2010 IEEE 3rd International Conference on Cloud Computing

A Runtime Model Based Monitoring Approach for Cloud

Jin Shao, Hao Wei, Qianxiang Wang, Hong Mei
School of Electronics Engineering and Computer Science, Peking University
Beijing, China
{*shaojin07,weihao09,wqx,meih*}*@sei.pku.edu.cn*

***Abstract*—Monitoring plays a significant role in improving the quality of service in cloud computing. It helps clouds to scale resource utilization adaptively, to identify defects in services for service developers, and to discover usage patterns of numerous end users. However, due to the heterogeneity of components in clouds and the complexity arising from the wealth of runtime information, monitoring in clouds faces many new challenges. In this paper, we 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. We applied RMCM in the implementation of a flexible monitoring framework, which can achieve a balance between runtime overhead and monitoring capability via adaptive management of monitoring facilities. Our experience of utilizing the monitoring framework on a real cloud demonstrates the feasibility and effectiveness of our approach.**

***Keywords*-runtime model, monitoring, cloud**

I. Introduction

As an effective approach to improve the quality of software products, monitoring has been widely used for software optimization, profiling, performance evaluation, etc. [1] As the form of software evolves to "SaaS", the demand for monitoring grows and brings forth new targets, methodologies and techniques, as reported in many research efforts concerned with monitoring of web services or web service compositions [2]–[4].

Such a demand for monitoring also needs to be satisfied in cloud computing. As an emerging service consumption and delivery model in the filed of service computing, cloud computing promised to deliver on-demand IT resources by dynamically scaling its service provision. Such an elastic nature of cloud computing results in a strong demand for monitoring. However, monitoring the cloud at runtime is very challenging. First, the large number of services and end users in clouds leads to wealth of raw runtime information, which is hard to comprehend for administrators, much less to process them automatically for adaptive scaling and evolution. Second, due to the heterogeneity of components in clouds, individual monitoring schemes and mechanisms need to be designed and implemented respectively, which is very costly. Third, much more monitoring concerns need to be covered in clouds than traditional software, and therefore runtime overhead resulted by cloud monitoring is sometimes unacceptable, which a main obstacle that impedes the application of monitoring in clouds. These challenges lead to the demand for a more intuitive and flexible way to implement monitoring in cloud.

Model based approaches are promising to face the first two challenges. A model is an abstraction or reduced representation of a software system for specific purposes [5]. It can provide a simple and easy-to-understand view of a complicated software system from a specific perspective. As the core of model-driven engineering (MDE) approaches, models have been applied widely in the design and implementation phrases of software development process nowadays. To extend the applicability of models produced in MDE approaches to the runtime environment, runtime models that concentrate on modeling the runtime behavior of software systems and their environments have been investigated by many researchers [5]–[7]. Inspired by these research efforts, we propose a runtime model for cloud monitoring (RMCM). RMCM seeks to abstract and organize the wealth of runtime information yielded in a cloud, and then provides an intuitive representation of the 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 the running cloud.

To face the third challenge discussed above, we implement a flexible monitoring framework based on RMCM. By exploiting the new instrumentation feature of JAVA 6, the framework is able to achieve a balance between runtime overhead and monitoring capability via adaptive management of monitoring facilities.

This paper makes the following main contributions:

- We propose a runtime model for cloud monitoring (RMCM), which provides an intuitive representation of a running cloud by focusing on common monitoring concerns.
- Based on RMCM, we implement a flexible and efficient monitoring framework, which is able to achieve a balance between runtime overhead and monitoring capability via adaptive management of monitoring facilities.
- We apply the runtime model based monitoring approach in a real cloud.

