A Proposed Fault Tolerance Model for Cloud System Based on the Distributed Shared Memory

Mohamed A. Elsharkawey, Hosam E. Refaat
Dept. of Information System Faculty of Computers and Informatics
Suez Canal University, Egypt
melshrkawey1964@yahoo.com, hosam.refaat@ci.suez.edu.eg

Abstract

Nowadays, the cloud computing appears as a magic solution for a lot of companies and scientific problems to have extendable resources with a suitable budget. The increasing of the utilization of cloud computing in different fields creates challenge in the reliability and availability of the system. In this paper, a middle layer is proposed to increase the reliability. This layer places between the application layer and the virtualization layer in cloud system. It aims to handles the failure of the nodes. This layer monitors the system resources and periodically updates the status of all the virtual machine functionality. Based on these statuses, when the failure is occurred, the appropriate virtual machine will be selected to handle object failure. Moreover, the failure is handled in two cases, firstly, when the failure occurred in one of the running virtual machine, secondly, when the failure is occurred at the virtual machine management.

Keywords: *Cloud system, Distributed shared memory, Fault Tolerance, Parallel systems.*

1. Introduction

Nowadays, Cloud computing plays the role key to present an immense shared resources to the internet users. The Locating of computing resources offer an effective alternative to deploy applications with high scalability, on-demand services and cost-effective manner [1]. The infrastructure of Cloud computing is constructed by communicating large-scale virtualized data centers to be able to attain with the heavy load of user demands. In addition, parallel computation is introduced as management means to capitalize on the aids from these shared resources [2].

Parallel systems can be familiarized in different classifications. The well-known classification is based on the Memory-Access strategies. The Memory-Access includes three types of parallel systems. They are distributed memory, shared memory, and distributed shared memory. In distributed memory (DM) system, the processor of each node has its own private memory that has its isolated address space. The job task is divided into small tasks that are distributed among the nodes of the system (workstation, PC or VM). Each node performs its dedicated subtask. When a node needs data from another, it will send a request message. So, this system is also called "message passing" [3]. In addition, each node sends messages must know the receiver node address. The parallel applications based on message passing are not portable, because they depend on the system machine addresses. Moreover, the direct connection between system nodes complicates the application structure making maintenance and debug are very difficult.

In the Shared-Memory Systems (SMS), all nodes share a single physical memory represents a global memory of the system. So, it is known as "single address space abstraction". Sharing of a single physical memory by all nodes allow the system to assign a new process to each idle node. So, (SMS) offers some advantages over the distributed memory. It simplifies the system implementation, a load balance between nodes. In addition, it simplifies the implementation of the fault-tolerance by re-using the node data stored in the global shred memory even with the failure of that node. Furthermore, the greatest advantage of (SMS) is the portability of system applications. This mean the application design doesn't depend on the system machine addresses [5].

The third system is the “Distributed Shared Memory System” (DSMS). Its strategy is based on distributing the data or processes among the nodes in a single shared memory. DSMS combines the advantages of the preceding systems. It simplifies the sharing of data and processes by using standard operations that make the application level of parallel programming portable and more readable. So, in cloud computing, the DSMS is preferred than the DMS that can’t distinguish between the local and the remote VM that causes a disclaimer performance [5].

In intensive computation systems, the running of the distributed system causes a lot of system failure to be occurred. These failures may a reason of losing data which disturbing the availability and durability of the system used by the applications. Actually, there are a lot of failure cases that can’t be observed by the client and affecting the computing service. Some of these failures are network congestion, hardware failure, server overload, or worker (virtual machine) failure [6]. Hence, it is important to address users’ reliability and availability concerns to handle the failure dynamically and to have a high available system. Obviously, the user code cannot handle all failure cases. Because, he cannot knows all fault tolerance techniques. Moreover, it is difficult to combine in the same system structure both of the cloud computing applications and fault tolerant technique that has a high complex system behavior [7]. This problem can be solved by presenting an independent fault tolerance layer to handle the failure and to achieve the desired availability and reliability [8].

