Journal of Software Engineering

ISSN 1819-4311

Academic Journals Inc.

www.academicjournals.com

OPEN ACCESS Journal of Software Engineering

ISSN 1819-4311
DOI: 10.3923/jse.2017.246.254

Case Report
CloudAmulet: Promoting Software Reuse in Datacenter Application Monitoring

[1]Bo Ding, [1]Huaimin Wang, [1]Dianxi Shi, [1]Hui Liu, [2]Chang Guo Guo and [2]Jie Zhang

[1]National Key Laboratory of Parallel and Distributed Processing, School of Computer, National University of Defense Technology, Changsha, China
[2]China Electric Equipment and Systems Engineering Ltd., Beijing, China

Abstract

Background: Online monitoring is of great importance to datacenter applications. Since a datacenter usually hosts a large number of applications, it is cost-effective to provide a generic and reusable application-monitoring platform instead of creating monitoring facilities from scratch for each application. However, it is of great challenge to fit diversified monitoring requirements of various applications by a unified platform. **Material and Methods:** This study proposes such a monitoring platform named CloudAmulet. It can effectively collect, aggregate, analyze and visualize various application-level monitoring data. To achieve this goal, this platform is designed to be a highly configurable monitoring infrastructure whose behavior can be customized by application-specific meta entities, including application metric meta data, visualization meta data and data aggregation rules. By dynamically loading different meta entities, CloudAmulet can support the monitoring of different applications. **Results:** CloudAmulet has been successfully applied to a set of real production systems. The case study based on a real-life datacenter application illustrates that by promoting software reuse, our approach can reduce nearly 85% lines of code in comparison with realizing the application monitoring capability from scratch. **Conclusion:** CloudAmulet significantly promotes the software reuse in datacenter application monitoring and efficiently reduces the effort in enabling the online monitoring of those applications.

Key words: Software monitoring, monitoring platform, software reuse, datacenter applcation, cloud computing

Received: September 23, 2016 Accepted: November 08, 2016 Published: March 15, 2017

Citation: Bo Ding, Huaimin Wang, Dianxi Shi, Hui Liu, Chang Guo Guo and Jie Zhang, 2017. CloudAmulet: Promoting software reuse in datacenter application monitoring. J. Software Eng., 11: 246-254.

Corresponding Author: Bo Ding, National Key Laboratory of Parallel and Distributed Processing, School of Computer, National University of Defense Technology, Changsha, China Tel: +8618670389632

Competing Interest: The authors have declared that no competing interest exists.

Data Availability: All relevant data are within the paper and its supporting information files.

INTRODUCTION

In order to accommodate large amounts of user requests simultaneously, datacenter applications are usually designed to be a large-scale distributed software system in the backend. It consists of multiple logical entities (e.g., load balancers, business components and databases), runs on hundreds or thousands of computing nodes and involves million lines of code with complex logic. For example, in a popular Chinese email service named Aliyun mail[1], a typical user request goes through dozens of physical nodes and invokes over 100 internal methods on average. Because of the scale and the complexity, even ifsuch an application has passed a full and comprehensive test, hidden bugs may still emerge under certain running circumstances and result in unexpected (or even disastrous) consequence.

Since software defects cannot be eliminated entirely before the deployment, alternative ways have to be explored. A feasible way, which has been adopted by many real production systems is online monitoring[2,3]. In this approach, the internal states of a datacenter application are deliberately exposed. Thus software anomalies can be detected in its early stage and the operators can take appropriate measures in time, for example, rescheduling resources, modifying configurations or fixing the bug before the situation gets out of hand. However, monitoring the internal states of datacenter applications is not a simple task. It usually needs a large amount of code to collect, transfer, handle and visualize the application internal states. Since a data center usually hosts a large number of applications, it is cost-effective to provide a generic monitoring platform instead of creating monitoring mechanisms for each application from scratch.

However, monitoring requirements on the application-level, i.e., what should be monitored and how to process/visualize them, are usually highly application-specific. It is of great challenge to fit diversified monitoring requirements of various applications by a unified platform. In this study, such a platform named CloudAmulet is proposed. To enable the software reuse[4] in datacenter application-level monitoring process, the run time architecture of this platform is deliberately split into two parts: The monitoring infrastructure and a set of application specialization mechanisms named meta-entities. The former consists of a set of components that are common in a monitoring system, such as the monitoring agents running on each node and the data aggregation service aggregating monitoring data in a distributed system. However, unlike the predefined and fixed behavior of their counterparts in traditional monitoring systems such as[5-7], the behavior of those entities can be dynamically configured by the meta-entities, which defines what should be monitored in a certain datacenter application, how to get/aggregate them and how to visualize them at runtime. As a result, CloudAmulet can support the monitoring of various datacenter applications effectively. With the development tools provided by CloudAmulet, the developers of monitoring capability can just focus on the definition of those meta-entities, developing them quickly and cost-effectively instead of realizing the whole monitoring process from scratch.

