Article

A combined metrics approach to cloud service reliability using artificial intelligence

Tek Raj Chhetri [1], Chinmaya Kumar Dehury [2,*], Artjom Lind [2], Satish Narayana Srirama [3] and Anna Fensel [1,4]

[1] Semantic Technology Institute (STI) Innsbruck, Department of Computer Science, University of Innsbruck, Innsbruck, Austria; tekraj.chhetri@sti2.at,anna.fensel@sti2.at
[2] Institute of Computer Science, University of Tartu, Tartu 50090, Estonia; chinmaya.dehury@ut.ee, artjom.lind@ut.ee
[3] School of Computer and Information Sciences, University of Hyderabad, Hyderabad 500046, India; satish.srirama@uohyd.ac.in
[4] Wageningen University & Research, Wageningen, The Netherlands
* Corresponding author.

Abstract: Identifying and anticipating potential failures in the cloud is an effective method for increasing cloud reliability and proactive failure management. Many studies have been conducted to predict potential failure, but none have combined SMART (Self-Monitoring, Analysis, and Reporting Technology) hard drive metrics with other system metrics such as CPU utilisation. Therefore, we propose a combined metrics approach for failure prediction based on Artificial Intelligence to improve reliability. We tested over 100 cloud servers' data and four AI algorithms: Random Forest, Gradient Boosting, Long-Short-Term Memory, and Gated Recurrent Unit. Our experimental result shows the benefits of combining metrics, outperforming state-of-the-art.

Keywords: Failure Prediction, Fault-tolerance, Cloud Computing, Artificial Intelligence, Reliability

1. Introduction

Cloud computing which has emerged as the fifth utility over the decade, is a backbone to the modern economy [1]. It is a model of computing that allows the flexible use of virtual servers, massive scalability, and management services for the delivery of information services [2]. With the low-cost pay-per-use model of on-demand computing [3], the cloud has grown massively over the years, both in terms of size and complexity.

Today, almost everyone is connected to the cloud in one way or the other because of cost-effectiveness with pay-as-you-go or subscription-based service model for on-demand access to IT resources [1], [3]. Industries rely on the cloud for its operations, academicians to accelerate and conduct scientific experiments, and ordinary end-users by using cloud-based services knowingly or unknowingly like Google Drive, Gmail, Outlook, and so on. Furthermore, the cloud today is more important than yesterday, as it supports smart city construction [4], smart manufacturing [5], enterprise business [6], scalable data analysis [7], [8], healthcare [9], [10] and also new evolving computing paradigm like Fog and Edge computing [11].

To date, despite the significant improvement in the performance of the hardware elements of the cloud infrastructure, the failure rate remains substantial. Moreover, the cloud is not as reliable as the cloud service provider claimed, which is more than 99.9% [12]. There have been multiple instances of failure reported, such as Amazon's cloud data servers went down, collapsing Reddit, Airbnb, and Flipboard in early October 2012, loss of Amazon AWS S3 on February 28, 2017, crashing of Microsoft cloud services on March 22, 2017 [12]. Such failures show that cloud service providers are not as reliable as they claim.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 November 2021 doi:10.20944/preprints202111.0548.v1

The public cloud vendor revenue is forecasted to be around 500 billion by 2026[1]. The majority of this revenue goes to platform-as-a-service (PaaS) and infrastructure-as-a-service (IaaS) 298.4 and 126 Billion, respectively. Any occurrence of the cloud's failure impacts the cloud-based environment and services it supports, its users, and the economy. Therefore, maintaining reliability is essential, and failure prediction is one of the mechanisms to obtain it. In this study, taking advantage of the advancement of artificial intelligence (AI), we focus on failure prediction based on artificial intelligence techniques Random Forest (RF), Gradient Boosting (GB), Long-Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU).

AI has the ability to learn patterns and make future predictions accordingly. AI can be manifested as machine exhibiting human intelligence [13] and is utilised in diverse domains such as healthcare, autonomous systems, in monitoring applications, predictive maintenance because it allows solving problems that before seemed to be unsolvable by computational processes alone[14]. The tremendous advancement in artificial intelligence today has resulted in state-of-the-art performance for many practical problems, especially in areas involving high-dimensional unstructured data such as computer vision, speech, and natural language processing [15]. This ability of AI to make future predictions based on learned patterns and advancement, we make use of it in our study.

1.1. Motivation

Today, the majority of businesses rely on cloud services to run their daily operations. Any failure of cloud services directly impacts the business, and repeated failures result in reputation damage. Reputation is an intangible asset that accounts for 85% of the value of a business [16], [17]. Significant effort has been made to increase the reliability of cloud services. For failure prediction, the studies (see Section 2) either use hard drive SMART (Self-Monitoring, Analysis and Reporting Technology) metrics or other system metrics such as CPU and memory utilisation. However, despite the demonstrated ability of SMART hard drive metrics and other system metrics such as CPU and memory utilisation to predict failures, to our knowledge, no study has combined both metrics. We hypothesised that combining both metrics will give us more information and, as a result, can help improve reliability further. The identified existing research gap on the combined use of SMART metrics and other system metrics, which we in our study refer to as *combined metrics*, to improve reliability is the main motivation for this study. The other motivation includes the limited number of studies in the direction of server failure prediction, the use of a traditional rule-based tool such as Prometheus [18].

1.2. Goal

The ability to accurately predict failure is an essential factor for reliable performance. This is because it allows us to take action that is proactive. The reliability of a process depends on how well we predict failures. The aim of this study is to improve the reliability of cloud services by improving cloud server failure prediction using selected AI techniques and the combined metrics approach.

1.3. Contributions

Based on our motivation and goal to achieve, the main contributions of our study can be summarised as follows.

- A novel approach to server failure prediction based on combined metrics.
- Use of AI approach to overcome dis-advantage rule-based failure prediction.
- A comprehensive evaluation of multiple artificial intelligence techniques like RF, GB, LSTM and GRU with the use of real data from more than 100 cloud servers.

[1] https://wikibon.com/wp-content/uploads/Wikibon-BGracely-Cloud-Computing-Nov-20152.pdf

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 November 2021 doi:10.20944/preprints202111.0548.v1

The rest of the paper is organised as follows. In Section 2, we present review of the state-of-the-art. In Section 3, we present our methodology and in Section 4 we present the performance evaluation. Finally, we conclude the paper in Section 5.

2. State-of-the-art

In this section, we present a brief survey of recent works in failure prediction in the domain of cloud computing.

2.1. Server-level Failure Prediction

Mohammed et al. [19] conducted a study on high-performance computing (HPC) system failure prediction using machine learning algorithms and evaluated on accuracy. The study makes use of hardware, software, human error, network and undetermined as a failure source. Using such, we cannot say if the system failed actually or there was human intervention. Further, it is not clear the scope of the undetermined error source.

Xu et al. [20] predicted the disk error using a ranking based machine learning approach. The early prediction allowed migration of a virtual machine (VM) to a healthy node, thereby improving the service availability of Microsoft Azure. However, this study is limited to only using disk information.

Lai et al. [21] using machine learning conducted a study to predict server failure within 60 days using hard drive data collected from SLAC Accelerator Laboratory[2] and maintained failure logs over 10 years. The authors also introduced a derived metric *time_since_prev_failure* that represents the time since the previous failure. This study is only based on hard disk data.

Das et al. [22] using long short term memory (LSTM) predicted failing nodes to migrate computation to healthy nodes. This study makes use of the data from Cray XC30, Cray XE6, Cray XC40, Cray XC40/XC30 and error phrase form the system log.

