MESSAGE TRANSFORMATION TO GAIN MAXIMUM WEB SERVER PERFORMANCE IN CLOUD COMPUTING

Sasko Ristov
Ss. Cyril and Methodius University
Faculty of Information Sciences
and Computer Engineering
Skopje, Macedonia
Email: sashko.ristov@finki.ukim.mk

Marjan Gusev
Ss. Cyril and Methodius University
Faculty of Information Sciences
and Computer Engineering
Skopje, Macedonia
Email: marjan.gushev@finki.ukim.mk

Goran Velkoski
Ss. Cyril and Methodius University
Faculty of Information Sciences
and Computer Engineering
Skopje, Macedonia
Email: velkoski.goran@gmail.com

***Abstract*—On-premise server performance depends on several parameters. Server's hardware resources, operating system (OS) and runtime environment are persistent during server and service life cycle; they provide constant performance for even server payload. This feature changes if the server migrates in a dynamic multi-tenant cloud. The server's hardware resources usually are shared among several tenants which impacts server overall performance.**

In this paper we analyze what runtime environment and OS achieves the best performance for web services in cloud PaaS layer for peak loads, particularly when the increased load happens due to huge number of small messages or by huge message sizes. We propose a middleware strategy to survive the peak loads of huge number of small messages and also a model to transform huge messages into smaller chunks and send to the server as separated sub messages.

***Index Terms*—Cloud Computing, Operating System, Performance, Virtualization, Web Services, SOAP Messages**

I. INTRODUCTION

Cloud computing is a paradigm that offers scalable and high quality resources, redundancy, elasticity and multi-tenancy. It is called as fifth generation of computing after Mainframe, Personal Computer, Client-Server Computing, and Web [1].

The concept of cloud computing reduces customers' cost. The on-demand concepts "rent whenever you need" and "pay when you rent" offers the customers to invest the money into their business rather to invest in advance for underutilized ICT equipment.

However, the overall cost is not always the key factor in business manager decisions. Cloud computing provides many benefits and detriments to business continuity. A comprehensive analysis for business information system security in cloud computing is given in [2]. Service unavailability for only several hours or even minutes can be source of costs bigger than those for IT equipment.

Implementing security often adds an overhead and outcomes with complex cryptographic operations that always degrades the service overall performance.

Faster web service response time is imperative for both the clients and the providers. A lot of proposals and solutions exist to speedup the web service response time. Algorithm transformation can highly improve the web service performance. Installing more hardware resources on web server is another solution. Cloud computing should facilitate this issue. However, both solutions add additional cost to service providers. The former costs concern additional software developer man hours. The latter costs concern additional OPEX (operating costs) for renting more instances of virtual machines or the instances with more resources and in the most of the cases additional system administrator man hours.

Our intention is to find a solution that will improve the overall performance of web services with less additional costs, or even without it if possible. The authors in [3] introduce a middleware layer implementation between the clients and the endpoint web service as a strategy to survive compute peak loads in cloud computing. Their experiments prove that although the middleware produces additional latency to overall response time, this solution provides better web service performance for compute intensive web services. This solution reduces the costs for additional hardware only during the peaks and also reduces the system administrator man hours since it automatically starts and shut downs instances with needed resources.

The middleware strategy benefits are also used in this paper. Ristov in [4] determines the totally different web server behavior and overall performance when web services are hosted on different operating systems, but on the same web server for the same input load. In this paper we analyze the benefits of middleware layer implementation between the clients and web services comparing the additional latency that middleware produces with reduced response time when the traffic is redirected on the server that provides better performance for such a load.

The paper is organized as follows. Section II overviews the installed cloud environment. We present the message transformation algorithm and new solutions to survive peak loads in Sections III and IV. The performance analysis is discussed in Section V. We conclude our work in Section VI and our plans for future work are presented in Section VII.

II. CLOUD TESTING ENVIRONMENT

The experiments are realized in cloud testing environment using OpenStack Compute project Cactus [5]. It is deployed in

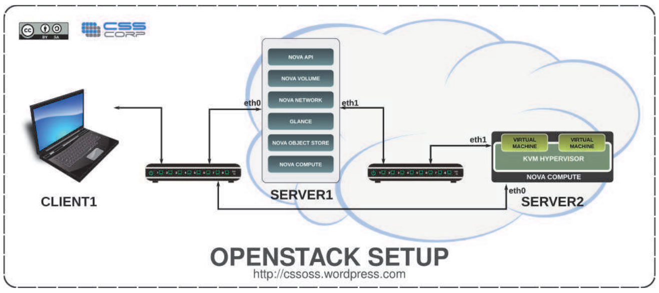

Fig. 1. Cloud Testing Environment [6]

