WIRELESS COMMUNICATIONS AND MOBILE COMPUTING
*Wirel. Commun. Mob. Comput.* 2016; **16**:3455–3465
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/wcm.2770

RESEARCH ARTICLE

# Clustering hypervisors to minimize failures in mobile cloud computing

Berihun Fekade, Taras Maksymyuk and Minho Jo*

Department of Computer Convergence Software, Korea University, Sejong Metro City, South Korea

## ABSTRACT

Resource virtualization has become one of the key super-power mobile computing architecture technologies. As mobile devices and multimedia traffic have increased dramatically, the load on mobile cloud computing systems has become heavier. Under such conditions, mobile cloud system reliability becomes a challenging task. In this paper, we propose a new model using a naive Bayes classifier for hypervisor failure prediction and prevention in mobile cloud computing. We exploit real-time monitoring data in combination with historical maintenance data, which achieves higher accuracy in failure prediction and early failure-risk detection. After detecting hypervisors at risk, we perform live migration of virtual servers within a cluster, which decreases the load and prevents failures in the cloud. We performed a simulation for verification. According to the experimental results, our proposed model shows good accuracy in failure prediction and the possibility of decreasing downtime in a hypervisor service. Copyright © 2017 John Wiley & Sons, Ltd.

**KEYWORDS**
mobile cloud computing; hypervisor clustering; hypervisor failure prediction; virtual server; physical server

***Correspondence**
Minho Jo, Department of Computer and Information Science, Korea University, Sejong Metro City, Korea.
E-mail: minhojo@korea.ac.kr
A preliminary version of this paper appeared at ACM IMCOM 2016 in Danag, Vietnam, January 2016.

## 1. INTRODUCTION

At present, we face quick development and integration of information and communications technology. Emerging paradigms, such as mobile cloud computing [1] and the Internet of Things [2], have opened a wide range of new opportunities for industry and human recreational life. Stable mobile cloud computing power resides on remote physical servers that are interconnected to give higher performance. Mobile cloud computing provides almost everything as a service to users [3,4].

Virtualization combines cloud-based computational elements designed to control physical entities. Cloud systems can be applied in different areas, such as healthcare, civil infrastructure, energy-saving systems, industrial manufacturing, and transportation control. Apparently, these separate cloud systems may share the same resources, such as computing power, storage, and network infrastructure. In such cases, the load on physical resources will be non-uniform and highly unpredictable. An interesting issue is raised when heavy resource-sharing requests are made from separate systems with different requirements in terms of processing time and complexity. Such requests will result in a higher number of server failures and thus lower reliability of the entire cloud infrastructure.

Therefore, dynamic failure prediction is one of the most important issues in cloud computing systems [5]. Existing solutions to improve reliability are mostly based on statistical learning techniques that use very straightforward assumptions for server failure, such as thresholds for current physical server attributes like central processing unit (CPU) load, and random access memory (RAM) utilization. [6]. We propose a dynamic, failure-prediction heuristic algorithm by using real-time resource status as well as previous failure experience in order to improve overall reliability of the mobile cloud computing system. The proposed heuristic prediction model is carried out based on groups of attributes, such as CPU load, RAM load, and the load from user requests. In our model, each hypervisor monitors the real-time status of its underlying physical server, including CPU load, RAM load, data input/output (IO) rate, and number of user requests. We assume that one hypervisor server has one corresponding hypervisor. The hypervisor is responsible for creating and hosting multiple virtual servers on a physical server, as shown in Figure 1. In addition, the hypervisor provides virtual servers with access to resource pools in the physical server. When a failure occurs in the physical server, its corresponding hypervisor consequently fails, too. In other words, physical server failure means hypervisor failure. We use the terminology *hypervisor failure* instead of physical server failure, taking the viewpoint of service consumers.

In order to satisfy quality of experience, it is very important to always keep the number of hypervisor service failures to a minimum. Thus, in this paper, we propose an effective idea to minimize the number of hypervisor failures. We group hypervisors in order to establish a fail-safe mechanism, and our proposed system is based on grouping hypervisors for that fail-safe mechanism. When a physical server becomes unavailable, which causes its hypervisor to fail, the virtual servers belonging to the failed hypervisor can be migrated to another hypervisor (or physical server). The two hypervisors (physical servers) are clustered in advance, preparing for migration just in case either of them suffers a failure. One cluster can contain more than two hypervisors. As the number of requests increases, a virtual infrastructure manager (VIM) gathers the attributes of each individual physical server by using heartbeat messages [5].

We use the naive Bayes classifier to predict expected times of failure based on the current status of the physical server by using previous-failure records. We chose the naive Bayes classifier because it is effective with only few variables, simple to use, and it computes the multiplication of independent distributions in the data [7]. The naive Bayes classifier has shown higher performance, compared with complex neural networks and decision trees [8]. The data on previous failures are collected into groups by interval times between failure occurrences. For instance, if hypervisor service failure occurs at time *t,* our proposed model checks the average value of each server attribute at time *t-2* min (2 min ahead) and groups them as "2 min before failure" for individual hypervisors. The average value of attributes at between 2 and 5 min before failure will be grouped as "5 min before failure." In the same way, other failure interval times from the existing data will be grouped so that the status of attributes will be checked in real time within these groups.

When the failure interval time is predicted for each hypervisor, the VIM will check previous predicted times before failure to check whether the pattern is increasing or decreasing. If the prediction shows a continuously decreasing pattern, the hypervisor will be considered at risk of failure. In that case, the VIM will choose another hypervisor from the cluster with the lowest tendency toward failure. The alternatively selected hypervisor should contain identical physical resources and enough available resources. Then, the migration of virtual servers between hypervisors can be performed. This re-clustering operation of virtual servers inside each cluster minimizes the number of failures and provides stable mobile cloud performance. This is the main advantage of our work.

