These log messages typically capture the health states of every component (e.g., nodes). For example, a log message may state that the memory of a particular node has been corrupted. As such, these event logs are critical for system administrators to assess the state of the system.

Although log files are nontrivial for analysis (e.g., they are often unstructured, duplicated or even incomplete [10]), extensive research on failure-related analysis using HPC system logs has been undertaken such as detecting anomalies e.g., [4], [42], [6], diagnosing the root causes of failures, e.g., [10, 14, 17], and detecting the errors that lead to system failures, e.g., [3, 41, 56, 80].

While error detection is important at system runtime, not all errors will lead to system failure due to in-built recovery procedures such as the use of ECCs. As such, any premature triggering of an error recovery technique would likely introduce extra overhead. Accordingly, to mitigate the impact of system failures on applications, it is critical to develop an efficient failure prediction mechanism alongside proactive failure management techniques [31]. Unfortunately, the failure prediction tools that determine when proactive failure management techniques should be activated is still insufficient [18]. This necessitates the development of failure prediction techniques that can flag impending failures ahead of time. Techniques that have been employed for failure prediction in HPC systems are, for example, support vector machines (SVM) [28], principal component analysis (PCA) [46], learning message patterns [75], Bayesian networks for hierarchical online failure prediction [61], and hidden semi-Markov models (HSMMs) [67]. Despite these contributions, these solutions have limited prediction accuracy or suffer from high computational overhead.

To the best of our knowledge, the recently proposed Long Short-term Memory (LSTM) and Bidirectional Long Short Term Memory (Bi-LSTM), used in [18] and [34] respectively, have been the most effective techniques for log-based failure prediction. However, they both suffer non-trivial weaknesses, e.g., due to recurrence learning, it is difficult to parallelize those approaches, leading to long training time. Another problem is the vanishing gradient problem, that causes the loss of earlier "memory" resulting in limited accuracy, i.e., long-range dependencies cannot be adequately captured.

As such, we develop **Clairvoyant**, a *self-supervised (no need for labels) transformer-decoder* based model to predict node failures in HPC systems by first predicting the future sequence of events (future health state) and then identifying if a failure is part of the sequence. Clairvoyant rectifies the limitations of LSTM implementations through the self-attention mechanism and parallelization. Denoting the predicted log sequence as $\mathbb{S}$ and a failure log event

[1]In the context of HPC systems, we will use node failures and system failures interchangeably.

[2]We will use the terms log events, events, log entries, and log messages interchangeably.

Conference'17, July 2017, Washington, DC, USA
2022. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

by $\mathcal{F}$, we then capture failure prediction of the node if $\mathcal{F} \in \mathbb{S}$, i.e., if a failure event appears in the predicted output log sequence. We run Clairvoyant on real-world datasets and the results obtained show that Clairvoyant *significantly* outperforms the state-of-the-art HPC failure predictor – *Desh* [18]. To the best of our knowledge, this paper is the first attempt to leverage the self-attention and transformer-decoder techniques to predict node failures in HPC systems. However, different log-based studies have utilized self-attention with different transformers variants for anomaly detection and log parsing, such as Trine [81], LAMA [38], LAnoBERT[48], NuLog[58], and [68].

We make the following contributions. (i) We develop *Clairvoyant*, a transformer-decoder based technique to predict component (node) failures in HPC systems. This is a generic model that can be applied to any other HPC systems or components since it is very common that the system failures are more or less correlated to the error messages. (ii) We evaluate the efficiency of Clairvoyant and Desh using two real-world logs from the Ranger supercomputer. The log data used in our experiments are very good representatives of large-scale HPC systems for failure analysis, as they are unlabelled, unstructured, and more complex than other HPC system logs (e.g., Cray and Blue Gene systems). (iii) Our results show that Clairvoyant significantly outperforms Desh both in prediction accuracy and in training and prediction time. To the best of our knowledge, this is the first paper to use transformer-decoder variant on failure prediction in HPC systems.

Paper structure: In Section 2, we present the system model and problem formulation. Section 3 presents the methodology behind Clairvoyant and we present the metrics used for performance evaluation in Section 4. We present the evaluation datasets in Section 5 and the results in Section 6. We discuss the related work in Section 7. We conclude the paper in Section 8.

2 MODELS AND PROBLEM FORMULATION

In this section, we present the system and fault models to be focused on in our research, as well as a formulation of the problem.

2.1 System Model

For HPC systems, a generic system model is described as follows. An HPC system consists of a set of compute nodes $C = \{C_1, \ldots, C_m\}$ provided to execute a set of jobs $J = \{J_1, \ldots, J_n\}$ over a set of production time-slots $T = \{T_1, \ldots, T_p\}$ [11]. A job scheduler and a collection of software, such as a file system and an operating system, are required to support these jobs execution. The job scheduler assigns jobs to production slots on specific nodes. A job may send data to and from the file system or to each other. As the system executes, log messages that capture the health of the system are generated and are sent to a central log file.

2.2 Fault Model

Without loss of generality, we assume that various discrete faults can be considered, depending on the abstraction level. One may consider faults occurring at the application level, the file system level or an aggregate cluster level. When a fault occurs, the resulting error leads to the output of an error message in the system log file. If the error is not adequately handled, a failure can occur, which will also be logged. In this paper, we only consider node failures [3] and focus on the prediction of such events though Clairvoyant can also be applied to failures of other components.

2.3 Problem Formulation

Challenges in log-based failure prediction: Informally, our approach for log-based failure prediction is as follows: given a sequence of log events, predict an incoming log sequence and identify if a failure log event is in the predicted sequence. However, there are two critical challenges, which are: (i) The instant at which the failure log event appears in the predicted sequence should neither be too soon nor too late as the failure management mechanism may be triggered at the wrong time and (ii) The component (i.e, node) that is going to fail needs to be clearly identified so that failure management mechanism is triggered at right "location".

We denote by $\mathcal{L}^r$, the set of log sequences of length at most r. Consider two sets: $\mathcal{L}^m$ and $\mathcal{L}^k, k \leq m$. The individuals in set $\mathcal{L}^k$ are called the possible extensions of the individuals in $\mathcal{L}^m$. Each individual in $\mathcal{L}^m$ can be assigned an output from $\mathcal{L}^k$, i.e., for each $s_i \in \mathcal{L}^m$, let $e_i \in \mathcal{L}^k$ be the true outcome to be predicted (i.e., the true log sequence that follows s_i). We model a (possibly randomised) predictor by a mapping $\mathcal{M} : \mathcal{L}^m \to \mathcal{L}^k$ such that $\mathcal{M}(s_i)$ is the predicted log sequence to follow s_i, i.e., $s_i \cdot \mathcal{M}(s_i)$ is a (future) predicted log sequence of length $(k + m)$, i.e., $s_i \cdot \mathcal{M}(s_i) \in \mathcal{L}^{m+k}$.

The two problems we address in this paper can be formulated as follows:

Definition 1 (Log Prediction). *Given a log sequence $s_i \in \mathcal{L}^m$, obtain a predictor $\mathcal{M}$ such that $\arg\min_{\mathcal{M}} \mathcal{D}(s_i \cdot \mathcal{M}(s_i), s_i \cdot e_i)$, where $\mathcal{D} : \mathcal{L}^{m+k} \times \mathcal{L}^{m+k} \to \mathbb{R}$ represents a distance metric for log sequences of length $(m + k)$ and where $\cdot$ represents sequence concatenation.*

In this case, we say that $\mathcal{M}$ *correctly extends* s_i if the distance is 0. Otherwise, we say that $\mathcal{M}$ *approximately extends* s_i. $\mathcal{D}$ is a distance metric on log sequences such that the distance is 0 when the logs are identical.

Definition 2 (Failure Prediction). *Given a log sequence $s_i \in \mathcal{L}^m$, its extension $e_i \in \mathcal{L}^k$ and a predictor $\mathcal{M}$ that approximately extends s_i, we say that $\mathcal{M}$ accurately solves the failure prediction iff $\mathcal{F} \in e_i \Leftrightarrow \mathcal{F} \in \mathcal{M}(s_i)$. We say that $\mathcal{M}$ approximately solves the failure prediction problem if $\mathcal{F} \in \mathcal{M}(s_i) \Rightarrow \mathcal{F} \in e_i$.*

It is interesting to observe that the (predicted) failure lead time is exact when $\mathcal{D}(s_i \cdot \mathcal{M}(s_i), s_i \cdot e_i)$ is 0, i.e., the failure event log does neither appear too soon nor too late. It is also worth noting that $\mathcal{D}$ will have a small value when the failure event appears at 'roughly' the correct time, i.e., at the correct place in the sequence. As such, developing an efficient mapping will help towards addressing the first challenge explained above.

