Predicting Node Failure in Cloud Service Systems

Qingwei Lin
Microsoft Research
Beijing, China

Ken Hsieh
Microsoft
Redmond, USA

Yingnong Dang
Microsoft
RedmondUSA

Hongyu Zhang
The University of Newcastle
NSW, Australia

Kaixin Sui
Microsoft Research
Beijing, China

Yong Xu
Microsoft Research
Beijing, China

Jian-Guang Lou
Microsoft Research
Beijing, China

Chenggang Li
Microsoft Research
Beijing, China

Youjiang Wu
Microsoft
Redmond, USA

Randolph Yao
Microsoft
Redmond, USA

Murali Chintalapati
Microsoft
Redmond, USA

Dongmei Zhang
Microsoft Research
Beijing, China

ABSTRACT

In recent years, many traditional software systems have migrated to cloud computing platforms and are provided as online services. The service quality matters because system failures could seriously affect business and user experience. A cloud service system typically contains a large number of computing nodes. In reality, nodes may fail and affect service availability. In this paper, we propose a failure prediction technique, which can predict the failure-proneness of a node in a cloud service system based on historical data, before node failure actually happens. The ability to predict faulty nodes enables the allocation and migration of virtual machines to the healthy nodes, therefore improving service availability. Predicting node failure in cloud service systems is challenging, because a node failure could be caused by a variety of reasons and reflected by many temporal and spatial signals. Furthermore, the failure data is highly imbalanced. To tackle these challenges, we propose MING, a novel technique that combines: 1) a LSTM model to incorporate the temporal data, 2) a Random Forest model to incorporate spatial data; 3) a ranking model that embeds the intermediate results of the two models as feature inputs and ranks the nodes by their failure-proneness, 4) a cost-sensitive function to identify the optimal threshold for selecting the faulty nodes. We evaluate our approach using real-world data collected from a cloud service system. The results confirm the effectiveness of the proposed approach. We have also successfully applied the proposed approach in real industrial practice.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ESEC/FSE '18, November 4–9, 2018, Lake Buena Vista, FL, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5573-5/18/11…$15.00
https://doi.org/10.1145/3236024.3236060

CCS CONCEPTS

• **Software and its engineering** → *Software testing and debugging*; *Maintaining software*;

KEYWORDS

Failure prediction, service availability, node failure, cloud service systems, maintenance.

ACM Reference Format:
Qingwei Lin, Ken Hsieh, Yingnong Dang, Hongyu Zhang, Kaixin Sui, Yong Xu, Jian-Guang Lou, Chenggang Li, Youjiang Wu, Randolph Yao, Murali Chintalapati, and Dongmei Zhang. 2018. Predicting Node Failure in Cloud Service Systems . In *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '18), November 4–9, 2018, Lake Buena Vista, FL, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3236024.3236060

1 INTRODUCTION

In recent years, deploying applications and services on large-scale cloud platforms, such as Microsoft Azure, Google Cloud, and Amazon AWS, has been widely accepted by software industry. These cloud service systems need to provide a variety of services to millions of users from around the world every day, therefore high service availability is essential as a small problem could cause serious consequences. Many service providers have made tremendous efforts to maintain high service availability. For example, Amazon EBS [1] claims to have "five nines", which represents the service availability of 99.999%, allowing at most 26 seconds down time per month per VM. Microsoft Azure [27] also claims similar service availability.

Although a lot of effort has been devoted to service quality assurance [2, 16, 23, 28, 37], in reality, cloud service systems still encounter many problems and fail to satisfy user requests. These problems are often caused by failures of computing nodes in cloud service systems. A cloud service system typically contains a large number of computing nodes, which supply the processing, network, and storage resources that virtual machine instances need. Some empirical studies have dedicated to the analysis of node failures. For

example, Vishwanath and Nagappan [38] classified server failures in a data center and found that 8% of all servers had at least one hardware incident in a given year. Ford et al. studied [11] the availability of Google distributed storage systems, and characterized the sources of faults contributing to unavailability. Dinu and Ng [9] analyzed Hadoop behavior when nodes fail and found that a single failure can result in unpredictable system performance. In Microsoft, each day, out of all the server nodes in its cloud system, less than 0.1% of the nodes encounter failures. While the failure rate of 0.1% may seem insignificant, it has drastic impact on services that target at 99.999% availability or beyond. According to our experience in Microsoft, node failure was one of the top causes of service down time.

To improve service availability, we propose to predict the failure-proneness of a node in cloud service systems before the failure actually happens. We apply machine learning techniques to learn the characteristics of historical failure data, build a failure prediction model, and then use the model to predict the likelihood of a node failing in the coming days. The ability to predict faulty nodes enables cloud service systems to allocate VMs (Virtual Machines) to the healthier nodes, therefore reducing the occurrences and duration of VM down time caused by the node failures. Furthermore, if a node is predicted as faulty, the cloud service system can perform proactive live migration - to migrate the virtual machines from the faulty node to a healthy node without disconnecting the service.

However, building an accurate prediction model for node failure in cloud service systems is challenging. We have identified three main reasons:

Complicated failure causes: Due to the complexity of the large-scale cloud system, node failures could be caused by many different software or hardware issues. Examples of these issues include software bugs, OS crash, disk failure, service exception, etc. There is no simple rule/metric that can predict all node failures in a straightforward manner.

Complex failure-indicating signals: Failures of a node could be indicated by many temporal signals produced by the node locally. They could also be reflected by spatial properties that are shared by nodes that have explicit/implicit dependency among them in different global views of the cloud. We need to analyze both temporal signals and spatial properties to better capture the early failure-indicating signals.

Highly imbalanced data: Node fault data is highly imbalanced as most of the time the cloud service system has high service availability. For example, in our system, the node ratio between failure and healthy classes is less than 1:1000 (i.e., less than 0.1% nodes contain failures). The highly imbalanced data poses great challenges to prediction.

To tackle the challenges, we propose MING, a novel technique for predicting node failure in cloud service systems. MING includes: 1) a LSTM-based deep learning model to incorporate the temporal data, 2) a Random Forest model to incorporate the spatial data, 3) a learning-to-rank model that embeds the intermediate results of the two models as feature inputs and ranks the nodes by their failure-proneness, and 4) a cost-sensitive function to identify the optimal threshold in the ranked results for selecting the faulty nodes.

We evaluate our approach using real-world data collected from a cloud-based service system in production. The results show that MING outperforms the baseline approaches that are implemented using conventional classification techniques. Furthermore, we have successfully applied MING to the maintenance of a cloud service system X in Microsoft since September 2017. In a typical day, the top 1% failure-prone nodes in Service X identified by MING capture above 60% of the failures in the next day. In an A/B testing conducted by the Service X team, MING is able to intelligently allocate new VMs to more healthier nodes and has achieved above 30% reduction in these VMs' failure rate. MING won the 2017 "Tech Transfer of The Year" award in a research division of Microsoft.

The main contributions of our work are as follows:

- We propose MING, which can improve service availability by predicting node failure in cloud service systems. Through failure prediction, intelligent VM allocation and migration can be achieved.
- To build an accurate failure prediction model, we design a two-phase training model, which can well handle the temporal and spatial features and is less sensitive to highly imbalanced data.
- We evaluate our approach using data collected from a cloud-based service system. MING achieves the average Recall of 63.5% and Precision of 92.4% on three datasets, and outperforms the baseline approaches built using conventional classifiers.
- We have applied MING to the maintenance of a large-scale cloud service system. The results confirm the effectiveness of MING in industrial practice. To the best of our knowledge, this is the first time node failure prediction is applied in a production cloud service environment.