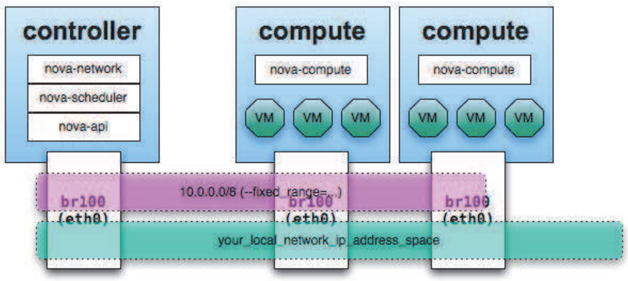

Fig. 2. Cloud Network [7]

dual node as depicted in Figure 1, i.e. two servers connected with two networks.

Server1 is Controller Node that controls the network and volumes, and schedules instances. Server2 is Compute Node that runs the instances of virtual machines. Server1 has also Compute service as a backup.

Eth0 is public network where the activated instances of virtual machine communicate with the outside world. Eth1 network interfaces are a part of the private or service network where the virtual machines communicate among each other. Network and Port address translation is used to spare the IP addresses, i.e. private IP addresses are used for a network on Eth0 interfaces although it is public network.

A. The Infrastructure

Hardware Infrastructure consists of two servers. Server1 is HP Server ML110 G6 with 4GB RAM. Server2 is Dell Optiplex 760 with 4GB RAM and Intel(R) Core(TM)2 Quad CPU Q9400 @ 2.66GHz.

The network consists of public and bridged private network as depicted in Figure 2. IP addresses of public pool are dedicated to virtual machine instances.

B. The Platform

Linux Ubuntu Server 11.04 64 bit is installed on both servers. One image of virtual machines is installed also with Linux 11.04 and another image of virtual machines is installed with Windows 2008 Server R2 Enterprise.

Apache Tomcat is installed as a runtime for web services both on the servers and in the virtual machines.

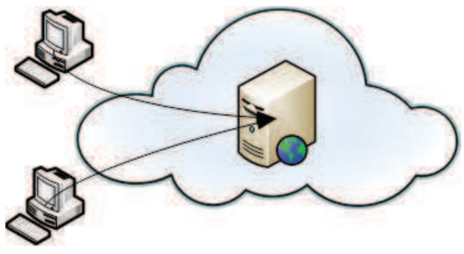

Fig. 3. The client server model in the cloud

C. The Client

SoapUI [8] is used to load web services with different message size and different number of concurrent messages.

III. The Message Transformation Algorithm

This Section presents the message transformation algorithm that can gain better performance. It also presents which web services can use this algorithm to gain better performance with less resources.

A. Traditional Client-Server Model in the Cloud

Traditional client web service server model is depicted in Figure 3. The arbitrary number of clients invoke in the same time one or more web services hosted on a web server installed in one instance of virtual machine in the cloud.

This solution is not prone to peak loads. Either web server should be underutilized during the most of the time or there will be nosedive drawback in the performance of the web service and web server. The authors in [3] propose an extension of this model for compute intensive algorithms to achieve better performance in peak loads. We use this model and extend it to gain even better performance with less resources.

B. New Client-Server Model in the Cloud with Middleware and Message Transformation

The proposed message transformation algorithm with middleware is depicted in Figure 4. Instead of renting one instance of virtual machine with more CPU and RAM resources at the beginning, we propose to rent one web server instance with minimum resources that cloud service provider offers and such that will satisfy the required performance. A middleware layer will be installed on this web server. It will receive all the requests from the customers and will forward the requests to the endpoint web service deployed in the same server. The operating system and runtime environment will be selected to provide the best performance for a nominal load as measured during the process of learning.

IV. New Solutions for Peak Loads

Besides the existing instance of virtual machine we propose a new solution that will rent additional server during the peak loads from different images with different platforms according peak load type. Peak load occurs when a high throughput is sent to the server. The high throughput can be produced by a huge number of concurrent messages or by huge messages,

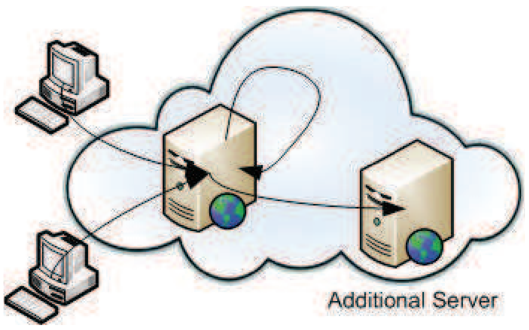

Fig. 4. The Message Transformation Algorithm

or even both. We propose a solution for each scenario in the following sections.