One of the most famous strategies that maintain the balance between the load of applications and the failure handling is the task clustering [9]. The task clustering combines small sized tasks into a comparatively large sized task which assigned to one of the system nodes. Once one of the small tasks exposes to the failure, it will be re-assigned to another node. The types of clustering are discussed in Section 2.

In this paper, a new middle layer is presented to increase the reliability and the availability. This layer based on DSM to give a generative communication to the system components. This system handles the failure in the case of VM or virtualization management level. The presented layer will be used by the applications of the system to interact with cloud system resources.

This paper is organized as follows; Section 2 gives a brief overview of cloud fault tolerance techniques, Section 3 discusses the basic concepts of DSM and the motivating scenario, Section 4 describes the proposed model and discusses the performance result. Finally, Section 5 gives an overview and introduces the future work.

2. Related Work

In spite of a great efforts exploited to integrate the fault tolerance techniques in the cloud systems, the majority of these techniques suffer from some shortcoming. For instance, lain Tchana et al. analyzes the implementation of fault tolerance techniques focusing on autonomic repair. He shows that in the most of current approaches, fault tolerance is exclusively handled by the customer which leads to partial or inefficient solutions. Moreover, he suggests a technique based on the integration of fault tolerance in all levels of the cloud. This technique is based on checkpoint strategy which can't give optimum failure handling [10]. Deng et. at. proposes a fault tolerant technique using a case study of matrix multiplication problem to discuss the reliability [11]. Also, Jhawaret. al. presents a fault tolerance technique based on the user choices for the reliability degrees, the availability and replicas type. But, this technique involves the users in complex system details [12]. Furthermore, synchronized server replication is another fault tolerance solution proposed by Zhao et. al. to solve the failure of the virtualization manager [13].

On the other hand, Arockiamet. al. inserts a fault tolerance technique between the application layer and the virtualization layer. However, this technique succeeded with the internal resources only. Namely, there are no borrowed resources from external cloud [14]. OpenNebula handles the core daemon failure of the VM level by storing all the information regarding infrastructure configuration and the state of the virtualized resources in persistent backend [15]. Also, "Windows Azure" presents a well-known cloud service with a fault tolerance for VM level. Its strategy is based on creating a replication for each VM. This strategy is limited by windows Azure application [16]. In addition, EC2 service is used by Amazon to provide reasonable compute capacity. This service handles VM failure using Simple Queue Service (SQS) and Amazon Machine Image (AMI) [1].

Another strategy known as the transparent check pointing is chosen by Slawinska et al. [17]. This strategy works at the user level based on the Distributed Multi-Threaded Check Pointing (DMTCP). Also, Weizhong et al. offered another multi-level fault-tolerant system using Multi-level Checkpoint/Restart for Cloud (CDMCR) [18]. The CDMCR system backups the complete state of applications periodically with a snapshot-based distributed check pointing protocol including file system. Unfortunately, the fault tolerance strategies, which are based on check pining, suffering from parallel thread dependability that causes the check pointing to enforce restarted from scratch.

3. The previous Scenarios

The most traditional scenario of fault tolerance used in the virtual machine failure is the Re-clustering and retry [19]. This scenario is based on customizing the task cluster size depending on the failure rate. It condones the system structure and resources. This scenario contains three major techniques. They are Dynamic Clustering (DC), Selective Re-clustering (SR) and Dynamic Re-clustering (DR). In the DC the size of the cluster will be shrinking if the failure rate is high. As shown in Figure 1, the cluster size is decreased from 4 to 2. The SR is based on selecting the failed tasks in a clustered job and encapsulates it into a new clustered job. SR is different to the DC in that DC technique retries all tasks of a failed job even though some of the tasks have succeeded. SR inherits the advantages of the SR and DC. It reduces the cluster size, like DC method, if the failure rate is too high. Moreover, it has ability to select the failed tasks like SR, while DC does not. Unfortunately, these techniques are not suitable for heterogeneous resources. Because the size of the cluster may cause overload in some VM while some other VM are idle.

