Towards RSA-Based High Availability Configuration in Cloud

Yihan Wu[12], Ying Zhang[12], Yingfei Xiong[12], Xiaodong Zhang[12], Gang Huang[12]

[1]Key Lab of High Confidence Software Technologies (Ministry of Education)
[2]School of Electronic Engineering & Computer Science, Peking University, China

{wuyh10,zhangxd10}@sei.pku.edu.cn
{zhang.ying,xiongyf,hg}@pku.edu.cn

Abstract. High availability (HA) is a crucial concern in cloud systems. However, guaranteeing HA is challenging because of the complex runtime cloud environment, large number of HA mechanisms available, error-prone HA configuration, and ever-needed dynamic adjustment. In this study, we leverage runtime system architecture (RSA) for automatic configuration of HA in a cloud. With a causal connection between RSA and its corresponding cloud system, our approach has the following advantages. 1) The runtime changes of the system are abstracted, collected, and reflected on the RSA continuously, simplifying HA information collection. 2) With a model-based HA analyzer working on the collected HA-related information, an appropriate HA style (HAS) can be automatically selected for application to the current system. 3) Changing a HAS on RSA at runtime changes the system HA mechanism, which is suitable for the complex and ever-changing cloud environment. We implemented a prototype of our approach and evaluated it using our model-driven Yan-cloud platform.

Keywords: high availability, cloud computing, dynamic configuration, model-driven approach, runtime software architecture.

1 Introduction

Since the commercial value of cloud computing was established, it has become very important to guarantee high availability (HA). First, the runtime information of a system as well as its context needs to be collected for HA monitoring, which is a continuous and difficult task. Second, an appropriate HA mechanism such as rebooting or hot-sparing has to be selected for application to the critical parts of the system. Different mechanisms have different costs and advantages, and they can be applied on multi-layers, e.g., the physical host layer, the virtual machine (VM) layer, or the application layer. The selection of the most suitable mechanism that takes into account the runtime changes of the system requires a considerable amount of effort. Third, the manual configuration of the selected HA mechanism is complex and error-prone. The configuration file is often text-based, lacking supporting documents or a friendly interface [16]. Fourth, the HA configuration is not a one-time effort. The HA mecha-

nism needs to be continuously adjusted according to the changes in the user behavior and the evolution of the system.

Therefore, in order to solve the above mentioned problems, we have developed an RSA-based HA configuration framework for a cloud system. In this paper, we propose this framework. With a causal connection between the RSA and its corresponding cloud system, our approach has the following advantages. 1) The runtime changes of the system are abstracted, collected, and reflected on the RSA continuously, which eases the work of HA information collection. 2) With a model-based HA analyzer working on the collected HA-related information, an appropriate HA style (HAS) can be automatically selected for application to the current system. 3) Further, changing a HAS on the RSA at runtime will change the HA mechanism on the system, which is more suitable for the complex and ever-changing cloud environment.

Our HA configuration framework consists of three components, namely a runtime system monitor, an HA mechanism selector, and an HA configurator. The runtime information is reflected on the RSA in time by the runtime system monitor. For the selection of an appropriate mechanism from multiple HA mechanisms, a classification is proposed and a selector is implemented. The HA configurator is implemented by the merging two models, namely RSA and HAS.

The main contributions of this study can be summarized as follows. First, an RSA-based HA configuration framework is implemented. Second, we abstract several HA mechanisms into HA styles, which can be analyzed by the existing model process language or tools and configured using the RSA. Third, compared with the existing work [1], [4], [5], [17], our approach allows the continuous adjustment of the HA mechanism according to the runtime state of the system.

The rest of this paper is organized as follows: Section 2 presents some related work. Section 3 discusses the overview of our approach. Section 4 describes the construction of the RSA. Section 5 details the process of HAS selection, illustrated by an example of dynamic configuration in Section 6. Section 7 concludes the paper and briefly describes the future work.