So, for each node $C_j \in C$, given an input of a (previous) sequence of m log events $E_1, \ldots, E_m$ logged as node C_j's current health state, the aim of our transformer-decoder based model is to predict the sequence $E_{m+1}, \ldots, E_n$ future log events (i.e., the extension of $E_1, \ldots, E_m$), including failure events. The failure prediction is repeated for every node in $C_j \in C$ and the identity of the node that is predicted to fail will be known, thereby addressing the second challenge mentioned above regarding the failure location. The model calculates and predicts an upcoming log event probability $P(E_{(m+1):n})$ as follows:

$$P(E_{(m+1):n}) = \prod_{i=m+1}^{n} P(E_i|E_1, ..., E_{i-1}) \quad (1)$$

3 METHODOLOGY FOR CLAIRVOYANT

In this section, we first provide a high level overview of the proposed approach followed by a detailed description.

An Overview of the Proposed Approach Due to the serious limitations of existing techniques (e.g., LSTM based methods) and challenges of the failure prediction problem, novel and scalable approaches are needed. Recently, *transformer* neural network has made tremendous progress, primarily in natural language processing (NLP) tasks (e.g., text prediction) and tackled LSTM limitations, through the *self-attention mechanism* and *parallelization* processing.

These properties benefit log-based analysis in multiple ways: (i) The self-attention mechanism emphasizes the important part of the input data and fades out the rest. Focusing on log-based analysis where, by analogy, we consider an event log entry as a word and a sequence of log entries as a sentence; self-attention will help focus on the important event log entries while moving focus away from irrelevant events. (ii) The self-attention feature is amenable to *parallelization*, meaning that training and prediction time can be drastically reduced, compared to LSTM.

Driven by the self-attention and parallelization learning – the crux mechanisms of the transformers neural network [74], we develop a novel approach namely *Clairvoyant* based on the transformer-decoder variant [62] to predict HPC nodes' failures by first predicting the future sequence of events (future health state) for each node and then identifying if a failure is part of the sequence. Predicting a compute node's (or a component's) failures ahead is achieved through accurately predicting the forthcoming log events $E_{m+1}, ..., E_n$ based on the previous log events $E_1, \ldots, E_m$ by that node. Our proposed transformer-decoder-based technique can be deployed in real-time to assist the large-scale systems administrator as nodes' failure predictor. As shown in Figure 1[3], the proposed model is based on a transformer-decoder consisting of a stack of attention blocks, preceded by log message preprocessing and an input embedding, followed by a log events prediction. We can incorporate these steps as two main phases; log message preprocessing and log events (log sequence) learning&(prediction), which are described in detail as follows.

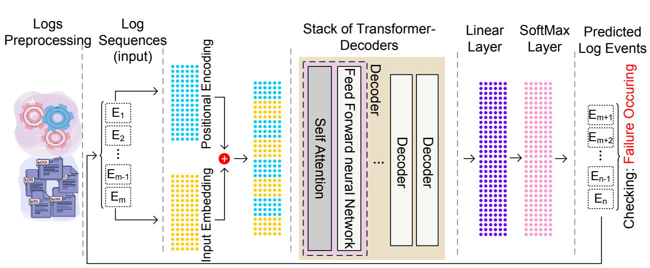


Figure 1: Failure and Health State Prediction Phases for Each Component (i.e., node)

[3]For simplicity, Figure 1 shows the prediction phases for one node.

3.1 PHASE I. LOG MESSAGE PREPROCESSING

In the first phase, standard NLP methods are used to clean textual log messages from all alphanumeric words, punctuation, stop words, variables that are not strings from log messages. After that, the duplicate messages are removed based on a time window as determined by the administrator. Then, (unique) text log messages are mapped onto corresponding log event IDs based on the unique events (templates) from log message preprocessing. Next, each node's log event IDs are concatenated into sequences (log event sequences) sequentially based on their timestamps, where each sequence contains 1024 events at most. Hence, the number of sequences from each node can be calculated by dividing the number of log events generated by that node on 1024. Each node's sequences of log events is tokenized by breaking them up and transforming them to their associated indices (i.e., numbers). Those indices are generated by taking all events present in the log data and creating a vocabulary dictionary. The decoder blocks is fed by nodes' log sequences one after another. Besides, transformer-decoder can, in parallel, perform until 1024 log events within the input sequence, which is an advantage over the recurrent neural network (RNN) architectures such as Long Short Term Memory networks (LSTM). Also, the Byte Pair Encoding (BPE) technique is employed in transformer-decoder architecture to tokenize the input, allowing the encoding of any unusual tokens, which are the IDs of log events in our case.

3.2 PHASE II. LOG EVENTS LEARNING AND PREDICTION

As stated before, our research aims to predict the failures of HPC system components (nodes) and the entire health state through generating a sequence of forthcoming log events based on their preceding log events sequence. Thus, our proposed approach is based on the transformer-decoder deep neural networks designed for sequence processing. The transformer's core component's self-attention mechanism considerably improves the connectivity among the elements in long sequences. Accordingly, we employ transformer-decoder neural network, a stack of decoder attention blocks preceded by an input layer to embed the sequence of real-time log events logged by the HPC node, and followed by linear and softmax layers to predict failures by two steps: predicting the future sequence of events and then identifying if a failure is part of the sequence. More design details are described in the following text. We refer the readers to read [62] for detailed background of the *transformer variant* which we will use to build our model.

3.2.1 *STEP 1: INPUT EMBEDDING*.

This step incorporates two types of encoding: log event embedding and log event's positional encoding, which are merged element-wise by dot-matrix multiplication:

Log Embedding: Each log event ID in input sequences is mapped into a vector of d_{model} dimension size, with continuous numeric values to represent that event learned through neural networks. By the end of the training, log event vectors' values represent the relation and dependency among these events.

Log Positional Encoding: Transformer avoids the recursion mechanism that is employed by RNNs, in order to enable parallel computation to minimize training time as well as the reduction in performance caused by lengthy dependencies. To this end, input embedding is associated with positional embedding to encode the order of the tokens (in our case, log events) and determine distances between the log events in the log sequences to the decoder blocks. The log event position i is encoded using sin and cosine periodic functions as follows:

$$\begin{cases} P_{(pos,2i)} = sin(\frac{pos}{10000^{\frac{2i}{d_{model}}}}) \\ P_{(pos,2i+1)} = cos(\frac{pos}{10000^{\frac{2i}{d_{model}}}}) \end{cases} \tag{2}$$

The even positions in the input vectors of log events in the sequence are calculated via the sin function, and the cos function is used for the calculation of odd positions. The positional vectors are then added to their corresponding log events input embeddings. Based on transformer-decoder architecture, each log event embedding in the input sequence incorporates one positional encoding vector for each of the 1024 positions(pos) [62] in the input; d_{model} refers to the size dimension (which is 768 in our design), and i refers to the index within the vector of log event.

Positional encoding is added to the input embedding to construct the input matrix X before being passed to the decoder stack to provide information about the position of those corresponding inputs (log events). For an input log event E_i, its embedding x_i in the input matrix X is defined as:

$$x_i = W_{embedding} * E_i + P_{E_i}, i \in 0, ..., I-1 \tag{3}$$

where **I** denotes the number of log events in the input sequence, $\boldsymbol{P_{E_i}}$ is positional encoding of $\boldsymbol{E_i}$, and the $\boldsymbol{W_{embedding}} \in \mathbb{R}^{E_{size} \times V_{size}}$ is the log event embedding matrix with embedding size $\boldsymbol{E_{size}}$ and the log events vocabulary size $\boldsymbol{V_{size}}$.

3.2.2 *STEP 2: DECODING AND LEARNING*.

In the next phase, the input matrix X, which is log events embedding vectors, is passed forward to a stack of decoders (12 decoder blocks) one after the another forming the main part of the model. These decoders are identical in their architecture and functions in which each decoder block consists of a multi-headed masked self-attention layer, feed-forward neural network (FNN) layer, and some normalization layers. Each decoder has its own weights in both sublayers (self-attention and FNN). The following details show how the decoder layers work.