This paper is organized as follows. We address related work in Section 2. Section 3 offers a description of hypervisors and clustering resources. Section 4 provides a detailed description and a performance analysis of our proposed heuristic failure-prediction approach. In Section 5, we present our proposed hypervisor-clustering mechanism and virtual server live migration method based on prediction results. Finally, we conclude the paper in Section 6.

## 2. RELATED WORK

TinyChecker provides transparent failure detection and recovery of virtual servers from hypervisor failures [9]. The technique in the TinyChecker approach is to leverage a special-purpose hypervisor using nested virtualization for failure detection and recovery [9]. TinyChecker uses a new mechanism called context-aware checking to detect hypervisor failures that can occur because of wrongly updated CPU states and corrupted virtual server memory.

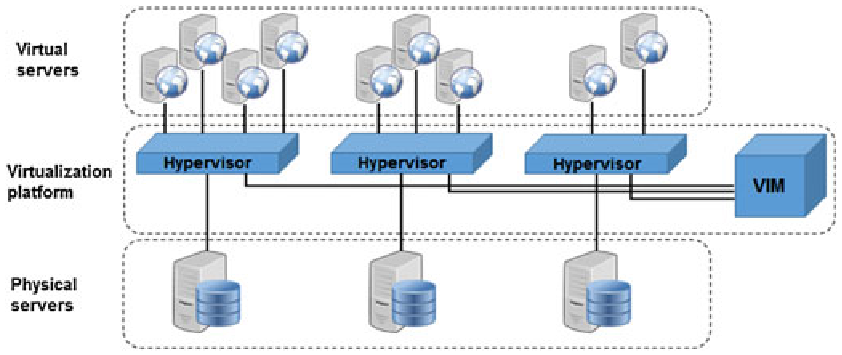

**Figure 1.** Resource virtualization and management in a mobile cloud computing system [4].

TinyChecker knows the context (clues or background) of hypervisor actions in order to detect failures. It only can detect failure of a hypervisor and recover that specific hypervisor from failure, whereas our proposed mechanism migrates the virtual servers in advance based on service failure prediction for the hypervisor.

Dabbagh and colleagues [10,11] presented issues in resource allocation challenges and presented some potential solutions to reduce cloud data–center energy consumption. The authors use the Google cluster for their proposed framework to determine the number of requests that will arrive in cloud data centers, plus the amount of CPU and memory resources. Their proposed system provides an accurate estimated number of physical servers that cloud data centers need in order to serve their clients. Zheng *et al.* [12] proposed online failure prediction for cloud computing systems using system runtime data. This is different from traditional tolerance techniques that require an in-depth knowledge of underlying mechanisms, such as cloud theory and the hidden Markov model.

The new virtual server live migration advancements in mobile cloud architectures allow virtual servers to migrate from one server to another without any downtime for the applications running inside [13]. This is good for load balancing.

## 3. HYPERVISORS AND CLUSTERING PHYSICAL RESOURCES

In mobile cloud computing systems, the physical servers are combined in a pool of interconnected networked environments to achieve more powerful computing performance for users. The mobile cloud is a service-oriented technology that delivers resources on demand [14]. In order to distribute physical resources between many users, virtualization techniques maximize the flexibility of the mobile cloud computing system.

A stand-alone physical server is designed to run an operating system and its applications. When there is a demand to use stand-alone services from clients, another stand-alone physical server has to be added with its operating system and applications to support the new demand. Stand-alone clients sometimes use 5% to 15% of the capacity of the physical server when there are only a few operations. This shows inefficient resource utilization and an immergence of virtualization. Virtualization enables a single physical server to run multiple virtual server instances and applications. It provides simpler and more efficient resource management. VIRTUALIZATION software solves problems by enabling several operating systems and applications to run on one physical server or *host*. Each self-contained virtual server is isolated from the others and uses as much of the host's computing resources as it requires.

In a virtualized environment, virtual machines (VMs) have a common shared mobile cloud storage device. They seem to consumers to be one "physical server." Owing to the interconnected physical servers in a virtualized environment, the physical servers share physical resources easily. And physical servers operate together to make the VMs active and more powerful when needed. Owing to these benefits, applications can be deployed faster, performance and availability increase, and operations become automatic. A virtualized environment becomes easier to implement and less costly to own, compared with the normal stand-alone physical infrastructure. Managing and allocating virtual servers in mobile cloud computing requires a special software control system called a hypervisor. The hypervisor manages multiple virtual servers or multiple instances of the same operating system on a single physical server. The hypervisor is responsible for providing virtual servers with access to physical resources.

A hypervisor is generally limited to one physical server and can only create and manage virtual images that use its physical server resources, as shown in Figure 1 [5]. In addition, the hypervisor provides features for controlling, sharing, and scheduling hardware resources, like power processing, memory, and network interfaces.

As mentioned earlier, the VIM administers multiple hypervisors across many physical servers. The VIM provides important functionalities to prevent failures of physical servers by clustering hypervisors across multiple physical servers (Figure 2a). This allows live migration of virtual servers from a failed hypervisor to a stable one, and thus, increases the reliability of the mobile cloud computing system as shown in Figure 2b. Clustered hypervisors exchange heartbeat messages among themselves and with a central VIM in order to maintain dynamic monitoring of resource status. Moreover, clustered hypervisors share the same storage, which may be further used to store virtual server resource files, as shown in Figure 2.

Further research to improve reliability and computing performance has resulted in a variety of new architectures and configurations. In the architecture presented in Figure 2, activities of the physical server nodes are orchestrated by hypervisors. The software layer that sits on top of each physical server provides mobile cloud consumers with access to the cluster of physical servers as one unified computing unit.

