A Performance Tree-based Monitoring Platform for Clouds

Xi Chen and William J. Knottenbelt
Department of Computing
Imperial College London
{x.chen12, wjk}@imperial.ac.uk

ABSTRACT

Cloud-based software systems are expected to deliver reliable performance under dynamic workload while efficiently managing resources. Conventional monitoring frameworks provide limited support for flexible and intuitive performance queries. In this paper, we present a prototype monitoring and control platform for clouds that is a better fit to the characteristics of cloud computing (e.g. extensible, user-defined, scalable). Service Level Objectives (SLOs) are expressed graphically as Performance Trees, while violated SLOs trigger mitigating control actions.

Categories and Subject Descriptors

C.4 [**Computing Systems Organisation**]: Performance of Systems—*Modelling Techniques*

Keywords

Cloud, Benchmarking, Performance, Modelling, Evaluation

1. INTRODUCTION

Active performance management is necessary to meet the challenge of maintaining QoS in cloud environments. In this context, we present a prototype monitoring and control framework for clouds which makes three contributions:

- We outline system requirements for an extensible modular system which allows for monitoring, performance evaluation and automatic scaling up/down control of cloud-based Java applications.
- We present a front-end which allows for the graphical specification of SLOs using Performance Trees (PTs). SLOs may be specified over both live and historical data, and may be sourced from multiple applications running on multiple clouds.
- We demonstrate how our monitoring and evaluation feedback loop system ensures the SLOs of a web application are achieved by autoscaling.

ICPE'15, Jan. 31–Feb. 4, 2015, Austin, Texas, USA.
ACM 978-1-4503-3248-4/15/01.
http://dx.doi.org/10.1145/2668930.2688063.

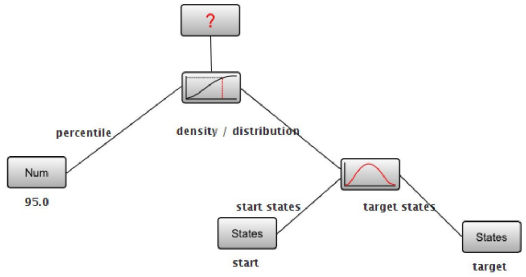

Figure 1: An Example of Performance Tree Query: "what is the 98.0th percentile of the passage time density of the passage defined by the set of start states identified by label 'start' and the set of target states identified by label 'target'?

2. SYSTEM REQUIREMENTS

In this section we present an overview of our system by discussing the requirements for an effective cloud-based monitoring platform and the techniques used to achieve them.

In-depth performance profiling. We require the ability to extract generic metrics on a per-application basis, such as CPU utilisation, memory percentage etc., as well as custom application-specific metrics [6]. This functionality is best delivered through a well-defined API.

Graphical performance query & online/offline evaluation. An important feature which distinguishes our platform from other available tools is the fact we use Performance Trees (PTs) [5, 3] for the comprehensive intuitive, and flexible definition of performance queries and evaluation, while most of the readily available monitoring tools provide users a textual query language [4]. We also incorporate support for historical trend analysis by retaining past performance data. A performance query which is expressed in a textual form as follow can be described as a form of hierarchical tree structure, as shown in Figure 1.

Extensible & Scalable. First, since multiple applications and multi-cloud environments may impose different choices of programming language and monitoring tool, a light-weighted platform independent data format (JSON) is used for monitoring data exchange. Second, all components of our system communicate using a publish-subscribe model, which allows for easy scaling and extensibility. Third, a NoSQL database is used due to ability to support large data volumes found in real-world use-cases. [1]

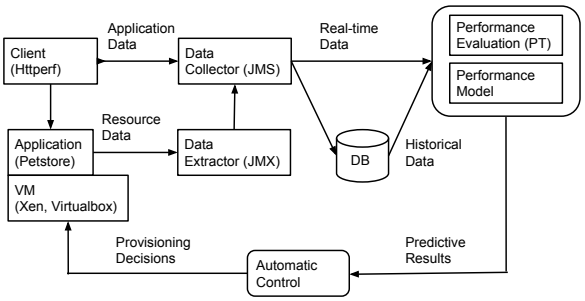

Figure 2: Testbed Architecture

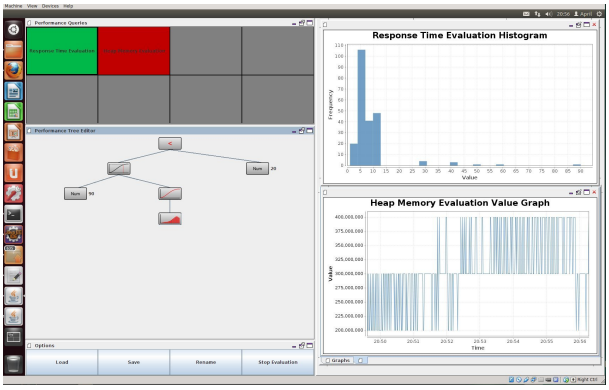

Figure 3: Performance Tree Evaluation Interface: the area on the top left is where the user can manage up 8 different PTs. Two performance evaluations are (a) Is it satisfied that 90% of response time is lower than 10 ms? (b) Is it true that heap memory is lower than 300 MB?

