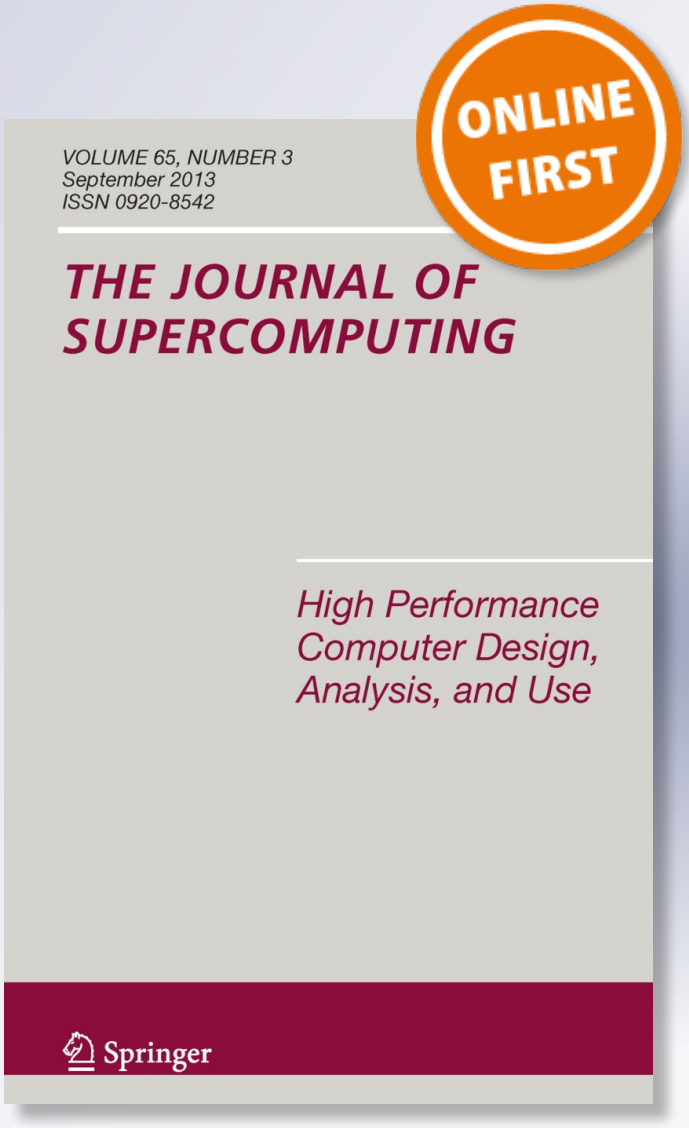
Early fault detection in IaaS cloud computing based on fuzzy logic and prediction technique

Dinh-Mao Bui, Thien Huynh-The & Sungyoung Lee

The Journal of Supercomputing
An International Journal of High-Performance Computer Design, Analysis, and Use

ISSN 0920-8542

J Supercomput
DOI 10.1007/s11227-017-2053-3



Springer

Your article is protected by copyright and all rights are held exclusively by Springer Science +Business Media New York. This e-offprint is for personal use only and shall not be self-archived in electronic repositories. If you wish to self-archive your article, please use the accepted manuscript version for posting on your own website. You may further deposit the accepted manuscript version in any repository, provided it is only made publicly available 12 months after official publication or later and provided acknowledgement is given to the original source of publication and a link is inserted to the published article on Springer's website. The link must be accompanied by the following text: "The final publication is available at link.springer.com".

Author's personal copy

J Supercomput
DOI 10.1007/s11227-017-2053-3

Early fault detection in IaaS cloud computing based on fuzzy logic and prediction technique

Dinh-Mao Bui[1] · **Thien Huynh-The**[1] · **Sungyoung Lee**[1]

© Springer Science+Business Media New York 2017

Abstract Availability is one of the most important requirements in production system. Keeping a persistent level of high availability in the Infrastructure-as-a-Service (IaaS) cloud computing is a challenge due to the complexity of service providing. By definition, the availability can be maintained by coupling with the fault tolerance approaches. Recently, many fault tolerance methods have been developed, but few of them adequately consider the fault detection aspect, which is critical to issue the appropriate recovery actions just in time. In this paper, based on a rigorous analysis on the nature of failures, we would like to introduce a method to early identify the faults occurring in the IaaS system. By engaging fuzzy logic algorithm and prediction technique, the proposed approach can provide better performance in terms of accuracy and reaction rate, which subsequently enhances the system reliability.

Keywords Approximate reasoning · IaaS cloud computing · Fault detection · Fuzzy logic · Prediction technique

1 Introduction

Cloud computing is a technology to deliver the infrastructure facilities such as computation, storage or network bandwidth over the Internet. This technology is actually

✉ Sungyoung Lee
sylee@oslab.khu.ac.kr

Dinh-Mao Bui
mao.bui@khu.ac.kr

Thien Huynh-The
thienht@oslab.khu.ac.kr

[1] Computer Engineering Department, Kyung Hee University, Suwon 446-701, Korea

Published online: 26 April 2017

Author's personal copy

based on the mechanism to elastically allocate the resources in a versatile manner. Subsequently, users can remotely initiate their services on-line and pay only for the amount that they actually use. The philosophy of this mechanism can be seen as the idea of re-usability. In addition, the scalability and the multi-tenancy are also the advantages of cloud computing which are transparent to end-users. With these kinds of features, users can focus more on the business model instead of spending time and effort to manage and maintain the underlying computing resources.

Among the technologies that are implemented in cloud computing, virtualization is one of the most basic things the system relies on. In this system, users actually access the physical resources *via* the virtual machines, which are created on top of the virtualization. Basically, these machines are parts of a three-layer model: the physical layer, the virtualization layer and the application layer. However, this three-layer system makes difficulties to unify the management, especially in the failures recovery [1]. In fact, managing failures in a cloud system is considered as a very high complexity duty because of various kinds of faulty sources.