A hypervisor cluster is a group of independent hypervisors with their corresponding physical servers working together as a single system to provide high-service availability to consumers. When failure occurs in one hypervisor, resources are redirected, and workload is redistributed to another hypervisor inside the cluster. The hypervisor cluster uses a shared cloud device to live-migrate virtual servers and their resources [5], and the ability to seamlessly migrate virtual servers is now an integral part of nearly every virtualization deployment. A hypervisor cluster consists of two or more hypervisors working together to provide a fail-safe system when any hypervisor might malfunction. The cluster mechanism gives a higher level of availability, reliability, and scalability, compared with a single hypervisor. The main goal of clustering hypervisors in mobile cloud computing is to

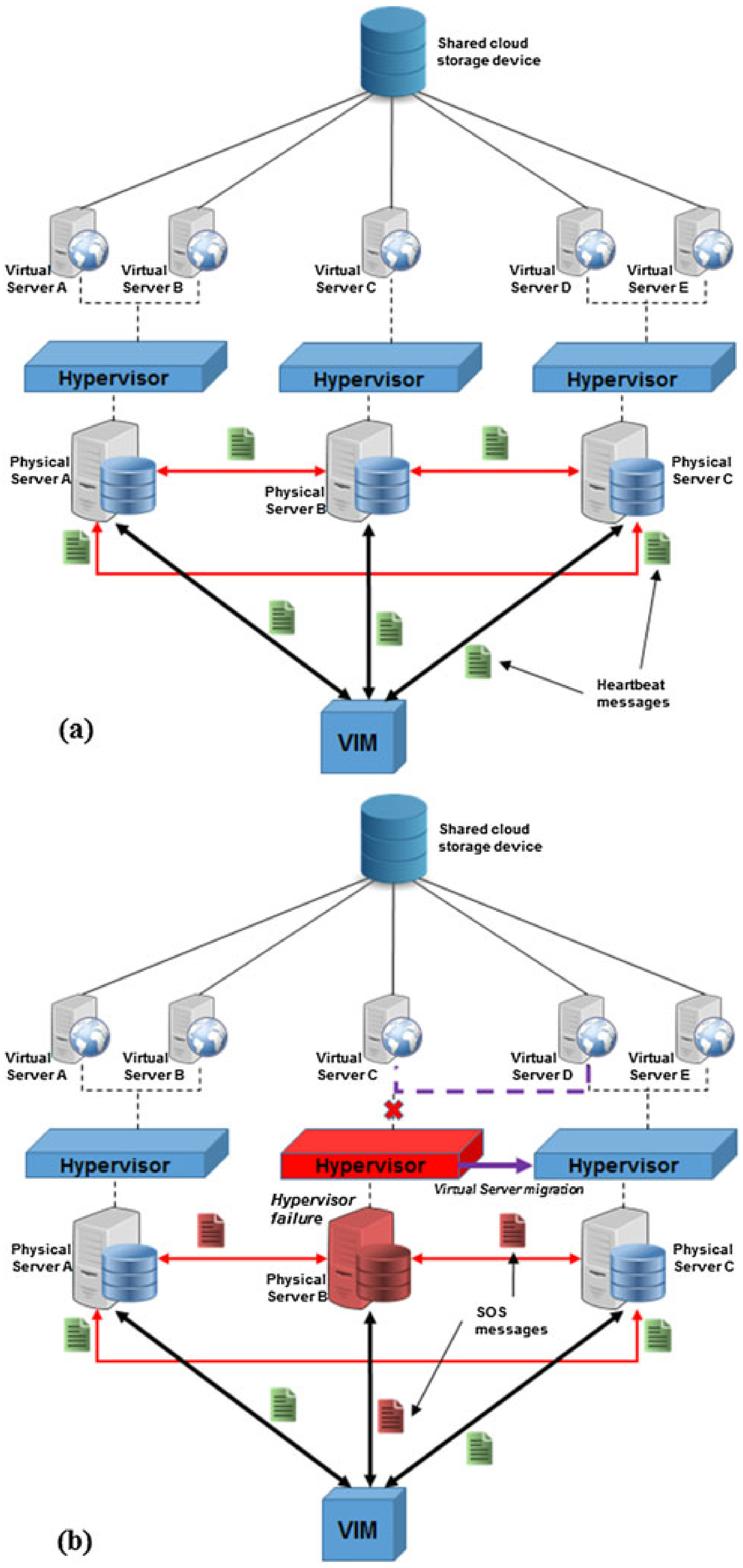

**Figure 2.** A mobile cloud architecture with hypervisors clustering - (a) and virtual server migration in case of hypervisor failure - (b) [4].

make each one of them highly available, even during a failure. Higher availability of mobile cloud resources is a crucial issue from the mobile cloud consumer's perspective.

## 4. PROPOSED APPROACH FOR SERVER FAILURE PREDICTION

When resource utilization exceeds a predefined threshold limit, the VIM migrates virtual servers to another hypervisor that has a lower load. In this paper, we only focus on service failure prediction because of increased system load. We do not focus on hardware failure. In other words, we only focus on service failure because of undercapacity of resources.

### 4.1. *Failure prediction model*

When there are more requests from mobile cloud users, the resources inside the physical server (or corresponding hypervisor) will exceed capacity, resulting in failure to properly handle new incoming users' requests on time. In turn, the hypervisor inside a cluster will fail to provide services to users on time. Total service requests, CPU usage status, RAM usage status, and data input/output status indicate the current load condition of the hypervisor or corresponding physical server. These multiple factors show the condition of the server (and hypervisor) and their status. By comparing the current hypervisor status with previous failure data, it is possible to know if the current load can lead to a failure or not.

