RT-PUSH: A VM FAULT DETECTOR FOR DEADLINE-BASED TASKS IN CLOUD

Sampa Sahoo, Bibhudatta Sahoo, Ashok Kumar Turuk
Department of Computer Science and Engineering, National Institute of Technology, Rourkela, India
Email:(sampaa2004, bibhudatta.sahoo, akturuk)@gmail.com

1 Abstract

Nowadays, the cloud is becoming an important paradigm due to cost-efficiency, scalability, availability and high resource utilization. An increasing number of deadline-based (real-time) applications are moving to cloud for high availability and low latency. Applications like financial transactions, health-care system, scientific workflows with their real-time nature, i.e., deadline bound, need provisioning of computing services, despite the presence of failure. So, cloud running deadline-based applications need a fault-tolerant framework that can assure availability and responsiveness. Fault can occur in any layer of cloud; physical, virtual, or application and can be handled by various fault management methods, which include fault avoidance, fault detection, and fault recovery. Here, our primary focus is on a Virtual Machine (VM) failure and its detection. We showed the VM behavior in normal working condition and with the failed state through state transition diagram. A timeout based VM fault detector RT-PUSH proposed for cloud running real-time applications. We used success ratio and execution drop rate as performance metrics to evaluate the effectiveness of the proposed fault detector.

2 Introduction

Cloud computing is a revolutionary paradigm, which allows on-demand provisioning of applications, platforms, and computing resources. Other unique features of the cloud include scalability, high availability, an economy of scale, etc [1, 2]. The layered architecture of cloud system presented in Figure 1. consists of physical, virtualization and application layer. A physical layer made up of physical resources, e.g., CPU, memory, RAM, network, etc., are used for computation, communication, and storage of data. Virtualization layer made up of hypervisor and VMs. Virtualization provides physical resource partitioning with adaptability, privacy, and resource sharing among VMs. Basic building blocks of application layer include user interfaces and functionalities, Application Programming Interfaces (APIs), etc. The dynamic nature of cloud environment leads to unexpected system behavior, resulting in defects and failure. Despite several benefits, the increased functionality, and complexity of the larger system, cause higher resource failure probability in the cloud data center.

Fault can occur in any layer of the cloud system and gradually affect the other layers too. For example, if the physical layer becomes faulty then it also strikes the virtualization layer as well as the execution time of applications running on it. So, it is essential to handle faults at each layer. Cloud service disruption may occur due to human error, software or hardware failure, and disasters. Fault tolerance represents the ability of the cloud system to safeguard and protect the delivery of assigned tasks even in the presence of failure. The job of fault-tolerant elements is to identify failures and resolve them as soon as possible [3]. Fault tolerance is one of the key issues to ensure reliability, availability, and robustness of services running in the cloud, especially for deadline based services.

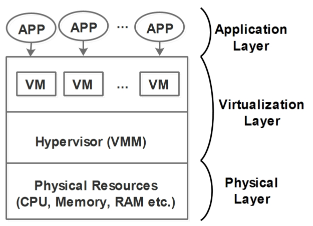

Figure 1: Cloud System Architecture Layers

As deadline-based (real-time) applications like financial transactions and scientific computing move to the cloud, it becomes critical for cloud service providers to guarantee Quality of Service (QoS) properties (e.g., timeliness, high availability, reliability) of these applications. These deadline-bound applications want uninterrupted services despite the occurrence of failures, which becomes a tough challenge for cloud [4]. An efficient mechanism expected to manage and deploy cloud resources in a way that maximize resource utilization, guarantee timeliness and high availability requirements. Fault tolerance for real-time applications becomes a challenge for the cloud system as it may increase systems response time. The absence of fault-tolerant capability may affect the QoS or SLA of applications, so if a fault occurred in a cloud system, then it should be detected and recovered without affecting the outcome [5].