Discussion on faults and fault tolerance is interesting. Indeed, even the cloud platforms consider a huge amount of functionalities, very few of them are related to fault tolerance [2]. Clearly, to make an effective solution dealing with the faults, all kinds of potential faults should be collected and identified in advance to issue the corresponding treatments [3]. However, not many studies concentrate on improving the fault detection in cloud computing. Due to this reason, the objective of this paper first is to analyze the current fault handling solutions. Also, the faulty sources, where the faults originate from, are considered thoroughly. After that, we propose our solution based on fuzzy logic and prediction technique to early detect the faults. Finally, the experiments are conducted to verify the effectiveness of this solution.

This paper is organized as follows. In Sect. 2, we provide a summary of the related works that are relevant to this topic. We also include the background knowledge of cloud computing as well as the potential failures in Sect. 3. In Sect. 4, we introduce the proposed approach to pro-actively detect the failures in IaaS cloud computing. The performance evaluation is conducted on Sect. 5. Eventually, the conclusion and direction for future work are summarized in Sect. 6.

2 Related works

There are two kinds of fault tolerance models which are reactive and proactive approaches [4,5]. The reactive approach is implemented to reduce the effects of faults on the system [6]. In the other hand, the proactive model avoids recovering from faults by predicting and pro-actively replacing the suspected component with the good one [7]. In "Byzantine Fault Tolerance for the Cloud" [8], the authors proposed an approach based on Byzantine fault tolerance (BFT). By investigating how to use BFT to develop fault and intrusion tolerance applications, the authors designed a modular architecture for BFT replication and built blocks for BFT consensus (configuration for various trust settings). Although BFT is an effective algorithm in the case of fault tolerance,

Author's personal copy

the cost paying for the redundancy and the voters is expensive in terms of time consumption and computation. Therefore, this approach might be not suitable for cloud computing.

In the research of "Autonomic Fault Tolerance using HA Proxy in Cloud Environment" [9], several instances of virtual machines running the same application are implemented. When one machine gets into trouble, the autonomic fault tolerance technique might detect and handle the failure to reassure the system reliability and availability. In addition, the authors also proposed a specific cloud virtualized system using HAProxy for monitoring. However, the metrics are not considered thoroughly, which subsequently decreases the accuracy of the fault detection.

Malik et al. [10] presented a model for result checking and decision making based on variant algorithm. By engaging a number of virtual machines (VMs) with the same configuration to be as the replications, this approach can provide a high level of reliability. However, there are still some drawbacks. First, the VMs, which belong to different users, are difficult to share the same configuration. Second, the VMs with high workload may not pass the reliability test due to insufficient resources. These kinds of VMs may be wrongly replaced without saving the working result. Intuitively, this approach is not a good choice for the large-scale and elastic cloud system.

The next study is "Towards a scalable, fault-tolerant, self-adaptive storage for the clouds" [11]. The authors focused on designing a storage infrastructure based on BlobSeer. Subsequently, a joint architecture was defined with the global behavior modeling phase. Besides, the author also provided the quality of services to achieve the desired goal for the storage system. Generally, the research partially focused on creating the fault tolerance scheme for cloud storage rather than making a solution for service providing cloud system.

In the research of "Algorithmic-Based Fault Tolerance for Matrix Multiplication on Amazon EC2" research [12], the authors extended the idea of implementing the algorithmic-based fault tolerance (ABFT) from high performance computing to the cloud. Nevertheless, this algorithm has the drawback of performance due to the high complexity. This issue actually slows down the system and might not be appropriate for the production [13].

Lastly, in the research of "Method of Fault Detection in Cloud Computing Systems" [14], the authors proposed a detection approach based on C4.5 decision algorithm. This algorithm is also combined with the characteristics of cloud computing to improve the correctness and effectiveness. However, the complexity of C4.5 is quite high (up to $O(mn^2)$), which might be slow when coping with a large number of data.

By examining the related works, we have come to the conclusion that although the research on fault tolerance exists, not many researchers have thoroughly considered the detection method with regard to the characteristics of the cloud system. Meanwhile, without a suitable detection mechanism, the fault tolerance solution cannot execute correctly at right time to maintain the availability and reliability. Because of that, there is a need to develop a low complexity and efficient method to early detect the failures. This is our motivation to propose the proactive fault detection approach based on fuzzy logic and prediction technique.

Author's personal copy

3 Background

3.1 Cloud system

IaaS cloud service providers tend to satisfy the users in terms of computing resources and relevances. To do that, the providers have to consider deploying the resources on a distributed system to meet the different quality of service as well as to efficiently load balance between clusters. In our research, we implement the following general IaaS cloud computing model (Fig. 1) to cover most of the requirement for service providers in real world. The system mainly consists of these following layers:

- Cloud balancing layer: this layer is a combined component of filters and authorization system. In this component, the input requests are classified and checked for the validity. By using the predefined rules created by the administrator, the requests are forwarded and answered based on the query results from cache system. After the classification process, the requests are passed to the appropriate servers for load balancing purpose.
- Service management (SM) layer: this layer takes responsibility to automate the provision of the cloud resources by collecting the capacity information and redirecting the virtual machine deployment requests.
- Orchestrator management (OM) layer: this layer takes responsibility to deploy the virtual machines to the Worker Nodes. In fact, the OM layer receives the requests from the service management layer and chooses the best worker node which satisfies all the constraints for VM deployment.