Error log files are used to retrieve previous failure data. Each log file contains the historical physical server status data at the time of failure and before failure. By analyzing the previous physical server status data, we can group the raw data into training data sets to model our prediction. The training data can be arranged into groups reflecting server status, depending on failure time. We assume that failure occurs at time $t$. The physical server attributes in this research are CPU status, RAM status, and the number of service requests. We categorize failure groups prior to time $t$. The failure data that are retrieved from the error log will create training data sets to model our predictions. The status data of each group according to the range of the time intervals are averaged to give more generalized data and are grouped into time sequential order.

The reason we take an average value for the data is for direct application of traditional classification analysis techniques to our time series classification problem [15]. By comparing the category groups based on failure interval times against the attributes of the physical servers at runtime, it is possible to know how much time is left before failure if the physical server load continues to increase.

The prediction algorithm is modeled using a naive Bayes classifier [7,16–18]. Despite its unrealistic independence assumption, it is remarkably successful in practice and uses a probabilistic prediction model to group the real-time physical server status against the training data. Once the group number is predicted, it is possible to know the amount of time left before failure.

### 4.2. *Assumptions for training data*

Training data are created from error log files and are grouped according to interval times before failure. We have $n$ groups of already trained data from $G_1$ to $G_n$. $G_1$ is the group that shows attribute data close to failure, whereas $G_n$ is far from failure. As the group number increases, the tendency to fail decreases. We have $m$ attributes that make up the training data from $A_1$ through $A_m$. These attributes can be CPU status, RAM status, the number of requests, and the amount of packet data sent/received by the server. We have real-time data ($D$) that has to be grouped according to physical server attribute values.

### 4.3. *Modeling the prediction*

Our aim is to determine the most related group for real-time data $D$ in relation to the training data. First, we calculate probability $P(G_n|D)$ that real-time record attribute data (D) is in group $G_n$ by using the naive Bayes classifier. After obtaining the probability value for $D$ towards each group, the higher probability group will be taken for $D$.

Using the Bayes rule, we can calculate $P(G_n|D)$ as

$$P(G_n|D) = \frac{P(D|G_n)P(G_n)}{P(D)} \tag{1}$$

$P(G_n|D)$ is the probability that the newly recorded data ($D$) are in group $G_n$, that is, the conditional probability of a given attribute set of recorded data appear in group $G_n$. $P(G_n|D)$ is the probability that (for a given group, $G_n$) the attributes in the real-time record data ($D$) correlate with group $G_n$. $P(G_n)$ is the prior probability of a given group in the training data, that is, the probability of real-time recorded data being in group $G_n$ without considering their attribute value information. $P(D)$ is the prior probability that specific real-time recorded data occurred.

To determine which group the new data $D$ should go in, we need to calculate $P(G_n|D)$ for each of the groups and find the highest probability.

$$P(G_n|D) = P(D|G_n)P(G_n) \tag{2}$$

$P(D)$ can be safely ignored, because we are interested in the relative (not absolute) values of $P(G_i|D)$, and $P(D)$ simply acts as a scaling factor and is common to all $P(G_n|D)$ calculations. Recorded data $D$ are split into a set of attributes, such as CPU load, RAM load, hard disk read/writes, error requests, and total requests from $A_1$ through $A_m$. $P(D|G_n)$ is the simple product of the probabilities for each attribute, given a specific group. To obtain

*P(D|G$_n$)*, we calculate the product of the probabilities for each attribute, that is, the likelihood that each attribute appears in $G_n$.

$$P(D|G_n) = P(A_1|G_n)P(A_2|G_n)\times\ldots\times P(A_m|G_n)$$
$$P(D|G_n) = \prod_{i=1}^{m} P(A_i|G_n) \tag{3}$$
$$P(G_n|D) = \left(\prod_{i=1}^{m} P(A_i|G_n)\right)P(G_n)$$

It is assumed that each attribute is random and normally distributed and can be represented by a Gaussian distribution. Each attribute will have its mean ($\mu_{Am}$) and variance ($\sigma^2_{Am}$) in each group $G_n$ . To calculate the value of each attribute's mean and variance in the available groups, we use the following formulas:

$$\mu_{A_m|G_n} = \frac{1}{N(G_n)} \sum_{i=1|G_n}^{N(G_n)} (A_m)_i \tag{4}$$
$$\sigma^2{}_{A_m|G_n} = \frac{1}{N(G_n)} \sum_{i=1|G_n}^{N(G_n)} \left((A_m)_i - \mu_{A_m|G_n}\right)^2$$

where *N(G$_n$)* is the amount of training data in group $G_n$.

Using a Gaussian equation, we can obtain the probability of each attribute value, *P(A$_{m,x}$|G$_n$)*, for each group of training data. For each attribute of data *D*, probability will be calculated with respect to the training data using the Gaussian probability density function as follows:

$$P\left(A_{m,x}|G_n\right) = \frac{1}{\sqrt{2\pi\sigma^2{}_{A_m|G_n}}} e^{\left(-\frac{\left(A_{m,x}-m_{A_m|G_n}\right)^2}{2\sigma^2{}_{A_m|G_n}}\right)} \tag{5}$$

where $A_{m,x}$ is one of the attribute values of the real-time data and Equation 5 is resolved using Weisstein's method [19].

Figure 3 shows how the Gaussian probability density graphs for two attributes will group a set of training data. We have new data to be tested in green that has to be grouped into one of the two groups shown in Figure 3. The two attributes show a Gaussian distribution surface shown in a circular contour in Figure 3. The training data are used to construct the Gaussian model inside each failure group. We use the naive Bayes classifier to obtain the most plausible group for the new set of data (green dot [data to be tested] in Figure 3) using the model generated from the training data. We use only two attributes in Figure 3 to clearly show our model.

