Journal of Universal Computer Science, vol. 19, no. 15 (2013), 2241-2265
submitted: 13/11/12, accepted: 30/8/13, appeared: 1/9/13 © J.UCS

Toward a Module-centralized and Aspect-oriented Monitoring Framework in Clouds

Kun Ma
(Shandong Provincial Key Laboratory of Network Based Intelligent Computing, University of Jinan, Jinan, China
ise_mak@ujn.edu.cn)

Runyuan Sun
(Shandong Provincial Key Laboratory of Network Based Intelligent Computing, University of Jinan, Jinan, China
sunry@ujn.edu.cn)

Ajith Abraham
(Machine Intelligence Research Labs, Scientific Network for Innovation and Research Excellence, Auburn, USA
ajith.abraham@ieee.org)

Abstract: Currently, monitoring plays an important role in managing the Cloud computing environment. However, the Cloud computing owners and tenants often lack the management and monitoring tools to ensure the performance, robustness, dependability, and security. To address this limitation, this paper describes the development of a lightweight module-centralized and aspect-oriented monitoring framework. This framework performs end-to-end measurements at virtual and physical machine instances, software and Web service in the Cloud. It monitors the quality of service (QoS) parameters of the IaaS and SaaS layer in the forms of plug-in bundles. In addition, we discuss the manager-agent monitoring of entity objects and aspect-oriented Cloud service monitoring in detail. All the modules constitute the entire proposed framework to improve the performance in hybrid Clouds.

Key Words: Cloud monitoring, aspect-oriented programming, performance evaluation, Cloud computing
Category: K.6, K.6.3, K.6.4, K.6.5

1 Introduction

Along with the Cloud's increasingly central role of the services in the industry, monitoring Cloud applications and services as well as the applications deployed on them is becoming a priority [Dastjerdi et al. 2012]. Cloud monitoring may be viewed as a specialization of distributed computing monitoring and therefore inherits many techniques from traditional computer and network monitoring. However, as Cloud computing environments are considerably more complex than those of legacy distributed computing, they demand the development of new monitoring methods and tools [Van et al. 2009]. Overall, the integration

of Cloud monitoring with related techniques to effect an end-to-end automated monitoring and provisioning process over Cloud environments is a hitherto neglected research area.

In order to solve this problem, this paper describes our experience with a hybrid Cloud, and discusses the architecture of a lightweight hybrid Cloud monitoring framework. The monitoring framework is composed of two layers in order to reduce the complexity: manager-agent monitoring of entity objects, and aspect-oriented Cloud service monitoring. The first part of our monitoring framework is manager-agent monitoring. The manager is a separate entity that is responsible to communicate with the agent. Enabling the agent allows it to collect the monitoring data from the device locally. However, this approach applies even more to monitor the entity objects. For the monitoring of Cloud Web services, this approach is inferior. Therefore, we propose another aspect-oriented Cloud service monitoring approach as a supplementary item. Aspect-oriented monitoring approach is scattered by virtue of the functions of Web services, which can measure the QoS parameters of Cloud Web service. Manager-agent and aspect-oriented monitoring approaches works in parallel, which constitute the whole monitoring framework.

Currently, third-party Cloud Web services have spread among tenants and end users, providing a number of advantages over infrastructures. Although Cloud providers claim a higher resilience, assessments of their actual availability are missing [Naldi 2013]. Despite the advantages of cloud computing, small and medium enterprises in particular remain cautious while implementing Cloud service solutions. The main reasons for the companies to adopt cloud computing as follows [Sunyaev and Schneider. 2013]. First, companies lack qualified and trustworthy benchmarks to assess and monitor Cloud services. Furthermore, companies lack approaches and metrics to adequately evaluate the quality of Cloud services. The motivation of our framework is just the requirement of such monitoring system for a hybrid Cloud.

We argue that there is no generic Cloud monitoring solution that may be applied to cover the three major cloud service models: infrastructure as a service (IaaS), software as a service (SaaS), and platform as a service (PaaS). We choose to address the usage of Cloud resources described by a set of quantitative parameters. These parameters are divided into three categories: IaaS parameters, SaaS parameters, and user experiences. As the PaaS models are emulated over an IaaS and SaaS base, we do not discuss it in this paper. IaaS parameters include the performance of physical hypervisor, the performance of virtual machines, and the availability. While SaaS parameters include the performance of applications in the Cloud, and the performance of Web services. We take user experience as an instructional factor, since it can help the tenants of the Clouds to learn the user habits of end users. This is one of the effective ways to improve

the quality of Cloud service.

The contributions of this paper are several folds. First, it presents the design and implementation of the hybrid Cloud monitoring framework with open source solutions and extra significant development work. Second, the extension fine-grained mechanism of manager-agent monitoring is based on plug-ins bundles, which is an effective way for the monitoring module to load custom plug-ins dynamically. Through a simple method built quickly from freely available parts, it is partially successful, suggesting this monitoring framework is used both in public and private Clouds. Besides, aspect-oriented Cloud service monitoring allows the developer to dynamically write references to aspects at join points to calculate the performance of Cloud services. Aspects thus eliminate many lines of scattered code that the developers would otherwise have to spend considerable time in writing the monitoring codes.

This study is the follow-up work of [Ma et al. 2011], [Ma et al. 2012a], and [Ma et al. 2012b]. This paper further develops the work on the lightweight framework for monitoring public Clouds [Ma et al. 2012c]. First, we have reorganized the structure of the paper, which would help the readers to gain insight into our motivations. Second, this extended framework supports both public and private Clouds. The extended entity objects support more types of monitoring, such as hypervisors and over-commit monitoring. Third, the monitoring framework in our previous work has some limitations. It does not support the monitoring of Cloud service. In the extended paper, we have extended the framework to support the Cloud Web service monitoring by aspect-oriented programming. Finally, some implementation details are described in the extended paper, such as the manager-agent architecture, aspect-oriented monitoring architecture, and database monitoring. It is important to note that our proposed monitoring system is intended as a system running by not only the third-party Cloud service providers, but also the Cloud tenants who provides Cloud service to the end users. With this mechanism, the monitoring system is also a Cloud software as a service.

The remainder of the paper is organized as follows. The related work and its limitations concerning the requirements for Cloud monitoring methods are discussed in Section 2 In Section 3, the monitoring models are introduced. Section 4 discusses the module-centralized and aspect-oriented monitoring framework. The implementation details of this framework is proposed in Section 5 and 6. Conclusions and future work are provided in the last section.

2 Related Work

2.1 Cloud monitoring categories

Different Cloud deployment models [Mell and Grance 2011] have different requirements. The differences are most distinct between private and public Clouds [Jamsa 2012]. Private Clouds, even if scalable, are limited to the resources owned by the operating organization. Regarding security, in private Clouds, the data is under the organization's control, whereas public Clouds require continual surveillance across multiple cyber attack vectors. Public Clouds often have geographically diffuse, large resource pools, which require more investment in monitoring traffic and ensuring scalability. Service metrics such as link availability and connection speed are essential information in public Clouds but are of little value in private Clouds. In public Clouds, firewall settings may limit what can be monitored between Cloud providers. Finally, public Clouds need to provide monitoring information for clients, which requires more flexibility, customizability, and security. Hybrid Cloud mixes the techniques from public and private Clouds [Peng et al. 2009]. The benefits and challenges are a combination of the items above. As a consequence, the topic of this paper relates to hybrid Cloud monitoring. In our paper, we attempt to design and implement a mass of modules to support further monitoring of hybrid Cloud.