The study by Chigurupati et al. [23] focuses on predicting communication hardware failure 5 minutes ahead using machine learning. While the study by Tehrani et al. [24] using a support vector machine (SVM) makes failure prediction. This study uses metrics temperature, CPU, RAM, and bandwidth utilization and lacks the benefit of other attributes like hard drive information as in our study.

Adamu et al. [25] conducted the study using SVM and data collected from Computer Failure Data Repository (CFDR) of the National Energy Research Scientific Computing Center (NERSC)[3]. The author presents the prediction of a disk, a dual in-line memory module (DIMM), CPU, and other failures separately. However, it is not clear the scope of failure, and also the network information is not used, which is also one of the reasons for failure.

2.2. VM-level Failure Prediction

Meenakumari et al. [26] conducted a study for earlier VM failure prediction using dynamic threshold approach. The study uses metrics CPU utilization, CPU usage, bandwidth, temperature and memory but no machine learning approach as our study.

Alkasem et al. [27] conducted a study on VM failure to start-up using machine learning and Apache Spark[4] streaming. The study, however, only makes use of the metrics CPU Utilization, Memory Usage, Network Overhead, IO Storage usage.

Qasem et al. [28] conducted a VM failure study using simulated data from cloudsim and neural network approach. Similar to Qasem et al., Liu et al. [29] conducted a study using a recurrent neural network (RNN) and hard drive SMART metrics.

[2] https://www6.slac.stanford.edu
[3] https://www.nersc.gov
[4] https://spark.apache.org

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 November 2021 doi:10.20944/preprints202111.0548.v1

2.3. Task-level Failure Prediction

Shetty et al. [30] conducted a study on task failure prediction using Google cluster dataset and XGboost classifier. Similarly, Jassas et al. [31] also uses the Google cluster trace dataset for job failure in the cloud computing environment. These studies, however, only focus on task failure prediction.

Bala et al. [32] using Naive Bayes, Random Forest, Logistic Regression, and Artificial Neural Networks (ANN) conducted a task failure prediction for scientific workflow applications. Similar to previous studies by Shetty et al. [30], Jassas et al. [31], this study also focuses on task failure prediction.

Rosa et al. [33] conducted a study on job failure prediction using metrics like CPU utilization, Memory utilization, Disk utilization, and data from Google cluster traces. The study makes use of a Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), and LR. Similar to other task failure studies, it is also focused on the failure of the task only.

Similar to previous studies, Gao et al. [34] also uses Google cluster trace, and a multi-layer Bidirectional Long Short Term Memory (Bi-LSTM) to identify tasks and job failures in the cloud. This study makes use of the deep learning approach but is focused on task and job failure.

3. Materials and Methods

This section details our approach. Section 3.1 contains information about data collection, Section 3.2 contains information about the metrics used in our study, and Section 3.3 contains information about data preprocessing. Similarly, Sections 3.4, 3.5, and 3.6 discuss our implementation of random forest, gradient boosting, and LSTM and GRU algorithms. Figure 1 shows our methodology.

3.1. Data Collection

There are various datasets available, including Google job failure dataset[5], SMART hard drive dataset[6]. However, these datasets either contain information for job failure or hard drive failure and lack information on other system metrics such as CPU utilisation for failure, necessitating data collection. Our data collection step entails downloading data from Prometheus[7]-monitored University of Tartu High-Performance Centre [35] cloud servers. Fig. 2 depicts the data collection process. A Python script converts the downloaded JSON[8] data to CSV (Comma-separated values).

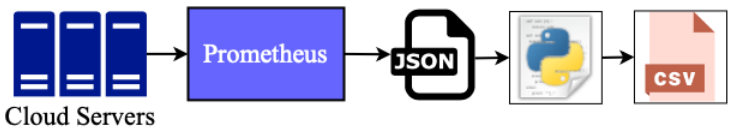


Figure 2. Data Collection

3.2. Metrics Selection

Prometheus logs a wide range of system information. All the data collected by Prometheus might not be useful. Random selection of metrics will not guarantee a precise result. Therefore, a careful selection of metrics should be made. We selected metrics shown in Table 1 by performing a detailed assessment of the available metrics and in-depth analysis of their effect on the system. The only selected metrics were downloaded

[5] https://github.com/google/cluster-data
[6] https://www.backblaze.com/b2/hard-drive-test-data.html
[7] https://prometheus.io/docs/introduction/overview/
[8] https://www.json.org/json-en.html

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 November 2021 doi:10.20944/preprints202111.0548.v1

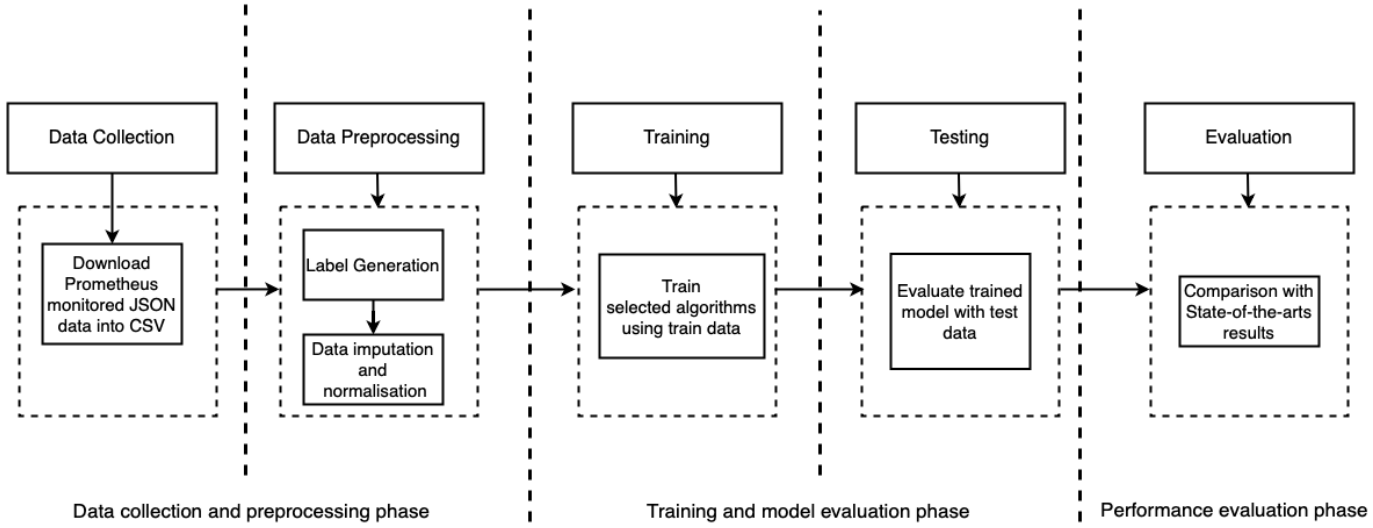


Figure 1. Methodology

SN	Metrics	Description
1	CPU Utilisation	Host CPU usage in %.
2	Memory Utilisation	Memory usage in bytes
3	Network Overhead	Network usage in bytes
4	IO Utilisation	IO usage in time
5	Bits Read	Data written out from disk in bytes
6	Bits Write	Data written into disk in bytes
7	Smart 188	Command time out
8	Smart 197	Current pending sector count
9	Smart 198	Uncorrectable sector count
10	Smart 9	Power-on hours
11	Smart 1	Read error Rate
12	Smart 5	Reallocated sectors count
13	Smart 187	Reported uncorrectable errors
14	Smart 7	Seek error rate
15	Smart 3	Spin up time
16	Smart 4	Start/stop count
17	Smart 194	Temperature
18	Smart 199	UltraDMA CRC error count

Table 1: Selected Metrics (Features)

during the data collection phase. The downloaded data is then preprocessed, which is discussed in Section 3.3.