Fault avoidance, detection, and recovery techniques used to achieve the fault tolerance in the cloud. Fault avoidance ensures cloud system does not fail and can be bypassed through replication and checkpoint based methods. If fault avoidance does not work properly, failure may occur, the only things left is to detect and recover from it. Various fault detection and recovery methods are acceptance test, heartbeat mechanism, checkpoint, VM migration, resubmission, redundancy, scheduling, etc. Some common solutions for high availability despite hardware and software failure are mirroring, replication, failover clustering, and snapshot [6]. The fault tolerant mechanism used in [4, 7] are Primary-Backup (PB) and VM migration. Scheduling plays a significant role in satisfying the applications QoS while maximizing resource utilization, despite the presence of hardware and software failures. VMware fault tolerance uses PB model, where execution is done at both primary and backup VMs [8]. Failure detection is a critical component of fault management. This phase is used to find systems fault and provide needed information for services so that problems can be treated to guarantee continuous service in the presence of failure [9]. In this study, our focus is on VM failure, the actual computing unit of the cloud system. Our most significant contribution includes

- We presented VM state transition diagram in normal condition (without fault) and with the failed state. We listed reasons of VM failure and discussed fault management techniques in the cloud.
- We proposed RT-PUSH a VM fault detector based on timeout and deadline for cloud system running real-time task. The effectiveness of the model is evaluated through success ratio and execution drop rate metrics.

The rest of the paper is organized as follows: Section 2 presents related work. Section 3 exhibits VM state transition diagram, causes of failure and fault management techniques. Section 4 shows RT-PUSH VM failure detector and evaluation. Section 5 presents conclusions and future work.

3 Related Work

It is challenging job to achieve fault tolerance in cloud system for real-time applications while optimizing resource utilization and guarantee QoS. In [3] authors proposed scheduling algorithm based on dynamic clustering league championship, to make cloud system fault tolerant. Their approach works as follows: first fault detector module detects fault status of the VM and then task migration performed to reassign tasks of faulty VM to an idle VM. In [4] authors proposed FESTAL, a fault tolerant scheduling algorithm based on PB model and VM migration technology for real-time tasks in the cloud. In [7] authors used fail-signal and acceptance test fault-detection methods to detect host failures. Further, proposed FASTER, a fault tolerant scheduling algorithm for real-time scientific workflows based on overlapping and VM migration. In [10] timeout-based node failure detection techniques push and pull model discussed for large-scale distributed systems. In the push model, every participating node sends a heartbeat message to fault detector. In the pull model, fault detector sends a heartbeat message to participating nodes and wait for their acknowledgment. In both the cases, if a message doesn't reach the destination (fault detector) within a time limit (timeout) then the corresponding node suspected of failure. In [6] authors use heartbeat method for cloud node failure detection and a smart adaptive snapshot replication technique employed to overcome the failure in a cost-efficient way.

Authors in [9] presented a systematic review of high availability solutions for cloud computing, such as check-pointing, load balancing and redundancy. In [8] architectural details of a middleware framework called Local Fault Manager (LFM), and Global Fault Manager (GFM) presented for fault tolerant cloud infrastructure running soft real-time applications. LFM collect resource information and sent to GFM, which is responsible for decision-making. LFM also runs High Availability Service (HAS) for synchronization of primary and backup VMs, making backup VM active when primary VM fails. In [11] a system level, modular replication based fault tolerance management framework discussed, which can be integrated with existing cloud infrastructure and provide the basis for realizing fault tolerance as a service. In [12] a replication based fault-tolerant service implemented with BPEL, where zookeeper is responsible for detecting faults using a callback mechanism called watches. Researchers in [5] surveyed different fault tolerance architecture in cloud computing. Their study shows that a PB method (Byzantine Fault Tolerance (BFT)) is used to detect faulty node, which is based on the output obtained from primary and backup nodes. If the results are different for primary and backup, then a node is considered faulty. Another way of fault detection is based on replication (Gossip Architecture). In [13] replication based fault tolerance approach investigated for real-time cloud computing. Here, the same input copy executed on different VMs, based on their outcome decision made whether to continue further or not. Checkpointing saves the system state in a separate region so that it can be restored if the primary instance fails. VM migration relocates a VM instance from failed Physical Machine (PM) to a new PM. Job migration occurs if a VM fails. In [14] authors use checkpointing and job migration techniques to address the tradeoff between monetary cost and reliability.