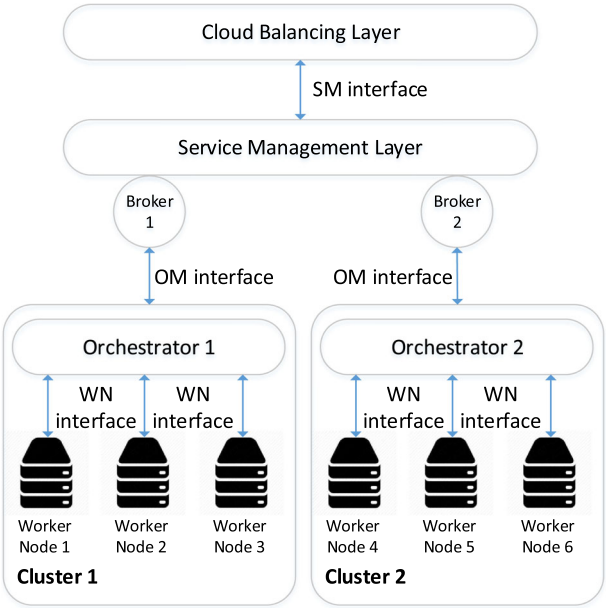


Fig. 1 IaaS cloud computing model

Author's personal copy

- Worker node (WN): this is the layer of physical servers. Many physical servers are put together into the same clusters with regard to their configurations and characteristics. In each worker node, we also implement the abstraction layer of virtualization by using the hypervisors such as Hyper-V, XEN or VMware. These hypervisors interact with the physical resources and manage all the VMs in worker nodes.

3.2 Failures in IaaS

Fault types can be classified according to the positions where they occur. The fault can come from physical infrastructure (worker node failure), from virtualized system (VM failure) or from orchestration manager (orchestrator failure). In the Open Nebula orchestrator [15], which is chosen to implement, the fault types and the corresponding pre-supports are as follows:

- Worker node failure: when a worker node is down, the predefined hook can be dispatched to solve the problem. Usually, this hook re-deploys the failed VM to another worker node to continue running the service.
- Virtual machine failure: there are two failure situations which can occur in the virtual machine life cycle: the VM fails and the VM crashes. In the VM fails, the error in the network prevents the image to be copied into the worker node. When this situation happens, the VM enters the "failed" state. To solve this situation, the same method dealing with the worker node failure is engaged. In the VM crash, the physical servers sometimes stop working due to some unknown reasons. In this case, a script might be embedded to restart the VM automatically.
- Orchestrator failure: the orchestrator can be recovered from a failures by restarting the core daemon. The running VMs are then reconnected and run as usual. In case of the pending VMs, these VMs might be re-deployed on a suitable host. However, the remaining VMs, which are not in the final state, may need to be recovered manually.

By default, Open Nebula takes care of any pending clean-up operation like removing image files or canceling the VMs. An external VM-collector script can be set up to automatically recover or delete the VMs when the core of Open Nebula is restarted after a crash. Although Open Nebula fault tolerance is mostly based on the reactive policies, this framework is actually supposed to reduce the effect of failures in the worker nodes, the VMs and the core of the orchestrator. However, the fault detection is really incompetent which sometimes results in the system in obsolete reactions. Obviously, the late response of the fault tolerance component makes the system unreliable. Due to this reason, the proactive fault detection is proposed to cover the blind spot of fault tolerance.

4 Proposed method

4.1 System description

In the perspectives of physical infrastructure, a cloud computing system can be considered as a group of clusters as seen in Fig. 2. These clusters consist of a number of

Author's personal copy

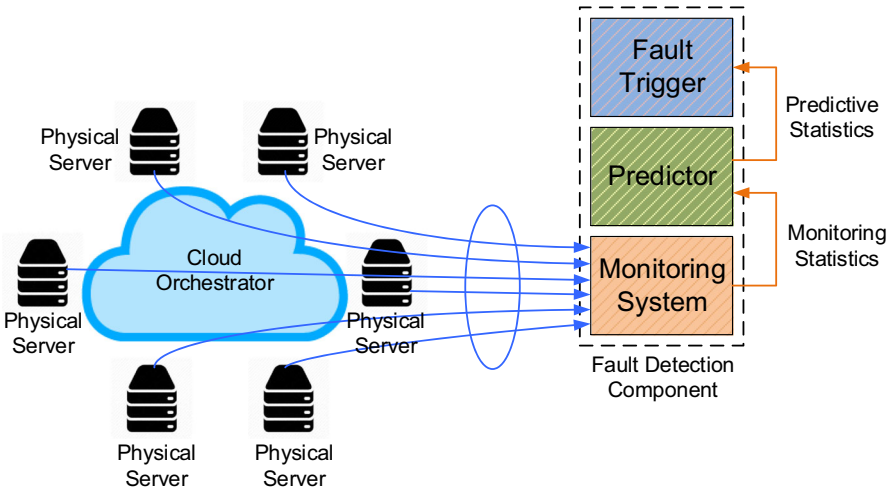


Fig. 2 Proposed fault detection architecture

physical servers which host many virtual machines. Each virtual machine possesses various utilizations and capacities of computing resources such as CPU, RAM, network bandwidth and storage volume. In order to establish the appropriate tolerance scheme, the proposed architecture is targeted to early detect the failures of the cloud in advance. To achieve this goal, the prediction technique is utilized to provide the information of the predictive monitoring statistics. After that, the faults are recognized by using fuzzy logic algorithm. Basically, the organization of the proposed architecture is as follows:

- The monitoring system: includes two applications, namely Ganglia [16] and HA Proxy [17]. The main task of this component is to collect system parameters as the input dataset for the prediction process. On the one hand, Ganglia is used to collect the working nodes' parameters such as CPU usage, RAM usage, disk I/O. On the other hand, HA Proxy [5] is aimed to monitor the network parameters such as response time, bandwidth, throughput, request/response ratio. Periodically, the monitoring system provides the input statistics to the predictor component.
- The predictor, which is built on the Gaussian process regression, is responsible to produce the predictive statistics of the next monitoring epoch. This futuristic utilization information is subsequently used in the fault trigger component.
- The fault trigger, as its name, does the fault detection based on the fuzzy logic method. Whenever a failure is identified, the result is forwarded to the fault tolerance system to recover in advance. Because the suitable actions would be applied in the early stage of failure, the consequence of potential errors is taken over much better.

