Towards Cross-Layer Monitoring of Multi-Cloud Service-Based Applications

Chrysostomos Zeginis, Kyriakos Kritikos, Panagiotis Garefalakis, Konstantina Konsolaki, Kostas Magoutis, and Dimitris Plexousakis

Institute of Computer Science
Foundation for Research & Technology – Hellas
Heraklion 70013, Greece
{zegchris,kritikos,pgaref,konsolak,magoutis,dp}@ics.forth.gr

Abstract. Cloud computing is becoming a popular platform to deliver *service-based applications* (SBAs) based on service-oriented architecture (SOA) principles. Monitoring the performance and functionality of SBAs deployed on multiple Cloud providers (in what is also known as *Multi-Cloud* setups) and adapting them to variations/events produced by several layers (infrastructure, platform, application, service, etc.) in a coordinated manner are challenges for the research community. This paper proposes a monitoring framework for Multi-Cloud SBAs with two main objectives: (a) perform cross-layer (Cloud and SOA) monitoring enabling concerted adaptation actions; (b) address new challenges raised in Multi-Cloud SBA deployment. The proposed framework is empirically evaluated on a real-world Multi-Cloud setup.

Keywords: Cloud computing, service-oriented architecture, monitoring, modeling, event processing, service dependencies.

1 Introduction

Cloud computing emerges as a dominant IT services paradigm that enterprizes increasingly acknowledge for its ability to flexibly host applications over managed virtualized infrastructures. As in any distributed application hosting environment, Clouds must support extensive monitoring mechanisms to aid in controlling application performance and adapt to infrastructure variations.

Considering the close relations between Cloud (IaaS, PaaS and SaaS) and SBAs layers (Business Process and Management (BPM), Service Composition and Coordination (SCC) and Service Infrastructure (SI) [9]), it is essential to perform and correlate monitoring across all layers. While it is hard to overestimate the value of effective monitoring (strong infrastructure control, support for elasticity policies and quality of service (QoS)), most related approaches are fragmented (confined within a specific Cloud provider or service layers) and not applicable/aligned across layers. Multi-Cloud SBA deployment further complicates this due to lack of cross-platform support for uniform monitoring solutions [3].

This paper addresses the cross-layer Cloud SBA monitoring by exploiting the dependencies among layers and using the *event patterns* concept. It supports

K.-K. Lau, W. Lamersdorf, and E. Pimentel (Eds.): ESOCC 2013, LNCS 8135, pp. 188–195, 2013.

Multi-Cloud SBA deployment by distributing a monitoring mechanism across Cloud providers. Our monitoring framework relies on an event model to specify the possible monitored SBA events in a Cloud environment, and a component model to describe component dependencies [8] and capture system snapshots at any particular time point. Our evaluation indicates that collecting monitored events can be effectively distributed across Cloud providers. Event retrieval and publication towards a rule engine can be efficiently performed from any location.

The paper is structured as follows. Section 2 describes the architecture of our monitoring engine and implementation details. Section 3 introduces the event model. Section 4 evaluates aspects of our monitoring system. Section 5 describes related work. Finally, section 6 draws conclusions and future work directions.

2 Architecture Overview

The architecture presented in this paper builds upon our previous work [10] on cross-layer SBA monitoring and adaptation, extending it to a Multi-Cloud setting. It comprises a Monitoring Engine, collecting cross-layer events during SBA execution, and an Adaptation Engine, performing cross-layer adaptation actions, which communicate events via a publish/subscribe mechanism (Figure 1). A Model Repository provides various information, such as service descriptions, Multi-Cloud deployment models, layer dependencies, and metric/SLA models.

In this Multi-Cloud setting, SBAs are deployed on various Clouds based on the provided requirements. Three monitoring components are used to perform monitoring at the SaaS, PaaS, and IaaS layers, while a manager retrieves their monitoring results, stores them at a time-series database, and reports detected violations via the publish/subscribe mechanism to Adaptation Engine instances.