2.2 Cloud monitoring methods

Monitoring of computing resources has been a hot topic of research interest and development for many years. However, monitoring in Clouds faces many new challenges. First, due to the heterogeneity of components in the Clouds, individual computer and network monitoring solutions and mechanisms need to be designed and implemented respectively, which is very costly [Shao et al. 2010]. Second, much more monitoring concerns need to be covered in Clouds than traditional monitoring software, such as multi-tenancy and hypervisors. Third, the large number of services, tenants, and end users in Clouds lead to the demand for a more intuitive and flexible way to implement monitoring in Cloud.

By exploring the recent literature, we discovered several monitoring systems, each one with its particular characteristics and abilities. Well-known Clouds in the industry have their particular monitoring utilities. We will classify the available mechanisms and point out the drawbacks and inefficiencies.

The first taxonomy is Cloud monitoring level. A typical cloud architecture would include the levels: server, infrastructure, platform and application [Aceto et al. 2013]. On the one hand, the lowest level is server, which contains physical machines and network [Ciuffoletti et al. 2010]. On the other hand, the remaining three levels contain the virtual resources provided by the tenants

and end users [Dhingra et al. 2012]. Current open-source monitoring tools are designed for the former. There are plenty of tools for this taxonomy. Nagios [Issariyapat et al. 2012] is the industry standard in the infrastructure monitoring, offering complete monitoring and alerting for servers, switches, software, and services about the status of resources. It is designed to run checks on hosts and services using several external plug-ins and return the status information to administrative contacts [Imamagic and Dobrenic 2007]. Although it includes valuable features and abilities, it does not provide a generic API and perform under small-time interval. Eucalyptus [Nurmi et al. 2009] is open source computer software for building Cloud computing environments. Systems that give users the ability to run and control entire virtual machines deployed across a variety physical resources. However, this tool is commonly referred to for IaaS only. Ganglia [Massiea et al. 2004] is a scalable distributed monitoring system for high-performance clusters and grid computing. It leverages widely used technologies and is at the base of a hierarchical design targeted at federations of clusters. The disadvantage of the method is not good for bulk data transfer (no windowed flow control, congestion avoidance, etc.). CloudSense [Kung et al. 2011] provided a new switch design that performs continuous fine-grain monitoring via compressive sensing. This framework used MapReduce straggler monitoring to report conventional status via the analysis and emulation. Messina et al. proposed a trust-based approach to make a node capable of finding the most reliable interlocutors [Messina et al. 2012]. This approach avoids the exploration of the whole node space.

The second taxonomy is Cloud monitoring vision. From a general perspective, client-side and Cloud-service-provider-side monitoring can be distinguished [Montes et al. 2013]. These two complementary visions address different Cloud monitoring requirements, creating differentiated views of the system behavior and evolution. There are plenty of approaches for this taxonomy. Clayman et al. find a monitoring framework for Clouds service side, which succeeds in the scalability of the monitoring [Tselentis et al. 2010]. This framework is spread in different layers (service, virtual environment, physical resources, etc.). The proposed framework offers the libraries and tools to build its own monitoring system. However, they do not present performance metrics and point out the boundaries. Besides, there are commercial Cloud solutions, which often make use of their own monitoring systems. However, in general these systems have limited functionality, providing only a fraction of the available information. Amazon CloudWatch [CloudWatch 2013] and OpenNebula [OpenNebula 2013] monitoring systems are the examples. For the private Cloud, it is easy to monitor the client-side and Cloud-service-provider-side [Chaves et al. 2011], since we have total control of it. For the public Cloud, we provide only client-oriented monitoring. Those Cloud monitoring levels and visions motivated us to implement a

lightweight framework that monitors hybrid Clouds. To this end, the design of a monitoring system should keep up with the expansion of the flexible ability, when an application or infrastructure scales up or down dynamically.

Dixon envisioned monitoring systems with interchangeable components focused on a single responsibility [Dixon 2012]. According to his advice, such a system architecture should show the following characteristics: resilient, self-service, automated, correlative and craftsmanship. As a result, the support of the monitoring entity objects in our framework depends on the plug-ins.

2.3 Open source monitoring tools

There are several famous monitoring tools. Round Robin Database Tool (RRDtool) [Russell and Cohn 2012] is the open-source industry standard, high-performance data logging and graphing system for time-series data. The data analysis part of RRDtool owns the ability to create graphic representations of the data values collected over a definable time period. However, the current plug-ins of RRDtool can not support the Cloud monitoring, such as the hypervisors. Collectd [Forster and Harl 2013] gathers statistics about the system it is running on and stores this information. Those statistics can then be used to find current performance bottlenecks and predict future system load. With these monitoring tools, it is possible to deploy a hybrid Cloud monitoring solution using extra significant development work. Although these tools are not designed for Cloud monitoring, the refactor of them may adapt to the Cloud.

2.4 Cloud Web service monitoring in the Cloud

For the service in the Cloud, the above-mentioned monitoring methods are not applicable because of the heterogeneous content. Shao et al. [Shao et al. 2010] propose a runtime model for Cloud monitoring (RMCM), which denotes an intuitive representation of a running Cloud by focusing on common monitoring concerns. Raw monitoring data gathered by multiple monitoring techniques are organized by RMCM to present a more intuitive profile of a running Cloud. In the SaaS layer, it monitors the applications with respect to their design models and required constraints. For this issue, it converts the constraints to a corresponding instrumented code and deploys the resulting code at the appropriate location of the monitored applications. This makes RMCM an invasive approach, since it modifies the source code of the applications. Cao et al. [Cao et al. 2009] propose a monitoring architecture for Cloud computing. It describes a Quality of Service (QoS) model that collects QoS parameter values such as response time, cost, availability, reliability and reputation. Although their architecture is interesting, the implementation of this architecture is not clear.

2.5 Aspect-oriented programming

Aspect-oriented programming (AOP) is based on the idea that computer systems are better programmed by separately specifying the various concerns (properties or areas of interest) of a system and some description of their relationships, and then relying on the mechanisms in the underlying AOP environment to weave or compose them together into a coherent program [Elrad et al. 2001]. The key difference between AOP and other approaches is that AOP provides component and aspect languages with different abstraction and composition mechanisms. A special language processor called an aspect weaver is used to coordinate the co-composition of the aspects and components. Modularized crosscutting concerns are called aspects. An aspect, if it can not be cleanly encapsulated in a generalized procedure. AOP distinguishes between two approaches. Static crosscutting affects the static type signature of a program, whereas dynamic crosscutting allows to intercept a program at well-defined points in its execution trace. After evaluating the flexibility of the monitoring system, we decided to pursue dynamic crosscutting option over ease of implementation.

In dynamic crosscutting, join points are well-defined points within an execution trace of a program. Pointcuts are sets of join points and advice are method-like constructs that define the behavior of these join points. The pointcut language defines abundant join points where the advice is integrated into the code. In this manner, aspect-oriented programming eases the development of reusable and maintainable code.

We adopt aspect-oriented programming techniques to implement the Web service monitoring in the Cloud. The capabilities of AOP in terms of isolating the aspect code from the source code of the used server make it a non-invasive monitoring approach. This framework can be fully integrated with the Cloud client and Cloud server. It should be pointed out that this framework is a generic approach that can be implemented in several heterogeneous distributed monitoring contexts.

