A MIDDLEWARE STRATEGY TO SURVIVE COMPUTER PEAK LOADS IN CLOUD

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*—Cloud computing offers scalable hardware resources on Infrastructure-as-a-Service (IaaS) service layer. However, instances of virtual machines (VMs) are often manually initiated by cloud clients producing degradation of service availability and service latency in peak loads.**

In this paper we propose a solution that handles the compute peak loads dynamically for web services hosted in cloud. Our solution introduces a middleware layer between clients and server. The middleware layer will instantiate additional VMs dynamically on demand as service load reaches defined minimum performance level and will forward the messages across VMs. The additional VMs will be shut down when service load returns to defined nominal value.

***Index Terms*—Cloud Computing, Performance, Virtualization, Middleware**

I. Introduction

Web services usage increases due to their advantages over other types of distributed computing architectures and benefits they provide. Microsoft defines several key benefits for software developers in [1]. Interoperability and usability are the most important ones. Standardization of the web services allows a possibility for developers to reduce learning curve for other web services following the standards. Easy deployment forces IT managers to transfer their services to web service technology.

On the other hand, web service customers want fast responses for their requests. Therefore web service performance is vital to preserve and even increase web service technology usage. There are external and internal parameters that impact the web service performance. Throughput expressed by the number of concurrent messages in a second, with their size and type are the most important external parameters that depend on customer activities. Web server hardware resources are internal parameters that depend on IT and business managers. Implementing web service security standards outcomes with message overhead and requires complex cryptographic operations for each message, thus decreasing the web service performance. The authors in [2] define quantitative indicators to determine risk of introducing web service security standards for SOAP messages for various message size and number of concurrent messages.

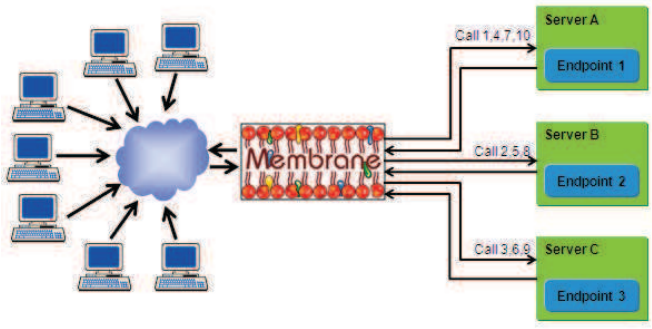

Fig. 1. Load Balancing HTTP and Web Services [3]

Web service payload is unpredictable most of the time and in most cases. Adding more hardware resources can increase overall web service performance if the number and size of the requests increases. However, the additional hardware will be underutilized for small payload. The costs will be increased due to additional power consumption as well.

We found a nice approach for Load Balancing HTTP and Web Services in [3] depicted in Figure 1. The authors also present Web Services Loadbalancing in the Amazon Cloud Using Membrane. However, the solution sets up a web services cluster in the Amazon Cloud and the cluster instances always run. Our proposed solution and strategy works with minimum necessary resources and dynamically utilize new resources in peak load.

Migrating the web services in the cloud can reduce the costs for hardware and power consumption. Cloud computing offers scalable and flexible infrastructure and platform for web service hosting. However, instancing new virtual machines with higher resources and migrating the services to new instances provides service unavailability in peak loads. In addition, another service unavailability is produced when peak loads finish and these resources will be released.

