A Reliability Assessment Framework for Cloud Applications

Xiaowei Wang
Institute of Computer Science
University of Goettingen
Goettingen, Germany
xiaowei.wang@cs.uni-goettingen.de

Jens Grabowski
Institute of Computer Science
University of Goettingen
Goettingen, Germany
grabowski@cs.uni-goettingen.de

***Abstract*—Cloud computing enables users to use computing resources, platforms and applications with reduced deployment and maintenance cost. The reliability of cloud applications becomes one of the key concerns of cloud service providers and users. Meanwhile, the deep dependency stack of layered cloud objects makes it challenging to evaluate the reliability of cloud applications. To tackle this problem, we propose a layered dependency graph-based reliability assessment framework. To verify our framework, we conduct an initial case study which shows its feasibility.**

Keywords-cloud application; reliability assessment; deep dependency.

I. INTRODUCTION

Cloud computing emerges as a promising paradigm that has potential to provide computing services as a utility [1]. It virtualizes computing resources (such as servers, networks, platforms and software) into resource pools, which can be used on demand via the Internet. In recent years, increasingly more companies and organizations have migrated their applications and data into clouds to reduce the in-house hardware and maintenance cost. Beside the rapid growth of cloud computing, the reliability of cloud applications is still on the road to satisfy cloud users. As unreliable cloud services may lead to revenue loss and data loss, the assessment and improvement of cloud system and applications' reliability attract significant attention of both academia and industry [2][3].

However, the deep dependency stack [4] of cloud objects, such as physical servers, virtual machines (VMs), platforms, services and management software etc., in different layers: Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS) and physical infrastructure, makes it a system-level task to assess the reliability of cloud applications, since the reliability of objects in upper layers is dependent on the reliability of objects in lower layers. The hierarchical dependency among cloud objects makes it tough to find out root causes of failures, i.e., where to put efforts to improve the reliability.

To address this issue, we propose a framework to assess and analyze the reliability of cloud applications. The framework utilizes a layered dependency graph to model dependencies between related cloud objects and the application deployed on clouds. Furthermore, a reliability assessment method is proposed based on the layered dependency graph. According to the modeled dependency and the field monitoring data, the reliability of each object and the application is assessed.

The rest of the paper is organized as follows: Section II discusses related work. Section III describes the layered dependency graph. Section IV illustrates the reliability assessment method of cloud objects. Section V introduces our framework. Section VI shows a preliminary case study of our framework. Section VII presents the conclusion and future work.

II. RELATED WORK

Recently, the assessment of cloud applications' reliability has become a hot research field. Zheng et al. [5] propose a framework to select the most significant components to determine the optimal reliability strategy for component-based cloud applications. But [5] does not take hardware components into consideration. In [6], Dai et al. divide the cloud service failures into request stage failures and execution stage failures, and then employ Markov model and graph theory to model and analyze the reliability of cloud services. Thanakornworakij et al. [7] propose a reliability model for high performance computing applications considering the correlation of software failures and hardware failures. However, neither [6] nor [7] considers the structure of the application. In [8], Tamura et al. propose a reliability model for open source cloud software focusing on the operational environment fluctuation. But [8] is more about the reliability of cloud systems rather than the reliability of cloud applications. Comparing with existing work, the framework proposed in this paper is capable to assess the reliability of cloud applications combining the reliability of software as well as hardware objects based on the structure of the application and the deployment of service instances.

III. LAYERED DEPENDENCY GRAPH

A cloud application is composed of several services, each of which has one or more service instances. For simplicity, we assume that one physical server can host more than one VM, but one VM can hold only one service instance. We define the chain of dependencies among service instances, VMs and physical servers as deep dependencies [4].

As assumption, some or all VMs used by a service may be deployed on the same physical server, as Figure 1 shows, which means that the failure of one physical server may bring down several service instances. The above case should be avoided when we improve the reliability of cloud applications. Therefore, a Layered Dependency Graph (LDG) is employed to model deep dependencies. A LDG contains three layers from bottom to top: physical server layer, VM layer and service instance layer, as shown in

Copyright (c) IARIA, 2015. ISBN: 978-1-61208-388-9

Figure 1. Cloud objects that are taken into consideration are service instances, VM instances and physical servers. In the service instance layer, service instances of one service type are clustered.