For the convenience of presentation, the predictor would be introduced first in the next section.

Author's personal copy

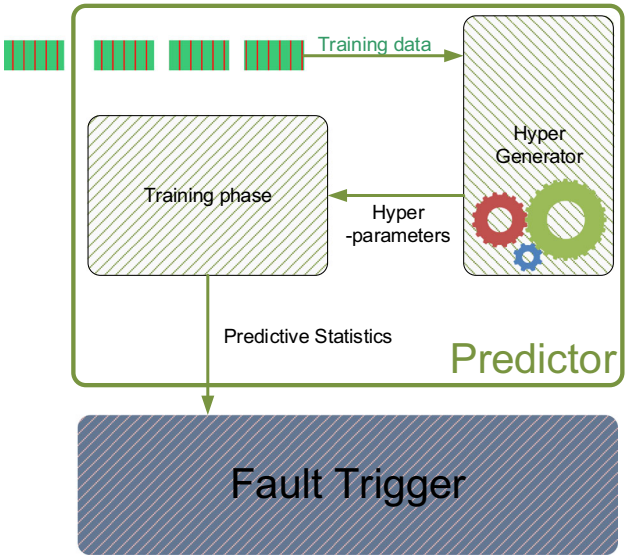


Fig. 3 Architecture of predictor component

4.2 Predictor

In our application, the objective of the prediction is to anticipate the utilization of resources. This anticipation process is mostly executed in the predictor component as seen in Fig. 3. Inside this component, the Bayesian learning and Gaussian process regression (GPR) are employed as the inference technique and probability framework, respectively. Because the input data for this model are the time-series utilization, the curve-fitting is preferred over the function mapping for mapping approach. It is important to note that the curve-fitting is more flexible with regard to the time-series data and non-stationary model.

The working of the prediction depends on the monitoring statistics, which reflect the historical utilization of cluster's facilities. To conveniently organize and execute this data, we define a special terminology, namely the monitoring window which describes a fixed period of statistics. Assuming that the input data are a limited collection of time location $x = [x_1, x_2, x_3, \ldots x_n]$ (the gap between two consecutive elements of this set x forms up a monitoring window denoted by m), a finite set of random variable $y = [y_1, y_2, y_3, \ldots y_n]$ represents the corresponding joint Gaussian distribution of historical monitoring statistics. Theoretically, the statistics consist of the physical resources utilization (CPU usage, RAM usage, disk I/O) and the network metrics (response time, bandwidth, throughput, request/response ratio) with regard to the time order. This set over the time constraint actually forms up the Gaussian process:

$$f(y|x) \sim \mathcal{GP}\big(m(x), k(x, x')\big) \tag{1}$$

Author's personal copy

with

$$m(x) = \mathbb{E}\big(f(x)\big) \tag{2}$$

$$k(x, x') = \mathbb{E}\Big(\big(f(x) - m(x)\big)\big(f(x') - m(x')\big)\Big) \tag{3}$$

in which, $m(x)$ is the mean function, evaluated at the time location variable x, and $k(x, x')$ is the covariance function, also known as the kernel function [18]. By definition, the kernel function is a positive-definite function which is used to define the prior knowledge of the underlying relationship. Basically, the kernel function is only a mandatory requirement when there is lack of finite dimensional form of the feature space. Otherwise, it can be dropped by directly calculating the sample. However, this feature space dimension is frequently infinite, which means that the kernel function cannot be directly calculated. For this reason, the kernel function technique is often chosen to tackle the Gaussian process regression. In addition, the kernel function comprises some special parameters that specify its own shape. These parameters are referred to as the hyper-parameters. Because the input data comes to the predictor as a set of n time locations, the kernel should be engaged in the matrix form.

$$\mathbf{K} = \begin{pmatrix} k(x_1, x_1) & k(x_1, x_2) & \cdots & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & \cdots & k(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_n, x_1) & k(x_n, x_2) & \cdots & k(x_n, x_n) \end{pmatrix} \tag{4}$$

Generally, the Square-Exponential (SE) kernel, also known as the Radial Basis Function (RBF) kernel, is chosen as the basic kernel function. In reality, the SE kernel is favored in most of the Gaussian process applications, because it requires the calculation for only few parameters. Moreover, there is a theoretical reason to choose this method, as it is an appropriately universal kernel for any continuous function whenever enough data are given. The formula for SE kernel is described as follows:

$$k_{SE}(x, x') = \sigma_f^2 \exp\left(-\frac{(x - x')^2}{2l^2}\right) \tag{5}$$

in which, σ_f is the output-scale amplitude and l is the timescale of the variable x from one moment to the next. l also stands for the bandwidth of the kernel and the smoothness of the function. In addition, l also plays the role of judgment for Automatic Relevance Detection (ARD) to discard the irrelevant input dimension. In the next step, we evaluate the posterior distribution of the Gaussian process. Assuming that the incoming value of the input data is (x_*, y_*), the joint distribution of the training output is y, and the test output is y_* as shown below:

$$p\left(\begin{bmatrix} y \\ y_* \end{bmatrix}\right) = \mathcal{GP}\left(\begin{bmatrix} m(x) \\ m(x_*) \end{bmatrix}, \begin{bmatrix} \mathbf{K}(x, x') & \mathbf{K}(x, x_*) \\ \mathbf{K}(x_*, x) & \mathbf{K}(x_*, x_*) \end{bmatrix}\right) \tag{6}$$