In this paper we propose a dynamic and automatic middleware strategy in the cloud that survives the peak loads. Although this model provides a small additional latency it offers great performance and security benefits. The Middle-

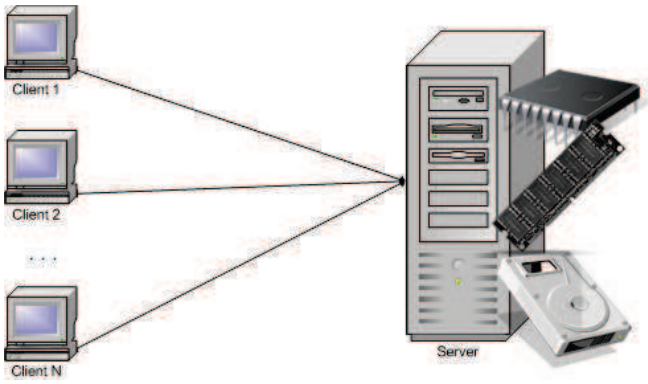

Fig. 2. Traditional web service client server model

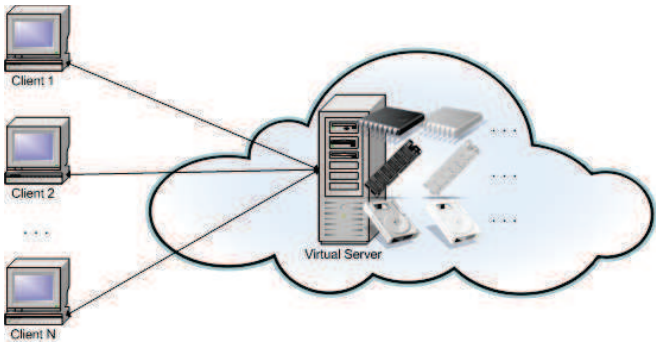

Fig. 3. web service client server model hosted in cloud

ware layer balances the requests among the existing and additional virtual machine and thus reduces the response time and increases the overall performance. It will shutdown dynamically unnecessary resources (VMs) when web service response time decreases to acceptable level.

The paper is organized as follows. Section II gives an overview of web service development from basic client server platform to modern cloud based hosting and Section III the cloud environment where the middleware is hosted. The middleware strategy is presented in Section IV. The experimental results are presented in Section V providing a proof that our middleware strategy can guarantee the constant performance to the customers. The pros and cons of this strategy are elaborated in Section VI. We conclude our work and we present the plans for future work in Section VII.

II. Background

A. Traditional Client-Server Concept

The first generation of web services are *traditional web services* which rely on client - server concept depicted in Figure 2. One or many clients request some service from one or several web services hosted on the web server. Web service can call another web service hosted on the same or other web server or can write or retrieve some data from some database server.

IT managers propose a hardware, system, network and software resources for web server according to prediction of server average load and risk management of possible peaks. A vast number of this generation servers are either underutilized for small loads and over-utilized for peak loads. The former is desired by the customers as they want the best service performance. The latter is desired by finance department as they want to cut all possible costs.

The solution for underutilization does not exist. Maybe the transfer of server's hardware resources or change the whole web server with other server will mitigate the underutilization. However, there will be some significant service downtime during hardware maintenance.

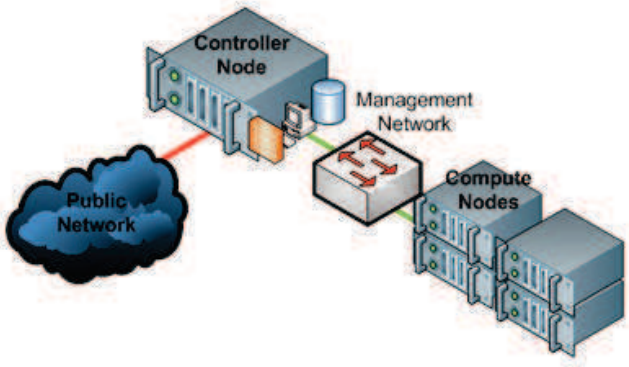

Fig. 4. OpenStack dual node deployment [5]

B. Client-Server Concept with virtualization

The second generation of web services are *virtualized web services* which also rely on client - server concept depicted in Figure 3. Virtualization solves some open issues in the traditional client server model. Web service can now easily utilize more hardware resources (CPU, RAM, HDD) for peak load or release the unnecessary hardware resources when load decreases. However, there will be still some significant service downtime for maintenance, although it will be smaller than traditional client server concept.