2 Related Work

HA is an important property of a cloud platform. All cloud products provide some degree of support on HA. In this section, we introduce the HA mechanisms of the most popular business and open-source products: VMware Cloud, XenServer cluster, and OpenStack. Some automatic fault tolerant frameworks are discussed subsequently. Finally, a conclusion based on these works is discussed.

2.1 VMware HA

VMware HA is based on a cluster of physical machines. All the VMs running on these physical machines (PMs) in a cluster share the storage. If any of the PMs has an outage, all the VMs running on it will be restarted on other PMs [1]. This process is described in Fig. 1. In addition, the VMware cloud provides another HA mechanism

named VMware-FT. In other words, it is a hot spare. If VM-1 is FT-enabled, VM-2, which is synchronized with VM-1, runs on another PM. The synchronization process is carried out on the basis of the synchronization directives [1]. The term "VMware-FT" means zero downtime in the case of a PM failure because VM-2 will continue to work after VM-1's outage.

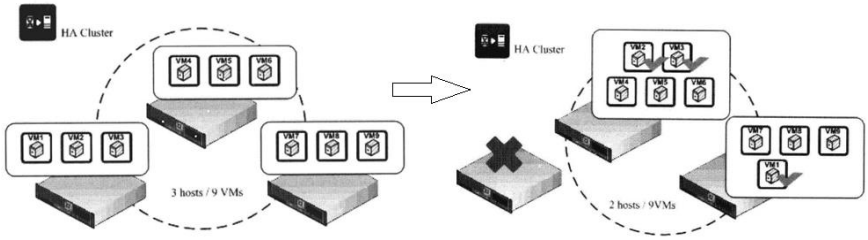

Fig. 1. Process of VMs booting on other PMs after a PM's outage

2.2 XenServer HA

Similar to VMware HA, XenServer HA deals with a PM failure. The structure of the XenServer resource pool is illustrated in Fig. 2. The resource pool is similar to the cluster in VMware, which is constituted by several PMs. XenServer Master is the control node that manages the VMs running on the XenServer PMs in this resource pool. All the VMs in a resource pool share the storage. If one or more PM failures occur, the master node will restart the VMs on the other live PMs. In the condition of a master node failure, the other PMs will elect a new master [4]. Remus [9] uses asynchronous VM replication to provide HA to VMs in the face of hardware failures. Just like VMware-FT, Remus has zero downtime. However, the asynchronous replication process is based on the memory state copy and not the synchronization directives as in VMware-FT.

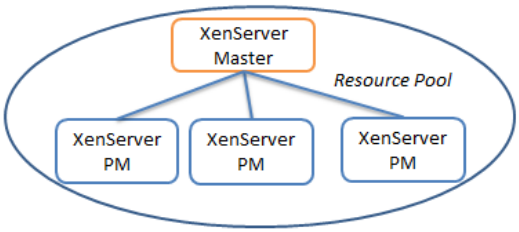

Fig. 2. Structure of a resource pool

2.3 CloudStack HA

CloudStack is a popular open-source cloud computing platform. Many successful business cases are based on this platform. In addition to the HA described in VMware, this platform provides more mechanisms: First, the HA mechanism will restart the VMs whose state is inconsistent with the state stored in the database records. Second, the HA mechanism will migrate some VMs when the load of a PM

exceeds the threshold value. Third, the master node can be deployed on multiple PMs used as hot spares because of its stateless property [5]. Fourth, the VMs on the failed PMs will be restarted on other working PMs. In CloudStack, multiple HA mechanisms are available.

2.4 Software Fault Tolerant (FT) Configuration

Some automatic FT configuration frameworks [17], [18] are proposed for cloud applications. BFTCloud [17] is a Byzantine FT Framework. An application or a service, which functions as the application-spare, is deployed on several PMs. For an incoming request, these services are executed simultaneously. A positive result is obtained on the basis of the voting of the results returned by these services. If a result from a PM is inconsistent with the positive result, the PM will be rejected from this service cluster. This FT configuration framework provides cloud HA on the application level.

Different from the application-spare in BTFCloud, [6] provides an FT configuration framework on the component level in an application. For component-based software, a suitable FT mechanism is selected for the components that affect system reliability more than others. The work described in this paper is a continuation of [6]. From the perspective of a cloud system, there are several component-level FT configurations because most of the applications are black-box applications.