4 VM State Transition Diagram

When a VM is deployed for execution of a task or application, it goes through several state changes as shown in Figure 2. The entire VM life progression goes through state changes which are mentioned below:

(i) NEW: Initial state, when a VM first created.

(ii) READY: After hypervisor successfully boots VM, it is ready to be used. VM comes to this state after NEW state and ready to run a task or application. From READY state VM may go to RUNNING state (if a task assigned to the VM) or SHUTDOWN state (if VM not required).

(iii) RUNNING: Tasks are assigned to VM by the scheduler, VM state set to RUNNING and assigned tasks get executed.

(iv) SLEEP: Sometimes to save energy and cost, or during VM migration it goes to SLEEP mode, i.e., temporary off state. If it is required (e.g., to meet QoS) the VMs wake up and VM state changes to RUNNING.

(v) SHUTDOWN: VM instance deleted, and all the resources acquired by it get released.

(vi) FAILED: When VM is not working properly, e.g., give erroneous output, delay the response then VM state goes from RUNNING to the FAILED state.

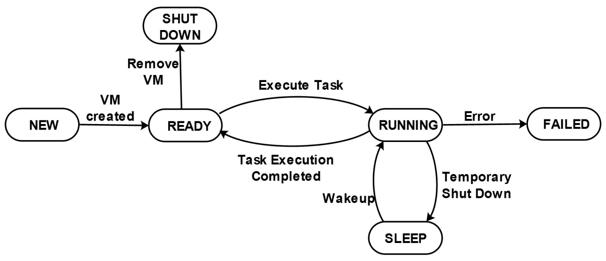

Figure 2: VM State Transition Diagram

Here we assume that, except RUNNING state all other states are fault free. Following transition diagram, Figure 3. shows further details regarding the VM transition diagram with FAILED state. VM at FAILED state, sent

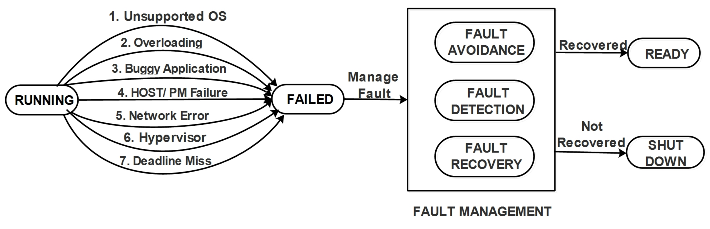

Figure 3: VM State Transition with FAILED state

to fault management for recovery. If recovery possible, then VM state changes to the READY state, otherwise it needs to be deleted, which changes the state to SHUTDOWN.

4.1 Causes of VM Failure

In the cloud, VM is the component which performs actual computation. So, there is a high probability of VM failure. A VM may fail due to following reasons:

(i) Unsupported OS: VM may fail due to the unsupported guest Operating System (OS).

(ii) Overloading: There is a limit on virtual hard disk file size. If the application file size running on it exceeds the maximum supported file size, then VM fails.

(iii) Buggy Application: Applications running on VM are buggy, which makes the VM behave abnormally.

(iv) HOST/ PM Failure: If the host or PM fails due to overload or any other reason (e.g., natural disaster), then VMs running on it fails.

(v) Network Error: VM failure due to network error prevents VM instance being copied into the PM.

(vi) Hypervisor: Unexpected hypervisor error cause VM failure.

(vii) Deadline Miss: A real-time (delay-sensitive) application need to complete its execution before deadline or timing constraint. So, if a VM running these type of applications, misses deadline most of the times (threshold limit), then it is a VM failure.

4.2 Fault Management Techniques

Runtime anomalies, faults at infrastructure and operational error can exhaust cloud resources that can adversely affect all the users sharing it. So, a cloud architect should design a resilient system that can accommodate failures at any layer, yet provide maximum performance and availability for their users. Various methods practiced to guarantee fault tolerance of cloud system are fault avoidance, fault detection, and recovery.

