Fault Tolerant Approaches in Cloud Computing Infrastructures

Alain Tchana, Laurent Broto, Daniel Hagimont
Institut de Recherche en Informatique de Toulouse (IRIT)
Toulouse, France
Email: alain.tchana@enseeiht.fr, laurent.broto@enseeiht.fr, daniel.hagimont@enseeiht.fr

***Abstract*—Based on the pay-as-you-go strategy, cloud computing platforms are spreading very rapidly. One of the main characteristics of cloud computing is the splitting into many layers. From a technical point of view, most cloud computing platforms exploit virtualization, which implies that they are split into 3 layers: hosts, virtual machines and applications. From an administration point of view, they are split into 2 layers: the cloud provider who manages the hosting center and the customer who manages his application in the cloud. This structuring of cloud makes it difficult to implement effective management policies. This paper focuses on fault tolerance in cloud computing platforms and more precisely on autonomic repair in case of faults. It discusses the implications of this splitting in the implementation of fault tolerance. In most of current approaches, fault tolerance is exclusively handled by the provider or the customer, which leads to partial or inefficient solutions. Solutions, which involve a collaboration between the provider and the customer are much promising. We illustrate this discussion with experiments where exclusive and collaborative fault tolerance solutions are implemented in an autonomic cloud infrastructure that we prototyped.**

***Keywords-Cloud Computing, Fault tolerance, Virtualisation*.**

I. INTRODUCTION

Due to the difficulty to maintain an internal infrastructure technology and the associated rising costs, companies are increasingly externalizing their IT services, which are therefore managed by specialized companies (called providers). This trend led to the emergence of the so-called cloud computing approach. One of the most important objective of cloud computing is to allow customers to pay only for the amount of resources they effectively consume. This option, summarized by the term pay-as-you-go, is permitted in cloud platforms through the partitioning of their resources.

Virtualization techniques are commonly used in cloud platforms to implement partitioning of resources. Instead of having direct access to cloud resources, customers have access to virtual machines, which represent a fraction of a physical machine. Then, we identify three layers in such a cloud infrastructure: the physical resource layer (containing the overall cloud resources), the virtualization layer (containing virtual machines) and the applications layer (containing applications of external companies, which are hosted in the cloud).

From an administration point of view, we consider two main roles, which correspond to the administration of the hosting infrastructure (the provider) and the administration of the application deployed in the cloud (the customer).

These multiple layers and roles make difficult the management of cloud platforms and particularly the management of failures in these infrastructures. Indeed, handling failures become more complex because those who intervene in the cloud (customers and provider) have different views (and access rights) of the different layers of the cloud. Costumers are limited to only detecting faults of virtual machines and their applications, while the provider can only manage real resources (physical machines) and virtual machines faults. Therefore, possible Fault Tolerance (FT) solutions vary according to the involved participants and according to the implementation level.

Although current cloud platforms take in account many challenges, their implementation usually propose no fault tolerance solution ([1], [2]) or basic FT solutions ([3]). For those who implement FT services ([4], [5], [6], [7]), we retain that their solutions only entrust the responsibility of fault management either to the customer or to the provider. No collaboration between the two types of participants is considered.

The purpose of this paper is to investigate FT policies in cloud platforms. We identify two types of policies: one, which is exclusively handled by one participant (customer or provider) and another, which is a collaborative management between the provider and customers. This second type constitutes an interesting tradeoff between exclusive management by the provider and exclusive management by the customer. This discussion is illustrated by experiments and evaluations with an operational prototype of autonomic cloud platform.

The rest of the paper is organized as follows. Section II introduces the cloud computing technology (concepts and architecture) and its challenges (including fault management). Section III covers fault detection and management techniques in the cloud. Section IV discusses related work. Section V presents experiments and evaluations, which illustrate our reflection. Section VI concludes and outlines areas for future works.