(1) **Masked Self-Attention:** The masked self-attention mechanism allows the model to associate each individual log event to its preceding log events in the input sequences; this leads to understanding and capturing the relation, dependencies, and order occurrence among the log events in the input log sequences. Therefore, all associated and relevant log events in the sequence that reveal the connection with a particular log event are identified. As the correlation between those log events preceding that log event as these events receive higher scores (given more attention). The self-attention is achieved by creating three matrices for the decoder's input sequence X (in our case, a sequence of log events embedding). As stated in the previous step, the log events embeddings are combined into the input matrix X, where each row in X corresponds to a log event in the input sentence. A Query matrix (Q), a Key matrix (K), and a Value matrix (V) are created by multiplying X by three weight matrices, Query weight matrix (W_Q), a Key weight matrix (W_K), and a Value weight matrix (W_V), are trained during the training process. The matrices (W_Q), (W_K), and (W_V) have a smaller size dimension (64) than the log events embedding vectors (768) for better performance calculation of multiheaded attention(explained later). The input matrix X is passed through three linear layers W_Q, W_K, and W_V to produce the Query (Q), Key matrix (K), and Value matrix (V) matrices, respectively, where each row associated with a log event in the input sequence is defined by the following three equations:

$$\begin{cases} Q = W_Q \cdot X + b_Q \\ K = W_K \cdot X + b_K \\ V = W_V \cdot X + b_V \end{cases} \tag{4}$$

After the three matrices are created, several calculations are conducted to generate the masked self-attention Z, which can be depicted in the following formula:

$$\begin{aligned} Z &= MaskAttention(Q, K, V) \\ &= Softmax(mask(\frac{Q.K^T}{\sqrt{d_k}})V \end{aligned} \tag{5}$$

The masked self-attention is calculating the score for each log event against the preceding log events in the sequence by multiplying the dot product of that log event's query vector with its key $q_x \cdot k_x$, where q_x and k_x refer to the vectors of Q and K, respectively. As the correlation between the log event and its preceding log events increases, these events receive higher scores (given more attention). Then, those scores are divided by the square root of the dimension of the key vectors. The results are then passed to a softmax layer and they will be normalized all positive numbers with a sum being equal to 1. The obtained score from the softmax operation decides how much each individual event receives attention (focus) with respect to its current position in the input sequence. The relevant log events receive higher scores than other irrelevant ones. Next, the softmax scores are multiplying by each value vector v_x. This process keeps the relevant log events gaining high scores in the previous step and opting out unrelated log events because they are multiplied by tiny scores. Lastly, the output of the self-attention layer for that log event at that position is calculated by summing up the weighted value vector and send this vector along to the FNN layer. All these processes are performed in the form of matrix calculation in parallel for all sequence log events.

A "multi-headed" attention technique is employed (8 attention heads) to improve the self-attention layer's performance for two reasons. First, the ability of the transformer can be increased to extend attention to various positions. Second, the input embeddings can be projected into a varied representation subspace. Multiplying the input matrix X by the 8 multi-headed attention separate sets W_Q, W_K, and W_V weight matrices produces 8 sets of Query (Q), Key matrix (K), and Value matrix (V) matrices, respectively. Then, 8 different Z_i matrices are obtained. The FNN layer is expecting a single matrix to handle, thus the Z matrices are concatenated and multiplied with an additional weight matrix W_O to obtain the attention layer's output Z matrix that captures the information from all multi-heads.

(2) **Residual Connection and Normalization Layers:** Each transformer-decoder contains residual connection and normalization layers to make the training (learning) more effective. The layer

normalization is calculated as:

$$Norm_{layer}(Z) = \gamma \frac{Z-\mu}{\sigma} + \beta \tag{6}$$

where γ and β are learnable parameters, μ and σ are the mean and standard deviation of the Z' vector's elements [74] [62].

(3) **Feed-Forward Neural Networks:** Each transformer-decoder also contains two-layer feed-forward networks with a ReLU activation function applied to each position separately and identically. The first layer is the input layer to receive the output of the preceding layer, hidden layers that capture the hidden correlations among those input log events. The second layer is the output layer to pass forward what has been captured to the next step. Given a log sequence of vectors $h_1, ..., h_n$, the calculation of a position-wise FNN layer on a h_i is represented as:

$$FNN(h_i) = ReLU(h_i \cdot Z_{normalized} + b^1) \cdot W^2 + b^2 \tag{7}$$

where the $Z_{normalized}$, W^2, b^1, and b^2 are learnable parameters.

The other decoders work as the first decoder, and the output of each decoder is sent to the next decoder. The output of the final decoder is passed on to the linear and softmax layers.

3.2.3 STEP 3: LOG EVENTS AND FAILURE PREDICTION.

The vector of float values returned by the last decoder in the stack is transformed into a vector (logits vector) via a dense linear layer whose size is equal to the number of unique log events (vocabulary size). For instance, in our case, if there are V unique log events in log dataset, this would result in a logits vector of V cells values where each cell corresponds to a unique log event score. Finally, those scores are converted into positive probabilities with a total sum of up to 1 through a softmax layer, and the index of logit cell with the highest probability is selected. Based on the model vocabulary, the log event associated with that index cell is predicted at this time step as the output (the predicted log event). A log sequence of desired length of events is predicted via a loop that begins by predicting the next event based on its preceding events, then appends it to the input log sequence, and continues to return the subsequent events. The outcome of this looping process is a prediction of the health state of the entire HPC component system. Consequently, a failure is predicted if it appears in the predicted log sequence (our main goal). This means that Clairvoyant predicts a node failure by first predicting its forthcoming sequence of events and then identifying if a failure is part of the sequence.

4 EVALUATION METRICS

To demonstrate the viability of Clairvoyant for failure prediction of HPC systems, we examine two large datasets, obtained from two different logging mechanisms operational at different times, from the same HPC system, namely Ranger. We now introduce our evaluation metrics and then present the system, datasets, the evaluation results in the next sections.

Clairvoyant predicts not only future failures for HPC nodes but also the entire health state of every node. Thus, our model will be evaluated in two aspects. First, we evaluate the accuracy with which our model generates (i.e., predicts) log events of HPC nodes before the actual log events are generated. Second, we evaluate the accuracy with which our model predicts nodes' failures. For that purpose, two prediction-accuracy metrics supported by standard metrics including recall, precision, F1_score, Matthew's correlation coefficient (MCC), false-positive rate and false-negative rate, are utilized to evaluate our model.

Evaluating text generation (i.e., text prediction) in the NLP domain remains challenging because the generation task is open-ended. However, after a careful analysis of different text prediction evaluation metrics, we make use of the following metrics, namely *Bleu* and *Rouge*, which will be detailed below. The reason for choosing Bleu and Rouge is that they complement each other for the text prediction task as Bleu measures the *precision* of generated text (log events in our case) while Rouge measures *recall*. We detail the metrics below.

4.0.1 *Bleu (Bilingual Evaluation Understudy [59]).*

Bleu is a precision-based metric calculated by comparing the degree of similarity between a text candidate to one or more text references (in our case, just one reference). It was initially designed as a translation evaluation metric, but later it was used to evaluate text generation tasks, more specifically, to compare a generated text sequence (candidate) against a reference sequence. A Bleu score ranges from 0 to 1, with 0 indicating a complete mismatch and 1 a perfect match, and 0.6 or above indicating a good result.

In this paper, we use Bleu to measure how many log events predicted by our model (candidate) appear in (i.e., overlap with) the actual log events generated by the HPC system (reference).

The Bleu metric is defined as shown in Eq 8 [59]:

$$Bleu = BP \times e^{(\sum_{n=1}^{N} w_n \log p_n)} \tag{8}$$

where BP indicates the brevity penalty. $BP = 1$ when $c < r$ and $BP = e^{(1-\frac{r}{c})}$ when $c \leq r$, and where r indicates the length of the reference sequence (events generated by the HPC system) while c indicates the length of the candidate sequence (log events predicted by the model). N represents the length of the ngrams; $w_n = \frac{1}{N}$ indicates the positive weights, and p_n is the modified precision score as defined in Eq 9 [59].

$$p_n = \frac{\sum\limits_{C \in \{Candidates\}} \sum\limits_{n\text{-}gram \in C} Count_{clip}(n\text{-}gram)}{\sum\limits_{\acute{C} \in \{Candidates\}} \sum\limits_{n\text{-}gram \in \acute{C}} Count(n\text{-}gram)} \tag{9}$$

where $Count(ngram)$ is the number of ngrams for the candidate in the test set, and $Count_{clip}(ngram)$ is the number of clipped ngrams for the candidate log sequence.

4.0.2 *Rouge (Recall-Oriented Understudy for Gisting Evaluation N-gram Co-Occurrence Statistics) [51]).*

Rouge is a recall-based metric calculated by counting the number of text reference ngrams (in our case, just one reference) that appear in the text candidate (i.e., predicted sequence). A Rouge score can range from 0 to 1, with 0 indicating a complete mismatch, 1 a perfect match, and 0.6 and above indicating a good result.

In our case, we use Rouge to measure the recall how many of the actual log events generated by the HPC system (reference) appear in (overlap with) log events predicted by our model (candidate), as defined in Eq 10 [51]:

$$Rouge = \frac{\sum_{S \in \{Reference\}} \sum_{(gram_n) \in S} Count_{match}(gram_n)}{\sum_{S \in \{Reference\}} \sum_{(gram_n) \in S} Count(gram_n)} \quad (10)$$

in which n indicates the number of ngrams, $Count(gram_n)$ is the count of ngrams that appear in the reference, and $Count_{match}(gram_n)$ refers to the maximum ngrams number occurring in both reference and candidate sets.

The standard metrics that are used are defined in equations (11) to (15). The symbols TP, FP, FN, and TN refer to True Positives (failures are predicted correctly), False Positives (failures are predicted incorrectly), False Negatives (failures are missed by our model) and True Negatives (non-failures correctly predicted by our model), respectively[4].

