Fuzzy Fault Detection in IaaS Cloud Computing

Dinh-Mao Bui
Computer Engineering Department
Kyung Hee University
Suwon 446-701, Korea
mao.bui@khu.ac.kr

Thien Huynh-The
Computer Engineering Department
Kyung Hee University
Suwon 446-701, Korea
thienht@oslab.khu.ac.kr

Sungyoung Lee
Computer Engineering Department
Kyung Hee University
Suwon 446-701, Korea
sylee@oslab.khu.ac.kr

ABSTRACT

Availability is one of the most important requirements in the production system. Keeping the level of high availability in Infrastructure-as-a-Service (IaaS) cloud computing is a challenge task because of the complexity of service providing. By definition, the availability can be maintain by using fault tolerance approaches. Recently, many fault tolerance methods have been developed, but few of them focus on the fault detection aspect. In this paper, after a rigorous analysis on the nature of failures, we would like to introduce a technique to identified the failures occurring in IaaS system. By using fuzzy logic algorithm, this proposed technique can provide better performance in terms of accuracy and detection speed, which is critical for the cloud system.

Categories and Subject Descriptors

B.8 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance; D.4.5 [**Reliability**]: Fault-tolerance

General Terms

Algorithms, Reliability

Keywords

Approximate Reasoning, IaaS, Cloud Computing, Fault Detection, Fuzzy Logic.

1. INTRODUCTION

Cloud computing is a technology to deliver the infrastructure resources such as computation, storage or network bandwidth over the Internet. This technology is actually based on the mechanism to elastically allocate the resources. Subsequently, the users can remotely initiate their services on-line and pay only for the amount and duration that they use in reality. The background philosophy for this mechanism is the idea of reusability in large-scale system. In addition, the scalability and the muti-tenancy are also the advantages of cloud computing which is transparent to the users. With these kinds of features, the user can focus more on the business model instead of managing the underlying computing resources.

Among the technologies implemented in the cloud computing, the virtualization is one of the most basic things to build the system. In mechanism, the users access the physical resources *via* the virtual machines, which are created in the virtualization. These two layers are part of a three-layer system: physical layer, virtualization layer and application layer. The multi-layer makes difficult to unify the management in cloud computing, especially in failures management [1]. In fact, managing the failures in cloud system is considered as a very high complexity duty because of various kinds of fault sources.

Discussion on fault and fault tolerance is interesting. Indeed, even the cloud platforms consider a huge amount of functionalities, none of them is related to fault tolerance or very basic level of fault tolerance [2]. Clearly, to make an effective solution dealing with the fault, all kinds of potential faults should be collected and classified in advance to issue the corresponding treatment [3]. However, not many studies concentrate in proposing serious improvement for detecting the faults in cloud system. Because of that, the objective of this paper first to analyze the current fault handling solutions in cloud platform. Besides, the sources where the faults originates from should be also considered thoroughly. After that, we propose our solution to detect the fault based on fuzzy logic and conduct the experiment to verify the effectiveness of this solution to the real world IaaS cloud system.

This paper is organized as follows. In Section 2, we provide a summary of the related works that are relevant to this topic. We detail the fault classification in Section 3. In Section 4, we introduce the architecture and fuzzy algorithm to detect failures in IaaS cloud computing.The performance evaluation is conducted on Section 5. Our conclusion and direction for future work are summarized in Section 6.

2. RELATED WORKS

There are two kinds of fault tolerance models which are reactive and proactive approach [4] [5]. The reactive model reduces the effects of fault on system after these faults occurred [6]. In the other hand, the proactive model avoids recovering from faults by predicting and pro-actively replacing the suspected component with the other 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

IMCOM '16, January 04-06, 2016, Danang, Viet Nam
© 2016 ACM. ISBN 978-1-4503-4142-4/16/01…$15.00
DOI: http://dx.doi.org/10.1145/2857546.2857612

tolerant applications, the author design a modular architecture for BFT replication and build blocks for BFT consensus (configuration for various trust settings). Although BFT is a effective algorithm in the case of fault tolerance, the cost paying for redundancy and voters is expensive in terms of time consumption and computation. Therefore, this approach might 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 troubled, the autonomic fault tolerance technique might detect and handle the failure to reassure the system reliability and availability. In addition, the authors also proposed the cloud virtualized system architecture using HAProxy for monitoring. However, some important metrics are missed which then decrease the accuracy of fault detection.

