Failure Prediction using Machine Learning in a Virtualised HPC System and application.

†*Bashir Mohammed · *Irfan Awan · *Hassan Ugail · ✦Younas Muhammad.

Received: date / Accepted: date

Abstract Failure is an increasingly important issue in high performance computing and cloud systems. As large-scale systems continue to grow in scale and complexity, mitigating the impact of failure and providing accurate predictions with sufficient lead time remains a challenging research problem. Traditional existing fault-tolerance strategies such as regular checkpointing and replication are not adequate because of the emerging complexities of high performance computing systems. This necessitates the importance of having an effective as well as proactive failure management approach in place aimed at minimizing the effect of failure within the system. With the advent of machine learning techniques, the ability to learn from past information to predict future pattern of behaviours makes it possible to predict potential system failure more accurately. Thus, in this paper, we explore the predictive abilities of machine learning by applying a number of algorithms to improve the accuracy of failure prediction. We have developed a failure prediction model using time series and machine learning, and performed comparison based tests on the prediction accuracy. The primary algorithms we considered are the Support Vector Machine (SVM), Random Forest(RF), k-Nearest Neighbors (KNN), Classification and Regression Trees (CART) and Linear Discriminant Analysis (LDA). Experimental results indicates that the average prediction accuracy of our model using SVM when predicting failure is 90% accurate and effective compared to other algorithms. This finding implies that our method can effectively predict all possible future system and application failures within the system.

Keywords Failure · Machine Learning · High Performance Computing · Cloud computing.

*School of Electrical Engineering and Computer Science University of Bradford, Bradford, UK, BD7 1DP
E-mail: †b.mohammed@bradford.ac.uk

✦Department of Computing & Communication Technologies Oxford Brookes University, Oxford, OX33 1HX UK.

1 Introduction

Failure prediction using machine learning is a major area of interest within the field of computing. It has received a considerable attention because it is an important issue in high-performance computing cloud system and plays an important role in proactive fault tolerance management. Research in large-scale computing relies on a thorough and deep understanding of what system failures in real systems look like. For instance, prior knowledge of failure characteristics can be used to improve system and node availability using resource allocation [1, 2]. Developing an accurate failure prediction model requires a critical understanding of the characteristics of real system failures. Additionally, certain statistical properties of failure can aid fault tolerance system designers to analyze and design an effective and reliable fault tolerance system [3-5]. Failures sources such as hardware, human error, software, malicious logic faults and network can hamper the execution of applications on high-performance computing cloud systems since the failure recovery process may require and unexpected large amount of time and resources. The impact of failure is even more consequential for large-scale distributed systems that consist of many computing nodes and clusters.

On the other hand, mitigating the impact of failure in a high-performance computing cloud datacen-

ter infrastructure is possible if accurate failure prediction mechanism are implemented [1]. For high speed, high performance computing systems such as large computer clusters with high risk of failure due to large number of system components, a reliable failure prediction technique is necessary [2]. Despite major efforts by researchers in both academia and industry, predicting system and component failure remains a primary issue in running large-scale computing infrastructure [3]. With the evolving new technological trends and growing system complexity, focusing on failure when designing systems for the next generation is vital. A particularly big concern is ensuring and maintaining high availability of the entire infrastructure. This is extremely important because failure to have a prior knowledge of the potential system failure might result in the following: Firstly, failure of any hardware component within the infrastructure might not only result to a temporary data unavailability, but in some extreme cases lead to permanent data loss. Secondly, market forces and technology trends may combine to make hardware system failures occur more frequently in the future. Thirdly, the size of hardware storage systems in modern large-scale high performance computing infrastructures might grow to an unprecedented scale with thousands of storage devices, making component failures even more difficult to detect. While there are several traditional fault tolerance techniques for dealing with and mitigating the impact of failures, there is a critical need to understand the future failure pattern and behavior of real systems[1],[4]. Such an understanding will not only help evaluate the future failure system component by fine tuning the existing techniques, but will aid in the design and development of new mechanisms.

However, to limit the impact of failure, applications, resources and services can be scheduled around predicted failures. Hardware or software failure can impede the execution of applications in the infrastructure since the amount of time and resources required to recover from failure can be unexpectedly high and expensive [5]. therefore, the availability and reliability of the system can be improved by predicting the system component and application failures. This will in turn enable us to improve the reliability of the system by predicting accurate future failures and enabling us to fully harness the potential of the next generation large-scale computing systems.

It is pertinent to note that our technique cut across on all the three layers of cloud computing, which are Infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS). Therefore, our model can predict possible hardware, software and application failure within the infrastructure. Details of the layers are briefly described as follows:

– Infrastructure as a service (IaaS), is the most basic and important cloud service model under which virtual machines, load balancers, fault tolerance, firewalls and networking services are provided. The client or cloud user, is provided with capability to provision processing, storage, networks and other fundamental computing resources, to deploy and run arbitrary software such as operating system and applications. Common examples of these services include Rackspace, Go-Grid, EC2, Google Apps, Concur, Cisco Webex, Citrix GoTo Meetings, Adobe Marketing Cloud, Facebook, Flickr) and Amazon cloud [14] .
– Under the PaaS model, a computing platform including APIs, operating system and development environments are provided as well as programming language execution environment and web servers. The client maintains the applications, while the cloud provider maintains the service run times,databases, server software, integrated server oriented architectures and storage networks. Various types of PaaS vendors offerings can include complete application hosting, development, testing and extensive integrated services that include scalability and maintenance. Some key players include Microsoft Windows Azure and Google Apps engine GoDaddy, Windows Azure, Apprenda, Google App Engine, Amazon Web Services, WordPress. The main benefit of these services include focus on high value software rather than infrastructure, leverage economies of scale and provide scalable go-to-market capability [15].
– SaaS provides clients the capability to use provider application executing on a cloud infrastructure. An entire application is available remotely and accessible from multiple client devices through thin client interfaces such as web browsers. Cloud user do not manage or control the underlying cloud infrastructure [2] but providers install and operate the application software. Example providers for this service include Salesforce, Facebook and Google Apps, Amazon EC2, Rackspace, Microsoft Azure, Google Compute Engine and Amazon Web Services [15-17].

The rest of the paper is structured as follows. Section 2 presents the related background of failure prediction using various strategies and algorithm. Section 3 presents the architecture of our study and a complete methodology of our research which include; the development of a predictive failure model using time series and machine learning.Section 4 presents an overview of ML approaches in container based HPC systems .Section 5 presents the discussion of our results obtained from

both the time series modelling and machine learning and summarizes the results. Section 6 provides some concluding remarks and findings and recommends a future research direction.

2 Related Work

Failure prediction using machine learning techniques in a large-scale distributed high performance cloud system has gained enormous attention in recent times, and a lot of research has been conducted in this area. However, very few work has attempted to fully analyze and predict high performance cloud system data empirically using a failure-in-production real-time data.The authors in [18] have made a good attempt to analyse the failure data of a large-scale production Cloud environment consisting of over 12,500 servers, which includes a study of failure and repair times and characteristics for both Cloud workloads and servers, but they never looked at the failure correlation between workload intensity and size of the system respectively. The author in [19] developed a machine learning approach for predicting individual component times until failure which they reported it as far more accurate than the traditional MTBF approach. Their algorithm was built to be able to monitor the health of 14 hardware samples and notify them of an impending failure well ahead of actual failure, providing adequate time to fix the problem before actual failure occurred. But the only drawback was that their model has not been trained on a module with real time failure. Hence, there is no assurance that this model will predict failure accurately and the data they used has not been made publicly available. While the authors in [20] introduced a new system failure prediction method using Support Vector Machines (SVM) based on the information contained in log files, where their proposed approach takes advantage of the sequential nature of log messages and determines which sequence of messages are precursors to failure. The author in [21] analyzed failure data that has been collected over the past 9 years at Los Alamos National Laboratory and includes 23000 failures recorded on more than 20 different systems, mostly large clusters of SMP and NUMA nodes. They studied the statistics of the data, including the root cause of failures, the mean time between failures and the mean time to repair, but they never applied any prediction techniques. Authors in [22] analysed the empirical and statistical properties of system errors and failures from a network of nearly 400 heterogeneous servers running a diverse workload over a year. Their results show that the system error and failure patterns are comprised of time-varying behavior containing long stationary intervals. These stationary intervals exhibit various strong correlation structures and periodic patterns, which impact performance but also can be exploited to address such performance issues. Authors in [23] characterizes the hardware reliability of Cloud datacenters from a number of data sources, but failed to analyze the failure of workloads and did not utilised a publicly available dataset in their experiment. Kavulya et al. [24] present workload failure characteristics from a production MapReduce supercomputing cluster, but this work is confined to MapReduce type jobs and does not consider workload repair or server failure characteristics.They also did not utilised publicly available dataset in their work. In [25], they used Bayesian network to predict failure probabilities. While the research seamed interesting, they did not disclose the dataset they used, thus making it hard to compare other Machine Learning (ML) Algorithms to their proposed model. Authors in [26] used an ensemble classifier to achieve hard drive failure prediction on a cloud infrastructure. The data they conducted their work on was acquired through two sources, Windows performance counts and Self-Monitoring, Analysis and Reporting Technology. This research closely resembles the intended work. However, they only considered hard disk failure in the cloud architecture while business systems relies on other components and not only hard drive. Rather a host of Hardware (such as: CPU, Disk, DIMM,Cable.etc) Recently,the authors in [27] used data acquired from cycles to predict Integrated Circuit (IC) failures. As in the case of [26] they also considered only one Hardware failure occurrence. They analyzed fourteen (14) hardware samples. The authors in [50] proposed an adaptive resource provisioning method using an application-aware machine learning technique that is based on the job history in heterogeneous infrastructures. Their experimental result indicates their proposed method can gratify user requests (cost and execution time) regarding its application and enhance resource usage effectiveness. The authors in [48] looked at an an adaptive overload threshold selection process using Markov decision processes of virtual machine in cloud data center. To address the problem, they modelled the overload threshold selection as a Markov decision process. With the solution of the improved Bellman optimality equation by the value iteration method, they claim their optimization model was resolved, and the optimum overload threshold is adaptively selected. The authors in [44][46], concentrated on software reliability modelling and software defect prediction using neural network classifier approaches. While the authors in [45] proposed a cost-effective and fault resilient reusability prediction mocek by using genetic algorithm, the authors in [47] proposed a deep