## 4.4. *Simulation results using experimental data*

It is difficult to obtain an error log file from a mobile cloud environment, and it is also necessary to group the previous data into failure groups for prediction. To replace the error log data, we used one of the servers of the MonALISA repository for ALICE [19]. Physical server attributes for a 1-month period were collected using the FZK server farm of ALICE. Collected server attributes from the FZK server are CPU usage, RAM usage, CPU IO wait time, and Transmission Control Protocol/User Datagram Protocol (TCP/UDP) packets. We assumed that the FZK server will fail at three points: when server traffic reaches 1.0 GB/s, 1.22 GB/s and 1.95 GB/s (Figure 4). The red line in Figure 5 shows the three failure points. Because there are no actual data for a failure point in a server, we chose these points depending on load intensity in server traffic. To obtain a set of failure points, we chose three different server traffic points: 1 GB/s, 1.22 GB/s, and 1.95 GB/s. The server traffic points provided us a set of data points before failure occurs. The data regarding these failure points are grouped as shown in Figure 5, and these points are arranged into failure groups to give us a view of how the server behaves when failure happens. From 1 month of data, we arranged failure groups from for 1 GB/s, 1.22 GB/s, and 1.95 GB/s.

We set six failure groups from the selected failure points. The selected groups are 15, 35, 70, 120, 150, and

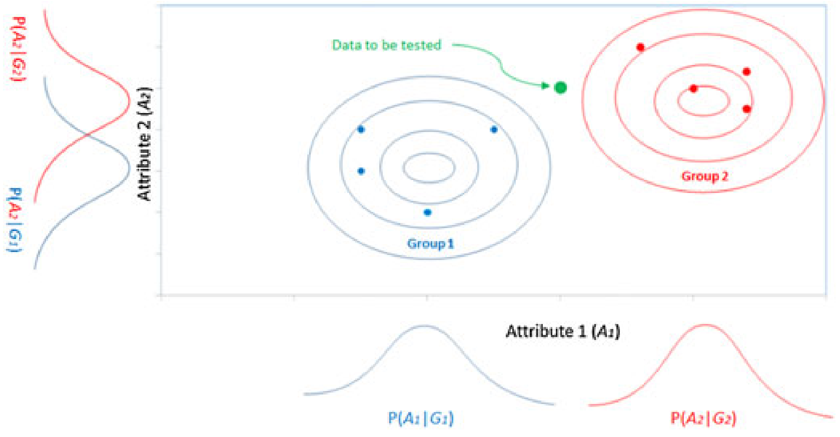

**Figure 3.** Gaussian probability density for two attributes in two training data groups.

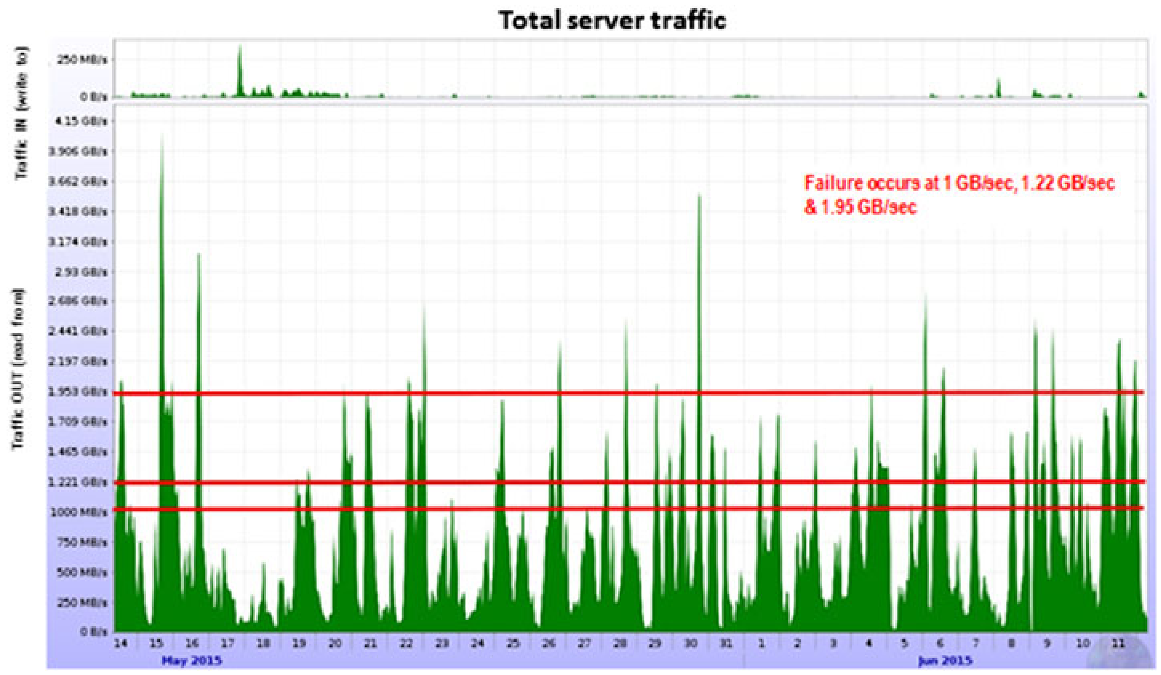

**Figure 4.** MonALISA repository server traffic.

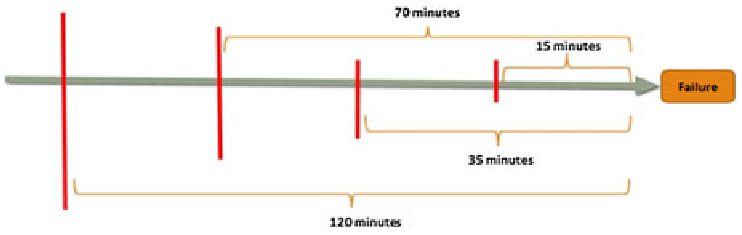

**Figure 5.** Arrangement of failure groups.

180 min before failure (Figure 5). Using each occurrence in a month, we compiled the failure groups before the server traffic reached the three failure points. We gave the failure groups names starting from 1, which is the one that occurs nearest to the points. We collected a set of groups within a 1-minute interval and averaged them to obtain a group of failures in accordance with each failure occurrence. We established a range of failure groups from 1 to 180 min and obtained training and test data sets. We selected 20% of the data points at random from the collected data as test data sets while keeping 80% for training data sets.