This paper focuses primarily on the Monitoring Engine and Model Repository implementation in Multi-Cloud setups. We define monitored events using OWL-Q [6], a semantic and extensible QoS description model for SBAs. It is designed modularly, incorporating several independent QoS-based SBA description facets, such as QoS offers, requests, metrics, attributes and constraints (requirements/capabilities). The SaaS monitoring component uses the Astro monitoring

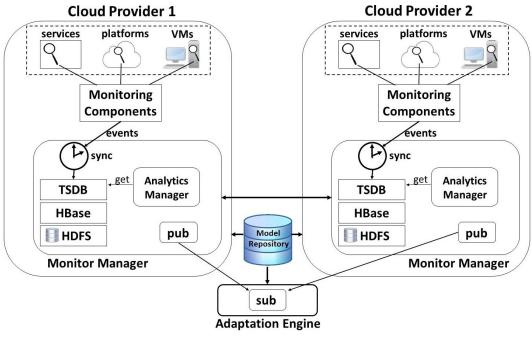

Fig. 1. Multi-Cloud deployment

tool [2] to collect events at the BPM and SCC layers. The supported QoS attributes (metrics) include service/SBA execution time (min, max), throughput (min, max, average) and availability.

The PaaS monitoring component exploits an existing cross-PaaS application management solution [11] which offers a Cloud technology-agnostic PaaS monitoring functionality and an SLA management layer, unifying diverse, provider-specific resource-level metrics. Supported metrics include application load, application and DB response time, and application container response time.

The IaaS monitoring component distinguishes between direct infrastructure monitoring and monitoring services offered by Cloud providers. We use Nagios (`http://www.nagios.org`) for direct monitoring of user-specified system resources and services via periodic checks on them. Monitored resources include memory usage, disk usage, and CPU load. We also use Amazon Cloudwatch as a Cloud monitoring service instance providing comprehensive monitoring for Cloud resources and applications run by customers on Amazon Web Services. To gain system-wide visibility of running EC2 VMs we enable a variety of metrics via the Cloudwatch API, including CPU utilization, disk read/write rate and volume of incoming/outgoing network traffic. Each Cloudwatch API request returns a datapoint that is handled as a monitored entity. Our requests are issued every few seconds to ensure that collected data are valid and can be reacted on at a reasonable latency.

Regarding event storing, standard solutions include stream processing engines and time-series databases (TSDBs). The former aim to meet stringent latency requirements when performing continuous queries on streaming data and minimize processing cost for large data sets. TSDBs differ as they focus more on persistent event storage and in performing *rollups* (e.g., aggregated metrics such as average, max, min) for user-specified intervals. Complex event processing (CEP) could also be exploited to aggregate events, but since we are interested to store both the raw events (even for a short period) and the rollups, our architecture uses (per-Cloud, federated) TSDBs. A variety of commercial and open source TSDBs can be used to handle timestamped events. We decided to use open-source OpenTSDB, a TSDB especially designed for distributed systems with high scalability requirements, to store monitored events.

A publish/subscribe mechanism handles transferring raw monitored events and TSDB rollups to the Adaptation Engine. Different adaptation engine instances may be deployed to distribute adaptation load across applications/Clouds, each interested only in relevant events and rollups. We use the Siena (`http://www.inf.usi.ch/carzaniga/siena`) pub/sub event notification service for communicating events and rollups between TSDB and Adaptation Engine. Siena is expressive enough to capture all appropriate event information via an extensible data model without sacrificing scalability and performance during event delivery.

One of our approach's main goals is to identify particular *event patterns* occurring during SBA execution that lead to critical violations so as to enable selecting the appropriate cross-layer adaptation actions. Since the publishing

order of events is significant, the Monitor Manager must time-synchronize them before being sent to the Adaptation Engine. Time synchronization is particularly important in Multi-Cloud settings as standard time synchronization solutions are rarely deployed across Cloud providers. Synchronized events are stored on a repository for post-processing to discover new patterns of interest in event streams. Various clock synchronization algorithms have been proposed to achieve temporal ordering of events produced by concurrent processes. The main approaches are those using logical clocks to create event sequence numbers and those using physical clocks to synchronize events. Physical clock-based algorithms, adjust the system components clocks based on server time or master machine time. As such algorithms are intended for use within intranets and require systematic adjustment of the machines' physical clock, a logical-clock algorithm seems more appropriate in a Multi-Cloud setting. We thus use Lamport's algorithm [7] to efficiently establish event ordering.