3. PROTOTYPE IMPLEMENTATION

Figure 2 illustrates the prototype of our system. The system is realised as a Java application capable of monitoring and evaluating the performance of a target cloud application, with concurrent scaling of the cloud system so as to meet user-specified SLOs.

The *data collector* (Java Message Service) extracts application metrics, e.g. response time and throughput. This is combined with the output of the *data extractor* (Java Management Extension), which provides hardware-related metrics, i.e. utilisation of each core of the VM, memory bandwidth, etc. The data collector can either feed this data directly into the *performance evaluator* or store it a database for future analysis. The performance evaluator evaluates metrics starting from the leaves of the PTs, and ending with the root, thus producing performance indices which are compared to target measurements for resource management. The *automatic controller* (autoscale) then optimizes the resource configuration to meet the performance targets [2].

4. DEMONSTRATION

4.1 Application and System

Oracle Java Petstore, a common HTTP-based web application, is used to expose a server to high HTTP request volumes which cause intensive CPU activity related to processing of input and output packets. The hypervisor (XenServer 6.2) is running on an a Dell PowerEdge C6220 compute server with two Intel Xeon E5-2690 8-core 2.9 GHz processors and two 1 TB hard drives. The network between each server is 10 Gbps. Each server virtual machine is assigned with on vCPU with 1 to 4 cores, 4 GB memory and one vNIC. Httperf is configured on the other servers to send a fixed number of HTTP requests rate incrementally for each Petstore instance [2].

4.2 GUI and Demo in Action

The user interface contains a dashboard where the user can manage up to 8 different PTs as shown in Figure 3. The graphical nature allows easy comprehension and manipulation by the users, and – thanks to their extensibility – new nodes can be added. The user can design their own PTs, either from scratch or by loading in a saved tree from a file. The user can specify the performance metrics, the application, and the server they want to evaluate. Once the tree has been designed and the evaluation has been started, the editor receives data from the data collector or the database, which it uses to update the square panel in the GUI. If the performance requirement is violated, this is represented by a red color applied to the evaluation box; otherwise it is green.

Figure 3 illustrates two cases: (a) the monitoring and control of a response-time-related SLO with autoscaling enabled, (b) the monitoring of a memory-consumption-related SLO with autoscaling disabled. In the first evaluation, once the PT detects the SLO is violated, the automatic controller module migrates the server to a larger instance. In this case, the server is migrated from a 1 core to a 2 core instance, so the response time decreases and the SLOs is not violated, represented as a 'green' PT block. The migration time is usually around 8 to 10 secs inside of the same physical machine. In the second case, the 'red' block illustrates the memory-consumption-related SLA is violated since autoscaling is not enabled.

5. REFERENCES

[1] P. Bar, R. Benfredj, U. D. Marks, Jonathon, B. Wozniak, G. Casale, and W. J. Knottenbelt. Towards a monitoring feedback loop for cloud applications. *Proc. MultiCloud'13*, 2013.

[2] X. Chen, C. P. Ho, R. Osman, P. G. Harrion, and W. J. Knottenbelt. Understanding, modelling, and improving the performance of web applications in multicore virtualised environments. *Proc. ICPE' 14*, pages 197–207, 2014.

[3] N. J. Dingle, W. J. Knottenbelt, and T. Suto. PIPE2: A tool for the performance evaluation of generalised stochastic Petri nets. *Proc. ACM SIGMETRICS*, 2009.

[4] F. Gorsler, F. Brosig, and S. Kounev. Performance queries for architecture-level performance models. *ICPE' 14*, 2014.

[5] T. Suto, J. Bradley, and W. Knottenbelt. Performance trees: A new approach to quantitative performance specification. *Proc. MASCOTS*, pages 303–313, 2006.

[6] Q. Zheng, H. Chen, Y. Wang, J. Zhang, and J. Duan. COSBench: cloud object storage benchmark. *Proc. ICPE' 13*, pages 199–210, 2013.