We selected test data sets in order to see the accuracy of our proposed naive Bayes classification method. From the results of the test data sets, the failures with the maximum probable values from naive Bayes classification were assigned as failure groups, and the predicted failure groups were compared with the original failure groups to check for correct predictions. The number of correctly predicted failure groups was used to calculate the accuracy rate of our proposed method.

The experiment showed, for the first failure point (1 GB/s), a 71.78% accuracy in the prediction of the pre-assigned failure group within 15 min of the failure group range. As we widened the range to 35 min, the prediction accuracy of the assigned failure range became 50.53% and we continued to increase the range up to 70, 120, 150, and 180 min, obtaining accuracies of 34%, 39.25%, 19.63%, and 15.22%, respectively. The results are shown in Figure 6. For the second failure point (1.22 GB/s), we obtained accuracies of 72.41%, 39.53%, 34.66%, 12.75%, 13.15%, and 7.98% for the 15, 35, 70, 120, 150, and 180 min groups, respectively. For the third failure point (1.95 GB/s), we obtained accuracies of 86%, 50%, 40.18%, 24.53%, 28.41%, and 25.98% for 15, 35, 70, 120, 150, and 180 min groups, respectively. Figure 7 shows the box plots of the failure groups with their respected tolerance ranges. The data points are generated from the differences between the generated data groups and the original predefined group numbers. Around failure occurrence time, the box plots show a normal and better result that resembles the original predefined failure groups. As the failure group ranges increase, the tolerance between the predicted and the pre-defined groups varies in big numbers.