MATERIALS AND METHODS

Motivated scenario: Considering a datacenter that hosts a set of cloud services, such as e-mail, cloud storage, instant message and video on demand. And more cloud applications are expected to be deployed in the future. As mentioned earlier, the effective runtime monitoring of those applications is of great importance. To realize this goal in a cost-effective way, a generic application-level monitoring platform is decided to be introduced (Fig. 1). Traditionally, while a monitoring system is developed, the syntax and semantics of monitoring targets are usually known in advance. For

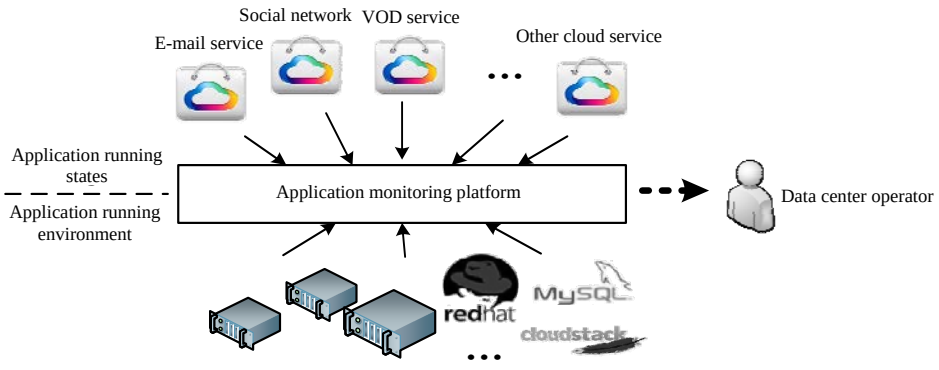

Fig. 1: Motivated scenario

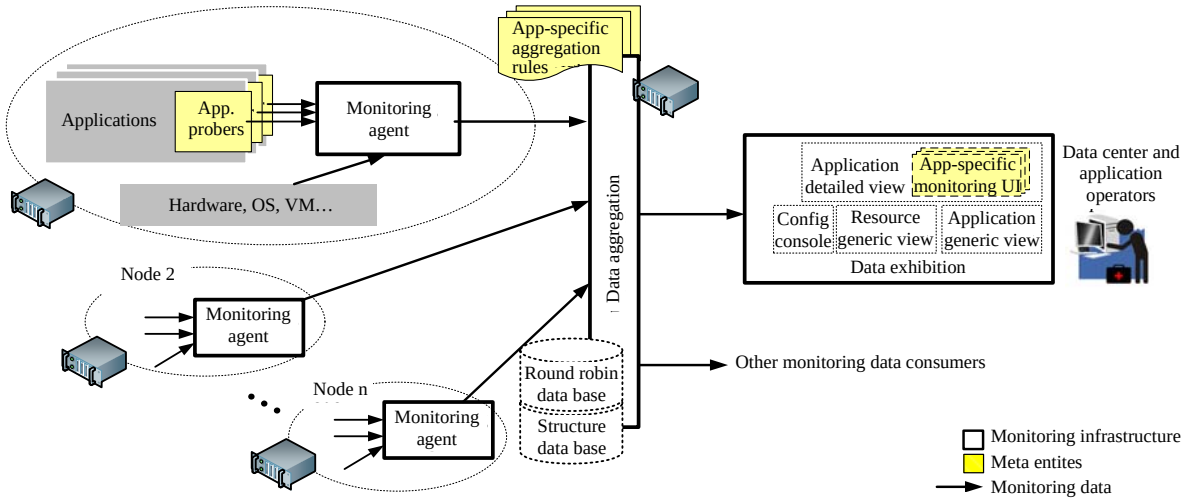

Fig. 2: Runtime architecture of CloudAmulet

example, before developing a CPU monitoring software, it is already known to the developers on how to get the CPU usage. Thus, this metric can be appropriately accessed, handled and visualized. In contrast, a generic application-level monitoring platform in this motivated scenario has to support a wide variety of cloud applications (including known and unknown ones which will be deployed in the future). While developing this platform, it is impossible for us to know exactly how to get various application-level metrics and how to appropriately handle, aggregate and visualize them.

It is challenging to fit diversified monitoring requirements of various cloud services by a single platform. In concrete, in contrast with a traditional monitoring system, this platform should support the following features: (1) Embracing application-specific metrics. Since this monitoring platform should be a "Generic" one, the target metrics cannot be assumed in advance. This platform should be able to handle various (known and unknown) metrics from various applications, (2) Supporting customized visualization. The monitoring interface of each application should be able to be customized. For example, in order to represent the running state of the e-mail service in an intuitive manner, the text line chart or pie chart may be used to exhibit different metrics and (3) Enabling application-level aggregation. Since a cloud application is usually a distributed system, the platform has to support the aggregation of application-specific metrics from different computing nodes. For example, a deep-seated anomaly may be able to be detected only by considering metrics synthetically which are collected from both the business logic layer and the database layer.