neural network based hybrid approach for software defect prediction using software metrics. Our approach is to use publicly available hardware dataset using time series and machine leaning (ML) algorithms to predict potential failure of all he system component and applications. Contrary to most of the state- of- the art research, we decided to use a public dataset so that other researchers in the field can compare their outcome with our proposed model and obtained results. Furthermore, in this work we are not limiting our experiments to a single system or application component failure, rather we attempt to predict several failures across the entire infrastructure. For more comprehensive review on other literatures the reader is referred to [28-36],[47-53].

3 Methodology

This section describes the complete methodology that would be deployed in this study. As the aim of this is to developed a model that that can predict possible failure system components and application in a high performance cloud system. We first start by using time-series modelling approach because the failure occurred a 5-year period, then we identified an ARIMA(1,1,1) model for the compounded failure dataset extracted from the computer failure database repository(CFDR) [38]. The data is a collection of different system components failure recorded over a given period of time. From the data set we were able to visualize the time stamps of the specific system component that failed (output), without the sources of the failure (input). Therefore, we were left with no option than to apply time series as each failure was recorded at a regular intervals at different point over time.

On the other hand, we went a step further by transforming the data and applying machine learning.The following steps were followed. We defined and identified our problem as multi-classification problem (multinorminal)which requires some prediction to enable us identify specific components that will fail in the future. We prepared and transformed our data and applied some supervised learning algorithms and also performed a comparison amongst them. We evaluated our algorithms and improved our result by selecting the best algorithm based on performance and accuracy The caret package in R was deployed because it provides a consistent interface into a number of machine learning algorithms and provides useful convenience methods for data visualization, data re-sampling, model tuning and model comparison. However, we decided to diversify the methodology by deploying five different ML algorithm which include support vector machines (SVM) and random forest (RF), linear discriminant analysis (LDA),k-nearest neighbors (kNN), and classification and regression trees (CART) and see the ramification. All of the above are supervised learning technique that require input and output so that the algorithms can learned. In order to apply different ML algorithm on our dataset, we generated the sources of failure of the system components from the study in [21], where the dataset used was also collected from CFDR database repository [37].

3.1 Data Collection

A historical dataset on system components failure for a period of five (5) years starting from 2001–2006 was collected [37]. The data was collected with the purpose of providing failure specifics for I/O related systems and components in as much detail as possible so that analysis might produce some useful findings. Data were collected for storage, networking, computational machines, and file systems in production use at NERSC. The data was extracted form a database used for tracking system troubles, called Remedy, and is currently stored in a MySQL database and available for export to Excel format. As part of the SciDAC Petascale Data Storage Institute (PDSI) project Collaboration this is the failure data for the High Performance Computing System-2 (MPP2) operated by the Environmental and Molecular Science Laboratory EMSL), Molecular Science Computing Facility (MSCF)[37],[38].

3.1.1 Data Pre-processing

The dataset [37] constituted an output variable representing the failed system components. In order to apply supervised machines learning algorithms, we needs to incorporates the input variables to dataset. As ealier mentioned we obtained the input variables from the study in [21] where the data used was extracted from the same domain [38].

In Table 1, we present five (5) sources of system components failure such as hardware; software; human error; network and undermined[21]. Throughout this study we would use the following acronyms HW, SW, HE, NW and UD respectively to denote the sources of system component failure. In the entire dataset, the undermined sources of failure appeared only once within the period under study. We therefore discard its effects in the analysis due to it less likelihood of occurrence.

We deployed combinatorics analysis[39],[40] and allocated the possible combinations of sources of system components failure to the output variables. Table 2 shows how the output variables are assigned to different combinations of input variables. This allocation is designed as follows:

Table 1 The sources of failure of a cloud-based systems considered in this study.

S/N	Sources of failure	Acronyms
1	Hardware	HW
2	Software	SW
3	Human Error	HE
4	Network	NW
5	Undetermined	UD

Supposed we have n different possible failure of a system components failure and out of which we are interested to take only k different combinations at a time. We are applying the combinatorics in order to avoid a repetition of the same combinations of sources of system components failure to a single output variables. The combinatorics is defined mathematically as: ${}^nC_k = \frac{n!}{k!(n-k)!}$ provided that $n \geq k$, where both n and k are positive integer. This procedure would be continue serially for each category of the input variable in accordance with the order of the dataset. We presents in short the overview of the transformed dataset in Table 2.

Table 2 Sample of transformed dataset used and their descriptions for model development.

S/N	X_1	X_2	X_3	X_4	*Output*
1	HW	SW	HE	NW	APPL
2	HW	SW	HE	UD	APPL
3	HW	SW	NW	UD	APPL
4	HW	HW	NW	UD	APPL
5	SW	HW	NW	UD	CABLE
6	SW	HW	NW	NW	CABLE
7	SW	HW	HE	NW	CABLE
8	SW	SW	HE	NW	CABLE
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
380	SW	HW	NW	UD	SCI_BP
381	SW	HW	NW	NW	SCI_BP
382	SW	HW	HE	NW	SCI_BP
383	SW	SW	HE	NW	SCI_BP

3.2 Failure Prediction using Time Series

We define time series in the context of a HPC or cloud-based infrastructure as a number of failures occurred to a system over a given period of time.

Let $X_1, X_2, X_3, \cdots, X_t$ be the number of failures of a system, and is mathematically defined as:

$$X_t = f(X_{t-1}, X_{t-2}, X_{t-3}, \cdots, X_{t-n}) + \varepsilon_t \quad (1)$$

where X_t is the value of X at time t,
then $X_{t-1}, X_{t-2}, X_{t-3}, \cdots, X_{t-n}$ represents the past values of X_t, and ε_t denotes white noise which has the distribution $\varepsilon_t \sim WN(0, \sigma^2)$. The ε_t is a stochastic term which does not follow any pattern and cannot be predicted. Basically, system failures are random, but it is rarely deterministic in a narrow sense from some identifiable causes.

For several decades, a time series models has been utilized in all fields of study for prediction[42]. The models like autoregressive (AR), moving average (MA) and exponential smoothing ranging from linear to non-linear regression and a host of many others. Box and Jenkins[43] developed a classical time series model called autoregressive integrated moving average (ARIMA). These techniques was successfully applied in various domain such as data center, complex industrial system and transportation networks and healthcare to predicts the failure of their systems..

3.2.1 The Autoregressive process (AR)

Suppose the time series $\{X_t\}$ at time t has p past values $X_{t-1}, X_{t-2}, X_{t-3}, \cdots, X_{t-p}$, then AR process of order p is denoted by $AR(p)$ defined as:

$$X_t = \vartheta_1 X_{t-1} + \vartheta_2 X_{t-2} + \vartheta_3 X_{t-3} + \cdots + \vartheta_p X_{t-p} + \varepsilon_t, \quad (2)$$

where $\varepsilon_t \sim WN(0, \sigma^2)$ and ε_t is uncorrelated with X_r for each $r < t$. Using backshift operator (2) can be written in short form as concise by $X_t = \vartheta(L)^{-1}\varepsilon_t$. In this process, the failure of a system dependent on the past causes or failure.