3.3. Data Preprocessing

Machine learning and deep learning algorithms are highly dependent on data to function properly. The quality of the data affects the accuracy of the algorithm. But the real-world data is often noisy, inconsistent and incomplete. In order to improve the result, better data is needed, or the quality of data has to be improved. Therefore, in an attempt to improve the quality of the data, we preprocess the data. To compensate for the missing failure information, we performed label generation as the first data preprocessing prior to any other preprocessing, as Das et al. [22], Jassas et al. [31], and Qasem et al. [36]. Algorithm 1 was utilised to generate the target label. Table 2 shows the threshold value in our target label generation algorithm. The threshold was

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 November 2021 doi:10.20944/preprints202111.0548.v1

defined using information available from hardware manufacturers as well as findings from the state-of-the-art. As shown in Algorithm 1, when the value of the metrics is within the threshold, we represent it as 0; otherwise, we represent it as 1. The 0 denotes that there is no failure, whereas the 1 denotes that there is a failure. The value 0 or 1 is then appended to the list *targetValue* via the *AppendTotargetValue*. The *targetValue* is then used to create a new *target* column, which is then combined with existing data columns and returned. Further, the preprocessing techniques that handle missing values, non-standard values were employed. Missing values and non-uniform values were handled by applying the scikit-learn[9] preprocessing module.

Algorithm 1: Label generator

Data: ipData ← input data without target label in dictionary format
Result: Data with target label
data ← ipData;
targetValue ← Initialize with an empty List;
resultdata ← Copy of data;
N ← len(data);
for *i=0 to N* **do**
  if *selected metrics within defined threshold* **then**
    AppendTotargetValue(0)
  else
    AppendTotargetValue(1)
  end
end
resultdata ← add targetValue to new target column;
return resultdata;

3.4. Random Forest

Random Forest (RF) is an ensemble learning method and learns using the randomised decision tree. Breiman [37] defines random forest as a classifier consisting of a collection of tree-structured classifiers $\{h(x, \theta_k), k = 1, ...\}$ where the $\{\theta_k\}$ are independent identically distributed random vectors, and each tree casts a unit vote for the most popular class at input x. The best split is chosen by optimising the Classification and Regression Trees (CART) split criterion, which is based on the Gini impurity for classification and prediction squared error for regression [38], [39].

RF, due to its ensemble nature, provides higher accuracy and is also robust against overfitting. Furthermore, RF is capable of dealing with higher-dimensional data. RF is one of the robust general-purpose algorithms. Studies such as [38], [40], [41] and [42] have demonstrated the robustness of RF in different domains. We chose RF in our study because of its robustness against overfitting, outliers, and ability to produce better results.

In our study, we used the scikit-learn library to implement the RF algorithm[10]. The scikit-learn implementation, on the other hand, differs little from Breiman's [37] original RF. Instead of allowing each classifier to vote for a single class, the scikit-learn implementation of RF combines classifiers by averaging their probabilistic predictions[11].

Furthermore, the scikit-learn GridSearchCV[12] was used to optimise hyperparameters, which also uses the K-Fold cross-validation technique to control overfitting. Data is

[9] https://scikit-learn.org/stable/
[10] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[11] https://scikit-learn.org/stable/modules/ensemble.html#forest
[12] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 November 2021 doi:10.20944/preprints202111.0548.v1

SN	Metrics	Threshold
1	CPU Utilisation	> 101
2	Memory Utilisation	> 1000000000
3	Network Overhead	> 5000000000
4	IO Utilisation	> 16089590032253278.0
5	Bits Read	> 38775016960.0
6	Bits Write	> 3189693620224.0
7	Smart 188	< 10
8	Smart 197	< 10
9	Smart 198	< 10
10	Smart 9	< 10
11	Smart 1	< 51
12	Smart 5	< 10
13	Smart 187	< 10
14	Smart 7	< 10
15	Smart 3	< 21
16	Smart 4	< 10
17	Smart 194	> 96
18	Smart 199	< 10

Table 2: Metrics and threshold used in algorithm

divided into k disjoint folds of approximately equal size in K-Fold cross-validation, with each fold used once as validation and the remaining k-1 fold as training.[43], [44], [44]. In our RF experiment, a value of 3 for K-fold cross-validation yielded the best result. Table 3 displays the selected hyperparameter and its value. Similarly, Table 4 displays the optimised values for the selected RF hyperparameters after experimenting.

Parameter Name	Value
n_estimators	[100,200,300,500]
max_features	['auto','log2','sqrt']
criterion	['gini','entropy']
min_samples_split	[3,5,8,9,10,30,50]

Table 3: Random Forest Hyperparameter

3.5. Gradient Boosting

Gradient Boosting (GB) is another tree-based ensemble method that we used in our research due to its superior performance in a variety of domains, including medicine for predicting RNA protein interactions, flight delays, and sentiment analysis [45], [46], [47], [48].

GB applies the boosting principle by shifting the focus to problematic observations that were difficult to predict in previous iterations and executing an ensemble of weak learners, typically decision trees[46], [48]. The GB model is built iteratively, with each new model relying on the previous one. Figure 3 depicts a visualisation of the GB algorithm, learning a weak learner. The GB algorithm consists of three major components: (i) a loss function, (ii) a weak learner, and (iii) an additive model [48]. The loss function optimises the loss, which is deviance by default in scikit-learn, also known as negative

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 November 2021 doi:10.20944/preprints202111.0548.v1

Parameter Name	Value
n_estimators	300
max_features	'auto'
criterion	'gini'
min_samples_split	8

Table 4: Tuned Random Forest Hyperparameter

log-likelihood loss[13] [14]. For making predictions, the decision tree is used as a weak learner. The additive nature of the algorithm adds trees sequentially on each iteration, minimising loss.

In our study, we use the scikit-learn library to implement GB, as we did in RF. GridSearchCV was used to optimise hyperparameters in the same way that RF was used. Table 5 displays the selected hyperparameters for the GB algorithm, while Table 6 displays the value of the optimised hyperparameter. Interestingly, similar to RF, we achieve the best results for GB using K fold cross-validation with k equals 3.

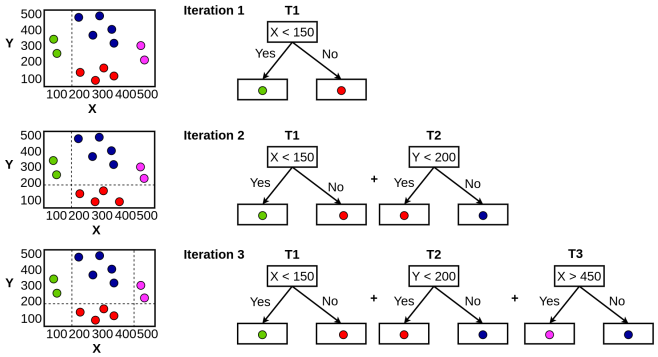


Figure 3. Visualisation of gradient boosting algorithm

Parameter Name	Value
loss	['exponential','deviance']
learning_rate	[0.001,0.01,0.0001]
max_features	[2, 3,5,7]
min_samples_split	[3, 4, 5,7,9]
n_estimators	[100,200,300,500]

Table 5: Gradient Boosting Hyperparameter

[13] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
[14] https://scikit-learn.org/stable/modules/ensemble.html

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 November 2021 doi:10.20944/preprints202111.0548.v1

Parameter Name	Value
loss	deviance
learning_rate	0.01
max_features	5
min_samples_split	3
n_estimators	500

Table 6: Tuned Gradient Boosting Hyperparameter

3.6. Recurrent Neural Network

Over the years, deep learning technology has advanced significantly, outperforming cutting-edge machine learning techniques. Deep learning is now used to solve complex tasks in areas such as computer vision, autonomous systems, and climate analysis due to its performance [49]. However, the neural networks assume data independence and break with sequential data [50]. The reason being that the standard neural network cannot account for temporal conditions and thus cannot make accurate predictions in cases of sequential data such as weather forecasting, a time-series event [51]. Recurrent neural network (RNN), on the other hand, incorporates a new design architecture with hidden state or memory to account for missing dependency in standard neural networks.