Finally, regarding monitor manager functionality, monitored events from within each Cloud are directed to local TSDB, which uses HBase (`http://hbase.apache.org`) (a non-relational, distributed database) to organize the event time-series. HDFS (`http://hadoop.apache.org`), a distributed file system replicating data across all Cloud providers, handles time series storage. For high performance during event collection, each Cloud's local replica is updated eagerly; remote replicas are updated in a relaxed (asynchronous) manner. Reads are performed from local copies when available. The monitor manager includes the synchronize and publish mechanisms on top of OpenTSDB. An analytics manager queries OpenTSDB to retrieve row and aggregated data to perform analysis for the adaptation engine. Stored events are tagged with other source information (service/software component, hosting resource, Cloud provider).

3 Event Model

This section presents an event meta-model describing the most common monitored event types and patterns that occur during the Cloud SBA execution. This model (Figure 2) is generic enough and extensible to incorporate any other event type defined by domain-specific service providers. A respective XML schema was designed to guarantee the validity of concrete event models defined in XML.

The main model class is *Event*. Its *CompositeEvent* and *SimpleEvent* subclasses represent simple and composite events, respectively. Composite events comprise two other (simple or composite) events (the *first* and *second*) which map to a particular *ordering*. For instance, consider a hardware event comprising a CPU overload and low available memory events. Simple events has a source component (defined in a component model not provided in this paper due to space limitations) and belong to a specific Cloud layer (SaaS, PaaS, IaaS). SaaS events can be further located at the BPM or SCC layers. Events are also characterized by their criticality as warning, critical or successful. A simple event can either be *Functional* or *Non-Functional*. Functional events refer to operational

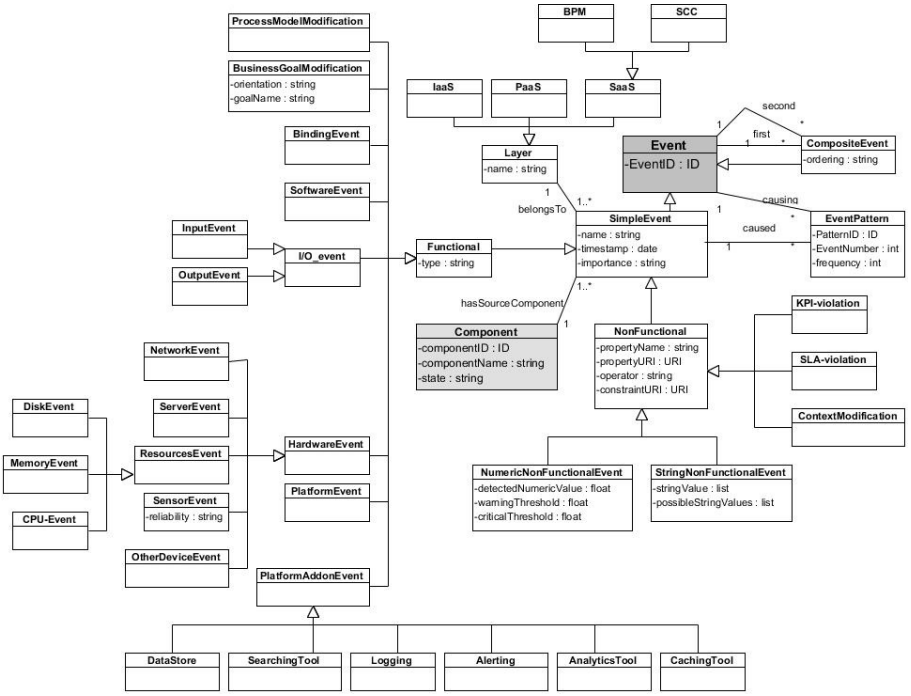

Fig. 2. The Event model

characteristics defining the overall SBA behavior, while non-functional events refer to quality attributes that are either measurable or get distinct qualitative values. Two additional and different classifications exist for non-functional events: (a) they can be classified as *KPI-violations*, *SLA-violations* or *contextModification* events, and (b) as numeric or string events. Sub-classes defined for functional events include: (a) *Process Model Modification*, (b) *Business Goal Modification*, *Software Event*, *I/O event*, *Hardware event*, and *Platform event*. Finally, the *EventPattern* class represents event pattern appearing during Cloud SBA execution and leading to critical violation events. Each event pattern has a (composite or simple) *causing event* and a simple *caused event*.