II. CLOUD COMPUTING OVERVIEW

Due to the lack of consensus on the definition of cloud computing, let us refer to the CISCO [8] one: "*IT resources and services that are abstracted from the underlying infras-*

 ISBN: 978-1-61208-187-8

tructure and provided on-demand and at scale in a multitenant environment". Therefore, cloud computing consists in: (1) providing on demand services to external customers with the illusion of infinite resource, (2) and then using the same resource pool for all customers. This strategy offers several advantages including:

- Reduced costs for the customer. He no longer needs to manage his own infrastructure and is billed according to the use of cloud services.
- Flexibility for the customer. He can increase the capacity of his infrastructure without major investments, resources of the cloud are dynamically allocated on demand.
- Less waste. Internal IT systems managed by customers are often under-utilized while the cloud infrastructure is shared between many customers, which increases the average resource utilization rate. A important example of this waste is the energy consumption of such infrastructures.

Several models of cloud are presented in the literature, including: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). In this article, we consider a cloud as an IaaS: a virtualized infrastructure managed by a provider, in which external customers deploy and execute their applications. Then, the parties involved in a cloud platform are grouped into three categories: cloud providers, cloud customers and end users. A cloud provider is responsible for the administration of the cloud resources (hardware and virtual machines(VM)) and services. He is responsible for managing the accommodation of the capacity of the cloud. Cloud customers use the resources provided by the cloud to deploy and execute their applications. They do not have a global view and direct access to the cloud environment. They use cloud resources through VMs, which host their applications and, which represent a confined portion of physical resource. End users are using customer applications deployed in the cloud. Figure 1 summarizes and shows the vertical cloud architecture and the scope of each cloud participant: the cloud provider has access to physical resources and VMs; cloud customers have access to VMs and their applications; and end users have access to customer applications.

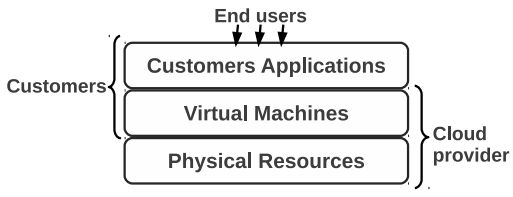

Figure 1. Cloud Computing architecture

III. Fault Tolerance Techniques in Cloud Platforms

As shown in Figure 1, three layers are identifiable in a cloud platform: resources, VMs and applications. Each of them is concerned with failures. Therefore, we identify three types of failure in a cloud platform: hardware failure, VM failure and application failure. A FT strategy includes two distinct phases: detection phase and repair phase. One of the difficulties to implement FT in a cloud architecture can be summarize by this question: which cloud participant (provider or customer) is the best able to implement the two phases of fault management, depending on its access rights in the cloud architecture? In other words, is it reasonable to leave exclusively the responsibility of FT to one cloud participant knowing that: hardware failures can only be detected and repaired by the cloud provider; VM failures can be detected by the two participants but only repaired by the cloud provider; and application failures can only be detected by the customer but can be repaired by the two participants.

We present in this section two visions of FT management in a cloud platform. The first one consists in giving both the detection and repair responsibilities to one cloud participant (exclusively) while the second is to harness the skills of the two types of participant. According to these two visions, we present some FT techniques for the three types of failure we have identified: hardware, VM and application.

A. Exclusive FT Management

We discus about exclusive FT solutions in this section.

1) Application FT: As mentioned above, application failures are detectable only at the customer level. The failure detection policy depends on the application. However, the mechanism used to implement a detection policy is generally the same. For each application, the customer deploys in the cloud special software components called sensors, which monitor the liveness of the application. According to this monitoring, a sensor may trigger the execution of a procedure for repairing the application when it is considered as malfunctioning.

Two methods are possible for repairing a faulting application. The first one concerns stateless application (such as loadbalancers, e.g., HAProxy or MySQL-Proxy used in our experiments). Repair consists in restarting the faulting server on the same VM. The second method concerns statefull servers (e.g., the MySQL database). In this case, the customer must implement a mechanism for saving the server state so that it can be restored before the server is restarted. In the case of a database server for example, this save/restore mechanism can be implemented by trapping and storing all modification requests, allowing replay of these requests.

The advantage of this solution is that the customer has full control over the solution. Then, the application can be hosted in any cloud platform without assuming that the

 ISBN: 978-1-61208-187-8

platform implements any FT solution or that the platform is dedicated to a particular kind of application. However, this solution requires that customers have expertise on their applications so that they can implement application state save/restore procedures notably, and they must implement their own monitoring system.