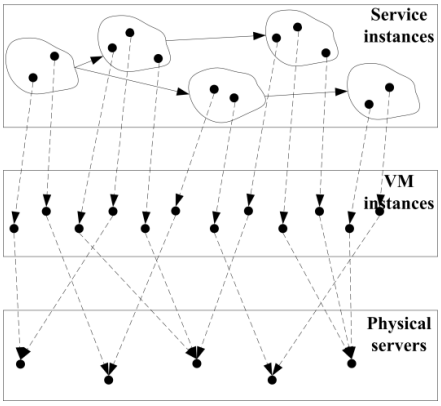

Figure 1. An example of the layered dependency graph.

We define two kinds of dependencies between cloud objects. The *function dependency* (solid arrows in Figure 1) is the relationship between two services that a service needs another one for its full function, e.g., a website needs a database to store users' information. And the *deployment dependency* (dashed arrows in Figure 1) is the relationship between objects in adjacent layers indicating that an object is deployed onto another one, e.g., a service instance is deployed on a VM.

IV. RELIABILITY ASSESSMENT

We define reliability as "the ability of a system or component to perform its required functions under stated conditions for a specified period of time" [9]. To illustrate the impact of dependencies to objects' reliability, we assume that the reliability (R) of a cloud object is determined by the reliability of itself (inner reliability, denoted by r) and the reliability (R_i) of objects on which it depends [10]. Based on the LDG model, the reliability of an object is represented as

$$R = r * \prod_{i=1}^{n} R_i \tag{1}$$

where n is the number of dependent objects. Equation (1), with the given inner reliability of all objects and time, can be used to obtain the application reliability based on the LDG. The reliability of cloud objects will be calculated in the following sections.

A. Physical Server Reliability

Physical server reliability (R_{PS}) is defined as the probability that a physical server performs its functions without failures in a period of time. Physical servers are considered failed when they crash or are unreachable. Physical servers depend on no other objects, as a result, their reliability is fully determined by their inner reliability (r_{PS}). Because physical servers work with constant failure rates (λ_{PS}) during the operational phase [6], we utilize the exponential reliability model to assess the physical server reliability with

$$R_{PS} = r_{PS} = e^{-\lambda_{PS} t} \tag{2}$$

where t is the working time of the physical server. λ_{PS} is usually evaluated by mean time to failure (MTTF) with

$$\lambda_{PS} = \frac{1}{MTTF}. \tag{3}$$

B. VM and Service Instance Reliability

As discussed in [8], the exponential distribution performs well for modeling software failures. Thus, we estimate the inner reliability of VM instances and service instances with the exponential reliability model by

$$r = e^{-\lambda t} \tag{4}$$

where λ is the failure rate of the object and t is the running time of the object. Software failures are not able to be tolerated by redundancy, except for timing or transient failures (called Heisenbugs) [11], which are usually caused by the complicated runtime environment. We assume that not only service instances of a service but also VMs on a physical server have the same failure rate.

VMs are considered failed if not in the running state or not reachable. Failures of the network, hypervisors and the cloud manager etc. that may lead to VM failures are deemed failures of VMs. Therefore, a VM will only fail due to VM failures or failures of the corresponding physical server, which means that all VMs on the same physical server will run or fail simultaneously, since they run in the same environment. The inner reliability of every VM on a physical server (r_{VM}) is

$$r_{VM} = e^{-\lambda_{VM} t} \tag{5}$$

where λ_{VM} is the internal failure rate of the VM.

Combining with the reliability of physical servers, we get the reliability of a VM with

$$R_{VM} = r_{VM} * R_{PS} \tag{6}$$

Equations (5) and (6) also apply for the inner reliability (r_{SI}) and reliability (R_{SI}) of service instances, respectively, with the service instance internal failure rate λ_{SI}, by

$$r_{SI} = e^{-\lambda_{SI} t} \tag{7}$$

$$R_{SI} = r_{SI} * R_{VM}. \tag{8}$$

C. Service Reliability

A service fails when all of its instances fail or at least one service that it depends on fails. Heisenbugs are usually caused by the complicated runtime environment, so, the inner reliability of all instances of a service is the same. Therefore, the inner reliability of a service is determined not only by the number of instances but also by the diversity of its VMs (i.e., deep dependencies).

Considering a service with three instances (Figure 2) deployed on three VMs, we will discuss the method to assess the inner reliability of the service (r_s) in different scenarios.

