Performance Analysis of Cloud-Based Application

Peter Budai(✉) and Balazs Goldschmidt

Department of Control Engineering and Informatics,
Budapest University of Technology and Economics, Budapest, Hungary
{budai,balage}@iit.bme.hu

Abstract. Cloud computing, and IaaS cloud services in particular suit well to resource-intensive applications by offering on-demand allocation of computing power, storage space and network bandwidth together with pay-as-you-go billing system.

Typical cloud applications consist of several interdependent components, all residing on one or more dedicated virtual computers. In order to be able to accurately estimate the resource requirements of a specific component, one must carry out detailed performance analysis.

In this paper, we present the general concepts and pitfalls of performance analysis in the cloud environment. Then we present a lightweight distributed framework that is capable of generating load to and collecting performance metrics from the component instances. The capabilities of our framework will be demonstrated on a case study of the scalability analysis of a distributed MySQL relational database management system.

Keywords: Cloud computing · Performance analysis · Load generation · Test framework

1 Introduction

From an emerging buzzword that it was a few years ago, cloud computing has became a more widely accepted solution for cost-effective provisioning of computational resources. The technology behind cloud computing is not completely new, it links and wraps existing technologies such as hardware virtualization, grid computing and service-oriented architectures. There is no universally acceptable definition of cloud computing, in general we could say that it covers computing power, storage capacity and software appearing as services. There is some distinguishing features that differentiates it from the aforementioned existing technologies like self-service operation, pay-per-use billing, elasticity, and options for customization [6].

There are researches that investigate the possibility of using infrastructure (IaaS) cloud computing services for resource-intensive scientific applications that demonstrated practical examples of use [12], involved financial [7], and performance [10] analysis of the field.

I. Lirkov et al. (Eds.): LSSC 2013, LNCS 8353, pp. 476–483, 2014.
DOI: 10.1007/978-3-662-43880-0_54,

1.1 Performance Analysis in the Cloud

Being it a scientific or standard multi-tier web application, we believe that performance testing and analysis plays an important role in resource and operational cost planning. Popular public IaaS cloud providers, like Amazon [2] offer several types of virtual resources each with a different pricing. To be able to make a well established choice regarding the types and the number of virtual machines that the application requires, resource consumption must be correctly measured for its designed capacity and even beyond.

However, there are many difficulties that have to be addressed when performing performance tests for applications that are hosted on the cloud environment compared to the traditional deployment model on physical servers. In the virtualized environments the properties of the host system are usually completely hidden from the end users, and it is not possible to obtain exact performance metrics for virtualized resources like CPU and memory utilization or I/O usage, or even deduce the status of the host system from the performance metrics regarding the virtual machine.

Applications deployed into the cloud are often a subject to performance degradation caused by the lack of resources which itself is caused by the behavior of other virtual environments hosted on the same physical server, which is called the noisy neighbor problem. Typical benchmarks are performed in sterile environments and do not take this kind of background load into consideration.

The remainder of the paper is organized as follows. In Sect. 2 we introduce our test framework for performance analysis of distributed systems. In Sect. 3 we present a case study to demonstrate the applicability of our framework on a practical example. Section 4 contains an overview of related commercial and academic solutions. The paper is closed with Sect. 5 presenting our conclusions and future plans regarding the test framework.

2 Framework for Performance Testing

In this section we present a distributed test framework that aims to overcome the difficulties discussed above by allowing the test engineer to generate various types of artificial workloads on large number of remote computers simultaneously.

2.1 Architecture and Operation

The framework consists of highly autonomous software components called *agents* that are deployed on all the virtual machines involved in the test. They form a homogenous distributed framework, every target machine runs exactly the same piece of agent software. To prevent unnecessary interference with the system under test, the agents are designed to have low resource footprint when they are idle. Basically, during their lifetime the agents are performing two kind of operations, messaging and job execution, which are discussed in details below.

Messaging. The agents are exchanging information regarding their knowledge about the statuses of the active agents and the currently executing jobs in the system by sending and receiving *messages* to and from other agents. Messages can be directed at a specific agent or a job within the agent, but can be broadcasted to every agents and/or every jobs within an agent.

Job execution. The main task of the agents is to execute various types of *jobs* determined by the user. Jobs are not scheduled, they are run parallel on their own thread of execution. Jobs can be started or stopped dynamically by sending an appropriate message to the agent. A special job called *watchdog* is automatically started on all agents by default. The watchdog is responsible for periodically connecting all known other agents and keeping their status information up to date.