3 Monitoring Framework

3.1 Cloud monitoring architecture

This section presents a generic layered cloud monitoring architecture adapted to the requirements of any cloud infrastructure, presented in Figure 1. The architecture is divided into four main components: the first one, regarding the IaaS monitored object, the second one, regarding the SaaS monitored object, the third one, regarding the monitoring access, and the last one, regarding the data gathering. Since the monitoring of Platform as a service (PaaS) depends on the specific platform, we do not discuss it in our paper.

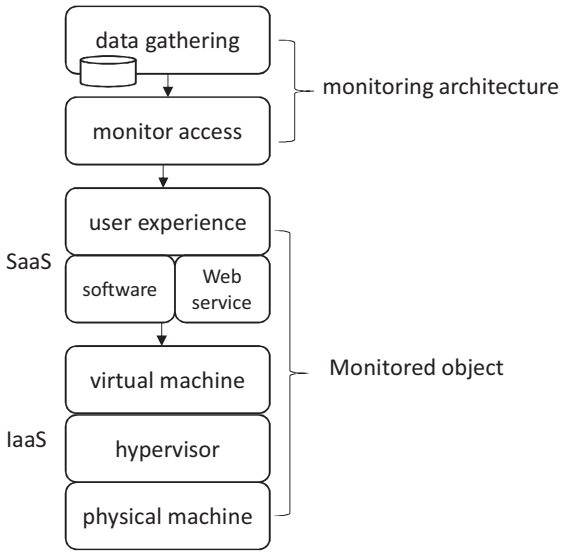


Figure 1: Cloud monitoring architecture

At the bottom is the physical resources, such as the hardware resource utilization (CPU, memory, disk, networking et al.).

The second layer from the bottom is the IaaS monitored object, including the physical machines, the hypervisors that manage the guest operating systems, and the virtual machines. They are all the monitored objects in IaaS level. This performance metrics, which stays the same regardless of distinguished infrastructures, are extracted to model the runtime infrastructure in the Cloud.

The third layer from the bottom is the SaaS monitored object, including the applications, the Web servers and the Web services. They are all the monitored objects in SaaS level. These applications provide a wide set of services that assist developers in delivering a professional and commercial service to end users. Commonly, the software in the SaaS layer will expose their own interface or API for monitoring and management, and this metric is gathered and organized to form the runtime monitoring model of this layer. Besides, the user experience is also an important entity to monitor. This point is generally omitted by the traditional monitoring tools. We put the user experience of Cloud tenants as a reference to discover the relationship between users' habits and Cloud performances. Also it is the guidance on the future management of the Cloud. Interactive information and action between end users and Clouds are monitored. It mainly contains the frequency of access, IP distribution, time of staying and so on. The service developers may adapt their software to attract more users of hybrid Clouds.

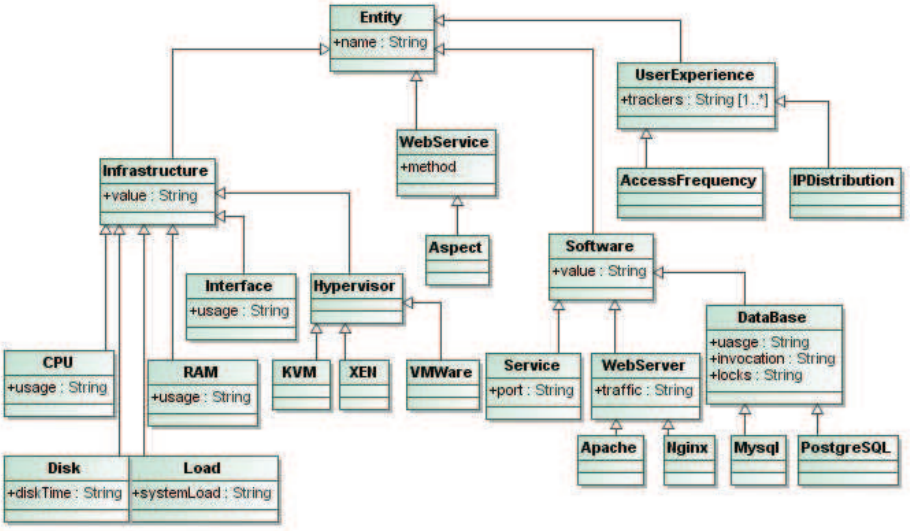


Figure 2: Hierarchy of entity models

The second layer from the top is monitor access, which provides an independent way to access the monitored information from the bottom. It is an abstraction layer that provides a unique view of both physical and virtual Cloud monitored objects, regardless of its different objective, composition and structure. This layer accesses both the virtual and physical systems.

At the top is data gathering, which provides the storage for the monitoring information obtained in the previous layer. It includes an historical archive of the evolution of the different cloud objects. Since each user has its own Cloud vision and monitoring level, this layer does not only store monitoring data but it coordinates the user queries. In this way, each user can access the monitoring level and vision required.

3.2 Hierarchy of resource entity models

Concrete resource entities in a running hybrid Cloud are organized in a hierarchy, as depicted in Figure 2. This hierarchy can be extended by adding new entities emerging in different Cloud layers. Such an organization makes an extension to the model more feasible. New entities can be added in directly by inheriting one of the existing entities. Each entity consists of a set of key/value pairs. Children entities can inherit attributes from their parents. And values are obtained from real-time Cloud monitoring statistics. Based on the entities, we design the mon-

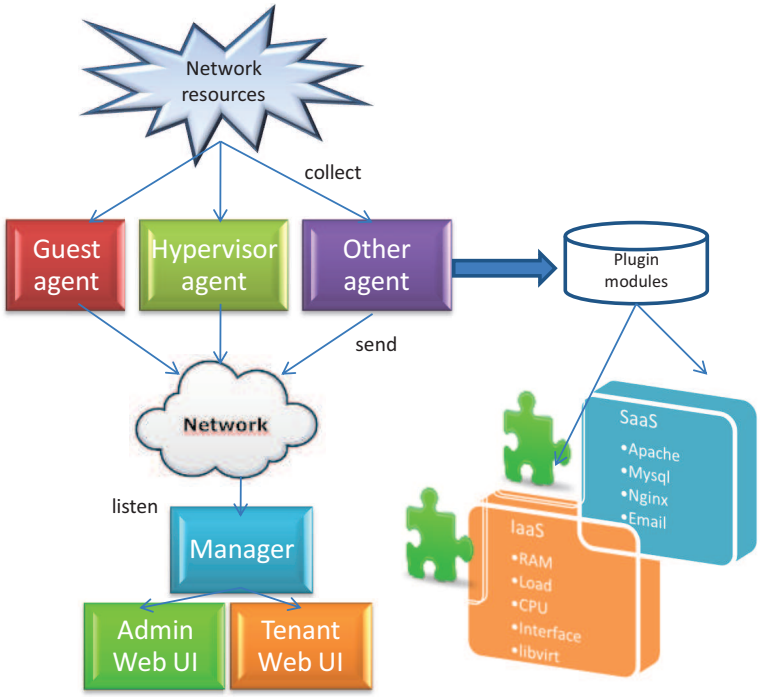


Figure 3: Manager-agent architecture and its plug-ins

itoring system. For further monitoring, we can extend the hierarchy of resource models.