In addition, we compute the following metrics: (i) the F1-Score that indicates the overall failure prediction accuracy with regards to the weighted average between recall and precision, (ii) The Matthew's correlation coefficient (MCC) is an adequate metric as it only returns a high score if it performs well in all four confusion matrix categories (TP, FP, FN, and TN), proportionate to the quantity of positive and negative classes in the test dataset. The MCC score can range from -1 to 1 where a score of -1 indicates a complete discrepancy between the actual and predicted results, a 0 score represents a random prediction and a score of 1 indicates that the prediction is perfect.

$$(failure)Recall, Precision = \frac{TP}{TP+FN}, \frac{TP}{TP+FP} \quad (11)$$

$$(nonfailure)Recall, Precision = \frac{TN}{TN+FP}, \frac{TN}{TN+FN} \quad (12)$$

$$F1\ Score = 2\frac{Recall \cdot Precision}{Recall + Precision} \quad (13)$$

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \quad (14)$$

$$FPRate, FNRate = \frac{FP}{FP+TN}, \frac{FN}{TP+FN} \quad (15)$$

In comparison to classification tasks, the automatic evaluation of text prediction tasks is a significant challenge [66]. Even though Bleu and Rouge are two of the few main metrics for Natural Language Generation (NLG), they have some drawbacks. However, and most importantly, these drawbacks do not apply to the problem of evaluating our model because only one reference and one candidate are used in our case. Nevertheless, for failure prediction experiments evaluation, Bleu and Rouge metrics are complemented using the standard metrics[5]. To use specific standard metrics, we use failure events and ignored all other log events from the candidate and reference.

5 EVALUATION SYSTEM AND DATASETS

In this section, we describe the Ranger system and the two log datasets from Ranger that we used for evaluating our approach.

[4]Precision and recall are calculated for both classes (failures and non-failures) and highlighted in the experimental results section even our test datasets are balanced.
[5]The results show the validity of using Bleu and Rouge to evaluate the health state and failure prediction because both correlate highly with the results of standard evaluation metrics.

Table 1: Data Logs before and after the Preprocessing Phase

Log Data	Durtion	From	To	# Before	# After
Syslogs	5 mon	Jan-11	May-11	43,639,722	2,346,780
RatLogs	6 mon	June-11	Nov-11	360,688,966	8,068,752

5.1 TACC RANGER AND LOG DATA

In our experiments, we adopt two real-world supercomputer system logs both generated by Ranger system, which have been widely used for failure analysis [3, 12, 23, 40, 70]. Ranger logs are very good representatives of large-scale systems for failure analysis. First, the Ranger system (operated by Texas Advanced Computing Center (TACC)) used to be one of the most powerful supercomputers in the world (ranked 5th in 2008 Top500 list and still ranked 50th in 2012 Top500 list). It contains 4K nodes with a total of 15,744 quad-core AMD Opteron microprocessors (featuring up to 62,976 cores) connected by a high-speed Infiniband network. During the entire life of Ranger, it served 4K+ scientists from 2,244 research projects, completing over three million simulation experiments. Second, the system/job failures exhibit similar characteristics across different supercomputers. For instance, the bestfit distribution of failure events is widely reported as Weibull distribution, based on many different prior studies [20, 65, 69, 78]. High spatial locality/correlation of the failures can be observed in both ORNL Titan supercomputer (made by Cray Inc.) [39] and ANL Mira supercomputer (manufactured by IBM) [20]. Last but not least, the Ranger system logs are unlabeled (no severity levels) and unstructured, whereas other cluster systems logs are labeled and more structured (e.g., Blue Gene systems), which means that the Ranger log data are much more challenging to handle than other system logs in practice. As such, we believe our evaluation results as well as our model are also applicable to other systems.

Ranger's jobs were managed by the Sun Grid Engine [26], generating two different system logs based on two *different* logging frameworks, namely SysLogs and Rationalized Logs. As shown in Table 1, our SysLogs dataset spans across five months (January to May 2011), while Rationalized Logs span six months (June to November 2011). We use both of them in our experiments and analysis.

5.1.1 ***Syslogs***. The Syslogs dataset is collected from a centralized logging system framework called syslog [1]. It uses POSIX standard similar to most linux cluster systems which allows flexible log formatting under different implementations. Table 2 describes the five fields for each log event in Syslogs.

Table 2: Description of Fields for Syslogs

field	example	description
timestamp	Jan 1 00:08:35	the time stamp
host	i151-306	the node where the job ran
system-id	kernel (linux)	ID of the system
application	Lustre	application name
text message	an error occurred ....[6]	detailed message of the event

5.1.2 ***Rationalized logs***. Rationalized logs was a new logging framework for TACC supercomputers instead of Syslogs. It differs

[6]full message of this example: "while communicating with 129.114.97.29 o2ib. The ost_write operation failed with -122 & key event message"

from Syslog, with a few additional fields being added to the message logs, for example, job-ID, which is a numeric identification number of each running job. This new logging framework was selected for the purpose of effectively analyzing the log-based failures in the system. Specifically, Rationalized logs would allow the system to parse the unstructured log messages more efficiently and may commit the error mappings and job failures more directly. The detailed fields and descriptions are given in Table 3.

Table 3: Description of Fields for Rationalized Logs

field	example	description
jobid	2184431	the identification number of job
timestamp	Oct 31 23:59:38	the time stamp
host	i119-107	the node where the job ran
prog	kernel	protocol name
text message	X Northbridge Error	key event message

Ranger Compute Node Soft Lockup Failures: Ranger administrators at TACC frequently encounter "compute node soft lockup" log events, indicating failures. A soft lockup is a state that causes the kernel of Linux OS to panic, be unresponsive, stuck, and loop endlessly, preventing other processes from being completed and eventually causing the nodes to crash. Soft lockup failures are recognized in log data by searching for the term *soft lockup*. Accordingly, the failures we aim to predict in this work are those soft lockups, which can be used to guide administrators in using mechanisms that will reduce the number of applications from failing [9] Several types of errors precede the failure of Ranger compute nodes (soft lock up), including Linux OS process errors, Lustre file-system errors, storage errors, network errors and software errors among others. Some of those errors propagate fast, quickly leading to soft lock up failures i.e., the sequence of log events between when the error message begins and when the soft lockup occurs is short. On the other hand, other errors take a long time to trigger their relevant failures to occur, resulting in lengthy sequences of log events being logged between the time those errors begin and the time they cause failures. Moreover, between Ranger failures and their induced error events, many interleaved & irrelevant log events in both short and long sequences are recorded, making the failure prediction process more challenging. For example, some errors take many hours to trigger the associated failures, resulting in extended and lengthy log sequences.

6 EVALUATION RESULTS

To show the efficacy and applicability of our technique across log data, we evaluate the performance of our model on two **unlabeled** real-world log datasets with different logging frameworks, SysLogs and Rationalized Logs, to predict potential soft lockup failures. Moreover, we compare our method to a state-of-the-art deep learning prediction technique, Desh [18], one of the best in class, which employs LSTM to predict HPC node failures. There are good failure prediction approaches as well such as CNN-LSTM based and Bi-LSTM based approaches proposed in [53] and [34], respectively. However, both techniques require long training time with slower prediction because they combine two neural networks CNN+LSTM and Bi-LSTM, respectively, with similar accuracy to Desh. In addition, our model is a self-supervised learning that does not need labels, unlike the supervised learning based methods (such as SVM, Random Forest, KNN, etc.) that depends on the labels. All our codes will be released once our paper is accepted.

6.1 Log Data Preprocessing

We developed a log preprocessor to sort the log events based on their timestamps, clean raw log messages, and remove the duplicate ones based on the spatial and temporal correlations, as determined by the administrator. After that, log messages are transformed into log sequences based on their associated nodes, as explained in the first phase of our methodology. Table 1 shows the quantities of both datasets' log messages before and after the preprocessing phase. 83087 log sequences are constructed from Syslogs, and 25272 log sequences are constructed from Rationalized Logs. Both logs are 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%. We verified that the testing part comprises the same number of log sequences containing a failure and log sequences having no failures (benign), to avoid an imbalanced dataset which could affect the evaluation metrics.

6.2 Predicting Entire Health State of Ranger Performance Evaluation

We implement our technique using a neural network architecture similar to the GPT2 model, which has a stack of 12 transformer-decoders with 12 attention heads for each layer, 768 dimensional states to encode log events into their embeddings, 1024 feed-forward sizes, and maximum log sequence input length being set to 1024. Additionally, we implement the baseline (Desh) model as explained in [18]. The training is conducted for 10 epochs and a batch size of 16 for our model as well as the baseline model.

6.2.1 *Training and Prediction Time Performance.*

Table 4 shows that the training time for learning using our model is drastically reduced 25.2× compared to Desh on average. Our approach requires only 1.64 and 0.71 hours in training on SysLogs and Rationalized Logs, respectively. By comparison, Desh requires at least 41 and 18 hours on both training sets, respectively.