Sheheryar 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 configurations as replications, this approach can provide a high level of reliability. However, there are some drawbacks. First, the VMs, which belong to different users, can not have the same configurations. Second, the VMs with high workload may not pass the reliability test due to the insufficient resources. This kind of VMs may be wrongly replaced without saving the working result. Intuitively, this approach is not a good choice for 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 for the cloud. 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 goals of the storage system. The research partially focused on creating fault tolerance scheme for cloud's storage rather than making a general solution for service providing cloud system.

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

Lastly, in the research of ”Method of Fault Detection in Cloud Computing Systems” [13], the authors proposed a detection approach based on C4.5 decision algorithm. This algorithm is also combined with the characteristics of the cloud to improve the correctness and the 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 detection exists, not many researchers have thoroughly considered the detection method with regards to the characteristics of the production cloud system. Because of that, there is a need to develop a low complexity but efficient method to detect the failures. This is also our motivation to propose the fuzzy logic based fault detection approach.

3. BACKGROUND

3.1 Cloud system

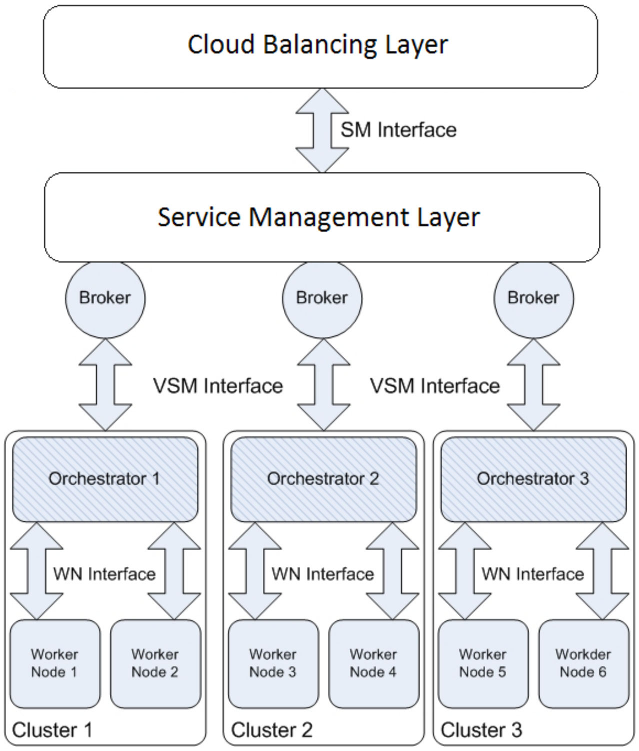


Figure 1: IaaS Cloud Computing Model.

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 efficiently load balance between clusters. In our research, we implement the following IaaS cloud computing model (Figure 1) to cover most of the requirement of service provider in real world:

- Cloud Balancing Layer: this layer is a combine component with a series of filters and authorization system. With this component, input requests are checked the validity and be classified. By using the predefined rules which created by administrator, the requests might be forwarded, be answered based on the query results from the cache system or be refused. After classification process, the requests will be passed to the appropriate servers for load balancing purpose.
- Service Management (SM) Layer: this layer has responsibility to automate the provisioning cloud resources by collecting capacity information and redirecting the virtual machine deployment request.
- Virtualization Management (VSM) Layer: this layer has responsibility to deploy virtual machine to the Worker Node. In fact, this layer receives the requests from the service management layer and chooses the best worker node which satisfies all constraints for the VM deployment.