4 Monitoring framework architecture

4.1 Manager-agent architecture

In a Cloud, multiple entities need to be monitored simultaneously to coordinate their resource utilization to achieve a balance. Therefore, the framework is required to monitor the resources individually and take decisions centralized. This paper achieves a manager-agent styled monitoring framework, as shown in Figure 3. The manager provides the interface between the human network manager and the management system. While the monitoring agent is a small daemon which collects system information periodically. In order to analyze the collected data, we provide the mechanisms to store and monitor the values in a variety of ways. These agents are responsible for collecting runtime information and sending the UDP packet to the manager. By default, we provide the basic monitoring of common metrics: IaaS monitoring plug-ins, and SaaS monitoring

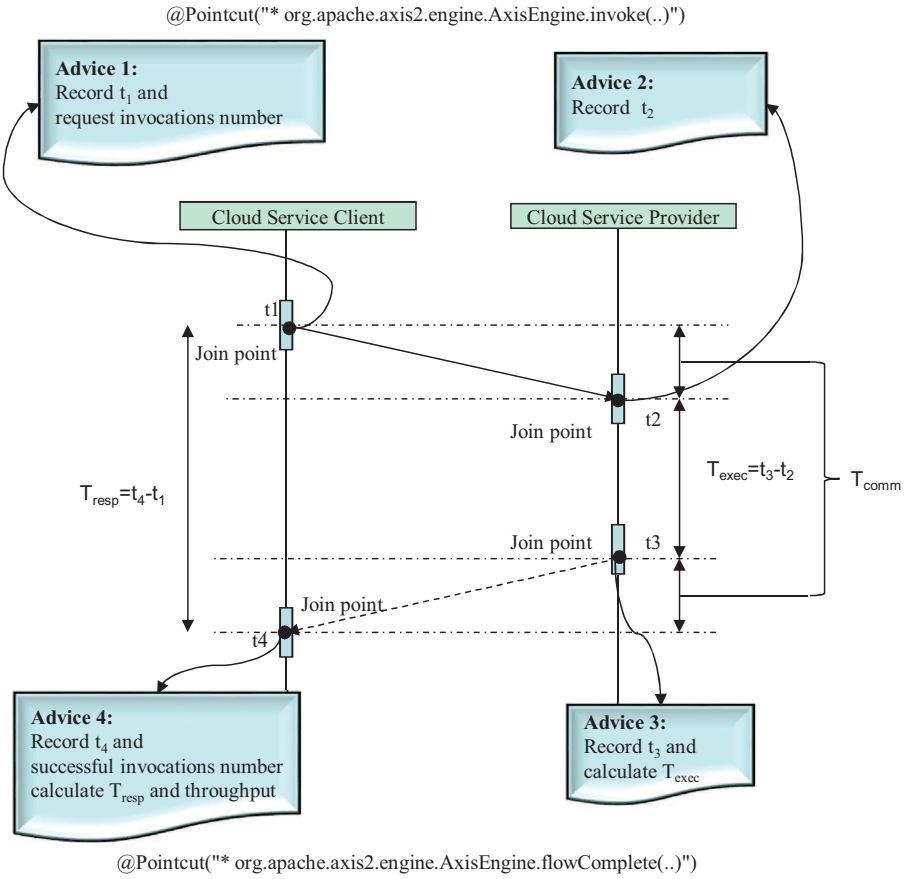


Figure 4: QoS parameters and the sequence diagram

plug-ins. Other plug-ins are written to allow the developer to further extend the default monitoring system. All the personalized plug-ins conforming to our interface specification can be remotely deployed and installed without requiring a reboot in the forms of bundles. We provide the user interface with the administrators and tenants of hybrid Clouds. This will help them view the operating condition of hybrid Clouds.

4.2 Aspect-oriented cloud service architecture

An overview of aspect-oriented Cloud service architecture is given in this section. As depicted from Figure 4, there are two objects. The first object is the Cloud service client, which is located at client side. From this point of view, monitoring

information of this type helps the tenant client to understand the QoS received and optimize their use. The second object is the Cloud service server, which is located in Cloud-service-provider side. From this point of view, monitoring information of this type provides the knowledge about the internal functioning of the different Cloud objects with the Cloud administrators in order to guarantee QoS. It can be also used as usage and performance log to optimize the service provided.

This framework measures five QoS parameters: the execution time, the response time, the communication time, the throughput and the availability. They are explained below. All the terminology is shown in Figure 4.

- The communication time T_{comm} is the time needed to transfer the request from the client to the server plus the time needed to transfer the response from the server to the client. It is the sum of the two parts. Thus, the response time shown in Equation 1 is the sum of the time necessary for executing the request and the communication time.
- The execution time T_{exec} measures the time needed to execute a request on the server.
- The response time T_{resp} defines the time needed to serve a request. It starts when the client sends its request and finishes when the client receives the corresponding response. As a result, it is the temporal difference between the instant (t_1) when the client invokes the request and the instant (t_4) when the client receives the corresponding response.

$$T_{resp} = T_{comm} + T_{exec} \tag{1}$$

- The *throughput* measures the number of successful requests ($SuccRequests$) during a period of time T. In other word, the *throughput* is measuring the success number of requests per second.

$$Throughput = \frac{SuccRequests}{T} \tag{2}$$

  where $SuccRequests$ represents the number of successful requests during a period T. T is a parameter that is set when this framework is configured. We mean that the successful request is with a successful response reached the server.

- The *availability* measures the accessibility of a service. It is calculated using the formula shown in Equation 3.

$$Availability = \frac{SuccRequests}{AllRequests} \tag{3}$$

where $AllRequests$ is the number of all requests sent during the period T.

Our framework is based on the aspect-oriented programming code that intercepts the methods of the client and server at well defined join points to collect data and measure these parameters at important instants of time. These instants are t_1 - t_4, where

- t_1 is the instant when the client invokes the request,
- t_2 is the instant when the server receives the request,
- t_3 is the instant when the server sends the response,
- t_4 is the instant when the client receives the response.

The proposed aspect code computes the number of request invocations by advice 1. At the mean time, the unique identity is also recorded. Generally, the unique identity is composed of the IP address of the client and the client session. In addition, the execution time is calculated by advice 3. Moreover, the number of successful sent requests and response time is evaluated by advice 4.

Recorded timestamps help us to calculate QoS parameters. The procedure is as follows: First, our framework creates four specific join points. Each one corresponds to an instant (t_1, t_2, t_3 and t_4) intercepting method calls. Second, it executes the corresponding advice (for each join point) which consists of recording the timestamp and calculating the number of invocations at the client (advices 1 and 4). When all instants have been processed, it computes the difference between t_4 and t_1 to deduce the response time. It also subtracts $t2$ from $t3$ to calculate the execution time. The difference between the response time and the execution time represents the communication time. It assesses the throughput by calculating the number of successful invocations at the client side and dividing this number by a period of time T as described in Equation 2. Moreover, it calculates the availability as described in Equation 3.

5 Implementation of Monitoring Modules

As depicted in Figure 1, this section discusses the implementation of the architecture. For the IaaS monitored objects, we implement the monitoring of the physical machine, the hypervisor, and the virtual machine. The monitoring of the physical machine presents an overall of metrics for the whole cloud; the monitoring of the hypervisor presents an overall of metrics for the guest virtual machines without the access to the internal machines; the monitoring of the guest virtual machine presents an overall of metrics for the internal machines separately. For the SaaS monitored objects, we implement the monitoring of the software, the Web service, and the user experience.