2.5 Comparison

The HA mechanisms of the above mentioned three platforms perform excellently in practice. For example, VMware-FT [1] or Remus [9] ensures VM zero-downtime in the face of PM failures. This is very important in certain mission-critical systems. However, the cost is unbearable in most situations. For example, when using VMware-FT described in Section 2.1, half of the system resources (e.g., CPU, memory, and storage) are used for HA because a VM-spare is deployed on another PM. Although CloudStack supports several HA mechanisms for the tradeoff between cost and effect, there is no systematic configuration process in the existing HA solutions.

In this paper, we have proposed a runtime-system-architecture-based HA configuration process. First, we developed an HA style library. A HA style (HAS) [11] is an UML model abstracted from a HA mechanism. Second, based on the system meta-model constructed by system administrators, the synchronization engine between the RSA and the system is generated by SM@RT [10]. Third, one or more HA styles are recommended by the analysis of the RSA. Finally, the HASs are configured with the RSA and propagated into the system by the synchronization engine. Using such an approach, suitable HA mechanisms are triggered in the current context that will have a better effect and save more cost.

3 Overview of Proposed Approach

The proposed approach consists of the following processes: First, the RSA is constructed. Second, suitable HASs are selected on the basis of the information in the RSA. Third, the RSA is configured using HASs by model merging. Finally, the RSA changes are propagated into the cloud system. The detailed process can be described as follows:

1. Construction of RSA. The system information is reflected in the RSA, which is the basis of the proposed approach. The RSA is a runtime model responsible for the synchronization with the running system. Based on the RSA, we obtain the in-time information of the cloud system. In this study, we use the SM@RT tool [2], [10] for constructing the RSA model.
2. Selection of HASs. Multiple HA mechanisms are considered by the proposed approach, which has different gains and costs. An HA style (HAS), represented by the UML model, is abstracted from the HA mechanism, for the sake of system analysis and configuration on the architecture level. An HA style selector is implemented in the proposed approach. Depending on the system information and user requirements in the RSA, it chooses one or more suitable HASs to satisfy the HA requirements.
3. Configuration of RSA with HAS. In this step, we configure the RSA with the HAS by model merging at the architecture level. The RSA is constructed in Step 1, and HAS is selected in Step 2. After the configuration process, an RSA with HA is produced. It will satisfy one (or more) HA requirement(s).
4. Synchronization of RSA with the runtime system. After Step 3, we obtained the HA-enabled RSA. In this step, the synchronization engine, constructed in Step 1, will propagate the changes of the RSA to the target system. Figure 3 shows the entire proposed approach.

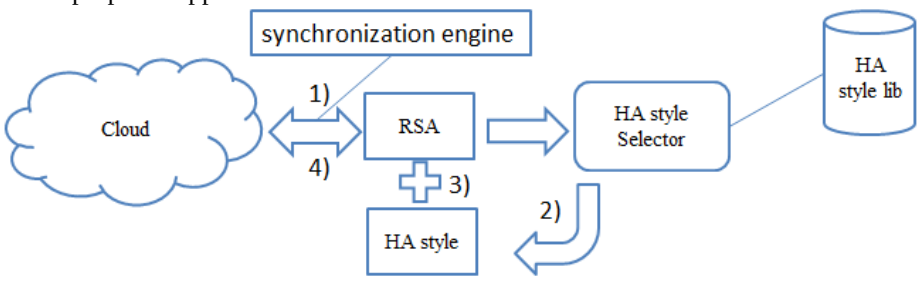

Fig. 3. Process of configuration of cloud with HA styles

4 Construction of RSA

For the construction of the RSA and the maintenance of a causal connection between the RSA and the running system, administrators need to define the meta-model and the access model of the target system.