2) VM Fault Tolerance: VM failures can be detected and repaired by the two cloud participants. We consider here repair policies that are implemented exclusively by one of them.

Customers implement VM FT by deploying sensors in the cloud, which monitor VM state during their lifetime. In this case, it is not recommended to give to a single sensor the responsibility of probing one VM because a failure of the VM hosting this sensor would compromise the repair mechanism. The repair of the failed VM is organized as follows: (1) the customer level requests the cloud to free the failed VM; (2) it allocates a new VM; (3) it deploys and starts the servers that were running on the failed VM; and (4) it restores the state of these servers in case of statefull servers. One disadvantage of this solution is that each customer must implement his own VM monitoring system, which leads to complexity and network resource wasting, while VM monitoring can be factorized and implemented by the cloud provider.

At the cloud provider level, an exclusive VM FT technique can be implemented. With a direct access to VM hypervisors, the cloud provider is more likely to implement VM FT. Firstly, such an implementation decreases the number of VM sensors (and their associated communication) as they are integrated in hypervisors. Actually, a single sensor per physical machine can monitor all the VMs hosted on this machine. Secondly, through the hypervisor, the provider can collect more detailed information about VM status. This information allows him to implement more accurate FT solutions according to VM status. In contrast, when a VM failure is detected by the customer, it is more precisely the VM inaccessibility, which is detected since the customer is not supposed to be granted access to hypervisor information. At the cloud level, the provider is able to identify several types of VM failure. For example, considering the Xen hypervisor, a VM can be broken when its state is blocked, pending, or error. The provider can then implement VM repair so that an appropriate repair method is applied for each state of failure. Also, the analysis of Xen logs allows releasing a VM (instead of reallocating a new VM as the customer level would do) whose state is blocked waiting for a device. A more agnostic solution for VM FT is to regularly store VM states using the checkpointing ability offered by virtualization systems. Thus, the cloud will just restart a failed VM from its last saved state.

The advantage of provider level solutions resides in the fact that it factorizes VM FT tasks that would be implemented by the customers. However, being not aware of the particularities of VM hosted applications, the provider repair solution may not be efficient in certain cases. For example, repairing a VM, which hosts MPI processes (used by other VMs) cannot be done by restoring the VM to its last state. MPI applications do not support rollbacks, even negligible. In this case, a collaboration between the provider and the customer can be the solution.

3) Physical Machine Fault Tolerance: Hardware failures are more difficult to take into account in the cloud because they trigger failures at the other levels (VM and application). We consider repair implemented exclusively at the customer or provider level.

At the customer level, it is impossible to detect a physical machine failure. At this level, a physical machine failure is perceived as multiple VM failures (the VMs running on the failing machine). The customer level only sees VMs and even if VM monitoring sensors are deployed in the cloud, they may not be aware of a hardware failure if they are all deployed on the same physical machine (sensors deployed on VMs hosted by the faulting machine). The customer would need to implement constraints regarding the physical location of sensors, which is a form of collaboration between the customer and the cloud provider (we will come back to this collaboration in the next section).

At cloud provider level, hardware FT is implemented with a monitoring system composed of sensors deployed on different physical machines. For repair, the provider will start on a new machine (or several machines according to its capacities) the same number of VMs, which were hosted on the failed machine. In addition, all VM states must be saved by checkpointing so that VM restoration is possible. This solution is used in [5] and [9] for example.

B. Collaborative Fault Tolerance Management

Exclusive FT techniques presented in the previous sections highlight difficulties regarding techniques applied for certain types of failure. As will be discussed in this section, these difficulties can be taken into account if a collaborative management between the cloud provider and customers is considered.