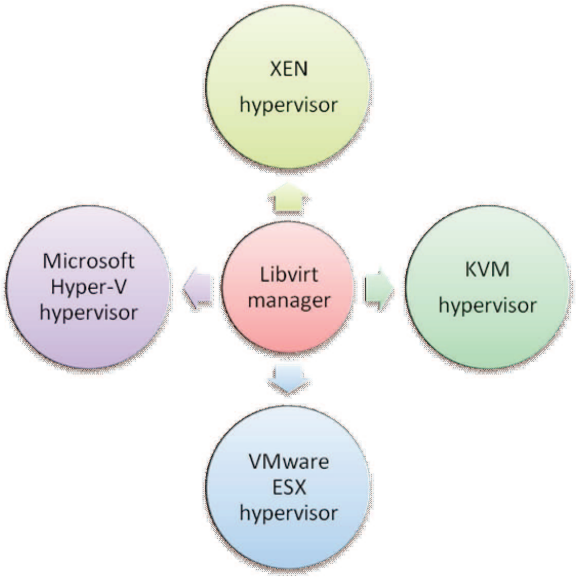


Figure 5: Infrastructure libvirt monitoring architecture

The proposed monitoring framework provides the comprehensive monitoring in the forms of different plug-ins. These plug-ins include virtualization, availability, performance via SNMP, user experience tracker, and over-commit monitoring. It provides many kinds of alert and information statistics to help the administrator find the problems in real time. Each module is a daemon, which collects the data from the monitored object.

5.1 Virtualization monitoring module

This monitoring module uses the libvirt API to gather the statistics of the guests on a system shown in Figure 5. The libvirt public APIs support many commonly hypervisor drivers, such as KVM, XEN, Hyper-V, VMware ESX and VirtualBox [Han et al. 2011]. With libvirt plug-in, CPU, memory, networking and device usage for each guest could be collected without installing any software on the guest. As the module receives statistics from the hypervisor directly, it is suitable for the physical and virtualized machines. The system load collected from the last one week is shown in Figure 6. Due to the delegation to one or more internal driver, the libvirt public API reveals the ability to support the new hypervisor.

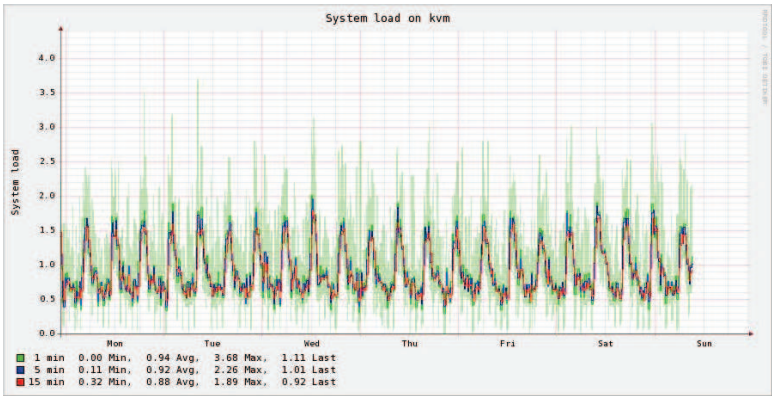


Figure 6: The system load collected from the last one week

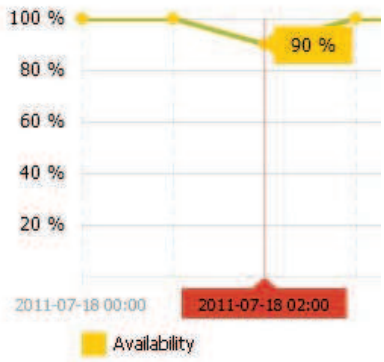


Figure 7: The availability curve graph of a virtual machine in hybrid Clouds

5.2 Service availability monitoring module

We suppose the Cloud service's availability events have a sequence of life times T_0, T_1, T_2, ..., and the Cloud service's unavailability events have a sequence of down times D_0, D_1, D_2, ... We refer to the probability that a lifetime t is shorter than units of time, $F(t) = P(T_i \leq t)$ as the time to failure distribution function; we also refer to $F(t)$ as the availability duration distribution. Then, the mean time to failure is $MTTF = \int_0^\infty (1 - F(t))dt$ from the start of an availability event. Under similar assumptions, the time to repair distribution function is $R(t) = P(D_i \leq t)$, and the mean time to repair is $MTTR = \int_0^\infty (1 - R(t))dt$. We also refer to $R(t)$ as the unavailability duration distribution. A service's average availability A, (or average unavailability U) is the fraction of time when a service is available (or unavailable) to an average Cloud client.

Availability monitoring module can refer to the process of collecting data and reporting on the status of the critical website. In other words, it means the availability for the WWW service in the Cloud. In order to get the accurate availability, the probe points are distributed around the world. First, we deploy some probe points in the backbone network of the popular Chinese Internet Service Providers (ISPs): China Unicom (CNC), China Telecom (CTC), and China CERNET. Second, we deploy some probe points in the Cloud host (in USA, UK, Australia, etc.) that is purchased from the Amazon Elastic Compute Cloud (EC2) service. We use tcping [Fulkerson 2013] command to determine the connectivity over TCP. Applying this monitoring module, the detail of the availability curve graph of a virtual machine in hybrid Cloud is shown in Figure 7. The notification system notified administrators of target availability status changes according the incident rulesets.

5.3 Performance monitoring module via SNMP

Performance monitoring can be divided into two main categories: computation-based and network-based. Computation-based tests are related to the following metrics: service, CPU speed, CPU utilization, memory usage, disk usage, etc.. Network-based tests are related to the following metrics: network throughput, packet loss, bandwidth, etc. We implement the manager-agent monitoring module to measure these metrics using Simple Network Management Protocol (SNMP). Furthermore, we present the statistics graphing of historical reference of these metrics in the browser. This module will appear an alert when the thresholds of the monitored objects were exceeded. Services and processes running on devices can be monitored to verify whether they are running or not. An example of the memory and disk monitoring in hybrid Cloud is shown in Figure 8.

In our solution, we use Java SNMP package [Sevy 2013] API to implement the SNMP monitoring manager, which is responsible to communicate with the SNMP agent. We take advantage of SNMP v2c API to get responses from agents. The JAVA method *snmpget* is using SNMP GET request to query for information on a network entity. An ip address, object identifier (OID), community string and SNMP version may be given as arguments of the method *snmpget*.

```
public String snmpget(String ip, String oid, String community
    , int version) {
  InetAddress hostAddress = InetAddress.getByName(ip);
  SNMPv1CommunicationInterface comInterface = new
      SNMPv1CommunicationInterface(version, hostAddress,
      community);
  comInterface.setSocketTimeout(20000);
  SNMPVarBindList newVars = comInterface.getMIBEntry(oid);
  SNMPSequence pair = (SNMPSequence) (newVars.getSNMPObjectAt
      (0));
  SNMPObjectIdentifier snmpOID = (SNMPObjectIdentifier) pair.
      getSNMPObjectAt(0);
```

