SIMULATION BASED LOAD TESTING IN WEB SERVICES

Izzat Alsmadi **and** **Sascha Alda**

Computer Information Systems Department,
Yarmouk University
Irbid, Jordan,
ialsmadi@yu.edu.jo

Department of computer science, Bonn-Rhein-Sieg University
Sankt Augustine, Germany
Sascha.alda@h-brs.de

Abstract

Testing in Web services and SOA environment can be far more distributed in comparison with testing stand alone or traditional applications. This is since such systems are composed of several hybrid components. Testing web services high level characteristics such as: security, usability, reliability, performance, etc. is extremely environment especially as the open Internet is the work yard for those web services. Focusing on performance related quality attributes, in this paper, we conducted several experiments for testing services' response based on the varying service loads. For a selected case study of several web services, load metrics such as: response time, throughput and availability are collected. Load variation was conducted using special tools to simulate virtual users at different numbers. We also used third part monitor services to collect performance related metrics. Results showed that all those tools and applications can assist in drawing a good picture of the service quality with respect to performance or its response to different sizes of loads.

Keywords - SOA, web services, load testing, performance testing, and software testing.

1 INTRODUCTION

The recent evolution of the Internet includes continuous expansion of what is called web services. Traditional software applications are offered to users to download or acquire from storage media such as: CDs, DVDs, etc. On the contrary, web services are available at their server hosts and can be called by clients or users through the network or the Internet. Users get real time access and connection to those services. This form of distribution system is continuously expanding specially with the continuous growth of the Internet. It is envisioned that in future most of software applications will be delivered to customers through web services.

Service Oriented Architecture (SOA) is a relatively new software paradigm that is continuously evolving and expanding with web services' evolution. While in Object Oriented Architecture (OOA) software design is built based on finding objects or entities from the problem and solution domains for the subject software, the core in SOA is the service. A service is user visible software functionality. This functionality is built or implemented on the server side and then called or used by users from the client side.

Software testing activities can be classified based into several perspectives. In one perspective, software testing activities can be divided into white and black box testing. In black-box testing, the software is tested based into its provided services and functionalities without looking into the software code. On the contrary, white box testing includes testing the software based on its internal structure. We think that for a large percent in the SOA web services paradigm, traditional black box testing methods are not going to change. Any web service should include some user level accessed inputs. Those exist on the client side of the web services' structure. This may make it independent of the lower level details of the overall structure. On the other hand, for white box testing, many changes are expected to occur in web services' testing in comparison with testing web and desktop applications. Web services and SOA architectures depend on several heterogeneous components and technologies. Each one of those components can be developed by a different company using different programming language and environment. In web services environment there are several major components that the system framework is composed from. Each one of those components needs to be tested alone. In addition, for each component, integration testing is important to make sure that the communication and messaging between the subject component and the rest of the framework is normal. Those components largely include: databases, web servers and their related components, server side applications, communication services and client side web services. In SOA, besides dividing testing activities into black or white box testing, or into the several components that form the architecture, testing each component separately and then testing it in integration with the others, in addition testing framework can be divided based on the SOA major testing concerns that include: testing services, security, governance, etc. In this composition, service testing represents the traditional functional.

For the users, what matter is to make sure is that the service they requested is working in their scope or environment as they expected. Unlike traditional software developments, web services are usually designed for a large spectrum of users who may use the same service in a different context. This may in principle be similar to designing bespoke or generic software applications (e.g. operating systems, databases, office applications, etc.). On the other hand, unlike those, largely standalone applications, web services have a distributed nature and service provider is in a continuous relation with service consumers. Black box or functional testing for web services then should be ideally conducted by users, service consumers, or third parties. In web services, public registers can be helpful in this regard as they record history of service invocations.

