PERFORMANCE ANALYSIS AND COMPARISON OF SOA SERVERS IN DIFFERENT APPLICATIONS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MACIEJ KUSZEWSKI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR

THE DEGREE OF MASTER OF SCIENCE

IN

COMPUTER ENGINEERING

JULY 2010

Approval of the thesis:

PERFORMANCE ANALYSIS AND COMPARISON OF SOA SERVERS IN DIFFERENT APPLICATIONS

submitted by **Maciej Kuszewski** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Adnan Yazıcı _____
Head of Department, **Computer Engineering**

Assoc. Prof. Dr. Ahmet Coşar _____
Supervisor, **Computer Engineering Dept., METU**

Examining Committee Members:

Prof. Dr. Faruk Polat _____
Computer Engineering Dept., METU

Assoc. Prof. Dr. Ahmet Coşar _____
Computer Engineering Dept., METU

Prof. Dr. Adnan Yazıcı _____
Computer Engineering Dept., METU

Prof. Dr. Hakkı Toroslu _____
Computer Engineering Dept., METU

Assoc. Prof. Dr. Uğur Güdükbay _____
Dept. of Computer Engineering, BILKENT

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Maciej Kuszewski

Signature:

ABSTRACT

PERFORMANCE ANALYSIS AND COMPARISON OF SOA SERVERS IN DIFFERENT APPLICATIONS

Kuszewski Maciej

M.Sc., Computer Engineering

Supervisor : Assoc. Prof. Dr. Ahmet Coşar

July 2010, 80 pages

One of the most crucial decisions when developing a system based on Service Oriented Architecture is to select an appropriate server which will be the ground for building the application. Similar to databases, an application server has significant influence on efficiency, stability, and security of entire system. During the preparation of architecture for system development one has to decide which available application server would be optimal for hosting and maintaining

Web Services in the given case. There are multiple significant criteria that lead to the proper choice. The impact on a decision among other things is type of the physical machine on which the application server is installed, estimated number of simultaneous clients, and sizes of requests and responses between clients and server. The goal for this thesis is to conduct the comparative analysis of the most commonly used application servers using Service Oriented Architecture and to determine which server should be applied in which particular cases. Performance and load tests will be conducted using SoapUI application.

Keywords: SOA, web services, application server, SOAP

ÖZ

Farklı Uygulamalar ile SOA Sunucularının Kıyaslanması ve Performans Analizi

FARKLI UYGULAMALAR İLE SOA SUNUCULARININ KIYASLANMASI VE PERFORMANS ANALİZİ

Kuszewski Maciej

Yüksek Lisans, Bilgisayar Mühendisliği Ana Bilim Dalı

Tez Yöneticisi : Doç. Dr. Ahmet Coşar

Temmuz 2010, 80 sayfa

Uygulama geliştirme ortamı olması açısından uygun sunucu seçimi, Servis Yönelimli Mimari tabanlı sistem geliştirmede en önemli kararlardan birisidir. Benzer olarak veritabanı için, uygulama sunucusunun tüm sistem üzerinde güvenlik, etkinlik ve güvenilirlik açısından çok önemli etkisi vardır. Sistem mimarisi geliştirmeye hazırlanırken, uygulama sunucusunun verilen durumdaki web hizmetlerini daha verimli şekilde barındıracak ve sürdürebilecek bir sunucu

olması gerekir. Doğru olan seçimi yapabilmek için bir çok önemli kriter vardır. Kararı etkileyen diğer konular ise uygulama sunucusunun ne tür bir fiziksel makine üzerine kurulacağı, tahmini eşzamanlı kullanıcı sayısı, sunucu ve alıcı arasındaki istem ve cevapların büyüklüğüdür. Bu tezin amacı; Servis Yönelimli Mimari kullanan en yaygın uygulama sunucularının karşılaştırmalı analizini yapmak ve özellikle belirtilen durumlarda hangi uygulama sunucusunun seçilmesi gerektiğini belirlemektir. Performans testlerinde SoapUI uygulaması kullanılmıştır.

Anahtar Kelimeler: SOA, web servisleri, uygulama sunucusu, SOAP

To My Parents

ACKNOWLEDGEMENTS

First of all I would like to thank my thesis supervisor Assoc. Prof. Dr. Ahmet Coşar for his abundant help and their prolific suggestions. I appreciate also his swift actions regarded with the issue of arranging the server on which I could conduct my experiments.

I am grateful to Assoc. Prof. Dr. Tolga Can for the all his help during my studies at Computer Engineering Department. Especially for his support with solving formal issues related with applying for the master program at Computer Engineering Department.

Lastly, I want to thank all my interviewees. Their positive attitude and willingness to participate in the study and to share their valuable experiences greatly motivated me to put forth a good work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Nowadays, the chances of a marketing success of a released product seem to be more difficult to achieve than it was a few or dozen years ago. Even if success is achieved, the subsequent product maintenance on the wave of popularity can be similarly difficult. Under the pressure of fierce competition, each company willing to be meaningful in the contemporary world, must reasonably control their spending. Only a balance between incomes and spending on new investment, wages and other costs, may allow for its continuous profitable existence [Endrei2004]. Companies in the IT industry not only try to cut down costs and to maximize use of existing technologies, but also strive to continuously offer their customers products which are more competitive and relevant to their needs.

Before taking on a given project, it must first be planned well. Apart from the decomposition of the project into individual tasks, one of the most important issues is the costs planning. One shouldn’t forget about the next stage in product life cycle, which is further maintenance of the developed system and the costs involved. In order to correctly estimate the profitability of a project, all potential costs to be faced during the project and costs associated with the continuing

operation and maintenance of the system, should be taken into account. Another cost factor, apart from staff costs, is expenses related to the infrastructure system. Assuming that the IT project uses the benefits of the Internet, it is most likely that the system designer will have to face the crucial challenge, which is to choose the appropriate application server. In fact, it doesn't matter that much, whether it is a large company that has its own data center resources, whether it is just a growing company that puts their servers into care of another external specialized company. In both of these cases, the cost of system maintenance usually depends on the actual demand on the resources of servers that support the system [Endrei2004]. For this reason, companies wishing to compete effectively with competitors, they should look for efficient solutions that will effectively lower the system maintenance costs. In this work, I wish to devote particular attention to the problem of selection of an appropriate server for systems with some particular load characteristics.

Currently in the market there are a number of available free and commercial application servers. Thorough review and justification of the choice of servers to compare will be given in the following chapters. Within the resources of the Internet it is possible to find a comparative performance tests aimed to check the quality of certain application servers. The vast majority of them, however, cover old, currently unused workloads and/or software versions. In designing the system, it is always good to use the latest versions of software, especially server software. During the maintenance phase of the system, it may be necessary to update the component, which in turn may require a newer version of software used so far. By using the latest version, any necessary change in the future, might be much easier. Another issue is the performance that most vendors are seeking to improve with each new version. Therefore, the tests used in this thesis will cover only the most recent versions of the test servers.

Nowadays more and more popular and trusted Internet systems are based on Service Oriented Architecture (SOA). This preference is because of some

advantages related with this type of architecture. SOA is no longer an experimental, uncertain, and new technology which users usually consider with a certain reserve and apprehension. According to [IBM2009], most companies have already recognized the benefits of the SOA approach. McKinsey's research[???] has shown that two-thirds of enterprise financial and insurance sectors declared in 2007 that they were involved in the implementation of this architecture. Of course, these are not the only areas where SOA can be successfully applied. This is due to many advantages that come from this architecture, i.e. greater flexibility, no need to be forced to use only one supplier, and the possibility of gradual expansion.

As it was mentioned earlier, in order to save costs and thereby increase the competitiveness of a product, one should select the appropriate application server built into the system. While the number of tests for server performance comparisons of static and/or dynamic pages such as PHP or CGI, is quite big, a competent analysis of server performance in support of SOA is very difficult to find. Particularly for studies conducted in a systematic and complete way where we have to inspect the exact configuration of environment on which the tests were carried out and the available precise description of the performed experiments. For this reason this research of application server performance will be conducted on SOA based systems.

1.1 Thesis Objective and Scope

This study will examine and compare the most popular servers that support SOA, and then determine the application server in particular cases. Performance tests will be carried out using the free application, SoapUI. It is intended to examine the performance of applications operating in a SOA environment. The software developed in this research on web services will run under different application servers. It will consist of receiving the requests sent by the load generator to the examined web service. By executing the same script and using the environment on the same physical server with various application servers, one can see the performance differences that are caused by different application servers. The test environment will consist of two physical servers. One of them will perform the task of generating service requests. In addition it will have to verify the received responses to requests that were previously sent to the examined server.

Another feature is the measurement of time which elapses between sending individual requests to the web service and reception of the corresponding response. The most important performance parameter is the average response time. Each application server will be subjected to different amounts of load, i.e. different frequency of requests generated using SoapUI . Another thing that will change is the size of a request. In the case of testing SOA applications, we are talking about the size of the SOAP messages sent to the web service. It may happen that some of the examined application servers perform better with requests of a certain size (XML payload size) than the others. One of the objectives of this work is to capture these differences and define a profile for the tested web application servers. The same applies to changes in the frequency of generated requests. Similarly, in case of a heavy load of a server, it will be checked, if the responses do not contain any errors, and if the client received a response for each request that has been sent. Differences in the behavior of servers, particularly under high, and extremely high loads may also be interesting.

The exact description of the configuration of the environment and the types of measurements that are being performed are given in Chapter 4.

1.2 Thesis outline

This thesis is organized in five chapters. The first one includes introduction part, where wide background and motivation are described. This chapter characterizes also the scope of the thesis and experiments that will be conducted.

Second chapter of thesis is focused on the background knowledge and technologies related with service oriented architecture, web services, and application servers. Being familiar with them is a crucial to understand the idea of experiments which will be conducted. Second part of this chapter contains review of literature related with the subject of the thesis.

Chapter 3 consists of detailed experimental setup. It includes description of performance tests, server parameters that will be compared, performance testing applications, test environments, measurements and test strategies. There is also the specification of physical server and description of how services interact with each other.

Chapter 4 contains all experimental results. It is divided into three parts, each for one SOA test environment. There are also brief descriptions of all tests and comments about obtained results.

Chapter 5 is the summarizing of the results from the previous chapter and subsequent conclusions. It contains also recommendations for further studies.

CHAPTER 2

BACKGROUND

2.1 Service Oriented Architecture and Web Services

The explicit definition of SOA is not an easy task. There are several different definitions, which are not always compatible with each other. One of the most common is [Newcomer2005]:

"…[an enabling] framework for integrating business processes and supporting information technology infrastructure as [loosely coupled and] secure, standardized components — services — that can be reused and combined to address changing business priorities."

In order to supplement this definition it is also necessary to describe the term of "web service", which is the basic element of service oriented architecture [Newcomer2005].

“a family of technologies that consist of specifications, protocols, and industry-based standards that are used by heterogeneous applications to communicate, collaborate, and exchange information among themselves in a secure, reliable, and interoperable manner.”

Services in SOA are modules of business or technical functionality with exposed interfaces to the functionality.