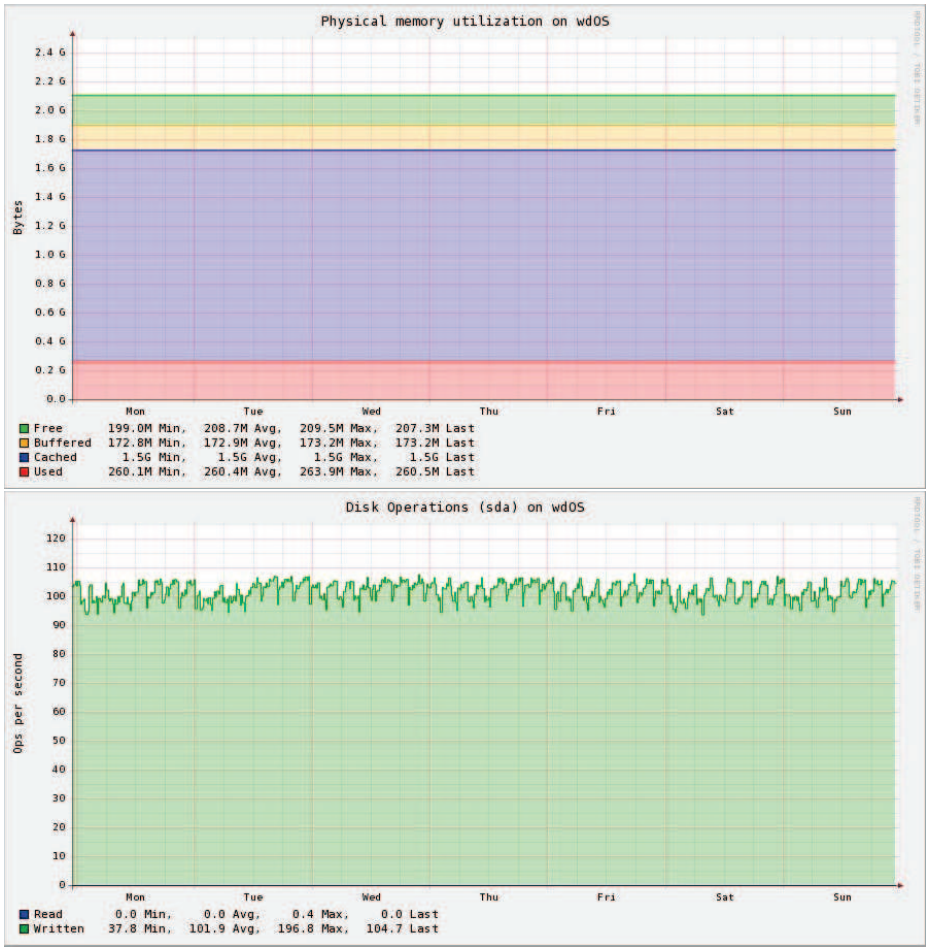


Figure 8: An example of the memory and disk monitoring in hybrid Clouds

```
  SNMPObject snmpValue = pair.getSNMPObjectAt(1);
  return snmpValue.toString();
}
```

In our solution, we deploy the open-source SNMP agent in the physical machine, the hypervisor and the virtual machine to collect the monitoring data. The extensible agent for responding to SNMP queries to management information includes built-in support for a wide range of the Management Information Base (MIB) information modules, and can be extended using dynamically loaded modules, external scripts and commands.

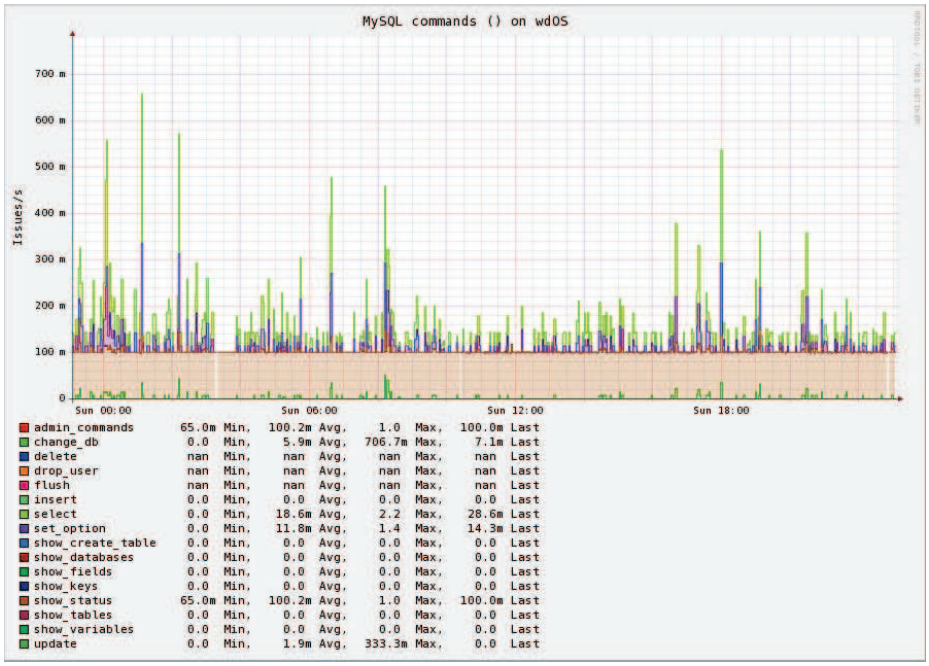
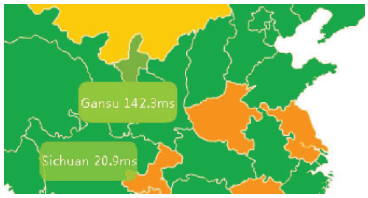


Figure 9: The number of issues per second for various SQL-commands of MySQL

5.4 Application monitoring module

Application monitoring module provides the most commonly monitoring in the software layer, including the Web server and database server. The monitored application often provides some public APIs. Regarding different Web servers, there are different monitoring methods, depending on the different APIs. Here are two examples of Web server monitoring. We use *mod_status* module of the Apache Web server to to keep watch on the operation of the server serves. The details given are: the number of worker serving requests and idle worker, the status of each worker, a total number of accesses and byte count served, averages giving the number of requests per second, the current percentage CPU used by each worker, etc. We provide *stub_status* module to get some status from Nginx Web server. The active connections and handled requests are calculated by this API. Here is an example of database server monitoring. We use Open Replicator API to capture the statement-based log events as they happen. We can intercept the database request (alter, select, update, insert and delete) to be analyzed. The number of issues per second for various SQL-commands are shown in Figure 9. The Java method *capture* is shown as follows, which is the implementation of



Province	Average response time (ms)
Ningxia	120.1
Gansu	142.3
Shanxi	50.5
Sichuan	20.9
Sinkiang	42.3
Tibet	52.1

Figure 10: An example of user experience tracker

interface *Capture*. We use Java method *selectMonitoring*, *insertMonitoring*, *updateMonitoring* and *deleteMonitoring* to record the performance of select, insert, update and delete respectively.

```
public capture(){
final OpenReplicator or=new OpenReplicator();
or.setBinlogEventListener(new BinlogEventListener() {
  String sql = ((String) event).getSql();
  if(event instanceof QueryEvent){
    selectMonitoring();
  }
  if (event instanceof WriteRowsEvent) {
    insertMonitoring();
  }
  if (event instanceof UpdateRowsEvent) {
    updateMonitoring();
  }
  if (event instanceof DeleteRowsEvent) {
    deleteMonitoring();
  }
}
});
or.start();
}
```

5.5 User experience tracker

Compared with the availability monitoring, user experience tracker provides a more accurate method to calculate the access speed and position real experience of each visitor. This module embeds a JavaScript in the webpage of the monitored website. The website in Clouds acts as the tenant, and the visitor of the website acts as the end user. The user access information (IP and location) is automatically recorded after the load of the webpage. For example, a multi-tenant social website is served for many cities. Although the visitors in different cities access the different entrances of the website, they actually visit the same website. Figure 10 shows the average response time of visitors of the website, which reflects the quality of our website. As depicted in Figure 10, the average response