Figure 4 illustrates the RNN structure. Similarly, Equations 1 and 2 mathematically represent RNN, where U is the input weight, W is the recurrence weight, and V is the output weight. In equations, h_t and x_t represent the hidden vector h and the input x at time t. Tanh is a nonlinear activation function that aids in overcoming vanishing gradients, and y_t is the output vector obtained by using the softmax activation function. The RNN, however, still suffers from the problem of vanishing and exploding gradients as the sequential dependency grows larger. RNN is only concerned with learning and not with selective forgetting [51]. The vanishing and exploding gradient issues occur when $|W| < 1$ and $|W| > 1$ or eigenvalue $\rho < 1$ and $\rho > 1$ for the scalar and matrix representations of the weight W, respectively [52]. The exploding gradient problem occurs when the accumulation of the gradient becomes so large that it exceeds the range. As a result, the weights become NaN and can no longer be updated. And, as the gradient decays over time and becomes very small, it is overshadowed by the most recent gradient, rendering it unable to look back and remember the past efficiently.

Long short-term memory (LSTM) and Gated Recurrent Unit (GRU) are RNN-based specialised techniques specifically designed to address the RNN problem. We use LSTM and GRU in our study, and the details of how we used LSTM and GRU are discussed in Sections 3.6.1 and 3.6.2 respectively.

$$h_t = tanh(W \times h_{t-1} + U \times x_t + b_h) \tag{1}$$

$$y_t = softmax(V \times h_t + b_v) \tag{2}$$

3.6.1. LSTM

Figure 5 depicts the architecture of the LSTM, and Equations 3 - 8, the computation steps involved in the LSTM [53]. The C_t in Figure 5 represents the cell state, which is an additional computational unit that LSTM uses to solve the RNN problem. The other gates are input gate, output gate, and forget gate, denoted by i_t, o_t and f_t, respectively. The t represents the time.

The forget gate is used to forget information, deciding whether to keep or remove information from the cell state based on the activation function sigmoid value. Tanh, on

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 November 2021 doi:10.20944/preprints202111.0548.v1

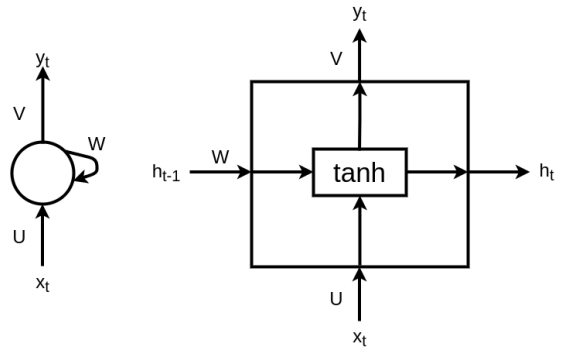


Figure 4. Recurrent Neural Network

the other hand, is an activation function that is used to add non-linearly. The output gate, which is also controlled by the sigmoid function, determines which value to take from the cell state and output. Similarly, the input gate is used to update the cell state with the new value.

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \tag{3}$$

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \tag{4}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \tag{5}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{6}$$

$$h_t = o_t \odot tanh(C_t) \tag{7}$$

$$Output(y_t) = softmax(Uh_t) \tag{8}$$

where in the Equations 3 - 8, x denotes input, h the hidden state, W the weights, $x*$ ($*$ denotes f,i,o) for weight W is the input-to-hidden layer, $h*$ for weight W is the hidden-to-hidden layer, U is the hidden-to-output layer weights.

Tensorflow[15] is used to implement LSTM in our study. The implemented LSTM architecture consists of 8 LSTM layers and 6 Dense layers, followed by one input and one output layer. The LSTM layer has 2048 hidden units, while the Dense layer has 1024 units. The activation function ReLU, a dropout layer with a dropout of 0.5, and L2 regularisation for the kernel regulariser are all included in each LSTM layer. The dropout, which functions as a switch, turns off the neurons based on the provided dropout value, effectively controlling overfitting [54]. On the other hand, regularisation, such as L2 regularisation, controls the overfitting issue by penalising the model. Furthermore, we used additional overfitting control measures, such as early stopping. The use of an additional overfitting measure was driven by the observed high overfitting during the experiment. Early stopping monitors the specific metrics that have been specified to monitor and terminates training when the model begins to overfit. In our experiment, we observed validation loss.

The learning rate is another critical parameter in deep learning algorithms. The learning rate indicates how far the algorithm should progress in the learning process. The learning rate is important because the model's accuracy is determined by how well it learns. If the learning rate is too low, the learning process will become stuck in the local minima and diverge if it is too high, both of which we want to avoid. The learning rate can be set either statically or dynamically. A static learning rate value, on the other hand, would not be adaptable to changing circumstances such as as increasing or decreasing

[15] https://www.tensorflow.org

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 November 2021 doi:10.20944/preprints202111.0548.v1

loss. As a result, in our study, we chose an adaptive learning rate to make the learning process more dynamic. We also included a learning rate scheduler to make it more dynamic. Tensorflow's InverseTimeDecay[16] learning rate scheduler was used, with an initial learning rate of 0.001, decay steps of 2000, the decay rate of 1, and staircase False.

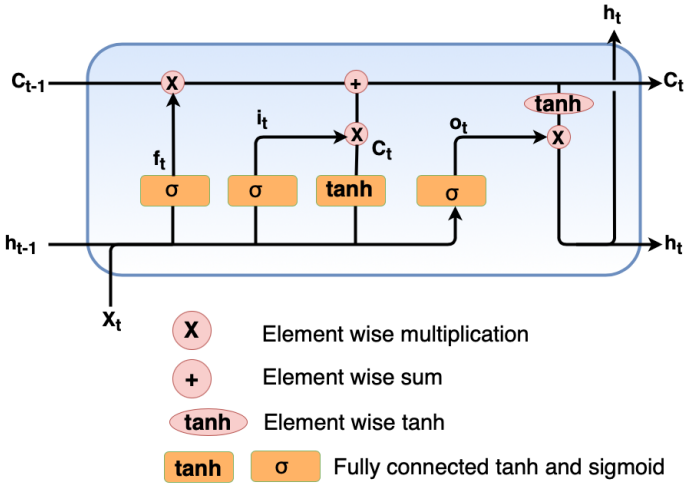


Figure 5. LSTM

3.6.2. GRU

GRU is another RNN-based model that is designed to address the long-term dependency issues of RNN. GRU, like LSTM, adds new gates; however, unlike LSTM, GRU only has a reset gate r_t, an update gate z_t, and one hidden state h_t. The t in reset gate, update gate, and hidden state represents time. The GRU has fewer gates and is faster than LSTM, and is often referred to as the simplified version of LSTM. Figure 6 depicts the architecture of the GRU cell, and Equations 10 - 12 represent the computational steps involved in GRU [53].