1) Application Fault Tolerance: In Section III-A1 we have described an application fault detection without specifying the impact on other levels of failure (VM and hardware). If the sensor is deployed on a separate VM than the application, then a detected failure can have three origins: application (already treated in Section III-A1), VM or hardware. In the case of a VM failure, the collaboration between the application sensor and the VM sensor (possibly at the customer level) will give more clarification to the application sensor. If the VM sensor does not detect any error, then it can conclude that it is an application failure. Otherwise, it can deduce that it is a VM or a hardware failure. In this latter case, the customer alone has no way to know much. A better collaboration with the cloud level

 ISBN: 978-1-61208-187-8

would provide confirmation of the failure type.

Concerning the repair of a failed application, a collaborative repair may be considered. Indeed, the cloud can offer regular backups of VM states so that a customer can subscribe to and invoke the cloud level for the restoration of a VM from a saved image.

2) VM Fault Tolerance: If the VM fault detection is done by the customer, we will have the same problem as described in the previous section. At the customer level, a VM failure detected by a sensor can be due to a hardware failure (the machine, which hosts the VM), which is out of scope for the customer. A VM fault detection at the cloud level allows getting a more accurate decision.

If the VM fault detection is implemented at the cloud level, a collaboration with the customer level can provide a better solution than the checkpoint-based one described in section III-A2. Concretely, once the fault is detected by the cloud, it starts a new VM with the same features (networking, memory, CPU, image) as the failed VM and then calls the customer to redeploy, restart and synchronize the new VM. This solution would probably perform better than the checkpointing one regarding the cost of save/restore operations.

3) Physical Machine Fault Tolerance: From the point of view of the customer, the failure of a physical machine is identical to the VM crashes. When detected by the cloud provider, such a failure can be resolved collaboratively during the repair of each VM hosted on the failed machine. Each VM is restarted with the same characteristics on a new physical machine and the completion of the repair is asked to the customer (as seen in the previous section).

C. Synthesis

We have discussed possibilities for implementing FT (repair) in a cloud environment. In a cloud environment composed of a hosting center managed by a provider and applications deployed in this hosting center and managed by customers, we distinguish:

- FT managed exclusively at the customer level. At this level, it is possible to detect application and VM faults, but detecting hardware faults is difficult and it is not possible to have details about detected VM fault (this information is only available at the hypervisor level). Repair can be implemented at this level by restarting VMs (VM and hardware faults) and redeploying and restoring applications (all faults). The main drawback is that each customer must implement the whole FT policy.
- FT managed exclusively at the provider level. At this level, application faults cannot be detected, but detailed information can be given for VM and hardware faults. Repair can be based on VM checkpointing, which is independent from applications, but can be quite costly.
- FT managed collaboratively at both levels. Applications faults must be detected and repaired at the customer level. VM and hardware faults can be detected at the provider level (with details). The repair of the VMs (upon VM and hardware faults) can then be accurately performed at the provider level, and finally the recovery of the application that were running on these VMs can be requested and performed at the customer level. The recovery (redeployment and restoration) of applications on VMs can also have a significant cost.

Naturally, collaborative techniques appear to be the best suited. However, most of the proposed solutions are exclusively implemented in one level, as it is strategically difficult to split an FT policy between two participants.

IV. Related work

As we mentioned in the introduction, few works addressed the issues of FT in cloud environments. Some platforms such as Eucalyptus [1] or CLEVER [6] provide no solution to take into account hardware, VM or customer application failures. CLEVER addresses FT management, but only for its own components.

OpenNebula [3] offers exclusive VM FT implemented at the cloud level. It allows the cloud provider to associate hooks (scripts or programs) with each type of VM failure (according to hypervisor information). Hardware failures are not addressed by OpenNebula for two reasons: it provides no hardware sensors and all VM sensors are located on the same machine than the VM. So a machine failure cannot be detected by OpenNebula.

As OpenNebula, the Microsoft Windows Azure platform [5] offers an exclusive FT management at the cloud level. Windows Azure replicates each VM so that a VM failure is covered by its replicas. The Azure solution is limited to web applications developed in the Windows Azure platform. Moreover, no solution is proposed to repair the failed VM. In addition, for VMs which are not instantiated by the Azure development platform, the entire responsibility of FT management is left to the customer. This is also the case in the Amazon EC2 [7] cloud platform.