The rest of this paper is organized as follows: In Section 2, we introduce the background and motivation of our work. Section 3 covers the proposed framework and algorithms. The evaluation is described in Section 4. We also discuss the results and presents the threats to validity. In Section 5, we share our success stories and experience obtained from industrial practice. The related work and conclusion are presented in Section 6 and Section 7, respectively.

2 IMPROVING SERVICE AVAILABILITY OF CLOUD SYSTEMS

2.1 Cloud Service Systems

Cloud computing has emerged as a new paradigm for delivery of computing as services via the Internet. It offers many service models such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Deploying applications and services on cloud computing platforms such as Microsoft Azure and Amazon Web Services has been widely accepted by software organizations and developers.

A typical cloud service system contains a large number of physical servers, or "nodes". For example, Amazon Web Services is likely to have 1.3 million servers[1]. The nodes are arranged into racks and a group of racks form a cluster. Virtualization is one of key technologies used in modern cloud computing, which offers better scalability, maintainability, and reliability. A physical node can host multiple virtual machines (VMs). VMs can be backed up, scaled or

[1] http://www.zdnet.com/article/aws-cloud-computing-ops-data-centers-1-3-million-servers-creating-efficiency-flywheel/

duplicated, making it easy to suit end users' needs. When a VM allocation request is sent out, the cloud service system will determine the appropriate node to host the VM. If a node fails, all VMs hosted on it will correspondinly fail. Cloud service systems also support *live migration*, which refers to the process of moving a running VM between different nodes without disconnecting the client or application [7]. Live migration is a powerful mechanism for managing cloud services, as it enables rapid movement of workloads within clusters with low impact on running services.

2.2 Service Availability

For large-scale software systems, especially cloud service systems such as Microsoft Azure, Amazon AWS, and Google Cloud, high service availability is crucial. Service availability is a state of a service being accessible to the end user. Usually expressed as a percentage of uptime, it represents the level of operational performance of a system or component for a given period of time. As cloud systems provide services to hundreds of millions of users around the world, service problems can often lead to great revenue loss and user dissatisfaction. Hence, in today's practice, the service providers have made every effort to maintain a high service availability, such as "five nines" (99.999%), meaning less than 26 seconds down time per month per VM allowed[2].

Although tremendous effort has been made to maintain high service availability, in reality, there are still many unexpected system problems caused by software or platform failures (such as software crashes, network outage, misconfigurations, memory leak, hardware breakdown, etc.). These problems become more severe with the ever-increasing scale of the service systems. For example, in February 2017, AWS experienced a massive outage of its S3 Storage services, causing a majority of websites which relied on AWS S3 unresponsive. 54 of the Internet's top 100 retailers observed website performance slow by 20% or more[3].

It has been found that node failure is one of the most common problems that cause system unavailability [38]. Our experience in Microsoft also shows that node failure is the dominant cause of service down time. If a node fails, all the VMs running on it will correspondingly fail, which could affect service availability.

2.3 Improving Service Availability by Node Failure Prediction

Different nodes may fail at different times. We propose to predict the failure-proneness of a node based on the analysis of historical fault data, before the node fails. The ability to predict node failure can help improve service availability from the following two aspects:

- VM allocation, which is the process of allocating a VM (virtual machine) to a node. To enable better VM allocation, we can always allocate VMs to a healthier node rather than to a faulty node.
- Live migration, which is the process of moving a running VM between different nodes without disconnecting the client or application. To enable more effective live migration of nodes, we can proactively migrate VMs from the predicted faulty nodes to the healthy ones, before node failure actually happens.

To achieve so, we can build a prediction model based on historical failure data using machine learning techniques, and then use the model to predict the likelihood of a node failing in the near future. The prediction model should have the following abilities:

- The prediction model should be able to rank all nodes by their failure-proneness so that the service systems can allocate a VM to the healthiest node available.
- The prediction model should be able to identify a set of faulty nodes from which the hosted VMs should be migrated out, under the constrains of cost and capacity.

There are several technical challenges in designing a failure prediction model for a large-scale cloud:

2.3.1 Complicated Failure Causes. Due to the complexity of a cloud service system, node failures could be caused by many different software or hardware issues. Examples of these issues include OS crash, application bugs, disk failure, misconfigurations, memory leak, software incompatibility, overheating, service exception, etc. According to the estimation of domain experts in Microsoft, the number of root causes of node failure is over one hundred. Simple rule-based or threshold-based models are not able to locate the problem and achieve good prediction results. To tackle this challenge, in this paper, we propose a machine learning based approach to node failure prediction in cloud systems.

2.3.2 Complex Failure-indicating Signals. Failures of a single node could be indicated by temporal signals coming from a variety of software or hardware sources of the node. Examples of the temporal signals are performance counters, logs, sensor data, and OS events. They are continuously monitored and changing over time.

In a large-scale cloud system, failures of a node could also be reflected by spatial properties shared by the nodes that are explicitly/implicitly dependent on each other. We have identified the following dependencies between two nodes: 1) Resource-based dependency: two nodes may compete for a computing resource (e.g., a router), 2) Location-based dependency: two nodes may co-exist in the same computing segment/domain (such as the same rack), 3) Load balancing based dependency: two nodes may be in the same group for load balancing. The mutually dependent nodes tend to be infected by the same failure-inducing cause. For example, if a certain portion of nodes fail, other nodes in the same segment could fail in the near future too. Therefore, the spatial properties that are shared among the mutually dependent nodes also have predictive power. Examples of the spatial properties include update domain, shared router, rack location, resource family, load balance group, batch operation group, etc.

We need to incorporate both temporal and spatial data in order to better capture the early failure-indicating signals and build an accurate prediction model. To tackle this challenge, in this paper, we construct two specific base learners to incorporate temporal and spatial data, respectively. We then ensemble the results to train a ranking model.

2.3.3 Highly Imbalanced Data. In a large-scale cloud service system of Microsoft, every day, only one in one thousand nodes could

[2]https://msdn.microsoft.com/en-us/library/ee799074(v=cs.20).aspx
[3]https://www.businessinsider.com.au/aws-outage-hurt-internet-retailers-except-amazon-2017-3?r=US&IR=T

become faulty. The extreme 1-in-1000 imbalanced ratio poses difficulties in training a classification model. Fed with such imbalanced data, a naive classification model could attempt to judge all nodes to be healthy, because in this way, it has the lowest probability of making a wrong guess. Some approaches apply data re-balancing techniques, such as over-sampling and under-sampling techniques, to address this challenge. Such approaches help raise the recall, but at the same time could introduce a large number of false positives, which dramatically decreases the precision. To tackle this challenge, in this paper, we propose a ranking model to rank the nodes by their failure-proneness. Unlike a conventional classification model whose objective is to find a best separation to distinguish all the positive and negative instances, a ranking model focuses on optimizing the top k returned results therefore it is more appropriate in our scenario.

3 THE PROPOSED APPROACH

3.1 Overview

In this paper, we propose MING, a novel technique for improving service availability by predicting node failure in a cloud service system. Figure 1 shows an overall workflow of the proposed approach. MING includes two phases of training. In Phase 1, two base classification models are trained: a LSTM model for temporal data and a Random Forest model for spatial data. In Phase 2, the intermediate results of the two base learners are embedded as features and fed as input to a ranking model. The ranking model ranks the nodes by their failure-proneness. MING identifies the top r ones that minimize the misclassification cost as the predicted faulty nodes. We describe the major steps in this section.