CloudAmulet runtime architecture: To address the above-mentioned challenges, the runtime architecture of CloudAmulet (Fig. 2) can be dived into two parts: The monitoring infrastructure and the meta entities, in which the latter specializes the former's behavior to fit the diversified monitoring need from application to application.

Monitoring infrastructure: The reusable monitoring infrastructure captures the commonalities across applications. The monitoring agents are deployed onto each node (VMs or physical machines) to collect various local monitoring data, not only the hardware resource status but also the states of software entities running on this computing node. The data aggregation service aggregates data collected by the agents, analyzing them and putting them into the database for future access. The data exhibition service generates web pages to visualize the aggregated monitoring data. Those facilities form a monitoring data "Collection-aggregation-exhibition" path. This path appears frequently in many datacenter monitoring solutions such as Gangalia, Zabbix and DARGOS[3]. However, unlike existing solutions, these facilities in CloudAmulet are not designed to only handle predefined monitoring data. They are highly "Reusable" from application to application, which means that their behavior can be customized by a set of dynamically-loaded meta entities.

Meta-entities: The meta-entities contain the monitoring knowledge specific to certain applications, making the monitoring infrastructure suitable for various application-level monitoring requirements.

Application probers are embedded in the target application to enable the real-time data collection of application-specific metrics for example, the "Mails sent per second" metric in an e-mail service or the "Average download speed for files above 100 M" metric in a cloud storage service. A prober observes and measures those metrics and publishes the result to the local monitoring agent in a self-descriptive data format. The prober also provides the app-specific monitoring UIs which exhibit the application-level monitoring data by a set of predefined monitoring UI controls such as simple texts, tables, line charts or pie charts. Those UIs will be rendered dynamically by the data exhibition service in Fig. 2.

App-specific aggregation rules directs the data aggregation service to gather and analyze monitoring data in the dimension of both space and time, which may trigger certain warnings and predefined actions. Various algorithmic, bool and string operations on the data collected from different computing nodes are supported, for example to calculate the average load of a cluster in a sliding time window.

Gathering and exhibitng application internal states: The following two subsections mainly focus on the meta entities in CloudAmulet. The application probers in CloudAmulet is a self-contained entity, which means that it contains not only the logic to collect the specified monitoring metrics but also the meta description of those metrics, including both the definition of those metrics (i.e., metric meta data) as well as how to exhibit those metrics at runtime (i.e., visualization meta data).

Definition 1: Metric meta data m is defined as <id, type, properties>.

The type of the metric can be divided into three kinds: (1) Simple types, including string, integer, float, boolean, datetime, etc. (2) Complex types, including sequences and entities, (3) Template types, which are predefined types with special monitoring semantics. The introduction of the template type is to enable the underlying monitoring infrastructure to handle the data flexibly. A typical example of this type in CloudAmulet is storage space, an integer type that the infrastructure automatically transfers it into appropriate formats (i.e., MB, GB, TB, etc.) while exhibiting it. A property is a key-value pair that guides the monitoring infrastructure to handle this metric. For example, the history length for an integer of a float type tells the infrastructure to maintain how many history values in its round-robin database.

Definition 2: Visualization meta data v is defined as. The metrics related to meta data v is referred to as <id, control, S_m, properties>. The metrics related to meta data v is referred to as metrics (v).

In this definition, the control is the visualization form of a specific metric or metric set, whose form may be simple text, table, pie chart, line chart, bar chart, etc. The S_m is the metric set which is shown in the control. The properties are key-value pairs which tell the monitoring infrastructure how to show the metrics, for example, the location and size of the control.

Then, a CloudAmulet prober can be defined as follows. In this definition, S_v is the metrics that the prober are supposed to collect is the visualization meta data set of those metrics and the monitoring logic is the concrete code to collect the metrics' current value.

Definition 3: Prober p is defined as, <$skeleton_p$, $monitoring\_logic_p$>, in which $skeleton_p$ = <id, description, S_m, S_v>, for each $v \in S_v \subseteq S_m$.