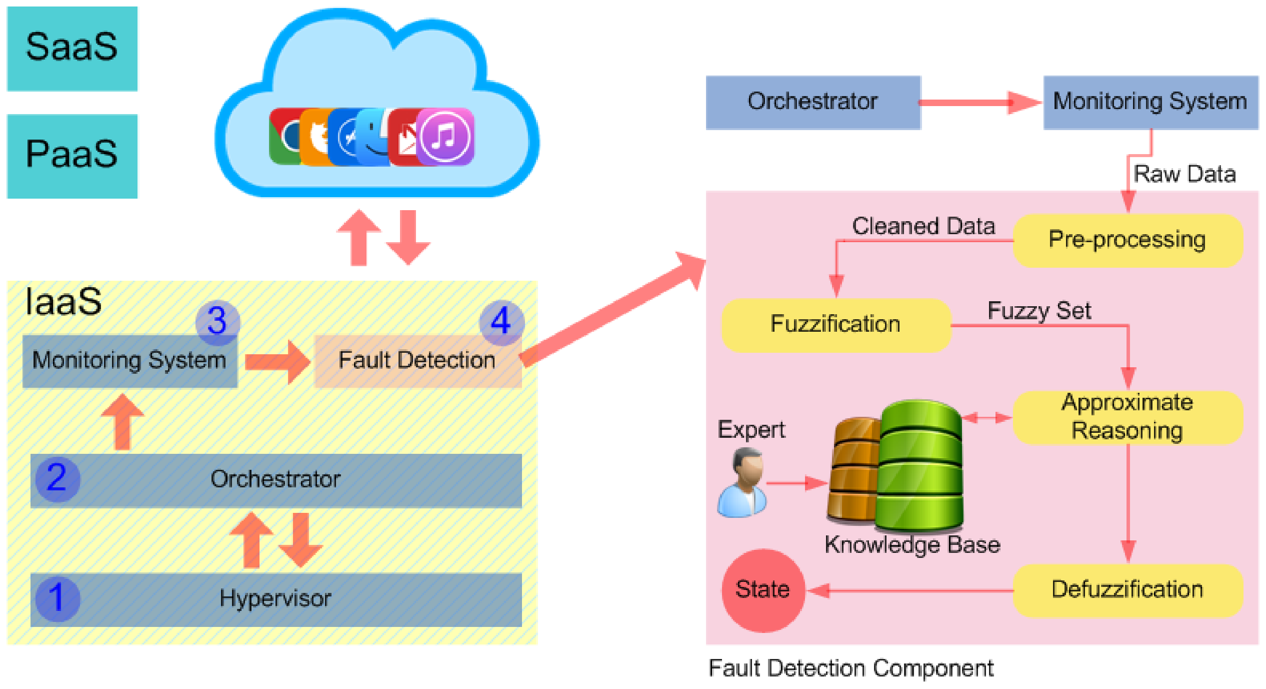


Figure 2: Proposed fault detection architecture.

- Worker Node (WN): this is actually the physical servers. Many physical servers are put together into clusters. In each worker node, we implement the abstraction layer of virtualization by using the hypervisor such as Hyper-V, XEN or VMware $\cdots$etc. The hypervisor interacts with physical resources and manages all the VMs in worker node.

3.2 Failures in IaaS

Fault types are classified according to the positions where they occur. The fault can come from the physical infrastructure (worker node failure), from the virtualized infrastructure (VM failure) or from the virtual infrastructure manager (orchestrator failure). In the Open Nebula orchestrator [14], 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 pre-defined hook can be dispatched to solve the problem. Usually, this hook will redeploy the failed VM to another worker node.

- Virtual machine failure: There are two failure situations which can occur in virtual machine life cycle: the VM fails and VM crashes. In the VM fails, the error in network prevents the image to be copied into the node. When this situation happens, the VM enters the 'failed' state. To solve this situation, the same method to solve the worker node failure is used. In the VM crash, the physical server sometimes crashes due to unknown reasons. In this case, a script might be embedded to restart the VM automatically.

- Orchestrator failure: the orchestrator can recover from a failures by restarting the core daemon. The running VMs will be reconnected and run as nothing happens. In case of the pending VMs, these VMs might be re-deployed on a suitable host, as well as other non-transient states. However, VMs not in a 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 VM. 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 reactive policies, it actually supports to reduce the effect of failures in the worker nodes, in the VMs and in the core of orchestrator. In the next section, an architecture enables the proactive fault tolerance policies is proposed.

4. PROPOSED METHOD

The proposed architecture described in Figure 2 is used to identify the failures in IaaS cloud computing system. In this system, there are two types of computing nodes: physical servers and virtual servers [15]. To handle these nodes, the architecture is equipped with two components: fault detection and monitoring system. These components are built independently with the orchestrator [16] [17] [18] and integrated into the IaaS cloud computing system as the plugins. The functionalities of the architecture is described as follows.