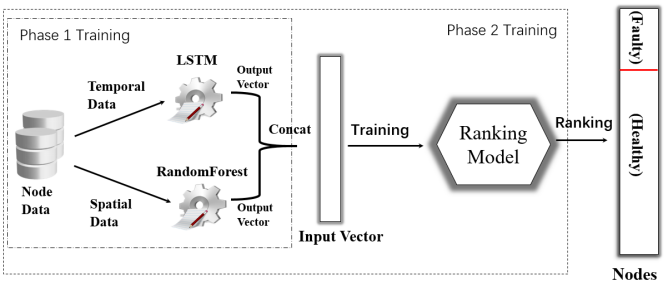

Figure 1: The overview of MING

3.2 Phase 1 Training

In this phase, we first train base learners on the training data. As stated in Section 2, to construct an effective node failure prediction model, we collect heterogeneous data for each node from diverse sources and identify features from the data with the help of product teams. Table 1 shows some examples of features. These features can be categorized into the following two types:

- Temporal features, which directly represent a node's local status in time (such as performance counters, IO throughput, resource usage, sensor values, response delays, etc.) or can be aggregated as temporal data from the original sources (such as log event counts, error/exception event counts, system event counts, etc.). We collect 137 of these features in total.
- Spatial features, which indicate explicit/implicit dependency in global relationships among nodes. Examples of these features include deployment segment, rack location, load balance group, policy group, update domain, etc. We collect 82 these features.

Table 1: Some examples of features

Feature	Type	Description
UpdateDomain	Spatial	The domain where nodes share same update setting.
MemoryUsage	Temporal	Memory consumption.
DiskSectorError	Temporal	Sector errors in a disk drive.
ServiceError	Temporal	Error counts from a deployed service.
RackLocation	Spatial	The location of the rack the node belongs to.
LoadBalanceGroup	Spatial	The group where nodes' load are balanced.
IOResponse	Temporal	I/O Response time.
OSBuildGroup	Spatial	The group where nodes have the same OS build.

It is known that different machine learning algorithms could work well only on some specific types of features, while performs weakly on others. To support specific type of features, feature conversion needs to be performed (e.g., converting categorical features into numeric values), which may incur much information loss. To cater for different types of features, in this phase we build a separate learner for each type of data: temporal and spatial.

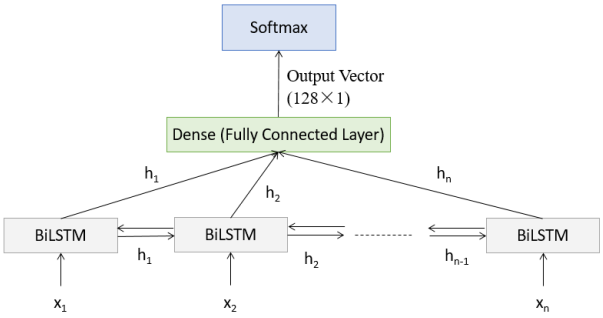

Figure 2: Training LSTM Model

For temporal features, we apply LSTM (Long Short-Term Memory), which is a widely adopted deep neural network (DNN) model [15, 19]. It can balance between retaining the previous state and memorizing new information. LSTM can better capture the patterns behind the time-series data and has proven to be successful in solving tasks such as machine translation and speech recognition. Figure 2 illustrates the training of the LSTM model in our approach. We use bi-directional LSTM [32] for the time series data $x_1, x_2, ..., x_n$, where x_i is the input vector of all temporal features at time i. The LSTM layer produces the representation sequence $h_1, h_2, ..., h_n$, which is fed to a dense layer (fully connected layer). The output of the dense layer is a *128 x 1* vector V_t. The vector is fed

to a $softmax$ function [15], which is the final layer of a DNN-based classifier.

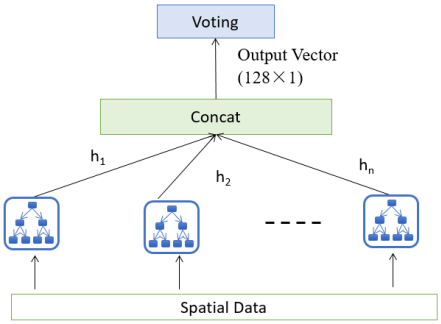

Figure 3: Training Random Forest Model

For spatial features, we apply the Random Forest learner, which is one of the most widely used classification methods. It builds a multitude of decision trees at training time and outputs the class of the voting result from the individual trees. Random Forest splits the trees based on the information gain, therefore it can better reflect the impact of discrete values. Figure 3 illustrates the training of the Random Forest model in our approach. Given the spatial data, an ensemble of trees (total 128) are trained. The results $h_1, h_2, ..., h_n$ are concatenated into a vector V_s and fed to a majority voting module, producing the final classification result.

3.3 Phase 2 Training

In Phase 2 training, we construct a prediction model to predict the failure-proneness of nodes in near future. Different nodes have different likelihood of failing. In this step, we formulate the prediction problem as a ranking problem. That is, instead of simply telling whether a node is faulty or not, we rank the nodes by their probability of being faulty. By giving relative order as opposed to a binary label, the results of a ranking algorithm can distinguish between different degrees of failure-proneness. The ranking results better serve the goal of VM allocation (allocating a VM to a healthier node) and on-demand live migration (migrating a VM from a faulty node to a healthier nodes). Furthermore, as the ranking approach is effective in optimizing the top k results, it could work better in the case of highly imbalanced data.

In this phase, we take the intermediate output vectors produced by the base learners as the input vector to the ranking model. More specifically, from the LSTM model, we use the output vector V_t of the Dense layer. From the Random Forest model, we use the output vector V_s produced by the trees. We then concatenate the two output vectors and form a *256 x 1* input vector V for the ranking model.

To train a ranking model, we obtain the historical failure data about the nodes, and rank the nodes according to the frequency and duration of failures. We adopt the concept of Learning to Rank [24], which automatically learns an optimized ranking model to minimize the cost of disorder, especially the cost for the top results (similar to the optimization of the top results in a search engine). Specifically, LambdaMART [4] is used here, which is the boosted tree version of learning-to-rank algorithm. It has proven to be a very successful algorithm and won the 2010 Yahoo! Learning To Rank Challenge (Track 1) [4].

3.4 Cost-sensitive Thresholding

To improve service availability, we would like to intelligently allocate VMs to the healthier nodes so that these VMs are less likely to suffer from node failures in near future. We also propose to proactively migrate live VMs residing on high-risk nodes to healthy nodes. To achieve so, we apply cost-sensitive thresholding to identify the faulty nodes for live migration.

As most of the nodes are healthy and only a small percentage of nodes are faulty, we select the top r nodes returned by the ranking model as the faulty ones. The optimal top r nodes are selected with historical data to minimize the total misclassification cost:

$$r = \arg\min_{r}(CostRatio * FP_r + FN_r),$$

where FP_r and FN_r are the number of false positives and false negatives in the top r predicted results, respectively. We denote by $Cost1$ the cost of failing to identify a faulty node (false negatives). We denote by $Cost2$ the cost of wrongly identifying a healthy node as faulty (false positives), which involves the cost of unnecessary live migration from the "faulty" node to a healthy node. We define $CostRatio$ as a ratio $Cost2/Cost1$. The value of $CostRatio$ is estimated by experts in product teams. In current practice, due to the concerns about cost and capacity, $CostRatio$ is set to 2 (i.e., precision is valued more than recall). The optimum r value is determined by minimizing the total misclassification cost with historical data. The top r nodes are predicted faulty nodes. They are high risky nodes and the VMs hosted on them should be migrated out.