Cloud computing concept offers on-demand dynamic and elastic resources which improve web service availability and scalability. Nevertheless, there is a service downtime for maintenance, although it will be the smallest compared to other timings for both virtual or traditional client server model.

III. Cloud Workload Environment

This Section presents the cloud workload environment used in the experiments that include middleware layer.

A. Cloud Architecture

Cloud virtual environment is developed using OpenStack Compute project [4]. It is deployed in dual node as depicted in Figure 4. We use one Controller Node that controls the network, volumes and instances, and one Compute Node that runs the instances of virtual machines using KVM virtualization standard.

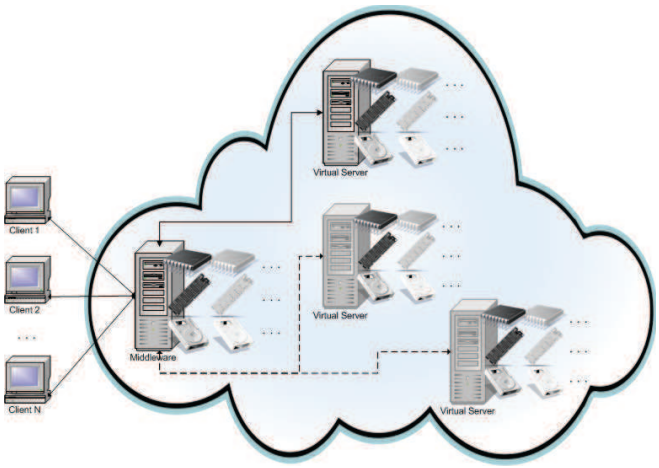

Fig. 5. middleare web service client server model in cloud

B. Hardware Infrastructure

Hardware Infrastructure consists of two servers, one server for Controller Node and one server for Compute Node. Compute Node Server is Dell Optiplex 760 with 4GB RAM and Intel(R) Core(TM)2 Quad CPU Q9400 @ 2.66GHz. Controller Node Server is HP Server ML110 G6 with 4GB RAM.

C. The virtual cloud platform

Linux Ubuntu Server 11.04 is installed on Controller and Compute Nodes, and in the image of virtual machine. Apache Tomcat is installed as a runtime for web services both on the Controller Node and in the virtual machine image.

IP addresses of public pool are dedicated to virtual machine instances.

D. The Client

Java desktop application is developed and deployed on client side to load web services which measures the response times for both the endpoint and middleware web services.

IV. Middleware Strategy

This Section describes the web services that are used in the experiments. They are deployed in a virtual machine image and on the Controller Node.

A. Middleware Architecture

The architecture using middleware layer is depicted in Figure 5. Standard endpoint web service is deployed in one active instance of virtual machine on the Compute Node. Additional middleware web service is deployed in the same instance logically between the endpoint web service(s) and clients. New infrastructure web service is deployed in the Controller Node to instantiate / shut down the additional instance(s) of virtual machine. The additional instances can be instantiated with arbitary CPU, RAM and HDD resources.

The following sections describe closely all web services deployed in the cloud workload environment.

1) The Endpoint Web Service: The endpoint web service is developed in Java EE and communicates with clients using SOAP messages. It receives two input strings and returns to the clients the concatenated string in SOAP message.

The endpoint web service is deployed in the virtual machine. One active instance of the virtual machine is activated with deployed endpoint web service. The same endpoint web service is deployed also in the additional instance of the virtual machine that is instantiated automatically by the middleware when peak load occurs.

2) The Middleware Web Service: The Middleware is also a web service developed in Java EE and deployed in the same active instance of virtual machine as the endpoint web service. It intercepts the requests from the clients and forwards to the endpoint web service if it is in the normal mode. If the middleware works in the peak mode, then it forwards the particular request randomly either to the endpoint web service hosted on the same active instance or to the endpoint web service hosted on additional instance of virtual machine. The transitions between the two modes are described in details in Section IV-B.