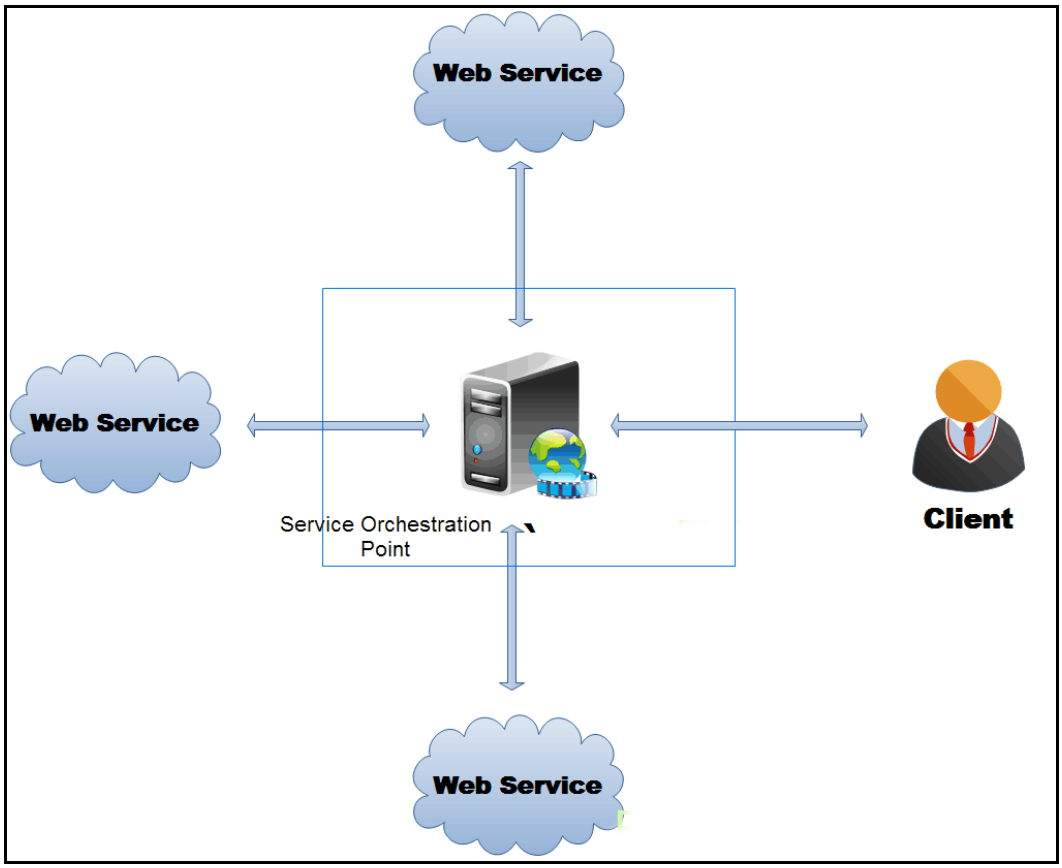

Figure 2.1 – Basic scheme of enterprise SOA-based application [Salter2008]

In other words, SOA is a way to build systems that focus on the applications as a combination of business services with other types of services. One of the objectives is to ensure flexibility and rapid reaction to any unexpected but

necessary changes. For this services must be independent from each unit and perform very specific functions. When designing a system based on SOA, it is necessary to know how the capabilities that are compliant with business requirements are organized (web service interfaces), and who and how will use these services. Properly designed architecture uses principles and a set of practices to meet business and technical requirements. At the implementation level, software architecture allows to achieve independency of the technology and then adapt it to specific technology and configuration. SOA refers to the architecture, which is a formal specification of services, their types and characteristics. It supports business processes and its relation with the whole architecture and design process [14a].

Being oriented to business processes architecture is crucial in building flexible and suitable for re-use services, and adaptation of these processes along with strategies and services for business purposes. To identify all needed services it is necessary to prepare business and information models. In addition, completely defined context of the enterprise helps to identify existing services and to assign to them responsibility for given functionalities. Service description contains information about what specific role it performs and how to use it. When all required services within the enterprise are being identified, there are three basic ways to implement them. One option is to purchase already existing services according to our requirements. The other one is to order the implementation from an external company (outsourcing). The most common is just to construct it on your own. When you choose the last option, it is advisable to use approaches such as middle-out, which usually turns out to be more practical and efficient than the typical bottom-up or top-down [14b].

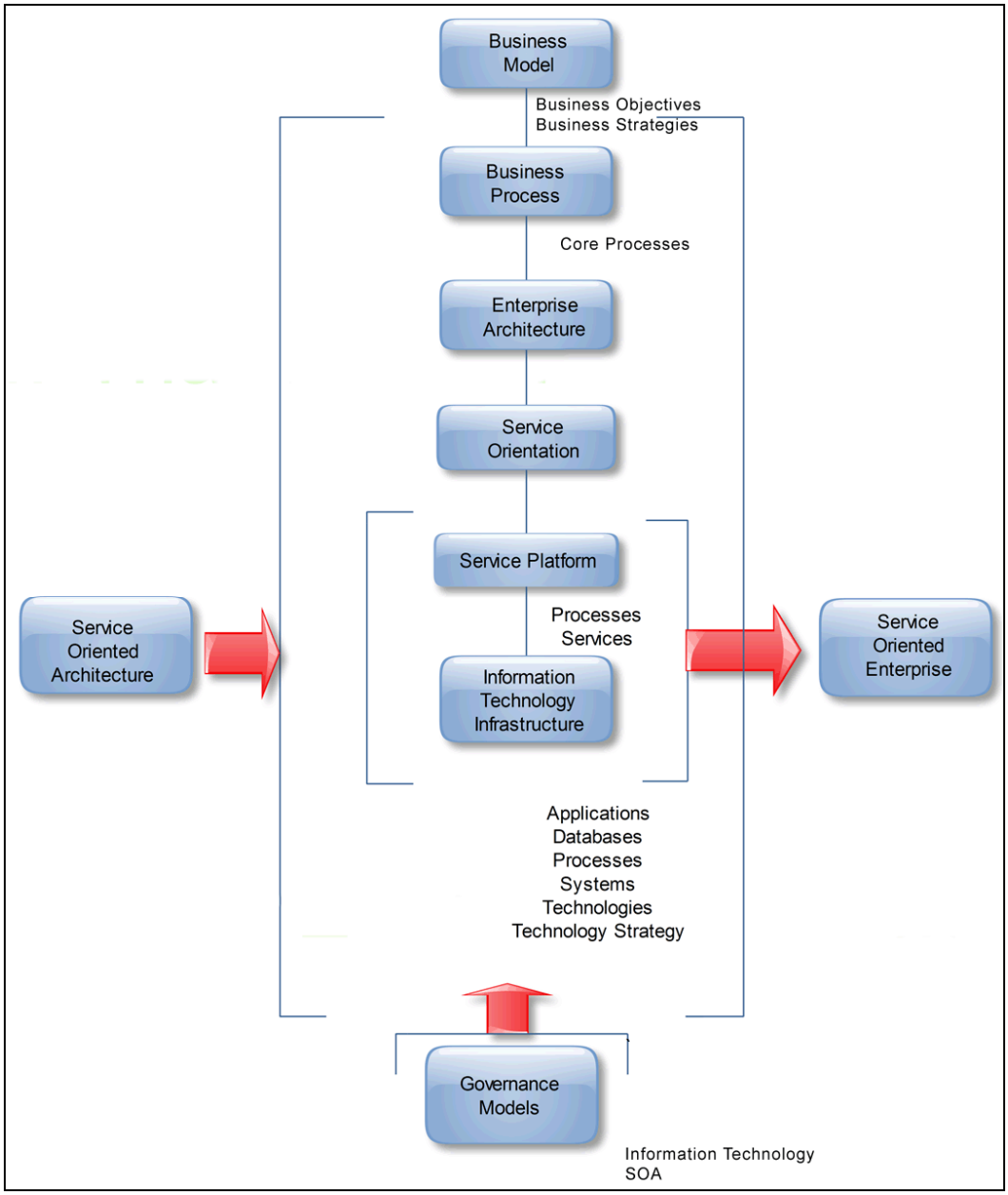

Figure 2.2 – Enterprise architecture in an SOA strategy [Lawler2008]

A project, which is based on SOA, is placed on a platform that supports Web Service. SOAP is the XML-based language used by the protocol for exchanging messages between interoperable services. Web Services Description Language (WSDL) is also an XML based standard for describing services. In particular it describes location and interfaces along with parameters of these services.

However, the standard of Universal Description, Discovery and Integration (UDDI) is used to publish services in the registry so that they can be found by any application. SOA is essentially focused on standards, which allow maintaining interoperability and independence from the technology or platform [Lawler2008].

The basic unit of communication within web services is a message. Message format is based on XML standard of SOAP. In order to transfer request and result data, SOAP uses HTTP protocol. The structure based on XML standard does not change under different operating systems or programming languages. SOAP body and header parts of the message must be included in an object named Envelope. It identifies the transmitted message as a SOAP message. In Figure 2.4 there is an example code of a weather forecast web service invocation based on a SOAP request message.

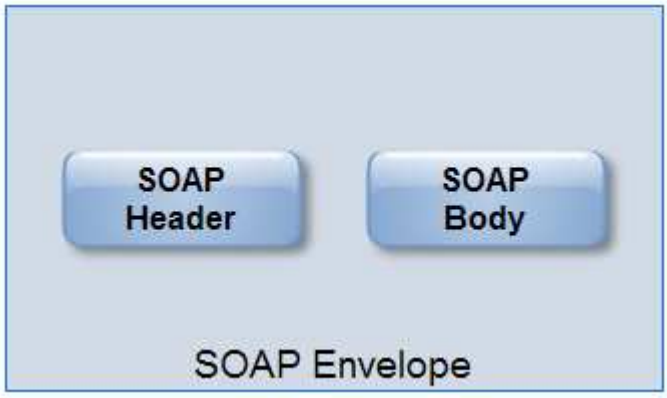

Figure 2.3 – Basic structure of SOAP message

```
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ndf="http://www.weather.gov/forecasts/xml/DWMLgen/wsdl/ndfdXML.wsdl">
    <soapenv:Header/>
    <soapenv:Body>
    <ndf:NDFDgenByDay
    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    <latitude xsi:type="xsd:decimal">39.0000</latitude>
    <longitude xsi:type="xsd:decimal">-77.0000 </longitude>
    <startDate xsi:type="xsd:date">2010-05-11T12:00 </startDate>
    <numDays xsi:type="xsd:integer">1</numDays>
    </ndf:NDFDgenByDay>
    </soapenv:Body>
</soapenv:Envelope>
```

Figure 2.4 – An example of SOAP request message with specified parameters and WSDL file

WSDL is an XML-based standard that specifies how a web service operates. It is defined essentially as a collection of interfaces with parameters. A client while having access to the WSDL file which defines a service is even not aware of actual location of the service. The exact location of web service is defined inside the WSDL file. The general idea is presented in the following diagram.