3.2.2 The Moving average process (MA)

This process is a memoryless, because the failure of a system does not depends on past causes or failure. In many situations, cloud-based infrastructure failed erratically and this type of process could be best to described such scenario. Let the time series $\{X_t\}$ is a moving average of order q denoted by $MA(q)$ and mathematically defined as:

$$X_t = \varepsilon_t + \phi_1\varepsilon_{t-1} + \phi_2\varepsilon_{t-2} + \phi_3\varepsilon_{t-3} + \cdots + \phi_q\varepsilon_{t-q} \quad (3)$$

where $\varepsilon_t \sim WN(0, \sigma^2)$ and $\phi_1, \phi_2, \phi_3, \cdots, \phi_q$ are constants. The X_t is a linear combination of $q+1$ white noise variables and are uncorrelated for all lags $s > q$, e.g X_t and X_{t-s}. The $MA(q)$ process can written in short form as $X_t = \phi(L)\varepsilon_t$

3.2.3 The Autoregressive moving average (ARMA)

This process is a hybrid of AR and MA where the pattern of the failure of a system can be attributed to two causes. The $ARMA$ model is a combination AR and MA models of order $p$and q respectively. Then, $ARMA(p,q)$ model is given by:

$$\vartheta(L)X_t \;=\; \phi(L)\varepsilon_t \tag{4}$$

3.3 Failure Prediction using Machine Learning

In this section we explore and discuss in detail some well-known and commonly used machine learning methods for the prediction of failure in a high performance computing cloud based environment. We fitted the algorithm into our model, then compared and selected the best out of all of them. The machine learning techniques deal with the issues of how to build and design computer programs that improve their performance and accuracy for some specific task based on past events or observations. As earlier stated, the methods we considered are: Linear Discriminant Analysis (LDA), Classification and Regression Trees (CART), k-Nearest Neighbors (kNN), Support Vector Machines (SVM) with a linear kernel and random forest (RF).

3.3.1 Linear Discriminant Analysis (LDA)

This is a method used in machine learning to find a linear combination of features that characterizes or separates two or more different classes of objects or events. The resulting combination may be used as a linear classifier or dimensionality reduction before later classification.
Consider a set of observations $\overline{x}$ for each sample of an even with known class $\mathbf{y}$. This is s set of sample called the training set. The problem is now to find a good predictor for the class $\mathbf{y}$ of any sample of the same distribution given an observation $\overline{x}$

Assuming that the conditional probability density functions (PDF)
$p(\overline{x}|y=0)$ and $p(\overline{x}|y=1)$ are both normally distributed with mean and covariance parameters ($\overline{\mu_0}$, Σ_0) and ($\overline{\mu_1}$, Σ_1) respectively.The Bayes optimal solution under this assumption is to predict points as being from the second class if the log of the likelihood ratios is bigger than some TH,
where $TH = Threshold$,
so that:
$(\overline{x} \; - \overline{\mu_0})^T)\sum_0^{-1}(\overline{x} \; - \overline{\mu_0}) + \ln|\Sigma_0| - (\overline{x} \; - \overline{\mu_1})^T$
$\sum_1^{-1}(\overline{x} \; - \overline{\mu_1})$ - $\ln|\Sigma_0| > TH$

3.3.2 Classification and Regression Trees (CART)

The CART algorithm is based on Classification and Regression Trees by Breiman et al (1984). A CART tree is a binary decision tree that is constructed by splitting a node into two child nodes repeatedly, beginning with the root node that contains the whole learning sample. Decision Tree is a recursive partitioning approach and CART split each of the input node into two child nodes, so CART decision tree is Binary Decision Tree. At each level of decision tree, the algorithm identifies a condition, which variable and level to be used for splitting input node (data sample) into two child nodes. Decision Tree building algorithm involves a few simple steps which are as follows:

- $Step-1$ Take labelled input data with a target variable and a list of Independent Variables.
- $Step-2$ Best Split: Find the best split for each of the independent variables.
- $Step-2$ Best variable: Select the best variable for the split.
- $Step-4$ Split the input data into left and right nodes.
- $Step-5$ Continue step 2-4 on each of the nodes until it meets the stopping criteria.
- $Step-6$ Decision tree pruning : Steps to prune decision tree built.

3.3.3 Random Forest (RF)

Random forest is another version of supervised learning algorithms in data mining. In this study, we proposed to build a model for predicting system components failure using this technique to enable comparing with SVMs.

Random forests consist of ensembles of classification or regression trees. This method allow the use bootstrap sampling on the training dataset and random feature selection during tree induction.

Suppose for a given number of features $\mathbf{M}$ the random forests samples $m \;>>\; \mathbf{M}$ to split at each creation of a tree node. In this case, the predictions are obtained through averaging. In random forest using regression trees, the results is the percentage increase of mean squared errors from 0-100%, with higher values indicating more important variables.

3.3.4 Support Vector Machines (SVM)

Support vector machines (SVMs) are supervised learning techniques used for regression analysis, classification problem and novelty detection. The fundamental idea behind SVMs is to choose the hyperplane that optimally differentiate two classes with the maximum margin.

For instance, given a training data $\mathbf{D}$, defined a set of n points and represented in the form:

$\mathbf{D} = \{(x_i, y_i)|x_i \epsilon \mathbf{R}^p, y_i \epsilon \{-1, -1\}\}_{i=1,2,\cdots,n}$ where y_i denotes the two classes either +1 or -1 to indicating the class for the point x_i belongs.

The hyperplane function $g(x)$ gives a linear discriminant in d-dimensions and splits the original space into two half-space: $g(x) = \omega^T x + b$, ω is a d-dimensional weight vector and b is scalar bias. If the dataset is linearly separable, a separating hyperplane can be found such that for all points with label -1, $g(x) < 0$ and $\forall$ points labelled +1, $g(x) > 0$.

SVMs are capable for solving problems with nonlinear decision boundaries by mapping the original d-dimensional space into d^*-dimensional space. So that the points $d^* > d$ are possibly be linearly separable. Using the transformation function ϕ, a new dataset is obtained in the form of transformation space $\mathbf{D}_\phi = \{\phi(x_i), y_i\}_{i=1,2,\cdots,n}$. This operation required to transformed space in the inner product $\phi(x_i)^T \phi(x_j)$, which is the kernel function ($\mathbf{K}$) between x_i and x_j. The kernels commonly used in SVMs are presented in Table 3.

Table 3 The kernel functions commonly used in SVM

Function	Kernels	Parameters
Polynomial	$\mathbf{K}(x_i, x_j) = (x_i^T x_j + 1)^q$	q
Gaussian	$\mathbf{K}(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$	σ
Radial basis	$\mathbf{K}(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}$	$\gamma \geq 0$

3.3.5 k-Nearest Neighbors (kNN)

The $k - nearest\ neighbors\ algorithm (kNN)$ is a non-parametric method used for classification and regression.In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether kNN is used for classification or regression:

In kNN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors, k is a positive integer, typically small). If $k = 1$, then the object is simply assigned to the class of that single nearest neighbor.

In kNN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors. Both for classification and regression, a useful technique can be to assign weight to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For instance, a common weighting scheme consists in giving each neighbor a weight of $1/d$, where d is the distance to the neighbor.

The neighbors are taken from a set of objects for which the class (for kNN classification) or the object property value (for kNN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required. A peculiarity of the kNN algorithm is that it is sensitive to the local structure of the data.The algorithm is not to be confused with k-means, another popular machine learning technique.

3.3.6 Architecture of the Study

In Fig. 1, we presents the proposed schematic diagram of failure prediction model of this study. This comprises of three (3) phases; The pre-processing phases, the training phase and the prediction phase respectively.

- Pre-processing phase (Model Identification and processing)
  The model identification is the first stage of building time series model after stationarity is achieved. With the aid of the autocorrelation function (ACF) and partial autocorrelation function (PACF), we can identify the appropriate model based on the pattern and order shown by the correlogram as shown in Fig. 4 and Fig. 5 receptively

- The Training phase (Model estimation and validation)
  After the appropriate model is identified, the next is the estimation of parameters using some conventional techniques such as least squares method, maximum likelihood estimation and method of moment etc. The model is then checked for accuracy and validation, even though postulation has been made that all models are wrong but some are better than others. For example, considering the properties of the residuals and check whether the residuals from an ARMA is normal regular distribution or random.
- The Model prediction phase
  At this stage, the identified model would be used to forecast future ahead. The estimated residuals of the model would be carefully examined to follow a white noise process.