4 EVALUATION

4.1 Research Questions

In this section, we evaluate our approach using real-world data. We aim at answering the following research questions:

RQ1: How effective is the proposed approach in predicting node failures in a cloud service system?

In this RQ, we evaluate the overall effectiveness of MING in predicting node failures in a cloud service system. As classification methods are commonly used in fault prediction, we also compare MING with the baseline approaches that are implemented using conventional classifiers including Logistic Regression, Random Forest, LSTM, and SVM. To perform the comparison, before applying the conventional classifiers, we convert features to the feature types that can be consumed by the target classifiers (e.g., converting the categorical data into numerical data). We then construct classification models and compare the prediction results with those returned by the proposed approach.

RQ2: Are the temporal and spatial features useful for failure prediction ?

As described in Section 2, we collect both temporal and spatial data for node failure prediction. In Phase 1 training, we construct a LSTM and a Random Forest based prediction model to incorporate the temporal and spatial features, respectively. In this RQ, we evaluate the usefulness of each type of the features. To answer this RQ, we train the prediction mode using temporal and spatial data

separately, and compare the results with those achieved by MING (which uses both types of data). To train the prediction model with temporal data alone, only the LSTM model described in Section 3 is needed. To train the prediction model with spatial data alone, only the Random Forest model is needed. The rest of the settings remain the same.

RQ3: Is the proposed ranking method effective ?

As described in Section 3, we propose a ranking method to rank nodes by the fault probability (Section 3.3). In this RQ, we evaluate if the proposed ranking method is effective by measuring the effectiveness of MING that replaces the ranking model with a conventional classification model (denoted as $MING_c$). To answer the RQ, in $MING_c$, we replace the ranking model with SVM, Logistic Regression, and Random Forest classifier, separately. The input to the classifiers are still the combination of output vectors produced by Phase 1 training. The rest of the settings remain the same between MING and $MING_c$. To enable comparison, the output of the classifiers are ranked by the probability values.

In MING, we use LambdaMART as the ranking algorithm. To show that MING is actually independent of a specific ranking algorithm, we also experiment with a variant of MING, which replaces LambdaMART with FastTree Ranker. The FastTree algorithm [12, 26] is an implementation of FastRank. It builds each regression tree (which is a decision tree with scalar values in its leave) in a step wise fashion. We will compare the effectiveness of MING with the two different rankers.

4.2 Evaluation Setup

4.2.1 Dataset. To evaluate the proposed approach, we collect data from a production cloud service system. For offline training, we collect over three month datasets and each dataset covers one month period in 2017. The data are from part of the data centers, containing over half a million of physical cloud computing nodes. All faulty nodes over the period are selected as positive samples, while healthy nodes (negative samples) are randomly selected with a 1:20 sampling rate. We use three datasets collected over three 7-day periods after the training periods for testing. Note that for all nodes, the feature data is collected at least 6 hours before the class label data is collected. The 6-hour gap is intended to simulate real-world usage (predicting node failures using the signals collected before the failures actually happen). Table 2 summarizes the datasets used in this experiment. Note that we did not apply the commonly-used cross validation, because cross validation may not reflect the real results of prediction, and sometimes may overclaim the effectiveness. More details will be discussed in Section 5.2.

Table 2: The experimental data

	Training Period	Test Period
Dataset 1	04/01/2017 - 04/30/2017	05/01/2017 - 05/07/2017
Dataset 2	05/01/2017 - 05/31/2017	06/01/2017 - 06/07/2017
Dataset 3	06/01/2017 - 06/30/2017	07/01/2017 - 07/07/2017

4.2.2 Tool Implementation. We implemented the proposed approach by leveraging the various components provided by AzureML[4], which is a production environment for development and deployment of machine learning models. The experimental evaluation is running on a Windows Server 2012 with (Intel CPU E5-4657L v2 @2.40GHz 2.40 with 500 GB Memory).

[4]https://studio.azureml.net/

4.2.3 Evaluation Metrics. Most of existing classification-based prediction models use Precision/Recall/F1 measure to evaluate the effectiveness of the models. In our experiments, we also use these metrics as evaluation metrics (although there are more metrics for evaluating a ranking-based model). Precision measures the percentage of identified faulty nodes that are actually faulty. Recall measures the percentage of faulty nodes that are correctly identified over all the faulty nodes. F1 measure is the harmonic mean of precision and recall, which weights recall and precision equally.

MING also ranks the nodes according to their failure-proneness. To evaluate the ranking ability of MING, we compute the Precision@k values, which are the Precision value for top k nodes (k = 10, 20, 50, 100, 200, and 500). An ideal failure prediction model should be able to correctly identify the failure-prone nodes in the top k returned results. Thus, the higher the metric value, the better the prediction performance, especially in actual practice when the cost of false positive is higher than false negative, as discussed in Section 3.4.

4.3 Evaluation Results

RQ1: How effective is the proposed approach in predicting node failures in a cloud service system?

Table 3 shows the evaluation results of MING in identifying faulty nodes (the top r returned nodes). MING achieves good results on all datasets and outperforms the baseline methods. The average precision, recall, and F1-measure values are 92.4%, 63.5%, and 75.2%, respectively. Considering the highly imbalanced nature of data and the complexity of the problem, it is very challenging to achieve both high recall and high precision. In our scenario, precision is more important than recall, therefore we tradeoff some recall to achieve higher precision. Our results show that MING outperforms the baseline approaches that are implemented in conventional classification algorithms (SVM, Logistic Regression, Random Forest, and LSTM) in both precision and recall. The average absolute improvement in F1-measure over Logistic Regression, SVM, Random Forest, and LSTM is 21.7%, 17.4%, 13.3%, and 14.6% respectively.

We also evaluate the ranking ability of MING by examining the top k returned results. Figure 4 shows the Precision@k values. For the top 10, 20 and 50 returned nodes, the precision values achieved by MING on all datasets are close to 100%. When the top 500 returned nodes are examined, the precision values are still higher than 92.0% on all datasets. The results show that MING can effectively rank the faulty nodes and consistently achieve high precision. We also compare the Precision@k results of MING with those of the classification algorithms (SVM, Logistic Regression, Random Forest, and LSTM). To enable the comparison, we rank the probability values returned by each classification algorithm. The results are also shown in Figure 4. MING outperforms the classification algorithms consistently, when Precision@k is concerned.