Table 4: Training Time Performance in Hours

	Clairvoyant		Desh	
	SysLogs	R. Logs	SysLogs	R. Logs
1 Epoch	0.16	0.07	4.17	1.82
Entire Training	1.64	0.71	41.74	18.23

Furthermore, our model predicts the forthcoming log sequence of events 15.4× faster than Desh during the testing. As illustrated in Figure 2, only 0.30–5.78 secs are needed to predict a log sequence chain of lengths 64 to 1024 respectively, whereas the Desh technique requires 3.36–98.00 secs. This can be explained by the transformer-decoder mechanism's parallelization capabilities and positional encoding, which takes substantially less training and prediction time than the RNN models such as LSTM, which lack parallel training and require sequential learning. Therefore, the prediction of upcoming events using our solution on the testing data is very suitable for the real-time failure prediction scenario that requires fast forecasting to trigger the appropriate proactive recovery actions and avoid costly failures.

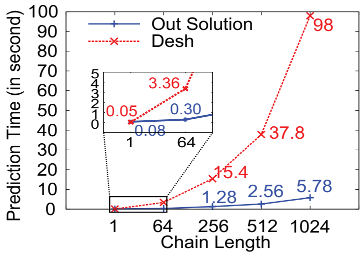
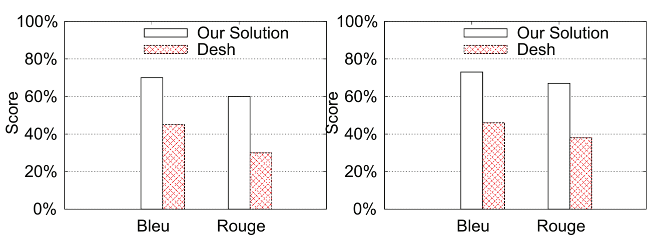


Figure 2: Chain Lengths(# Log Events) Prediction Time

6.2.2 *Overall Learning and Log Events Prediction Performance.*

We employ Bleu and Rouge metrics to measure our model's overall accuracy in generating (predicting) the nodes' forthcoming log events (whether informational, errors, or failures) before the actual log events are generated, with respect to the entire system future health state. Bleu and Rouge measure the degree of similarity (overlapping) between the candidate (predicted log events by our model and the baseline (Desh)) and the reference (log events generated by the Ranger system in realtime).



(a) SysLogs (b) Rationalized Logs

Figure 3: Bleu and Rouge for Entire Health State Prediction

As can be seen in Figures 3(a) and 3(b), our transformer decoder-based approach achieves a Bleu score of 0.70 and 0.73 on SysLogs and Rationalized Logs, respectively, in predicting forthcoming log events. In contrast, the Desh baseline obtains only 0.45 and 0.46, respectively. This means that on average, 72% of log events predicted by our model (the candidate) appeared in the events generated by the HPC system (the reference), compared to just 45.5%by Desh.

Also, as illustrated in Figures 3(a) and 3(b), results demonstrate that our technique obtains Rouge scores of 0.60 and 0.67 on SysLogs and Rationalized Logs, respectively. Desh, however, obtains only 0.30 and 0.38, respectively. This means that on average, 63.5% of events generated by the HPC system appear in the log events predicted by our model, compared to Desh's 34%.

The Bleu and Rouge scores imply that it is difficult for Desh, an LSTM-based model, to capture long-range dependencies/correlations between events in long sequences due to a loss of memory for earlier events caused by the vanishing gradient problem. On the other hand, our technique successfully predicts the future log events sequence depending on the preceding lengthy log sequence (predicting the upcoming health state from previous& current health state). This is indicated by a high match (overlap with) between the forthcoming log events predicted by our model(candidate), and the

Table 5: Failure Prediction Performance Evaluation on Both Data Logs

	SysLogs		Rationalized Logs	
	Our Sol	Desh	Our Sol	Desh
Bleu	0.89	0.56	0.91	0.59
Rouge	0.75	0.57	0.80	0.59
Failure Precision	0.97	0.76	0.99	0.86
Failure Recall	0.52	0.21	0.61	0.21
non-Failure Precision	0.67	0.54	0.72	0.55
non-Failure Recall	0.98	0.93	0.99	0.96
Overall Precision	0.82	0.65	0.86	0.70
Overall Recall	0.74	0.57	0.80	0.59
F1-Score	0.74	0.51	0.80	0.52
MCC Score	0.6	0.2	0.7	0.3
FP-Rate	0.02	0.07	0.01	0.04
FN-Rate	0.48	0.79	0.39	0.79

events generated by the HPC system (reference). The key reason is that the masked self-attention mechanism, which is the crux of our model, efficiently identifies 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.

6.3 Node Failure Prediction Performance Evaluation

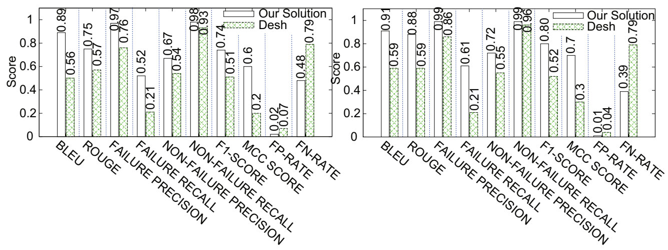
The main objective of this research is to predict the failures of nodes in an HPC system. As explained, our solution, Clairvoyant, predicts node failures by first predicting the future sequence of events for every node as evaluated above and then identifying if a failure is part of the sequence, as evaluated below using Bleu and Rouge, and the standard metrics mentioned in the Section 4. To this end, we calculate the values of the evaluation metrics by focusing on only failure prediction events, unlike the evaluation of the prediction of the entire health state, which involves multiple log events predicted by our model and the events generated by the HPC system.

Ranger often encountered compute node lockup failures; thus, our goal is to predict those soft lockup failures ahead of their occurrences, to trigger proactive failure management procedures in the system. Accordingly, we demonstrate our approach's effectiveness by evaluating the performance of our technique and comparing the results with baseline results on two different logs as follows:

As presented in Table 5, Figure 4 (a), and Figure 4 (b), our model predicts upcoming failures with a Bleu score of 0.89 and 0.91 on SysLogs and Rationalized Logs, respectively. In comparison, the Desh baseline scores just 0.56 and 0.59. In other words, on average, 90.0% of Ranger failures predicted by our model (the candidate) appear in the events generated by the HPC system (the reference), compared to only 57.5% by Desh.

Moreover, results show that our technique obtains better recall accuracy. Specifically, Clairvoyant achieves a Rouge score of 0.75 and .80 on SysLogs and Rationalized Logs, respectively. Desh obtains only 0.57 and 0.59, respectively. This means that on average, 77.5% of failures generated by the Ranger appear in the log events predicted by our model, compared to Desh's 58%.

As presented in Table 5, Figure 4 (a), and Figure 4 (b), the result of standard metrics shows that Clairvoyant predicts failures on SysLogs and Rationalized Logs with a high-precision score of 0.97 and 0.99, and the recalls can reach up to 0.52 and 0.61, respectively. In comparison, the Desh baseline achieves lower precision scores of 0.76 and 0.86 and lower recall scores of 0.21 and 0.21, respectively.



(a) SysLogs (b) Rationalized Logs

Figure 4: Failure Prediction Performance

Also, Clairvoyant predicts non-failure sequences correctly (benign sequences) on SysLogs and Rationalized Logs with a precision score of 0.67 and 0.72, and its recalls can reach up to 0.99 and 0.99, respectively. The Desh baseline, on the other hand, achieves lower precision scores (0.54 and 0.55) and also high recall scores (0.93 and 0.96), respectively.

We also check MCC, which is a reliable metric as it only returns a high score if it performs well in all four confusion matrix categories (TP, FP, FN, and TN), proportionate to the quantity of positive class (failure) and negative (non-failure) class in the test dataset. Moreover, we also utilize the weighted average of f1-score from the positive class (failure) and negative (non-failure) class for more accurate evaluation even our test set is balanced between the two classes. The results (see Table 5, Figure 4 (a), and Figure 4 (b)) show that our model (Clairvoyant) achieves better prediction on SysLogs and Rationalized Logs with MCC scores of .6 and 0.7, and the f-scores reach 0.74 and 0.80, respectively. In comparison, the Desh baseline achieves MCC scores of 0.2 and 0.3 and f-scores of 0.51 and 0.52, respectively.

Moreover, false positive rate (FP-rate) and false negative rate (FN-rate) also demonstrates the substantial improvement of our model. The FP-rate shows that 7% and 4% of Desh alarms on SysLogs and Rationalized Logs are false alarms (which would cause incorrect recovery actions), respectively. On the other hand, our model only drives 2% and 1% false alarms, leading to rare incorrect trigger recovery actions. Also, based on FN-rate, Desh significantly missed real node failures on SysLogs and Rationalized Logs (both 79%), while Clairvoyant missed only 48% and 39%.