Kaushal [4] proposes an FT solution in the cloud at the customer level by replicating servers queries, based on the HA-Proxy load distributor. Other researches such as [10] and [11] propose a collaborative solutions for specifics applications (MPI for example). However, they do not consider the splitting of the cloud between a (VM) provider and its customers, so their works are only applicable to an SaaS cloud.

Uesheng Tan [12] describes a replication solution for VM FT, exclusively managed by the cloud provider. It proposes to improve efficiency by using passive VM replicas (with very few resources), which become active when a failure is detected. A mechanism is introduced to transfer/initialize the state of VM. This solution is similar to the exclusive FT

 ISBN: 978-1-61208-187-8

solution (at the provider level) that we presented previously and that we evaluate in the next section.

Finally, Wenbing Zhao [13] proposes a FT middleware, which can be used by a cloud customer to implement software FT. Their main purpose is to implement a synchronized server replication strategy so that a failed server can be repaired with a consistent state.

V. EVALUATIONS

In these experiments, we evaluate the two visions that we presented above: exclusive and collaborative FT strategies. Due to the limited space in this paper, we only present our evaluation with VM FT. In these evaluations, we detect a VM failure when the VM does not respond to a *ping* request or when the hypervisor indicates an error state.

The evaluation environment is composed as follows. For the cloud platform, we use a prototype called CloudEngine, based on an adaptable autonomic management system called TUNeEngine, which were both implemented in our research group. Briefly, CloudEngine is similar to the OpenNebula system, but it is more adaptable and flexible because it is based on an adaptable system (TUNeEngine). For example, it allows easy addition of new functionalities, management policies and offers collaborative API. This prototype allows, at the cloud level, the deployment of VMs and the definition of VM level reconfiguration policies (repair in our case), and at the customer level, the deployment of application servers and their reconfiguration/repair.

Our prototype is used to allocate VM in the cloud, deploy and start a J2EE application (RUBIS [14]) as our customer application. Our J2EE experimental application is composed of an Apache web server, two Tomcat servlet containers, a MySQL-Proxy database loadbalancer and two MySQL database servers. The MySQL stage is our replicated layer in which failures will be triggered. For the evaluation of our two FT techniques, we apply a pyramidal RUBIS workload (upload phase, constant load phase and download phase) in which we simulate a VM failure during the constant load phase. The objective of the experiment can be summarized by this question: what does the FT technique cost in term of RUBIS performance (throughput). To answer this question, we ran the same workload without failure in order to have a reference execution with which the others can be compared. We observe the request throughput during the benchmark and the number of untreated requests during repair.

A. Exclusive Fault Tolerance

The exclusive FT technique we evaluate in this experiment is implemented in the cloud level. The implemented FT policy starts a checkpointing program on each IaaS node, which role is to save the status of each VM every 7 seconds. The choice of the backup frequency should not be too small nor too large for two reasons: the risk of penalizing the performance of the VM (as discussed) and the risk of having a large gap between VM status after and before the failure. Upon failure, the last checkpoint is used to restore a recent image of the failed VM.

Figure 2 shows the comparison between the experimental landmark (red curve) and the experiment with the checkpointing (green curve) in which no failure was simulated. These first curves allow us to observe the overhead of VM checkpointing. We estimate this overhead is about 46% due to the Xen VM checkpointing implementation. Actually, the checkpointing implemented by Xen causes the unavailability of the VM during the checkpoint, which explains this overhead. Notice however that this unavailability does not cause request loss since the TCP/IP protocol (on which our network is based) retransmits a request when communication fails. Thus, during checkpointing, the downtime of the VM does not break TCP/IP communications and it only postpones request handling. It explains the large fluctuations in the green curve.