The probers are expected to be constructed by the developers of application monitoring capability. To facilitate this activity, CloudAmulet provides a GUI tool to define the metric meta data (Fig. 3a) as well as a tool to define the visualization meta data in a control "Drag and drop" style (Fig. 3b). Then CloudAmulet can generate the prober skeletonin C++ or Java according to the metric meta data and the visualization meta data.The monitoring logic is leftas blank functions which are supposed to be filled by the developer. By importing the skeleton into the application (as a library) and filling in the blank functions, the monitoring capability is added into the target application. At runtime, the monitoring infrastructure of CloudAmulet actively inquires the interfaces of the probers and gets all the meta information, knowing what should be collected and how to exhibit those metrics. It dynamically interprets the meta data and automatically starts the application-level monitoring process. For example, the visualization meta data are transferred to the data exhibition service in Fig. 2 and dynamically rendered as web pages to exhibit the application states. The whole process is illustrated in Fig. 4.

Aggregating and analyzing application metrics: In addition to gathering and exhibiting the application states, another common monitoring action in a data center is to aggregate the value of a set of application metrics to find existing or potential software anomalies. To support this kind of actions, CloudAmulet introduces a domain-specific language[8],

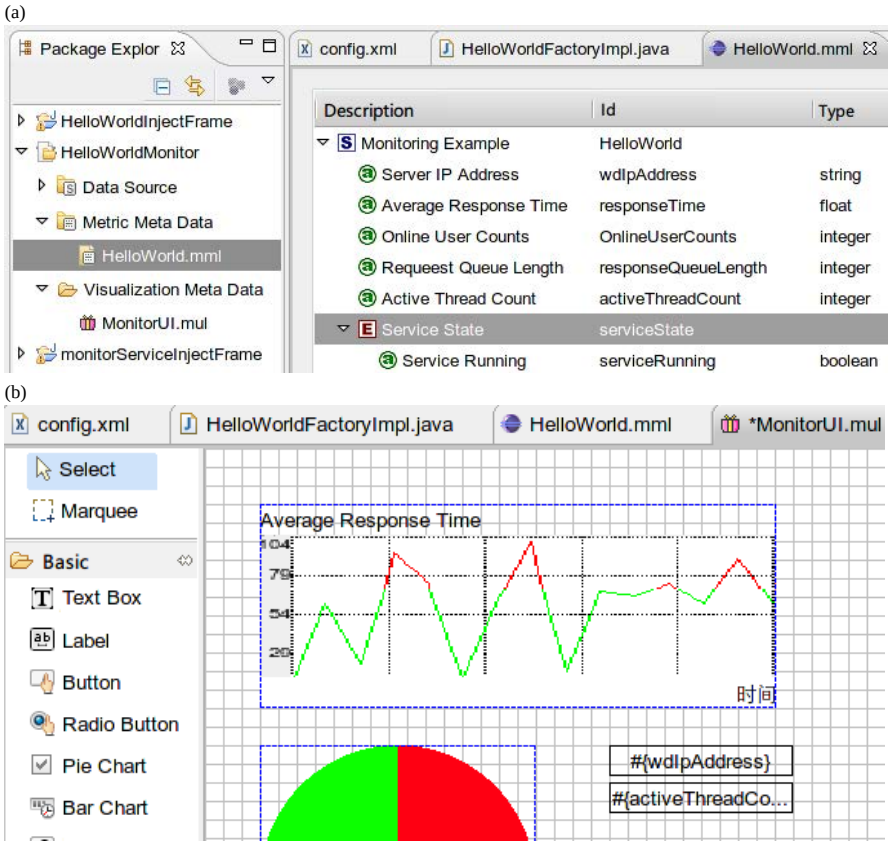

Fig. 3(a-b): Meta data defintion tools in CloudAmulet, (a) Metric meta data definition and (b) Visualization meta data definition