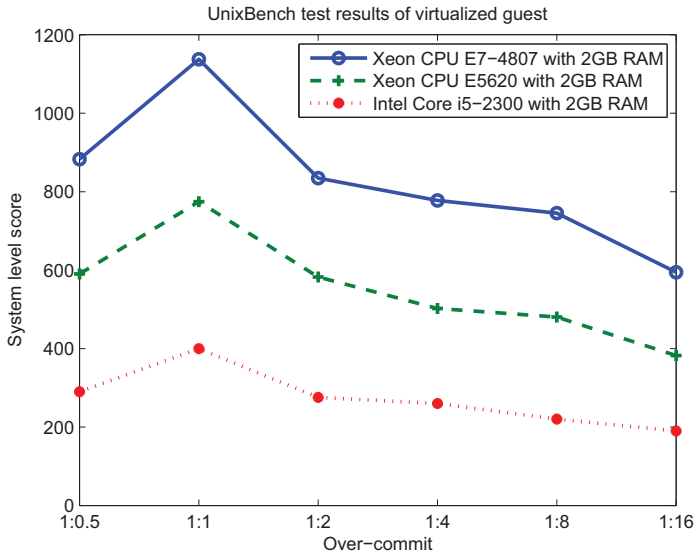


Figure 11: UnixBench test results of virtualized guest

time of NingXia is the longest, while the response time of Tibet is the shortest. We can conclude that the visitors in NingXia have poor user experience, and the visitors in Tibet have a better user experience. That is the guidance for the Cloud owners to optimize the network. It plays an important part in discovering the relationship between end user habits and Cloud performances. Furthermore, the metrics from the tracker is a indirect guidance on how to attract more users in the hybrid Clouds.

5.6 Over-commit monitoring

In the area of IaaS, over-commit refers to the practice of committing more virtual resources to customers than the actual resources available on the underlying physical cluster. Most of the prevailing hypervisors, such as Hyper-V, VMWare ESX, KVM, and XEN, support both CPU and memory over-commit. When an IaaS service provider practices over-commit, the over-commit parameters are usually unknown to the end user. When an end user creates a VM that is labeled as 1 CPU core and 1 GB memory, the CPU usage limit for that particular VM might be 1, 0.5 or even 0.1 physical processor core. Similarly, a host server with 16 GB physical memory may be committing 24GB or even 32 GB virtual memory to virtual machines. Therefore, we provide over-commit monitoring with the Cloud tenants, which help the tenants find out the actual performance of the

virtual machines. We have built in a local interface in the guest virtual machines that is served for Cloud tenants. This interface is implemented in open-source benchmark suite UnixBench [Niemi 2013], which provide a basic indicator of the performance of a Unix-like system.

The experiments show that the system scores of UnixBench rather than the parameters of CPU core, and the amount of memory indicate the actual performance of the virtual machine in the same hypervisor. Figure 11 shows the UnixBench test results of virtualized guest. For virtual machines in the same hypervisor, as the virtual machine gets bigger, the performance gets better in the none over-commit circumstance; for virtual machines from the same hypervisor, as the virtual machine gets bigger, the performance gets worse in the over-commit circumstance.

6 Implementation of aspect-oriented Cloud service monitoring

The aspect-oriented Cloud service monitoring has been implemented within Axis2. The implementations include the client and server of AOP service monitoring.

6.1 Implementation of the monitoring client of AOP service

```
@Aspect
public class MonitorClient {
  @After("execution(* org.apache.axis2.engine.AxisEngine.
      invoke(..)")
  public void advice1(){
    t1 =System.nanoTime();
    AllRequests++;
  }
  @After("execution(* org.apache.axis2.engine.AxisEngine.
      flowComplete(..)")
  public void advice4(){
    t4 = System.nanoTime();
    tresponse = t4 - t1;
    SuccRequests++;
  }
}
```

The first component of our AOP monitoring approach is the monitoring client, which is the aspect-oriented advice 1 and 4 that intercepts the client at t_1 and t_4. Its implementation is based on the identification of the methods that the service engine invokes at $t1$ and $t4$. In Axis2, the method invoked at t1 is voke of the AxisEngine class located in org.apache.axis2.engine package (see Listing 6.1). When the client sends a request, the Axis engine on the client side is invoked

via the method invoke(...). At the instant t4, the method flowComplete of the AxisEngine class located in org.apache.axis2.engine package.

6.2 Implementation of the monitoring server of AOP service

```
@Aspect
public class MonitorServer {
  @After("execution(* org.apache.axis2.engine.AxisEngine.
      invoke(..)")
  public void advice2(){
    t2 =System.nanoTime();
  }
  @After("execution(* org.apache.axis2.engine.AxisEngine.
      flowComplete(..)")
  public void advice3(){
    t3 = System.nanoTime();
    texec = t3 - t2;
  }
}
```

The second component of our AOP monitoring approach is the monitoring server. It is the aspect-oriented advice 2 and 3 (see Listing 6.2) that calculates the execution time. To implement the monitoring server using Axis2, it is necessary to identify the pointcut 2 and pointcut 3. Pointcut 2 corresponds to the instant t_2 when the request arrives at the server side, and pointcut 3 describes the instant t_3 when the response leaves the server side. The execution of the method *invoke*(...) of the class *AxisEngine*located in the *org.apache.axis*2.*engine* package is in charge of the request processing at the server side. This means that recording the instants before and after the execution of this method lead to the computation of the execution time value (see Listing 6.2). Thus, pointcut 2 and 3 correspond to the interception of the execution of this method, and the related advices should be applied before and after this method, respectively (see Listing 6.2).

AOP service monitoring also handles multiple clients, which is able to distinguish between clients by using their IP addresses. Furthermore, AOP service monitoring uses the request session ID to differentiate between concurrent requests running on the same client. Therefore, it associates monitored QoS parameters to the corresponding client. Based on AOP, the monitoring server extracts the client and the server IP addresses corresponding to the collected monitored QoS parameters. Furthermore, the monitoring client distinguishes between the concurrent requests running on the same client, while extracting their session IDs. It should be pointed out that the AOP monitoring components are completely independent of the original server. In fact, the developed aspect codes are not located inside the server source code. With the help of the weaving mechanism of AOP, they intercept the methods at defined join points and record the relevant timestamp information without modifying the source code of the monitored service.

7 Conclusion and Future Work

Nowadays, monitoring plays an important part in hybrid Clouds. However, current monitoring solutions on the popular metrics of the hypervisors, the virtual machines, and the physical machines in hybrid Clouds are minimal at best. Towards our goal of building such a lightweight and scalable framework, we integrate some open-source monitoring tools with some extra significant secondary development work to implement some modules. First of all, we introduce the monitoring architecture and hierarchy of resource entity models. User experience is pointed out as a reference to discover the relationship between users' habits and Cloud performance. In addition, we discuss the manager-agent and aspect-oriented architecture to perform end-to-end measurements at both virtual and physical machines and software in hybrid Clouds. By default, we provide the basic monitoring of common metrics. Each monitoring is implemented in the forms of plug-ins. Aspect-oriented Cloud service monitoring allows the developer to dynamically write references to aspects at join points to calculate the performance of Cloud services. Thus, they eliminate many lines of scattered code. Finally, the implementations of this framework and some experiments have demonstrated the high performance of the proposed framework.

Future work is targeted in two directions to complete and improve the current proposal. The first target is to define a general QoS equation that combines all QoS parameters to describe the state of the service. This equation could then be integrated into the policies of the fault tolerance framework to further improve the recovery results. Furthermore, we plan to provide a visual monitoring platform, which is integrated with the current virtualization management system for Cloud computing.

Acknowledgment

This work was supported by the Doctoral Fund of University of Jinan (XBS1237), the Youthful Science and Technology Star Plan of Jinan (20110112), the Technology development Program of Shandong Province (2011GGX10116), and the National Key Technology R&D Program (2012BAF12B07).