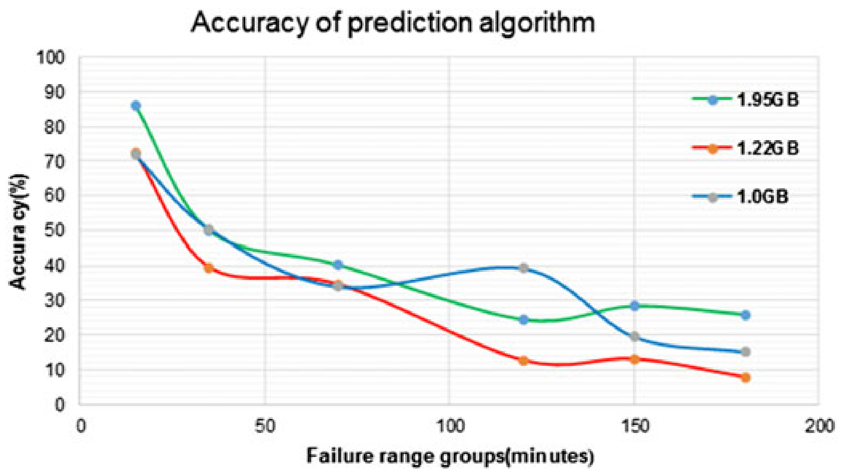

**Figure 6.** Results for accuracy of the prediction algorithm.

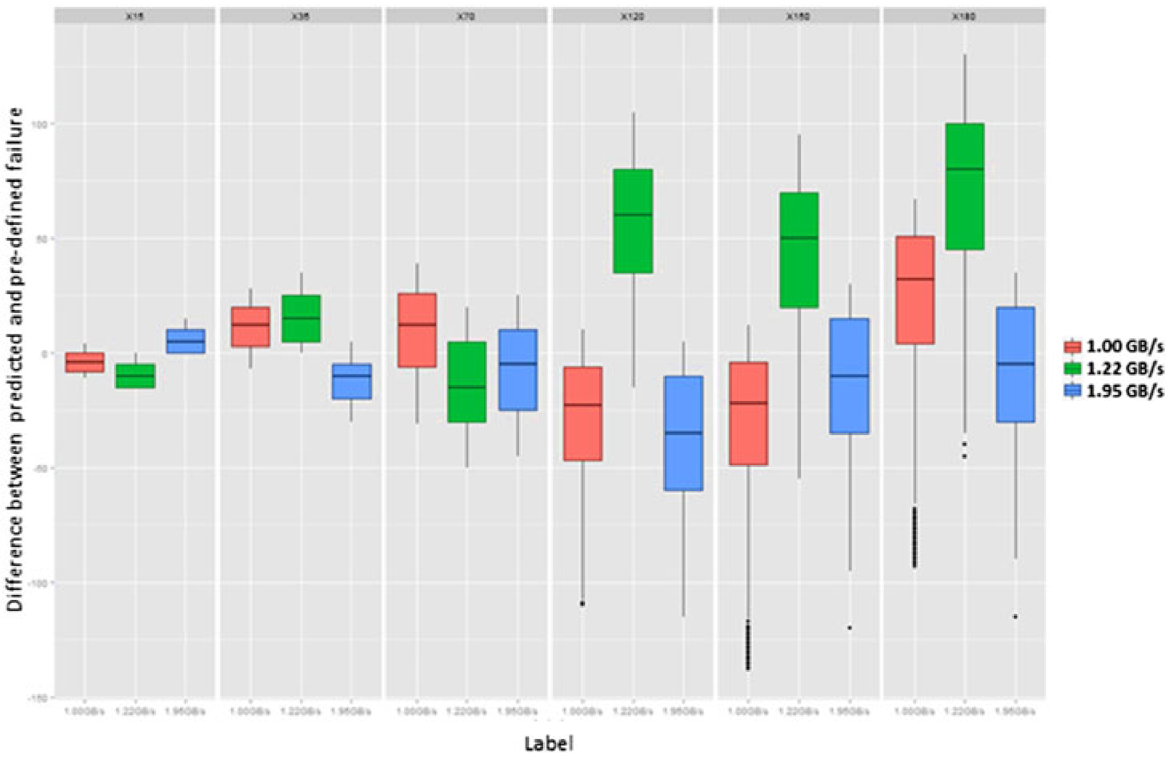

**Figure 7.** Box plots for experiment results.

As we can see in both Figures 6 and 7, the accuracy of the predictions is better for failure group 3 (1.95 GB/s).

## 5. PROPOSED CLUSTERING AND VIRTUAL SERVER MIGRATION MECHANISM

In a clustered hypervisor environment, the physical servers within a cluster should have the same configuration and component types. In that case, applications running on one physical server can easily be transferred to the other without compatibility issues. The new host physical servers must have enough resources, such as CPU processing capability, free storage space, and available RAM, to accept the new requests.

When a hypervisor fails and needs to migrate its hosted virtual servers, the capacity of the new hypervisor should be higher than the failed hypervisor. If there is no hypervisor with enough capacity, the virtual servers will be shared among different hypervisors in the cluster. The VIM controls the hypervisor clustering process. Because each hypervisor is a part of a cluster, it shares its status and basic information as to RAM type and amount, CPU architecture and load, and storage device types. Depending on their properties and failure prediction results in real time, the VIM will migrate the virtual servers within a cluster.

When the migration of virtual servers is initiated, the VIM will check for the hypervisor inside the cluster that has the lowest load. The hypervisor with the lowest load has a higher failure-prediction group result. In other words, the hypervisor that has a lower load has a lower probability of failure, and vice versa. So, the VIM will check for those hypervisors with a higher failure-prediction group result and migrates virtual servers from the lower group hypervisor into the higher group hypervisor to level off the risk of failure.

When we obtain a group prediction, and it passes the threshold, the number of virtual servers of the hypervisor has to be decreased, or some of the virtual servers will move to other hypervisors inside the cluster. Thus, the VIM checks the available resources in order to host at least one of the virtual servers. Then, the VIM migrates one virtual server and checks the group prediction again for whether the hypervisor remains stable or not. If the prediction group still does not change, the system migrates other virtual servers to new hypervisors with the smallest load. By conducting prediction and migration continuously, our proposed systems make the hypervisors inside a cluster provide higher availability for mobile cloud consumers. There are many existing clustering methods for hypervisors. For example, the K-means clustering algorithm is one of the oldest methods, which has been used by different mobile cloud providers with minor modifications [20]. This paper does not depend on a specific clustering model, so any clustering model can be applied to hypervisors to improve the reliability of the mobile cloud computing system.

### 5.1. *Future research topics in migration schemes*

The choice of different probability thresholds in our proposed method depends on the type of physical server, operating systems, applications deployed on the virtual servers, and the load on the physical servers. We conducted a simulation on data collected from a third-party physical server, but could not include these attributes in this research. In future work, we will consider such a factor.

The proposed method predicts the possible failure of a hypervisor and its corresponding physical server(s) and suggests migrating the VM to an alternative VM before a failure occurs. However, a false prediction signal from our propose method is possible. In case of a false positive, the cost is unnecessary overhead from migration, but with a false negative, the cost is the downtime of the hypervisor. Both such false alarms incur additional costs. In addition, the hypervisor failure prediction will ask where to migrate. The location of the migrating hypervisor may influence the performance of cloud computing, such as bandwidth, tardiness, computing time, and other factors. We will conduct more research on these issues in the future.

## 6. CONCLUSION

In a virtualized mobile cloud environment, clustering of hypervisors is beneficial, in that, if a hypervisor failure occurs, active virtual servers can be transferred to a stable one. A failure prediction model was proposed in this paper to forecast whether a hypervisor has a tendency to fail under its current load. The prediction model was designed with a naive Bayes probability classifier and a Gaussian density function to attain high accuracy for failure predictions in real time. The predictions show high accuracy around the range in which the failures occur. But when we increase the range of the failure groups, the accuracy lowers. The reason for this is that the data-point attribute values become similar and do not show significant change as they approach the selected failure point. With failure prediction, we can achieve live migration of virtual servers, ahead of any outage, from the hypervisor at risk to a more stable one. Hence, migration of virtual servers decreases the load, and prevents failures in the mobile cloud system.

## ACKNOWLEDGEMENTS

This research was supported by a Korea University Grant.

## REFERENCES

1. Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, Zaharia M. A view of cloud computing. *Communications of the ACM* 2010; **53**(4): 50–58
2. Gubbia J, Buyyab R, Marusic S, PalaniswamiJ M. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems* 2013; **29**: 1645–1660
3. Sanaei Z, Abolfazli S, Gani A, Buyya R. Heterogeneity in mobile cloud computing: taxonomy and open challenges. *IEEE Communications Surveys & Tutorials* 2014; **16**(1): 369–392
4. Liu F, Shu P, Jin H, Ding L, Yu J, Niu D, Li B. Gearing Resource-Poor Mobile Devices with Powerful Clouds: Architecture, Challenges and Applications. *IEEE Wireless Communications Magazine Special Issue on Mobile Cloud Computing* June 2013; **20**(3): 14–22
5. Erl T, Mahmood Z, Puttini R. *Cloud Computing: Concepts, Technology & Architecture*. US: Prentice Hall Service Technology Series, 2014.
6. A. Oliner, R. Sahoo, J. Moreira and M. Gupta, "Fault-Aware Job Scheduling for BlueGene/L Systems,"