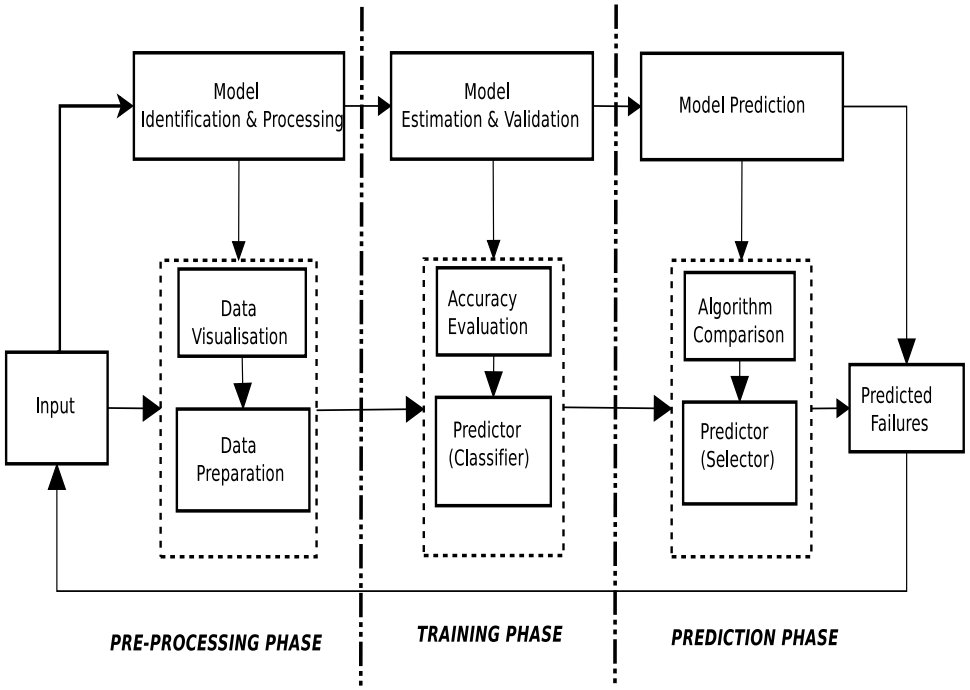

Fig. 1 Proposed System Model

4 Overview of ML approaches in Container based HPC systems

Container is a type of virtualization that takes place at the OS level[54]. It communicates with the host OS through system calls to the kernel. It simplifies and accelerate the process of building and isolating applications. They are lightweight and come with low overhead. They enable easier application sharing and reproducibility, because the container image includes both the application and its development environment, thereby making the OS kernel the interface layer between user-space containerized applications and the hardware resources of the host system that the application accesses during deployment[55], [56]. Recently, researchers have shown an increased interest in container based HPC systems and specifically how machine learning systems and approaches can be combined with its deployment capabilities[54], [55], [57]. Docker, LXC, LXD and Singularity are examples of some of the technologies which enable containerization[58]. They all have their merits and demerits and their suitability based on different use case scenarios. With respect to HPC systems which enables application processing at a much larger scale, such as MPI and schedulers (slurm, torque), Singularity may be the preferred option[59][56]. On the other hand, if the use case scenario is a micro service-based scaling, Docker, combined with orchestration technologies such as Kubernetes or Docker swarm may provide a better option[54], [60]. In addition, the recent development in the field of data science have led to the renewed interest in containerization especially with respect to Deep learning which is part of a broader family of machine learning methods[57], [61], [62]. Every deep learning framework has many dependencies and each dependent library has special version requirements resulting to all deep learning frameworks changing frequently. Containerization helps developers overcome these challenges[63]. Everything is bundled up into a single package that includes all the necessary pieces and parts[57]. However, deploying ML applications as containers and clustering them has the following advantages:

- The ability to make ML applications self-contained. They can be mixed and matched on any number of platforms and can operate in a highly distributed environment.
- The ability to expose the ML services systems that exist inside the containers as services or microservices. This in turn allows external applications and container-based to utilize those services at any time, without having to move the code inside the application.
- The ability to cluster and schedule container processing which allows the ML application that exists in containers to scale optimally. Even though the applications can be placed on cloud-based systems that are more efficient, container management

systems such as Google's kubernetes or Docker's Swarm are mostly recommended[64].
– Containers have mechanisms built in for external and distributed data access, so it they have the ability to leverage on common data-oriented interfaces that support many data models, thereby having access to data using well-defined interfaces that deal with complex data using simplified abstraction layers.

Following from the above summary and discussion, we can extend our ML based approach for container based HPC system, even though both the container and machine learning technologies are still at their emerging and development stages, both are based upon past technology patterns.

5 Discussion of Results

In this section we divided the discussion of results into two: The first presents and discusses the time series approach and the obtained results while the second discussion presents the five ML algorithm and their comparison based on our model . Fig. 2 presents the distribution of different system components failure based on their frequencies of occurrences. This is preliminary analysis in order to gain insight into the pattern of the components failure and to test for the normality of the failure data.

5.1 Discussion 1

In this section, we presents the analysis of system failure distribution across the time under study. The model formulation and their properties as well as prediction and evaluation are also presented.

5.1.1 System failure model

In Fig. 2, we plot the frequency of time dependent system failure in order to understand the pattern of its occurrence. The pattern of the system failure shows that it is not stationary as the mean and variance of the series keep changing over time. To remedy this scenario, we needs to deploy some technique of data transformations such as log-transformation and differencing method as well as discussing their properties.

We presents the plots of system failure frequency transformation in Fig. 3 using differencing and log, respectively. The log-transformation would not be appropriate in this case because there are some indefinite outcome as a results of zero recorded system failure. This is shown by the pattern of the system failure where the means and variances of the series are considerably unstable. In this study, we choose differencing method over log. This is because, the pattern exhibited by the differencing shows that mean and variance of system failure series is fairly constant.

5.2 Model Identification

We plots correlogram showing autocorrelation function (ACF) and partial autocorrelation function (PACF) of the system failure series (see Fig. 4 and Fig. 5). Using the ACF correlogram, we were able to identify moving average model of order 1, MA (1). While the autoregressive model is of order 1 as well, AR (1). The combination of the two models gives ARIMA (1,1,1) model, where 1 at the centre is the number of times the model is differenced. The system failure frequency of the series was differenced just once, because stationarity was achieved. All the values of the autocorrelation that fall within the two blue dotted lines are indicating non-significant at 95%. While those values that falls outside the 95% confidence interval indicates that they are significant.

5.2.1 Estimation of Parameter

Having identified the ARIMA (1,1,1) model for the system failure frequency, we can then estimate the parameters of the model. The ARIMA (1,1,1) model is mathematically represented by

$$y_t = \phi y_{t-1} + \varepsilon_t + \vartheta_1 \varepsilon_{t-1} \tag{5}$$

where ε_t is the random shock occurring at time t, which has the distribution as $\varepsilon_t \sim WN(0, \sigma^2)$. We estimated the following parameters $\phi_1 = -0.1016$ and $\vartheta_1 = -0.5784$, log-likelihood = -178.3, AIC = 362.61, and the association standard errors of the model are 0.2531 and 0.2085 respectively. We therefore write the ARIMA (1,1,1) model for system failure frequency as

$$y_t = -0.1016_{t-1} + \varepsilon_t - 0.5784\varepsilon_{t-1} \tag{6}$$

where $\varepsilon_t \sim WN(0, 1.534)$.

We presents in Fig. 6 the decomposition of additive time series plot and Fig. 7 the prediction capability of ARIMA (1,1,1) which shows the prediction region. We also evaluates the accuracy of the model. Having evaluated the model, we were able to obtained the following indices RMSE = 38.6%, MAE = 31.6% and MASE = 23.5% respectively. This shows that the model is very robust as it gives error allowance of less than 40%.