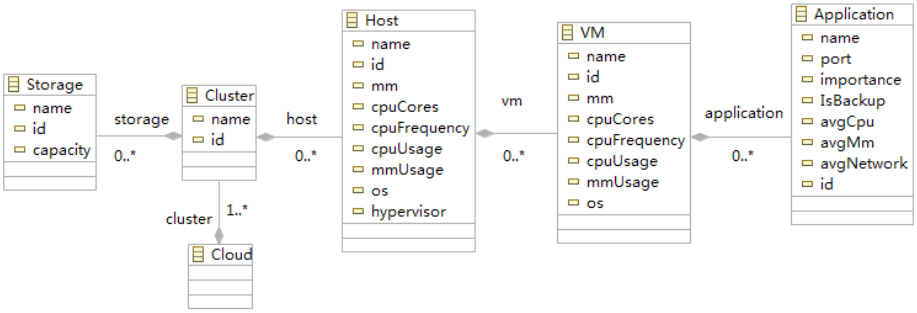

Fig. 4. Simplified HA meta-model of our cloud platform

The meta-model defines the structure of the RSA, including property, class, and association between classes. Fig. 4 shows a simplified meta-model of our RSA. The access model defines the methods used for obtaining (or setting) the runtime system information. In this study, the meta-model is constructed by using an eclipse modeling framework (EMF). We use a SM@RT engine [2] to automatically generate the access model and instantiate the RSA.

SM@RT [3], [10] provides a generative approach to assist the development of synchronization engines. It generates a synchronization engine that automatically reflects the running system onto a MOF-compliant model that conforms to the system meta-model.

5 Selection of HA Mechanisms

For simplifying the HA mechanism selection described in this section and the configuration discussed in Section 6, we propose a cloud architecture-based classification on HA. On the basis of this classification, we abstract the HA mechanism into an UML model, called HA style in this paper.

5.1 HA Mechanism Classification and Modeling

A cloud platform is often organized into multiple layers, as shown in Fig. 5. In the proposed approach, we provide HA mechanisms on four layers: APP (application), VM (virtual machine), PM (physical machine), and PM2PM (between PMs). In this four-layered HA classification, the lower layer provides HA for itself and the upper layers. For example, VM-HA can improve the availability of VM (③ in Fig. 5) and the applications (② in Fig. 5) running on it. The HA on PM and for APP (④ in Fig. 5), however, is not suitable for this virtualization infrastructure because the applications running on VMs are transparent to PM. In a cloud platform, a cloud management node is required for the management of the other VMs, which implies that the HA mechanisms between PMs (PM2PM) are available (⑦ in Fig. 5), such as the migration of a VM from a PM to another PM.

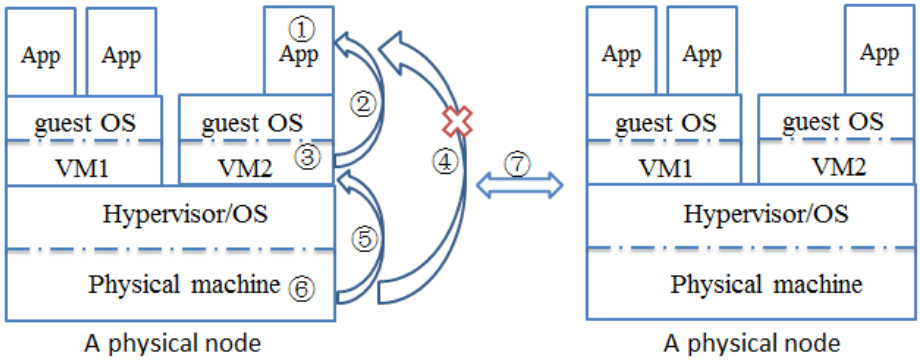

Fig. 5. HA architecture of cloud platform

For the architecture-level configuration, we abstracted these HA mechanisms into HA styles. An HA style is an MOF-compliant model that describes the structure and the behavior of a HA mechanism. Further information on HA styles can be found in our previous work [11]. We developed eight HA styles for the different layers, including PM2PM, PM, VM, and Application. These styles are detailed in Table 1. After analyzing the system information in the RSA, which contains user demands, we chose suitable HASs for a specific target with the help of a model-based HA analyzer. For example, a backup VM was started on another PM after a cloud user deployed a new VM.