Author's personal copy

here, $\mathbf{K}(x_*, x_*) = k(x_*, x_*)$, $\mathbf{K}(x, x_*)$ is the column vector made from $k(x_1, x_*)$, $k(x_2, x_*) \cdots, k(x_n, x_*)$. In addition, $\mathbf{K}(x_*, x) = \mathbf{K}(x, x_*)^T$ is the transposition of $\mathbf{K}(x, x_*)$. Subsequently, the posterior distribution over y_* can be evaluated with the below mean m_* and covariance C_*.

$$m_* = m(x_*) + \mathbf{K}(x_*, x)\mathbf{K}(x, x')^{-1}(y - m(x)) \tag{7}$$

$$C_* = \mathbf{K}(x_*, x_*) - \mathbf{K}(x_*, x)\mathbf{K}(x, x')^{-1}\mathbf{K}(x, x_*) \tag{8}$$

then

$$p(y_*) \sim \mathcal{GP}(m_*, C_*) \tag{9}$$

The best estimation for y_* is the mean of this distribution:

$$\overline{y}_* = \mathbf{K}(x_*, x)\mathbf{K}(x, x')^{-1} y \tag{10}$$

In addition, the uncertainty of the estimation is captured in the variance of the distribution as follows:

$$var(y_*) = \mathbf{K}(x_*, x_*) - \mathbf{K}(x_*, x)\mathbf{K}(x, x')^{-1}\mathbf{K}(x, x_*) \tag{11}$$

Theoretically, the predictive statistics retrieved from the GPR is highly accurate compared to other regression methods [19]. However, the standard implementation of GPR costs $O(n^3)$ for computational complexity and $O(n^2)$ for storage complexity when calculating n training points of the dataset [20]. Most of the complexity comes from calculating the matrix inverse and log determinant in hyper-parameter learning phase. This can be seen as the drawback of GPR, which prevents the prediction from being quickly calculated on a large dataset. To solve this problem, we use the complexity reduction technique applying to the hyper-parameters learning phase of the prediction process. Theoretically, the law of log determinant, the fast Fourier transform and the stochastic gradient descent are combined to reduce the complexity to $O(nlogn)$ with n representing the number of training points. For more information, the detail of the complexity reduction has been already proposed in our previous research [21]. Further discussion can be found also in this original paper. Subsequently, the predictive monitoring statistics could be generated and utilized for the failure detection in the fault trigger component, which is presented in the next section.

4.3 Fault trigger

The architecture of the fault trigger is described in Fig. 4. This architecture is used to identify the failures in IaaS cloud computing system based on fuzzy logic and predictive statistics. Intuitively, the predictive statistics provides the futuristic awareness of the system status. With this ability, the fault tolerance would make more sense and more appropriate to cover the potential issues. On the other hand, the attractive benefit of fuzzy logic algorithm is the approximate reasoning with imprecise proposition based on the fuzzy set theory. In order to implement the fault trigger component, the

Author's personal copy

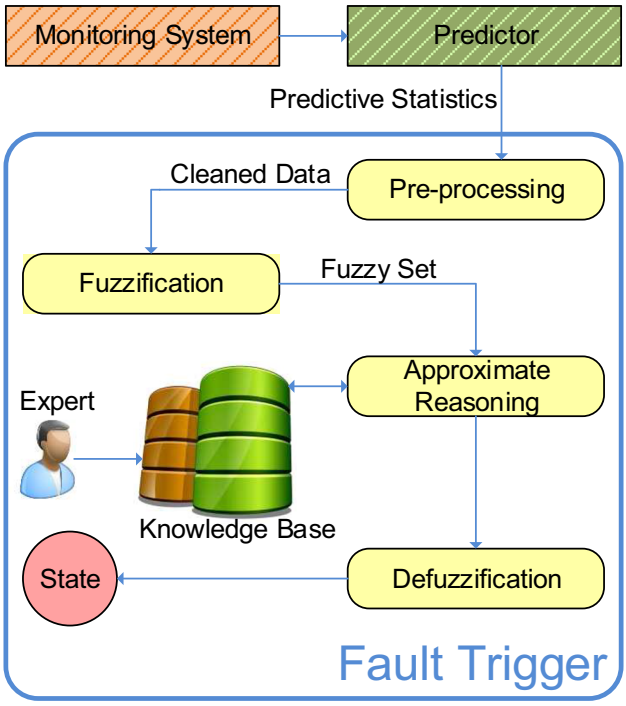


Fig. 4 Architecture of fault trigger component

rule form of fuzzy logic algorithm is formulated as below:

$$R_{ui} : \text{if } (X_1 \text{ is } F_{1i}) \text{ and } \cdots \text{ and } (X_n \text{ is } F_{ni}) \text{ then } Y \text{ is } G_i. \tag{12}$$

where X_j, $j = 1,\ldots,n$ are called antecedent variables defined on a domain U_j. Similarly, Y is the consequent variable defined on a domain V. Each F_{ji} is a linguistic term expressed by a fuzzy subset over the corresponding U_j. For any $u_j \in U_j$, the degree of membership function $\mu_{F_{ji}u_j}$ shows the magnitude to which u_j is compatible with the term F_{ji}. Similarly, G_i is a linguistic term expressed by means of a fuzzy subset on V. For any $v \in V$, the degree of membership function $\mu_{G_i}(v)$ is the degree which v is conformant to the term G_i.

In the proposed approach, the aforementioned predictive information is used as the input data to early identify the potential problem. In order to do that, the first step of establishing the fault trigger component is to identify the desired metrics [22]. Based on the studies of system performance and network measurement [23–25], the most fundamental metrics can be considered as below:

- Response time: the time elapsed between the moment that the node receives the request to the time that this node returns the response.
- Throughput: the amount of data transfers in a given unit of time.
- Bandwidth: the bandwidth of computing nodes.