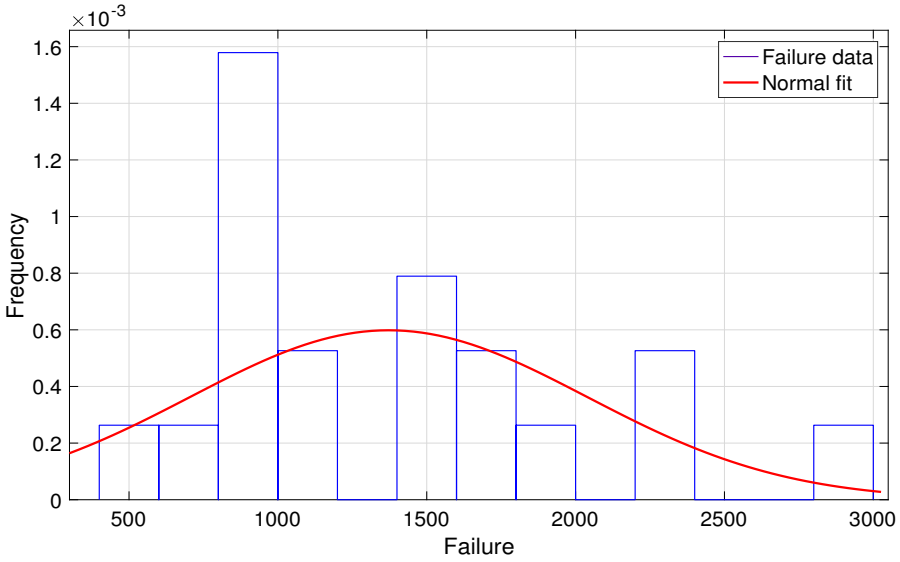

Fig. 2 Failure data Normal distribution test

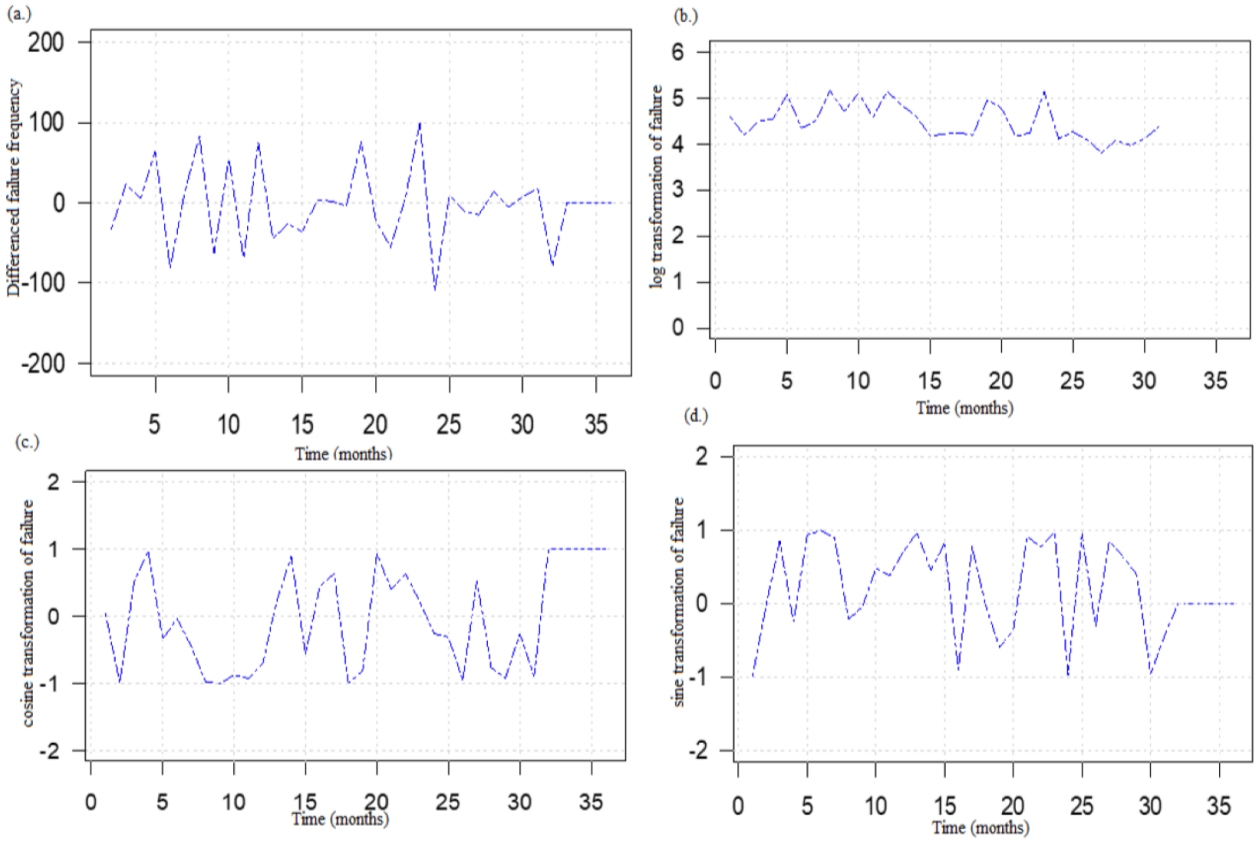

Fig. 3 Differenced and log failure frequency transformation plots

5.3 Discussion 2

In this section we developed some models of our failure data and estimated their prediction accuracy . In other to achieve more concrete estimate of the accuracy of the best model , we first split the loaded dataset into two, 80 % of which we will use to train our models and 20% that we will hold back as a validation dataset. The following steps were followed:

1. Set-up the test harness to use 10-fold cross validation

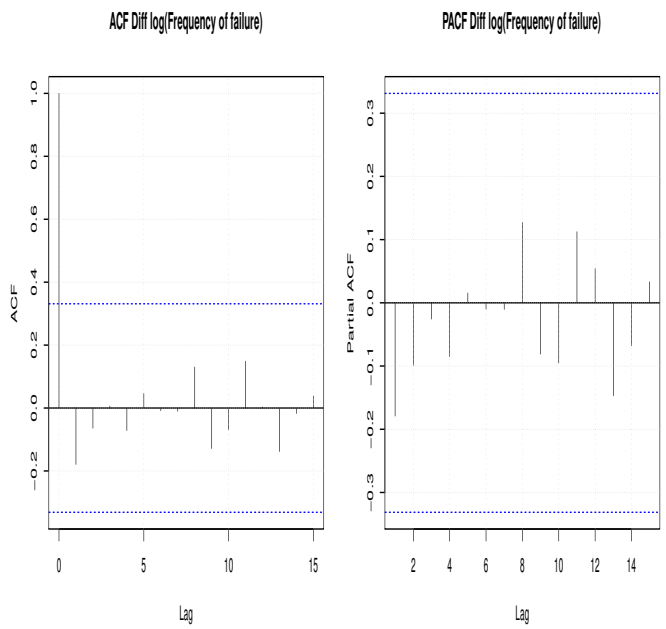

Fig. 4 ACF and PACF diff log function

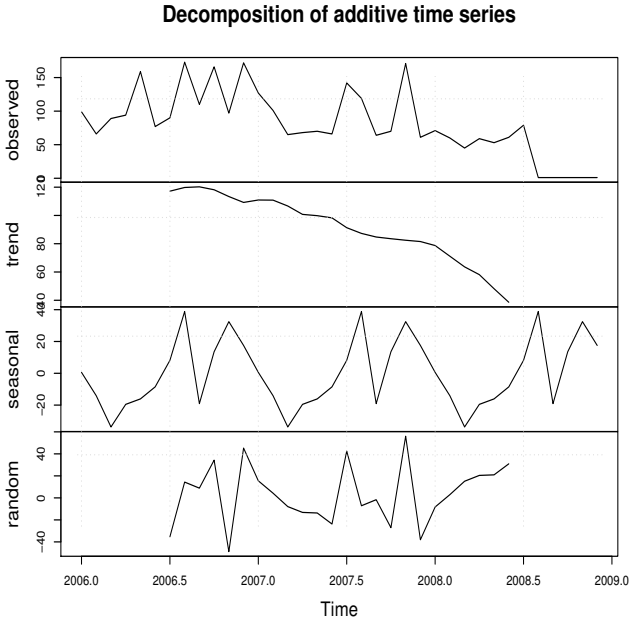

Fig. 6 Decomposition of additive time series plot of failure

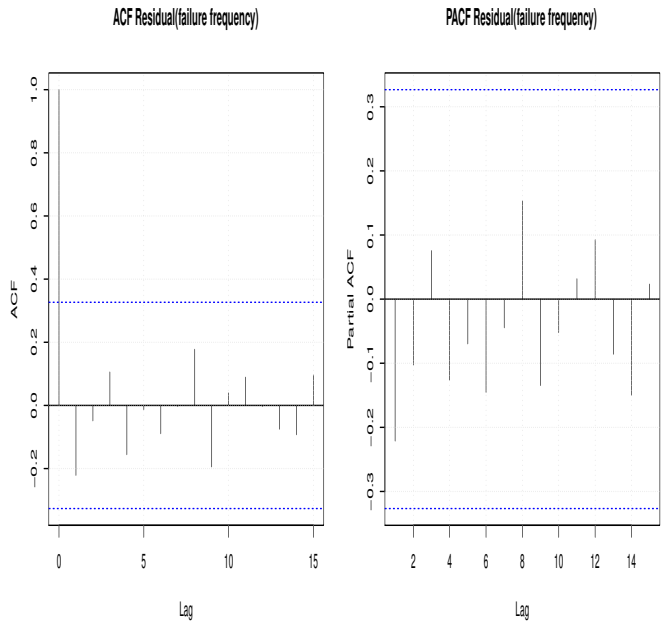

Fig. 5 ACF Residual and PACF residual diff log function

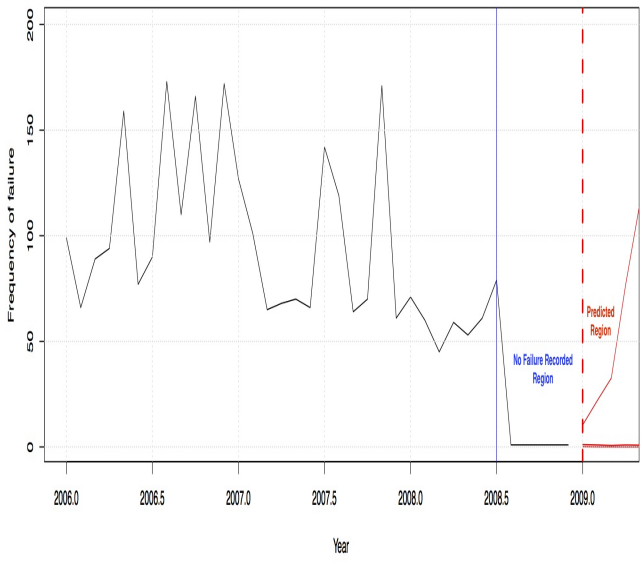

Fig. 7 Predicted region of the failure data

to estimate accuracy.
2. We developed five different models to predict the component failure pattern from the failure data.
3. The best model is selected.