After the middleware layer gets the response from some of the endpoint web services, it forwards the concatenated string to the client. The response time of the middleware layer is measured for each client's request.

3) The Infrastructure Web Service: This is also a web service in Java EE and deployed in the Controller Node. It's job is to start or shut down the additional instance of the virtual machine when it is invoked by the middleware web service.

B. The Strategy

1) Traditional Scenario: This scenario is the standard client server concept where the clients directly invoke the endpoint service. The response time is measured for each client's request for different test cases.

2) Middleware Scenario: This scenario is our new middleware strategy and has two sub scenarios: normal and peak. The flag is set for peak mode and unset for normal mode. The clients in this scenario always invoke the middleware web service instead of endpoint web service. The middleware then checks the flag to determine the mode in which it should work.

In normal mode it forwards the particular request to the endpoint web service on the same machine measuring the response time of the endpoint service. If the response time of the endpoint web service exceeds the threshold, then the middleware sets the flag that the peak mode is reached. In the same time it invokes the infrastructure web service to instantiate the additional virtual machine. The middleware forwards back the response to the client after it gets from the endpoint.

If the middleware layer realizes that a peak mode is reached, then it sends dummy request to the additional endpoint web service. It sets the flag for the peak mode when the additional endpoint web service is started, i.e. the additional instance of the virtual machine is started. Otherwise the middleware layer

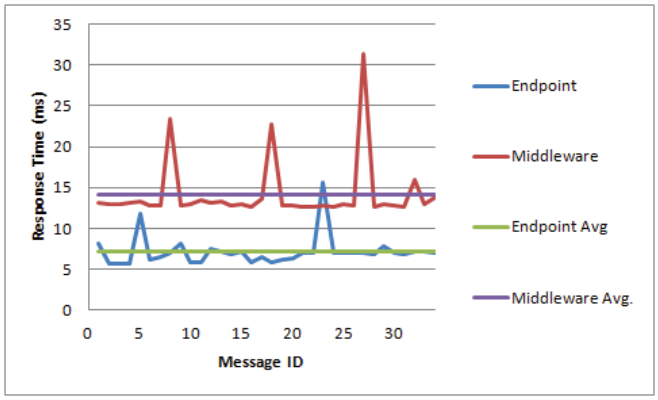

Fig. 6. Latency for simple web service after introduction of middleware

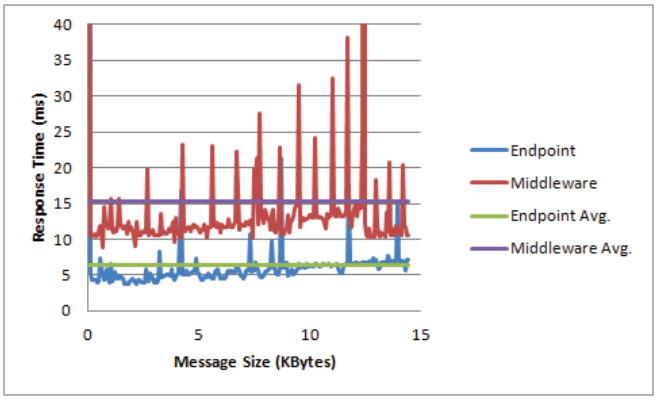

Fig. 7. Scenario comparison for middleware without load balancing

Parameter size	Traditional	Middleware	Latency	Relative
100 B	4.32	10.24	5.92	137.0%
1 KB	4.99	9.74	4.75	95.4%
10 KB	6.43	14.85	8.42	131.0%

TABLE I
COMPARING THE NOMINAL PERFORMANCE

still forwards the requests to the active endpoint web service deployed on the same machine.

If a peak mode is activated then the middleware web service uses random function to determine where to forward the requests, i.e. on endpoint web service on the same machine or to the additional for which it concludes that is already active.