Author's personal copy

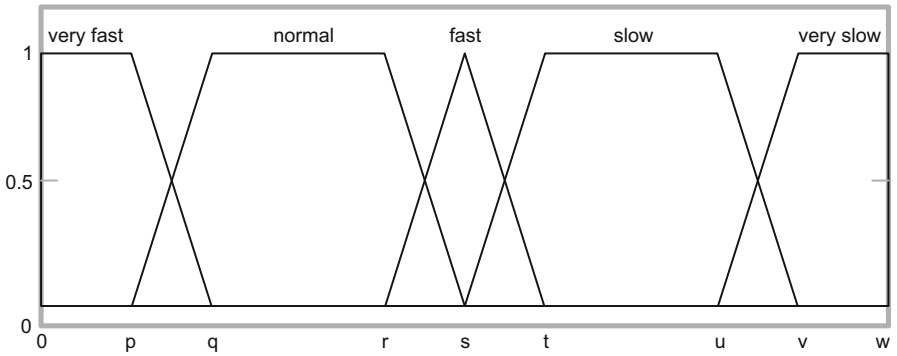


Fig. 5 Fuzzy response time metric

- Resource utilization: the total amount of resources is actually allocated and served the request.

Because of the similarity in the reasoning procedure, we would like to demonstrate the estimation process for only one metric: the response time. Additionally, the detection process for other metrics can be implemented similarly with minor changes and adjustments in the parameters.

4.4 Response time

In this demonstration, five linguistic terms are used to describe the value of response time: *very fast*, *fast*, *normal*, *slow*, *very slow*. By definition, the input parameter x determines the value of fuzzy variables. Figure 5 explains how the determination process is conducted. Depending on the network status, the value of fuzzy variable might fluctuate from 0 to 1. Accordingly, the trapezoidal membership functions μ are described by using the aforementioned parameter x with p, q, r, s, t, u, v, and w being the thresholds. These membership functions are given by the expressions below:

$$\mu_{veryfast}(x) = \begin{cases} 0 & \text{if } x \geq q \\ \frac{q-x}{q-p} & \text{if } p \leq x < q \\ 1 & \text{if } x < p \end{cases} \tag{13}$$

$$\mu_{fast}(x) = \begin{cases} 0 & \text{if } (x < p) \text{ or } (x > s) \\ \frac{x-p}{q-p} & \text{if } p \leq x < q \\ \frac{s-x}{s-r} & \text{if } r \leq x < s \\ 1 & \text{if } q \leq x < r \end{cases} \tag{14}$$

$$\mu_{normal}(x) = \begin{cases} 0 & \text{if } (x < r) \text{ or } (x \geq t) \\ \frac{x-r}{s-r} & \text{if } r \leq x < s \\ \frac{t-x}{t-s} & \text{if } s \leq x < t \end{cases} \tag{15}$$

Author's personal copy

$$\mu_{slow}(x) = \begin{cases} 0 & \text{if } (x < s) \text{ or } (x \geq v) \\ \frac{x-s}{t-s} & \text{if } s \leq x < t \\ \frac{v-x}{v-u} & \text{if } u \leq x < v \\ 1 & \text{if } t \leq x < u \end{cases} \tag{16}$$

$$\mu_{veryslow}(x) = \begin{cases} 0 & \text{if } x < u \\ \frac{x-u}{v-u} & \text{if } u \leq x < v \\ 1 & \text{if } x \geq v \end{cases} \tag{17}$$

After calculating the membership functions *via* fuzzy process, we locally de-fuzzify by applying the function $D_{centroid}$, which is given by the expression below.

$$D_{centroid}(x) = \frac{\int_0^w x\mu(x)dx}{\int_0^w \mu(x)dx} \tag{18}$$

This local de-fuzzified value is used to evaluate status of the response time and judge this metric based on the rules that is extracted from the knowledge base, which is created by the administrator. Because of the similarity in the reasoning procedure, other metrics such as throughput, bandwidth and resource utilization can be calculated similarly by defining the suitable thresholds.

4.5 Metric priority

There is a bit different between the physical servers and the virtual machines in terms of using and providing the services, and then the reasoning procedure for each case should be quite different. Specifically, the physical servers consider any issue occurring on the resource utilization as the main reason for errors. These errors subsequently affect the quality of services. Due to this reason, the issue related to the resource utilization needs to be marked as a failure. For the virtual machines, the issue of the resource utilization only affects the individual clients, and then the errors related to this metric are marked as the regular problems only and assigned a lower priority than the other metrics. Similarly, the bandwidth makes a major influence on the quality of services of the virtual machines. Therefore, this metric should be considered as an important factor. In order to avoid the multi-event problem affecting the result of reasoning, the priority of the metrics will be specified with regard to each target system as below:

With physical servers:

1. Resource utilization
2. Throughput
3. Response time
4. Bandwidth

With virtual machines:

1. Response time
2. Throughput

Author's personal copy

3. Bandwidth
4. Resource utilization

By the end of the approximate reasoning, the issues causing the system failures can be identified fairly and accurately within a fast reaction rate.

5 Performance evaluation

The proposed architecture is implemented and tested on the datacenter of Vietnam Datacommunication Company (VDC). The computing system consists of 7 IBM 3650 M3 servers with the following specification: CPU Quad Core Intel Xeon E5600 series—2.4 GHz—12M cache, SAS 600 GB, 8 GB RAM DDR3, Linux CentOS 5.8. For the cloud configuration, OpenNebula 2.2 and XEN Hypervisor 4.0.4 are chosen as the orchestrator and the hypervisor, respectively. This system hosts around 75–85 virtual machines as time progresses. The simulated errors consist of four categories: resource utilization, throughput, response time and bandwidth. These errors are randomly dispatched into the system following the uniform distribution with the probability of 0.3. In addition to the implementation of proposed method, the Algorithmic-Based Fault Tolerance (ABFT) is also implemented for the purpose of comparison. There are three measurements for the evaluation: the accuracy, the response time of the fault detection and the system reliability. As time goes by, the data are cumulatively calculated after each period of 5 min.