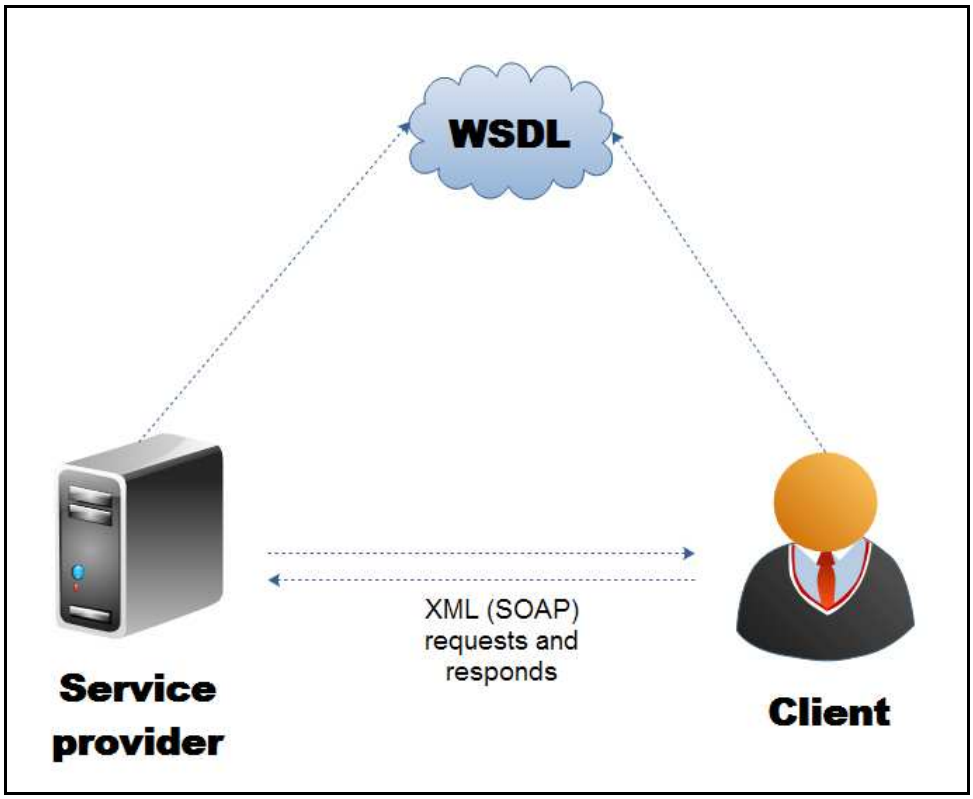

Figure 2.5 – General idea of WSDL

2.2 Application servers, Apache Tomcat, and Oracle Web Logic

Application Server is based on components. It is situated in the middle tier of web application architecture. It provides security services, maintenance applications, along with an access to services provided on this server. Application Server is usually based on J2EE (Java 2 Platform, Enterprise Edition), which uses a distributed multi-layer model. It contains client tier, middle tier and enterprise information system tier. The client tier can be a web browser or other application that uses services and data hosted on the server. Middle tier is composed mainly of http web server and EJB server. It can also be extended by further sub-levels. Enterprise information system tier (EIS) houses all existing applications along

with all files and database. The structure shows the diagram below [AppServ2010].

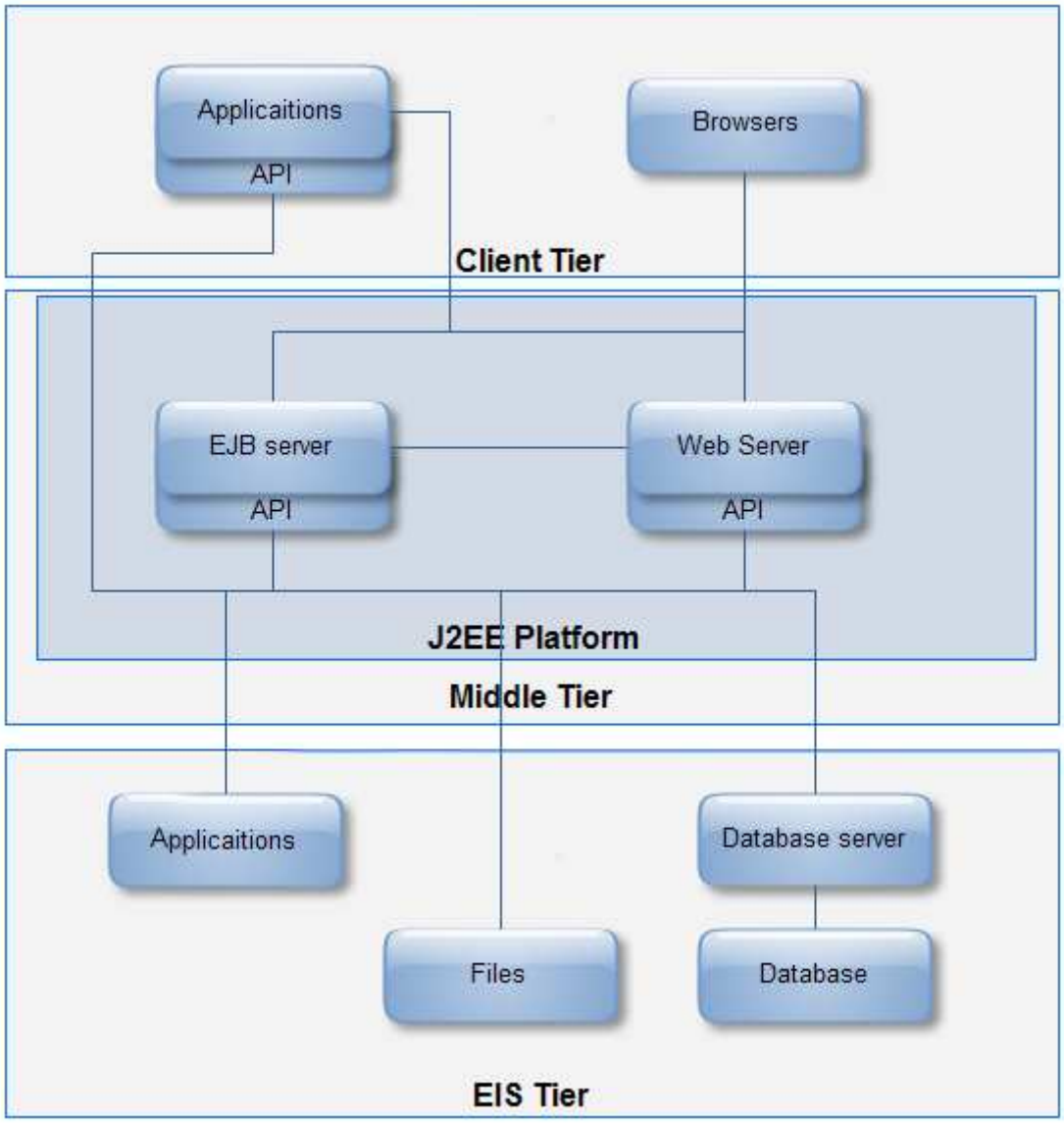

Figure 2.6 – Application server scheme [AppServ2010]

The most popular application servers are Apache Tomcat and Oracle Web Logic, which was a separate company until 2008 when it has been taken over by Oracle and was renamed as BEA Web Logic. Both servers support SOAP standard used for sending XML-based messages. This is an essential element of communication used in SOA.

Apache Tomcat uses Java HTTP Servers (Coyote). It listens for incoming requests on a particular TCP port. Then passes the requests to Tomcat Engine in order to be handled and returned directly to the client that sent the request. In addition, the Tomcat has also servlet container Catalina, which implements the Java Servlets and JSP (JavaServer Pages). Apache Tomcat is a completely free product offered under the GPL license. This application server is constantly being developed by the community of Apache Software Foundation. A very important thing related with this development process is having collaboration between developers and users, which notify about potential vulnerabilities and bugs. This applies to either stable or beta versions of Apache Tomcat application server.

Apache Tomcat Architecture is composed of a series of components, interacting in accordance with the predefined rules. The structure of the server installation is defined in the XML file. Tomcat allows also the capability of clustering. For this purpose there are different types of data replication. These are replication of session amongst server clusters, replication of context attributes and deployment via WAR files. The latter provides the same application running on all available clusters. The other two maintain an open session on the different clusters and access to attributes on all the servers [Zambon2007].

Oracle Web Logic is a scalable, Java EE based application server. It is a very powerful system made up of many components. Infrastructure provides

deployment of many types of distributed applications and is designed specifically for service-oriented architecture (SOA). The complete implementation of the Java EE specification provides a set of APIs for creating distributed web applications with access to various network services. The end user obtains the access to these applications through a web browser or other client applications. Oracle Web Logic also supports the Spring Framework. In case of building applications, which have to handle heavy traffic, it offers the capability of clustering, in order to distribute the load within particular nodes. This can be also used as a protection against data loss in case of hardware failure. Oracle Web Logic application server has also many diagnostic tools that allow system administrators to monitor in detail applications that are deployed on the server. Oracle Web Logic also offers powerful server management capabilities such as a very advanced administrative panel [OracleCorp2009].

Another very popular application server is GlassFish, which includes web services implementation according to Java EE 5 specification. Nevertheless the engine of GlassFish application server is based on Apache Tomcat. That is why GlassFish will not be taken into further performance comparison. To justify this decision has been conducted a load test for the SOA environment described in subsection 3.4.3. Load was equal to 5 threads generating constantly requests to the service provider.

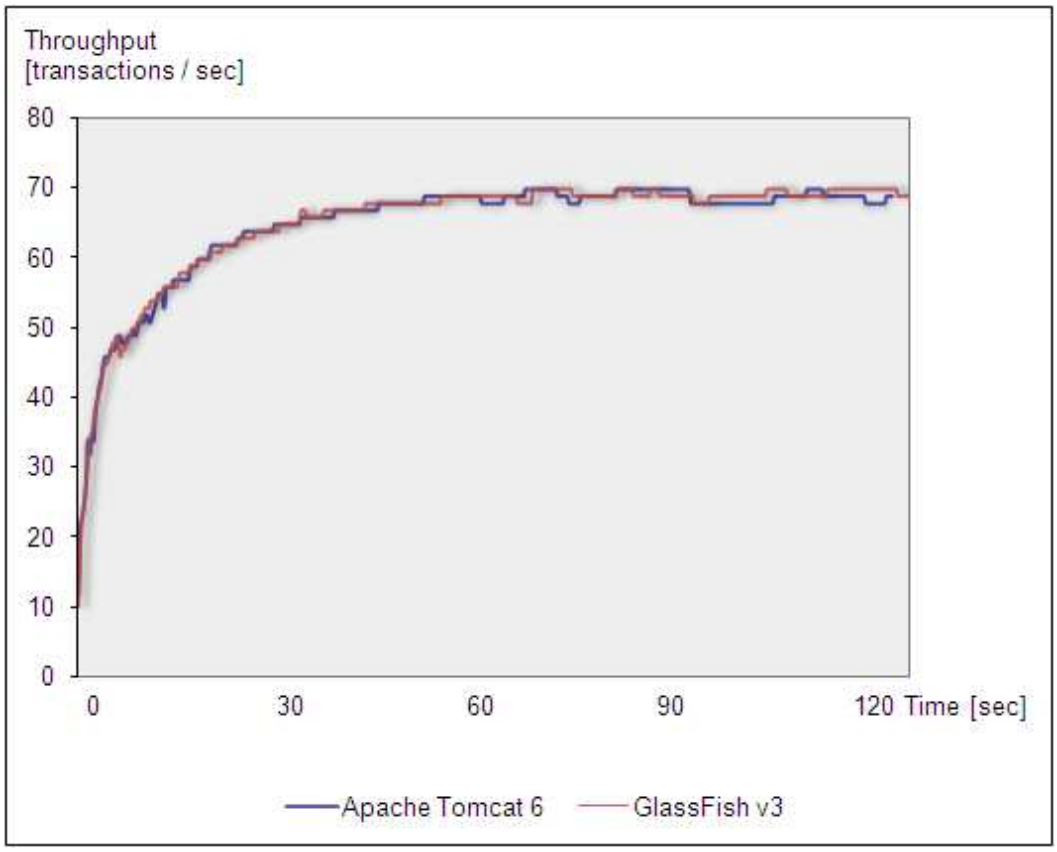

Figure 2.7 – Test results for 5-thread load test

The results only confirm that it would be better to focus on Oracle Web Logic and Apache Tomcat application servers, rather than include also GlassFish V3.