As follows, we give a detailed explanation why our model significantly advances Desh in node failure prediction. As stated before, different lengths of log sequences are observed between Ranger's node failures and their associated errors&faults (such as software& kernel OS process, file-system errors, memory&storage errors, and network errors) for each Ranger component (e.g., nodes). Those sequences contain numerous interleaved & irrelevant log events, making the failure prediction process more challenging. For example, some errors take many hours to trigger the associated failures, resulting in extended and lengthy log sequences (e.g., over 2000 events even after the preprocessing phase). Nevertheless, due to multi-head masked attention layers and the positional encoding technique, our transformer-decoder-based model outperforms the recurrent neural network baseline (Desh) in that it completely avoids recursion, processing log sentences as a whole and understanding associations between log events. In other words, our approach's effectiveness in identifying the relationship between Ranger soft lockup failures and their preceding inducing errors comes from masked attention neural networks, which is the main component of the transformer-decoder and positional encoding layer that is combined with log events embedding. Accordingly, our solution can successfully predict node failures before they occur based on evaluation scores. The baseline – Desh, however, achieved lower accuracy and slower prediction because it is an RNN-based model that requires recurring recursion and sequential processing (log sequences processed event by event). Moreover, some log sequences are too long, and LSTM fails to capture the long relationship dependency range between failures and inducing error events as a result of the vanishing gradient problem, causing memory loss for earlier occurring events.

6.4 Node Failure Prediction Performance with Different Decoding Techniques

Transformer-decoder neural networks can be employed for text prediction with different decoding methods, including greedy search, beam search, basic sampling, top-K sampling, and top-P (Nucleus) sampling, and our model can work with each of them flexibly. This section examines the performance of two of these techniques on Rationalized Logs (similar results appeared with Syslogs) and how they affect the node failure prediction accuracy.

6.4.1 ***Greedy Search***. In the greedy search decoding technique, the next log event is predicted as the log event with the highest probability, and the next log event is updated through the following Eq 16 at each time step t.

$$E_t = argmax_E P(E_t | E_{1:t-1}) \tag{16}$$

Using the greedy technique, our model achieves high scores of Bleu and Rouge, f1-score, MCC 0.91, 0.80, 0.80, and 0.7 respectively, in predicting future soft lockup failures in Ranger Rationalized Logs.

6.4.2 ***Sampling Decoding with Different Temperature Values***. In NLP prediction task, the unpredictability of the predicted text (log events in our case) is controlled by a temperature (hyper-parameter), so we explored our model performance using basic sampling decoding with different temperature values. As shown in Figure 5, we observe that the failure prediction accuracy increases as the temperature value decreases, meaning that log events with high probability will be selected over the ones with low probability. Thus, we suggest using low-temperature values (≤ 0.5) to predict HPC systems to avoid predicting log events with low probability over those with high probability. In contrast, it is recommended to use ≥ 0.7 to perform well with NLP open-ended tasks.

7 RELATED WORK

Although this paper focuses on failure prediction, our work is also closely related to log processing (the first phase of log analysis) and error&failure detection. Thus, we discuss all the three categories, log preprocessing, error detection, and failure prediction.

Log processing is the first step for log analysis, and different log parsers are built based on self-attention mechanism and transformer models. For instance, [81] developed a log parser based on BERT, [58] built **NuLog** self-supervised parser based on a transformer-decoder, and [68] built a GPT-2 transformer based parser to preprocess Cowrie Secure Shell (SSH) honeypot logs. Several other

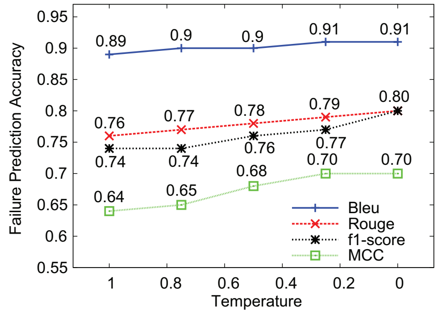


Figure 5: Failure Prediction with Different Temperature Scale based on Sampling Decoding

log parsers are developed with different methods but none of them are transformer based, such as **LogAider**[22] deployed for IBM BlueGene HPC series [21], **Craftsman** [79], etc.

In the area of error&failure and anomaly detection, different studies utilize self-attention with different transformer variants to detect HPC errors and anomalies. For example, Zhao et al. [81] proposed **Trine**, which is a generative adversarial network (GAN) based model including three transformer encoders to identify anomalies in system log data. Also, [38] proposed **LAMA**, which is a self-attention-based transformer-decoder to detect anomalies of large-scale systems. The LAMA model is applied for anomaly detection, where our model is applied for failure prediction. [76] also employed the transformer-encoder architecture to develop an unsupervised anomaly detection technique called **A2Log**. There are other recent research studies that utilized the self-attention with different transformer variants for error and anomaly detection such as LAnoBERT[48], LogAttention [24], and [47]. However, our model utilized self-attention and transformer neural network architecture to predict failures in HPC system components (nodes). Also, several detection techniques are proposed with different approaches that are not transformer based such as sentiment analysis based technique [3], **OVIS** for monitoring error system [5], etc.

Failure prediction is beyond and more challenging than the error and anomaly detection, because failure prediction requires to predict the upcoming failures significantly ahead of occurrence time such that various precautions can be executed in time. Basically, the failure prediction methods developed for HPC systems can be categorized into 4 classes[7]: rule-based method, mathematics/analytic-based method, machine-learning based method, and deep-learning based method. Rule-based failure prediction methods [15, 37, 55, 64, 75] generally try to establish some predicate rules such as if/then statements, which are extracted from the offline log datasets. mathematics/analytic based approaches [5, 13, 27, 29–31, 49, 73, 82] often perform failure prediction by probability analysis, correlation analysis, or curve fitting. Machine-learning based approaches [2, 7, 8, 25, 33, 35, 36, 45, 50, 54, 57, 60, 71–73, 77, 83] include failure prediction methods using any machine-learning (ML) techniques including decision tree/forest, regression, classification, Bayesian network, and Markov chain, etc. For instance, Ana et. al. [32] introduced a novel hybrid approach that combines signal analysis and data mining to predict failures in large systems integrated with failure avoidance techniques. Nie et. al. proposed a serious of different ML models to predict GPU error in HPC systems. The deep learning based approaches [18, 43, 53] leverage deep neural networks which generally are composed of much more layers than the plain neural networks, thus they often need a relatively long training on top of a large amount of samples (i.e., log messages).

From among all the four classes of failure prediction methods, the deep-learning based approaches have gained a significant favor over other types especially because of their outstanding accuracy. For example, Lu et al. [53] leveraged hybrid technique of the convolutional neural network and long short-term memory (CNN-LSTM) in disk fault prediction, which can reach a high accuracy for 10 days prediction horizon. Desh [18] is a deep Learning based approach obtaining high accuracy in HPC nodes failure prediction. Das et al. [16] proposed *Aarohi*, which is an extension to Desh with higher inference performance but it still suffers as inferior failure prediction capability and long training time, because it focuses only the inference stage. Moreover, Aarohi needs re-training and re-generation if any new failure patterns occur.

In comparison with Desh, we develop a more efficient failure prediction technique by leveraging the self-attention mechanism and transformer-decoder architecture, which outperforms Desh significantly in all evaluation metrics based on our experiments. Also, unlike Desh, our model does not eliminate interleaved logs before predicting failures to replicate the real-time HPC system in production. On the other hand, some other failure prediction techniques are supervised-learning based models, which obtain high accuracy, but they need labels. The self-attention mechanism and transformer-decoder is introduced in "Attention Is All You Need" research work by Vaswani et al [74]), followed by several variants of transformers (Language modeling (LM)) in NLP tasks domain such as BERT[19], GPT-2[62], Roberta[52], T5[63], etc. To the best of our knowledge, this paper is the first attempt to leverage the transformer-decoder in HPC failure prediction.

8 CONCLUSION

In this paper, we propose a novel self-supervised log-based approach called *Clairvoyant* to predict node failures. Clairvoyant solves two main problems with state of the art solution, such as Desh, by (i) being able to capture long-range dependencies and (ii) being amenable to parallelisation. To the best of our knowledge, Clairvoyant is the first attempt to leverage the transformer-decoder technique for failure prediction. Our experiments using two different datasets demonstrate a significant improvement in both prediction accuracy and learning/training performance over Desh – a LSTM-based failure predictor that has been verified as the best in class. The key findings are summarized as follows:

- Clairvoyant can obtain much higher Bleu score (0.90), Rouge score (0.78), MCC score (0.65) and F1-score (0.77) than Desh does (0.58, 0.58, 0.25, and 0.52, respectively).
- Clairvoyant is about 25× and 15× faster than Desh, respectively, during the training and prediction phases.

In the future, we plan to evaluate Clairvoyant using more system logs such as Cray and Blue Gene systems.

[7]The finer taxonomy (9 classes) for the HPC failure prediction methods can be found in Jauk et al.'s survey paper [44].

REFERENCES

[1] 2018. IEEE Standard for Information Technology–Portable Operating System Interface (POSIX(TM)) Base Specifications, Issue 7. *IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)* (2018), 1–3951. https://doi.org/10.1109/IEEESTD.2018.8277153

[2] Bikash Agrawal, Tomasz Wiktorski, and Chunming Rong. 2015. Analyzing and Predicting Failure in Hadoop Clusters Using Distributed Hidden Markov Model. In *Cloud Computing and Big Data*, Weizhong Qiang, Xianghan Zheng, and Ching-Hsien Hsu (Eds.). Springer International Publishing, Cham, 232–246.