In the accuracy test, according to Fig. 6a and Table 1, the proposed method scores around 11% better than the ABFT by the end of the experiment. In the response time test, although the reaction rate is fluctuated, the proposed method is measured to be 23.4% faster than the ABFT in the best case (Fig. 6b). To summarize, the proposed approach is quite good to pro-actively trigger the fault detection when failures occur in the cloud computing system. Furthermore, this improvement significantly reduces the degradation of system reliability, which is caused by the simulated errors as mentioned above. It is worth noting that the measurement of system reliability is calculated based on the ratio of "failed nodes" over the properly working ones. For more information, the terminology 'failed nodes' stands for the physical servers and/or the virtual machines that can not be or wrongly recovered. Intuitively, the result can be clearly seen in Fig 7 and Table 2.

The last part is about the evaluation of the prediction in terms of accuracy. In this evaluation, we use the assessment of CPU and memory as the demonstrated metrics. Whereby, the prediction procedure of other metrics is very similar. Thus, in order to make the presentation more focused, these procedures are not included in this section. In order to assess the aforementioned metrics, we would like to include the error measurement mechanism. The error measurement is also referred to as the error function which is based on the root mean square error (RMSE) method. Note that the RMSE method is stricter than the popularly used mean square error (MSE) method. Let n stand for the size of dataset, and by using the RMSE error function, the

Author's personal copy

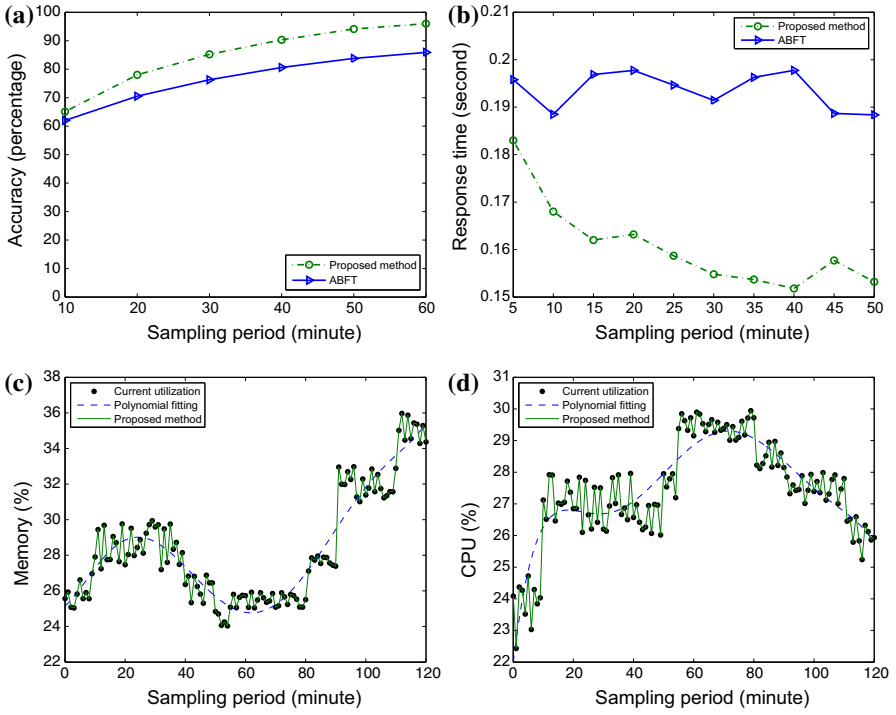


Fig. 6 **a** Accuracy evaluation in fault detection (*higher* is better). **b** Response time evaluation in fault detection (*lower* is better). **c** Accuracy evaluation in memory utilization prediction. **d** Accuracy evaluation in CPU utilization prediction. Performance evaluation of proposed method on cloud computing system.

Table 1 Summary of algorithmic accuracy

Time (min)	20	30	40	50	60
Proposal (%)	78	85.2	90.3	94.1	96
ABFT (%)	70.5	76.3	80.6	83.8	85.9

gap between the current value and the predicted one can be evaluated at every steps of the anticipation. Following is the represented equation of the RMSE:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} \mathcal{R}_i^{(m)} - \mathcal{P}_i^{(m)}}{n}} \tag{19}$$

where $\mathcal{R}_i^{(k)}$ and $\mathcal{P}_i^{(k)}$ stand for the current and the predicted values of the statistics at the monitoring window m, respectively. The variable i stands for the order of the value in the data array. As shown in Fig. 6c, d, the proposed method outperforms the polynomial fitting regression in both memory and CPU utilization predictions, respectively. It is worth noting that the polynomial fitting regression is very popular in various prediction applications because of the convenience in implementation.

Author's personal copy

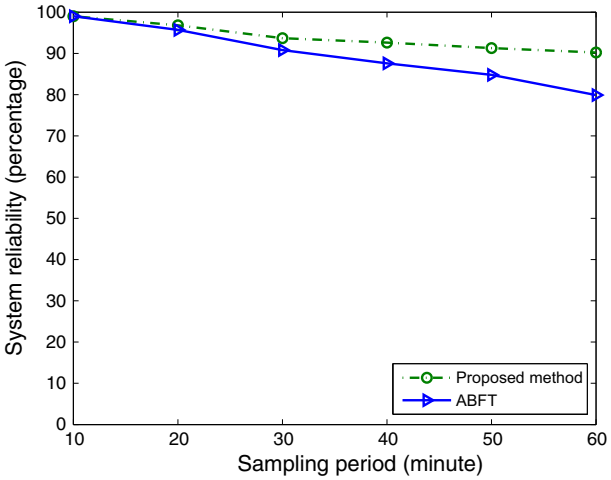


Fig. 7 System reliability over benchmarking time (higher is better)