One of the important characteristics to evaluate in web services is performance. In simple words, performance measures the speed of service response for users' requests. Several terms and metrics are related to performance. In this paper our focus is on load testing. Load testing extends performance measurement in trying to see not only service response for one request, but for many users. A web service that can only handle one or two service requests simultaneously is unreliable service in the Internet open world where each service should expect a significant number of requests throughout the days, months, etc. Service should not degrade significantly when number of service requests starts to increase. In relation to this, robustness or stress testing further tests the service response upon a large volume of requests (possibly above service limit). The goal then for stress testing is not to focus on service response, rather to observe service failures. In principle, even in failures, services should not fail catastrophically (e.g. complete crash, data loss, etc.).

The rest of the paper is organized as the following: In section 2, several samples of related work to the paper subject are presented, section three described methodology, experiments conducted along with their analysis. The paper is concluded with a conclusion and possible future work.

2 RELATED WORK

A significantly large percent of papers and documents that are related to SOA and web services' testing is of technical natures published in companies' websites, discussion boards, etc. Testing in web services, cloud computing and SOA environments is relatively new especially from the research perspective where there is a somewhat gap between the academia and the industry. The industry in this regard is witnessing a larger volume of tools, applications and technical papers discussing testing in those areas. However, in this section, we will select some research papers that focused on evaluating or discussing issues related to load testing in SOA and web services.

(Thakur paper 2010)[1] Surveyed tools that can be used for performance and load testing in web services and the cloud. Of those listed, we used JMeter, a popular open source testing tools that can be used in web services and cloud testing, particularly for load or performance testing.

(Mostefaoui and Simpson 2007)[2] Paper evaluated using response time and throughput for services performance. The paper discussed some of the difficulties and inconsistencies in performance testing especially as service performance may vary based on several factors such as: the date or time (e.g. busy or free time) and also the specific functionality that is called from the service.

Schieferdecker and Apostolidis 2005 [3] paper discussed system level testing of service based applications. Scalability is one of the important system level quality attributes that was discussed in this paper. Evaluating scalability is important in testing web services to make sure the performance of services is not downgraded significantly when a large number of service users or consumers are requesting the service at the same time. This is also related to another quality attribute: robustness, where web services should be tested under stressed or abnormal environments to see their behaviour.

Srirama et al paper 2010 [4] focused on load testing for mobile based web services. They also proposed a load balancer structure to improve mobile based systems scalability. Mobile load testing can have some other dimensions to add that may cause overhead on load and performance. The two major factors in which it is expected that mobile can have more serious load related challenges are: wireless and security. Relative to the bandwidth and speed mobile users have less power in comparison with users of desktops, or laptops. In addition, encryption and other security measures that are more urgent for mobile devices, may also cause some load and performance issues in comparison with the more stable desktop and laptop devices.

Singh paper 2012 [5] discussed and analyzed some of the enterprise applications aspects such as performance and the usage of Jmeter, Apache and Microsoft Web Application Stress (WAS) tools for load testing.

Gao et al. 2011 paper [6] discussed testing challenges in cloud computing environment. Authors pointed the need for testing activities to consider the variety and differences between the different cloud computing options (e.g. Software as a Service Saas, Platform as a Service PaaS, etc.). Based on the cloud service option, testing activities, including load, performance and scalability may vary. Authors also made a distinction in testing performance and scalability in traditional distributed systems and the cloud where cloud based environment is more dynamic and scalable.

Solomon and Litoiu paper 2010 [7] evaluated performance aspects of business processes using a case study with the usage of IBM Web Sphere components (e.g. Business modeller, Business monitor, integration developer). For each service request, they measured queuing and service times. They also evaluated options to improve response time such as optimizing resources' allocation. Such resource allocation optimization needs to be dynamic and continuously monitor possible bottlenecks.

Zheng et al 2010 [8] paper contained an experiment of performance related Quality of Service (QoS) metrics for a large number of web services. Their experiment showed also that a significant number of those web services were inaccessible due to several different purposes. They showed that some response times can be large just because the process may include a large amount of data transfer. This may tempt the need to further divide this metric into several component metrics to see the source of some possible delays. In fact, other response time related metrics such as: round trip delay, execution and transaction times are used for this purpose.