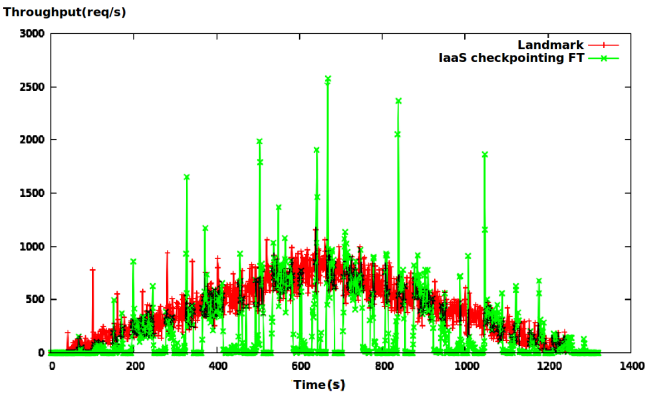

Figure 2. Checkpointing cost during VM repair in the IaaS

Figure 3 shows the result of applying our FT method (at the cloud level) with fault simulation on a MySQL server VM. T_p represents the failure date while T_r marks the end of the repair. The repair time in this experiment is about 22 seconds. This time includes the time taken to detect the failure and also the duration of VM restarting (from its last saved state).

Notice here that requests are lost when we recover from the last saved checkpoint. This can be tolerated for a web application but it would not for more sensitive applications such as MPI applications.

B. Collaborative Fault Tolerance

The second FT technique we evaluated is a collaborative one in which VM fault detection and repair operations are performed (collaboratively) by CloudEngine (the IaaS) and the customer (via our TUNeEngine autonomic administration system at the application level). CloudEngine detects VM failure, restarts the VM from its original image and finally invokes the customer level TUNeEngine system to

 ISBN: 978-1-61208-187-8

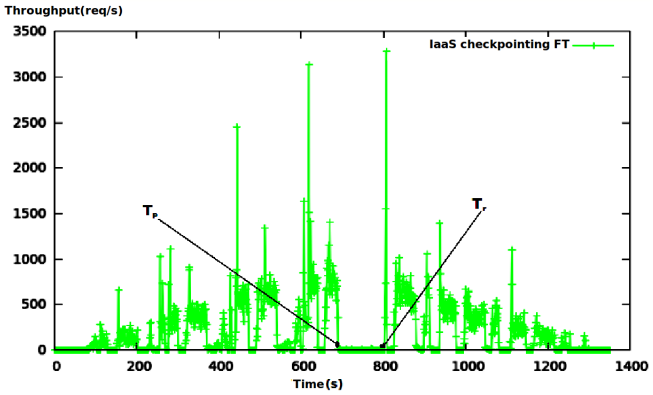

Figure 3. Exclusive VM Fault Tolerance Technique

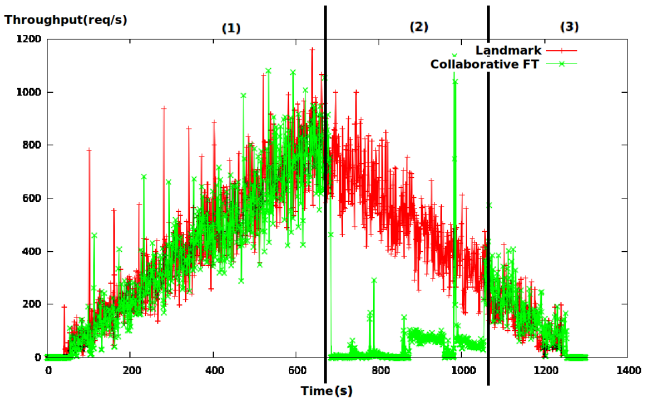

Figure 4. Collaborative VM Fault Tolerance Technique

complete the repair. Then, TUNeEngine deploys on the restarted VM the application that was running on it before the failure (MySQL in this experiment), configures and starts the MySQL application. The size of the deployed application and the reconfiguration and start operations influence the overall duration of the repair.

Figure 4 shows two curves: the red curve represents the reference execution (without failure nor FT management) while the green curve represents the result of our collaborative FT technique. We observe that this repair method, unlike the previous one, has no impact on RUBIS performance when no failure is involved. This is shown in Figure 4 areas (1) and (3). Zone (2) represents the duration of the repair. It includes: the failure detection (at the cloud level), the VM deployment and restart, the MySQL server binaries copy and restart. For these reasons, we measured in this experiment a repair time of 5 minutes 30 seconds. Notice that the use of a mirror server, which is usually the case for such applications, would keep the service available during the repair of the failed server.