Furthermore, we will split our dataset into 10 parts, train in 9 and test on 1 and release for all combinations of train-test splits. Table 4 shows the summary of each attribute distribution and the failure sources. In an effort to get a more accurate estimation, the process will be repeated 3 times for each algorithm with different splits of the data into 10 groups. We used the metric of accuracy(MOE) to evaluate our model. This is simply defined as the ratio of the number of correctly predicted instances divided by the total number of instances in

the dataset multiplied by 100 to give a percentage of above 90% accuracy.

5.3.1 Model Evaluation and Prediction

In this section, we evaluate our developed models. We need to compare the models to each other and select the most accurate. Based on the results obtained from Fig. 8, Fig. 9 and Fig 10 which shows the density plot and box plot of the failure data by class value and also the failure distribution based on individual components, we now a better insight and idea of how the pattern of how our data looks like, which shows that some of the classes are partially linearly separable in some dimensions. We then reset the random number seed before reach run to ensure that the evaluation of each algorithm is performed using exactly the same data splits. This ensures the results are directly comparable.Here we compare the five models and their respective accuracy estimations. In Table 5 we selected the best three models out of the five while Table 6 presents the accuracy of measurement.

Table 4 Summary of each attribute distribution

FS	Min	1stQu.	Med.	Mean	3rdQu.	Max
HW	1.0	35.0	43.0	46.8	52.7	103.0
SW	2.0	37.0	45.0	48.5	54.0	104.0
HE	3.0	38.0	47.0	50.0	56.0	105.0
NW	4.0	41.0	50.0	51.8	57.0	106.0

Table 5 Comparison analysis between the support vector machines, random forest and linear discriminant analysis by the their ability to predict the actual output variables.

S/N	Output	SVM	RF	LDA
1	Disk	Disk	Dimm	Dimm
2	Disk	Dim	CNTRL	Disk
3	APPL	APPL	APPL	CNTRL
4	Disk	Disk	Dimm	APPL
5	Disk	Dim	CNTRL	Disk
6	APPL	APPL	APP	CNTRL
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
18	Disk	Dim	CNTRL	Dimm
19	APPL	APPL	APPL	CNTRL
20	Disk	Disk	Dimm	APPL

Table 6 Accuracy measurement between the predicted the the actual output variable

Algorithm	Accuracy	Kappa
RF	0.70861	0.54586
LDA	0.83853	0.61902
SVM	0.90761	0.75429
KNN	0.48385	0.39293
CART	0.58825	0.41725

Table 7 Error measurement between the predicted and the actual output variable using five algorithm.

ML Algorithm	RMSE	ROC	Sensitivity
RFs	0.2572	0.8320	0.8360
LDA	0.2980	0.7790	0.7130
SVM	0.1718	0.9370	0.6753
KNN	0.3294	0.6740	0.9841
CART	0.2724	0.45930	0.9253

5.4 Summary of Analysis of our Failure prediction models using ML algorithm

We can infer from Table 6 and Table 7 that the SVM has the highest prediction accuracy (0.90761) and sensitivity(0.6753) , where LDA also performs better than RF with a accuracy (0.83853) and (0.70861) respectively. It is apparent from Table 7 that SVM is the best performed model with a sensitivity of (0.6753) , followed by LDA with a sensitivity of (0.7130) , then RF with sensitivity of (0.8360) and CART and KNN with (0.58825) and (0.48385) respectively. Furthermore, Table 7 also shows the calculated RMSE and ROC , where SVM has it's maximum RMSE value of 0.1718 and LDA,RF,KNN and CART has their different maximum value of RMSE to be (0.2980),(0.2572),(0.3294) and (0.2724) respectively. Overall KNN has the highest RMSE value which indicates it is the lowest performed model and SVM has the lowest RMSE value which shows it is the best performed out of all of them. Overall in all ramification, it is apparent that SVM is our best model. Hence, the results clearly validate that our model using SVM has accurate enough to predict system failures with minimum values of RMSE and Sensitivity.Hence, SVM based model has higher prediction accuracy as compared to other models.

6 Conclusion and Future work

This paper provides an effective approach to failure prediction using time series and machine learning. Our proposed models have been exemplified with the dataset collected from the NERSC with the purpose of provid-