3 EXPERIMENT AND ANALYSIS

It should be mentioned first that in typical scenarios load testing should be conducted on the server side and on not the client side. A certain load of users or call services is simulated and then from the server side, and based on the server interaction with those requests, several load testing metrics can be collected. The users can be real or virtual (i.e. simulated from one system). Some tools allow one machine to simulate load testing through virtual users or also called emulated browsers. Figure 1 shows a typical load testing architecture. Advanced virtual emulation tools try to emulate users' actual behavior even their frustration with possible long response time (e.g. closing the session, opening more than one session, etc.).

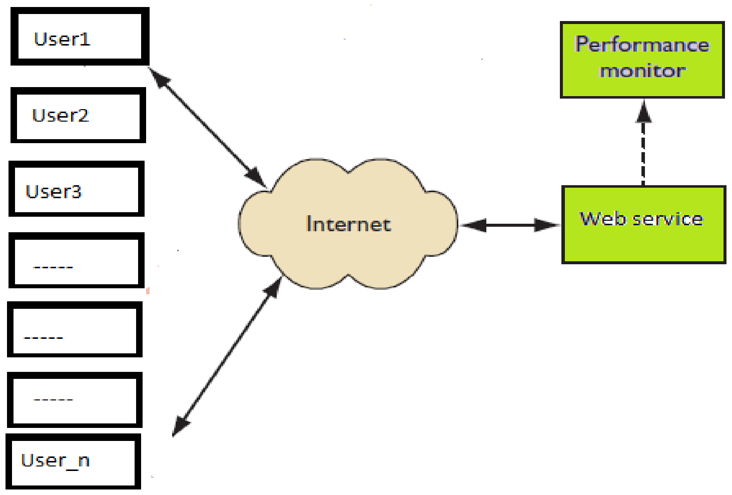


Fig. 1 A typical web service load testing architecture

However, some tools and experiments can be utilized to get some server load aspects from the client side. Users can be also real or simulated. The major difference is that performance tools can be utilized from the client side as well to measure performance metrics such as: availability, response time, throughput, etc.