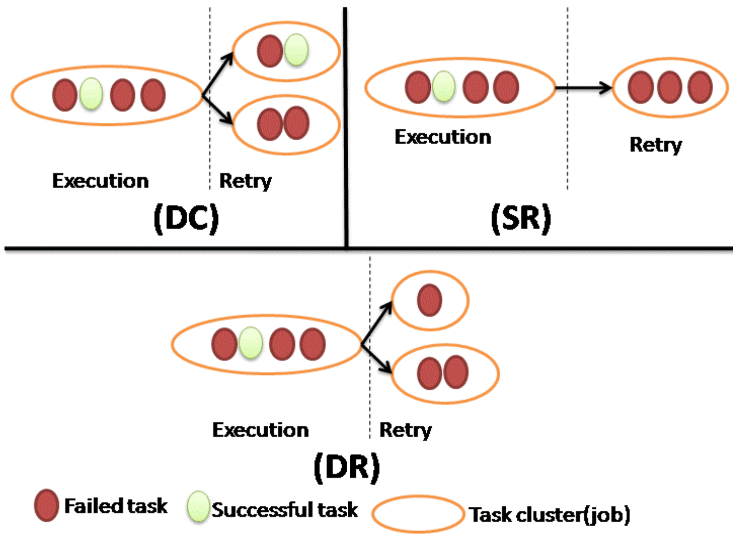

Figure1. Re-clustering and retry mechanisms

4. Proposed Model

All previous models are based on there-clustering and retry mechanisms of the tasks. These models are suffered from unbalanced loads problem between system nodes. So, the proposed model operation is performed based on three main steps. The first step, all job tasks are inserted into the DSM. In the second step, each task is taken from the DSM and assigned to one of the worker to process it. Finally, when the task operation is accomplished by worker, the operation result is returned to the DSM and another task is assigned to that worker. This strategy avoids the unbalanced load problem inseparable to re-clustering and retries mechanisms. On another word, the tasks are assigned to the workers one after another to ensure that no worker will be idle or overloaded. In addition, the capability of each worker that based on its resources is the ruling factor in finalizing its assign task.

This section is portioned into two main subsections. The first of them will introduce an overview for the basic components that constitute the proposed model and the basic operations performed by these components. In the second subsection, the operation scenario and the interaction among the layers of the system are investigated.

4.1 Model Overview

The introduced parallel processing model is based on the master-worker architecture. As shown in Figure 2, the proposed model is designed to contain multi of Data centers. Each data center has set off workers virtual machine (WVM); one of them works as Master Virtual Machine (MVM). In each data center, an intermediate layer known as Resources Manager is inserted between the virtualization layer and the application layer. This layer is acting on behalf as SaaS that identifies suitable resources and handles the failure. Each client application's is provided by Resources Manager inter face to deal with the intermediate Resources Manager layer. All WVM are controlled by the MVMs via the Resources Manager. So, The Resources Manager layer is employed for managing services and hiding the effect of the failure and recovery mechanism. In addition, the Resources Manager assigns the MVM to one of three DSM types. These types are Master Space, Replica Space or Passive Space. Only two of them are active type, the Master Space and the Replica Space while the third type of MVMs belonged to the passive type. When one of the two active types is failed, the still alive space of Master Space or Replica Space will be the Master Space and one of the Passive Space will be selected by Resources Manager to work as Replica Space. So, Resources Manager will make a copy of all Master Space contents to the new Replica Space to be used in covering any expected failure. The contents of Master Space as Replica Space should be include all meta-processing-data like the virtual machine status table (VMST), current processing tasks (CPT) and task objects (task Entry) that waiting to be assigned. In addition, any changes will be occurred in the Master Space must be redone in the Replica Space. In general, the Resources Manager of the master node receive the client job, divides it into small tasks and saved them in the DSM. Each task Entry will be assigned to WVM one after another to process it. This model is unlike task clustering strategies, which assigns multi-tasks to each node in each time causing unbalanced load between nodes. Using the DSM between the master node and the workers will increase the system flexibility. Additionally, the number of workers can be easily increased as the users need.