The agent continuously maintains a list of known peer agents based on status information obtained from other agents. Upon startup, the agent connects to an arbitrary agent on the network, signals its presence and requests peer information. By propagating their peer list to each other, eventually all agents will become aware of the complete network status, forming a fully connected network of agents. The maximum number of peer requests needed for this process is $O(n^2)$.

Suppose we have a fully connected network of N agents labelled $a_1, a_2, \ldots, a_n$ and a new agent a_0 joins the network by connecting to agent a_0 and requesting the peer information about all other agents. At this point (after one peer request) only agents a_0 and a_1 are aware of the correct network status. Now the watchdog job on an arbitrary agent a_2 starts scanning all peers known to it $(a_1, a_3, \ldots, a_n)$ one by one in unspecified order. In the worst case it takes $N-2$ requests to reach agent a_1 and obtain information about the new agent a_0. Now perform this procedure on agent a_3. This time there are 3 agents (a_0, a_1, a_2) with up to date information, thus it takes a maximum of $N-3$ peer request for agent a_3 to became up to date itself. If we repeat the procedure for all remaining agents, the number of peer request in the worst case is

$$1 + (N-2) + (N-3) + \ldots + 2 + 1 = 1 + \sum_{i=1}^{N-2} i = 1 + \frac{(N-2)(N-1)}{2} = O(N^2)$$

This number could be greatly reduced using a more sophisticated algorithm for peer status propagation – our ongoing work aims to employ an $O(\log N)$ algorithm –, but it was not in focus upon design. The test framework only requires a vertex-connectivity (κ) of 1 for correct operation, fully connected agent network only serves fault tolerance purposes in case of one or more virtual machines hosting the agents become unavailable.

After the initialization phase described above, any agent can serve as a gateway for sending control (e.g. starting and stopping jobs) or informational (e.g. querying job status) messages to other agents in the network. The agents and jobs are controlled via a command-line client interface which is built into the agent software or can be run standalone.

2.2 Implementation

The framework was implemented in Java programming language. The main reason we chose Java was the platform independence it provides, the agents running on different operating systems are able cooperate seamlessly. Java also provides an easy-to-use, lightweight remote method invocation (RMI) library, which serves as a basis of the messaging subsystem used in the test framework. Request and response messages are transferred as arguments and return values of standard RMI remote procedure calls.

Since it would be impossible to prepare for all existing test scenarios and load generators, extensibility was key concern upon designing the test framework. RMI ensures extensibility via dynamic classloading and class serialization. Test authors are able to define new jobs and message types as standard Java classes and utilize them in the test framework even without recompiling the agents.

2.3 Features and Benefits

When designing the test framework, our primary goal was to make it possible to apply artificial load on several virtual machines simultaneously and have fine-grained control over each load generator component. We distinguish two types of artificial workloads:

In-band. In-band load is caused by the normal functionality of the application being under test. Request numbers approximating or exceeding the designed capacity may cause heavy load on the application. A practical example for this is issuing a large number of HTTP requests towards a web server from many clients.

Out-of-band. This type of load is independent of the application under test, but it still affects its performance by using up shared resources on the virtual host machine. For the hypothetic web server from above, a high I/O demanding backup process run by the operating system is an example of out-of-band load.

Our test framework supports generation of both load types. It features built-in job types with either CPU-intensive or memory-consuming operations and jobs that induce high disk or network I/O traffic. Built-in job types can be parameterized which allows fine-grained control over the generated load volume. Because of their dependence on the actual application under test, there are no built-in jobs for generating in-band load, but the test framework is designed to be easily supplemented with new job and message types that can be adapted for a specific application domain.

These properties of the test framework allows us to effectively perform load and stress tests on various cloud applications as it is demonstrated in the next section.

3 Case Study — Performance Test of MySQL Cluster

In this section we present an example case study on the performance analysis of a distributed application which employs the test framework for generating in-band requests and collecting performance metrics.

3.1 MySQL Cluster

The MySQL Cluster is an open source distributed database management system, which is a development branch of the popular MySQL database. The main difference is the database engine, in the MySQL Cluster the NDB engine is used instead of the usual InnoDB or MyISAM engines. MySQL Cluster offers high availability, redundancy and increased performance for large number of parallel transactions. One of the main distinguishing features of NDB is that it stores all data in memory by default and only the transaction logs are written to the disk.

A MySQL Cluster deployment consists of three different software components which are usually deployed on separate physical or virtual servers that are interconnected via high-speed TCP/IP network links. The three software components are as follows.