C. Configuration Parameters

The threshold time in this scenario is set to $50ms$ and the flag is set in the next five minutes and after that automatically unset. The random function is set to evenly balance the forwarding to the particular endpoint service. All these configuration can be configured differently according to cloud platform environment and IT and Quality managers' decisions.

V. The Experiments and the Results

This Section presents the results of the experiments realized to determine the performance impact of the middleware introduction in the cloud environment.

A. Middleware Additional Latency

At the beginning we measure the nominal performance of traditional endpoint scenario and the additional latency that middleware produces without instantiating additional instance. The response time and their average for particular parameter size of 14.5KB in normal web service load is depicted in Figure 6.

We repeat the experiments for different parameter sizes of 100B, 1KB, 10KB and the average response time and the comparison of the two scenario are presented in Table I.

Columns *Traditional* and *Middleware* present the average response time in miliseconds for the corresponding scenario. Column *Latency* presents the latency that middleware introduces and column *Relative* the relative increase of the response time in middleware scenario compared to traditional.

Table I presents that the traditional scenario provides better performance than middleware.

B. Load Balancing Introduction

Next we analyze the web services' behavior for increased memory load, i.e. we increase the parameter size by 64B each, starting from 64B to 14.5KB.

1) Middleware without Load Balancing: In this experiment the clients invoke the middleware layer which works only in normal mode without load balancing. Figure 7 depicts the results of the experiments realized in traditional and middleware scenario without load balancing.

The average response time for traditional scenario is $5.84ms$ and for middleware scenario without load balancing is $15.03ms$. We can conclude that traditional scenario also provides better performance than middleware for increased memory load. Even more, the middleware scenario produces many peaks in response time, a lot more compared to the traditional scenario.

2) Middleware with Load Balancing and Peak Mode: In this experiment the clients invoke the middleware layer which works with load balancing in both modes. Figure 8 depicts the results of the experiments realized in traditional and middleware scenario with load balancing.

The average response time for traditional scenario is similar to previously experiment, i.e. $5.78ms$ and for middleware scenario with load balancing is much lower than middleware without load balancing, i.e. $11.96ms$. Traditional scenario also provides better performance than middleware for increased memory load even with load balancing. Also the middleware scenario makes many peaks in response time, much more than traditional scenario.

We can conclude that introducing load balancing in the middleware improves the overall performance, but still not enough as traditional endpoint web service.