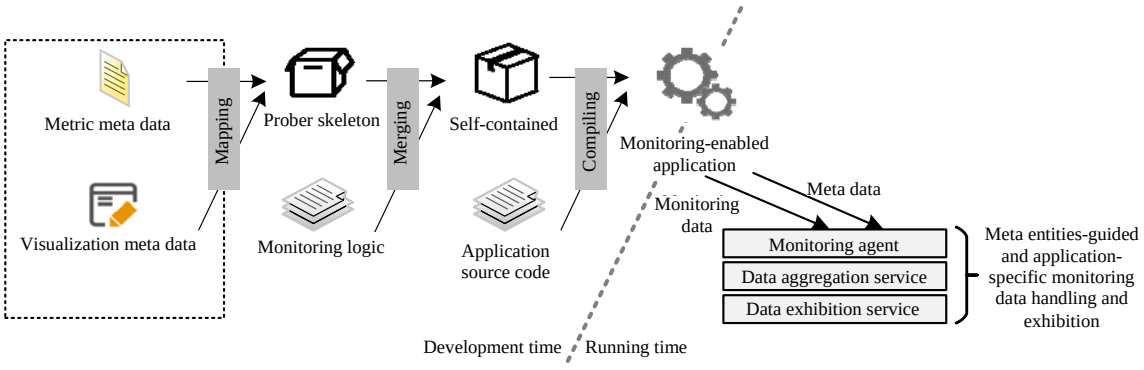

Fig. 4: Gathering and exhibiting application internal states

MonDSL. It enables the definition of application-level monitoring rules, which tells the infrastructure how to analyze application metrics at runtime. Basically, the rules are defined in the form "When-then". Table 1 shows that, the when part is a boolean expression involving a set of application metrics. Those metrics can be from different applications or even physical nodes. MonDSLalso provide a set of built-in functions to calculate those metrics on a sliding window, for example, the average value or the peak value in 10 min. In the Then part, the operator can invoke predefined actions, such as warning in the data exhibition service, sending email, rebooting an application or a computing node, etc. Those MonDSL rules are dynamically interpreted by the data aggregation service in Fig. 2. It subscribes the necessary monitoring data from the monitoring agents and executes those rules while new data arrive.

Table 1: Grammar and example of mondsl rules

Grammar	Example
Rule "Name"	Rule "Load alert"
When	When
LHS//conditions	Avg (node (192.168.10.2).app ("mail", 0). metric ("Request response time"), 600)>5)
Then	And
Action//actions	Avg (node (192.168.10.2).app ("mail",0). metric ("Request response time"), 600)>5)
End	Then
Timer<timespan>	Alert ("CPU overloaded in cluster 1")
//Sample time	Send mail ("xxx@a.com", self.description)
	End
	Timer 30

RESULTS

CloudAmulet implementation: CloudAmulet supports the monitoring of C++/Java applications both on windows and Linux-compatible platforms currently. The runtime communication inside the monitoring infrastructure is based on DDS (data distribution service for real-time systems)[9], a QoS-enabled publish/subscribe middle ware specification proposed by Object Management Group (OMG). It enables CloudAmulet to transfer scalable monitoring data in a large-scale data center. To enable the dynamic loading and interpretation of MonDSL rules, CloudAmulet adopts Drools Fusion (JBoss Community. http://drools.jboss.org/drools-fusion), an open-source complex event processing engine. The MonDSL rules are mapped into the rules of Drools Fusion and the monitoring data are encapsulated as events which can be feed into Drools Fusion. The metric meta data output by the probers significantly facilitates the transformation from CloudAmulet data to Drools events.

Real-life case study: CloudAmulet has been applied to a set of real-life data center production systems. This subsection concerns an email cloud service that provides sending/receiving email functions by browsers as well as POP3/IMAP protocols. To support a large number of concurrent users ($\geq$1.8 K users at peak), the backend implementation of this service is made up of a set of clusters in the datacenter, including the mail handling cluster, the database cluster, the distributed file system cluster and the remote access cluster (being responsible for generating web pages). Figure 5a is the home web page of CloudAmulet at runtime, Fig. 5b is a snapshot of designing the monitoring UI in Eclipse with the visualization definition tool and Fig. 5c is the web page rendered by CloudAmulet according to the visualization meta data in Fig. 5b as well as the real-time monitoring data collected.

Before migrating the software entities running on those clusters to CloudAmulet, they have been built in the monitoring capabilities. However, all of them are realized in an ad hoc manner, i.e., for each software entity, the application developers developed the monitoring probers, monitoring facilities and monitoring tools from scratch. As shown in Fig. 5d, although some general reusable libraries such as distributed computing middleware have been adopted, the Line of Codes (LoC) of monitoring capability in this system is 50235. After migrating to CloudAmulet, there are only 7367 LoC (about 15% of the original one) to realize the similar monitoring capabilities. In particular, the LoC of monitoring facilities is dropped from 37435 (several monitoring GUI tools) to 244 (MonDSL rules), since CloudAmulet provide the capability to design monitoring UIs (i.e., defining the visualization meta data) without coding as well as the reusable facilities which can interpret the UIs and MonDSL rules dynamically. Software reuse plays a significant role in reducing development effort. It also reduces the operating cost of the monitoring system since only one instance of this system for all applications needs maintaining in the datacenter.

In addition to the benefit of software reuse, another major concern is the performance cost of introducing CloudAmulet probers. Since the monitoring logic is realized by the application developers, CloudAmulet provides no assurance on it. However, as a typical example, the performance cost in the mail handling cluster beingmade up of 8 physical servers is tested in our experiment. Figure 5e, when the load (i.e., the number of concurrent users) is light and medium, there is nearly no loss in a major performance indicator, the average request response time, while handling the mails. When the load is near the upper bound of the cluster (i.e., 2400 concurrent requests per second), there is 20-30% inevitable performance loss, depending on the configured refresh rate of the monitoring data. Furtherly, we compared the performance before and after the migration of the monitoring capability of the email cloud service to CloudAmulet, in which the difference can indicate the precise over head brought by the CloudAmulet prober skeletons itself.