4 Evaluation

This section describes an experimental evaluation of our monitoring architecture under three deployments: single TSDB server in single Cloud provider (simple setup *1-1*); three TSDB servers in the same Cloud provider (scalability setup *3-1*); three TSDB servers in three different providers (one TSDB server in each Cloud, Multi-Cloud setup *3-3*). We use a monitored events dataset consisting of one million (1M) events comprising six metric types provided by the following sources: a service-level middleware based on the Astro monitoring tool

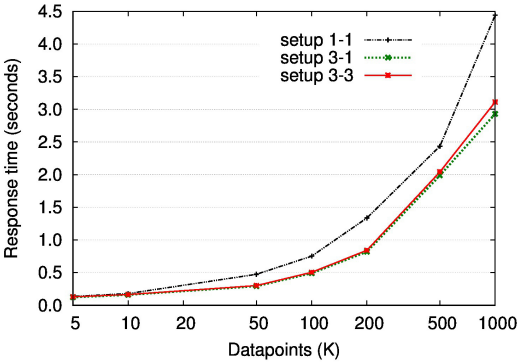

Fig. 3. TSDB read-query response time with varying scope under different setups

(Web service availability, execution time, and throughput); the Amazon Cloudwatch service (CPU utilization, data transfer, and disk usage metrics for underlying VMs). In our experiments, the Siena pub/sub mechanism retrieves events of interest from the 1M event TSDB dataset via HTTP queries reflecting Siena filters (e.g., with interest on specific metrics and/or event sources). Retrieved events are then passed to an adaptation engine (where no further action is taken).

Our first experiment evaluates the three setups in terms of TSDB query completion time with increasingly broader scope. The query ranges from returning 5K to 1M events out of the 1M event dataset. In the *1-1* setup, TSDB, HBase, and HDFS run on a single VM. In the *3-1* setup, the same software stack (TSDB, HBase, HDFS) is deployed on three VMs in a single Cloud provider (Flexiant). HDFS is configured with two data nodes (single replica per block) and a single name node (responsible for metadata). HBase is configured with two region servers and a single master. In the *3-3* setup the three VMs reside on different providers (one VM in Amazon, Microsoft Azure, and Flexiant) using the same HBase and HDFS configurations. In the *3-1*, *3-3* setups, the 1M event dataset is created on all three servers (each creates a different third of the dataset) and thus events are spread over the HBase region servers and HDFS data nodes.

Figure 3 summarizes our results. Queries with smaller scope (returning 5-10K events out of 1M examined) perform similarly on all setups. As the scope increases, setups *3-1* and *3-3* outperform *1-1* due to simultaneously involving two HBase/HDFS servers for data retrieval. *3-1* seems to outperform *3-3* only for the 1M query due to cross-Cloud communication starting to impact overall time. Although that impact is small, replication will reduce it further since local-copy reads will mask the network latency of cross-Cloud communication.

Our next experiment measures the integrated (TSDB plus publish/subscribe engine) system performance focusing on end-to-end latency (time to complete one or more queries over 1M data points) and throughput (publish ops per second). Table 1 reports our results focusing on a single query going over 1M data points with increasing scope. Our results show that latency and throughput increase with an increasing number of publish-event operations. In practice such

Table 1. End-to-end (TSDB+Siena) response time, throughput under different setups

Number of events published (K)	5	10	50	100	200	500	1000
Single query latency (sec)	0.59	0.82	1.5	2.21	3.68	7.65	11.88
Single query throughput (Kops/sec)	8.5	12.2	33.3	45.2	54.3	65.4	84.2

large queries are expected to hurt responsiveness. Smaller, more frequent queries should result into longer end-to-end latencies (although response time of individual event publish operations will improve) and lower aggregate throughput. Experiments with 100 consecutive queries over 10K data points each, publishing a total of 1M events, take 15 sec (compared to 11.88 sec with a single query) and result in a 67 Kops/sec throughput (compared to 84.2 Kops/sec for one query).