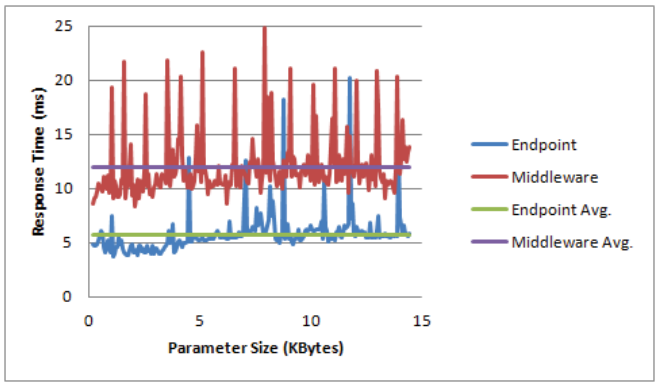

Fig. 8. Scenario comparison for middleware with load balancing

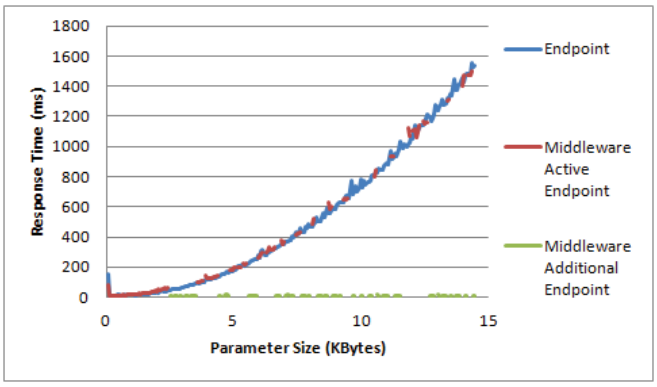

Fig. 10. Each endpoint responses

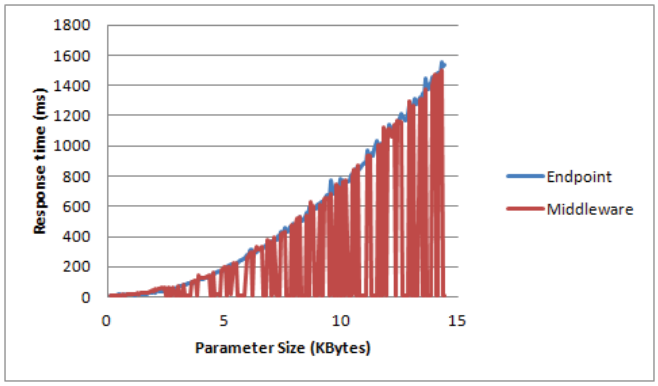

Fig. 9. Scenario with Overall Improvement introducing middleware

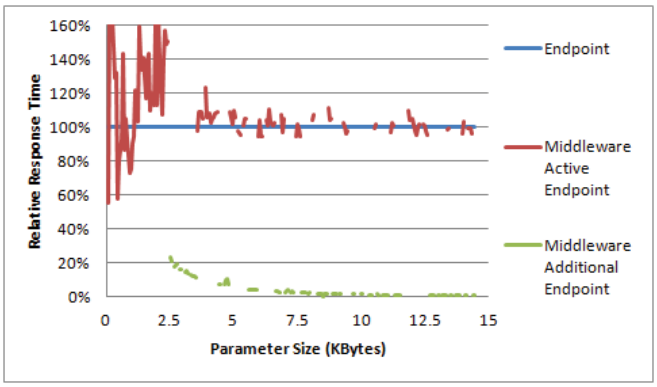

Fig. 11. Middleware relative performance

C. Why (or when) to Introduce Middleware Layer?

The experiments from the previous section show that introducing middleware with load balancing for string concatenation provides worse performance compared to traditional client server concept. Then why to introduce the middleware strategy when it degrades the performance? Or maybe the more important issue is if there is any kind of web services and some payload such that introducing the middleware and additional resources will improve the overall web service performance?

We set a hypothesis that the additional latency that middleware produces can be compensated if the web service and its payload utilizes the server's processing unit, i.e. a huge part of the response time is due to the execution of web service methods rather than the invoke itself (the case for compute intensive calculations where computational time is dominant in comparison to the communication).

For this purpose in the string concatenation example, we develop another endpoint web service that sorts the input strings before concatenation. Figure 9 depicts the results of these experiments. Introducing middleware layer for this web service reduces average latency improving the overall performance compared to the same traditional endpoint web service.

Figure 9 clearly depicts that for this scenario the overall response time increases for bigger parameter size. Also middleware response time has two different response points, i.e. one similar to the traditional endpoint web service and the other is small response time.

Figure 10 depicts the responses from the traditional endpoint and from each middleware endpoint. The active middleware endpoint provides similar performace as traditional endpoint web service. The additional middleware endpoint response time is much better than the middleware endpoint on the active instance because the both the tradtional and middleware endpoint web services are deployed together on the same instance.

Figure 11 depicts the relative response time for each middleware endpoint. Traditional endpoint response time is depicted as 100%.

The middleware works in two modes for different workloads. Increasing the message size from 64B to 2432B the middleware web service work in normal node, i.e. only middleware active endpoint. For greater message size the additional middleware endpoint web service is activated and the workload is balanced between the active and the additional middleware endpoint web services.