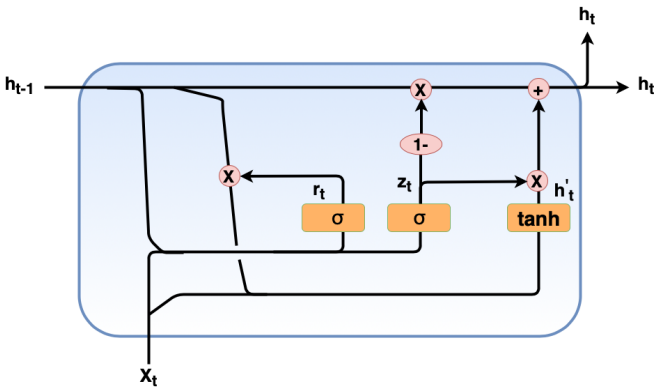


Figure 6. GRU

[16] https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/schedules/InverseTimeDecay

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 November 2021 doi:10.20944/preprints202111.0548.v1

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \tag{9}$$

$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \tag{10}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h) \tag{11}$$

$$Output(y_t) = softmax(Uh_t) \tag{12}$$

where in the Equations 10 - 12, x denotes input, h denotes the hidden state, and o denotes output, W the weights, b is the bias, $x*$ ($*$ represents z,r) for weight W is the input-to-hidden layer, $h*$ for weight W is the hidden-to-hidden layer, $\odot$ is element wise multiplication, U is the hidden-to-output layer weights.

The reset gate r, which is similar to the LSTM forget gate, decides what to keep from the previous layer, and the update gate z decides what to move on to the next step. The reset gate and the update gate decisions are based on the sigmoid function value, as shown in Equations 9 and 10, respectively. The GRU performs the same function as the LSTM, and the reason for considering GRU is its comparable performance to the LSTM [50], [51]. As a result, our GRU implementation has the same architecture as LSTM, with the exception that the LSTM layer has been replaced by the GRU layer.

4. Performance Evaluation

This section details our experiment, describing how the experiment was conducted and details how the experiment was evaluated. Section 4.1 contains the details for experimental setup and the conduct of the experiment in Section 4.3, the evaluation metrics that will be used in Section 4.2 and experimental results in Section 4.4.

4.1. System Setup

The information in this section provides a complete breakdown of our system and the software used for conducting an experiment.

Computation-intensive tasks like training of machine learning algorithms require a lot of processing power. Due to this machine learning training is almost always done using high-performance computing facilities, such as AWS (Amazon Web Service), and cloud service providers, such as Google Cloud. We have, however, in our experiment, utilised our system. Our study's system consists of 62 GB of random access memory (RAM) with a 16 core Intel i7 3.8 GHz processor. Additionally, the system has two Nvidia RTX 2080 Ti Graphics Processing Unit (GPU). The GPU has 4352 CUDA cores, with 11 GB of GDDR6 Standard Memory, and supports the Base Clock of 1350 MHz for CUDA cores and 14 Gbps memory speed and 616 GB/sec memory bandwidth[17].

The implementation can be performed using many different languages like Python[18], R[19] and libraries such as Tensorflow[15], Keras[20], Scikit-learn[21]. The experiment was conducted using Python version 3, along with libraries such as Scikit-learn 0.22 and TensorFlow 2. The Scikit-learn library was employed in the design and implementation of random forest classifier and gradient boosting classifier. As in the implementation of LSTM and GRU, the implementation of TensorFlow was used.

The Tensorflow is able to take advantage of the available GPU. Tensorflow, however, can be further accelerated by using TensorRT[22]. TensorRT is a C++-based library provided by Nvidia that help with high-performance inference on NVIDIA GPUs (GPUs). TensorRT is typically used to enhance and expedite the deep learning training process. Furthermore, TensorFlow includes TensorRT[22]. So, in order to use the GPU as efficiently

[17] https://www.nvidia.com/en-eu/geforce/graphics-cards/rtx-2080-ti/
[18] https://www.python.org
[19] https://www.r-project.org
[20] https://keras.io
[21] https://scikit-learn.org/stable/
[22] https://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide/index.html

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 November 2021 doi:10.20944/preprints202111.0548.v1

as possible and to optimise and accelerate the deep learning training process, we also use TensorRT. Our study used the TensorRT version 6.0.1. TensorRT additionally needs CUDA, a parallel programming platform. In order to allow for TensorRT, we installed CUDA 10.0.130.

4.2. Evaluation metrics

In this section, we will go over evaluation metrics and explain why they are necessary, as well as which evaluation metrics were chosen and why they were chosen.

Evaluation is one of the important steps of any implementation. It helps to understand the quality of implementation. In machine learning, evaluation helps to understand the model's quality and is performed using evaluation metrics. It tells us how well the model will perform in a similar unseen scenario. Therefore, the correct use of model evaluation is vital in academic machine learning research, and many industrial settings [55].

Studies such as [56], [57] and [58] have shown that relying on single evaluation metrics, especially in the case of unbalanced data is not safe as they can sometimes be misleading. Because of this reason, other studies such as Das et al. [22], Islam et al. [59] also makes use of multiple evaluation metrics. Therefore, we have selected multiple evaluation metrics for our work: precision, recall, accuracy, F1 score, and adjusted F1 score (AGF).

4.2.1. Accuracy

Accuracy is the measure of correct overall predictions. Accuracy can be calculated by:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{13}$$

where, TP is true positive value, TN is true negative value, FP is false positive value and FN is false negative value.

4.2.2. Precision

The precision, also known as positive predicted value (PPV) is a measure of a model's exactness and a higher precision value for a classifier is preferred [56]. The precision can be calculated using Equation 14.

$$precision = \frac{TP}{TP + FP} \tag{14}$$

4.2.3. Recall

The recall is also called sensitivity or true positive rate, evaluates the classifier's effectiveness on the positive/minority by measuring the accuracy of positive cases [56]. Recall can be calculated using Equation 15.

$$recall = \frac{TP}{TP + FN} \tag{15}$$

4.2.4. F1 Score

F1 Score, also called F-measure, is the harmonic mean of precision and recall [60] and is calculated using Equation 16.

4.2.5. Adjusted F-measure

The Adjusted F-measure (AGF) is an improved version of the F1 Score, can account for data imbalance and thus tell us how well our model performs.

$$F1 \quad Score = 2 \times \frac{precision \times recall}{precision + recall} \tag{16}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 November 2021 doi:10.20944/preprints202111.0548.v1

Equation 19 can be used to calculate AGF [56]. The AGF calculation is followed by two intermediate equations, Equation 17 and Equation 18. Equation 18 calculates the inverse F-measure by switching from positive to negative classes and vice versa, resulting in a new confusion matrix that is then used to calculate AGF [56].

$$F_2 = 5 \times \frac{precision \times recall}{(4 \times recall) + precision} \tag{17}$$

$$InvF_{0.5} = \frac{5}{4} \times \frac{precision \times recall}{(0.5^2 \times recall) + precision} \tag{18}$$

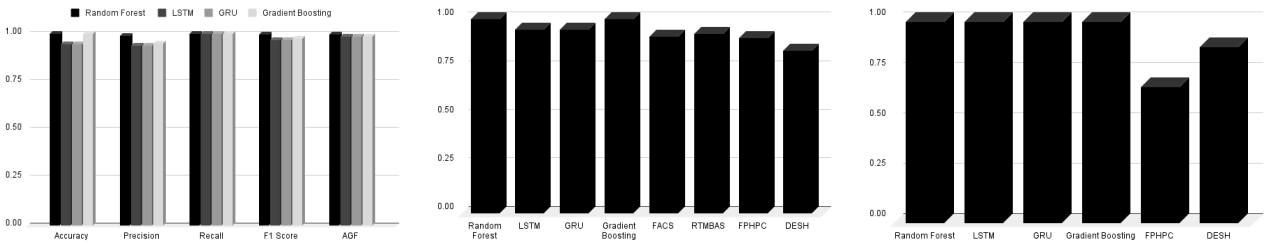
$$AGF = \sqrt{InvF_{0.5} \times F_2} \tag{19}$$

4.3. Training and Testing

In this section, we describe how we trained and tested our model.