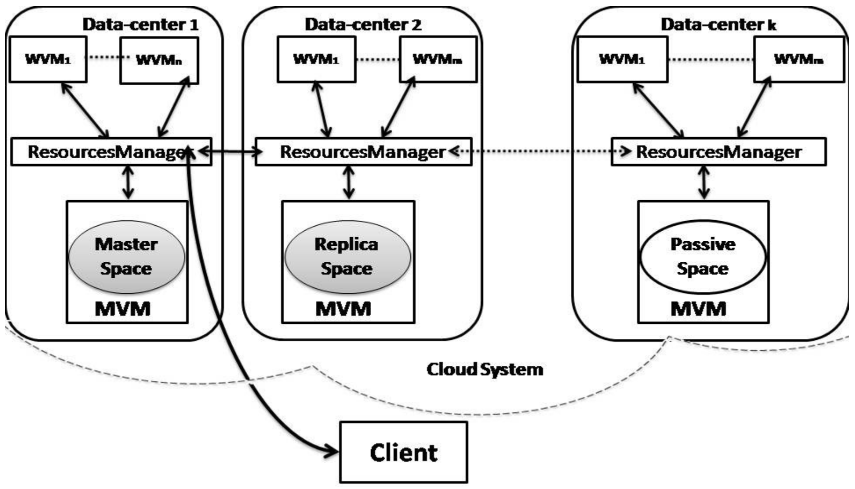

Figure2. The proposed model

4.2. Workflow Scenario

The workflow scenario represents the interaction among the services of the system layers and the client job in the proposed model is shown in Figure 3.In the first step, the client parses the job to the Resources Manager layer, which converts it into set of tasks. The role of the Space Fault Detection Recovery service (SFDR) is to periodically check the aliveness of the active spaces of the Master Space(MVM) and the Replica Space (RVM) founded in the DSMs. Based on the result of this checking and the followed updated status of SFDR, the Resources Manager layer will be informed by the location of the active space. So, it submits the tasks to the alive MVM and RVM, which represent the second step in the workflow scenario. In similarity the WVM checker periodically check the aliveness of the WVMs. So, in the third step the MVM will submit each WVM its assigned task. Additionally, the MVM and RVM keeps tracks of all WVM by register all its associated date such as current executing task Entry and previous taken task Entry in a vector known as WVM task allocator (WVM TaskAloc). This vector contains a history of all WVMs. Moreover, there is another vector for keeping track of the current status of the WVMs such as fail, idle, busy. This vector is known as “VM Status” and periodically updated by the WVM Checker service. The Resources Manager layer should have up-to-date copy from these two vectors. In case of active space failure, the responsibility of Resources Manager layer is to determine the new Replica Space and copying it with the Master Space contents.

In general, the user request is executed by passing in different sequences to ensure success of handling. However, each WVM will perform the transaction of encapsulated task Entry located in the DSM Master Space/ Replica Space. The result of transaction will be formed as “result Entry” and saved in the DSM. The MVM will collect the results of WVMs and the Resources Manager will be informed by these results to be used for updating the Replica Space. In case of transaction failure, the performed transaction will be cancelled, and the taken task Entry will be re-performed with a new transaction. I.e. task Entries will be return back to the Master Space and replica Space and accordingly this task Entry will be re-assigned to other WVM. After finishing the user task, the user receives its result Entry and the transaction is ended. Obviously, the failure in this model can occur in two cases. The first case is the DSM failure that can be handled by Resources Manager based on replication. The second case at the WVM which is handled based on the transaction technique.