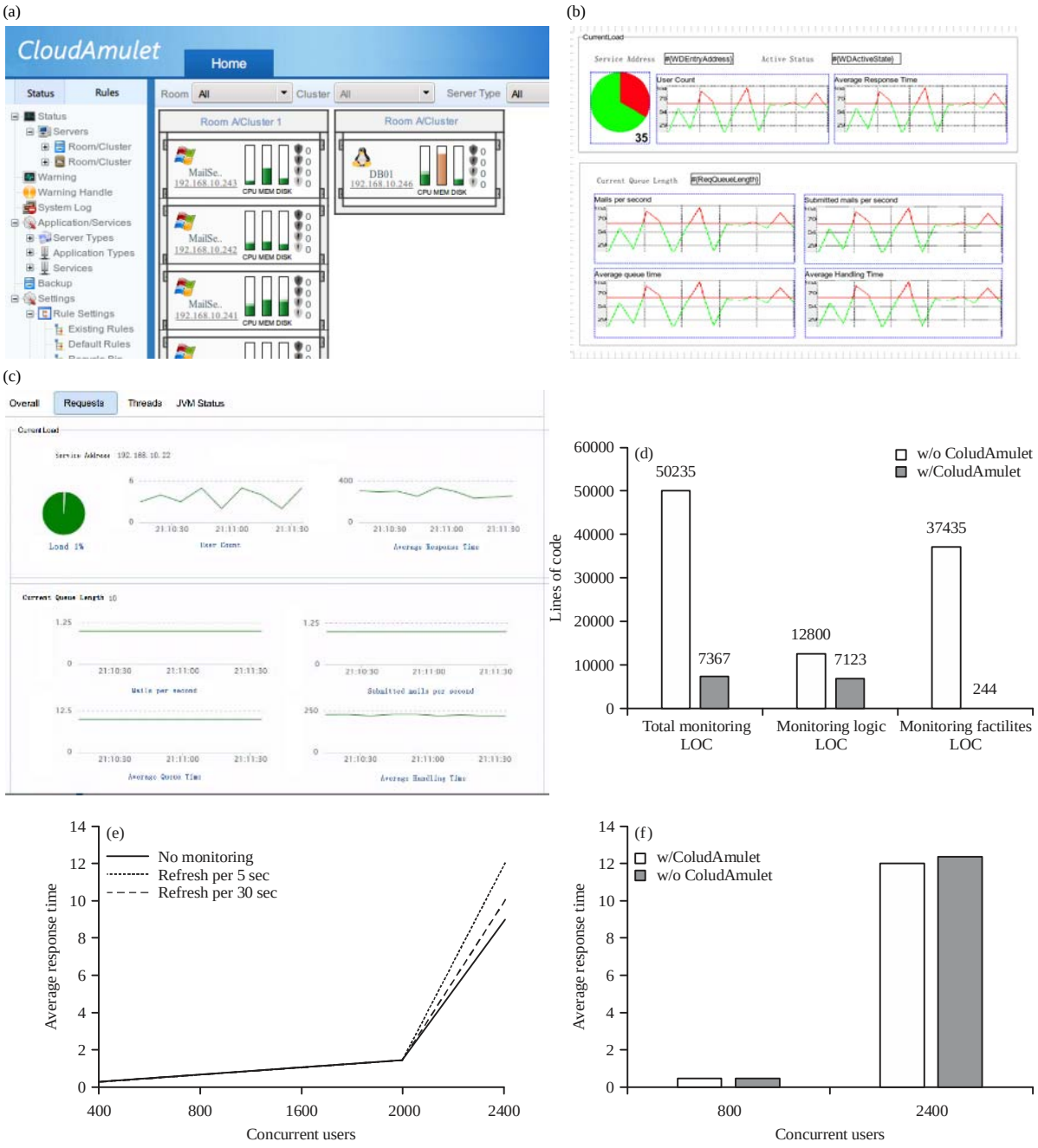

Fig. 5(a-f): CloudAmulet real-lif case study, (a) CloudAmulet home webpage, (b) Designing monitoring UI in eclipse, (c) Rendering monitoring UI in browser, (d) Lines of code (LoC) comparison, (e) Performance cost of monitoring and (f) Performance cost of CloudAmulet

In Fig. 5f, when getting the same metric sets at the same refresh time 5 sec, the performance gap between these two implementations (before and after adopting CloudAmulet) is no more than 3% under both a light load (800 concurrent requests per second) and a heavy load (2400 concurrent requests per second). In contrast with the great benefits from software reuse, this extra performance overhead is acceptable in most cases.

DISCUSSION

There have been many attempts to provide monitoring solutions in cloud computing environment. However, a large portion of them (e.g., PCMONS[10], DARGOS[5] and Lattice[11] are mainly designed to support IaaS or PaaS-level resource management. Their focus point is the running state of low-level datacenter infrastructure, such as hardware, virtual

machines, operating systems and common PaaS services, instead of high-level applications this study focuses on.