The middleware provides 13.87% worse performance than traditional endpoint due to additional layer in normal mode. In peak mode the active middleware endpoint provides similar performance as traditional endpoint, but additional middleware

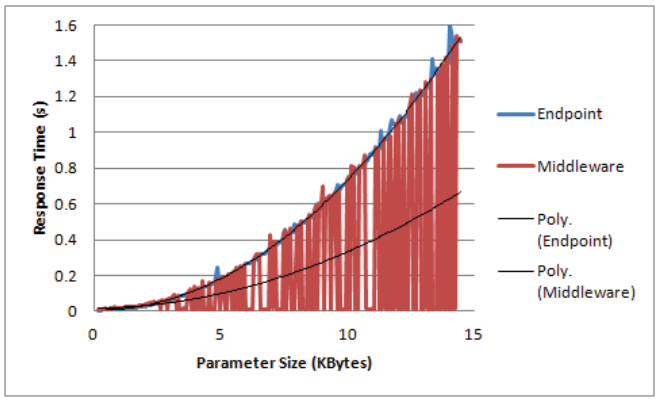

Fig. 12. Better performance introducing middleware

endpoint provides much better performance.

VI. Pros and Cons

This Section presents the pros and cons of our proposed middleware strategy.

A. Middleware Cons

The main deficiency of the middleware strategy is introducing additional latency in overall response time. The results show that introducing middleware layer even doubles the response time compared to traditional endpoint web service. We can conclude that introducing middleware achieves bad performance for web services with small response time comparable to additional latency that middleware produces. This happens for web services where communication costs are dominant in comparison to the computational.

Another deficiency is the middleware layer bottleneck which is not examined in this paper.

B. Middleware Pros

Several benefits can be achieved from introducing a middleware layer between the clients and endpoint web service in a cloud environment.

1) Better Performance - Smaller Average Response Time: Our solution provides better performance for a web services when huge part of the response time is spent on web service methods execution rather than the invoke itself, such as the web service for string concatenation and sorting explained in Section V-C.

Figure 12 depicts the average response time of the two scenarios for compute intensive web services. Comparing the two trend lines we can conclude that the average response time when introducing middleware with load balancing is smaller than the traditional endpoint web service.

2) Load Balanced Server Utilization: Our solution provides load balanced server utilization for a compute intensive web services. Figure 10 clearly depicts that the two endpoint services in middleware scenario are less utilized than traditional endpoint web service. Even more, the load balancing can be reconfigured if the additional instance has less or more resources than the active instance.

3) Increased Service Availability: Introducing middleware can greatly improve the web service availability. The single point of failure in traditional endpoint web service is improved in this scenario. If the additional instance of virtual machine fails to instantiate or becomes unavailable, then the middleware web service will not forward the requests in that direction.

VII. Conclusion and Future Work

Reducing the cost and improving the performance simultaneously is the imperative for each IT and quality manager. Our strategy proposed in this paper can provide it along with cost reduction that cloud computing paradigm offers.

In this paper we propose a middleware strategy to survive compute peaks loads in the cloud environment. Despite the latency for simple web services, the experiments prove that the middleware improves the performance of commute intensive web services (where huge part of the response time is spent for service calculations). We believe that web services with implemented web service security standards, such as XML Signature and XML Encryption, are most promissing for implementation of middleware strategy.

We plan to extend our efforts to analyze the middleware behavior when a peak is performed by a huge throughput. Another direction of the research will be performance improvement for such endpoint web service where parallelism implementation is applicable.

References

[1] MSDN. (2012, Mar.) Web service benefits. [Online]. Available: http://msdn.microsoft.com/en-us/library/cc508708.aspx

[2] S. Ristov and A. Tentov, “Performance impact correlation of message size vs concurrent users implementing web service security on linux platform,” 2012, to be published in ICT Innovations 2011 Proceedings.

[3] Membrane. (2012, Jan.) Load balancing http and web services. [Online]. Available: http://www.membrane-soa.org/soap-loadbalancing.htm

[4] Openstack. (2012, Feb.) Openstack compute. [Online]. Available: http://openstack.org/projects/compute/

[5] ——. (2012, Feb.) Openstack dual node. [Online]. Available: http://docs.stackops.org/display/documentation/Dual+node+deployment