[3] Khalid Ayedh Alharthi, Arshad Jhumka, Sheng Di, Franck Cappello, and Edward Chuah. 2021. Sentiment Analysis based Error Detection for Large-Scale Systems. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 237–249.

[4] Andrea Borghesi, Antonio Libri, Luca Benini, and Andrea Bartolini. 2019. Online anomaly detection in HPC systems. In *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 229–233.

[5] Jim Brandt, Ann Gentile, Jackson Mayo, Philippe Pébay, Diana Roe, David Thompson, and Matthew Wong. 2009. Methodologies for Advance Warning of Compute Cluster Problems via Statistical Analysis: A Case Study. In *Proceedings of the 2009 Workshop on Resiliency in High Performance (Resilience '09)*. Association for Computing Machinery, New York, NY, USA, 7âĂŞ14.

[6] Sathya Bursic, Vittorio Cuculo, and Alessandro DâĂŹAmelio. 2019. Anomaly Detection from Log Files Using Unsupervised Deep Learning. In *International Symposium on Formal Methods*. Springer, 200–207.

[7] Carlos A. C. Rincon, Jehan-FranĂğois PĂćris, Ricardo Vilalta, Albert M. K. Cheng, and Darrell D. E. Long. 2017. Disk failure prediction in heterogeneous environments. In *2017 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*. 1–7. https://doi.org/10.23919/SPECTS.2017.8046776

[8] Thanyalak Chalermarrewong, Tiranee Achalakul, and Simon Chong Wee See. 2012. Failure Prediction of Data Centers Using Time Series and Fault Tree Analysis. In *2012 IEEE 18th International Conference on Parallel and Distributed Systems*. 794–799. https://doi.org/10.1109/ICPADS.2012.129

[9] Edward Chuah. 2020. *Features correlation-based workflows for high-performance computing systems diagnosis*. Ph.D. Dissertation. University of Warwick.

[10] Edward Chuah, Arshad Jhumka, Samantha Alt, Daniel Balouek-Thomert, James C Browne, and Manish Parashar. 2019. Towards comprehensive dependability-driven resource use and message log-analysis for HPC systems diagnosis. *J. Parallel and Distrib. Comput.* 132 (2019), 95–112.

[11] Edward Chuah, Arshad Jhumka, James C Browne, Bill Barth, and Sai Narasimhamurthy. 2015. Insights into the diagnosis of system failures from cluster message logs. In *2015 11th European Dependable Computing Conference (EDCC)*. IEEE, 225–232.

[12] Edward Chuah, Shyh-hao Kuo, Paul Hiew, William-Chandra Tjhi, Gary Lee, John Hammond, Marek T. Michalewicz, Terence Hung, and James C. Browne. 2010. Diagnosing the root-causes of failures from cluster log files. In *2010 International Conference on High Performance Computing*. 1–10. https://doi.org/10.1109/HIPC.2010.5713159

[13] Carlos H.A. Costa, Yoonho Park, Bryan S. Rosenburg, Chen-Yong Cher, and Kyung Dong Ryu. 2014. 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*. 707–718. https://doi.org/10.1109/SC.2014.63

[14] Anwesha Das and Frank Mueller. 2018. Holistic root cause analysis of node failures in production HPC. *SC Poster Session* (2018).

[15] Anwesha Das, Frank Mueller, Paul Hargrove, Eric Roman, and Scott Baden. 2018. Doomsday: Predicting Which Node Will Fail When on Supercomputers. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. 108–121. https://doi.org/10.1109/SC.2018.00012

[16] Anwesha Das, Frank Mueller, and Barry Rountree. 2020. Aarohi: Making Real-Time Node Failure Prediction Feasible. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 1092–1101. https://doi.org/10.1109/IPDPS47924.2020.00115

[17] Anwesha Das, Frank Mueller, and Barry Rountree. 2021. Systemic assessment of node failures in HPC production platforms. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 267–276.

[18] 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 *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*. 40–51.

[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[20] Sheng Di, Hanqi Guo, Eric Pershey, Marc Snir, and Franck Cappello. 2019. Characterizing and Understanding HPC Job Failures Over The 2K-Day Life of IBM BlueGene/Q System. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 473–484. https://doi.org/10.1109/DSN.2019.00055

[21] S. Di, H. Guo, E. Pershey, M. Snir, and F. Cappello. 2019. Characterizing and Understanding HPC Job Failures Over The 2K-Day Life of IBM BlueGene/Q System. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 473–484.

[22] Sheng Di, Rinku Gupta, Marc Snir, Eric Pershey, and Franck Cappello. 2017. Logaider: 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, 442–451.

[23] Jack Dongarra, Thomas Herault, and Yves Robert. 2015. *Fault Tolerance Techniques for High-Performance Computing*. https://doi.org/10.1007/978-3-319-20943-2_1

[24] Qingfeng Du, Liang Zhao, Jincheng Xu, Yongqi Han, and Shuangli Zhang. 2021. Log-Based Anomaly Detection with Multi-Head Scaled Dot-Product Attention Mechanism. In *International Conference on Database and Expert Systems Applications*. Springer, 335–347.

[25] Nosayba El-Sayed, Hongyu Zhu, and Bianca Schroeder. 2017. 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)*. 1333–1344. https://doi.org/10.1109/ICDCS.2017.317

[26] Oracle/Sun Grid Engine. [n. d.]. https://www.oracle.com/it-infrastructure/. Online.

[27] 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 *2014 IEEE International Conference on Cluster Computing (CLUSTER)*. 103–112. https://doi.org/10.1109/CLUSTER.2014.6968768

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

[29] Ana Gainaru, Mohamed-Slim Bouguerra, Franck Cappello, Marc Snir, and William T. C. Kramer. 2013. Navigating the Blue Waters : Online Failure Prediction in the Petascale Era. https://www.mcs.anl.gov/papers/P5219-1014.pdf.

[30] Ana Gainaru, Franck Cappello, and William Kramer. 2012. Taming of the shrew: Modeling the normal and faulty behaviour of large-scale HPC systems. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium*. IEEE, 1168–1179.

[31] Ana Gainaru, Franck Cappello, Marc Snir, and William Kramer. 2012. 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*. 1–11. https://doi.org/10.1109/SC.2012.57

[32] Ana Gainaru, Franck Cappello, Marc Snir, and William Kramer. 2013. Failure prediction for HPC systems and applications: Current situation and open issues. *The International journal of high performance computing applications* 27, 3 (2013), 273–282.

[33] Sandipan Ganguly, Ashish Consul, Ali Khan, Brian Bussone, Jacqueline Richards, and Alejandro Miguel. 2016. 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)*. 105–116. https://doi.org/10.1109/BigDataService.2016.10

[34] Jiechao Gao, Haoyu Wang, and Haiying Shen. 2020. Task failure prediction in cloud data centers using deep learning. *IEEE Transactions on Services Computing* (2020).

[35] Jiexing Gu, Ziming Zheng, Zhiling Lan, John White, Eva Hocks, and Byung-Hoon Park. 2008. Dynamic Meta-Learning for Failure Prediction in Large-Scale Systems: A Case Study. In *2008 37th International Conference on Parallel Processing*. 157–164. https://doi.org/10.1109/ICPP.2008.17

[36] Qiang Guan, Ziming Zhang, and Song Fu. 2011. Proactive Failure Management by Integrated Unsupervised and Semi-Supervised Learning for Dependable Cloud Systems. In *2011 Sixth International Conference on Availability, Reliability and Security*. 83–90. https://doi.org/10.1109/ARES.2011.20

[37] Luanzheng Guo, Dong Li, Ignacio Laguna, and Martin Schulz. 2018. FlipTracker: Understanding Natural Error Resilience in HPC Applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '18)*. IEEE Press, Article 8, 14 pages.

[38] Yicheng Guo, Yujin Wen, Congwei Jiang, Yixin Lian, and Yi Wan. 2021. Detecting Log Anomalies with Multi-Head Attention (LAMA). *arXiv preprint arXiv:2101.02392* (2021).

[39] Saurabh Gupta, Devesh Tiwari, Christopher Jantzi, James Rogers, and Don Maxwell. 2015. 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, 37–44.

[40] Nentawe Gurumdimma, Gideon Dadik Bibu, Desmond Bala Bisandu, and Mammuan Titus Alams. 2018. Identifying Recovery Patterns from Resource Usage Data of Cluster Systems. *Science World Journal* 13, 4 (2018), 87–94.

[41] Nentawe Gurumdimma, Arshad Jhumka, Maria Liakata, Edward Chuah, and James Browne. 2016. Crude: Combining resource usage data and error logs for accurate error detection in large-scale distributed systems. In *2016 IEEE 35th Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 51–60.

[42] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. 2016. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 207–218.