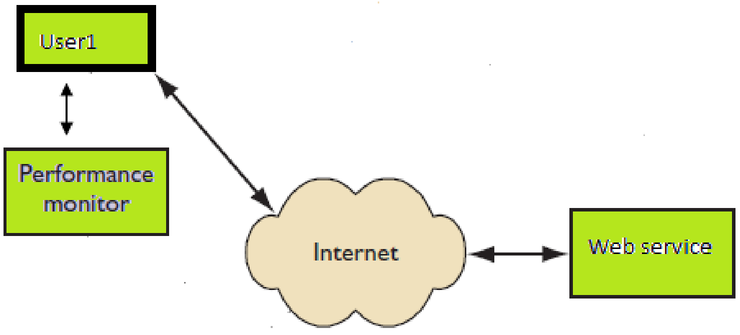


Fig.2 An alternative web service load testing architecture

3.1 Load testing tools

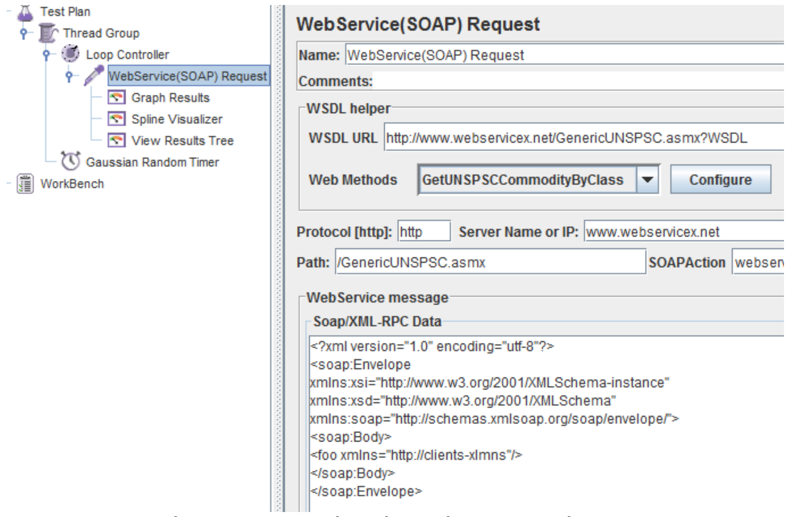
Several tools, open source, free or commercial exist to perform load testing in web applications, distributed systems, web services and alike. Each may have its own characteristics, strengths and weaknesses.
In the following experiments, we used JMeter for load testing of web services. Apache JMeter is a popular testing tool particularly for distributed systems, web applications and web services that was first released in 2001. In this specific scope, we used it for testing some important high quality attributes in software products in general and in web services in particular. Other examples of tools that can be used in high level testing of web services in general specially for load testing include: SOAPUI, LoadRunner, Rational performance studio, Silk performer, OpenSTA, Ants load, Test complete, etc. The main reasons for our usage of JMeter in particular is that since we have Apache server, synchronization and performance can be optimized with Apache JMeter in comparison with other web services' load testing tools. In addition, the ability for JMeter to simulate many or large number of threads or users is simple and straightforward. One drawback of JMeter according to some websites or papers is that it relatively consumes a large amount of resources which we did not confirm as we did not evaluate and compare efficiency related metrics such as: memory, CPU, etc. usage or consumption.

3.2 Load testing quality attributes and metrics

Those quality attributes or characteristics related to the evaluation in this experiment are: performance, load, stress and robustness. There are some common attributes that relate or connect those 4 characteristics together. This is since they all focus on testing response time and testing the interaction between service provider and consumer(s) at different load levels. In performance, the main goal is to measure the time it takes to get a service from a service provider. In its simplest form, this can measure the time from requesting a service till receiving the proper response. Of course, in each one of those characteristics several low level metrics can be collected to eventually make a judgment on those high level characteristics. Examples of performance related low level metrics include: response time, throughput, resource utilization or efficiency, availability, and workload. In this paper we focused on: throughput, response time and availability.

Throughput: This is a load metric to measure the rate of successful handled requests or message sizes. It can be measures per unit of time for either data or requests (e.g. KB/Kb/sec/min, request/sec/min). (www.owasp.org, www.w3c.or.kr, www.oasis-open.org, Zheng et al 2010, Degeler et al 2010, etc.). A high throughput value indicates a web services ability to handle many requests together (whether in parallel or sequence). Some references focus throughput definition on the amount of data processed per unit of time and assume request/time as another throughput related metric.
Response time: This metric measures the end to end time from service request to receiving the response successfully. It is measured for each service request and hence is considered as a primitive

load based metric. Related latency metrics tried to differentiate between latency sources: client, network or service.
Availability: Availability measure the amount or percentage of service up time as services are expected to be available all or most of the time.
Robustness and stress testing all focus on testing the service at high level loads or abnormal situations to evaluate the service response in those situations. While some degradation of performance is expected at high level loads, however, that should not be significant and if service fails to respond, that failure should be smooth and it should not fail catastrophically causing data, software or network damage.
In the first load test, we set up JMeter with the following parameters (loop count 1, number of threads or users 5, 10, 20, or 30, ramp up period/sec 10, loop count 100). Web services are collected from: http://www.webservicex.net-/WS/CATs.aspx?CATID=2&DESC=Business and Commerce. Notice that 100 loop count (i.e. number of execution times) means that each test will be repeated 100 times. This can simulate a stress testing for the subject web services. Figure 3 shows interface setup for JMeter load testing experiment.