As described in Section 4.1, the system we used has two GPUs. As a result, to make the best use of the available resources and accelerate the training process, we used the distributed training strategy for LSTM and GRU. We used MirroredStrategy out of the available distributed training strategies such as *MultiWorkerMirroredStrategy*, *TPUStrategy*, and *CentralStorageStrategy*[23]. The reason being the MirroredStrategy supports synchronous distributed training on multiple GPUs on one machine, and our system has both GPUs on the same machine. Further, we used the batch size of 556 and 430 epoch for training. Similarly, for the training of random forest and gradient boosting, we used Joblib[24] with 12 parallel jobs using selected hyperparameters and K-Fold cross-validation as described in Section 3.4 and 3.5. Further, the details on used parameters for LSTM and GRU is presented in Sections 3.6.1 and 3.6.2 respectively. The experiment was conducted with a 60% :40% , 55% :45% , 70% :30% , and 80% :20% train test split. However, only the best result was recorded following the observations from the studies such as Bala et al. [32], Liu et al. [29] where the reported result is from only a combination of train test split. We obtained with a train test split of 60% :40%, which will be discussed in Section 4.4.

4.4. Results



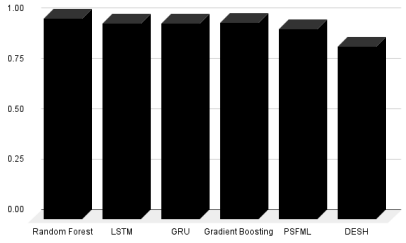
(a) Experimental result **(b)** Comparison of Accuracy **(c)** Comparison of Recall

Figure 7. Experimental result and its comparison to the state-of-the-art.

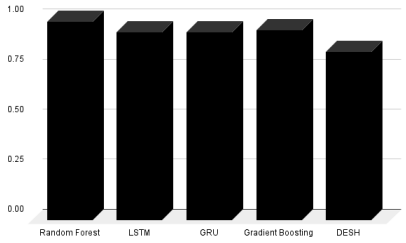
In Fig. 7a, the Y-axis represents the percentage, and the X-axis includes the evaluation metrics accuracy, precision, recall, F1 score, and AGF for each of the four algorithms. Overall, we see high accuracy, precision, recall, F1 score, and AGF with all four algorithms tested using our combined metrics approach, with results averaging at least 95% for each evaluation metric. The consistent high value of all evaluation metrics with all four selected algorithms demonstrates the benefits of the combined metrics, validating our claim. Furthermore, the high value of F1 score and AGF demonstrate that our approach will perform well even with unbalanced data (or minority class). However,

[23] https://www.tensorflow.org/guide/distributed_training
[24] https://joblib.readthedocs.io/en/latest/

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 November 2021 doi:10.20944/preprints202111.0548.v1



(a) Comparison of F1 score



(b) Comparison of Precision

Figure 8. Experimental F1 score and precision versus state-of-the-art.

when we examine the results at a finer level, we can see that RF outperforms GB, LSTM, and GRU. The GB comes in second on the list, with accuracy as close to RF as possible. The GB, on the other hand, falls short in precision by nearly 4% when compared to the RF. LSTM and GRU have comparable performance, but GB and RF outperform them by about 4-5% in terms of accuracy and precision and by about 2-3% in terms of F1 score. The difference between the F1 score and the AGF is very small, ranging from 1-2 percent and averaging to 97 percent. In terms of recall, we can see that all four algorithms produced the same result.

The consistent high value of the evaluation metrics demonstrates the advantages of our approach. However, we cannot say how good our approach is unless we compare it to relevant state-of-the-art studies. As a result, in order to understand where our approach stands in comparison to existing studies, we compared our findings to those of cutting-edge research. Fig. 7b, 7c, 8a and 8b shows the comparison of our accuracy, recall, precision and F1 score with the closest state-of-the-art studies, namely, FACS [59], RTMBAS [27], FPHPC [19], DESH [22] and PSFML [21]. When we compare accuracy, recall, precision and F1 score, we can see that all of our tested methods using combined metrics outperform the existing state-of-the-art. In terms of accuracy, our combined metrics approach outperforms others by up to 11% and recall by up to 32%. Similarly, the F1 score demonstrates that our approach outperforms the compared state-of-the-art by nearly up to 5% and precision by up to 10%.

Finally, the results from our experiment and comparison to state-of-the-art studies substantiate our claim that using SMART metrics in conjunction with other system metrics such as CPU and memory utilisation can improve failure prediction. Furthermore, with improved failure prediction, we can detect potential failures accurately before they occur, allowing us to take proactive action and, as a result, improve reliability.

5. Conclusion

We hypothesised that the combined metrics approach would improve failure prediction and presented our work on cloud server failure prediction in this paper. Furthermore, we substantiated our hypothesis and demonstrated the competitive advantage of combined metrics by comparing our results to state-of-the-art studies based on the experimental results. Future work would involve expanding on this approach of combined metrics with additional data and implementing the production method.

Author Contributions: **Tek Raj Chhetri**: Conceptualisation, Formal Analysis, Investigation, Methodology, Data curation, Software, Validation, Visualisation, Writing - original draft, Writing - review & editing; **Chinmaya Kumar Dehury**: Conceptualisation, Supervision, Writing - review & editing, Validation; **Artjom Lind**: Supervision, Writing - review & editing, Software, Validation; **Satish Narayana Srirama**: Supervision, Writing - review & editing; **Anna Fensel**: Writing - review & editing.

Funding: This research received no external funding.

Acknowledgments: We would like to thank the University of Tartu High-Performance Computing (HPC) centre, particularly *Sander Kuusemets* and *Anders Martoja*, for their assistance. Additionally, we would like to express our profound gratitude to *Amnir Hadachi*, head of the ITS lab, University of Tartu, for his support.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations representing the research studies are used in this manuscript:

FACS	FaCS: Toward a Fault-Tolerant Cloud Scheduler Leveraging Long Short-Term Memory Network
RTMBAS	Cloud Computing: A model Construct of Real-Time Monitoring for Big Dataset Analytics Using Apache Spark
FPHPC	Failure prediction using machine learning in a virtualised HPC system and application
DESH	Desh: deep learning for system health prediction of lead times to failure in HPC