A. Peak load with huge number of concurrent messages

This scenario is more probable rather than the scenario in Section IV-B. Increasing the total number of users can often provide this peak. Even more, the peak is more weighty for compute or memory intensive web services. E-testing or E-voting are typical representatives of this scenario where a huge majority of users in the same time will load the web service. Only in short period of time the clients concurrently are taking the exams or they vote. Thus the web service will be overloaded with huge number of concurrent small sized messages. This scenario utilizes the web server's processor rather than occupying the memory.

Nice solution for this scenario for compute intensive web services is presented in [3], i.e. introducing a middleware layer between the clients and endpoint web service. The middleware starts and shuts down the instances if a peak load occurs.

In this paper we extend and even improve this solution. If the load of the middleware server reaches its limit then additional instance with web server will be started. Our new idea is what type of virtual machine image should be started? Ristov in [4] found that Microsoft Windows operating system provides better performance than Linux Ubuntu operating system for messages above 10KB, i.e. huge messages. Opposite, Linux Ubuntu operating system provides better performance than Microsoft Windows operating system for huge number of concurrent messages and for small messages.

Therefore, we propose the middleware web server to be installed with Linux server based operating system. Further on, we extend the solution in two directions. That is, if the number of concurrent messages increases but the messages are small sized and the performance reaches its limits, then the additional web server that should be instantiated should be also installed with Linux as it performs better for huge number of small sized messages. Otherwise, if the messages are above 10K then additional server with Windows Server based operating system should be instantiated.

B. Peak load with huge messages

This scenario does not depend directly on the total number of users but depends on the message size and type that clients send to the web service. Implementing web service security standards always increases the original message size. The authors in [9] determine the message overhead increment both for XML Signature and XML Encryption. The size of signed SOAP message with XML Signature is always greater than the original message by a constant value regardless the size of the original message. The size of encrypted SOAP message with XML Encryption increases linearly compared to the size of the original message for each message size. This scenario utilizes the web server's memory rather than the processor.

A huge size message can be provided if the message has a lot of parameters or the input parameters are huge. We focus for the latter case.

We propose a solution based on middleware strategy to improve the web service performance in this scenario. At the begging we propose the middleware layer to be implemented on front-end web server installed with Linux server based operating system. In normal mode the middleware layer forwards the requests to the endpoint deployed on the same server. If the load of the middleware server reaches it's limit then additional instance with web server will be started. If the middleware is invoked with a huge message then the middleware splits it to smaller chunks and forwards them to the endpoint web service. The middleware layer thus will create a connection to the endpoint only once to send all parts of the original request and will not cause a big latency to create a connection for each part of the original message.

This solution doesn't rent additional resources but transforms the original message to smaller chunks that web server with Linux server based operating system handles better.

V. The Performance Analysis and Discussion

This Section analyzes if our solution provides better performance than the same endpoint web service on one platform.

A. Scenario 1 - Peak load with huge number of concurrent messages

Figure 5 depicts the response time comparison for peak load with small messages of 0.2KB, for example, a message with two parameters, 3 bytes each. Both operating systems provide similar performance, Linux in front of Windows, for up to 500 messages per second. Increasing the load, Windows's performance reduces more than Linux's. For load of 1000 messages per second, Windows provides average response time of $162.74ms$ compared to Linux's $26.26ms$. Our new proposed solution produces smaller latency compared to traditional endpoint, thus implementing middleware will provide better performance than traditional endpoint on Windows for peak loads with small messages.

B. Scenario 2 - Peak load with huge messages

Figure 6 depicts the response time comparison for peak load with huge messages of 1MB. Windows operating systems

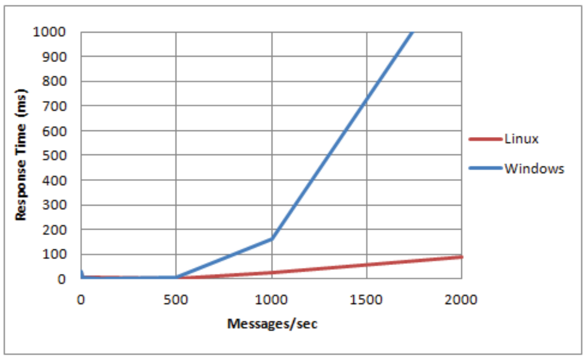

Fig. 5. Response time for peak load with small messages of 0.2KB [4]