WebService(SOAP) Request

Name: WebService(SOAP) Request
Comments:
WSDL helper
WSDL URL http://www.webservicex.net/GenericUNSPSC.asmx?WSDL
Web Methods GetUNSPSCCommodityByClass
Protocol [http]: http Server Name or IP: www.webservicex.net
Path: /GenericUNSPSC.asmx SOAPAction webser
WebService message
Soap/XML-RPC Data

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<foo xmlns="http://clients-xlmns"/>
</soap:Body>
</soap:Envelope>
```

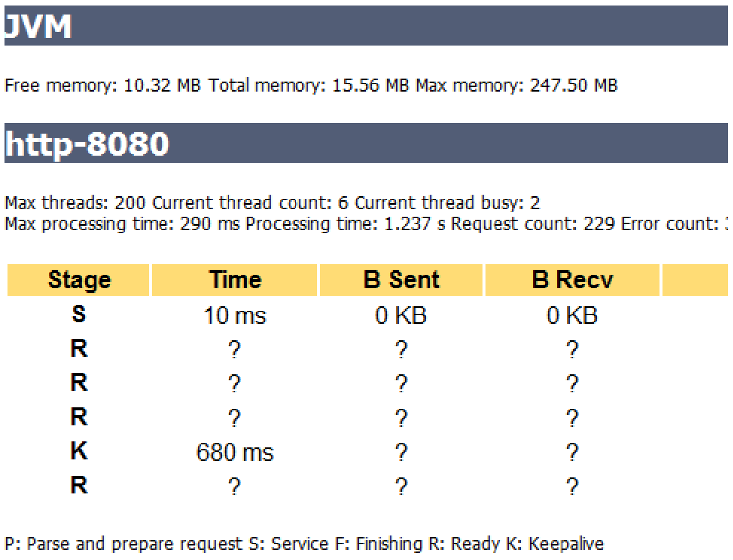
Fig. 3 JMeter load testing experiment

One good characteristic of JMeter is that it can easily simulate scenarios of several simultaneous users. This is done through changing the (number of threads or users) parameter. Table 1 shows a simple case study for evaluating several web services with changing the number of threads parameter while fixing all other parameters. Throughput represents the amount of data that the virtual users receive from the server at any given second. Numbers given in Table 1 represent the average of overall throughput measures at the end of the process. An important finding from Table 1 is that while some services may have some extreme values (e.g. FedWire service at 250 threads), however, those can't be generalized from one burst of testing. Some transient events may cause such extreme case and hence several experiments should be conducted to verify if this is a steady condition.
Results showed that comparing the 10 evaluated web services, to a large extent, all web services behave similarly with increasing the number of threads. For each tested web service, we used the first available method in the service with generic default parameters. In reality, changing the method to call and the parameters can significantly impact response time (which may need a separate experiment). In some cases, the time of testing (e.g. morning, afternoon, night) can also impact performance related metrics.

Table 1 Throughput (requests/sec) based on number of threads, using JMeter (Excerpt)

NO	service WSDL	Number of threads							
		5	10	20	30	100	250	500	1000
1	Generic-NAICS	13	24	48	74	248	572	703	838
2	Generic-UNSPSC	13	25	50	71	239	585	871	843
3	FedACH	14	24	49	73	247	586	880	883
4	FedWire	13	26	48	74	241	1425	876	886
5	OFACSDN	12	26	50	72	242	581	746	749
Average		13	25	49	72	242	647	818	821

Building a web service requires a web server. There are many performance related metrics that can be tested based on the web server. This is since this server performance can highly impact the service performance as well. Figure 4 shows a sample screen of related metrics to performance testing. Those include the memory related attributes: Free, total, and maximum, maximum threads, max processing time: processing time, request count, error count, bytes received, and bytes sent. Those are dependent on the service, as well as the machine and its related hardware or components.



JVM

Free memory: 10.32 MB Total memory: 15.56 MB Max memory: 247.50 MB