978-0-7695-4130-3/10 $26.00 © 2010 IEEE
DOI 10.1109/CLOUD.2010.31

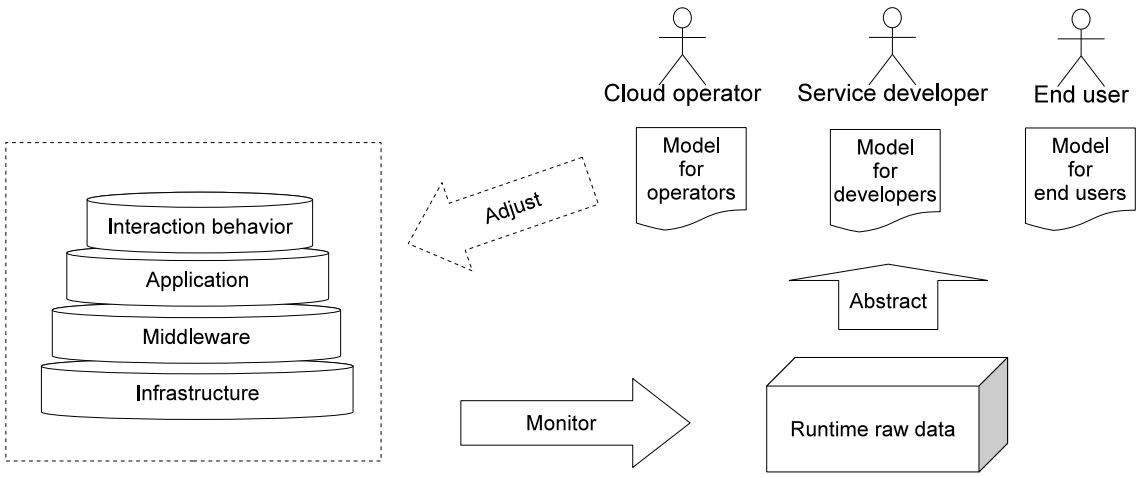

Figure 1. Modeling process in RMCM

The remainder of this paper is organized as follows. Section 2 describes the RMCM we proposed. Section 3 shows the flexible monitoring framework we implement and monitoring techniques we used. Section 4 illustrated the application of our approach in a real cloud. Section 5 presents some related work. Section 6 concludes the paper and discusses some open issues in our future work.

II. Runtime Model for Cloud Monitoring (RMCM)

RMCM is an intuitive representation of a running cloud based on common monitoring concerns. It abstracts the runtime information collected during the execution of a cloud, and then presents them in an understandable and operable way. Such an abstraction can hide the heterogeneity of underlying infrastructures and platforms, and render the system at a higher level. Additionally, the runtime model is linked in such a way that it constantly mirror the system and its current state and behavior [5]. Consequently, the model can serve as a bridge between the running cloud and human roles that interact with it. Any changes in the real running system will be reflected in the model immediately, and therefore human roles can inspect runtime status of the cloud via the model, instead of investigating disordered runtime data. When problems arise, human roles perform adaptive actions against the model, rather than against the running cloud directly, so as to avoid low-level operation mistakes.

Different human roles need individual views of RMCM in clouds. Commonly, there are three types of roles in the cloud: cloud operators, service providers, and end users. The cloud operator is who owns IT computing resources and delivers them to service developers. The cloud operator needs to inspect the current running status of the cloud and its environment, so as to acquire indications for when to expand or shrink its service provision. Additionally, the cloud operator needs to keep an eye on malicious users by identifying anomalous usage behaviors. By purchasing service provided by cloud operators, service developers do not have to worry about how many hardware should be bought any more, and can focus on developing their applications. Besides, they expect to discover usage habits of numerous end users brought by the cloud via monitoring, so as to produce valuable suggestions for ongoing evolution of their applications and targeted recommendations to improve user experience. End users are who use the service provided by clouds directly to support their business. They can also choose several single services and then assemble them into a business process to achieve complex business goals. Monitoring can provide end users with evidence about which service is the most appropriate one among various candidates. To sum up, all roles in the cloud can all benefit from monitoring.

A running cloud is presented with runtime models from multiple views based on various monitoring requirement from different roles. Each type of human roles focuses on their own view. The view shown to cloud operators provides an integral picture of the running cloud, which concentrates on resource utilization and allocation. The view shown to service developers concentrates on the status of their own application, with which to check whether the cloud has supplied enough resource as promised. The view shown to end users focuses on comparison among multiple services. Integration of these views provide a consistent and comprehensive runtime model for monitoring of a cloud system, as shown in Figure 1.

A. Categories of Entities to be Modeled

Entities need to be modeled in a running cloud are divided into four categories, as shown in the left of Figure 1.