References

1. Buyya, R.; Srirama, S.N.; Casale, G.; Calheiros, R.; Simmhan, Y.; Varghese, B.; Gelenbe, E.; Javadi, B.; Vaquero, L.M.; Netto, M.A.; others. A manifesto for future generation cloud computing: Research directions for the next decade. *ACM computing surveys (CSUR)* **2018**, *51*, 1–38.
2. Saif, M.A.N.; Niranjan, S.; Al-Ariki, H.D.E. Efficient autonomic and elastic resource management techniques in cloud environment: taxonomy and analysis. *Wireless Networks* **2021**, pp. 1–38.
3. Periola, A. Incorporating diversity in cloud-computing: a novel paradigm and architecture for enhancing the performance of future cloud radio access networks. *Wireless Networks* **2019**, *25*, 3783–3803.
4. Jiang, D. The construction of smart city information system based on the Internet of Things and cloud computing. *Computer Communications* **2020**, *150*, 158–166.
5. Javadzadeh, G.; Rahmani, A.M. Fog computing applications in smart cities: A systematic survey. *Wireless Networks* **2020**, *26*, 1433–1457.
6. Saini, H.; Upadhyaya, A.; Khandelwal, M.K. Benefits of Cloud Computing for Business Enterprises: A Review. *Available at SSRN 3463631* **2019**.
7. Varadarajan, V.; Neelanarayanan, V.; Doyle, R.; Al-Shaikhli, I.F.; Groppe, S. Emerging Solutions in Big Data and Cloud Technologies for Mobile Networks. *Mobile Networks and Applications* **2019**, *24*, 1015–1017.
8. Langmead, B.; Nellore, A. Cloud computing for genomic data analysis and collaboration. *Nature Reviews Genetics* **2018**, *19*, 208.
9. Doheir, M.; Basari, A.S.H.; Hussin, B.; Yaacob, N.M.; Al-Shami, S.S.A.; others. The New Conceptual Cloud Computing Modelling for Improving Healthcare Management in Health Organizations. *International Journal of Advanced Science and Technology 351-362* **2019**, *28*, 351–362.
10. Liu, Y.; Zhang, L.; Yang, Y.; Zhou, L.; Ren, L.; Wang, F.; Liu, R.; Pang, Z.; Deen, M.J. A novel cloud-based framework for the elderly healthcare services using digital twin. *IEEE Access* **2019**, *7*, 49088–49101.
11. Chuck Byers (IIC staff), Ron Zahavi (Microsoft), J.K.Z.N.C.T.U. The Edge Computing Advantage. 2019.
12. Luo, L.; Meng, S.; Qiu, X.; Dai, Y. Improving failure tolerance in large-scale cloud computing systems. *IEEE Transactions on Reliability* **2019**, *68*, 620–632.
13. Huang, M.H.; Rust, R.T. Artificial intelligence in service. *Journal of Service Research* **2018**, *21*, 155–172.
14. Ropinski, T.; Archambault, D.; Chen, M.; Maciejewski, R.; Mueller, K.; Telea, A.; Wattenberg, M. How do Recent Machine Learning Advances Impact the Data Visualization Research Agenda? *IEEE VIS Panel. Phoenix* **2017**.
15. Ramachandram, D.; Taylor, G.W. Deep Multimodal Learning: A Survey on Recent Advances and Trends. *IEEE Signal Processing Magazine* **2017**, *34*, 96–108.
16. Protecting intangible assets: Preparing for a new reality. https://assets.kpmg/content/dam/kpmg/uk/pdf/2020/08/lloyds-intangibles-6-aug-2020-.pdf. [Online; accessed 24-October-2020].

17. Ajour El Zein, S.; Consolacion-Segura, C.; Huertas-Garcia, R. The Role of Sustainability in Brand Equity Value in the Financial Sector. *Sustainability* **2020**, *12*, 254.
18. Turnbull, J. *Monitoring with Prometheus*; Turnbull Press, 2018.
19. Mohammed, B.; Awan, I.; Ugail, H.; Younas, M. Failure prediction using machine learning in a virtualised HPC system and application. *Cluster Computing* **2019**, *22*, 471–485.
20. Xu, Y.; Sui, K.; Yao, R.; Zhang, H.; Lin, Q.; Dang, Y.; Li, P.; Jiang, K.; Zhang, W.; Lou, J.G.; others. Improving service availability of cloud systems by predicting disk error. 2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18), 2018, pp. 481–494.
21. Lai, B. Predicting Server Failures with Machine Learning. Technical report, SLAC National Accelerator Lab., Menlo Park, CA (United States), 2018.
22. Das, A.; Mueller, F.; Siegel, C.; Vishnu, A. Desh: deep learning for system health prediction of lead times to failure in hpc. Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing, 2018, pp. 40–51.
23. Chigurupati, A.; Thibaux, R.; Lassar, N. Predicting hardware failure using machine learning. 2016 Annual Reliability and Maintainability Symposium (RAMS). IEEE, 2016, pp. 1–6.
24. Fadaei Tehrani, A.; Safi-Esfahani, F. A threshold sensitive failure prediction method using support vector machine. *Multiagent and Grid Systems* **2017**, *13*, 97–111.
25. Adamu, H.; Mohammed, B.; Maina, A.B.; Cullen, A.; Ugail, H.; Awan, I. An Approach to Failure Prediction in a Cloud Based Environment. 2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud), 2017, pp. 191–197. doi:10.1109/FiCloud.2017.56.
26. Meenakumari, J. Virtual Machine (VM) Earlier Failure Prediction Algorithm. *International Journal of Applied Engineering Research* **2017**, *12*, 9285–9289.
27. Alkasem, A.; Liu, H.; Zuo, D.; Algarash, B. Cloud computing: a model construct of real-time monitoring for big dataset analytics using apache spark. Journal of Physics: Conference Series. IOP Publishing, 2018, Vol. 933, p. 012018.
28. Qasem, G.M.; Madhu, B. Proactive fault tolerance in cloud data centers for performance efficiency. *Int J Pure Appl Math* **2017**, *117*, 325–329.
29. Liu, D.; Wang, B.; Li, P.; Stones, R.J.; Marbach, T.G.; Wang, G.; Liu, X.; Li, Z. Predicting Hard Drive Failures for Cloud Storage Systems. Algorithms and Architectures for Parallel Processing; Wen, S.; Zomaya, A.; Yang, L.T., Eds.; Springer International Publishing: Cham, 2020; pp. 373–388.
30. Shetty, J.; Sajjan, R.; Shobha, G. Task Resource Usage Analysis and Failure Prediction in Cloud. 2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence). IEEE, 2019, pp. 342–348.
31. Jassas, M.; Mahmoud, Q.H. Failure analysis and characterization of scheduling jobs in google cluster trace. IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society. IEEE, 2018, pp. 3102–3107.
32. Bala, A.; Chana, I. Intelligent failure prediction models for scientific workflows. *Expert Systems with Applications* **2015**, *42*, 980–989.
33. Rosa, A.; Chen, L.Y.; Binder, W. Predicting and mitigating jobs failures in big data clusters. 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. IEEE, 2015, pp. 221–230.
34. Gao, J.; Wang, H.; Shen, H. Task Failure Prediction in Cloud Data Centers Using Deep Learning. 2019 IEEE International Conference on Big Data (Big Data). IEEE, 2019, pp. 1111–1116.
35. University of Tartu. UT Rocket, 2018. doi:10.23673/PH6N-0144.
36. Qasem, G.M.; Madhu, B. A Classification Approach for Proactive Fault Tolerance in Cloud Data Centers. *International Journal of Applied Engineering Research* **2018**, *13*, 15762–15765.
37. Breiman, L. Random forests. *Machine learning* **2001**, *45*, 5–32.
38. Scornet, E.; Biau, G.; Vert, J.P.; others. Consistency of random forests. *The Annals of Statistics* **2015**, *43*, 1716–1741.
39. Biau, G.; Scornet, E. A random forest guided tour. *Test* **2016**, *25*, 197–227.
40. Kulkarni, A.D.; Lowe, B. Random forest algorithm for land cover classification **2016**.
41. Ramo, R.; Chuvieco, E. Developing a random forest algorithm for MODIS global burned area classification. *Remote Sensing* **2017**, *9*, 1193.
42. Khaidem, L.; Saha, S.; Dey, S.R. Predicting the direction of stock market prices using random forest. *arXiv preprint arXiv:1605.00003* **2016**.
43. Berrar, D. Cross-validation. *Encyclopedia of Bioinformatics and Computational Biology* **2019**, pp. 542–545.
44. Wong, T.T. Performance evaluation of classification algorithms by k-fold and leave-one-out cross validation. *Pattern Recognition* **2015**, *48*, 2839–2846.
45. Athanasiou, V.; Maragoudakis, M. A novel, gradient boosting framework for sentiment analysis in languages where NLP resources are not plentiful: a case study for modern Greek. *Algorithms* **2017**, *10*, 34.
46. Dangeti, P. *Statistics for Machine Learning*; Packt Publishing, 2017.
47. Chakrabarty, N.; Kundu, T.; Dandapat, S.; Sarkar, A.; Kole, D.K. Flight Arrival Delay Prediction Using Gradient Boosting Classifier. In *Emerging Technologies in Data Mining and Information Security*; Springer, 2019; pp. 651–659.
48. Allen Yu, Claire Chung, P.D.; Yim, A. *Numerical Computing with Python*; Packt Publishing, 2018.
49. Kurth, T.; Treichler, S.; Romero, J.; Mudigonda, M.; Luehr, N.; Phillips, E.; Mahesh, A.; Matheson, M.; Deslippe, J.; Fatica, M.; others. Exascale deep learning for climate analytics. SC18: International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, 2018, pp. 649–660.