Table 1. HA styles

HA layer	HA style
Application	HAS1: Fault tolerant mechanism in app HAS2: App redundancy HAS3: VM redundancy
VM	HAS4: VM hot-spare HAS5: VM snapshot
PM	HAS6: PM hardware monitor
PM2PM	HAS7: VM reboot on other PM HAS8: VM migration

Multiple HA styles could be selected simultaneously. For example, HAS6 and HAS8 could be used together. However, it is not advisable to deploy a large number of VMs on a PM in a cloud system as doing so would adversely affect the performance of the VMs running on this PM. Further, some VMs need to be migrated to other PMs in the situation of a PM hardware alarm.

5.2 Selection of HA Style

With the help of the different mechanisms on these four layers, we can find a way to conserve the cloud resources. For example, App-spare costs less than VM-spare, while the effect is the same for the application layer HA. Therefore, an HA style

(HAS) selector is implemented in the proposed approach. Two factors are considered in this HAS selector: cloud system information and user requirements.

A typical HAS selector could be implemented as follows: First, the user HA requirements are classified into the four layers (Application, VM, PM, and PM2PM) described in Table 1. Second, together with system information, a HAS is selected in the layers after considering the effect and cost. Here, we use an example to explain how the HA selection procedure works. As a popular web server, Apache [8] is used very often in a cloud platform. After the application "Apache" is deployed on a VM and the "HA-enable" tag is selected by the user, the system information in the RSA is analyzed. For ensuring high availability of "Apache" and for saving resources, the HAS selector will choose HAS2 as given in Table 1: app redundancy. On the other hand, if there are many HA-enabled applications on a VM, the HAS selector will choose HAS3: VM redundancy because the VM-spare avoids the redundancy for each application. The implementation of the HAS selector is ongoing. The development of a relatively efficient selection algorithm is part of our future work.

6 Configuration

With the advantages of RSA, we use a model merging process for the guidance of configuration. Model merging is a special kind of model transformation, and the function of model merging is to merge two models m_a and m_b, conforming to meta-model mm_a and mm_b, respectively, and the result is m_c, conforming to meta-model mm_c. M_a is called the receiving model, m_b is called the merged model, and the merging process is to merge the elements in m_b into m_a and produce a resulting model m_c [6]. In this paper, the merged model is HAS, and the receiving model is RSA. The resulting model is RSA with HA. After the model merging of RSA and HAS, changes on RSA will be propagated into the system by SM@RT automatically.

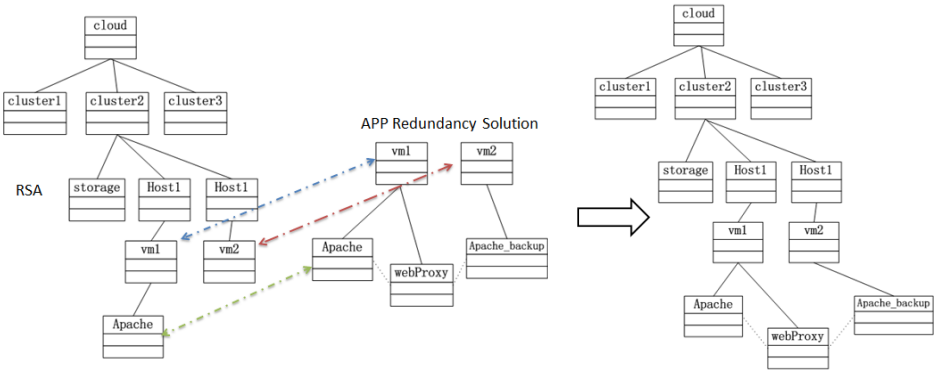

Fig. 6. Model merging process of RSA and HAS2

In this configuration process, a model merging process [6] is automatically carried out (see Fig. 6). Each mechanism is abstracted as an HA style, which is merged into the RSA. There are two phases in model merging: comparison and merging. In the first phase, it needs to determine the match relationship by element name in RSA and

HAS. The second step, merging, adds the elements in HAS to the RSA automatically in light of the match relation. Considering the example discussed in Section 5, we use "app redundancy" for the illustration in Fig. 6. A *webProxy* (in Fig. 6) is a proxy [7] in the front of the two web servers. It will not transmit the web request to *Apache_backup* unless the Apache on VM1 is down.