At the bottom is the supporting infrastructure, such as operating systems on both physical machines and virtual ma-

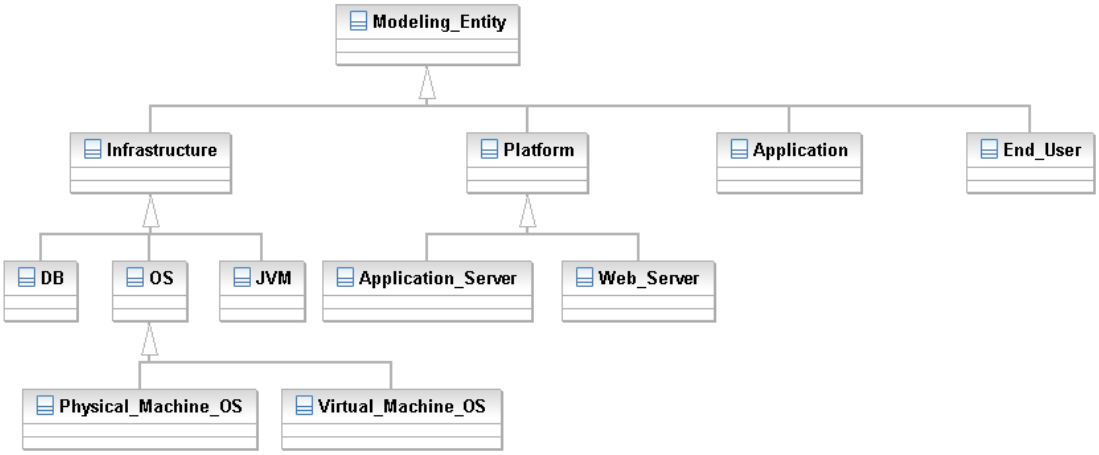

Figure 2. Hierarchy of entities to be modeled in RMCM

chines, databases and Java virtual machines. The operational state of the infrastructure can directly affect the quality of service, so monitoring this part of cloud is very essential. The infrastructure monitoring mainly focuses on the current resource utilization status reflected by performance metrics, such as CPU utilization, memory utilization, and so on. These performance metrics, which stay the same regardless of distinguished infrastructures, are extracted to model the runtime infrastructure in cloud.

On top of the infrastructure is middleware (such as application servers, web servers, and so on) that provides an appropriate run time environment for applications. These middleware provide a wide set of services that assist service developers in delivering a professional and commercial service to end users. These services include: negotiation of the quality of service, admission control, execution management, and so on [8]. Commonly, these middleware will expose their own interface for monitoring and management, and these interfaces are gathered and organized to form the runtime monitoring model of this layer.

Since applications are designed and implemented by variety of service developers, the application monitoring is more complicated than the two entities discussed earlier, and additional information needs to be included in the runtime monitoring model. Applications in the cloud are monitored from two perspectives. First, they are monitored against their design and implementation model, to validate whether the application satisfies its requirement. Second, they are monitored against some rules and constraints predefined by the cloud operator, to prevent malicious, defective or inefficient code from degrading the performance of the cloud. To address the challenges in application monitoring, we proposed an application runtime monitoring model that combines design models and constraints in our earlier work [4]. With the structure information that design model provides, constraints are converted to corresponding instrumented code and then deployed at appropriate locations in the application.

The interaction behavior between various roles and clouds is also an important entity to monitor. The managing actions of cloud operators are monitored against consequent running status of the cloud, in order to discover the relationship between configuration parameters and cloud performance. This can be used to provide guidance for future management of the cloud. Information related to the interaction between end users and clouds, such as access frequency, IP distribution, time of staying and so on, are monitored to suggest usage habits which are concerned by service developers. The service developer can adapt and improve their applications based on the monitored usage habits, so as to attract more users. These usage behavior is also valuable to cloud operators. For example, cloud operators can monitor the ups and downs in the number of users and the locations of users to make a better plan for resource allocation.

B. Hierarchy of Entities in RMCM

Concrete entities in a running cloud are organized in a hierarchy in RMCM, as shown in Figure 2. This hierarchy can be extended by adding new entities emerged in different clouds. Such an organization makes extension on the model more feasible. New entities can be added in directly by inheriting one of the existing entity. Each entity consists of a set of attribute-value pairs. Children entities can inherit attributes from their parents. And values are obtained from real-time monitoring statistics.