References

[Aceto et al. 2013] Aceto, G., Botta, A., Donato, W. , and Pescape, A.: "Cloud monitoring: A survey"; Computer Networks, online (2013).

[Cao et al. 2009] Cao, B., Li, B., Xia, Q.: "A Service-Oriented Qos-Assured and Multi-Agent Cloud Computing Architecture"; CloudCom'09, Springer-Verlag, Heidelberg (2009), 644-649.

[Chaves et al. 2011] Chaves, S. A., Uriarte, R. B., and Westphall, C. B.: "Toward an architecture for monitoring private Clouds"; IEEE Communications Magazine, 49, 12 (2011), 130-137.

[Ciuffoletti et al. 2010] Ciuffoletti, A. : "Monitoring a virtual network infrastructure: An Iaas Perspective"; ACM SIGCOMM Computer Communication Review, 40, 5 (2010), 47-52.

[CloudWatch 2013] Amazon Web Services, Inc.: "Amazon CloudWatch"; http://aws.amazon.com/es/cloudwatch (Accessed 20 July 2013).

[Dastjerdi et al. 2012] Dastjerdi, A. V., Tabatabaei, S. G. H., and Buyya, R.: "A dependency-aware ontology-based approach for deploying service level agreement monitoring services in Cloud"; Software: Practice and Experience, 42, 4 (2012), 501-518.

[Dhingra et al. 2012] Dhingra, M., Lakshmi, J., and Nandy, S. K.: "Resource Usage Monitoring in Clouds"; Proc. of GRID'12, IEEE Computer Society, New York (2012), 184-191.

[Dixon 2012] Dixon, J.: "The state of open source monitoring: The good, the bad, the fucking terrible, and a glimpse into our future"; Github, Tech. Rep. (2012).

[Elrad et al. 2001] Elrad, T., Filman, R. E., and Bader, A.: "Aspect-oriented programming: Introduction"; Communications of the ACM, 44, 10 (2001), 29-32.

[Forster and Harl 2013] Forster, F. and Harl, S.: "collectd C The system statistics collection daemon"; http://collectd.org (Accessed 20 July 2013).

[Fulkerson 2013] Fulkerson, E. : "Tcpping - ping over a tcp connection"; http://www.elifulkerson.com/projects/tcping.php (Accessed 20 July 2013).

[Imamagic and Dobrenic 2007] Imamagic, E. and Dobrenic, D.: "Grid infrastructure monitoring system based on nagios"; Proc. of HPDC'07, IEEE Computer Society, New York (2007), 23-28.

[Han et al. 2011] Han, F., Peng, J., Zhang, W., Li, Q., Li, J., and Jiang, Q.: "Virtual resource monitoring in Cloud computing"; Journal of Shanghai University (English Edition), 15, 5 (2011), 381-385.

[Issariyapat et al. 2012] Issariyapat, C., Pongpaibool, P., Mongkolluksame, S., and Meesublak, K.: "Using Nagios as a groundwork for developing a better network monitoring system"; Proc. of PICMET'12, IEEE Computer Society, New York (2012), 2771-2777.

[Kung et al. 2011] Kung, H., Lin, C. K., and Vlah, D.: "Cloudsense: Continuous fine-grain cloud monitoring with compressive sensing"; Proc. of HotCloud'11, ACM , New York (2011), 1-6.

[Ma et al. 2011] Ma, K., Yang, B., Chen, Z, Ma, B., and Li, Q.: "From the modern Web applications to the multi-tenant SaaS solution"; Chinese Journal on Communications, 32, 9A (2011), 133-138.

[Ma et al. 2012a] Ma, K., Yang, B., and Abraham, A.: "A Template-based Model Transformation Approach for deriving multi-tenant SaaS applications"; Acta Polytechnica Hungarica, 9, 0 (2012), 25-41.

[Ma et al. 2012b] Ma, K., and Fang, C.: "A Security Extension Framework Based on SOAP Header"; Journal of Information and Computational Science, 9, 17 (2012), 5249-5256.

[Ma et al. 2012c] Ma, K., Sun, R., and Abraham, A.: "Toward a lightweight framework for monitoring public Clouds"; Proc. of CASoN'12, IEEE Computer Society, New York (2012), 361-365.

[Massiea et al. 2004] Massiea, M. L., Chunb, B. N., and Cullera, D. E.: "The ganglia distributed monitoring system: Design, implementation, and experience"; Parallel Computing, 30, 7 (2004), 817-840.

[Messina et al. 2012] Messina, F., Pappalardo, G., Rosaci, D., Santoro, C., and Sarn, G. M. L.: "A Trust-Based Approach for a Competitive Cloud/Grid Computing Scenario"; Proc. of IDC'12, Springer-Verlag, Heidelberg (2012), 129-138.

[Mell and Grance 2011] Mell, P. and Grance, T.: "The NIST definition of Cloud computing"; NIST Special Publication 800-145, Timothy, Grance (2011).

[Montes et al. 2013] Montes, J., Snchez, A., Memishi, B., Perez, M. S., and Antoniu, G.: "GMonE: A complete approach to cloud monitoring"; Future Generation

Computer Systems, online (2013).
[Naldi 2013] Naldi, M.: "The availability of cloud-based services: Is it living up to its promise?"; Proc. of DRCN'13, IEEE Computer Society, New York (2013), 282-289.
[Niemi 2013] Niemi, D. C. : "UnixBench"; `http://code.google.com/p/byte-unixbench` (Accessed 20 July 2013).
[Nurmi et al. 2009] Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., and Zagorodnov, D.: "The eucalyptus open-source cloud-computing system"; Proc. of CCGRID'09, IEEE Computer Society, New York (2009), 124-131.
[OpenNebula 2013] OpenNebula Project: "OpenNebula: the open source toolkit for cloud computing"; `http://www.opennebula.org` (Accessed 20 July 2013).
[Peng et al. 2009] Peng, J., Zhang, X., Lei, Z., Zhang, B., Zhang, W., and Li, Q.: "Comparison of several Cloud computing platforms"; Proc. of ISISE'09, IEEE Computer Society, New York (2009), 23-27.
[Jamsa 2012] Jamsa, K.: "Cloud Computing"; Jones and Bartlett Learning, Massachusetts (2012).
[Russell and Cohn 2012] Russell, J. and Cohn, R.: "RRDtool"; Book on Demand Ltd., California (2012).
[Sevy 2013] Sevy, J.: "Java SNMP Package"; `http://gicl.cs.drexel.edu/people/sevy/snmp` (Accessed 20 July 2013).
[Shao et al. 2010] Shao, J., Wei, H., Wang, Q., and Mei, H.: "A runtime model based monitoring approach for Cloud"; Proc. of CLOUD'10, IEEE press, New York (2010), 313-320.
[Sunyaev and Schneider. 2013] Sunyaev, A., and Schneider, S.: "Cloud Services Certification"; Communications of the ACM, 56, 2 (2013), 33-36.
[Tselentis et al. 2010] Tselentis, G., Galis, A., Gavras, A., Krco, S., Lotz, V., Simperl, E., Stiller, B., and T. Zahariadis: "Monitoring service Clouds in the future internet"; Towards the Future Internet - Emerging Trends from European Research, IOS press, Netherlands (2010), 115-126.
[Van et al. 2009] Van, H. N., Tran, F. D., and Menaud, J. M.: "Autonomic virtual resource management for service hosting platforms"; Proc. of ICSE'09, IEEE Computer Society, New York (2009), 1-8.