http-8080

Max threads: 200 Current thread count: 6 Current thread busy: 2
Max processing time: 290 ms Processing time: 1.237 s Request count: 229 Error count:

Stage	Time	B Sent	B Recv	
S	10 ms	0 KB	0 KB	
R	?	?	?	
R	?	?	?	
R	?	?	?	
K	680 ms	?	?	
R	?	?	?	

P: Parse and prepare request S: Service F: Finishing R: Ready K: Keepalive

Fig. 4 A server response example

Table 2 shows web services response metrics for single user call. As mentioned earlier, this may vary based on the time of usage or the specific called function, or its parameters. The relation between throughput and response time is that throughput measures for the subject web service, the number of requests it was able to handle in a unit of time (e.g. seconds or minutes), where as response time measures the time it took for sending the request to the web service till receiving it. It can then be clearly noticed that if response time is short throughput will be large as then the web service can handle more number of users.

Table 2 Web services average response (single user from ManageEngine tool) (Excerpt)

		Milliseconds		
NO.	Service	Min.	Max.	Av.
1	Generic-NAICS	151	158	154
2	Generic-UNSPSC	152	250	163
3	FedACH	151	158	154
4	FedWire	150	156	153
5	OFACSDN	150	194	156

We also evaluated using third party monitoring tools to monitor the web services under study. Figure 5 shows their response time over two hours collected from ManageEngine monitoring tool (http://www.manageengine.com). Results showed the fluctuation and the increase of response time in some cases.

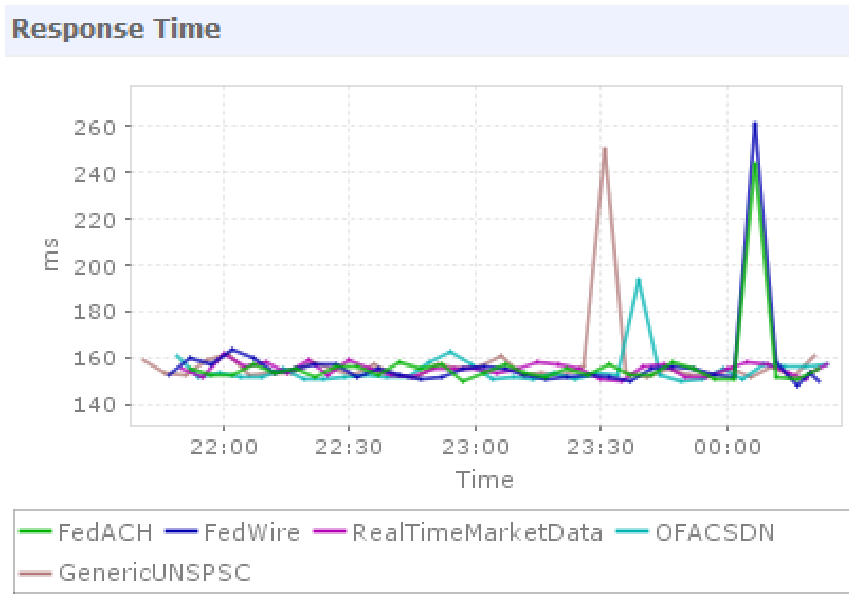


Fig. 5 Response time over two hours period

Using JMeter, Table 3 summarized the impact of increasing the number of users on response time for the 9 evaluated web services.

Table 3 Response time Vs NO. of threads, using JMeter (Excerpt)

NO	Response time (millisecond) per Number of threads							
	5	10	20	30	100	250	500	1000
1	3	3	2	2	2	3	5	14
2	2	2	2	2	2	2	4	15
3	2	2	2	2	2	2	3	11
4	2	2	2	2	2	2	9	11
5	2	2	2	2	2	4	65	54
AV	2.1	2.1	2	2	2	3.2	16.5	22

Figure 6 shows a simple diagram of average for throughput and response time versus number of requests or threads. This load test diagram shows web services' response with the increase number of users. Gaussian random timer is used to ensure random and no synchronization between virtual threads or users. Thorough study is needed to see if there is a cutting edge for number of users where throughput starts to increase rapidly which is not clearly indicated in Figure 6 as line values are not equally distributed. It is noticed in the graph that the median value for response time varies or is far-off from the average when the number of threads or virtual users is large (e.g. 250, 500, and 1000). Several papers tried to study the edge (i.e. of number of users) where response time has its sharp increase. For many services, our experiments showed that it is somewhere between 250-500 users. This is based on the default parameters for evaluated methods. Real requests may require more transferred data.