Table 2 Summary of system reliability evaluation

Time (min)	20	30	40	50	60
Proposal (%)	96.8	93.7	92.6	91.3	90.2
ABFT (%)	95.7	90.8	87.6	84.8	79.9

6 Conclusion

In this research, we introduce a proactive approach to early detect the failures in IaaS cloud computing based on the fuzzy logic method and the Gaussian process regression technique. With the proposed architecture, the faults can be detected accurately without requiring the precise input dataset. With this advantage, the treatment for the faults can be engaged in advance and improves the system reliability significantly. Besides, because the reaction rate is accelerated, then the cost paying for this solution is quite acceptable. For the future works, we might engage the probabilistic approach to alternate the knowledge base to increase the ability of "self-decision making" for the fault tolerance system.

Acknowledgements This work is supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) NRF-2014R1A2A2A01003914 and the Industrial Core Technology Development Program (10049079, Development of mining core technology exploiting personal big data) funded by the Ministry of Trade, Industry and Energy (MOTIE, Korea). This work is also supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (2011-0030079). Corresponding author is Prof. Sungyoung Lee.

Compliance with Ethical Standard

Conflict of Interest The authors declare that they have no potential conflict of interest.

Author's personal copy

References

1. Jhawar R, Piuri V, Santambrogio M (2013) Fault tolerance management in cloud computing: a system-level perspective. Syst J IEEE 7(2):288–297
2. Jhawar R, Piuri V, Santambrogio M (2012) A comprehensive conceptual system-level approach to fault tolerance in cloud computing. In: Systems conference (SysCon), IEEE international. IEEE, pp 1–5
3. Lu K, Yahyapour R, Wieder P, Yaqub E, Abdullah M, Schloer B, Kotsokalis C (2016) Fault-tolerant service level agreement lifecycle management in clouds using actor system. Future Gener Comput Syst 54:247–259
4. Deng J, Huang SC-H, Han YS, Deng JH (2010) Fault-tolerant and reliable computation in cloud computing. In: GLOBECOM workshops (GC Wkshps), 2010 IEEE, pp 1601–1605
5. Singh TK, RaviTeja GT, Pappala PS (2013) Fault tolerance-challenges, techniques and implementation in cloud computing. Int J Sci Res Publ 3(6):698–703
6. Amin Z, Sethi N, Singh H (2015) Review on fault tolerance techniques in cloud computing. Int J Comput Appl 116(8):11–17
7. Tamura Y, Yamada S (2016) Practical reliability and maintainability analysis tool for an open source cloud computing. Qual Reliab Eng Int 32(3):909–920
8. Reiser HP (xxxx) Byzantine fault tolerance for the cloud, University of Lisbon Faculty of Science, Portugal. http://cloudfit.di.fc.ul.pt
9. Kaushal V, Bala A (2011) Autonomic fault tolerance using haproxy in cloud environment. Int J Adv Eng Sci Technol 7(2):222–227
10. Malik S, Huet F (2011) Adaptive fault tolerance in real time cloud computing. In: Services (SERVICES) (2011) IEEE world congress on. IEEE 2011, pp 280–287
11. Chihoub H-E, Antoniu G, Pérez M (2011) Towards a scalable, fault-tolerant, self-adaptive storage for the clouds. In: EuroSys' 11 doctoral workshop
12. Ballard G, Carson E, Knight N (2009) Algorithmic-based fault tolerance for matrix multiplication on amazon ec2 COMPSCI 262A class project. https://people.eecs.berkeley.edu/~knight/ballardcarsonknight_paper.pdf
13. Tamura Y, Yamada S (2015) Software reliability analysis considering the fault detection trends for big data on cloud computing. In: Industrial engineering, management science and applications. Springer, pp 1021–1030
14. Jiang Y, Huang J, Ding J, Liu Y (2014) Method of fault detection in cloud computing systems. Int J Grid Distrib Comput 7(3):205–212
15. What is open nebula? http://docs.opennebula.org/4.12/index.html. Accessed 13 Aug 2015
16. What is ganglia? http://ganglia.sourceforge.net/. Accessed 13 Aug 2015
17. What is ha proxy? http://www.haproxy.org/. Accessed 13 Aug 2015
18. Muller K, Mika S, Ratsch G, Tsuda K, Scholkopf B (2001) An introduction to kernel-based learning algorithms. IEEE Trans Neural Netw 12(2):181–201
19. Rasmussen C, Williams C (2005) Gaussian processes for machine learning, ser. adaptive computation and machine learning. MIT Press, (Online). http://www.gaussianprocess.org/gpml/chapters/
20. Chalupka K, Williams CKI, Murray I (2013) A framework for evaluating approximation methods for gaussian process regression. J Mach Learn Res 14:333–350
21. Bui D-M, Nguyen H-Q, Yoon Y, Jun S, Amin MB, Lee S (2015) Gaussian process for predicting cpu utilization and its application to energy efficiency. Appl Intell 43(4):874–891
22. Wu X (1999) Performance evaluation, prediction and visualization of parallel systems, ser. the international series on asian studies in computer and information science. Springer US, (Online). http://books.google.co.kr/books?id=IJZt5H6R8OIC
23. Feitelson DG (2003) Metric and workload effects on computer systems evaluation. Comput 36(9):18–25. doi:10.1109/MC.2003.1231190
24. Kounev S (2008) Software performance evaluation. Wiley Encyclopedia of Computer Science and Engineering, New York
25. Andras P, Idowu O, Periorellis P (2006) Fault tolerance and network integrity measures: the case of computer-based systems. In: Symposium on network analysis in natural sciences and engineering, p 3