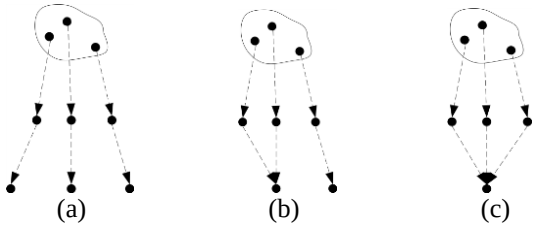

Figure 2. Scenarios of deploying a service with three instances.

a) If all three VMs are deployed on three different physical servers, as Figure 2(a) shows, then according to the assumption, the service will fail only when all the VMs fail (possibly caused by failures of all the physical servers), so, the service inner reliability is estimated with

$$r_s = r_{SI} * [1 - \prod_{i=1}^{3} (1 - R_{VM_i})] \quad (9)$$

where R_{VM_i} is the reliability of the *i*th VM.

b) If two of three VMs (e.g., VM_1 and VM_2) are deployed on the same physical server, because they run or fail simultaneously, it equals the scenario that the service has only two VMs deployed on two different physical servers and the service inner reliability is estimated with

$$r_s = r_{SI} * [1 - (1 - R_{VM_2})(1 - R_{VM_3})]. \quad (10)$$

c) If all three service instances are deployed on the same physical server, which means that any failures of a service instance, a VM or a physical server will make the service fail. It equals the scenario that only one service instance is running on this physical server. The service's inner reliability can be estimated with

$$r_s = r_{SI} * R_{VM}. \quad (11)$$

We can draw the conclusion from case *b)* and *c)* that if we want to improve the reliability of a service by increasing the amount of the service instance, the redundant services must be on different physical servers (regardless of performance issues).

Finally, the service reliability (*Rs*) is the multiplication of its inner reliability and the reliability of all services it depends on

$$R_s = r_s * \prod_{i=1}^{n} R_{s_i} \quad (12)$$

where R_{s_i} is the reliability of the *i*th dependent service and *n* is the number of dependent services.

D. Application Reliability

The application reliability is equal to the reliability of the service (e.g., S_1), which directly interacts with users and no other services are dependent on it, and is calculated with

$$R_{app} = R_{s_1}. \quad (13)$$

With the method of calculating the reliability of different kinds of cloud objects and the structure of the application, we can assess the reliability of cloud applications.

V. FRAMEWORK

In this section, we present a framework containing three components: a monitor, a dependency analyzer and a reliability analyzer, as Figure 3 shows.

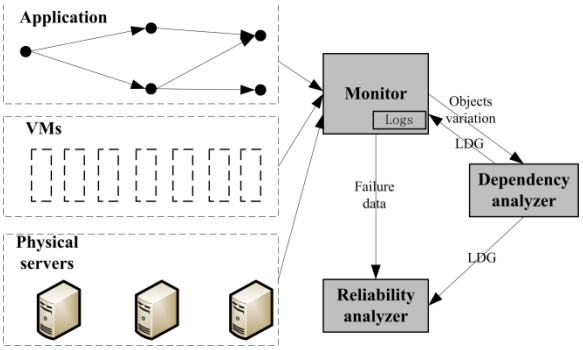

Figure 3. The reliability assessment framework.

A. Monitor

The responsibility of the monitor is twofold. The first task is to monitor and log the status, especially the failures, of all objects included in the LDG. The second task is to inform the dependency analyzer when any object fails, recovers from failures or joins the system. For instance, when a new service instance is started, the monitor transfers the name of the service to the dependency analyzer. The dependency analyzer will firstly query the information of the new service instance, the corresponding VM and physical server from the cloud manager, and secondly update and return the new LDG to the monitor.

B. Dependency Analyzer

The dependency analyzer is designed to create and update the LDG of the application. When deploying an application to the cloud, users need to input the initial

Copyright (c) IARIA, 2015. ISBN: 978-1-61208-388-9

function dependencies between services to the dependency analyzer. When the application is deployed, the dependency analyzer gets deployment dependencies from the cloud manager to build the LDG.

C. Reliability Analyzer

The reliability analyzer is responsible for assessing the reliability of the application and each object in the LDG by field failure data of all objects obtained from the monitor and the dependencies obtained from the dependency analyzer.

VI. Preliminary Case Study

To evaluate the framework, we implement a prototype with Java based on a Cloudify [12] PaaS cloud which is built on top of a private OpenStack [13] IaaS cloud.

The dependency analyzer collects the dependency information from Cloudify and OpenStack to create LDGs. Cloudify is employed to monitor service instances and VMs, and Ganglia [14] is used to monitor physical servers.