2.3 Related works

At today's pace of Internet technologies development each subsequent year brings many new versions of popular application servers or emergence of totally new products. Recently quite common is also taking over a company by another one and offering old products under new name. One can find a lot of researches

and comparisons on the performance of pure http servers, however most of which is no longer developed. One example might be the tests performed by Jef Poskanzer (ACME Labs) [Poskanzer2010]. They refer to the typical http servers, but unfortunately they are not very detailed and extensive, which may lead to incorrect conclusions. There is a lack of tests for different cases and configurations that could simulate the server work on real users. Most web servers, which are subject to testing, including the Apache version 1.3 are in general no longer used. Experiments were conducted under Solaris operating system. The results showed significant differences in performance between the Apache 1.3.0, and victorious Zeus 3.1.4 server, which was faster to use small files even up to 320%.

Further tests [Zeuscat2010] conducted by Andrew Ho focus on comparison of two versions of the Apache HTTP 1.3 and 2.0. In addition, there is also thttpd server included. Tests, which were carried out differ from each other mainly by degree of server load. Each test is based on ramp-up load injection profile, which progressively increases the number of threads that generate requests. The author analyzes each test in terms of average throughput for a specific range of threads, and latency. A single test takes 30 seconds. The tests explicitly indicated thttpd as a winner. The slowest HTTP server was Apache 1.3.

In more recent studies [LitespeedTech2010] there is performance comparison of different versions of popular Web servers (Apache, thttpd, Lighttpd, LightSpeed). Used tool for testing was the Apache Benchmark. Generated requests regards to small static files, dynamically generated pages PHP, CGI, Fast CGI, or a simple Perl script. The test results explicitly showed that the fastest server was LiteSpeed. However, because of the fact that they were published at the LiteSpeed official website, it may be better to perceive them with some degree of

skepticism and distrust. Nevertheless description of the tests, client and server environments, have been prepared very accurately and in detail.

Another similar performance test has been conducted by Sun [SunJava2010]. Sun Java System Web Server 6.1 and Apache Tomcat 2.0 was compared. The latter server turned out to be a bit slower. Apart from the load test which consisted of static and dynamic requests (JSP), also different kinds of experiments were conducted. Amongst the tests the characteristic of server behavior under very heavy load was analyzed as well. In this kind of tests, the difference was significant. As a result Apache Tomcat was generating many more error responses than the Sun Java System Web Server. On average Apache Tomcat was generating 24 times more errors than Sun Java System Web Server. The greater the load was and the longer the test ran, the difference was greater as well. Also in case of Apache Tomcat, error responses were occurring much earlier (lower load) than in the Sun Java System Web Server.

Other types of tests on the web servers performance can be find in articles "Linux Is The Web Server's Choice"[Nichols2010] and "Linux Up Close: Time To Switch" [LitespeedTech2010] by Steven J. Vaughan-Nichols and Eric Carr. They focus on a comparison of performance under different operating systems, mostly of Linux systems but in addition there was also Windows NT 4.0. The final results showed that Windows NT was the slowest operating system that supports Apache server. It was about 50% slower than the fastest Caldera OpenLinux.

A. Van Abs and Jason Brittain compared Apache Tomcat and Oracle Web Logic [Brittain2009]. They state that sometimes it is even worth to take up the challenge of migration from one server to another in an already implemented and

functioning system. In their work they put forward a lot of arguments against using Oracle Web Logic. One of the most apparent differences is obviously the type of license on which the two servers are available. Apache Tomcat in contrary to Oracle Web Logic is a completely free application. Another reason may be also too high complexity of the Oracle Web Logic. Although this is a robust tool it is also quite elaborate, both for developers or administrators. The authors justify this argument by the time required for configuration or for an update of application. Usually it also requires halting and restarting the server. It causes a problem with availability provision. Apache Tomcat requires a far shorter time. If a certain application requires less maintenance costs, and fewer employees, it can be a good argument for moving to some other technologies. Similarly, the speed of reaction to unexpected changes of system requirements, which in the case of a complex application server environment requires more time and hence a certain product may not be able to keep the pace of competition.

CHAPTER 3

EXPERIMENTAL SETUP

3.1 Performance testing tool

A significant issue related with conducting performance and load tests is selection of an appropriate testing tool. Doubtless, the most popular one which is used for examining J2EE web application server is definitely Apache JMeter. This tool is completely developed in Java and is aimed to measure application efficiency and server utilization. The most relevant physical server parts that undergo measurements are utilization of server's CPU resources and utilization of its RAM memory. Certainly from a web application developer point of view an important measurement is the maximum number of users of an application that the server can handle satisfactorily. Modern performance and load testing tools can naturally deal with such tasks. The basic parameter that is used for efficiency assessment is the time which lapse from the moment of a request sent by a tool that generates workload to the time of reception of response from the tested web server. Depending on load, which web server is subjected to, the response time will vary. Apache JMeter and other similar tools have a wide range of

functionalities for a more thorough study of the server's behavior during server work with multiple parallel requests.

Despite so many advantages and features offered by the Apache JMeter, to conduct performance tests a different tool has been used this thesis. An alternative testing tool is proposed by Eviware SoapUI. It is also a powerful tool to perform various tests on a web system, but with the difference that its design goal was not testing the traditional web applications but those based on SOA architecture, or just a single web service. Practical advantage of this application is the automatic identification of interfaces (PortTypes) defined as a WSDL or REST. This is graphically represented in the program as a hierarchy of these interfaces. There is also available a preview of feedback messages received from the web service. Besides this, there is functionality, which is helpful in carrying out performance tests, called SOAP Monitor. It is used to capture information about traffic that has been generated and sent out in the form of service requests for their further analysis. SoapUI also has many other features, which will not be used within this thesis for performing tests. These include support for WS-Security, NTLM authentication Web Service, Basic, Digest, WS-Addressing. Other types of tests that can be carried out by this tool are: functional tests, attachments testing, compliance testing and the possibility of simulation of a certain web service using MockServices built just from a WSDL file. Due to the type of research that is being conducted in this thesis, the choice of SoapUI as the main testing tools seem to be natural.

3.2 Test environment

The next step in building Experimental Setup right after establishing SoapUI as the principle tool for testing, is to develop a test environment. The main goal of this thesis is to examine the efficiency of particular web servers in certain configurations. In the case of testing a specific system based on web services it is advisable to specify the general purpose of testing. It can be for example, the estimated number of potential users that the system can efficiently handle. It is then needed to access a working system installed on the target computers and the injection of the database (if any) generated values to simulate test conditions that are the most similar to a real workload, which the server will be very likely to face while performing it's work in a real environment.

If only the web server is being tested, the configuration in which the web service also obtains and processes data from the database is unnecessary and may be subject of a separate research work. Another issue is the functional tests, which in the case of this thesis makes no sense. In order to ensure the greatest reliability of the obtained results the physical server that maintains tested web service should be physically an independent entity. It is necessary to prevent occurrence of a situation where the response time to a request will also depend on external factors, which could be for example, another external-WAN web service which has an impact on the total response time of the tested service. It is similar if we wanted to access a database server. It is not necessary but can only negatively affect the quality of measurements.

3.3 General test environment description

The web services that will be tested are assumed to be located on a single, and only server. While configuring the server, which will be specifically described in

the remaining part of this thesis, it is necessary to close all unnecessary processes running on this machine. Traffic on the test server will be generated through the SoapUI application from a different external computer. This second computer will generate a specified number of requests that will be forwarded to the service provider. The specification of requests and responses is in SOAP (Simple Object Access Protocol) standard. Of course, clients also should have an access to the web service definition language (WSDL), compatible file that will be used by the service provider. Then, it will be possible to generate the correct request (SOAP). Only if the request is compatible with the specification of interfaces defined in WSDL it will be handled. Every single request generated by a load generator, then waits for a response from a web service to which the request was sent. The time that has elapsed since sending a single request by any individual client, until receiving a reply, will be included when calculating the server average response time. The methods of testing will be described in detail in the following paragraphs. SoapUI will use another computer, which will be used for generating requests sent to the service provider.

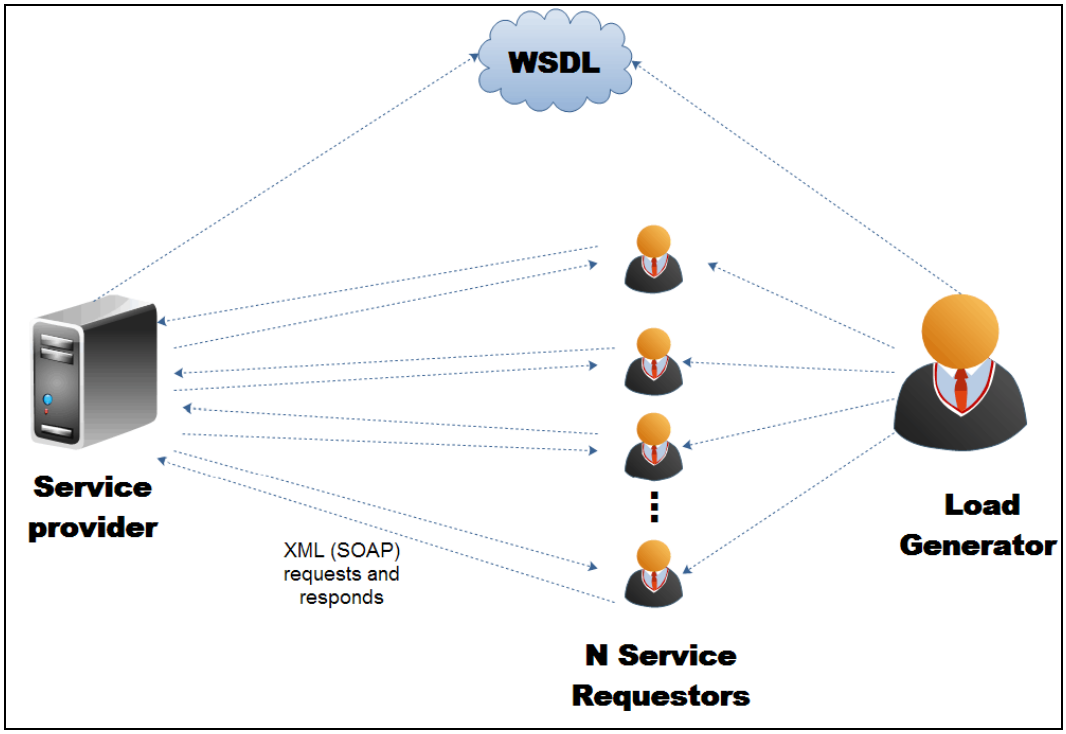

Figure 3.1 - General scheme of the test environment

3.4. Performance configurations

In order for these experiments to make sense, before testing it should be considered which configurations of physical server and service provider may have the greatest impact on the results that will be obtained. Particularly interesting can be those configurations that can play a role in finding significant differences in the application web servers.

3.4.1 Payload size

A parameter that is specific for web services is the payload size. During the tests the size of a SOAP request messages will be different in different test cases. The change of request size sent to Service Provider can expose if any of the Web servers, if any, performs well only for small-sized requests. The aim is to identify a Web server that is doing well with both smaller and larger request sizes that are sent from a great number of clients.

In each case the server will be subjected to different loads. This means that, a different number of requests will be generated in a unit of time. Thus it is possible to observe certain characteristics (profile) of tested web servers. Of course, each server has certain limits which, if exceeded, can cause completely unpredictable performance results. Such a threshold may be e.g. too long average response time that cannot be accepted. Another possibility is to receive erroneous responses from the Service Provider. These are obvious signs of overloading the server, which should never be allowed. This type of tests helps developers to raise awareness and to estimate the number of users of the system, which can cause hazard in the system availability for its clients. This is even more important because any behavior of that kind reduces the security of the data stored in a server.