Different compositions of these entities consist of a specific view of monitoring model for corresponding human roles. Access to these entities are controlled based on

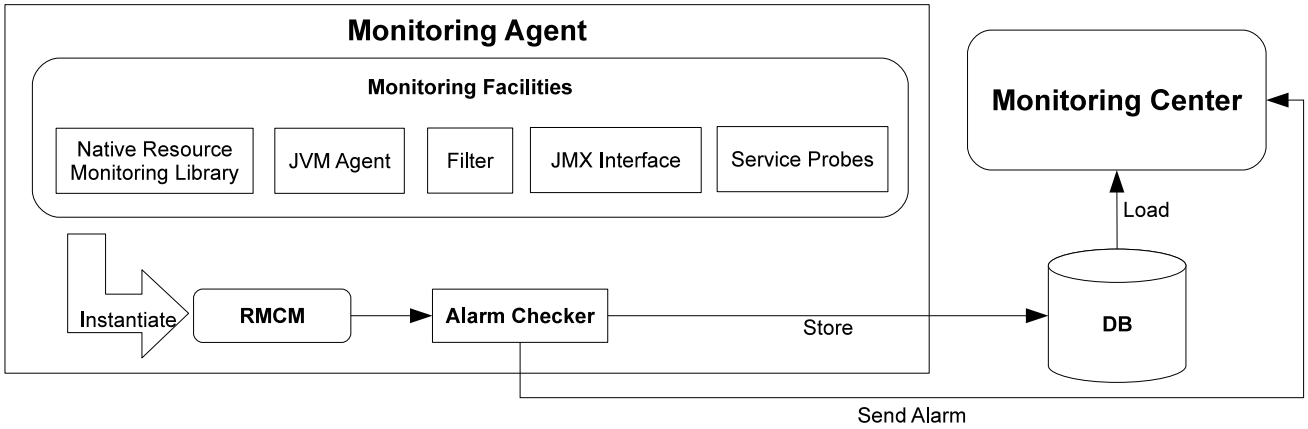

Figure 3. Implementation of the monitoring framework

authority of each types of roles for security considerations. As a result, different roles can only focus on monitoring information they care without confused by complex numbers. Also, the cloud can be protected from over-authorized malicious users.

III. Implementation of Monitoring Framework

We next describe the cloud monitoring framework we implemented based on RMCM. We first introduce the architecture of our cloud monitoring framework, and then discuss about used monitoring mechanisms and balance strategy between runtime overhead and monitoring capability.

A. Server-Agent Architecture

In a cloud, multiple entities need to be monitored simultaneously, so as to coordinate their resource utilization to achieve a balance. Therefore, the monitoring framework is required to monitor individually and make decisions centralized. We implement our monitoring framework in a server-agent style architecture, as shown in Figure 3. In our monitoring framework, a monitoring agent is deployed on each virtual machine. These agents is in charge of collecting runtime information of all the entities on the same virtual machine. Different types of monitoring facilities are contained in a monitoring agent to collect runtime information from entities of each level. Collected raw runtime information is then used to instantiated corresponding RMCM. These RMCMs will be checked against some pre-defined rules, to determined whether alarms should be sent to the monitoring center. If the running status represented by the RMCM is acceptable, they will be stored in the DB. Administrators can view and query these monitoring information from the monitoring center later on. Also administrators can modify the monitoring configuration for each monitoring agent.

B. Monitoring Mechanisms

Due to the diversity of entities in RMCM, multiple monitoring techniques are employed in our monitoring framework to implement different monitoring facilities. We next discuss where and how these techniques are applied.

Resource Monitoring Native Library The infrastructure and platform in the cloud is monitored with some native libraries. In our implementation, we use an open source API called SIGAR [9]. Hyperic's System Information Gatherer and Reporter (SIGAR) is a cross-platform API for collecting software inventory data. The core API is implemented in pure C, and provides bindings currently implemented for Java. With this native library, runtime information about the infrastructure can be obtained in real time, regardless of their architectures or platforms.