After the merging process, we obtain the RSA with HA. The synchronization engine will propagate the RSA changes to the cloud system. First, the Apache service will start on VM2. Second, the *webProxy* will be started on VM1, and it will be configured automatically. For example, the ip and the port of *Apache* and *Apache_backup* will be monitored by *webProxy*.

With the evolution of the system context, the HAS may change dynamically. For example, if we deploy more applications on VM1 and all these applications are labeled as "HA-enabled," the HAS selector will choose HAS3, as given in Table 1. Then, a new VM will be created as a backup, which will run all the application labeled as "HA-enabled" in VM1. The evolution of the RSA is shown in Fig. 7. It consists of two steps: Revocation (to carry out a revocation operation on the current RSA for the repeal of HAS2) and Merging (to configure HAS3 with the RSA). In HAS3, a heartbeat is deployed between VM1 and VM1-backup. Heartbeat [13], [14], the most famous subproject in Linux-HA [15], allows applications to identify the presence (or disappearance) of peer applications on other machines and to easily exchange messages with these applications. For example, the abovementioned heartbeat monitors the status of "Apache" on VM1 and that of "Apache-backup" on VM2. "Apache-backup" is visible to clients upon the failure of "Apache" or VM1.

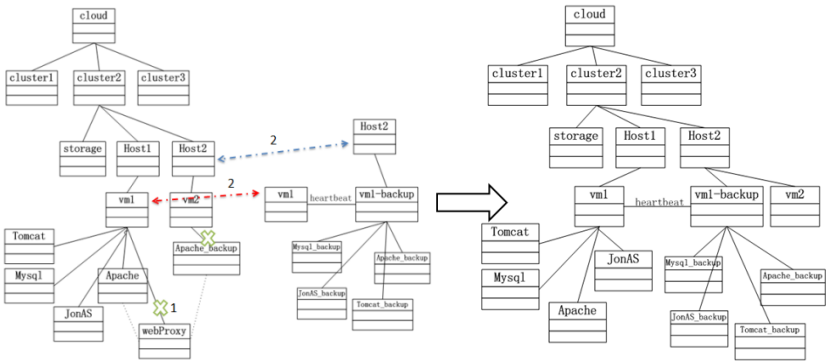

Fig. 7. The model merging process of RSA and HA3

Further information on model merging [12] can be found in our previous work [6]. The process is implemented by query/view/transformation (QVT), which is a standard set of languages for model transformation defined by the Object Management Group. After the model merging process, the RSA is configured with a new HAS, which is more suitable for the current system situation. With the help of the synchronization engines generated by SM@RT [2], the changes on RSA will be automatically propagated into the system.

7 Evaluation

This work is evaluated on Yan-Cloud. Yan-Cloud is an IaaS (Infrastructure as a Service) platform, which is developed based on CloudStack [5]. Some basic management modules have been realized: VM management, storage management, network management, project and user management, etc. We implement HA separately as a module. A CloudStack RSA is maintained by SM@RT. Based on this RSA, system information is analyzed by HA style selector. The result is an optimal solution to the trade-off between system availability and resource cost. Availability is estimated by the success rate of service requests. Resource cost is calculated by the CPU and memory consumption. As the system state is rapidly changing, HA mechanisms will be selected dynamically according to the current system information. After we deployed more applications in the VMs, some new HA mechanisms are recommended. Compared with others, the recommended mechanisms are the optimal solutions to the trade-off between availability and resource cost. More work and experimental data will be published in our future work.

8 Conclusion and Future Work

In the current cloud systems, there is no RSA-based HA configuration framework. In this paper, we proposed an RSA-based configuration approach of the HA mechanism for a cloud platform. Further, a prototype of the proposed approach was discussed.