3.4.2 Service provider configurations - Single web service request

In this case all generated load will attempt to access a single web service, that is a self-dependent entity.

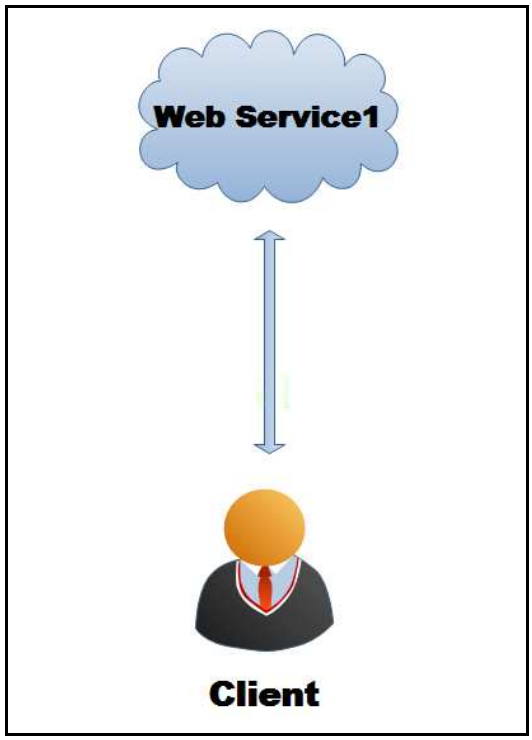

Figure 3.2 - Single web service with many requests.

3.4.3 Service provider configurations - A web service dependent on two other web services

In this case a service provider consists of three web services among which the main one relies on the other two. A client sends a request to the web service with certain request parameters that is further propagated to the remaining two web services. Eventually the client receives a response from the main web service, which got initial client request.

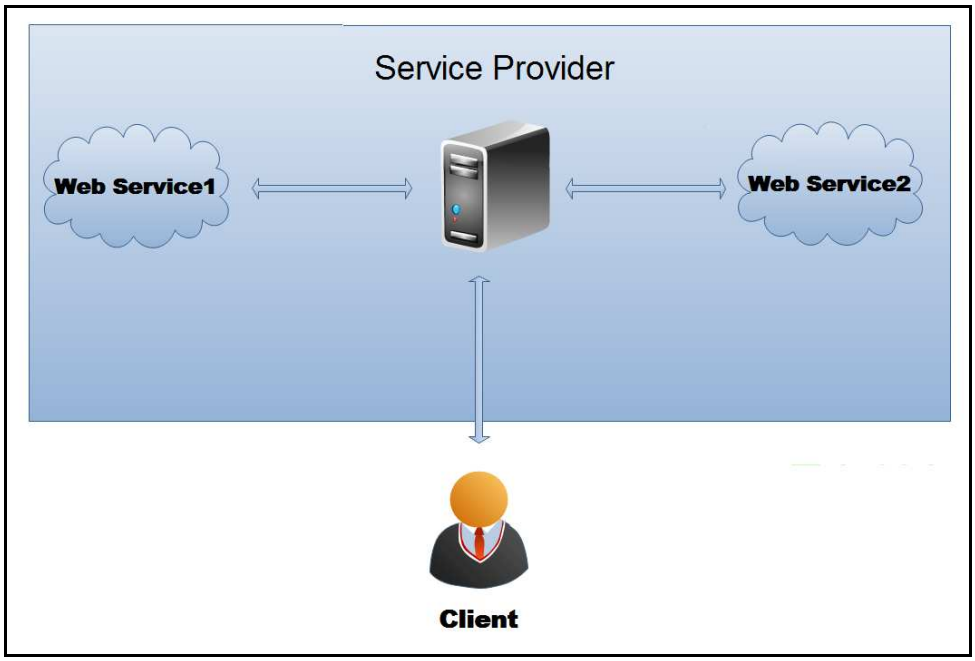

Figure 3.3 - A web service dependent on two other web services.

3.2.3 Service provider configurations - very complex SOA environment

The third environment on which tests will be conducted is extremely complex. It consists of multiple web services, which are tightly related with each other. Client sends SOAP requests message, which contains a string-type parameter. This parameter is further propagated amongst whole SOA environment, and eventually all responses from all web services are being joined according to service orchestration point approach and returned to the client.

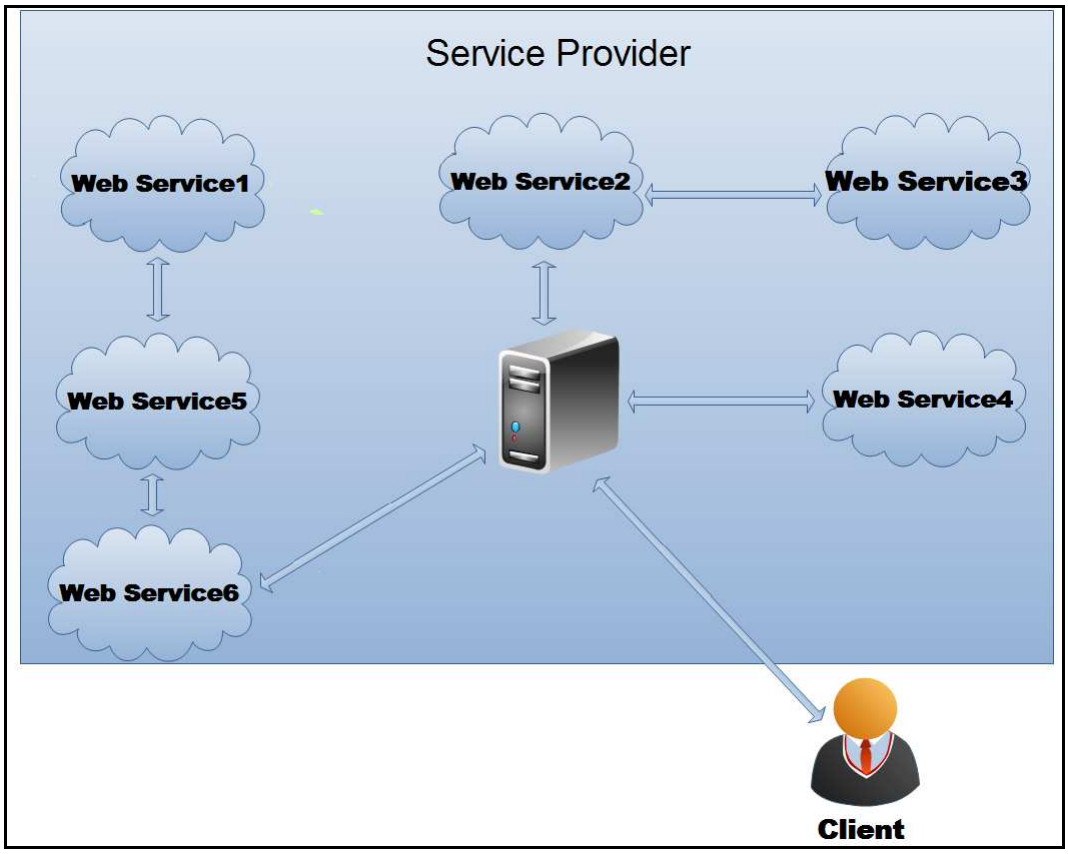

Figure 3.4 - Highly dependent web service

3.3 Performance measurements

Another important point in developing a test plan is deciding on which measurements will be collected. The first measurement is naturally the web server response time. For performance testing, the test server will be subjected to very heavy load (stress test). Especially near the upper limits of the capacity of the server, the response times may be significantly different from those for light loads. Under extreme loads a web service may give incorrect results, or not respond at all. For this reason only the average response time is meaningful. This

is by far the most important factor that determines the quality of the examined web server [Molyneaux2009].

As it already has been mentioned, there are situations in which the response time measurement is flawed. These cases will also be investigated. It is possible that certain web server would provide responses to requests after a long time, but on the other hand, a higher percentage of correct answers will be received. In the final assessment all these factors should be included. Examining the two above quality evaluation criteria for the examined web servers, one can determine cases for which they suit the best.

3.4 Performance test strategy

In order to decide whether the efficiency test and the quality test of SOA-based system, was duly carried out, usually one type of test is not enough. This is because of the different goals of the different types of tests. The most common of them is of course load test, where the primary task is to examine the response time for requests sent to the server [Priyanka2008]. These requests are generated with different frequencies. Usually the measurement of average response time is made at a certain threshold, ranging from very small server load. With increasing load on the server, the response times will be longer as well. At a high frequency of generated requests the server starts to behave less predictably. This may manifest itself in many ways. One of the most common observations is lack of response to the request or receiving a response after an unreasonably long period of time. Other possible cases are receiving a large number of wrong answers or not being able to connect to the service. For a specifically studying this kind of behaviors of a server stress tests have been established. It lasts as long as any of

these behaviors will start to occur. Basically, the aim is to determine the limits at which the test system may still work.

In this way, we can estimate the upper limit to the number of users that the server can handle. In contrary to the load test, this one is focused solely on bringing the server to an invalid state or behavior and to determine when it occurs. When testing the efficiency of various web servers, a task for a service provider will be very simple, but sufficient to perform a proper comparison of performance of web servers. For this reason, in my case, conducting of stress test does not make greater sense. Stress tests are usually used for testing a complete system, estimating the maximum number of its users. Not to test the performance of the web server. When performing the load test, the test will receive correct answers, and those affected by the error will be included in the final evaluation of a web server.

Another type of test that you can perform is so-called baseline test [Priyanka2008]. It determines a specific reference point for subsequent tests. Thanks to it you know the best possible outcome with only a single request. To make tests results more reliable, the test should be carried out repeatedly and there should be selected the best possible result. Please note that while obtaining the results the server may be busy with other, secondary processes, which could distort the result. For my research, baseline test is redundant, mainly due to the small amount of information that gives this type of test in my case. Differences between servers in a single request, where the server is not loaded are essentially negligible. In real systems, the waiting time for the single request to the server, depends mainly on the quality of network connection, between a client and a service provider. In a real system, the differences which could reach milliseconds are irrelevant.

3.5 Injection profile

In developing a performance test plan, one can also consider the characteristics of requests generating by the load generator (injection profiles). The most popular and the most commonly used method is called ramp-up [Priyanka2008]. It starts from only one client that simulates a web system user. Within next regular intervals the number of requests to the service provider is being increased, until the desired number of requests per second.

This method can also be used with slight modification. It involves the introduction of breaks while the increasing frequency of requests to the server. After a short break, again frequency of generated requests is increasing, and then soon another break, that occur at some fixed threshold. This is helpful when one want to observe the server load at certain significant thresholds of the intensity of requests.

Another common method is called big bang. Main idea is to start generating all the requests in the same moment. Unlike the previous method the server is subjected to a constant load.

In my research I decide to use ramp-up method with the modification. My main goal is to compare the behavior of each server at the specified load. Thanks to modification, it is relatively easy to observe significant differences at certain thresholds.