JVM Agent Current services on our platform are all Java based and run on a JVM. The running status of JVM is critical to the quality of service. To monitor JVMs, we implemented a JVM agent using JVM tool interface (JVM TI) [10]. A JVM agent is written in C and loaded before the JVM starts. It can observe some key events during the execution of JVM and report resource utilization status of each Java class.

Filter Filters, also called interceptors, can obtain details of the messages to and from the monitored service. They are used to monitor the interacting behavior between end users and services, so as to discover usage habits from thousands of requests and responses. Filters in Tomcat, handlers in AXIS are all well-known interceptors that can do some processing to messages. The most important feature of interceptors is that it is independent of the target system both at coding time and runtime.

JMX Interface Java Management Extensions (JMX) is a standard that provides the tools for building distributed, web-based, modular and dynamic solutions for managing and monitoring devices, applications, and service-driven networks [11]. It has been implemented by many kinds of middleware, such as Tomcat, JBoss. It is convenient to obtain monitoring and management information about these middleware by calling the interface provided by JMX.

Service Probe Instrumentation is the most widely used

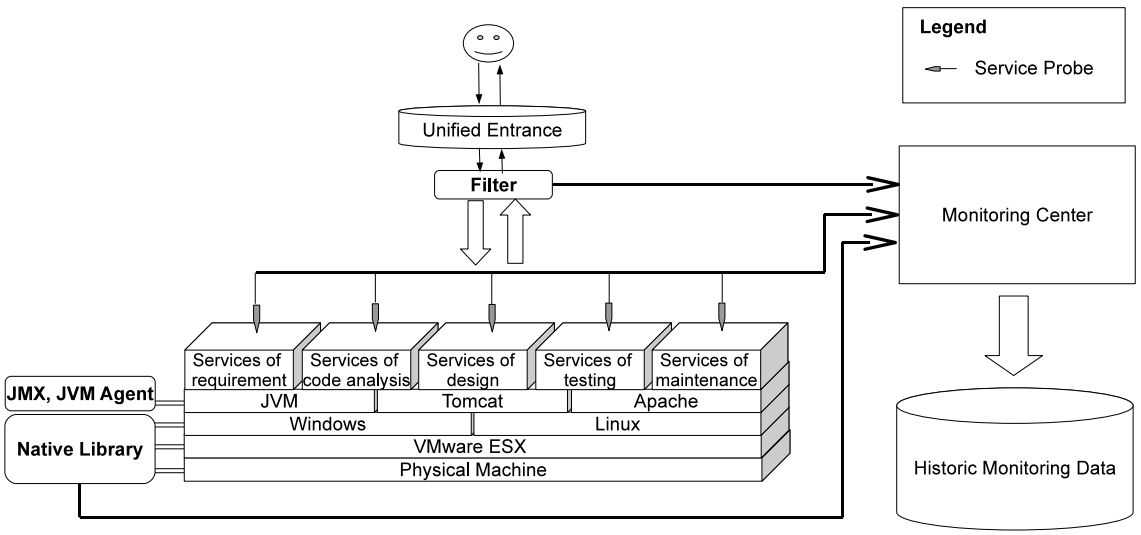

Figure 4. Monitoring framework for a real cloud SESP

monitoring mechanism. It is also widely used in program testing. In this approach, the monitoring code is embedded inside the target code. Traditionally, the instrumentation code is inserted manually by the programmers. The most important feature of the instrumentation approach is that: the code can be inserted freely into any location of the monitored code. Another reason for using instrumentation is that it does not need support from the platform. Some instrumentation tools (such as Javaassit and AspectJ) are developed recently, so as to instrument code automatically and conveniently according to configuration information.

C. Balance between Runtime Overhead and Monitoring Capability

While providing insurance to the high-quality of running clouds, equipping a running cloud with monitoring capabilities also brings about side effects. All the monitoring techniques discussed in Section 3.1 need consume certain computing resources to work. Consequently, it will lead to undesired runtime overhead to a running cloud. How to restrict such runtime overhead within an acceptable range is a challenging issue to investigate. On one hand, a running cloud requires enough monitoring ability to ensure its healthy operation. On the other hand, the stronger the monitoring ability is, the higher the runtime overhead is. Consequently, it is an important issue to balance the tradeoff between runtime overhead and monitoring ability. Such a balance should be made in real-time, but without interrupting the normal execution of a cloud.