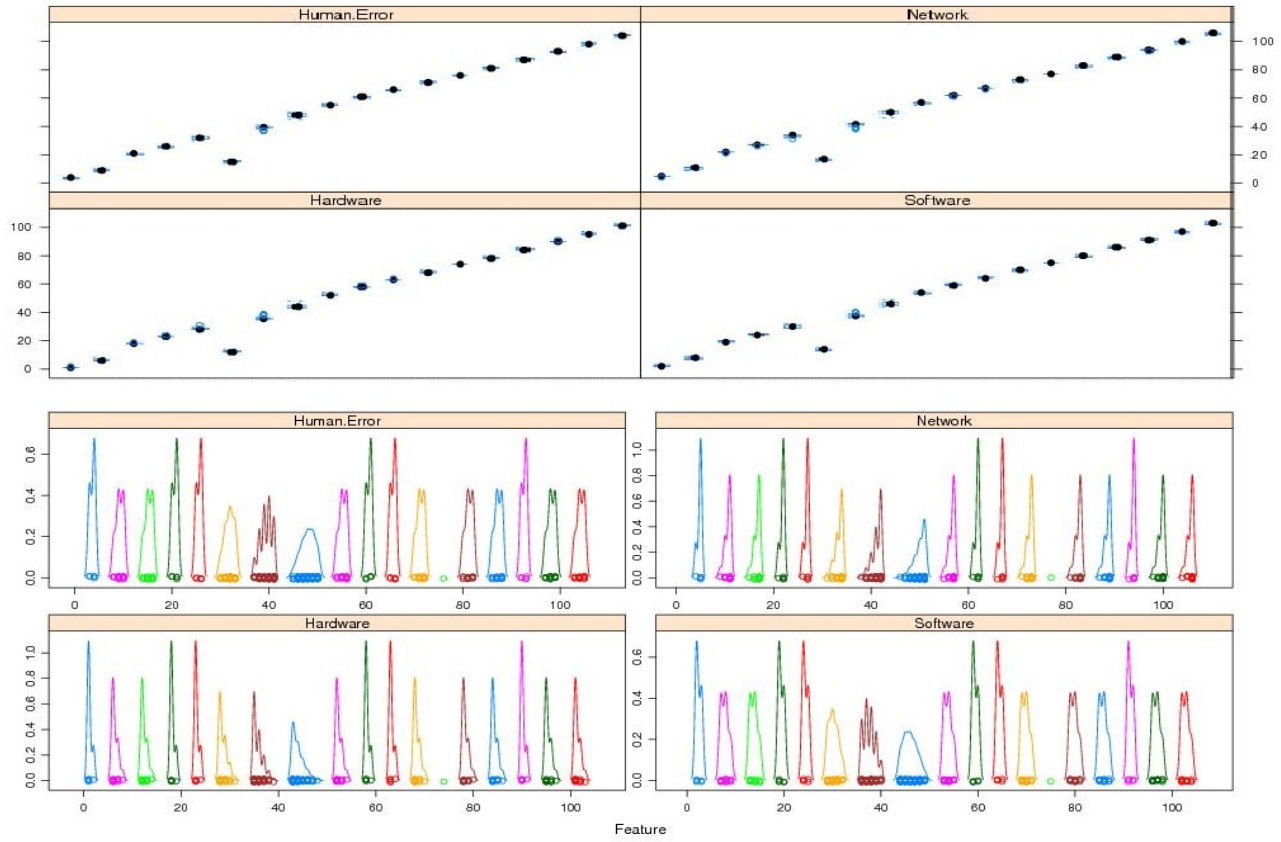

Fig. 8 Density plot and box plot of the failure data by class value

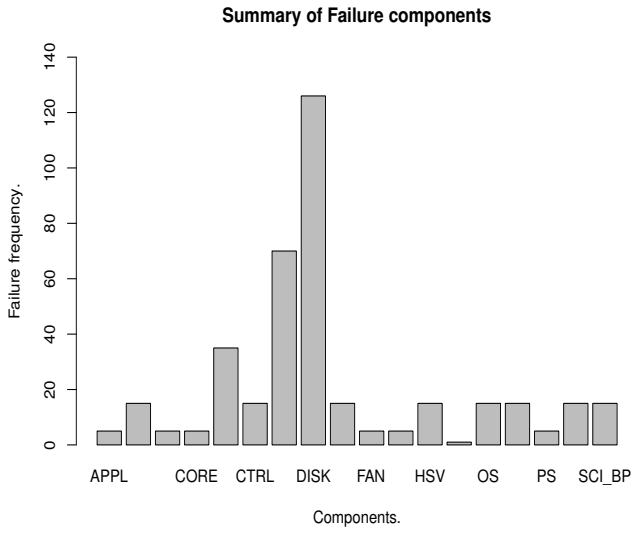

Fig. 9 Distribution of reported system component failure frequency

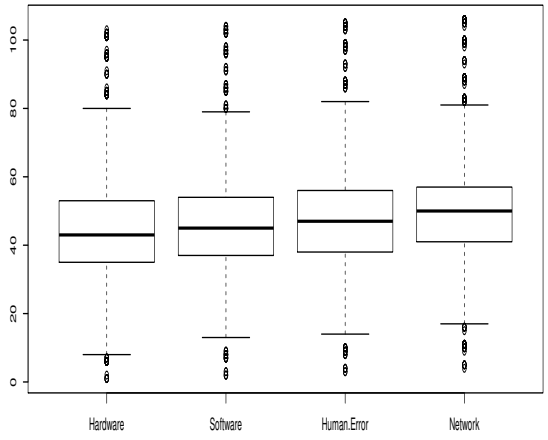

Fig. 10 Box plot of the components failure sources

ing failure specifics for I/O related systems and components. The Data was collected for storage, networking, computational machines, and file systems in production use at NERSC from the 2001-2006 time-frame [37].

The experimental results for failure prediction models have also been evaluated through support vector machine, random forest, k-nearest neighbors, linear discriminant analysis and classification and regression trees. As the aim was to develop a model that can accurately predict possible system and application failure, we first used time-series modeling approach and identified an ARIMA(1,1,1) model for the compounded failure dataset extracted from the Computer Failure Data Repository (CFDR) [38]. Then we applied ML algorithm on our dataset by comparing and fine-tuning our model before selecting the best algorithm. The analysis of our results indicates SVM as the best model in terms of sensitivity, followed by LDA and RF. This was achieved with SVM by comparing the actual system component failures and the predicted failures. Finally, the failure prediction model using support vector machine has shown the effectiveness with average accuracy of (90%) as compared to RF, KNN, CART and LDA. In the future, we suggest selecting the best performed model and further applying model-tuning to achieve a 100% accuracy for predicting failures in a large scale high performance cloud system. We will also aim to demonstrate more clearly the specific application of the results obtained in a cloud,HPC traditional network and containerization approaches such as Docker and Vertex.

Acknowledgement

The authors would like to thank the anonymous reviewers for their useful review in improving the quality of this paper. We would also like to thank Bill Kramer and Akbar Mokhtarani from NERSC for collecting the data and sharing it. One of the authors Bashir Mohammed is a Petroleum Technology Development Fund (PTDF) scholar. We would like to express our sincere gratitude to PTDF for its funding support under the OSS scheme with grant number (PTDF/E/OSS/PHD/MB/651/14).

References

1. O. Beaumont, L. Eyraud-Dubois, and J. A. Lorenzo-Del-Castillo, Analyzing real cluster data for formulating allocation algorithms in cloud platforms, Parallel Comput., vol. 54, pp. 8396, 2016.
2. K. Singh, S. Smallen, S. Tilak, and L. Saul, Failure analysis and prediction for the CIPRES science gateway Kritika, Concurr. Comput. Pract. Exp., vol. 22, no. 6, pp. 685701, 2016.
3. P. Garraghan, P. Townend, and J. Xu, An empirical failure-analysis of a large-scale cloud computing environment, Proc. - 2014 IEEE 15th Int. Symp. High-Assurance Syst. Eng. HASE 2014, pp. 113120, 2014.
4. J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. Ferreira, and C. Engelmann, Combining partial redundancy and checkpointing for HPC, Proc. - Int. Conf. Distrib. Comput. Syst., pp. 615626, 2012.
5. B. Mohammed, M. Kiran, K. M. Maiyama, M. M. Kamala, and I.-U. Awan, Failover strategy for fault tolerance in cloud computing environment, Softw. Pract. Exp., 2017.
6. R. Ghosh, L. Francesco, F. Frattini, S. Russo, and S. T. Kishor, Scalable analytics for IaaS cloud availability, IEEE Trans. Cloud Comput., vol. 2, no. 1, pp. 5770, 2014.
7. T. Chalermarrewong, T. Achalakul, and S. C. W. See, The Design of a Fault Management Framework for Cloud, 2012 9th Int. Conf. Electr. Eng. Comput. Telecommun. Inf. Technol., pp. 14, 2012.
8. A. Elzamly, B. Hussin, A. Samad, H. Basari, and C. Technology, Classification of Critical Cloud Computing Security Issues for Banking Organizations: A cloud Delphi Study, Int. J. Grid Distrib. Comput., vol. 9, no. 8, pp. 137158, 2016.
9. ITProPortal, ITProPortal.com: 24/7 Tech Commentary and Analysis, 2012. [Online]. Available: http://www. itproportal.com/. [Accessed: 24-Jun-2016].
10. Bilal K, Khalid O, Malik SU, Khan MUS, Khan S, Zomaya A. Fault tolerance in the cloud. In Fault Tolerance in the Cloud Encyclopedia on Cloud Computing, vol. 2015. John Wiley and Sons: Hoboken, NJ, USA, 2015: 291300.
11. C. Modi, D. Patel, B. Borisaniya, A. Patel, and M. Rajarajan, A survey on security issues and solutions at different layers of Cloud computing, J. Supercomput., vol. 63, no. 2, pp. 561592, 2013.
12. D. Gnanavelu and D. G. Gunasekaran, Survey on Security Issues and Solutions in Cloud Computing, Int. J. Comput. Trends Technol., vol. 8, no. 8, pp. 126130, 2014.
13. B. Wang, Y. Zheng, W. Lou, and Y. T. Hou, DDoS attack protection in the era of cloud computing and Software-Defined Networking, Comput. Networks, vol. 81, pp. 308319, 2015.
14. Z. Pantic and M. Babar, Guidelines for Building a Private Cloud Infrastructure, ITU Tech. Rep. - TR-2012-153TR-2012-153, 2012.
15. O. Sefraoui, M. Aissaoui, and M. Eleuldj, Cloud computing migration and IT resources rationalization, 2014 Int. Conf. Multimed. Comput. Syst., pp. 11641168, Apr. 2014.
16. A. Sen and S. Madria, Off-Line Risk Assessment of Cloud Service Provider, 2014 IEEE World Congr. Serv., pp. 5865, Jun. 2014.
17. S. Yadav, Comparative Study on Open Source Software for Cloud Computing Platform: Eucalyptus , Openstack and Opennebula, Res. Inven. Int. J. Eng. Sci. Vol.3, Issue 10, vol. 3, no. 10, pp. 5154, 2013.
18. G. Bontempi, S. Ben Taieb, and Y. A. Le Borgne, Machine learning strategies for time series forecasting, Lect. Notes Bus. Inf. Process., vol. 138 LNBIP, pp. 6277, 2013.
19. A. Chigurupati, R. Thibaux, and N. Lassar, Predicting hardware failure using machine learning, 2016 Annu. Reliab. Maintainab. Symp., pp. 16, 2016.
20. E. Fulp, G. Fink, and J. Haack, Predicting Computer System Failures Using Support Vector Machines., Proc. First USENIX Conf. Anal. Syst. logs, pp. 55, 2008.