3.6 Experimental setup summary

Experimental setup for the tests is as follows:

Servers to be compared:

- Apache Tomcat 6.0
- Oracle WebLogic 10

Testing tool: Eviware SoapUI

Server hardware specification:

- CPU: XEON (2 core of 2.0 GHz each)
- 2 GB RAM
- 80 GB WD SATA HDD

Physical test environment:

Single server for load generator (SoapUI) and another one (xeon) as a service provider

Configurations:

- Load generator settings are determined by the size of XML request message (size of payload) and by requests per second (load level)
- Service Provider will be represented as a single web service, a web service dependent on two other web services, and as very complex SOA environment

Comparison criteria:

- Average response time

- Responds per second
- Number of service responses saddled with error or lack of response

Test strategy

- Load test
- Stress test

Injection profile

- Ramp-up (with steps)
- Big bang

CHAPTER 4

EXPERIMENTAL RESULTS

The following Chapter is divided into three sections. Each section represents another SOA environment. Tests included in this chapter are as follows:

Table 4.1 – Tests conducted on single, self-dependent web service

	Test type	**Injection profile**	**Load**	**Payload size**	**Total time**
Test 1	Simple load	Big bang	5 threads	259 bytes	120 sec
Test 2	Simple load	Big bang	15 threads	259 bytes	120 sec
Test 3	Simple load	Big bang	30 threads	259 bytes	120 sec
Test 4	Ramp-up load	Ramp-up	1 to 40 threads	259 bytes	300 sec

Test 5	Burst load	Big bang	35 thread	259 bytes	150 sec
Test 6	Ramp-up load	Ramp-up	1 to 20 threads	129 000 bytes	120 sec

Table 4.2 – Tests conducted on a web service dependent on two other web services

	Test type	**Injection profile**	**Load**	**Payload size**	**Total time**
Test 7	Simple load	Big bang	5 threads	262 bytes	120 sec
Test 8	Simple load	Big bang	15 threads	262 bytes	120 sec
Test 9	Simple stress	Big bang	25 threads	262 bytes	120 sec
Test 10	Ramp-up load	Ramp-up	1 to 20 threads	262 bytes	300 sec
Test 11	Ramp-up stress	Ramp-up	25 to 50 threads	262 bytes	300 sec
Test 12	Burst load	Periodical big bang	35 threads	262 bytes	150 sec

Table 4.3 – Tests conducted on a very complex SOA environment

		Injection profile	**Load**	**Payload size**	**Total time**

	Test type				
Test 13	Simple stress	Big bang	1 threads	260 bytes	120 sec
Test 14	Ramp-up load	Ramp-up	1 to 3 threads	260 bytes	300 sec

4.1 Single self-dependent web service tests

Tests conducted in this section are aimed to compare the performance of web servers when they handle simple web service.

4.1.1 Test 1: Simple 5 thread load test

During this test the load generator uses five threads that simultaneously send requests to the server. We do not take into account the simplicity of single web service. The load is constant for the whole period of test.

The test configuration is as follows:

- Number of threads generating requests: 5
- Injection profile: big bang
- Environment: single web service
- Test time: 120 seconds
- SOAP message payload size: 259 bytes

Table 4.4 – Results for Test 1

	Apache Tomcat	Oracle Web Logic
Average response time [ms]	15,150	15,018
Average throughput [transactions/sec]	312,795	314,121
Standard deviation of response times	1,204	0,292

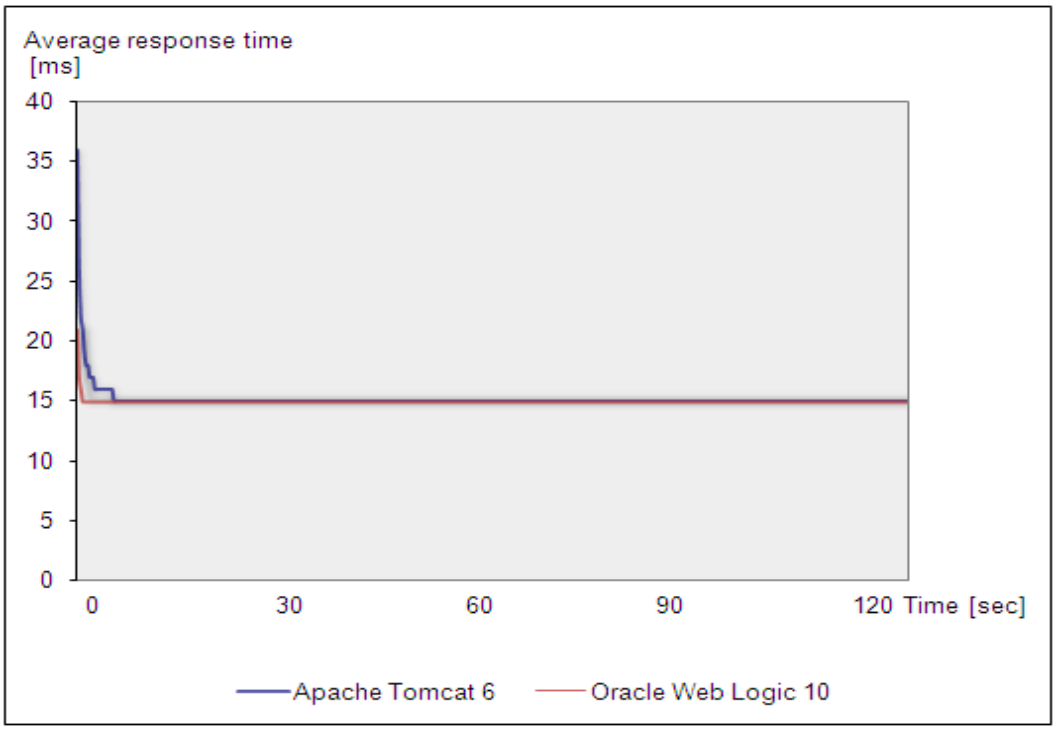

Figure 4.1 – Average response time for Test 1

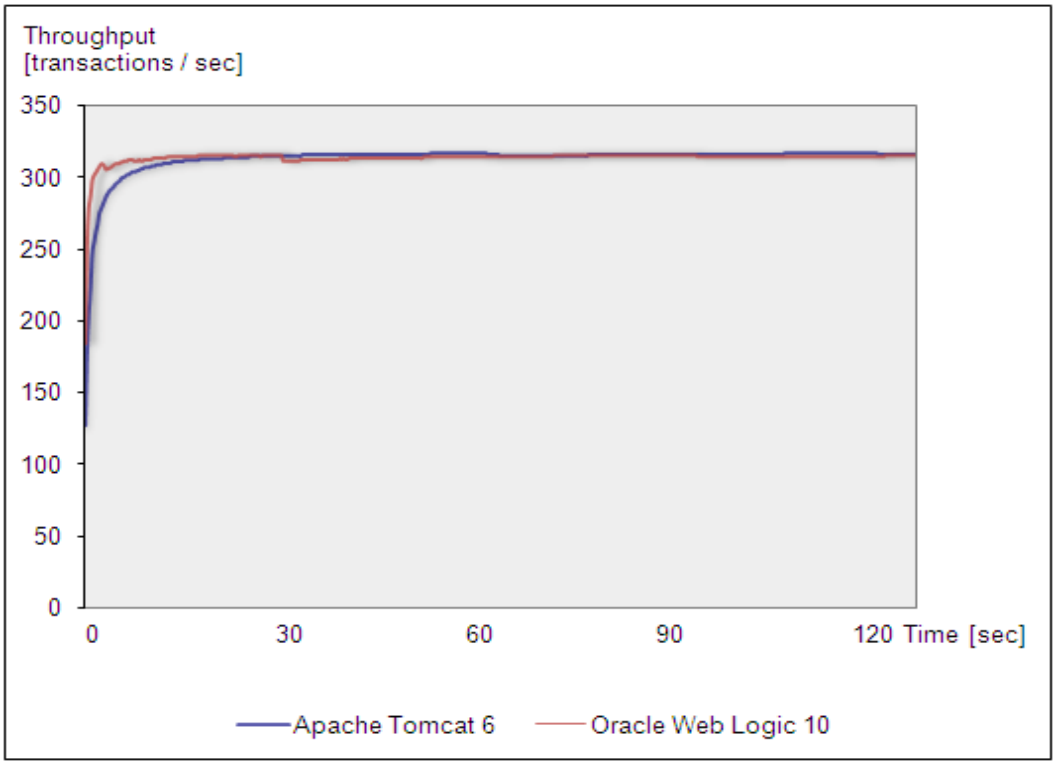

Figure 4.2 –Throughput for Test 1

As expected the server wasn't loaded very much. According to the table 4.1 in both cases average response time was around 15 ms. The characteristics of throughput (Figure 4.2) were very similar as well. Only standard deviation of Apache Tomcat average response time was slightly greater than Oracle Web Logic. Apparently it is related with the early beginning of the test (first 1-2 seconds). Response times of the initial responses of Oracle Web Logic were more aligned. The remaining part of the test looks nearly identical for both web servers.

4.1.2 Test 2: Simple 15 thread load test

In this test the load has been increased to 15 threads that are generating requests. The remaining parameters are the same as in last test case (Test 1).

The test configuration is as follows:

- Number of threads generating requests: 15
- Injection profile: big bang
- Environment: single web service
- Test time: 120 seconds
- SOAP message payload size: 259 bytes

Table 4.5 – Results for Test 2

	Apache Tomcat	Oracle Web Logic
Average response time [ms]	18,971	19,937
Average throughput [transactions/sec]	716,150	680,923
Standard deviation of response times	0,216	0,655

