Towards a Monitoring Feedback Loop for Cloud Applications

Piotr Bar, Rudy Benfredj, Jonathon Marks, Deyan Ulevinov,
Bartosz Wozniak, Giuliano Casale, William J. Knottenbelt*

Department of Computing, Imperial College London, U.K.
{firstname.lastname10, g.casale, wjk}@imperial.ac.uk

ABSTRACT

Performance monitoring is fundamental to track cloud application health and service-level agreement compliance, but with the emergence of multi-cloud deployments, it may become increasingly important also to create a feedback loop between runtime operation in multi-clouds and design-time reasoning. This is because the developer needs to acquire more information on the specific performance features of a cloud platform to better leverage its specificities.

To support this goal, we have developed a set of open source components that extract quality-of-service (QoS) data from a target Java application using JMX, aggregate it in a time-series database, and finally deliver it in a prototype Java dashboard that may be integrated in a development environment, such as Eclipse, to display either live or historical QoS data. The architecture is not only limited to collection, aggregation, and display of QoS data, but it also allows the evaluation of hierarchical queries expressed using the Performance Trees graphical language. It is our intention that this will provide a cloud-independent uniform interface for developers to specify monitoring queries. Initial evaluation suggests that Cube on MongoDB provides appropriate scalability for this application.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*Performance Measures*

General Terms

Measurement, Performance

Keywords

Monitoring, cloud computing, QoS, performance trees

1. INTRODUCTION

In recent years, there has been a major trend for deploying enterprise applications on public clouds in order to improve resource management and reduce operational costs. The lack of full control over cloud services, along with the ever-increasing complexity of applications deployed on the cloud, exposes a need for monitoring tools to help not just system administrators, but also software engineers, to better understand the performance implications of their decisions. This may be helpful, for example, to verify in the early stages of the design of a new application, how architectural choices interact with service-level agreements. We shall refer to such as system as the *monitoring feedback loop* system.

In general, the requirements for an effective monitoring feedback loop are multiple: (i) one expects the ability to acquire and report to the developer basic performance metrics for a cloud application over a time horizon; (ii) since multi-cloud environments may impose different choices of programming language and monitoring technology, e.g., inability to use JMX monitoring in Google App Engine, one needs to rely on lightweight platform-independent open data formats (e.g., JSON) for monitoring data exchange; (iii) the designer should have the ability to express arbitrarily complex monitoring queries in a language independent of the way this data will be acquired; (iv) the feedback system should have the ability to support persistence of the monitoring data in the developer working space, for example by maintaining a small database local to the development environment; (v) finally, the underlying database technology should be scalable enough to support queries on thousands of data points in few seconds across multiple time windows, for example to use this data for state space exploration. To address point (iii) in particular, we propose the Performance Trees (PT) language as a simple mechanism for non-experts to express monitoring queries in a graphical manner [8]. This approach is abstract with respect to the target cloud, therefore it can help in developing a feedback loop that adapt to multi-cloud applications. In fact, our implementation supports an arbitrary number of cloud monitoring sources.

To deal with these requirements we describe an architecture capable of extracting performance data for any Java application and present it to the user in an intuitive way. Our frontend allows for the creation of custom PTs to evaluate arbitrarily complex queries. Together our modules allow for the analysis of both live and historical data as well as ensuring the reliability and consistency of data in the system. Lastly, the platform is designed to support multiple applications and multiple clouds. Among the monitoring metrics, we are able to dynamically record offered response times of a complex application such as Apache OFBiz. We use Cube [1] as a time series database implementation to enhance performance of querying time-stamped data. The components of our architecture communicate over a publish-subscribe system, which has a significant impact on the scal-

*Giuliano Casale and William Knottenbelt are partially supported by the EU project MODAClouds (FP7-318484).

MultiCloud'13, April 22, 2013, Prague, Czech Republic.
ACM 978-1-4503-2050-4/13/04.

ability of our system. The solution we present is open to extension by adding support for offline alerts, more comprehensive statistical measures and actions executed based on certain performance conditions being met by the system.

Related work. In [8] a performance-tree based monitoring architecture is proposed, based on client side agents that retrieve statistical data. This data would then be stored in a database and used to generate a compliance report when evaluating the performance tree. Our architecture follows a similar idea, although rather than generating a compliance report, our front end will dynamically show the evaluation of a performance tree, as well as being able to plot live individual sub-graphs of the performance tree.

Similar to the workload in [4, 7], our architecture offers ease of deployment and configurability by means of distributed configuration files. We have also evaluated the Amazon CloudWatch monitoring solution. While our architecture has common functionality for plotting performance data and registering custom metrics, the use of PTs in our architecture allows for more comprehensive performance queries to be defined and custom granularity for the measurements.