Even if the repair time is much higher with this solution, it seems best suited as it does not incur any overhead on execution.

VI. Conclusion and Future Works

In this paper, we studied two visions for FT management in cloud computing platforms: the first one consists in leaving exclusively the responsibility of FT management to one cloud participant (cloud customer or provider); the second one consists in sharing the responsibility between the two cloud participants. We reviewed all possible fault situations in the cloud: application level, virtualization level and hardware level. We proposed for each of them some solutions involving exclusive or collaborative FT visions. Given the limited space in this article, we only evaluated two FT techniques (one exclusive and one collaborative). However, this evaluation illustrates that sharing FT management between the two cloud participants opens interesting perspectives.

In the near future, we intend to complete our experiments with all the techniques described in this paper. Furthermore, VM FT techniques based on checkpointing can be improved by new VM checkpointing solutions, which consist in storing only the difference between successive VM states (rather than the entire VM state).

References

[1] Daniel Nurmi, Richard Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov "The eucalyptus open-source cloud-computing system," in *9th International Symposium on Cluster Computing and the Grid (CCGRID), vol. 0, pp. 124-131. Washington, DC, USA*, 2009.

[2] University. of Chicago, "Nimbus is cloud computing for science," http://www.nimbusproject.org/, [retrieved: january, 2012].

[3] OpenNebula, "Opennebula.org: The open source toolkit for cloud computing," http://opennebula.org, [retrieved: january, 2012].

[4] Vishonika Kaushal and Vishonika Kaushal, "Autonomic fault tolerance using haproxy in cloud environment," *International Journal of Advanced Engeneering Sciences and Technologies*, vol. 7, 2010.

[5] Microsoft, "Windows azure: Microsoft's cloud services platform," http://www.microsoft.com/windowsazure/, [retrieved: january, 2012].

[6] Francesco Tusa, Maurizio Paone, Massimo Villari, and Antonio Puliafito, "Clever: A cloud-enabled virtual environment," in *IEEE Symposium on Computers and Communications (ISCC), pp. 477-482. Riccione, Italy*, 2010.

[7] Amazon, Inc, *Amazon Elastic Compute Cloud (Amazon EC2)*. Available: http://aws.amazon.com/ec2/#pricing, [retrieved: january, 2012].

 ISBN: 978-1-61208-187-8

[8] Kapil Bakshi, “Cisco cloud computing - data center strategy, architecture, and solutions point of view white paper for u.s. public sector 1st edition,” 2009, http://www.cisco.com/web/strategy/docs/gov/CiscoCloudComputing_WP.pdf, [retrieved: january, 2012].

[9] Walters John Paul and Chaudhary Vipin, “A fault-tolerant strategy for virtualized hpc clusters,” *The Journal of Supercomputing*, vol. 50, 2009.

[10] Qin Zheng, “Improving mapreduce fault tolerance in the cloud,” in *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), pp. 1-6*, 2010.

[11] Magdalena Slawinska, Jaroslaw Slawinski, and Vaidy Sunderam, “Unibus: Aspects of heterogeneity and fault tolerance in cloud computing,” in *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), pp. 1-10*, 2010.

[12] Uesheng Tan, Dengliang Luo, and Jingyu Wang, “Cc-vit: Virtualization intrusion tolerance based on cloud computing,” in *2nd International Conference on Information Engineering and Computer Science (ICIECS), pp. 1-6. Wuhan, China*, 2010.

[13] Wenbing Zhao, Michael Melliar-Smith, and Louise E. Moser, “Fault tolerance middleware for cloud computing,” in *3rd International Conference on Cloud Computing (CLOUD 2010), pp. 67-74. Miami, FL, USA*, 2010.

[14] Cristiana Amza, Emmanuel Cecchet, Anupam Chanda, Alan L. Cox, S Elnikety, Romer Gil, Julie Marguerite, Karthick Rajamani, and Willy Zwaenepoel, “Specification and implementation of dynamic web site benchmarks,” in *5th Annual Workshop on Workload Characterization (WWC-5), pp. 3-13. Austin, Texas, USA*, 2002.

 ISBN: 978-1-61208-187-8