In MING, the faulty nodes are identified by selecting the optimal set of top r nodes that minimize the total misclassification cost. We also experimented with different thresholds (the r values) that

Table 3: The effectiveness of MING

	MING			Logistic Regression (LR)			SVM			Random Forest			LSTM		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Dataset 1	92.3%	64.2%	75.7%	69.8%	48.3%	57.1%	66.9%	53.4%	59.4%	71.6%	51.1%	59.6%	76.2%	52.3%	62.0%
Dataset 2	90.1%	67.3%	77.0%	78.6%	34.7%	48.1%	54.8%	61.1%	57.8%	80.6%	58.3%	67.7%	61.7%	60.4%	61.0%
Dataset 3	94.7%	59.1%	72.8%	59.7%	51.3%	55.2%	76.2%	44.6%	56.3%	76.3%	47.4%	58.5%	80.3%	46.3%	58.7%
Average	92.4%	63.5%	75.2%	69.4%	44.8%	53.5%	66.0%	53.0%	57.8%	76.2%	52.3%	61.9%	72.7%	53.0%	60.6%

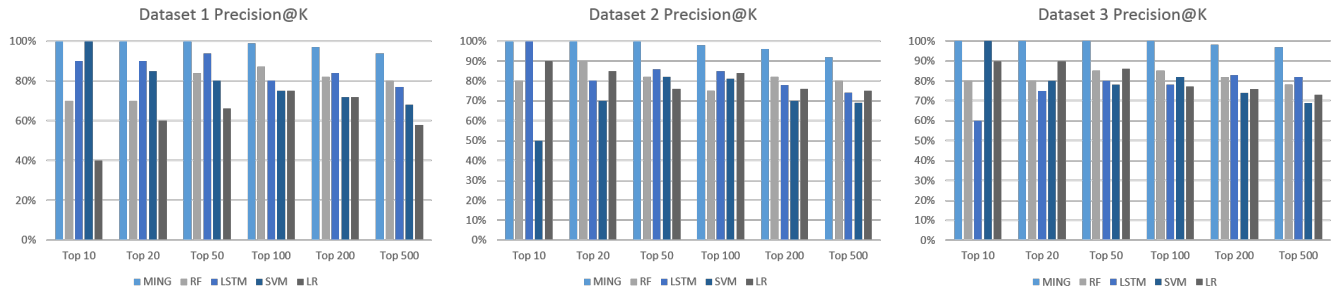

Figure 4: The Precision of the top k returned nodes

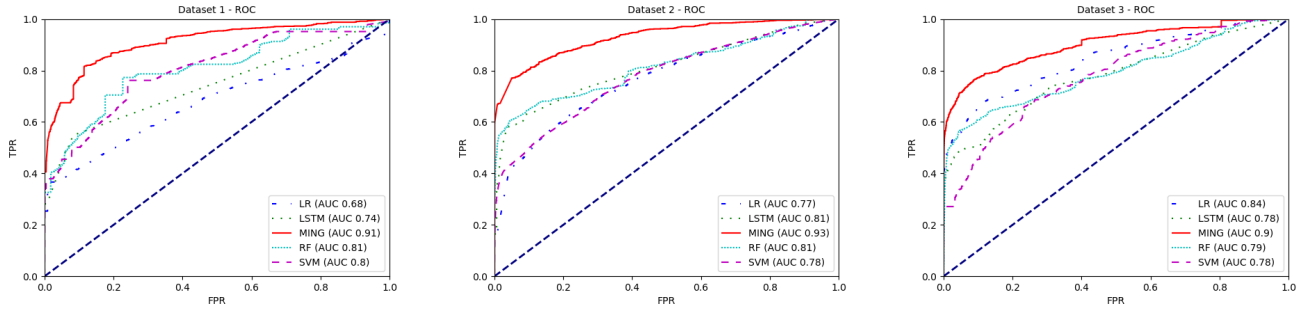

Figure 5: The ROC curve of the comparative approaches

classify faulty and healthy classes. The results are shown in Figure 5, where the ROC curve[5] plots TPR (True Positive Rate) versus FPR (False Positive Rate) with a varying threshold value. The results show that MING outperforms the baseline approaches consistently under different FPR/TPR ratios. For example, on Dataset 1, the AUC (Area Under Curve) value achieved by MING is 0.91, while the value for Logistic Regression, SVM, Random Forest and LSTM are 0.68, 0.80, 0.81, and 0.74, respectively.

In summary, the experimental results show that the proposed approach is effective in predicting node failures in a cloud service system and outperforms the baseline methods.

RQ2: Are the temporal and spatial features useful?

MING utilizes two base learners (Random Forest and LSTM) to incorporate the temporal and spatial features, respectively. In this RQ, we evaluate the usefulness of each type of the features. The results are shown in Table 4.

If both types of the features are used, MING achieves an average F1-measure of 75.2%. If the temporal features are used alone, only the LSTM model is trained and the average F1-measure drops from 75.2% to 48.8%. If the spatial features are used alone, only the Random Forest model is trained and the average F1-measure drops from 75.2% to 57.1%. We can see the ensemble model adopted by MING achieves the best overall results. Also, the results achieved by spatial features are higher that those achieved by temporal features, indicating that the spatial features are more predictive than the temporal features.

In summary, the experimental results show that both types of features are useful for node failure prediction, and the spatial features have more predictive power. The results also confirm that each base learner is useful and the proposed ensemble model is more effective.

RQ3: Is the proposed ranking method effective ?

In this RQ, we evaluate if the propose ranking method is effective by comparing MING (the proposed approach with the ranking model) and $MING_c$ – the variant of MING that replaces the ranking

[5]https://en.wikipedia.org/wiki/Receiver_operating_characteristic

Table 4: The effectiveness of the ensemble model

	Temporal+Spatial (MING)			Temporal only (LSTM)			Spatial only (Random Forest)		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Dataset 1	92.3%	64.2%	75.7%	70.6%	36.2%	47.9%	66.5%	49.3%	56.6%
Dataset 2	90.1%	67.2%	77.0%	63.8%	46.7%	53.9%	72.1%	54.7%	62.2%
Dataset 3	94.7%	59.1%	72.8%	51.4%	39.6%	44.7%	79.6%	39.1%	52.4%
Average	92.4%	63.5%	75.2%	61.9%	40.8%	48.8%	72.7%	47.7%	57.1%

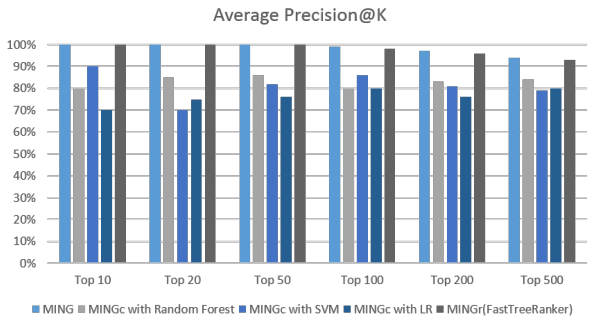

Figure 6: Ranking vs. Classification (average Precision@k)

model with a conventional classifier such as Random Forest (RF), SVM, and Logistic Regression (LR). The results are shown in Table 5. Clearly, MING outperforms $MING_c$ with all classifiers in both precision and recall. We also evaluate the accuracy of the top k returned results. Figure 6 shows the average Precision@k values on all the three datasets. Clearly, MING outperforms $MING_c$ under all k values. Furthermore, $MING_r(FastTreeRanker)$ - the MING variant with the FastTree Ranker also outperforms $MING_c$ with all classifiers and achieves comparable performance with MING. This indicates that the results achieved by MING is independent of a specific ranker that we have chosen. It is the general design of a ranking model in our approach that helps improve the effectiveness of MING.

Table 5: The effectiveness of the ranking model

	Precision	Recall	F1
MING	92.4%	63.5%	75.2%
$MING_c$ with Random Forest	85.1%	57.6%	68.7%
$MING_c$ with SVM	79.3%	56.7%	66.1%
$MING_c$ with LR	81.4%	52.6%	63.9%
$MING_r(FastTreeRanker)$	91.3%	62.4%	74.1%

4.4 Discussions

4.4.1 Why Does MING Work. To build an effective prediction model, we collected more than 210 features from a wide variety of data sources. The features identified from temporal and spatial data contain early signals of node failures. The prediction model is trained using a large amount of real-world data. It is widely known that a traditional machine learning algorithm works better on a certain type of features, while performs weakly on other types of features. To work with all types of features, feature conversion (like converting categorical features into numerical features) needs to be performed before training a model. However, a lot of information could be lost during the conversion process, thus decreasing the accuracy. As shown by our experimental results, the two base models in MING (the LSTM model and the Random Forest model) can better capture the characteristics of temporal and spatial features, therefore producing better results.