- Monitoring component: this component includes two applications: Ganglia [19] and HA Proxy [20]. The main task of this component is to collect system parameters as the input dataset for the fault detection

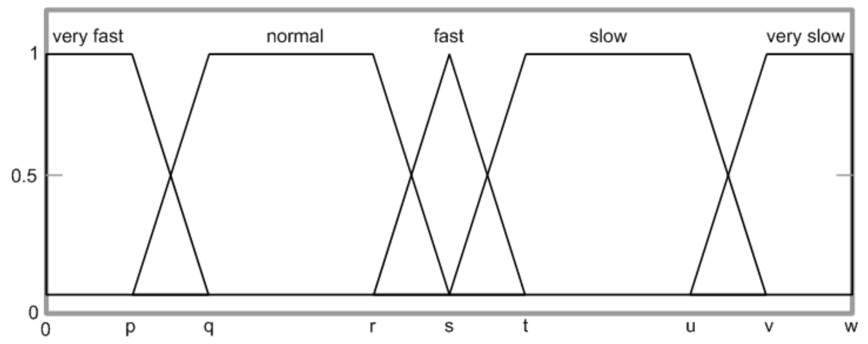


Figure 3: Fuzzy response time metric.

process. Ganglia is used to collect the different parameter values such as CPU usage, RAM usage, disk I/O...etc. In the other hand, HA Proxy [5] is aimed to monitor network parameters such as response time, bandwidth, throughput, request/response ratio...etc.

- Fault Detection Component: the fault detection technique implemented in this component is based on fuzzy logic.

The attractive benefit of fuzzy logic algorithm is approximate reasoning with imprecise propositions by using fuzzy set theory. In fault detection component, the rule form of fuzzy logic algorithm formulates as below:

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

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 degree 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.

System parameters collected by Ganglia and HA Proxy will be used as input data set to identify the problem. The first step to build fault detection component is to identify the metrics [21]. Based on studies of performance and network measurement [22] [23] [24], the most fundamental metrics can be considered as follows:

- Response time: the time elapsed between the moment that the node receiving the request to the time that node returns the response.

- Throughput: the amount of data transfers in a given unit of time.

- Bandwidth: the bandwidth of computing nodes.

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

Because of similarity in reasoning procedure, we would like to present the estimation process for one metric: the response time.

4.1 Response time

Five fuzzy sets are used to describe the value of response time: very fast, fast, normal, slow, very slow. The value of input parameter rt determines the value of fuzzy variables. Figure 3 explains how the determination process is conducted. Depending on the network status, the value of fuzzy variable will change from 0 to 1. Trapezoidal membership functions μ are described by parameter rt, (p, q, r, s, t, u, v, and w are thresholds). These parameters are given by the expressions below.

$$\mu_{veryfast}(rt) = \begin{cases} 0 & \text{if } rt \geq q \\ \frac{q-rt}{q-p} & \text{if } p \leq rt < q \\ 1 & \text{if } rt < p \end{cases} \quad (2)$$

$$\mu_{fast}(rt) = \begin{cases} 0 & \text{if } (rt < p) \text{ or } (rt > s) \\ \frac{rt-p}{q-p} & \text{if } p \leq rt < q \\ \frac{s-rt}{s-r} & \text{if } r \leq rt < s \\ 1 & \text{if } q \leq rt < r \end{cases} \quad (3)$$

$$\mu_{normal}(rt) = \begin{cases} 0 & \text{if } (rt < r) \text{ or } (rt \geq t) \\ \frac{rt-r}{s-r} & \text{if } r \leq rt < s \\ \frac{t-rt}{t-s} & \text{if } s \leq rt < t \end{cases} \quad (4)$$

$$\mu_{slow}(rt) = \begin{cases} 0 & \text{if } (rt < s) \text{ or } (rt \geq v) \\ \frac{rt-s}{t-s} & \text{if } s \leq rt < t \\ \frac{v-rt}{v-u} & \text{if } u \leq rt < v \\ 1 & \text{if } t \leq rt < u \end{cases} \quad (5)$$

$$\mu_{veryslow}(rt) = \begin{cases} 0 & \text{if } rt < u \\ \frac{rt-u}{v-u} & \text{if } u \leq rt < v \\ 1 & \text{if } rt \geq v \end{cases} \quad (6)$$

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

$$D_{centroid}(rt) = \frac{\int_0^w rt\mu(rt)drt}{\int_0^w \mu(rt)drt} \quad (7)$$

This local de-fuzzified value is used to evaluate status of the response time and present this metric into rules.

- Rule $rt1$: if (Response Time is slow) then (networkProblem is Risk).

- Rule $rt2$: if (Response Time is very slow) then (networkProblem is Danger).
- Rule $rt3$: if (Response Time is not slow) and (Response Time is not very slow) then (networkProblem is none).

Because of the similarity in reasoning procedure, other metrics such as throughput, bandwidth and resource utilization can be calculated similarly.