Management node. This component is responsible for the entire system administration. Its task is to register and manage all the other components in the system and administer any changes in the architecture or the configuration parameters. It is required to run at configuration time but it does not have any jobs during normal operation of the cluster besides monitoring other nodes and receiving node logs.

Data node. This components stores all the actual data. A maximum of 48 data nodes can be present in a cluster. The number of servers is defined by the degree of redundancy and the amount of data that needs to be stored. For an R-times redundancy a number of $P \times R$ servers are needed, where P is a positive integer. The MySQL cluster divides the data to $2 \times P$ partitions, where every two partition is served by a *node group* containing R instances as shown on the following figure. This structure provides the scalability and high availability property of the database system, as it is able to serve SQL request when one or more data nodes are down, provided that one data node in each partition is still operational.

SQL node. This component acts as a traditional SQL server, it provides an interface for the clients. This node interprets the incoming queries, computes SQL execution plans and retrieves data from the data nodes. For load balancing, higher throughput and fault tolerance purposes this component could also be multiplied.

3.2 Configuration of the Test Framework

The performance test discussed in this section was intended to provide performance and scalability metrics for the MySQL Cluster database management system by measuring query execution times at various different system

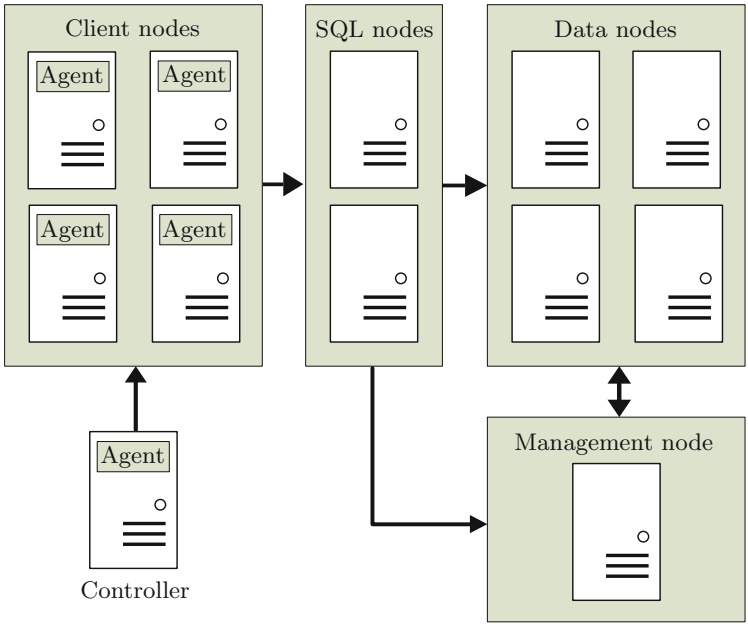

Fig. 1. Schematic configuration of the MySQL Cluster performance test showing the virtual servers hosting each software component

configurations. Figure 1 shows an overview of the test configuration, which consisted of several virtual machines hosted on the Amazon Elastic Compute Cloud (EC2) [2]. Besides the virtual machines that hosted software components of the MySQL Cluster, we have allocated multiple instances to act as database clients. We have also implemented new specialized job types for performing different types of SQL queries using the Java Database Connectivity (JDBC) API. These jobs and related message classes were included in the test framework agents that were deployed on the client nodes.

Upon initialization a sample database was created on the MySQL Cluster, then all client nodes began to execute the query jobs simultaneously. The client jobs selected a random SQL node for each query and measured the execution time of it. A total of six query types were implemented, and they were performed repeatedly by the corresponding jobs to minimize the effects of transient errors: *(a)* inserting rows; selecting rows from a single table on *(b)* indexed; or on *(c)* unindexed columns; *(d)* selecting rows from joined tables; performing aggregation functions on *(e)* a single; or on *(f)* multiple joined tables.

After the test run finished, the query execution time metrics were gathered from the agents by the controller machine for further analysis.

3.3 Results

In order to comprehend scalability behavior of MySQL Cluster and identify key factors on database performance, we have repeated the tests described above with a total of 17 different configurations of MySQL Cluster. They consisted of 8–21 virtual machines and differed in redundancy level, partition count – these two are directly affected by the number of data nodes –, and the number of SQL nodes, and each were tested with a wide range of database sizes.

The output of the whole test procedure were performance metrics for over 350 000 SQL queries, a result which could have been hardly achieved without the help of the test framework. The analysis of the resulting dataset showed that database performance scales well with server number, but not with database size. Further discussion of the results are out of the scope of this paper.