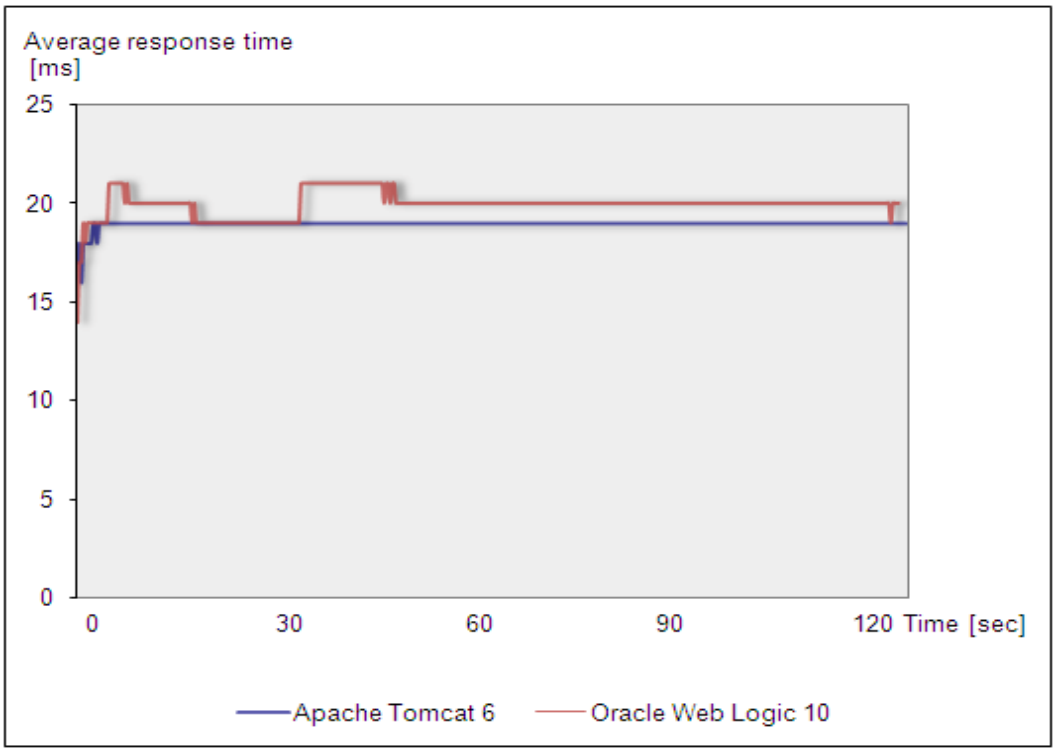

Figure 4.3 – Average response time for Test 2

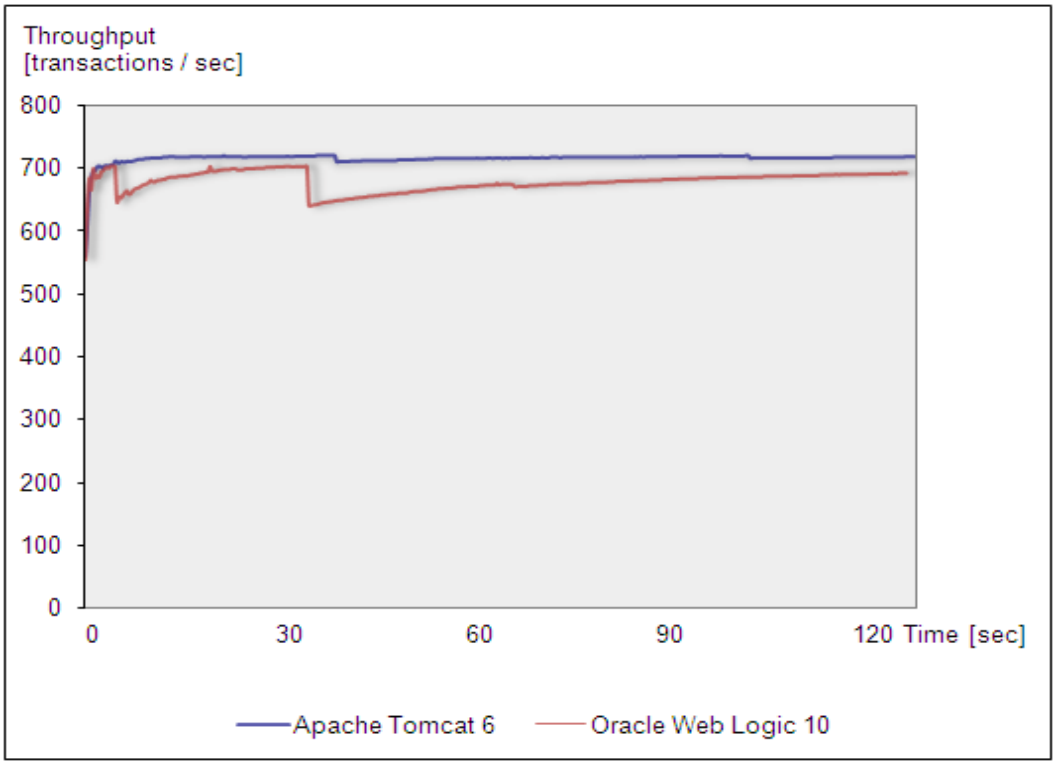

Figure 4.4 – Throughput for Test 2

Load generated by 15 threads turned out to be still too little in order to observer any significant differences (Table 4.2). The average response time increased just to 19-20 ms from 15 ms in last test where were used only five threads. In the test 1, slightly better performance had Oracle Web Logic (~0,9%) but in this test Apache Tomcat was faster about 5%.

4.1.3 Test 3: Simple 30 thread load test

In previous two tests generated load turned out to be quite little for the server. So in this test the load was consequently increased to 30 threads.

The test configuration is as follows:

- Number of threads generating requests: 30
- Injection profile: big bang
- Environment: single web service
- Test time: 120 seconds
- SOAP message payload size: 259 bytes

Table 4.6 – Results for Test 3

	Apache Tomcat	Oracle Web Logic
Average response time [ms]	38,692	38,120
Average throughput [transactions/sec]	710,365	718,115
Standard deviation of response times	1,429	0,712

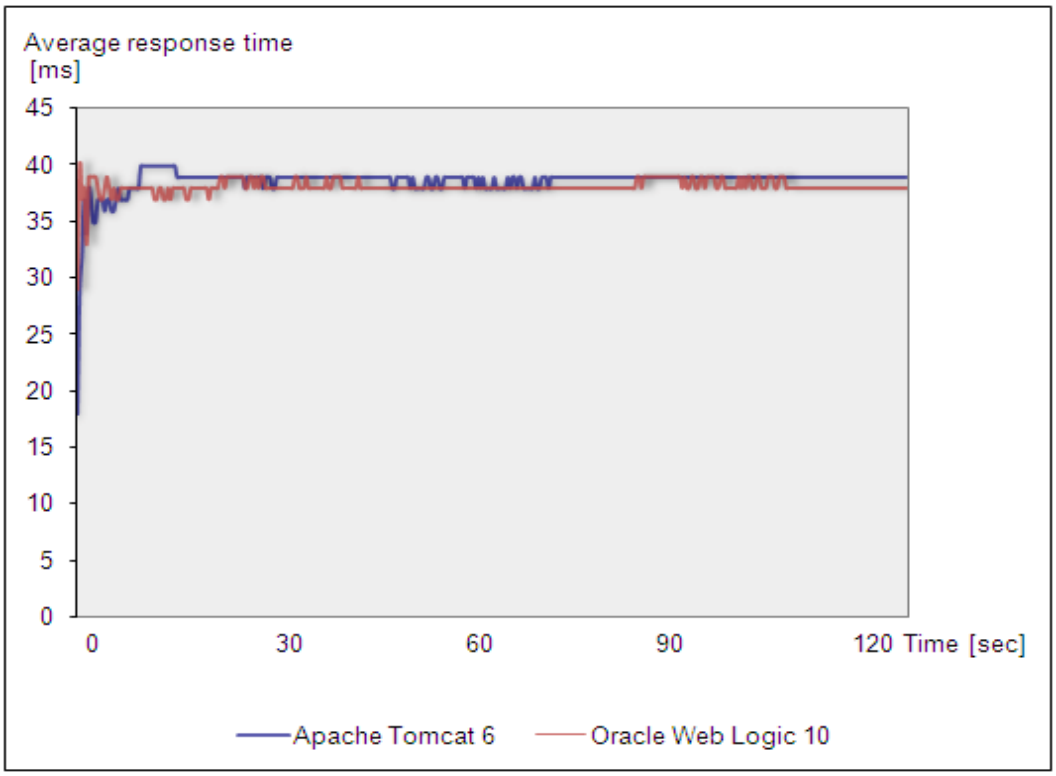

Figure 4.5 – Average response time for Test 3

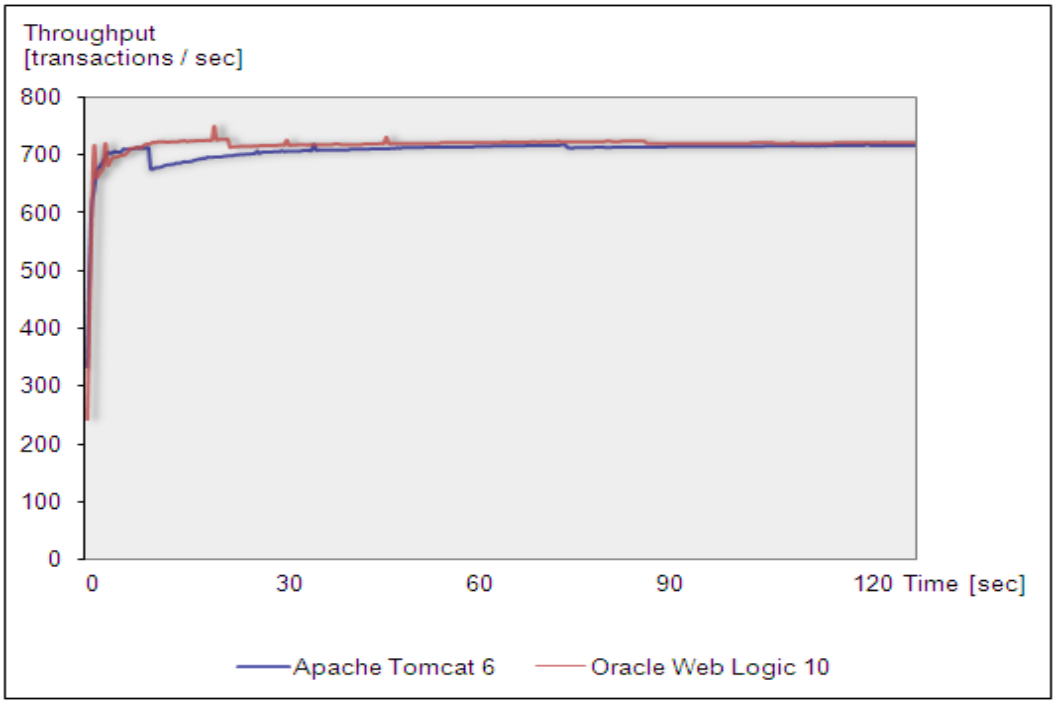

Figure 4.6 – Throughput for Test 3

Comparing to the test 2, in this one the load was doubled (Table 4.3). Also the average response (Figure 4.5) time was nearly doubled and the average throughput remains same as in test 2. It means that the threshold of maximum efficiency has been reached in the previous test. Comparing these two servers a little bit faster (1,1%) was Oracle Web Logic but the difference is very small (Figure 4.6).

4.1.4 Test 4: Ramp-up performance test

This is the first test where intensity of load is varied. This test is aimed to examine behavior of the servers when load is being gradually increased from 1 to 40 threads used by load generator.

The test configuration is as follows:

- Initial number of threads generating requests: 1
- Final number of threads generating requests: 40
- Injection profile: ramp-up
- Environment: single web service
- Test time: 300 seconds
- SOAP message payload size: 259 bytes

Table 4.7 – Results for Test 4

	Apache Tomcat	Oracle Web Logic
Average response time [ms]	22,846	23,26
Average throughput [transactions/sec]	678,359	666,462
Number or error or missing responses	0	0