However, the proposed approach needs to be implemented entirely and polished in the future. First, the classification of the existing HA mechanisms is not perfect. We intend to introduce more HA mechanisms and classify them more accurately in our future work. Second, the selection of an appropriate HA style is in progress. We plan to develop a more accurate and user-defined algorithm. Third, the HA configuration process will be implemented more automatically and be evaluated further in practice.

ACKNOWLEDGMENTS. This work is supported by the National High Technology Research and Development Program of China (863 Program) under Grant No. 2013AA01A208; the National Natural Science Foundation of China under Grant No. 61300002; the China Postdoctoral Science Foundation under Grant No. 2013M530011.

References

1. http://www.vmware.com/
2. SM@RT: Supporting Models at Run-Time, http://code.google.com/p/smatrt/
3. H. Song, G. Huang, F. Chauvel, Y. Xiong, Z. Hu, Y. Sun, H. Mei. Supporting Runtime Software Architecture: A Bidirectional-Transformation-Based Approach. Journal of Systems and Software, doi:10.1016/j.jss.2010.12.009.
4. http://www.citrix.com/
5. Apache CloudStack. “Apache CloudStack 4.0.0-incubating CloudStack Administrator's Guide”.

6. Y. Wu, G. Huang, H. Song, Y. Zhang, Model Driven Configuration of Fault Tolerance Solutions for Component-Based Software System. ACM/IEEE 15th International Conference on Model Driven Engineering Languages & Systems (MODELS 2012)
7. http://haproxy.1wt.eu/
8. http://www.apache.org/
9. B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: High availability via asynchronous virtual machine replication. In Proc. NSDI, April 2008.
10. Song, H., Xiong, Y., Chauvel, F., Huang, G., Hu, Z., Mei, H., 2009. Generating synchronization engines between running systems and their model-based views. In: Models in Software Engineering, the MoDELS Workshops (LNCS 6002). pp. 140-154
11. J. Li, X. Chen, G. Huang, H. Mei, and F. Chauvel. Selecting Fault Tolerant Styles for Third-Party Components with Model Checking Support. Component-based Software Engineering (CBSE) 2009.
12. H. Song, G. Huang, Franck C. , W. Zhang, Y. Sun, W. Shao, H. Mei, Instant and Incremental QVT Transformation for Runtime Models(MODELS), 2011,273-288
13. http://linux-ha.org/wiki/Heartbeat
14. I. Jody. Linux-HA: High-availability cluster, FreeBSD, Linux,2012,3-6.published: Cred Press
15. http://www.linux-ha.org/wiki/Main_Page
16. Z. Yin, X. Ma, J. Zheng, Y. Zhou, L. Bairavasundaram, and S. Pasupathy. An empirical study on configuration errors in commercial and open source systems. In Proceedings of 23rd ACM Symposium on Operating Systems Principles (SOSP), pages 159-172. ACM, 2011.
17. Y. Zhang, Z. Zheng, and M. Lyu, BFTCloud: A Byzantine Fault Tolerance Framework for Voluntary-Resource Cloud Computing. In Proc. of CLOUD'11, 2011, pp. 444-451.
18. Z. Zibin, T. C. Zhou, M. R. Lyu, and I. King, FTCloud: A Component Ranking Framework for Fault-Tolerant Cloud Applications. Proceedings of the 21th IEEE International Symposium on Software Reliability Engineering (ISSRE 2010), 2010, pp. 398-407.
19. P.n Oreizy, N. Medvidovic, R. N. Taylor. Architecture-based runtime software evolution. In Internetional Conference on Software Engineering, 1998, pages 177–186.
20. N. Abhigna. "Why Enterprises Need Multi-layered Defense Architecture" PC Quest (2011). LexisNexis. Web. 31 May 2011.
21. B. Charles. "When Amazon's Cloud Turned On Itself". Information Week (2011): 31.
22. S. Page. "CloudComputing-Availability". ACC 626-Section2.
23. T. Claburn. "Gmail Outage 'A Big Deal,' Says Google". TECHWEB (2009). LexisNexis.