Recently, some works in this field realize the magnitude of providing generic support for application-level monitoring. GMonE[12] is a general purpose cloud monitoring tool, whose design goal is to be "Applicable to all areas of cloud computing". It introduces the software entity named "Monitoring plugins" to support customized application metrics. The MISURE[13] builds the monitoring infrastructure for cloud applications on a streaming process framework. It can collect and aggregate various metrics from disparate cloud applications in a near-real time way. Katsaros *et al.*[14], a multi-layered monitoring framework for measuring QoS at both application and infrastructure levels is presented. It uses a script-based data collector to collect monitoring data, which is a responsibility of the service developers mOSAIC[15] offers a set of tools whose goal is to generate warnings when the target application and/or the associated resources are in conditions which may lead to runtime problems. Kieker is an extensible framework for runtime monitoring of the behavior of distributed software systems, which collects the application-level measurements by instrument probers into target software system[16]. Besides, both in the commercial and open source society[3,17], some well-known monitoring solutions such as Ganglia, Nagoisand Zabbix can support customized metrics by adding plugins, which can be used to support the application-level monitoring.

However, these closely-related work just focus on the generic mechanisms to enable application-specific metrics such as service Key Performance Indicators (KPIs). In contrast, our platform takes this step further. As a generic application-level monitoring platform which lays strong emphasis on software reuse, it supports not only customized metrics but also customized visualization of monitoring data and customized application-level aggregation. They are essential to effective application monitoring in real large-scale production systems. In concrete, the customized visualization mechanism can help the human operators to understand the current behavior of a specific application efficiently and the customized aggregation mechanism can cope with monitoring requirements on the scale of a system instead of a physical node. By providing reuse on those two aspects, CloudAmulet can support the construction of the monitoring capability of various complex datacenter applications cost-effectively.

Another highly-related field is cloud application SLA and performance monitoring. Service Level Agreement (SLA) supervision is common in the management of datacenter applications[18], in which the application behavior is monitored to detect the violation of predefined SLAs. The CASViD[19] is an architecture that monitors and detects the application-level SLA violation, which includes tools for resource allocation, scheduling and deployment. M4Cloud[20] provide a generic application level monitoring system to detect SLA violation in a resource-shared cloud environment. An approach allows application developers to specify and monitor high-level application performance metrics is presented in Leitner *et al.*[21], which adopts the complex event processing paradigm to handle correlated events. Since SLA is a contract between the client and Cloud Service Provider (CSP), these existing work mainly focus on the externally observable metrics of cloud applications. In contrast, CloudAmulet concerns the internal states of cloud applications in this study, which can contribute to datacenter application maintenance in two ways: Detecting software anomaly as early as possible and promoting the diagnose accuracy of the root cause of a software anomaly. Application Performance Management (APM) is another booming field in cloud computing practices[22]. It aims at detecting application performance problems as well as their roots by integrating mining approaches on monitoring data into performance indicator collecting tools. Although, some solutions support the manual instrumentation of application code which is similar to our study[23], it does not concern customized visualization of monitoring data and customized application-level aggregation, what makes our work a generalized and reusable application monitoring platform instead of just a performance anomaly detection/analyzation tool.

CONCLUSION

This study presents CloudAmulet, a monitoring platform for data center applications which aims at promoting software reuse in the application-level monitoring process. To achieve this goal, the runtime architecture of CloudAmulet is divided into two parts: The common monitoring infrastructure and the application-specific meta entities. The dynamically-loading meta entities can guide the behavior of the common monitoring infrastructure, customizing them to fit the monitoring requirement of different applications. The application in real production systems has validated the great benefits of introducing CloudAmulet into datacenter application monitoring.

ACKNOWLEDGMENT

This study is supported by the National Natural Science Foundation of China (No. 91118008, No. 61202117). The authors would like to thank all those who contributed to the implementation of CloudAmulet.