MING embeds the intermediate output of the two base models as the feature input for a ranking model and formulates the node failure prediction problem as a ranking problem. By optimizing the order of failure-proneness, the faulty nodes with higher failure-proneness are raised to the top, resulting in the high accuracy of the predicted results. The order can be utilized in VM allocation and live migration practice as VMs can be moved to a much healthier node. Furthermore, traditional classification algorithms are naturally sensitive to the distribution of classes (as they optimize to split the data instances into classes), while a ranking-based method like MING focuses on the order of data instances and is therefore less sensitive to the imbalanced class problem.

4.4.2 Evaluation Metrics. Much research work [10, 45, 46] show that in a large software system, the distribution of faults is skewed - that a small number of modules accounts for a large proportion of the faults. In our work, we find that the distribution of faulty nodes is also skewed. The ratio between faulty and healthy nodes could be as high as 1:1000. The highly imbalanced data imposes challenges for failure prediction. In general, it is difficult for a machine learning technique to distinguish a small number of faulty modules from a large number of modules.

The highly imbalanced failure data also has implications on evaluation metrics. Each day, out of all the computing nodes in the cloud service system we studied, less than 0.1% of the nodes encounter failures. While the failure rate of 0.1% may seem insignificant, the absolute number of failed nodes is significant as the total number of nodes is very large. Therefore, the 0.1% failure rate has significant adverse impact on service availability.

In literature, Zhang [47] also pointed out that prediction results may not be always satisfactory in the presence of imbalanced data distribution. They found that high probability of detection (pd, i.e., true-positive rate) and low probability of false alarm (pf, i.e., false-positive rate) do not necessarily lead to high precision. The reason is that the percentage of faulty modules could be very small. The Zhangs' equation for Precision is defined as follows:

$$Precision = \frac{TP}{TP + FP} = \frac{1}{1 + \frac{FP}{TP}} = \frac{1}{1 + \frac{NEG * PF}{POS * PD}}, \quad (1)$$

where NEG is the number of negative instances and POS the number of positive instances. From the Equation (1), we can see that even if pd is high and pf is low, the $Precision$ would be low if the number

of negative instances (NEG) is much more than the number of positive instances (POS). Therefore, the metrics pf, pd, and the ROC curves should be used with caution. In our study, we show Recall/Precision values as well as the ROC curves to confirm the effectiveness of the proposed approach.

4.4.3 Oversampling. Before constructing a prediction model, one could apply an imbalanced data handling approach, such as SMOTE [6], to balance the data. SMOTE is a commonly used oversampling technique in which the minority class is over-sampled by creating "synthetic" examples through finding k-nearest neighbors along the minority class. We tried to apply SMOTE to our approach but did not get promising results. This is because after balancing there are still many false positives due to the highly imbalanced data.

4.4.4 Parameter Settings. In the experiments and comparative evaluations described in Section 4.3, we use the default settings of the machine learning algorithms. It has been observed that there is a bias in the comparison between different algorithms with default parameter settings [35]. To evaluate the impact of parameter settings, we have also experimented with different values of several important parameters: a) iterate the "output vector size" parameter used by LSTM and Random Forest from 100 to 150 with a step of 10, b) iterate "the number of boosting iterations" parameter used by LambdaMART from 100 to 150 with a step of 10, c) iterate the "number of leaves" parameter used by Random Forest from 20 to 100, with a step of 20. With different parameter settings, the resulting average F1 scores on three datasets are quite stable - ranging from 74.0% to 76.1% (with a delta of -1.2% to +0.9% compared to the average F1-measure of 75.2% shown in Table 3). These results show that MING is insensitive to parameter settings.

4.4.5 Threats to Validity. We have identified the following threats to validities:

Subject systems: In our experiments, we only collect data from one cloud service system of one company. Therefore, our results might not be generalizable to other systems. However, it is challenging to get access to data from many cloud systems. The system we studied is a typical, large-scale cloud service system, from which sufficient data can be collected. Furthermore, we have applied our approach in the maintenance of an actual cloud service system. In future, we will reduce this threat by evaluating MING on more subject systems and report the evaluation results.

Evaluation metrics: We used the Precision/Recall/F-measure metrics to evaluate the prediction performance. These metrics have been widely used to evaluate the effectiveness of a prediction model. Prior work [36] points out that a broader selection of metrics should be used in order to maximize external validity. In our future work, we will reduce this threat by experimenting with more evaluation measures such as the cumulative lift chart (CLC) and the fault-percentile-average (FPA) metrics used in [43].

5 SUCCESS STORIES

5.1 Success Stories

We have successfully applied MING to the maintenance of Service X, which is a large-scale cloud service system in Microsoft. Service X allows developers and IT professionals to build, deploy, and manage applications. The cloud service achieves global scale on a worldwide network of data centers across many regions.

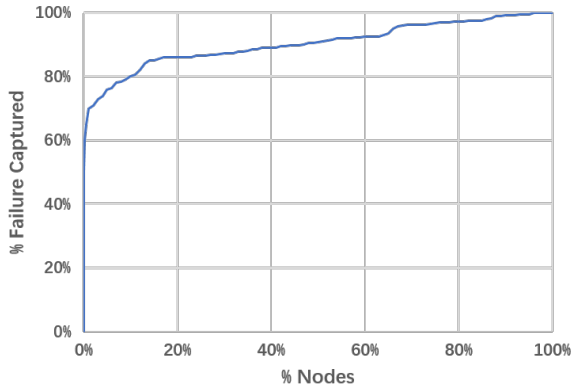

Figure 7: The %failures captured vs. the %nodes examined

MING is currently used by Service X to preferentially select healthier nodes for VM allocation. The current in-production model is trained and scored using AzureML. The automation relies on the AzureML batch web service feature. We invoke daily jobs across the entire Azure stack to refresh daily the failure-proneness scores of all the nodes in Service X.

After deploying MING, the product team computes the percentage of failures captured by the nodes that are ranked by their failure-proneness score. The results are shown in Figure 7. In a typical day, the top 1% most failure-prone nodes predicted by MING capture above 60% of the failures in the next day. The product team also conducted an A/B testing in a large-scale cloud environment. The results show that MING is able to intelligently allocate VMs to more healthier nodes and has achieved above 30% reduction in these new allocated VMs' failure rate.

The ability to predict node failure in cloud service systems also helped product teams diagnosis service problems. We work with domain experts in product teams to identify the influential features using feature selection methods, and help product teams analyze the root causes of the node failures based on the influential features. For example, we found that upgrading system software to a specific version caused a potential node failure, and some configuration changes to the cluster also resulted in node failure. We even found that, the nodes on the upper layer rack have much higher failure probability than the nodes on the lower rack (as the hot air rises up to the top rack).

5.2 Lessons Learned

Freshness of the training data: Extending the training period does not bring many benefits to the prediction accuracy. Currently we use one month data for training. Extending the length of the training period can only improve the results slightly. The reason is that, the cloud service systems are evolving rapidly and the fault patterns vary a lot over time. Also, new applications, new deployment, and new versions happen frequently and could introduce new faults that are previously unseen. Learning from very old data will not have much gain in the prediction, as some old fault patterns could have vanished. Therefore, in practice we need to train the

model frequently (e.g., daily) with new data, in order to catch the new fault patterns emerged recently.