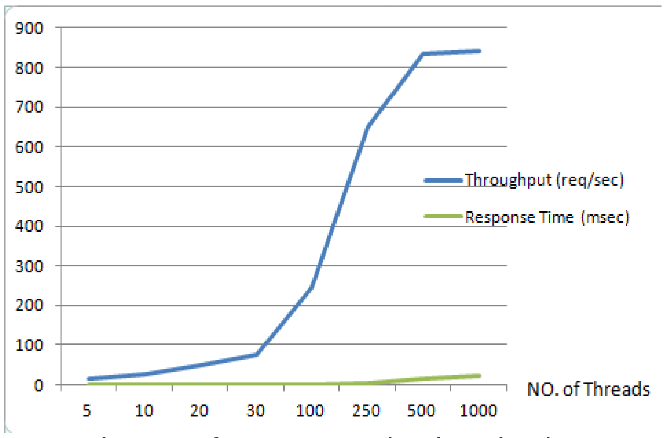


Fig.6 Performance vs. load evaluation

Availability is another important performance related quality metrics especially for web services. Services are expected to be continuously available 24/7. In the following Tables and Figures, we will show some of the metrics and assessment that can be used to evaluate web services in terms of availability. Table 4 shows a sample of the availability evaluation for the web services under study (collected from ManageEngine tool).

Table 4 Web services availability metrics (Excerpt)

Service	*MTTR*	*MTBF*	*Up-time %*
Mortgage-Index	*10:9:3 0*	*54:31*	*8.21*
Generic-UNSPSC	*5:0*	*22:3:3*	*99.6*
FedACH	*5:0*	*22:3:3*	*99.6*
FedWire	*2:41*	*22:5:21*	*99.8*

3.3 An architecture for load based testing

Experiments showed that performance and possible delay may not come only from the service itself. The end to end time for a service request-response is dependent on: the client software and hardware, the network between client and service, and the service provider software and hardware components. One metric may not successfully capture all those elements or may not be able to tell the source of delay. As such, a composition of several related metrics in which each one can measure one or more aspects of this complete structure should be implemented or collected. One metric may only draw one aspect of performance or delay and may even mislead judgments if it was taken in isolation. We assume a socket-like architecture to describe all possible factors that may impact one or more of the load based metrics for web services (Figure7). Different tools can be used to measure the different elements.

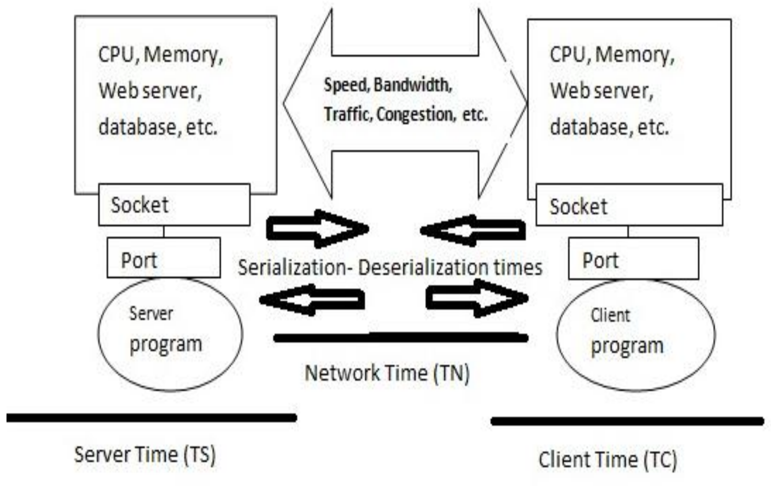

CPU, Memory, Web server, database, etc.
Speed, Bandwidth, Traffic, Congestion, etc.
CPU, Memory, Web server, database, etc.
Socket
Socket
Port
Serialization- Deserialization times
Port
Server program
Client program
Network Time (TN)
Server Time (TS)
Client Time (TC)

Fig. 7 A socket-like architecture for load based metrics

Logging mechanisms can be utilized to track service requests from end to end with time stamps. Those logs can be then analyzed to look for sources of possible delays. However, logging should be implemented carefully so that, logging itself will not be an overhead and a source or delay or latency. Experiments showed also that most load and performance related metrics focus on response without distinguishing whether the request was successfully serviced or not. For example, in Figure 7, we divided message time into three parts" Client Time (TC), Network Time (TN) and Server Time (TS). Each one of those can have its own efficiency factors and own load metrics. In addition, ignoring the fact that all those time related metrics may depend on some of the input and output parameters of the