Fault avoidance or fault prevention intends to restrict the introduction of faults. Replication and checkpoint based methodologies usually used to bypass VM failure. A VM fault can be avoided by making use of prior knowledge about the usage of VM resources, i.e., resources available and allocated, future request, etc. Based on all these information scheduler decides whether to assign it new tasks or not and thus evade the chances of VM failure due to overloading, deadline miss. Another way is based on fault probability, where scheduler first analyses fault probability of VM and then make a decision whether to assign the task or not. As VM may suffer from the single point of failure, replication of VM instances can help to sustain failures. The replication policy works on the principle that, possibilities of single VM failure are more as compared to multiple VM failure. With the redundant copies, the cloud can provide uninterrupted service despite VM failure. In checkpointing, state of the machine is saved to stable storage at regular interval. If a failure occurs, VM starts execution from the last-registered checkpoint. But, fault avoidance alone is not sufficient to provide fault tolerance.

Fault detection techniques such as acceptance test, monitoring, timeout-based (heartbeat), replication or redundancy may be used for VM. Timeout-based fault detection may be reactive (push model) or proactive (pull model) [10]. In the reactive technique, VM periodically sends heartbeat (message) to a failure detector. A failure detector suspects a VM if it does not receive the message within a certain time limit (timeout period (T)). In the proactive technique, fault detector checks and monitor services of VMs at regular intervals (timeout period). If it noticed an unexpected behavior of a VM, then take the necessary action. In replication based fault detection, incoming request (task) assigned to primary and backup VMs. Backup VM always synchronized with primary VM, and any discrimination in the VM outputs considered as VM failure. In [6] snapshot based fault detection method used to indicate host failure. A snapshot manager and snapshot agent in each host exchange heartbeat, the absence of which for a certain amount of time indicates host failure. Failure detection mechanism implemented in FESTAL and FASTER are fail-signal and acceptance test, which provides information of host failure. In [13] acceptance test (AT), time checker (TC) and reliability assessor (RA) modules used to check the reliability of a VM. Different replicas of the input provided to VM, where different real-time algorithms run. In RA module reliability of VM evaluated based on AT and TC module output and a VM with reliability less than minimum threshold limit is removed. A fault discovery technique used in [3] works for OS level, VM level, and application level, where VM fault detected by Virtual Machine Manager (VMM) through a fault detection component. In [11] a combination of heartbeat and primary-backup methods used for VM fault detection. Here the primary VM instance sends liveliness request to all backup VM instances, and a timer is set. On receiving liveliness message backup VM sends an acknowledgment message to the primary. If a backup fails to do so for N consecutive requests within the timeout threshold, then it is suspected of failure.

Various *Fault recovery* mechanisms are resubmission, job/VM migration, checkpointing, and replication. In job migration, if a VM failed to perform its service, then the job is transferred to another similar VM instance. VM failure due to host/ PM failure can be recovered by assigning the VM instance to a new host or any under load host, whichever is preferable. In checkpointing, the VM can recover from the failure by starting its execution from the last saved checkpoint. In replication, if one copy (primary) fails, then VM failure salvaged from another copy (backup).

5 RT-PUSH Fault Detector

Failure detector plays a vital role in fault tolerant cloud computing. The job of the failure detector is to detect faults completely, efficiently in the presence of unexpected system behavior and have a high impact on the systems performance. Heartbeat mechanism based VM/PM fault detection is a conventional procedure used by researchers. Here, we proposed a heartbeat mechanism based VM fault detector, RT-PUSH for cloud system running deadline bound applications. Our approach differs from the popular push model in the following sense: (i) In a regular push-based fault detection method; fault detector suffers from a single point of failure (if centralized fault detector fails, then its hard to detect node failure). In the RT - PUSH we use distributed approach, where fault can be detected by both participating VMs and fault detector (hypervisor). (ii) Our approach reduces the message overhead due to increase in the number of participating nodes (VM), as VM sent a message to fault detector only when it suspects a VM. (iii) It is scalable as we can add or remove VMs from the list easily. (iv) Conventional push model only detects timeout fault, but we extend it to detect deadline miss fault also. To detect deadline miss fault, we consider a threshold limit on the number of times a VM can miss the tasks deadline. If any VM crosses this threshold limit, then we identify that VM to be faulty.