4 Related Work

There are many commercial test frameworks that are capable of load generation, however, most of them focus on web applications specifically. *AgileLoad* [1] simulates end user activity and behavior that can even be automatically captured instead of manual specification. Similarly to our solution, AgileLoad load injectors are separate software components deployed on physical or virtual machines. *LoadImpact* [4] is a SaaS solution for load testing websites, it runs entirely from the cloud and does not employ any deployable client-side software. *Keynote* [3] offers a similar service with an addition of geographically distributed load generation network.

This area is also the target of many academic research. Dumitrescu et al. developed and used *DiPerF* [9,11], a distributed performance evaluation framework with great success for performance analysis of client/server applications running on various grid computing environments. DiPerF uses an approach very similar to our framework, allowing users to submit workload generator jobs to a pool of client agents and it also provides clock synchronization between the clients, a feature which is not supported by our solution yet. However, unlike our fully distributed and homogenous test framework, DiPerF relies on a single central component to manage and control client components.

The CLIF [8] project aims to provide a generic, scalable, and user-friendly platform for performance testing. Besides load injector components similar to the jobs in our test framework, it features so called probes that can be deployed either on the workload generator or the test target system and are constantly collecting performance metrics.

5 Conclusions and Future Work

In this paper we have discussed that obtaining accurate performance metrics for virtual machines and applications hosted on an IaaS cloud service has many difficulties. Then we have presented a lightweight, distributed framework that is

capable of generating artificial load to simulate the cases that may cause performance degradation of an application, caused by either normal operation or an environmental factor unrelated to the application. As we have demonstrated on a case study, our test framework suits well to perform load testing and performance testing of practical applications.

Our ongoing work involves improving the messaging subsystem using peer-to-peer (P2P) technologies, to avoid unnecessary network communications and adding performance monitoring capabilities similarly to the idea presented by Bizenhöfer et al. [5] to offer a more complete solution for performance analysis.

Acknowledgements. The work reported in the paper has been developed in the framework of the project "Talent care and cultivation in the scientific workshops of BME". This project is supported by the grant TÁMOP-4.2.2.B-10/1–2010-0009.

References

1. AgileLoad website. http://www.agileload.com/ (2013). Accessed 15 Mar 2013
2. Amazon Elastic Compute Cloud website. http://aws.amazon.com/ec2/ (2013). Accessed 15 Mar 2013
3. Keynote Internet Testing Environment website. http://kite.keynote.com/ (2013). Accessed 15 Mar 2013
4. Load Impact website. http://loadimpact.com/ (2013). Accessed 15 Mar 2013
5. Binzenhöfer, A., Tutschku, K., Graben, B., Fiedler, M., Arlos, P.: A P2P-based framework for distributed network management. In: Cesana, M., Fratta, L. (eds.) Euro-NGI 2005. LNCS, vol. 3883, pp. 198–210. Springer, Heidelberg (2006)
6. Buyya, R., Broberg, J., Goscinski, A.: Cloud Computing: Principles and Paradigms. Wiley, New York (2010). (Wiley Series on Parallel and Distributed Computing)
7. Deelman, E., Singh, G., Livny, M., Berriman, B., Good, J.: The cost of doing science on the cloud: the montage example. In: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, SC '08, pp. 50:1–50:12. IEEE Press, Piscataway, NJ, USA (2008)
8. Dillenseger, B.: CLIF, a framework based on fractal for flexible, distributed load testing. Ann. Telecommun. **64**(1), 101–120 (2009)
9. Dumitrescu, C., Raicu, I., Ripeanu, M., Foster, I.: DiPerF: an automated distributed performance testing framework. In: Proceedings of 5th IEEE/ACM International Workshop on Grid Computing, pp. 289–296 (2004)
10. Ostermann, S., Iosup, A., Yigitbasi, N., Prodan, R., Fahringer, T., Epema, D.: A performance analysis of EC2 cloud computing services for scientific computing. In: Avresky, D.R., Diaz, M., Bode, A., Ciciani, B., Dekel, E. (eds.) Cloudcomp 2009. LNICST, vol. 34, pp. 115–131. Springer, Heidelberg (2010)
11. Raicu, I., Dumitrescu, C., Ripeanu, M., Foster, I.: The design, performance, and use of DiPerF: an automated distributed performance evaluation framework. J. Grid Comput. **4**(3), 287–309 (2006)
12. Vecchiola, C., Pandey, S., Buyya, R.: High-performance cloud computing: a view of scientific applications. In: 2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks (ISPAN), pp. 4–16. IEEE (2009)