In our cloud monitoring framework, we achieve such a balance by employing a new feature of instrumentation in JAVA 6 [12]. The new instrumentation feature enables "hotswap" of a JAVA class during the execution of JVM(i.e., to replace an existing class in the JVM with another version of it). Consequently, monitoring facilities (i.e., instrumented probes, interceptors, etc.) can be added or removed without interrupting the normal execution of the cloud. In the current implementation of our cloud monitoring framework, the controlling of monitoring facilities are performed manually to achieve balance between the cloud performance and the monitoring capabilities. In our future work, such a balance will be conducted automatically based on pre-defined policies.

IV. Case Study

We implemented our monitoring framework on a real cloud named SESP (more details about it can be accessed at http://sase.seforge.org), which stands for Software Engineering Service Platform. The vision of SESP is to collect plenty applications which are related to software developing and maintenance, and then provide them as services to software developers and researchers. As the cloud operator, we provide services to both end users and service developers. For the service developers, they are served with Platform as a Service (PaaS) by utilizing storage resources, computing resources and network bandwidth resources provided by SESP. For end users, they are served with Software as a Service (SaaS) by utilizing variety of tools deployed on SESP.

The architecture of SESP is shown in Figure 4, which can be seen as an instance of RMCM shown in Figure 1. At the bottom is a cluster of physical servers. VMware ESX is installed directly on top of each physical server and partition it into multiple virtual machines that can run simultaneously, sharing the physical resources of the underlying server. Then multiple operating systems are installed on these virtual machines. These three layers constitute the layer of infrastructure shown in Figure 1. Middleware, such

as application servers and web servers, are deployed on top of the infrastructure layer and provided as PaaS to service developers. Service developers deploy their applications on this layer, and then provide them as SaaS to the end users.

Adaptive resource provision is a key characteristic of SESP. Several different services are deployed on the same middleware, and copies of a single service are deployed on several middleware that distributed on different virtual machines. Instead of accessing these services directly, end users access these services through a unified entrance. This entrance also acts as a load balancer. It forwards requests to appropriate service node based on the information derived from the monitoring center, so as to achieve a balance between different virtual machines.

Besides the fundamental services, SESP provides a series of additional management services for its users. When service developers submit their services to SESP, the source code of these services is analyzed by both static and dynamic analysis techniques to detect possible defects. Only services that are confirmed without known defects can be deployed on SESP by cloud operators. Then end users can enjoy the services provided by SESP via visiting multiple types of software engineering services. The interaction between end users and SESP, together with runtime status of SESP are observed by the monitoring framework via variety of monitoring facilities at different layers, as shown Figure 4. Collected runtime data are recorded for further investigation. For example, they can be analyzed to discover usage habits of end users so as to provide targeted service recommendation, or to identify possible performance bottlenecks so as to optimize the service quality.

In the operating practice of SESP, the monitoring framework has played a key role in guaranteeing the service quality. Additionally, with the help of hotswap, the monitoring framework is flexible and can restrict the runtime overhead under control.

V. Related Work

A. Monitoring of Traditional Software

Software monitoring has been explored for a long time in various fields, such as performance evaluation and enhancement, fault tolerance, and autonomic computing [13]–[16]. Based on these efforts, Delgado et al. [1] proposed a taxonomy and catalog for runtime monitoring based on a comprehensive survey of the area, which analyzed and summarized research concerned with monitoring of traditional software. After that, a new branch in the runtime monitoring research field, which named runtime verification, is paid a lot of attention by researchers. Unlike earlier approaches that focus on performance of running software, runtime verification aims at checking whether a running software conforms to certain specifications extracted from requirements [17]–[19]. Runtime verification techniques are included in our runtime monitoring framework to support monitoring applications in the cloud.