5.1 Working Principle

Here we assumed that all VMs in a PM arranged in a circular list. A VM stores, three types of information: its ID, previous VM ID, next VM ID. Each VM sends a heartbeat to its next VM on the list. If a VM does not receive a heartbeat from its previous VM within a certain time limit (timeout period (T)), then it informs hypervisor through the request message (R). The request message contains a previous VM ID. On receiving a request, message hypervisor sends a check message (C) to VM ID specified in R message. If a VM fails to send acknowledgment (A) message within T time, then hypervisor suspect the VM. Hypervisor removes the suspected VM from the list, change the number of VM available, inform precursor VM whose ID is in message R. Now precursor VM changes its next field to the VM that sent R message. After the necessary action, If failed VM is ready to be used then, hypervisor add it to the VM list.

5.2 Example

The overall working of RT-PUSH is shown with an example in Figure. 4. Let there be 4 VMs on a physical machine. All the VMs are arranged in a circular list (VM1-VM2-VM3-VM4-VM1). Steps to detect VM fault are as follows:

1. VM sends heartbeat (Im ok) to its next VM, i.e., VM1 to VM2, VM2 to VM3, VM3 to VM4, VM4 to VM1 respectively.

2. Let VM3 doesn't receive a heartbeat from VM2 as it failed, within the timeout period,T. Then VM3 send a message R with VM ID 2 to the hypervisor.

3. On receiving message R from VM3, hypervisor sends a check (C) message (are you ok) to VM2. If it doesn't receive acknowledgement ($A(ack)$) message from VM2 within T time, it suspects VM2.

4. Hypervisor removes VM2 from the VM list and do the needful. It informs VM3 through info message (I) and direct VM1 to send heartbeat to VM3.

5. After checking the VM status, if it's ok, deadline miss status is checked. Let n_i shows current count for deadline miss of a VM and N_l is the threshold on the number of times a VM can miss the deadline.

6. Every time a VM miss deadline of a task n_i value is updated. If n_i greater than N_l then VM is suspected. In the example n_4, i.e., VM4 counter exceeds the number of times it misses the deadline. So hypervisor isolates it and does the needful.

7. If failed VM recovered, then hypervisor add it to the list. The above example is shown for a single PM, which can be extended to multiple PM also.

5.3 Evaluation

We evaluate the fault detection method with respect to following two metrics:

- Success Ratio (SR): defined as the percentage of tasks that are completed within deadline successfully among all the submitted task.
- Execution Drop Rate (EDR): defined as the total execution time of tasks completed within the deadline with and without VM failure among all the submitted task.

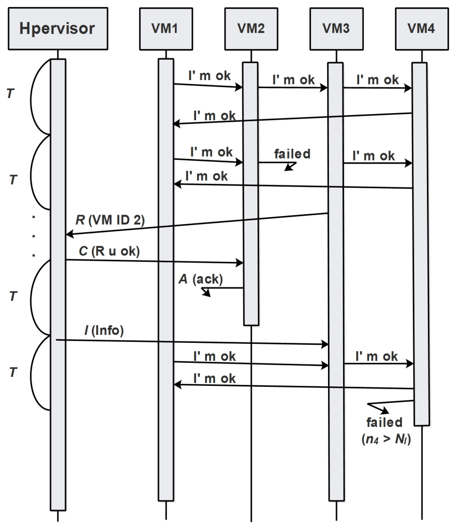

Figure 4: Working of RT-PUSH

The detailed setting and parameters used in experiments are listed as follows:

- Heartbeat period (T) = 10 ms.
- Deadline is calculated as di= ai+ slack time, where slack time is uniformly distributed in the range (3-10) s. di deadline of ith task, ai arrival time of ith task.
- Number of deadline miss threshold (Nl)=20.
- *Scenario 1:* Task count grows from 200 to 1000 in the gap of 200. A PM has 4 types of VM with processing capability equivalent to 1000, 2000, 3000 and 4000. MIPS. Task size uniformly distributed in the range (1000-10000) MI.