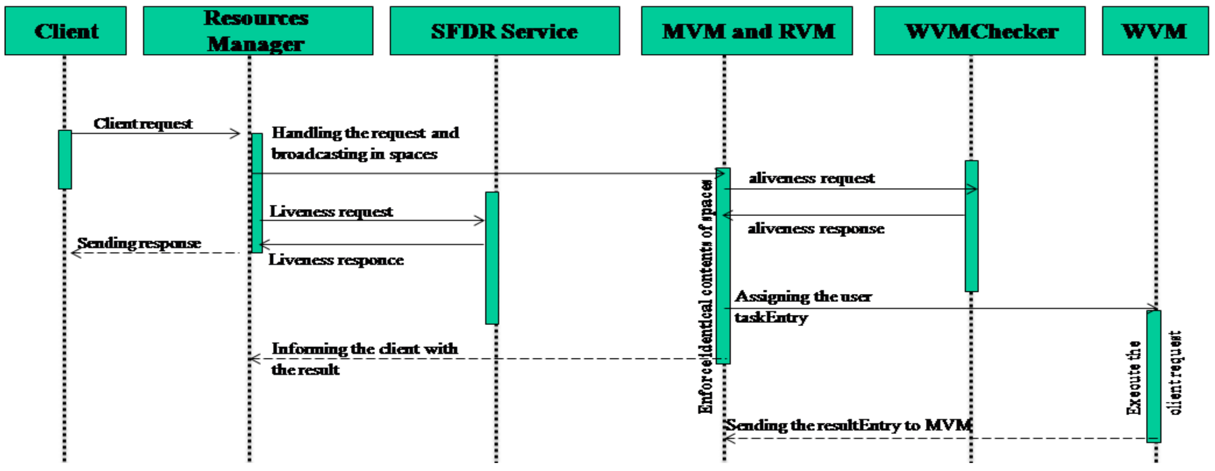
Client
Resources Manager
SFDR Service
MVM and RVM
WVMChecker
WVM
Client request
Handling the request and broadcasting in spaces
Liveness request
Liveness responce
Sending response
aliveness request
aliveness response
Assigning the user taskEntry
Informing the client with the result
Enforce identical contents of spaces
Execute the client request
Sending the resultEntry to MVM

Figure 3. Sequence diagram of workflow represents the interaction among system layers and the client request

5. Model Evolution

In order to measure the performance of our proposed model, its simulation is implemented on the WorkflowSim [20]. WorkflowSim is an open source workflow simulator that extends CloudSim [21]. CloudSim is run on the Java Net Beans interface. It contains a large number of java classes to simulate cloud components like Cloudlet, Scheduler, Data Center, Hosts, etc. By using WorkflowSim tool, the proposed Resources Manager for managing services and fault-tolerant is simulated. In this simulation, homogenous VM is used to discover the effect of the other parameters such as failure rate, number of VM, and number of data-centers on the system performance. The characteristics of the used resources are defined as follows; each virtual machine (VM) has 1GH processor and 512MB of memory. The default network bandwidth is 15MB, which is the maximum allowed data transfer speed between any pair of virtual machines. These assumptions are coherent with the setting of many real cloud experiments [22]. The number of VM and datacenters is depending on the experiment. The proposed method is compared with the most implemented clustering fault tolerant methods; they are DC, DR, and SR methods.