The time gap between data and label: As we are dealing with a prediction problem, when training the model, we need to take care of the time gap between the collection of feature data and the collection of labels. If we did not set the time gap between them, the model could learn from some feature data coming with the failures. However, in the case of failure prediction, we need features that are early indicators of failures. The feature data collected at the time of failure may not have predictive power. In practice, to secure the time gap, we collect feature data at least 6 hours before the collection of labels. In this way, we always learn from early signals.

Cross-validation and online prediction: Cross-validation is often used to evaluate machine learning models. A k-fold cross-validation randomly divides a data set into k partitions and uses $k-1$ partitions to train the prediction model and the remaining 1 partition to test the model. Therefore, it is possible that knowledge that should not be known at the time of prediction is utilized in cross-validation. For example, an incident may cause many nodes to fail around the same time. In cross-validation, data about these new failures may be randomly selected for training, which increases the prediction accuracy in testing for the nodes affected by the same incident around the same time. Therefore, cross-validation is not suitable for evaluating our model in practice, even though it can lead to better results than online prediction. In real-world online prediction, training and testing data are strictly split by time and the testing period is always after the training period. Therefore, the characteristics of data appear in the future is not used for training. The problem of cross-validation in an online prediction scenario is also observed by others [34].

6 RELATED WORK

6.1 Failure Prediction

In recent years, we have witnessed a lot of interest in developing software defect prediction models [5, 8, 17, 18, 21, 33, 43, 44]. It is widely believed that some internal properties of software (e.g., metrics) have relationship with the external properties (e.g., quality). Software defect prediction refers to building a prediction model for new software modules using historical metric and fault data collected from existing projects. For example, Menzies et al. [25] performed an extensive study on 8 NASA datasets using three classification techniques with 38 static code metrics. Nam et al.[29] proposed a heterogeneous defect prediction method that matches up different metrics in different projects. Jing et al. [21] proposed a heterogeneous defect prediction method based on Unified Metric Representation and Canonical Correlation Analysis. Kim et al. [22] addressed the data quality issue in software defect prediction and found that a small degree of data noise does not affect the prediction results significantly. Recently, deep learning techniques have been applied to software defect prediction as well [39, 42].

There are also related work on predicting disk failures and computing system failures [3, 13, 30, 31, 41]. For example, Pinheiro [30] attempted to find variables that may be used to predict disk failures from observations of a large disk drive population in a production Internet services deployment. Gaber et al. [13] used machine learning algorithms to extract compound features representing the behavior of the drives and predict the failure of the drives. Xu et al. [41] utilized both disk-level sensor data and system-level signals for predicting disk errors in cloud systems.

Our work is about node failure prediction for a cloud service system, which has a larger scope and requires analyzing more heterogeneous features. Node failure can be triggered by any software or hardware issue, or a mixture of both, which brings new challenges compared to software/disk failure prediction.

6.2 Analysis of Failures in Cloud Systems

There have been some previous studies in the literature on failures of a data center. For example, Ford et al. [11] studied data collected from Google storage systems over a one year period , and characterized the sources of faults contributing to unavailability. Their results indicate that cluster-wide failure events should be paid more attention during the design of system components, such as replication and recovery policies. Gill et al. [14] presented a large-scale analysis of failures in a data center network. They characterized failure events of network links and devices, estimated their failure impact, and analyzed the effectiveness of network redundancy in masking failures. Zhou et al. [48] performed an empirical study on the quality issues of a production big data platform used in Microsoft. They analyzed 210 real service quality issues and investigated the common symptom, causes and mitigation solutions. Their finding shows that 21.0% of escalations are caused by hardware faults and 36.2% are caused by system side defects.

Several methods [40] have also been proposed to detect node failures in a cloud data center based on the network structure. Besides, there have been studies on detecting the "gray" failures, which are "component failures whose manifestations are fairly subtle and thus defy quick and definitive detection" [20]. These approaches provide solutions to detect node failures and improve the service quality, but they did not provide systematic methods for predicting node failure in cloud service systems.

7 CONCLUSION

To maintain and improve service availability of a cloud service systems, we propose MING, a node failure prediction approach. Using MING, we can intelligently allocate/migrate VMs to the healthier nodes so that these VMs are less likely to suffer from node failures. We propose an ensemble of machine learning models to combine heterogeneous data from diverse sources. To better handle the highly imbalanced data, we rank the nodes according to their failure-proneness and select the top r nodes that minimize the misclassification cost. We have evaluated the proposed approach using real-world data and have successfully applied MING to the maintenance of a production cloud service system. We believe that given the importance of service availability, failure predictors will play increasingly important roles in the design and maintenance of cloud service systems. Our proposed approach is an important step in this direction.

ACKNOWLEDGEMENT

We specially thank our product team partners Ervin Peretz, Geoffrey Goh, David Dion, Bertus Greeff, John Miller, Girish Bablani for the collaboration and suggestion, and our intern students Pu Zhao, Yuchen Sun, Wenchi Zhang for the development and experiments.

REFERENCES

[1] Amazon. 2017. Amazon EBS Product Details. https://aws.amazon.com/ebs/details/. [Online; accessed 23-Oct-2017].

[2] David Ameller, Matthias Galster, Paris Avgeriou, and Xavier Franch. 2016. A Survey on Quality Attributes in Service-based Systems. *Software Quality Journal* 24, 2 (June 2016), 271–299.

[3] Mirela Madalina Botezatu, Ioana Giurgiu, Jasmina Bogojeska, and Dorothea Wiesmann. 2016. Predicting Disk Replacement towards Reliable Data Centers. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 39–48.

[4] Christopher J.C. Burges. 2010. From RankNet to LambdaRank to LambdaMART: An Overview. In *Microsoft Research Technical Report MSR-TR-2010-82*. Microsoft.

[5] Gerardo Canfora, Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella. 2015. Defect prediction as a multiobjective optimization problem. *Software Testing, Verification and Reliability* 25, 4 (2015), 426–459.

[6] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2002. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research* 16 (2002), 321–357.

[7] Christopher Clark, Keir Fraser, Steven H, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. 2005. Live Migration of Virtual Machines. In *In Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 273–286.

[8] Marco D'Ambros, Michele Lanza, and Romain Robbes. 2012. Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empirical Software Engineering* 17, 4-5 (2012), 531–577.

[9] Florin Dinu and T.S. Eugene Ng. 2012. Understanding the Effects and Implications of Compute Node Related Failures in Hadoop. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing (HPDC'12)*. 187–198.

[10] Norman E. Fenton and Niclas Ohlsson. 2000. Quantitative Analysis of Faults and Failures in a Complex Software System. *IEEE Transactions on Software Engineering* 26, 8 (Aug. 2000), 797–814.

[11] Daniel Ford, François Labelle, Florentina I. Popovici, Murray Stokely, Van-Anh Truong, Luiz Barroso, Carrie Grimes, and Sean Quinlan. 2010. Availability in Globally Distributed Storage Systems. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*. USENIX Association, Berkeley, CA, USA, 61–74.

[12] Jerome H. Friedman. 2000. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics* 29 (2000), 1189–1232.