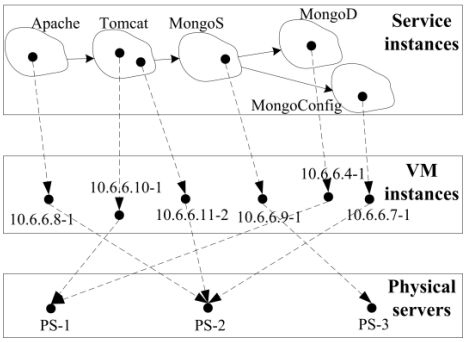

Figure 4. The created layered dependency graph.

We deploy a website using the Apache HTTP server [15] as the load balancer, Apache Tomcat [16] as the application server and MongoDB [17] (including three kinds of services, Mongos, MongoConfig and MongoD) as the database. The load balancer interacts directly with users and depends on Tomcat to fulfill functions. Tomcat is dependent on MongoS which depends on MongoConfig and MongoD. The LDG created by the dependency analyzer is shown in Figure 4.

We monitored the website and the cloud system for three days and obtained usage information of 71 visitors with 905 hits. From the monitoring logs, the internal failure rates of Apache HTTP server and the website deployed on Tomcat are 11/72 per hour and 4/72 per hour respectively. Based on the proposed reliability assessment method, the inner reliability of Apache HTTP server and the website is 0.8583 and 0.9460 respectively. So, the reliability of Apache HTTP server in one hour is

$$R_{apache} = 0.8583 * 0.9460 = 0.8120 .$$

No VM failures or physical server failures are observed during the three days, so, according to the proposed reliability assessment method, the application reliability is also 0.8120 in one hour.

VII. Conclusion and Future Work

In this paper, we propose a framework for assessing the reliability of cloud applications based on LDGs. As the preliminary case study shows, the framework can assess the reliability of cloud objects and applications. However, the preliminary experiment shows no VM or physical server failures. We are going to integrate a fault injector into our framework in the future. By setting the failure mode of cloud objects, our framework will be validated with more usage information and on more complex structures.

References

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," Future Generation Computer Systems, vol. 25, no. 6, Jun. 2009, pp. 599–616.

[2] G. DeCandia et al., "Dynamo: Amazon's Highly Available Key-value Store," ACM SIGOPS Oper. Syst. Rev., vol. 41, no. 6, Oct. 2007, p. 205–220.

[3] W. Zhao, P. M. Melliar-Smith, and L. E. Moser, "Fault Tolerance Middleware for Cloud Computing," Proc. of IEEE 3rd International Conference on Cloud Computing, Jul. 2010, pp. 67–74.

[4] M. Almorsy, J. Grundy, I. Müller, "An analysis of the cloud computing security problem," Proc. of APSEC 2010 Cloud Workshop, Sydney, Australia, Nov. 2010, pp. 109–114.

[5] Z. Zheng, T. C. Zhou, M. Lyu, and I. King, "Component Ranking for Fault-Tolerant Cloud Applications," IEEE Transaction on Service Computing, vol. 5, no. 4, Jul. 2011, pp. 540–550.

[6] Y. Dai, B. Yang, J. Dongarra, and G. Zhang, "Cloud Service Reliability : Modeling and Analysis," Proc. of the 15th IEEE Pacific Rim International Symposium on Dependable Computing, 2009.

[7] T. Thanakornworakij, R. F. Nassar, and C. Leangsuksun, "A Reliability Model for Cloud Computing for High Performance Computing Applications," Proc. of the 18th International Conference on Parallel Processing Workshops, Aug. 2012, pp. 474–483.

[8] Y. Tamura, M. Kawakami, and S. Yamada, "Reliability modeling and analysis for open source cloud computing," Proc. of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability, vol. 227, no. 2, Apr. 2013, pp. 179–186.

[9] IEEE, "IEEE Standard Glossary of Software Engineering Terminology," IEEE Std. 610.12-1990, IEEE, 1990, pp. 1–84.

[10] V. Cortellessa and V. Grassi, "Reliability Modeling and Analysis of Service-Oriented Architectures," Test and Analysis of Web Services, pp. 339–362, 2007.

[11] M. R. Lyu, Handbook of Software Reliability Engineering, IEEE Computer Society Press and McGraw- Hill, pp. 574, 1996

[12] http://www.getcloudify.org/, [retrieved: Nov. 2014]

[13] http://www.openstack.org/, [retrieved: Nov. 2014]

[14] http://www.ganglia.info, [retrieved: Nov. 2014]

[15] http://httpd.apache.org/, [retrieved: Nov. 2014]

[16] http://tomcat.apache.org/, [retrieved: Nov. 2014]

[17] http://www.mongodb.org/, [retrieved: Nov. 2014]

Copyright (c) IARIA, 2015. ISBN: 978-1-61208-388-9