[43] Tariqul Islam and Dakshnamoorthy Manivannan. 2017. Predicting Application Failure in Cloud: A Machine Learning Approach. In *2017 IEEE International Conference on Cognitive Computing (ICCC)*. 24–31. https://doi.org/10.1109/IEEE.ICCC.2017.11

[44] David Jauk, Dai Yang, and Martin Schulz. 2019. Predicting faults in high performance computing systems: An in-depth survey of the state-of-the-practice. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–13.

[45] Jannis Klinkenberg, Christian Terboven, Stefan Lankes, and Matthias S. MÃijller. 2017. Data Mining-Based Analysis of HPC Center Operations. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. 766–773. https://doi.org/10.1109/CLUSTER.2017.23

[46] Zhiling Lan, Ziming Zheng, and Yawei Li. 2009. Toward automated anomaly identification in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems* 21, 2 (2009), 174–187.

[47] Van-Hoang Le and Hongyu Zhang. 2021. Log-based anomaly detection without log parsing. *arXiv preprint arXiv:2108.01955* (2021).

[48] Yukyung Lee, Jina Kim, and Pilsung Kang. 2021. LAnoBERT: System Log Anomaly Detection based on BERT Masked Language Model. *arXiv preprint arXiv:2111.09564* (2021).

[49] Yinglung Liang, Yanyong Zhang, A. Sivasubramaniam, M. Jette, and R. Sahoo. 2006. BlueGene/L Failure Analysis and Prediction Models. In *International Conference on Dependable Systems and Networks (DSN'06)*. 425–434. https://doi.org/10.1109/DSN.2006.18

[50] Yinglung Liang, Yanyong Zhang, Hui Xiong, and Ramendra Sahoo. 2007. Failure prediction in IBM Bluegene/L event logs. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. IEEE, 583–588.

[51] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*. 74–81.

[52] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[53] Sidi Lu, Bing Luo, Tirthak Patel, Yongtao Yao, Devesh Tiwari, and Weisong Shi. 2020. Making disk failure predictions smarter!. In *18th {USENIX} Conference on File and Storage Technologies ({FAST} 20)*. 151–167.

[54] Xu LU, Hui qiang WANG, Ren jie ZHOU, and Bao yu GE. 2010. Autonomic failure prediction based on manifold learning for large-scale distributed systems. *The Journal of China Universities of Posts and Telecommunications* 17, 4 (2010), 116–124. https://doi.org/10.1016/S1005-8885(09)60497-0

[55] Ao Ma, Fred Douglis, Guanlin Lu, Darren Sawyer, Surendar Chandra, and Windsor Hsu. 2015. RAIDShield: Characterizing, Monitoring, and Proactively Protecting against Disk Failures. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST'15)*. USENIX Association, USA, 241âĂŞ256.

[56] Weibin Meng, Ying Liu, Shenglin Zhang, Federico Zaiter, Yuzhe Zhang, Yuheng Huang, Zhaoyang Yu, Yuzhi Zhang, Lei Song, Ming Zhang, et al. 2021. LogClass: Anomalous Log Identification and Classification with Partial Labels. *IEEE Transactions on Network and Service Management* (2021).

[57] Nithin Nakka, Ankit Agrawal, and Alok Choudhary. 2011. Predicting Node Failure in High Performance Computing Systems from Failure and Usage Logs. In *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*. 1557–1566. https://doi.org/10.1109/IPDPS.2011.310

[58] Sasho Nedelkoski, Jasmin Bogatinovski, Alexander Acker, Jorge Cardoso, and Odej Kao. 2020. Self-supervised log parsing. *arXiv preprint arXiv:2003.07905* (2020).

[59] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 311–318.

[60] Alejandro Pelaez, Andres Quiroz, James C. Browne, Edward Chuah, and Manish Parashar. 2014. Online failure prediction for HPC resources using decentralized clustering. In *2014 21st International Conference on High Performance Computing (HiPC)*. 1–9. https://doi.org/10.1109/HiPC.2014.7116903

[61] Teerat Pitakrat, Dušan Okanović, André van Hoorn, and Lars Grunske. 2018. Hora: Architecture-aware online failure prediction. *Journal of Systems and Software* 137 (2018), 669–685.

[62] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[63] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683* (2019).

[64] Raghunath Rajachandrasekar, Xavier Besseron, and Dhabaleswar K. Panda. 2012. Monitoring and Predicting Hardware Failures in HPC Clusters with FTB-IPMI. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*. 1136–1143. https://doi.org/10.1109/IPDPSW.2012.139

[65] Narasimha Raju, Yudan Liu, Chokchai Box Leangsuksun, Raja Nassar, and Stephen Scott. 2007. Reliability Analysis in HPC clusters. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.88.1149&rep=rep1&type=pdf.

[66] Ananya B Sai, Akash Kumar Mohankumar, and Mitesh M Khapra. 2020. A survey of evaluation metrics used for NLG systems. *arXiv preprint arXiv:2008.12009* (2020).

[67] Felix Salfner and Miroslaw Malek. 2007. Using hidden semi-Markov models for effective online failure prediction. In *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*. IEEE, 161–174.

[68] Febrian Setianto, Erion Tsani, Fatima Sadiq, Georgios Domalis, Dimitris Tsakalidis, and Panos Kostakos. 2021. GPT-2C: A GPT-2 parser for Cowrie honeypot logs. *arXiv preprint arXiv:2109.06595* (2021).

[69] S. Sharma and A. D. Clark. 2018. Introducing a Reliability Analysis Framework for High Performance Computing Environments. In *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. 1131–1138. https://doi.org/10.1109/ASONAM.2018.8508245

[70] Nikolay A. Simakov, Joseph P. White, Robert L. DeLeon, Steven M. Gallo, Matthew D. Jones, Jeffrey T. Palmer, Benjamin Plessinger, and Thomas R. Furlani. 2018. A Workload Analysis of NSF's Innovative HPC Resources Using XDMoD. arXiv:cs.DC/1801.04306

[71] Alina Sîrbu and Özalp Babaoglu. 2016. Towards Operator-less Data Centers Through Data-Driven, Predictive, Proactive Autonomics. *CoRR* abs/1606.04456 (2016). arXiv:1606.04456 http://arxiv.org/abs/1606.04456

[72] Mbarka Soualhia, Foutse Khomh, and Sofiene Tahar. 2015. 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*. 58–65. https://doi.org/10.1109/HPCC-CSS-ICESS.2015.170

[73] Joshua Thompson, David W. Dreisigmeyer, Terry Jones, Michael Kirby, and Joshua Ladd. 2010. Accurate fault prediction of BlueGene/P RAS logs via geometric reduction. In *2010 International Conference on Dependable Systems and Networks Workshops (DSN-W)*. 8–14. https://doi.org/10.1109/DSNW.2010.5542626

[74] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[75] Yukihiro Watanabe and Yasuhide Matsumoto. 2014. Online failure prediction in cloud datacenters. *Fujitsu scientific & technical journal* 50, 1 (2014), 67–71.

[76] Thorsten Wittkopp, Alexander Acker, Sasho Nedelkoski, Jasmin Bogatinovski, Dominik Scheinert, Wu Fan, and Odej Kao. 2021. A2Log: Attentive Augmented Log Anomaly Detection. arXiv:cs.LG/2109.09537

[77] Li Yu, Ziming Zheng, Zhiling Lan, and Susan Coghlan. 2011. Practical online failure prediction for Blue Gene/P: Period-based vs event-driven. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)*. 259–264. https://doi.org/10.1109/DSNW.2011.5958823

[78] Yulai Yuan, Yongwei Wu, Qiuping Wang, Guangwen Yang, and Weimin Zheng. 2012. Job failures in high performance computing systems: A large-scale empirical study. *Computers & Mathematics with Applications* 63, 2 (2012), 365–377. Advances in context, cognitive, and secure computing.

[79] Shenglin Zhang, Ying Liu, Weibin Meng, Jiahao Bu, Sen Yang, Yongqian Sun, Dan Pei, Jun Xu, Yuzhi Zhang, Lei Song, et al. 2020. Efficient and robust syslog parsing for network devices in datacenter networks. *IEEE Access* 8 (2020), 30245–30261.

[80] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 807–817.

[81] Zhenfei Zhao, Weina Niu, Xiaosong Zhang, Runzi Zhang, Zhenqi Yu, and Cheng Huang. 2021. Trine: Syslog anomaly detection with three transformer encoders in one generative adversarial network. *Applied Intelligence* (2021), 1–10.

[82] Ziming Zheng, Zhiling Lan, Rinku Gupta, Susan Coghlan, and Peter Beckman. 2010. 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)*. 15–22. https://doi.org/10.1109/DSNW.2010.5542627

[83] Bingpeng Zhu, Gang Wang, Xiaoguang Liu, Dianming Hu, Sheng Lin, and Jingwei Ma. 2013. Proactive drive failure prediction for large scale storage systems. In *2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*. 1–5. https://doi.org/10.1109/MSST.2013.6558427