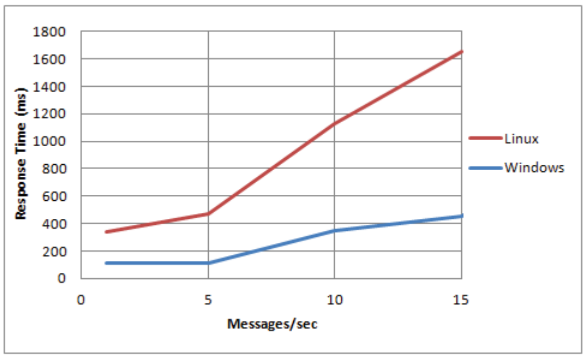

Fig. 6. Response time for peak load with huge messages of 1MB [4]

provides better performance than Linux for huge message of 1MB. The Linux's performance reduces more than Linux's increasing the number of messages per second. For example, for load of 5 messages per second, Windows provides average response time of $110.83ms$ compared to Linux's $470.76ms$.

If we compare the results of Figures 5 and 6 we can conclude that our solution for this scenario has two benefits. Those messages that will be divided and forwarded to the same web server as middleware will use the better performance that Linux provides compared to Windows for peak load with small messages. The messages forwarded to the other instance of virtual machine with Windows operating system will use the better performance that Windows provides compared to Linux for peak load with huge messages.

VI. Conclusion

This paper describes solutions for two possible peaks in web service response time, peaks that appear due to increased number of concurrent requests and peaks with increased load due to huge message size. For the former peaks we propose a middleware based solution that will dynamically instantiate and shut additional instances. The middleware should be deployed on the same machine as the endpoint web service on Linux server based operating system since it provides better performance than Windows operating systems for small load.

When peak load occurs, the middleware forwards the client requests among two endpoint web services, the first deployed on the same machine as the middleware and the other on the additional instance.

The middleware based solution for peak loads with huge message size will forward the requests from the clients to the endpoint web service deployed on the same machine. We assume that Linux server based operating system will be installed for small number of huge messages. If the response time increases beyond the threshold, then the middleware strategy will split the input parameters into smaller chunks that Linux operating system can process faster rather than the whole message. If the peak is even bigger, then the middleware will start additional instance installed with Windows Server based operating system and forwards the client requests among two endpoint web services, the whole messages to Windows Server based operating system and the messages divided into smaller chunks on Linux Server based operating system.

Therefore, the additional latency that the middleware produces will be compensated with faster response from the endpoint web services. This solution will provide better performance for both peak loads and sometimes even with smaller resources for peak load with huge messages.

VII. Future Work

In this paper we used the results in [4] to develop a middleware strategy where virtualization with ESXi is used. Our future plan is to analyze the performance of instance in the cloud with different operating systems and different web servers to develop a strategy for different platforms including operating system, runtime environment and middleware.

Another research will be performed in the area of high performance computing for loads with huge message size. There are some problems that are easily and efficiently parallelized that can provide even better performance.

References

[1] S. Rajan and A. Jairath, "Cloud computing: The fifth generation of computing," in *Communication Systems and Network Technologies (CSNT), 2011 International Conference on*, june 2011, pp. 665 –667.
[2] S. Ristov, M. Gusev, and M. Kostoska, "Cloud computing security in business information systems," *International Journal of Network Security & Its Applications (IJNSA)*, vol. 4, no. 2, pp. 75–93, 2012.
[3] S. Ristov, M. Gusev, and G. Velkoski, "A middleware strategy to survive peak loads in cloud," 2012, to be published in CiiT2012 Proceedings.
[4] S. Ristov, "Analysis of web service security and its impact on web server performance," May 2011, Master Thesis.
[5] Openstack. (2012, Feb.) Openstack compute. [Online]. Available: http://openstack.org/projects/compute/
[6] ——. (2012, Feb.) Openstack setup. [Online]. Available: http://docs.openstack.org/cactus/openstack-compute/starter/content/Introduction-d1e390.html
[7] ——. (2012, Feb.) Openstack network. [Online]. Available: http://docs.openstack.org/cactus/openstack-compute/admin/content/configuring-flat-networking.html
[8] SoapUI. (2012, Jan.) Soapui functional testing tool for web service testing. [Online]. Available: http://www.soapui.org/
[9] S. Ristov and A. Tentov, "Performance impact correlation of message size vs. concurrent users implementing web service security on linux platform," in *ICT Innovations 2011*, ser. Advances in Intelligent and Soft Computing, vol. 150. Springer Berlin / Heidelberg, 2012, pp. 367–377.