B. Monitoring of Web Services

The specific characteristics of web service that differs a lot from traditional software (such as dynamism and loose coupling) have posed additional issues for investigation when implementing web service monitoring. Some of them focus on performance monitoring. Bai et al. [20] proposed a model-driven approach to facilitate automatic sensor generation and policy enforcement to support monitoring in web services. The sensors and policies are decoupled from the software and are defined at the abstraction model level, including structure and behavior models. Other researchers focus on functional requirement monitoring. Mahbub and Spanoudakis [21] introduced five different types of system behavior deviations from requirements. And their target of monitoring is the event stream obtained from the BPEL engine, instead of services. Another BPEL-oriented framework [22] emphasizes the ability of self-healing. It instruments the original BPEL specification to allow execution of the required rules. The instrumentation feeds the monitoring manager, a proxy service that is responsible for understanding the Web service. Barbon et al. [23] proposed an architecture that separates the business logic of a Web service from its monitoring functionality, and supports both “instance monitors” and “class monitors”. Li et al. [24] proposed a web service monitoring framework, which focuses on the monitoring of service interaction protocols.

C. Monitoring of Cloud

Well-known clouds in the industry all have their own monitoring utilities. Google App Engine [25] provides an App Engine System Status Dashboard to show the service developers how their applications work in the cloud. Some third-party tools are also developed to keep watch over the clouds, such as CloudStatus developed by Hyperic Inc. [26], which can both monitoring Amazon service and Google App Engine.

Research work concerned with monitoring in the cloud is relatively less. In [27], the authors proposed a RESTful approach to monitor and manage cloud infrastructures. Entities (including computing, network and storage resources) in the cloud are modeled with REST in a tree structure. Such an organization of runtime monitoring and management information is suitable for cloud infrastructure. However, other entities in the cloud, which are also in a strong demand for monitoring, can not be modeled appropriately.

VI. Conclusion and Future Work

The on-demand service provision style of cloud computing has contributed a lot to the recent upsurge in its popularity. Such an elastic nature leads to a strong demand for monitoring in clouds. However, cloud monitoring is an

essential but challenging work. In this paper, we 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 the running cloud. We applied RMCM in the implementation of a flexible monitoring framework, which can achieve a balance between runtime overhead and monitoring capability via adaptive management of monitoring facilities. Our experience of utilizing the monitoring framework on a real cloud demonstrates the feasibility and effectiveness of our approach.

Some open issues that will be addressed in our future work are also discussed here. Besides providing evidence for cloud operators and service developers to perform manual adjustment, runtime models for monitoring can also be exploited for adaptive adjustment. Non-functional requirements obtained from SLA can be added as constraints to RMCM, and corresponding actions that need to perform when problems occur are defined as policies in the monitoring center. During the execution of cloud, RMCM will be checked against to the predefined constraints. When violation is discovered, RMCM will inform the monitoring center, and the monitoring center will act according to corresponding policies to adjust the cloud by modifying certain configuration parameters. Data will be accumulated from the interaction between RMCM and adjusting actions. By continuously collecting these data and explore the relationship between them, valuable knowledge of configuration will be mined and regarded as guidance for next-step adaptive adjustment. Our future work includes mining the data gathered by monitoring the real execution of cloud to find some connection between different configurations and corresponding performance, and utilizing these knowledge to perform adaptive adjustment and management.

REFERENCES

[1] N. Delgado, A. Gates, and S. Roach, "A taxonomy and catalog of runtime software-fault monitoring tools," *Software Engineering, IEEE Transactions on*, vol. 30, no. 12, pp. 859 – 872, dec. 2004.

[2] K. Mahbub and G. Spanoudakis, "Run-time monitoring of requirements for systems composed of web-services: initial implementation and evaluation experience," in *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*, july 2005, pp. 257 – 265 vol.1.

[3] W. Robinson, "Monitoring web service requirements," in *Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International*, sept. 2003, pp. 65 – 74.

[4] Q. Wang, J. Shao, F. Deng, Y. Liu, M. Li, J. Han, and H. Mei, "An online monitoring approach for web service requirements," *Services Computing, IEEE Transactions on*, vol. 2, no. 4, pp. 338 –351, oct.-dec. 2009.

[5] G. Blair, N. Bencomo, and R. France, "Models@ run.time," *Computer*, vol. 42, no. 10, pp. 22 –27, oct. 2009.

[6] G. Huang, H. Mei, and Q. Wang, "Towards software architecture at runtime," *SIGSOFT Softw. Eng. Notes*, vol. 28, no. 2, p. 8, 2003.