[13] Shiri Gaber, Oshry Ben-Harush, and Amihai Savir. 2017. Predicting HDD Failures from Compound SMART Attributes. In *Proceedings of the 10th ACM International Systems and Storage Conference (SYSTOR '17)*. ACM, Article 31, 31:1–31:1 pages.

[14] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. 2011. Understanding network failures in data centers: measurement, analysis, and implications. In *SIGCOMM*. 350–361.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[16] Georgios Gousios and Diomidis Spinellis. 2009. Alitheia Core: An Extensible Software Quality Monitoring Platform. In *ICSE '09: Proceedings of the 31st International Conference on Software Engineering — Formal Research Demonstrations Track*. IEEE, 579–582.

[17] Jeremy Greenwald, Tim Menzies, and Art Frank. 2007. Data Mining Static Code Attributes to Learn Defect Predictors. *IEEE Transactions on Software Engineering* 33 (2007), 2–13.

[18] Tracy Hall, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. 2012. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering* 38, 6 (2012), 1276–1304.

[19] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (Nov. 1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[20] Peng Huang, Chuanxiong Guo, Lidong Zhou, Jacob R. Lorch, Yingnong Dang, Murali Chintalapati, and Randolph Yao. 2017. Gray Failure: The Achilles' Heel of Cloud-Scale Systems. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems (HotOS '17)*. ACM, New York, NY, USA, 150–155.

[21] Xiao-Yuan Jing, Fei Wu, Xiwei Dong, Fumin Qi, and Baowen Xu. 2015. Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning. In *FSE*. 496–507.

[22] Sunghun Kim, Hongyu Zhang, Rongxin Wu, and Liang Gong. 2011. Dealing with Noise in Defect Prediction. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE '11)*. ACM, New York, NY, USA, 481–490.

[23] Zheng Li, He Zhang, Liam O'Brien, Rainbow Cai, and Shayne Flint. 2013. On evaluating commercial Cloud services: A systematic review. *Journal of Systems and Software* 86, 9 (2013), 2371–2393.

[24] Tie-Yan Liu. 2009. Learning to Rank for Information Retrieval. *Found. Trends Inf. Retr.* 3, 3 (March 2009), 225–331.

[25] Tim Menzies, Jeremy Greenwald, and Art Frank. 2007. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering* 33, 1 (2007), 2–13.

[26] Microsoft. 2017. Machine Learning Fast Tree. https://docs.microsoft.com/en-us/machine-learning-server/python-reference/microsoftml/rx-fast-trees

[27] Microsoft. 2017. Microsoft Azure. https://azure.microsoft.com/en-au/services/storage/unmanaged-disks/. [Online; accessed 23-Oct-2017].

[28] Ivan Mistrik, Rami Bahsoon, Peter Eeles, Roshanak Roshandel, and Michael Stal. 2014. *Relating System Quality and Software Architecture* (1st ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[29] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan. 2017. Heterogeneous Defect Prediction. *IEEE Transactions on Software Engineering* PP, 99 (2017), 1–1. https://doi.org/10.1109/TSE.2017.2720603

[30] Eduardo Pinheiro, Wolf-Dietrich Weber, and Luiz André Barroso. 2007. Failure Trends in a Large Disk Drive Population. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST '07)*. USENIX Association, Berkeley, CA, USA, 2–2.

[31] Teerat Pitakrat, André van Hoorn, and Lars Grunske. 2013. A Comparison of Machine Learning Algorithms for Proactive Hard Disk Drive Failure Detection. In *Proceedings of the 4th International ACM SIGSOFT Symposium on Architecting Critical Systems (ISARCS '13)*. ACM, New York, NY, USA, 1–10. https://doi.org/10.1145/2465470.2465473

[32] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'14)*. MIT Press, Cambridge, MA, USA, 3104–3112. http://dl.acm.org/citation.cfm?id=2969033.2969173

[33] Mark Syer, Meiyappan Nagappan, Bram Adams, and Ahmed E. Hassan. 2015. Replicating and Re-evaluating the Theory of Relative Defect-Proneness. *IEEE Transactions on Software Engineering* 41, 2 (2015), 176–197.

[34] Ming Tan, Lin Tan, Sashank Dara, and Caleb Mayeux. 2015. Online Defect Prediction for Imbalanced Data. In *Proceedings of the 37th International Conference on Software Engineering - Volume 2 (ICSE '15)*. IEEE Press, Piscataway, NJ, USA, 99–108. http://dl.acm.org/citation.cfm?id=2819009.2819026

[35] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto. 2016. Automated Parameter Optimization of Classification Techniques for Defect Prediction Models. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. 321–332.

[36] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto. 2016. Comments on 'Researcher Bias: The Use of Machine Learning in Software Defect Prediction'. *IEEE Transactions on Software Engineering* 42, 11 (Nov 2016), 1092–1094.

[37] Scott Tilley. 2012. *Software Testing in the Cloud: Perspectives on an Emerging Discipline: Perspectives on an Emerging Discipline*. IGI Global.

[38] Kashi Venkatesh Vishwanath and Nachiappan Nagappan. 2010. Characterizing cloud computing hardware reliability. In *SOCC*. ACM, New York, NY, USA, 193–204.

[39] Song Wang, Taiyue Liu, and Lin Tan. 2016. Automatically learning semantic features for defect prediction. In *ICSE*. 297–308.

[40] Lei Xu, Wenzhi Chen, Zonghui Wang, Huafei Ni, and Jiajie Wu. 2012. *Smart Ring: A Model of Node Failure Detection in High Available Cloud Data Center*. Springer Berlin Heidelberg, Berlin, Heidelberg, 279–288.

[41] Yong Xu, Kaixin Sui, Randolph Yao, Hongyu Zhang, Qingwei Lin, Yingnong Dang, Peng Li, Keceng Jiang, Wenchi Zhang, Jian-Guang Lou, Murali Chintalapati, and Dongmei Zhang. 2018. Improving Service Availability of Cloud Systems by Predicting Disk Error. In *2018 USENIX Annual Technical Conference, USENIX ATC 2018, Boston, MA, USA, July 11-13, 2018*. 481–494.

[42] X. Yang, D. Lo, X. Xia, Y. Zhang, and J. Sun. 2015. Deep Learning for Just-in-Time Defect Prediction. In *2015 IEEE International Conference on Software Quality, Reliability and Security*. 17–26. https://doi.org/10.1109/QRS.2015.14

[43] X. Yang, K. Tang, and X. Yao. 2015. A Learning-to-Rank Approach to Software Defect Prediction. *IEEE Transactions on Reliability* 64, 1 (March 2015), 234–246.

[44] Feng Zhang, Audris Mockus, Iman Keivanloo, and Ying Zou. 2016. Towards building a universal defect prediction model with rank transformed predictors. *Empirical Software Engineering* 21, 5 (2016), 1–39.

[45] Hongyu Zhang. 2008. On the Distribution of Software Faults. *IEEE Transactions on Software Engineering* 34, 2 (March 2008), 301–302.

[46] Hongyu Zhang. 2009. An investigation of the relationships between lines of code and defects. In *2009 IEEE International Conference on Software Maintenance*. 274–283.

[47] Hongyu Zhang and Xiuzhen Zhang. 2007. Comments on "Data Mining Static Code Attributes to Learn Defect Predictors". *IEEE Transactions on Software Engineering* 33, 9 (2007), 635–637.

[48] H. Zhou, J. G. Lou, H. Zhang, H. Lin, H. Lin, and T. Qin. 2015. An Empirical Study on Quality Issues of Production Big Data Platform. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2. 17–26.