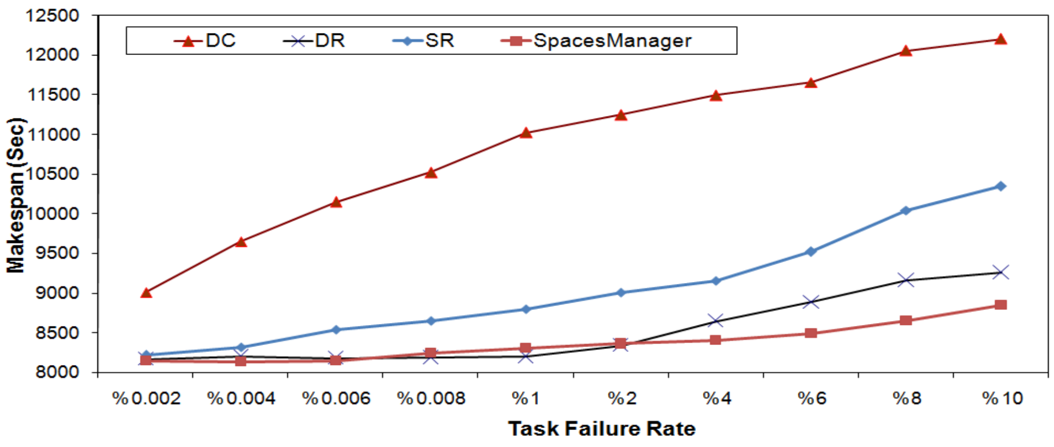

Figure.4. Failure rate effect in runtime performance

The first test, measure the effect the failure rate in the job make span. In this test, the cloud system is supposed to contain 50 VM distributed on 4 data center and the total number of tasks is 1000. This test measures the VM failure effect on the performance. As shown in the Figure 4, DC has the worst performance. Because the job is started from scratch if failure occurred. The performance curves of DR and ResourcesManager are closed at low failure rate. This is due to the low failure rate and the small number of failed tasks. Generally, the reassigning of small numbers of tasks is not affected. But, in high failure rate the reclustering the failed tasks will affect the performance.

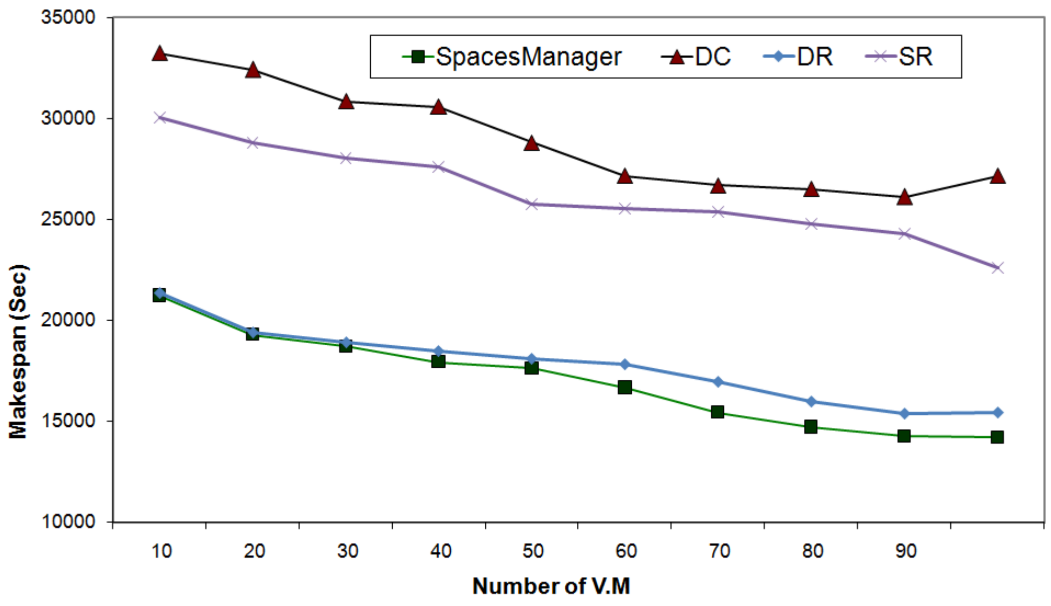

Figure.5. Performance Measurement Using Different Number of VMs

The second test measures the effect of increasing the number of VMs in the system performance. This test is performed using 4 datacenter and the failure rate 1%. As shown in Figure 5, when the numbers of VMs are small the performance of DR and Resources Manager models are closed. While, the increasing number of VMs will maximize the performance of the Resources Manager system curve with respect to the other curves.

6. Conclusion and Future work

A fault tolerance model for the cloud systems is presented. The proposed system combines two fault tolerance strategies to handle the different possible failures. These strategies include the replication of DSM and transaction for VM failure. The replication of the DSM is managed by the Resources Managers while the failure of the worker is handled by the transaction strategy. The Resources Managers is used as a layer for the interaction between the applications of the clients and the cloud resources. It gives the parallel systems the portability and the flexibility. The simulation results evidence the performance enhancement of the proposed model compared to the re-clustering and re-try models. The future work will mainly be driven towards the implementation of the proposed model on fog computing system.