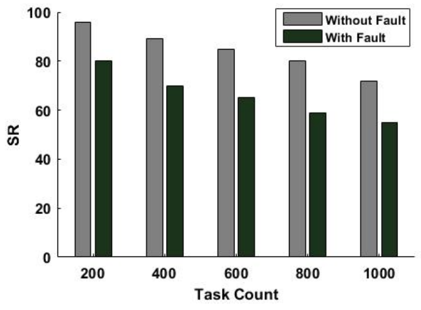

Figure 5: SR Vs Task count

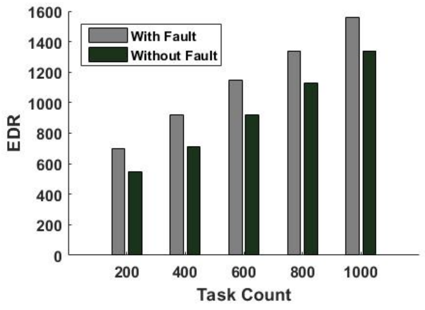

Figure 6: EDR Vs Task count

Figure 5. and 6. show that as the number of tasks increases EDR increases and SR decreases. The rise in task input raises the probability that task will not complete within its deadline. Thus, reduces the success ratio (SR) and as the fault reduces the number of healthy VM, execution time rises.

- *Scenario 2:* A PM has multiple VM in the range [4-20] with speed capability between [2000-4000] MIPS. Task count is fixed to 500 with size 6000 MI. Rest values are mentioned above.

Figure 7. and 8. show that with the increasing number of VMs, the SR increases and EDR is reduced. Because the large number of VM reduces the impact of VM fault, as chances of failed VM instance replaced by another healthy VM increases.

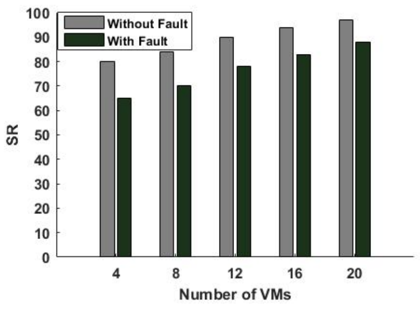

Figure 7: SR Vs Number of VMs

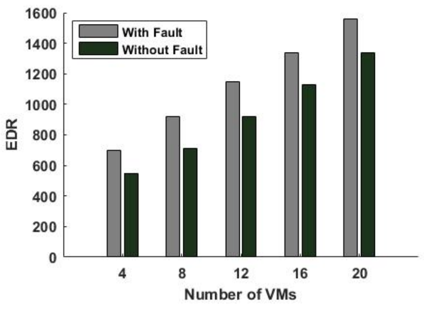

Figure 8: EDR Vs Number of VMs

6 Conclusions

Fault tolerance aims to ensure that the presence of faults does not lead to system failures, and mainly relies on prevention, detect and recovery policies. A cloud system can be made fault tolerant with efficient fault detection and recovery techniques. This paper presented a timeout based VM failure detector (RT-PUSH) for real-time applications in the cloud. Several experiments with different workload and VMs conducted to evaluate the performance of the RT-PUSH. The experimental results show the effect of VM fault on success ratio and execution drop rate. For future study, we will extend our model to detect VM fault in multiple PM and implement it in a real cloud environment. We will design and implement a fault tolerant scheduling algorithm to improve the execution time and success ratio.

References

[1] Sahoo, S., Nawaz, S., Mishra, S. K., and Sahoo, B. (2015, December). Execution of real time task on cloud environment. In India Conference (INDICON), 2015 Annual IEEE (pp. 1-5), IEEE, doi: 10.1109/INDICON.2015.7443778.