5 Related Work

While several Cloud monitoring approaches have been proposed, few comprehensively consider cross-layer issues. Alcaraz Calero et al. [1] present an analysis of a wide distributed monitoring solution set analyzing the features, requirements, and topology of a cross-layer monitoring system for Cloud computing. A number of EU-funded research projects are currently examining Cloud monitoring solutions: IRMOS (`http://www.irmosproject.eu`) offers a Cloud infrastructure, comprising a service management system acting as a link between SaaS and IaaS to manage the application service component negotiation, reservation, execution and monitoring. RESERVOIR (`http://www.reservoir-fp7.eu`) introduces the Lattice non-intrusive monitoring framework for Cloud applications. Lattice features probes to collect and transmit data to the service management part. VISION Cloud (`http://www.visioncloud.eu`) proposes a monitoring framework able to aggregate events, apply rules on them, and generate new events, representing complex system states. Cloud4SOA (`http://www.cloud4soa.eu`) proposes a cross-PaaS management and monitoring system for applications hosted on multiple Clouds, to ensure that their performance consistently meets expectations and Cloud resources are being effectively utilized. In terms of cross-layer SBA monitoring, Guinea et al. [5] present an integrated approach for multi-layered SBA monitoring and adaptation which is based on a variant of MAPE control loops. Gjørven et al. [4] propose a coarse-grained approach exploiting mechanisms across SCC and SI layers in a coordinated fashion to support both monitoring and adaptation. All these related approaches do not consider all layers (Cloud and SOA) as well as Multi-Cloud setups (see [3] for an overview), while their main target is on non-functional properties. Our approach's main strength is that it deals with both service and Cloud-based applications while considering challenges raised in a Multi-Cloud environment.

6 Conclusions and Future Work

We have presented a cross-layer monitoring framework for Multi-Cloud SBAs. The framework integrates monitoring mechanisms within each Cloud layer and

across Cloud providers. Our architecture uses an event and a component model (not analyzed due to space limitations) to describe monitored events and their source Cloud components. Evaluation of the cross-layer monitoring framework in different deployment settings shows that TSDB performance scales with the number of storage servers and minimally impacts a Multi-Cloud setup. Our next step is to complete developing the adaptation engine and performing larger-scale end-to-end Multi-Cloud experiments involving long-running SBAs.

Acknowledgements. We thankfully acknowledge the support of the PaaSage (FP7-317715) EU project.

References

1. Alcaraz Calero, J., König, B., Kirschnick, J.: Cross-layer monitoring in Cloud computing. In: Using Cross-layer Techniques for Communication Systems, Premier reference source. Igi Global (2012)
2. Barbon, F., Traverso, P., Pistore, M., Trainotti, M.: Run-time monitoring of instances and classes of web service compositions. In: ICWS, pp. 63–71 (2006)
3. Baryannis, G., Garefalakis, P., Kritikos, K., Magoutis, K., Papaioannou, A., Plexousakis, D., Zeginis, C.: Lifecycle Management of Service-based Applications on Multi-Clouds: A Research Roadmap. In: MultiCloud (2013)
4. Gjørven, E., Rouvoy, R., Eliassen, F.: Cross-layer self-adaptation of service-oriented architectures. In: MW4SOC, pp. 37–42. ACM (2008)
5. Guinea, S., Kecskemeti, G., Marconi, A., Wetzstein, B.: Multi-layered monitoring and adaptation. In: Kappel, G., Maamar, Z., Motahari-Nezhad, H.R. (eds.) ICSOC 2011. LNCS, vol. 7084, pp. 359–373. Springer, Heidelberg (2011)
6. Kritikos, K., Plexousakis, D.: Semantic QoS Metric Matching. In: IEEE European Conference on Web Services, Zurich, Switzerland (2006)
7. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM 21(7), 558–565 (1978)
8. Magoutis, K., Devarakonda, M.V., Joukov, N., Vogl, N.G.: Galapagos: Model-driven discovery of end-to-end application - storage relationships in distributed systems. IBM Journal of Research and Development 52(4-5), 367–378 (2008)
9. Zeginis, C., Konsolaki, K., Kritikos, K., Plexousakis, D.: ECMAF: An Event-Based Cross-Layer Service Monitoring and Adaptation Framework. In: Pallis, G., et al. (eds.) ICSOC 2011. LNCS, vol. 7221, pp. 147–161. Springer, Heidelberg (2012)
10. Zeginis, C., Konsolaki, K., Kritikos, K., Plexousakis, D.: Towards proactive cross-layer service adaptation. In: Wang, X.S., Cruz, I., Delis, A., Huang, G. (eds.) WISE 2012. LNCS, vol. 7651, pp. 704–711. Springer, Heidelberg (2012)
11. Zeginis, D., D'Andria, F., Bocconi, S., Gorronogoitia Cruz, J., Collell Martin, O., Gouvas, P., Ledakis, G., Tarabanis, K.: A user-centric multi-PaaS application management solution for hybrid Multi-Cloud scenarios. Scalable Computing: Practice and Experience 14(1), 17–32 (2013)