4.2 General approximate reasoning

There is a bit different between physical and virtual servers in terms of using and providing the services, then the reasoning procedure for each case will be quite different. Specifically, the physical servers consider any issue occurring on the resource utilization as the main reason for several errors, which subsequently affect the quality of services. These issues need to be marked as failures. For the virtual servers, the issues of the resource utilization only affect the individual clients, then the errors related to this metric might be marked as problem only and have lower priority than the other metrics. Similarly, the bandwidth makes a major influence on the quality of services of virtual servers, this metric should be considered as an important factor. In order to avoid multi-event affecting the result of reasoning, the priority of the metrics will be specified as follows.

With physical servers:

1. Resource Utilization
2. Throughput
3. Response Time
4. Bandwidth

With virtual servers:

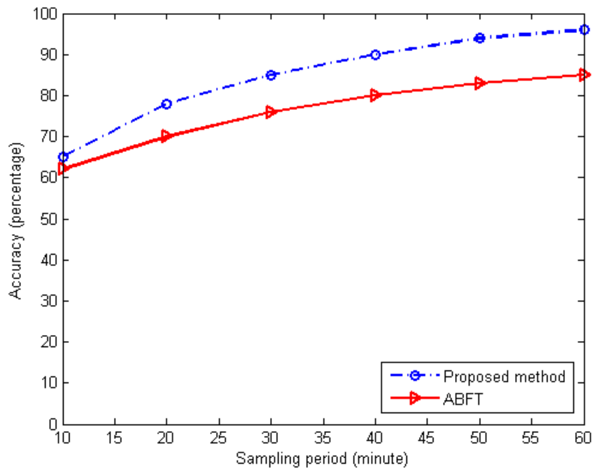
1. Response Time
2. Throughput
3. Bandwidth
4. Resource Utilization

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

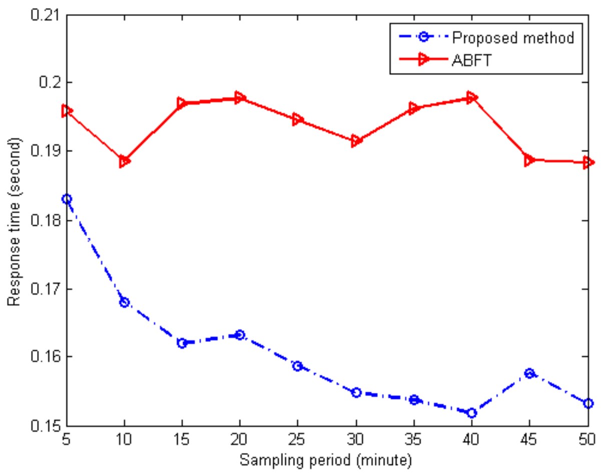
5. PERFORMANCE EVALUATION

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 - 2.4GHz - 12M cache, SAS 600 GB, 8GB RAM DDR3, Linux CentOS 5.8. For the cloud configuration, OpenNebula 2.2 is chosen for the orchestrator and XEN Hypervisor 4.0.4 is chosen for the hypervisor. This system hosts 75 to 85 virtual servers from time to time.

The errors consist of four categories: resource utilization, throughput, response time and bandwidth. These error are randomly dispatched into the system. In addition to the implementation of proposed method, the Algorithmic-Based Fault Tolerance (ABFT) is also implemented for the purpose of comparison. There are two measurements for the evaluation: the accuracy and the response time of fault detection.



(a) Accuracy statistics in fault detection (higher is better).



(b) Response time statistic in fault detection (lower is better).

Figure 4: Performance evaluation of proposed method on cloud computing system.

The results are cumulatively calculated after a period of 5 minutes. In the accuracy test, according to Figure 4*a* the proposed method scores 13% 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 (Figure 4*b*). To summarize, these results are quite good to trigger the treatments for failures occurring in the cloud computing system.

6. CONCLUSION

In this research, we introduce a new approach to detect failures in the IaaS Cloud Computing. With proposed architecture, the faults can be classified accurately without requiring the precise input dataset. Besides, because the reaction rate is very fast, then the cost paying for this solution is acceptable. For the future works, we might engage the probabilistic approach to increase the ability of "self-decision making" for fault tolerance system.

7. ACKNOWLEDGMENT

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).

8. REFERENCES

[1] R. Jhawar, V. Piuri, and M. Santambrogio, "Fault tolerance management in cloud computing: A system-level perspective," *Systems Journal, IEEE*, vol. 7, no. 2, pp. 288–297, 2013.