message or the service can be misleading. This is why some tools have options to generate random options for those parameters to minimize any possible bias in results. Experience from network sniffers (e.g. Snort) can be utilized and possibly classify rules to take actions in case of problems (e.g. log, alert, etc.). Serialization and de-serialization activities occur when data is travelling from or to the network. Messages may have some overhead in processing and time due to such activities.

4 CONCLUSION

Software testing is a high level generic activity that is required for any type of software regardless of its nature, development methodology, target, etc. In this paper, several experiments are conducted to collect metrics related to load and performance in web services. A small dataset of several open web services is selected. JMeter and other tools are used to collect those metrics. Analysis showed that quality attributes in general, and performance in particular are important characteristics to judge a web service in addition to its services or functionalities.

5 ACKNOWLEDGEMENT

This paper is conducted part of postdoc scholarship for AVEMPACE (Erasmus Mundus, Action 2 Strand 2 Lot 5 AVEMPACE) in 2012.

6 REFERENCES

[1] Neha Thakur, Performance Testing in Cloud:A pragmatic approach, White Paper Submitted for STC 2010, 2010.

[2] G. Koaudri Mostefaoui and A. C. Simpson. Practical experiences of testing web services. In Proceedings of the Fifth International Workshop on SOA and Web Services Best Practices at OOPSLA 2007, pages 43--58, Montreal, QC, Canada, 2007.

[3] Schieferdecker, G. Din, and D. Apostolidis, "Distributed functional and load tests for Web services," International Journal on Software Tools for Technology Transfer, vol. 7, pp. 351-360, 2005.

[4] Srirama,Satish, Eero Vainikko, Vladimir Šor, and Matthias Jarke, Scalable Mobile Web Services Mediation Framework, The Fifth International Conference on Internet and Web Applications and Services, ICIW 2010, 9 - 15, 2010 - Barcelona, Spain.

[5] Singh, Tejinder, Structural, Technically and Performance Aspects in Enterprise Applications or Projects, International Journal of Scientific and Research Publications, Volume 2, Issue 4, April 2012.

[6] Gao, Jerry, Xiaoying Bai, and Wei-Tek Tsai, Cloud Testing- Issues, Challenges, Needs and Practice, Software Engineering : An International Journal (SEIJ), Vol. 1, No. 1, Sep. 2011.

[7] Solomon, Andrei, and Marin Litoiu, Using Simulation Models to Evolve Business Processes, Proceedings of the fourth workshop on a research agenda for maintenance and evaluation of service oriented systems (MESOA 2010).

[8] Zheng, Zibin, Yilei Zhang, and Michael R. Lyu, Distributed QoS Evaluation for Real-World Web Services, 2010 IEEE International Conference on Web Services.

[9] Degeler, Viktoriya, Ilce Georgievski, Alexander Lazovik, Marco Aiello: Concept mapping for faster QoS-aware web service composition. SOCA 2010: 1-4.