50. Sujit Pal, A.G. *Deep Learning with Keras*; Packt Publishing, 2017.
51. Armando Fandango, Rajalingappaa, S.; Bonaccorso, G. *Python: Advanced Guide to Artificial Intelligence*; Packt Publishing, 2018.
52. Vasilev, I. *Advanced Deep Learning with Python*; Packt Publishing, 2019.
53. Ravichandiran, S. *Hands-On Deep Learning Algorithms with Python*; Packt Publishing, 2019.
54. Gal, Y.; Ghahramani, Z. A theoretically grounded application of dropout in recurrent neural networks. Advances in neural information processing systems, 2016, pp. 1019–1027.
55. Raschka, S. Model evaluation, model selection, and algorithm selection in machine learning. *arXiv preprint arXiv:1811.12808* **2018**.
56. Akosa, J. Predictive accuracy: a misleading performance measure for highly imbalanced data. Proceedings of the SAS Global Forum, 2017, pp. 2–5.
57. Juba, B.; Le, H.S. Precision-recall versus accuracy and the role of large data sets. Proceedings of the AAAI Conference on Artificial Intelligence, 2019, Vol. 33, pp. 4039–4048.
58. Hossin, M.; Sulaiman, M. A review on evaluation metrics for data classification evaluations. *International Journal of Data Mining & Knowledge Management Process* **2015**, *5*, 1.
59. Islam, T.; Manivannan, D. FaCS: Toward a Fault-Tolerant Cloud Scheduler Leveraging Long Short-Term Memory Network. 2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom). IEEE, 2019, pp. 1–6.
60. Capellman, J. *Hands-On Machine Learning with ML.NET*; Packt Publishing, 2020.

Short Biography of Authors

Tek Raj Chhetri holds a Master's degree in Computer Science from the University of Tartu, Estonia. He was awarded a full-time exemption scholarship from Tartu University, a grant from the Estonian Ministry of Foreign Affairs and a Kristjan Jaak scholarship from the Archimedes Foundation in Tartu, Estonia. He worked as a research assistant on an autonomous vehicle at the Intelligent Transportation System Lab, Tartu and as a software developer at Auve Tech. He is currently a PhD student at the University of Innsbruck. His current research interests include Data Analysis, KGs, Responsible AI, Privacy, Intelligent Transportation Systems and Distributed Systems.

Chinmaya Kumar Dehury received a bachelor's degree from Sambalpur University, India, in June 2009 and an MCA degree from Biju Pattnaik University of Technology, India, in June 2013. He received a Ph.D. Degree in the Department of Computer Science and Information Engineering, Chang Gung University, Taiwan. He is currently a Lecturer of Distributed System, member of Mobile & Cloud Lab in the Institute of Computer Science, University of Tartu, Estonia. His research interests include scheduling, resource management and fault tolerance problems of Cloud and fog Computing, the application of artificial intelligence in cloud management, edge intelligence, Internet of Things, and data management frameworks. His research results are published by top-tier journals and transactions such as IEEE TCC, JSAC, TPDS, FGCS, etc. He is a member of IEEE and ACM India. He is also serving as a PC member of several conferences and reviewer to several journals and conferences, such as IEEE TPDS, IEEE JSAC, IEEE TCC, IEEE TNNLS, Wiley Software: Practice and Experience, etc.

Artjom Lind Artjom Lind received the B.Sc. and M.Sc. degrees in computer science from the University of Tartu, in 2008 and 2010, respectively. From 2008 to 2011, he was a Research Assistant with the Distributed Systems Group, Institute of Computer Science. From 2011 to 2012, he was a Visiting Researcher with the University of Kassel, Germany. From 2013 to 2017, he was a Research Assistant with the ITS Team, Institute of Computer Science, University of Tartu. His interests include distributed systems, peer-to-peer networks, mobile telecommunications, and intelligent transportation systems.

 doi:10.20944/preprints202111.0548.v1

Satish Narayana Srirama is an Associate Professor at the School of Computer and Information Sciences, University of Hyderabad, India. He is also a Visiting Professor and the honorary head of the Mobile & Cloud Lab at the Institute of Computer Science, University of Tartu, Estonia, which he led as a Research Professor until June 2020. He received his PhD in computer science from RWTH Aachen University, Germany in 2008. His current research focuses on cloud computing, mobile web services, mobile cloud, Internet of Things, fog computing, migrating scientific computing and enterprise applications to the cloud and large-scale data analytics on the cloud. He is an IEEE Senior Member, an Editor of Wiley Software: Practice and Experience, a 51 year old Journal, was an Associate Editor of IEEE Trans- actions in Cloud Computing and a program committee member of several international conferences and workshops. Dr. Srirama has co-authored over 165 refereed scientific publications in international conferences and journals.

Anna Fensel was born in Novosibirsk, Russian Federation. She has a university degree in mathematics and computer science from Novosibirsk State University, Russia (2003). She received both her doctoral degree (2006) and her habilitation (2018) in computer science at the University of Innsbruck, Austria. In 2006-2007, she worked as a Research Fellow at the University of Surrey, UK, and then as a Senior Researcher at FTW – Telecommunications Research Centre Vienna, Austria (2007-2013), and afterwards as Assistant and then, currently, as Associate Professor at the University of Innsbruck, Austria. Since 2021, Anna Fensel is Associate Professor at Wageningen University and Research, in Wageningen, the Netherlands. She has been a co-organizer or a Program Committee member of more than 100 scientific events including being in chair roles at key events in her field such as SEMANTiCS and ESWC, a reviewer for numerous journals and a project proposals evaluator for funding agencies (EU H2020 and FP6, Eureka-Eurostars, national funding). She is a (co-)author of more than 120 refereed publications.