[2] Sahoo, S., Sahoo, B., Turuk, A. K., and Mishra, S. K. (2016). Real Time Task Execution in Cloud Using MapReduce Framework. Resource Management and Efficiency in Cloud Computing Environments, pp. 190-209, PA: IGI Global. DOI:10.4018/978-1-5225-1721-4.ch008.

[3] Latiff, M. S. A., Madni, S. H. H., and Abdullahi, M. (2016). Fault tolerance aware scheduling technique for cloud computing environment using dynamic clustering algorithm. Neural Computing and Applications, pp. 1-15, ISSN: 1433-3058, DOI: 10.1007/s00521-016-2448-8.

[4] Wang, J., Bao, W., Zhu, X., Yang, L. T., and Xiang, Y. (2015). FESTAL: fault-tolerant elastic scheduling algorithm for real-time tasks in virtualized clouds. IEEE Transactions On Computers, pp. 2545-2558, ISSN: 0018-9340, DOI: 10.1109/TC.2014.2366751.

[5] Cheraghlou, M. N., Khadem-Zadeh, A., and Haghparast, M. (2016). A survey of fault tolerance architecture in cloud computing. Journal of Network and Computer Applications, pp. 81-92, ISSN: 1084-8045, DOI: 10.1016/j.jnca.2015.10.004.

[6] Chan, H., and Chieu, T. (2012). An approach to high availability for cloud servers with snapshot mechanism. In Proceedings of the Industrial Track of the 13th ACM/IFIP/USENIX International Middleware Conference, pp. 6:1-6:6, ACM, ISBN: 978-1-4503-1613-2, DOI: 10.1145/2405146.2405152.

[7] Zhu, X., Wang, J., Guo, H., Zhu, D., Yang, L. T., and Liu, L. (2016). Fault-tolerant scheduling for real-time scientific workflows with elastic resource provisioning in virtualized clouds. IEEE Transactions on Parallel and Distributed Systems, pp. 3501-3517, ISSN: 1045-9219, DOI: 10.1109/TPDS.2016.2543731.

[8] An, K., Shekhar, S., Caglar, F., Gokhale, A., and Sastry, S. (2014). A cloud middleware for assuring performance and high availability of soft real-time applications. Journal of Systems Architecture, pp. 757-769, ISSN: 1383-7621, DOI: 10.1016/j.sysarc.2014.01.009.

[9] Endo, P. T., Rodrigues, M., Gonalves, G. E., Kelner, J., Sadok, D. H., and Curescu, C. (2016). High availability in clouds: systematic review and research challenges. Journal of Cloud Computing, pp. 16, ISSN: 2192-113X, DOI: 10.1186/s13677-016-0066-8.

[10] Hayashibara, N., Cherif, A., and Katayama, T. (2002). Failure detectors for large-scale distributed systems. In Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems, pp. 404-409, IEEE, ISSN: 1060-9857, DOI: 10.1109/RELDIS.2002.1180218.

[11] Jhawar, R., Piuri, V., and Santambrogio, M. (2013). Fault tolerance management in cloud computing: A system-level perspective. IEEE Systems Journal, pp. 288-297, ISSN: 1932-8184, DOI: 10.1109/JSYST.2012.2221934.

[12] Behl, J., Distler, T., Heisig, F., Kapitza, R., and Schunter, M. (2012, April). Providing fault-tolerant execution of web-service-based workflows within clouds. In Proceedings of the 2nd International Workshop on Cloud Computing Platforms, pp. 7:1-7:6, ACM, ISBN: 978-1-4503-1161-8, DOI: 10.1145/2168697.2168704.

[13] Malik, S., and Huet, F. (2011). Adaptive fault tolerance in real time cloud computing. In IEEE World Congress onServices (SERVICES), pp. 280-287, ISSN: 2378-3818, DOI: 10.1109/SERVICES.2011.108.

[14] Yi, S., Andrzejak, A., and Kondo, D. (2012). Monetary cost-aware checkpointing and migration on amazon cloud spot instances. IEEE Transactions on Services Computing, pp. 512-524, ISSN: 1939-1374, DOI: 10.1109/TSC.2011.44.