[7] Q. Wang, G. Huang, J. Shen, H. Mei, and F. Yang, "Runtime software architecture based software online evolution," in *COMPSAC '03: Proceedings of the 27th Annual International Conference on Computer Software and Applications*. Washington, DC, USA: IEEE Computer Society, 2003, p. 230.

[8] D. Duncan, X. Chu, C. Vecchiola, and R. Buyya. (2009, 9) The structure of the new it frontier: Cloud computing part i. [Online]. Available: http://texdexter.wordpress.com/2009/09/09/the-structure-of-the-new-it-frontier-cloud-computing-part-i/

[9] System Information Gatherer and Reporter (SIGAR). Hyperic Inc. [Online]. Available: http://www.hyperic.com/products/sigar

[10] Java Virtual Machine Tool Interface (JVM TI). Oracle Corporation. [Online]. Available: http://java.sun.com/javase/6/docs/technotes/guides/jvmti/

[11] Java Management Extensions (JMX) Technology. Oracle Corporation. [Online]. Available: http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/

[12] JDK 6 Documentation. Oracle Corporation. [Online]. Available: http://java.sun.com/javase/6/docs/api/java/lang/instrument/package-summary.html

[13] M. Feather, S. Fickas, A. van Lamsweerde, and C. Ponsard, "Reconciling system requirements and runtime behavior," in *Software Specification and Design, 1998. Proceedings. Ninth International Workshop on*, apr 1998, pp. 50 –59.

[14] S. Fickas and M. Feather, "Requirements monitoring in dynamic environments," in *Requirements Engineering, 1995., Proceedings of the Second IEEE International Symposium on*, mar 1995, pp. 140 – 147.

[15] D. Peters and D. Parnas, "Requirements-based monitors for real-time systems," *Software Engineering, IEEE Transactions on*, vol. 28, no. 2, pp. 146 –158, feb 2002.

[16] B. Schroeder, "On-line monitoring: a tutorial," *Computer*, vol. 28, no. 6, pp. 72 –78, jun 1995.

[17] E. Bodden, P. Lam, and L. Hendren, "Finding programming errors earlier by evaluating runtime monitors ahead-of-time," in *SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. New York, NY, USA: ACM, 2008, pp. 36–47.

[18] F. Chen and G. Roşu, "MOP: an efficient and generic runtime verification framework," in *OOPSLA '07: Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications*. New York, NY, USA: ACM, 2007, pp. 569–588.

[19] M. B. Dwyer, A. Kinneer, and S. Elbaum, "Adaptive online program analysis," in *ICSE '07: Proceedings of the 29th international conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 220–229.

[20] X. Bai, Y. Liu, L. Wang, W.-T. Tsai, and P. Zhong, "Model-based monitoring and policy enforcement of services," in *SERVICES '09: Proceedings of the 2009 Congress on Services - I*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 789–796.

[21] K. Mahbub and G. Spanoudakis, "Run-time monitoring of requirements for systems composed of web-services: initial implementation and evaluation experience," in *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*, july 2005, pp. 257 – 265 vol.1.

[22] L. Baresi and S. Guinea, "Towards dynamic monitoring of ws-bpel processes," in *ICSOC 2005, Third International Conference of Service-Oriented Computing, volume 3826 of Lecture Notes in Computer Science*. Springer, 2005, pp. 269–282.

[23] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti, "Run-time monitoring of instances and classes of web service compositions," in *Web Services, 2006. ICWS '06. International Conference on*, sept. 2006, pp. 63 –71.

[24] Z. Li, Y. Jin, and J. Han, "A runtime monitoring and validation framework for web service interactions," in *Software Engineering Conference, 2006. Australian*, april 2006, pp. 10 pp. –79.

[25] Google App Engine. Google Inc. [Online]. Available: http://code.google.com/appengine/

[26] Cloudstatus. Hyperic Inc. [Online]. Available: http://www.cloudstatus.com/

[27] H. Han, S. Kim, H. Jung, H. Y. Yeom, C. Yoon, J. Park, and Y. Lee, "A restful approach to the management of cloud infrastructure," in *Proc. IEEE International Conference on Cloud Computing CLOUD '09*, Sep. 21–25, 2009, pp. 139–142.