[2] ——, "A comprehensive conceptual system-level approach to fault tolerance in cloud computing," in *Systems Conference (SysCon), 2012 IEEE International*. IEEE, 2012, pp. 1–5.

[3] K. Lu, R. Yahyapour, P. Wieder, E. Yaqub, M. Abdullah, B. Schloer, and C. Kotsokalis, "Fault-tolerant service level agreement lifecycle management in clouds using actor system," *Future Generation Computer Systems*, 2015.

[4] J. Deng, S. C.-H. Huang, Y. S. Han, and J. H. Deng, "Fault-tolerant and reliable computation in cloud computing," in *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*. IEEE, 2010, pp. 1601–1605.

[5] T. K. Singh, G. T. RaviTeja, and P. S. Pappala, "Fault tolerance-challenges, techniques and implementation in cloud computing," *International Journal of Scientific and Research Publications*, vol. 3, no. 6, 2013.

[6] Z. Amin, N. Sethi, and H. Singh, "Review on fault tolerance techniques in cloud computing," *International Journal of Computer Applications*, vol. 116, no. 18, 2015.

[7] Y. Tamura and S. Yamada, "Practical reliability and maintainability analysis tool for an open source cloud computing," *Quality and Reliability Engineering International*, 2015.

[8] H. P. Reiser, "Byzantine fault tolerance for the cloud," *University of Lisbon Faculty of Science, Portugal, at http://cloudfit. di. fc. ul. pt*.

[9] V. Kaushal and A. Bala, "Autonomic fault tolerance using haproxy in cloud environment," *International Journal of Advanced Engineering Sciences and Technologies*, vol. 7, no. 2, pp. 222–227, 2011.

[10] S. Malik and F. Huet, "Adaptive fault tolerance in real time cloud computing," in *Services (SERVICES), 2011 IEEE World Congress on*. IEEE, 2011, pp. 280–287.

[11] H.-E. Chihoub, G. Antoniu, and M. Pérez, "Towards a scalable, fault-tolerant, self-adaptive storage for the clouds," in *EuroSys' 11 Doctoral Workshop*, 2011.

[12] Y. Tamura and S. Yamada, "Software reliability analysis considering the fault detection trends for big data on cloud computing," in *Industrial Engineering, Management Science and Applications 2015*. Springer, 2015, pp. 1021–1030.

[13] Y. Jiang, J. Huang, J. Ding, and Y. Liu, "Method of fault detection in cloud computing systems," *International Journal of Grid and Distributed Computing*, vol. 7, no. 3, pp. 205–212, 2014.

[14] "What is open nebula?" http://docs.opennebula.org/4.12/index.html, accessed: 2015-08-13.

[15] S. Alrwais, "Behind the scenes of iaasimplementations."

[16] S. McIlvenna, M. Dumas, and M. T. Wynn, "Synthesis of orchestrators from service choreographies," in *Proceedings of the Sixth Asia-Pacific Conference on Conceptual Modeling-Volume 96*. Australian Computer Society, Inc., 2009, pp. 129–138.

[17] C. Liu, Y. Mao, J. Van der Merwe, and M. Fernandez, "Cloud resource orchestration: A data-centric approach," in *Proceedings of the biennial Conference on Innovative Data Systems Research (CIDR)*, 2011, pp. 1–8.

[18] A. J. Younge, R. Henschel, J. T. Brown, G. Von Laszewski, J. Qiu, and G. C. Fox, "Analysis of virtualization technologies for high performance computing environments," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE, 2011, pp. 9–16.

[19] "What is ganglia?" http://ganglia.sourceforge.net/, accessed: 2015-08-13.

[20] "What is ha proxy?" http://www.haproxy.org/, accessed: 2015-08-13.

[21] X. Wu, *Performance Evaluation, Prediction and Visualization of Parallel Systems*, ser. The International Series on Asian Studies in Computer and Information Science. Springer US, 1999. [Online]. Available: http://books.google.co.kr/books?id=IJZt5H6R8OIC

[22] D. G. Feitelson, "The effect of metrics and workloads on the evaluation of computer systems."

[23] S. Kounev, "Software performance evaluation," *Wiley Encyclopedia of Computer Science and Engineering*, 2008.

[24] P. Andras, O. Idowu, and P. Periorellis, "Fault tolerance and network integrity measures: the case of computer-based systems," in *Symposium on Network Analysis in Natural Sciences and Engineering*, 2006, p. 3.