2. TECHNICAL ARCHITECTURE

The monitoring architecture we have developed is composed of a back-end layer, which includes the extractor, the collector and the database, and a presentation layer.

Communication. All components of our architecture communicate using a publish-subscribe model. This is implemented with the Java Messaging System (JMS) [2] and openJMS [5] as the JMS provider. The use of a publish-subscribe mechanism allows for easy scaling and extensibility.

Extractor. The Extractor module is responsible for extracting performance metrics for applications deployed on the cloud. As we rely on the Java monitoring extensions (JMX) to extract performance data, the tool is currently only capable of monitoring Java applications. The Extractor is implemented as a publisher in the messaging system.

Collector. The Collector is responsible for aggregating all the performance data published by applications running on monitored hosts and writing this data to permanent storage. It is implemented as a subscriber in the messaging system. The Collector is also responsible for verifying if the received data conforms to type restrictions.

Database. A database is necessary to provide historical data. We used NoSQL technology due to its scalability as we have predicted a large amount of data in a real-world use-case. For example, a system administrator monitoring five metrics of one application distributed over 3 clouds, receiving data values at a rate of one per second will collect approximately 1.3M data points each day. We then considered a time series database (TSDB) to optimize the performance when querying time-stamped data. We used Cube, a simple JavaScript wrapper on top of MongoDB [3], to provide the desired TSDB functionality. Cube exposes the underlying datastore through external data collectors to which the receiver posts statistics, and evaluators which can efficiently compute derived measures on the data, and stream it to clients [6]. Unlike OpenTSDB, Cube evaluators and data collectors operate fully in JSON which greatly simplifies interfacing the PT dashboard with it. Cube provides a simple query language to retrieve events by only specifying the time range and an expression to filter events by type and field values. An additional advantage of using Cube was its

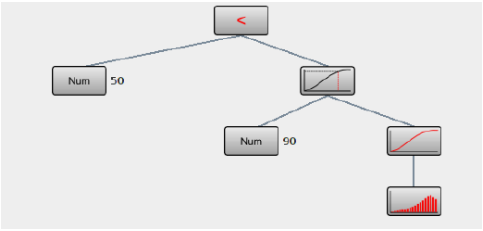

Figure 1: Performance Tree query specification editor. The query computes the 90th percentile of a response time histogram and compares it with a threshold of 50 seconds. Leave nodes specify the input metric.

in-built functionality to derive metrics and statistics from our underlying dataset, for example averages and sums.

Analyser. The analyser is the evaluation engine for the PTs. It uses the database for retrieval of historical data and subscribes to the live performance data stream for continuous re-evaluation.

User Interface. The user interface contains a Performance Tree dashboard, see Figure 1. The user can create PTs of arbitrary complexity. A graphs panel, not shown in the figure, is used to display values of performance metrics over time, history of performance query evaluation and histograms representing distributions of performance data.

3. EVALUATION AND FUTURE WORK

We have described a modular architecture which allows for comprehensive monitoring of cloud-based Java applications. We have presented how we were able to incorporate PTs as means for defining performance queries. Our initial tests suggest that the architecture scales well. For example, we have considered queries involving computing percentiles of a week of a CPU metric collected at 5 seconds resolution (approx. 200,000 data points). Updating a query on such dataset to include a new monitoring point with our architecture took, on average, just $0.584ms \pm 0.07ms$ when the query is executed alone, and $1.90ms \pm 0.12ms$ when it is run with other 6 identical queries on a 4 CPU virtual machine with 4GB of RAM hosted by Flexiscale. Future work will focus on integration with Eclipse and with tools for offline performance data analysis, for example service demand estimation that is needed for the definition of software performance models.

4. REFERENCES

[1] Cube. *http://square.github.com/cube/.*
[2] Java Message Service Concepts. *http://docs.oracle.com/javaee/6/tutorial/doc/bncdq.html.*
[3] MongoDB. *http://www.mongodb.org/.*
[4] New relic: Web app. perf. management. *http://newrelic.com/.*
[5] OpenJMS. *http://openjms.sourceforge.net/.*
[6] *https://github.com/square/cube/wiki/Evaluator.*
[7] Yahoo Finance. New relic to bring SaaS-based app performance monitoring to windows Azure developer portal. *http://finance.yahoo.com/news/relic-bring-saas-based-app-160000220.html.*
[8] W.J. Knottenbelt, N.J. Dingle and T. Suto. Performance Trees: A Query Specification Formalism for Quantitative Performance Analysis. Parallel, Distributed and Grid Computing for Engineering, Chap. 9, Saxe-Coburg, Apr 2009, 165-198.