Proc. 18th Int'l Parallel and Distributed Processing Symp. (IPDPS ',04), p. 64, 2004.
7. Tang Y, Pan W, Li H, Xu Y. Fuzzy Naive Bayes classifier based on fuzzy clustering. *IEEE International Conference on Systems, Man and Cybernetics* Oct. 2002; **5**: 6
8. Mitchell TM. *Machine Learning*. The McGraw-Hill Companies, Inc., 1997.
9. C. Tan, Y. Xia, H. Chen, B. Zang, "TinyChecker: Transparent Protection of VMs against Hypervisor Failures with Nested Virtualization," IEEE/IFIP 42nd International Conference on Dependable Systems and Networks Workshops (DSN-W'2012), pp.1-6, June, 2012.
10. Dabbagh M, Hamdaoui B, Guizani M, Rayes A. Toward energy-efficient cloud computing: Prediction, consolidation, and overcommitment. *IEEE Network* April 2015; **29**(2): 56–61
11. Dabbagh M, Hamdaoui B, Guizani M, Rayes A. Energy-efficient resource allocation and provisioning framework for cloud data centers. *IEEE Transactions on Network and Service Management* Sep. 2015; **12**(3): 377–391
12. Weiwei Zheng, Zhili Wang, Haoqiu Huang, Luoming Meng, and Xuesong Qiu, "EHMM-CT: An Online Method for Failure Prediction in Cloud Computing Systems", KSII Transactions on Internet and Information Systems, to be published in near future.
13. A. Singh, M. Korupolu, and D. Mohapatra, "Server-Storage Virtualization: Integration and Load Balancing in Data Centers," Proc. 20th ACM/IEEE Conf. Supercomputing (SC'08), p. 53, 2008.
14. W.-T. Tsai, "Service-Oriented Cloud Computing Architecture," IEEE Seventh International Conference on Information Technology: New Generations (ITNG), pp.684-689, 2010.
15. MacQueen JB. Some Methods for classification and Analysis of Multivariate Observations. In *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*. University of Berkeley, California Press, 1967. 281–297.
16. A. Niculescu-Mizil, R. Caruana, "Predicting Good Probabilities With Supervised Learning," Proceedings of the 22nd IEEE International conference on Machine learning, pp. 625 - 632, Aug. 2005.
17. Kelemen A, Zhou H, Lawhead P, Liang Y. Naive Bayesian classifier for microarray data. *Proceedings of the IEEE International Joint Conference on Neural Networks* July, 2003; **3**: 1769–1773
18. J. Rennie, L. Shih, J. Teevan, D. Karger, "Tackling the Poor Assumptions of Naive Bayes Text Classifiers," Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), vol. 3, pp. 616-623, Aug. 2003.
19. Weisstein, Eric W, "Gaussian Function", From MathWorld A Wolfram Web Resource, http://mathworld.wolfram.com/GaussianFunction.html.
20. B. Wang, S. Zhang, "A Novel Text Classification Algorithm Based on Naive Bayes and KL-divergence," IEEE Sixth International Conference on Parallel and Distributed Computing, Applications and Technologies, pp.913-915, Dec. 2005.

## AUTHORS' BIOGRAPHIES

**Berihun Fekade** is now a PDEng student with Eindhoven University of Technology, The Netherlands. He received his MS degree in computer and information science from Korea University, Korea, in 2016, and BA degree in electrical engineering from Arba Minch University, Ethiopia, in 2007. He has been working as senior software engineer in "Cybersoft PLc," Addis Ababa, Ethiopia, from 2007 to 2013. His research interest include Internet of Things, mobile cloud computing, machine-to-machine communications, and fog computing.

**Taras Maksymyuk** is now an assistant professor in the Telecommunications Department, Lviv Polytechnic National University, Ukraine, and an invited research professor in the Department of Computer Convergence Software, Korea University, Sejong Metropolitan City, Korea. He received his PhD degree in telecommunication systems and networks in 2015, his MS degree in information communication networks in 2011, and his BA degree in telecommunications in 2010, all from the Lviv Polytechnic National University. He was recognized as the Best Young Scientist of Lviv Polytechnic National University in 2015. He is currently an editor of KSII Transactions on Internet and Information Systems (SCI and SCOPUS indexed) and an editor of International Journal of Internet of Things and Big Data. His research interests include cognitive radio networks, LTE in unlicensed spectrum, mobile cloud computing, massive MIMO in 5G, machine-to-machine communication, and heterogeneous network in 5G, software-defined radio access networks, and the Internet of Things.

**Minho Jo** is now a professor with the Department of Computer Convergence Software, Korea University, Sejong Metropolitan City, Korea. He received his BA degree from the Department of Industrial Engineering, Chosun University, Korea, in 1984, and his PhD degree from the Department of Industrial and Systems Engineering, Lehigh University, in 1994. He is one of the founders of Samsung Electronics LCD Division. He is the founder and editor-in-chief of KSII

Transactions on Internet and Information Systems (SCI and SCOPUS indexed). He was awarded the Headong Outstanding Scholar Prize 2011. He is currently an editor of IEEE Wireless Communications and an associate editor of the IEEE Internet of Things Journal, Security and Communication Networks, and Wireless Communications and Mobile Computing. He is now the vice president of the Korean Society for Internet Information and was the vice president of the Korea Information Processing Society. His current research interests include LTE-unlicensed, cognitive radio, IoT, deep learning AI in IoT, HetNets in 5G, green (energy-efficient) wireless communications, mobile cloud computing, network functions virtualization, 5G wireless communications, optimization and probability in networks, network security, and massive MIMO.