REFERENCES

1. Mi, H., H. Wang, Y. Zhou, M.R.T. Lyu and H. Cai, 2013. Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems. IEEE Trans. Parallel Distrib. Syst., 24: 1245-1255.
2. Wang, H. and B. Ding, 2016. Growing construction and adaptive evolution of complex software systems. Sci. China Inform. Sci., 59: 1-3.
3. Aceto, G., A. Botta, W. De Donato and A. Pescape, 2013. Cloud monitoring: A survey. Comput. Networks, 57: 2093-2115.
4. Krueger, C.W., 1992. Software reuse. ACM Comput. Surv., 24: 131-183.
5. Povedano-Molina, J., J.M. Lopez-Vega, J.M. Lopez-Soler, A. Corradi and L. Foschini, 2013. DARGOS: A highly adaptable and scalable monitoring architecture for multi-tenant clouds. Future Gener. Comput. Syst., 29: 2041-2056.
6. Calero, J.M.A. and J.G. Aguado, 2015. MonPaaS: An adaptive monitoring platformas a service for cloud computing infrastructures and services. IEEE Trans. Serv. Comput., 8: 65-78.
7. Barth, W., 2008. Nagios: System and Network Monitoring. 2nd Edn., No Starch Press, San Francisco, CA., ISBN: 9781593271794, Pages: 720.
8. Mernik, M., J. Heering and A.M. Sloane, 2005. When and how to develop domain-specific languages. ACM Comput. Surv., 37: 316-344.
9. Pardo-Castellote, G., 2004. OMG data-distribution service: Architectural overview. Real-Time Innovations, pp: 200-206. https://community.rti.com/sites/default/files/DDS_Architectural_Overview.pdf
10. De Chaves, S.A., R.B. Uriarte and C.B. Westphall, 2011. Toward an architecture for monitoring private clouds. IEEE Commun. Magaz., 49: 130-137.
11. Clayman, S., A. Galis and L. Mamatas, 2010. Monitoring virtual networks with lattice. Proceedings of the IEEE/IFIP Network Operations and Management Symposium Workshops, April 19-23, 2010, IEEE Computer Society Washington, DC, USA., pp: 239-246.
12. Montes, J., A. Sanchez, B. Memishi, M.S. Perez and G. Antoniu, 2013. GMonE: A complete approach to cloud monitoring. Future Gener. Comput. Syst., 29: 2026-2040.
13. Smit, M., B. Simmons and M. Litoiu, 2013. Distributed, application-level monitoring for heterogeneous clouds using stream processing. Future Gener. Comput. Syst., 29: 2103-2114.
14. Katsaros, G., G. K ousiouris, S.V. Gogouvitis, D. Kyriazis, A. Menychtas and T. Varvarigou, 2012. A self-adaptive hierarchical monitoring mechanism for clouds. J. Syst. Software, 85: 1029-1041.
15. Rak, M., S. Venticinque, T. Mahr, G. Echevarria and G. Esnal, 2011. Cloud application monitoring: The mOSAIC approach. Proceedings of the IEEE 3rd International Conference on Cloud Computing Technology and Science, November 29-December 1, 2011, IEEE Computer Society, Washington, DC, USA., pp: 758-763.
16. Van Hoorn, A., J. Waller and W. Hasselbring, 2012. Kieker: A framework for application performance monitoring and dynamic software analysis. Proceedings of the 3rd Joint WOSP/SIPEW International Conference on Performance Engineering, Boston, MA, USA., April 22-25, 2012, ACM Press, New York, pp: 247-248.
17. Alhamazani, K., R. Ranjan, K. Mitra, F. Rabhi and P.P. Jayaraman *et al*., 2015. An overview of the commercial cloud monitoring tools: Research dimensions, design issues and state-of-the-art. Computing, 97: 357-377.
18. Baset, S.A., 2012. Cloud SLAs: Present and future. ACM SIGOPS Operat. Syst. Rev., 46: 57-66.
19. Emeakaroha, V.C., T.C. Ferreto, M.A. Netto, I. Brandic and C.A.F. De Rose, 2012. CASViD: Application level monitoring for SLA violation detection in clouds. Proceedings of the IEEE 36th Annual Computer Software and Applications Conference, July 16-20, 2012, EEE Computer Society Washington, DC, USA., pp: 499-508.
20. Mastelic, T., V.C. Emeakaroha, M. Maurer and I. Brandic, 2012. M4Cloud-Generic application level monitoring for resource-shared cloud environments. Proceedings of the CLOSER 2012, 2nd International Conference on Cloud Computing and Services Science, April 18-21, 2012, Porto, Portugal, pp: 522-532.
21. Leitner, P., C. Inzinger, W. Hummer, B. Satzger and S. Dustdar, 2012. Application-level performance monitoring of cloud services based on the complex event processing paradigm. Proceedings of the 5th IEEE International Conference on Service-Oriented Computing and Applications, December 17-19, 2012, IEEE Computer Society Washington, DC, USA., pp: 1-8.
22. Rabl, T., S. Gomez-Villamor, M. Sadoghi, V. Muntes-Mulero, H.A. Jacobsen and S. Mankovskii, 2012. Solving big data challenges for enterprise application performance management. Proc. VLDB Endowment, 5: 1724-1735.
23. Ahmed, T.M., C.P. Bezemer, T.H. Chen, A.E. Hassan and W. Shang, 2016. Studying the effectiveness of Application Performance Management (APM) tools for detecting performance regressions for web applications: An experience report. Proceedings of the 13th International Conference on Mining Software Repositories, Austin, TX, USA., May 14-15, 2016, ACM Press, New York, pp: 1-12.