References

1. Amazon Elastic Compute Cloud [Online]. Available: http://aws.amazon.com/ec2/
2. M. Armbrust, A.Fox, R. Griffit,et al. A view of cloud computing. Communications of the ACM, vol. 53, (2010) 50–58.
3. D. Fiedler, K. Walcott and T. Richardson, M. Gregory, A. Amer and Panos K. Chrysanthis. Towards the Measurement of Tuple Space Performance. ACM SIGMETRICS Performance Evaluation Review, December, 2005, 33-42.
4. M. Tomasevic, J. Protic, and V. Milutinovic. Distributed shared memory: Concepts and systems. IEEE Parallel and Distributed technology, vol. 4 (1996) 63– 79.
5. A. Anbar, V. K. Narayana, and T. El-Ghazawi. Distributed Shared Memory Programming in the Cloud. In Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012) (CCGRID '12). (2012) 707-708.
6. K. V. Vishwanath and N. Nagappan. Characterizing cloud computing hardware reliability. In Proc. 1st ACM Symp. Cloud Comput. (2010) 193–204.
7. R. Jhawar and P. Vincenzo. Fault Tolerance Management in IaaS Clouds. Satellite Telecommunications (ESTEL), 2012 IEEE First AESS European Conference. (2012) 1 –6.
8. D. Gelernter. Getting the job done (linda, parallel programming language). j-BYTE , Vol. 12. (1988) 301–308,
9. Y. M. Essa. Survey of Cloud Computing Fault Tolerance: Techniques and Implementation. International Journal of Computer Applications Vol. 138. (2016) 0975 – 8887
10. T. Alain et. al. Fault Tolerant Approaches in Cloud Computing Infrastructures. The Eight International Conference on Autonomic and Autonomous Systems. (2012) 42 – 48.
11. J. Deng, Scott C.-H. Huang, Yunghsiang S. Han, and Julia H. Deng. Fault-Tolerant and Reliable Computation in Cloud Computing. GLOBECOM Workshops (GC Wkshps), 2010 IEEE , (2010) 1601–1605.
12. R. Jhawar, P. Vincenzo ,and M. Santambrogio. Fault Tolerance Management in Cloud Computing: A System-Level Perspective. Systems Journal, IEEE. Vol.7. (2013) 288 – 297.
13. Z. Webbing et. al. Fault Tolerance Middleware for cloud computing. Third International Conference on Cloud Computing. vol. 4, (2010) 67– 74.
14. L. Arockiam and Geo Francis E. FTM-A Middle Layer Architecture for Fault Tolerance in Cloud Computing. Special Issue of International Journal of Computer Applications, vol. ICNICT, (2012) 0975 – 8887.
15. Opennebula [Online]. Available: http://opennebula.org/documentation:archives:rel2.2:ftguide.
16. Windows Azure [online]. http://www.davidchappell.com/writing/white_papers/ introducing _windows_azure_v1-chappell.pdf
17. Eucalyptus Systems [Online]. Available: http://www.eucalyptus.com/

18. Q. Weizhong, J. Changqing, R. Longbo, Z. Deqing, and J. Hai. CDMCR: multi-level fault-tolerant system for distributed applications in cloud. Security Comm. Networks, doi: 10.1002/sec.1187. (2015) 2089– 2125.

19. Weiwei Chen, Rafael Ferreira da Silva, EwaDeelman, and Thomas Fahringer. 2016. Dynamic and Fault-Tolerant Clustering for Scientific Workflows. *IEEE Trans. Cloud Comput.* Vol. 4,(January 2016)

20. W. Chen and E. Deelman, "Workflowsim: A toolkit for simulating scientific workflows in distributed environments," in 2012 IEEE 8th International Conference on E-Science, ser. eScience, 2012, pp. 1–8. [Online]. Available:https://github.com/WorkflowSim

21. R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," Software: Practice and Experience, vol. 41, no. 1, 2011.

22. Weiwei Chena, Rafael Ferreira da Silva a, EwaDeelmana, RizosSakellariou, "Using imbalance metrics to optimize task clustering in scientific workflow executions", 2015, Future Generation Computer Systems.