21. B. Schroeder and G. a Gibson, A Large-Scale Study of Failures in High-Performance Computing Systems, IEEE Trans. Dependable Secur. Comput., vol. 7, no. 4, pp. 337350, 2010.
22. R. K. Sahoo, M. S. Squillante, A. Sivasubramaniam, and Y. Z. Y. Zhang, Failure data analysis of a large-scale heterogeneous server environment, Int. Conf. Dependable Syst. Networks, 2004, pp. 110, 2004.
23. K. V. Vishwanath and N. Nagappan, Characterizing Cloud Computing Hardware Reliability, Proc. 1st ACM Symp. Cloud Comput. - SoCC 10, p. 193, 2010.
24. S. Kavulya, J. Tany, R. Gandhi, and P. Narasimhan, An analysis of traces from a production MapReduce cluster, CCGrid 2010 - 10th IEEE/ACM Int. Conf. Clust. Cloud, Grid Comput., pp. 94103, 2010.
25. A. Abu-Samah, M. K. Shahzad, E. Zamai, and A. Ben Said, Failure prediction methodology for improved proactive maintenance using Bayesian approach, IFAC Proc. Vol., vol. 48, no. 21, pp. 844851, 2015.
26. A. Khan, B. Bussone, J. Richards, and A. Miguel, A practical Approach to Hard Disk Failure Prediction in Cloud Platforms, in 2016 IEEE Second International Conference on Big Data Computing Service and Applications??, 2016, pp. 105116.
27. G. H. Thomas Gentner, Klau p. Gungl, Patent US9319030 - Integrated circuit failure prediction using clock duty cycle recording 2016.
28. S. A. E. Keke Gai, Meikang Qiu, Security-Aware Information Classifications Using Supervised Learning for Cloud-Based Cyber Risk Management in Financial Big Data, in 2016 IEEE 2nd International Conference on Big Data Security on Cloud, IEEE International Conference on High Performance and Smart Computing, IEEE International Conference on Intelligent Data and Security, 2016, pp. 197202.
29. L. Zhang, K. Rao, R. Wang, and Y. Jia, Risk Prediction Model Based on Improved AdaBoost Method for Cloud Users, Open Cybern. Syst. Journal, 2015, vol. 9, pp. 4449, 2015.
30. D. Pop, Machine Learning and Cloud Computing: Survey of Distributed and SaaS Solutions, Inst. e-Austria Timisoara, Tech. Rep 1, 2012.
31. S. Bsch, V. Nissen, and A. Wnscher, Automatic classification of data-warehouse-data for information lifecycle management using machine learning techniques, Inf. Syst. Front., 2016.
32. D. Fall, T. Okuda, Y. Kadobayashi, and S. Yamaguchi, Risk Adaptive Authorization Mechanism (RAdAM) for Cloud Computing, J. Inf. Process., vol. 24, no. 2, pp. 371380, 2016.
33. C. Guo, Y. Liu, and M. Huang, Obtaining Evidence Model of an Expert System Based on Machine Learning in Cloud Environment, J. Internet Technol., vol. 16, no. 7, pp. 13391349, 2015.
34. Z. Amin, N. Sethi, and H. Singh, Review on fault tolerance techniques in cloud computing, Int. J. Comput. Appl., vol. 116, no. 18, pp. 1117, 2015.
35. A. Pellegrini, P. Di Sanzo, and D. R. Avresky, Proactive Cloud Management for Highly Heterogeneous Multi-cloud Infrastructures, in 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2016, pp. 13111318.
36. S. P. P. K.S. Thakur., T. R.Godavarthi., 10.1.1.416.6042, vol. 3, no. 6, pp. 698703, 2013.
37. B. Schroeder and G. Gibson, The computer failure data repository (CFDR), Reliab. Anal. Syst. Fail. Data , no. March, p. 6, 2007.
38. B. Schroeder and G. Gibson, The Computer Failure Data Repository (CFDR): collecting, sharing and analyzing failure data, SC 06 Proc. 2006 ACM/IEEE Conf. Supercomput., no. March, p. 154, 2006.
39. V. N. Vapnik, An Overview of Statistical Learning Theory, IEEE Trans. Neural Networks, vol. 10, no. 5, pp. 988999, 1999
40. M. C. Medeiros, A. Veiga, and M. G. C. Resende, A Combinatorial Approach to Piecewise Linear Time Series Analysis, J. Comput. Graph. Stat., vol. 11, no. 1, pp. 236258, 2002.
41. M. Coombs, A. S. Jarrah, and R. C. Laubenbacher, Foundations of Combinatorial Time Series Analysis, System, pp. 117.
42. Y. Zhou, Failure Trend Analysis Using Time Series Model, 2017 29th Chinese Control and Decision Conference., no. 1, pp. 859862, 2017.
43. S. Ho, M. Xie, and T. Goh, A comparative study of neural network and Box-Jenkins ARIMA modeling in time series prediction, Comput. Ind. Eng., vol. 42, no. 24, pp. 371375, 2002.
44. R. Jayanthi and L. Florence, Software defect prediction techniques using metrics based on neural network classifier, Cluster Comput., pp. 112, 2018.
45. N. Padhy, R. P. Singh, and S. C. Satapathy, Cost-effective and fault-resilient reusability prediction model by using adaptive genetic algorithm based neural network for web-of-service applications, Cluster Comput., vol. 9, pp. 123, 2018.
46. K. Kumaresan and P. Ganeshkumar, Software reliability modeling using increased failure interval
47. C. Manjula and L. Florence, Deep neural network based hybrid approach for software defect prediction using software metrics, Cluster Comput., pp. 117, 2018.
48. Z. Li, An adaptive overload threshold selection process using Markov decision processes of virtual machine in cloud data center, Cluster Comput.,
49. C. Shen, W. Tong, K. K. R. Choo, and S. Kausar, Performance prediction of parallel computing models to analyze cloud-based big data applications, Cluster Comput., pp. 116, 2017.
50. J. Choi and Y. Kim, Adaptive resource provisioning method using application-aware machine learning based on job history in heterogeneous infrastructures, Cluster Comput., vol. 20, no. 4, pp. 35373549, 2017.
51. D. Kwon, H. Kim, J. Kim, S. C. Suh, I. Kim, and K. J. Kim, A survey of deep learning-based network anomaly detection, Cluster Comput., pp. 113, 2017.
52. D. Muthusankar, B. Kalaavathi, and P. Kaladevi, High performance feature selection algorithms using filter method for cloud-based recommendation system, Cluster Comput., vol. 0, no. i, pp. 112, 2018.
53. S. H. H. Madni, M. S. A. Latiff, Y. Coulibaly, and S. M. Abdulhamid, Recent advancements in resource allocation techniques for cloud computing environment: a systematic review, Cluster Comput., vol. 20, no. 3, pp. 24892533, 2017.
54. E. Casalicchio, A study on performance measures for auto-scaling CPU-intensive containerized applications, Cluster Comput., vol. 1, 2019.
55. L. Nussbaum, F. Anhalt, O. Mornard, J. Gelas, L. Nussbaum, F. Anhalt, O. Mornard, J. G. Linux-based, L. Nussbaum, and O. Mornard, Linux-based virtualization for HPC clusters To cite this version: Linux-based virtualization for HPC clusters, Montr. Linux Symp., 2009.
56. L. Benedicic, F. A. Cruz, A. Madonna, and K. Mariotti, Portable, high-performance containers for HPC, Cornell Univ., 2017.

57. S. Nanda and T. J. Hacker, Racc: Resource-Aware Container Consolidation using a Deep Learning Approach, Proc. First Work. Mach. Learn. Comput. Syst. - MLCS18, no. May, pp. 15, 2018.
58. CANONICAL LTD, Linux Containers, Infrastructure for container projects, 2018. [Online]. Available: https://linuxcontainers.org/. [Accessed: 21-Jan-2019].
59. T. Dwyer, A. Fedorova, S. Blagodurov, M. Roth, F. Gaud, and J. Pei, A practical method for estimating performance degradation on multicore processors, and its application to HPC workloads, Int. Conf. High Perform. Comput. Networking, Storage Anal. SC, 2012.
60. R. Buyya, R. Ranjan, and R. N. Calheiros, Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities, Proc. 2009 Int. Conf. High Perform. Comput. Simulation, HPCS 2009, pp. 111, 2009.
61. C. Manjula and L. Florence, Deep neural network based hybrid approach for software defect prediction using software metrics, Cluster Comput., pp. 117, 2018.
62. D. Kwon, H. Kim, J. Kim, S. C. Suh, I. Kim, and K. J. Kim, A survey of deep learning-based network anomaly detection, Cluster Comput., pp. 113, 2017.
63. Adeesh Fulay, Database Containerization Platform Checklist - Container Journal, CONTAINER JOURNAL, 2016. [Online]. Available: https://containerjournal.com/2016/09/19/1860/. [Accessed: 21-Jan-2019].
64. C. Onur, Utilizing Containers for HPC and Deep Learning Workloads — CIO, DELL EMC: INNOVATING TO TRANSFORM, 2018. [Online]. Available: https://www.cio.com/article/3269351/analytics/utilizing-containers-for-hpc-and-deep-learning-workloads.html. [Accessed: 21-Jan-2019].