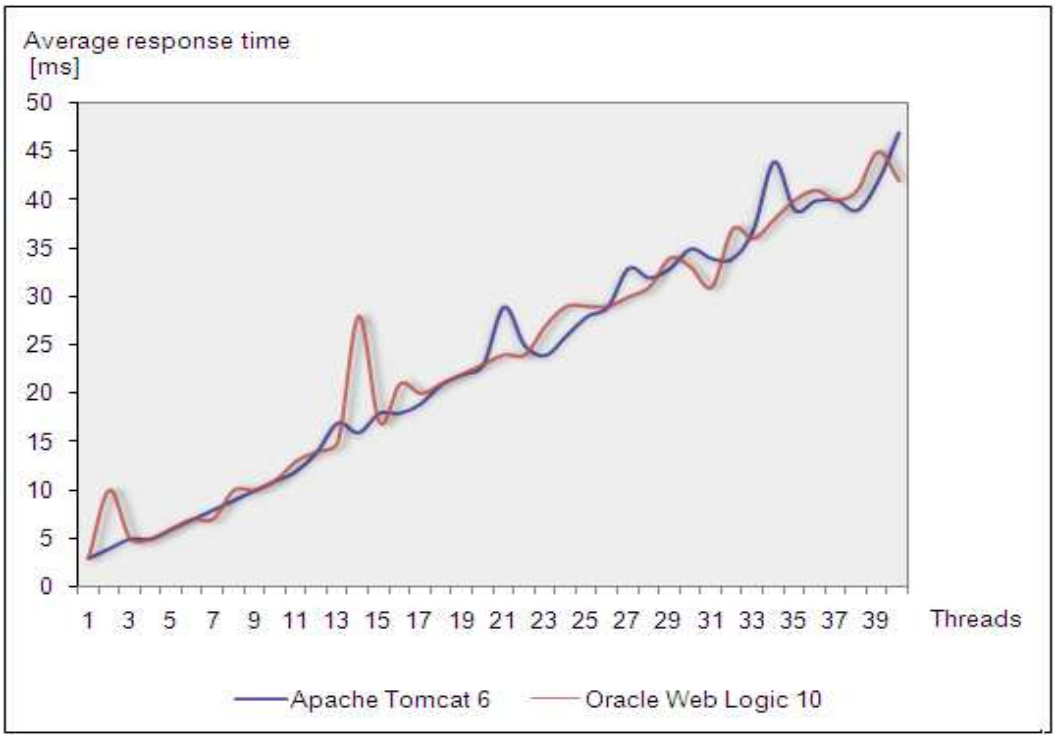

Figure 4.6 – Average response time for Test 4

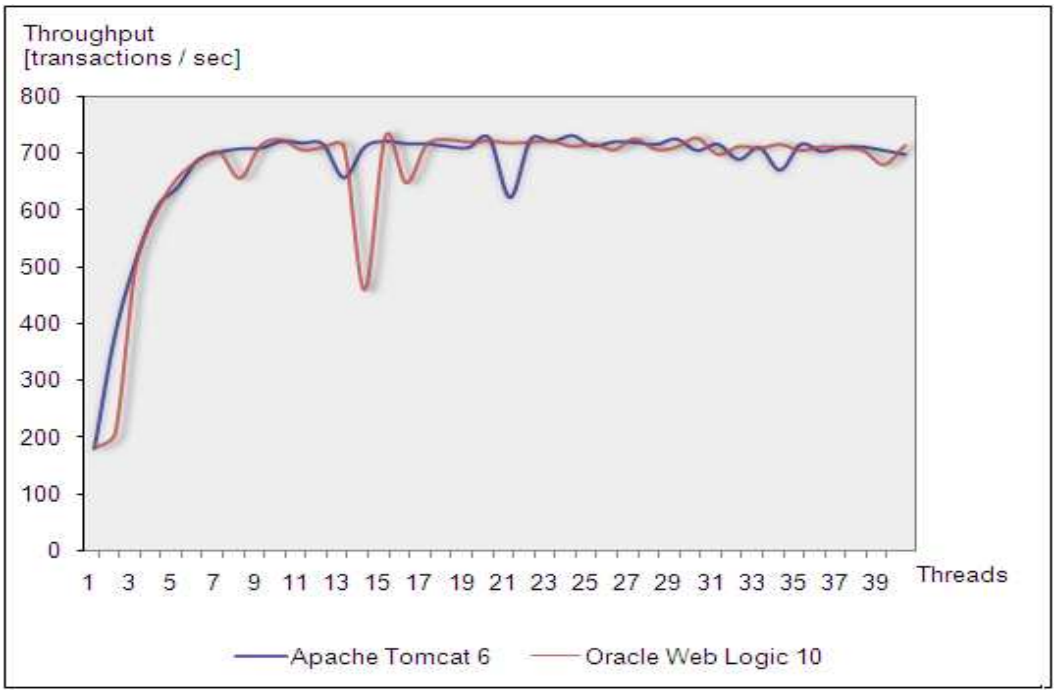

Figure 4.7 – Throughput for Test 4

As it has been already stated in last test summary, the maximum throughput was already reached in test 2 because even higher number of threads didn't make throughput higher as well. In the figures 4.7 and 4.8 we can see that in fact the maximum throughput is already reached by around 8-10 threads used by load generator. In both servers when using more than about 20 threads throughput is being a little decreasing. Overall performance of Apache Tomcat is a bit better than Oracle Web Logic. In spite of all the difference is just around 2%. Despite quite high load in the end of the test (35-40 threads) both servers behaved stably and didn't return any errors.

4.1.5 Test 5: Burst load test

The main goal is to examine the behavior of servers when they have to deal with rapidly changing load (burst test). After 10 seconds of idle server state all 35 threads immediately start sending requests to the server. After 30 seconds of load server is becoming idle again for 10 seconds, and then the load is generated again. The period when server is totally idle is not shown at the charts.

The test configuration is as follows:

- Number of threads generating requests: 35
- Injection profile: periodical big-bang
- Burst time 30 seconds
- Time of break between bursts: 10 seconds
- Environment: single web service
- Test time: 180 seconds

- SOAP message payload size: 259 bytes

Table 4.8 – Results for Test 5

	Apache Tomcat	Oracle Web Logic
Average response time [ms]	43,29	42,89
Average throughput [transactions/sec]	695,2	712,25
Number or error or missing responses	0	4

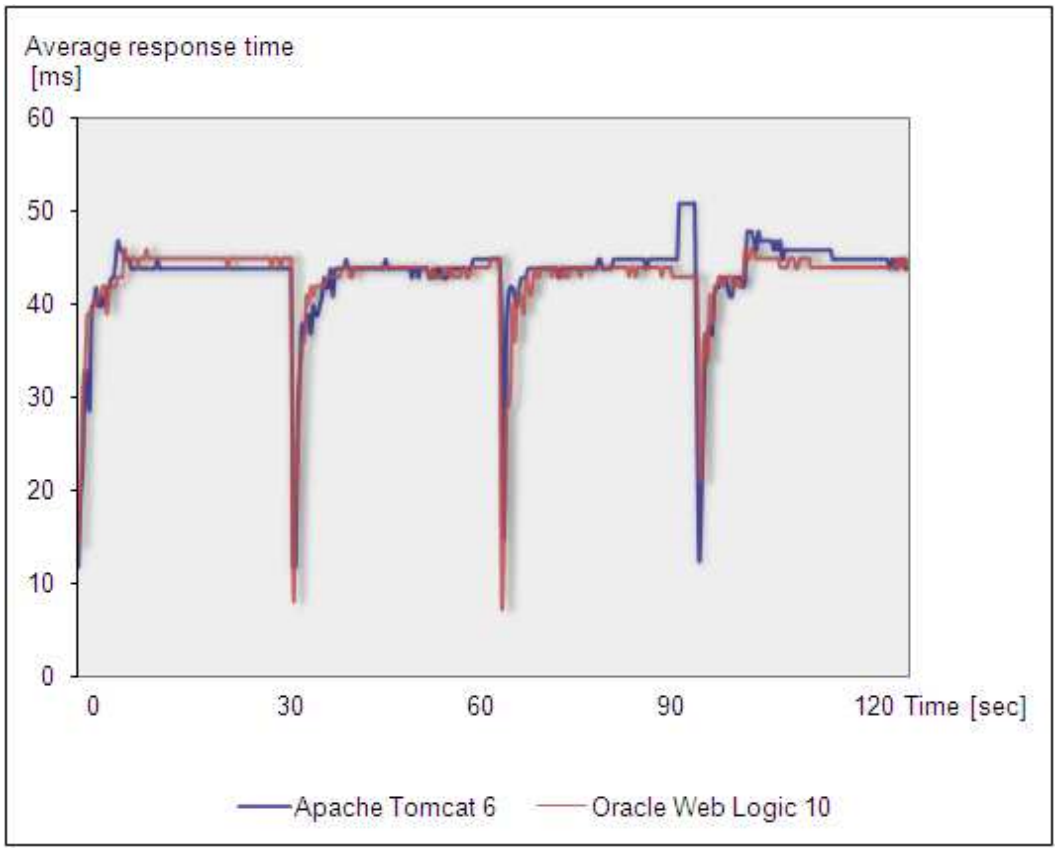

Figure 4.8 – Average response time for Test 5

The characteristics of average response times look quite similar (Figure 4.7). The only small difference is that Apache Tomcat was behaving a bit less regularly than Oracle Web Logic. After some short time from the start of given load burst the average response times are being aligned to approximately 43-45 ms in both cases. Oracle Web Logic returned 4 error messages but if we take into account the total number of requests and responses it become meaningless. Similar like in previous test differences in performance are quite narrow (0,9%) but this time Oracle Web Logic was faster (Table 4.5).

4.1.6 Test 6: Ramp-up performance test with increased SOAP message payload size

What differs this test from the previous is the size of the messages sent to the server as a SOAP XML request message and the respond message. This test is aimed to examine whether the size of the message has any meaning in comparison of both web servers.

The test configuration is as follows:

- Initial number of threads generating requests: 1
- Final number of threads generating requests: 20
- Injection profile: ramp-up
- Environment: single web service
- Test time: 300 seconds
- SOAP message payload size: 129 890 bytes

Table 4.9 – Results for Test 6

	Apache Tomcat	Oracle Web Logic
Average response time [ms]	989,5	1013,1
Average throughput [transactions/sec]	7,3	7,7
Number or error or missing responses	0	0

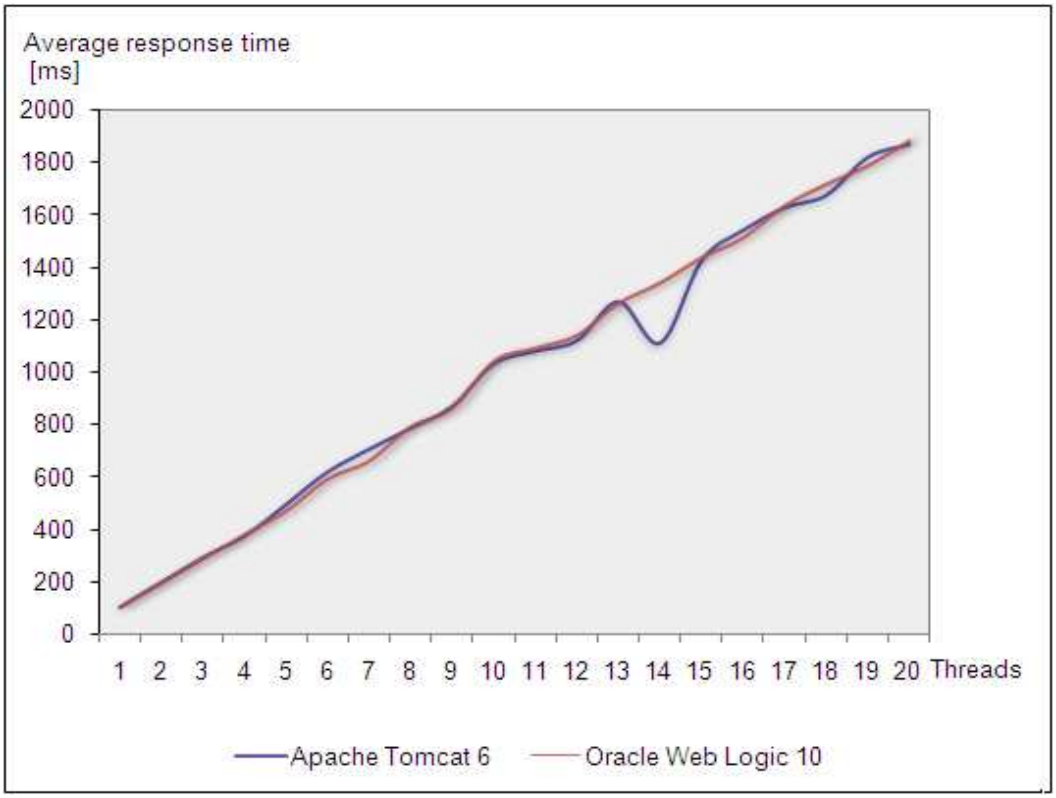

Figure 4.9 – Average response time for Test 6

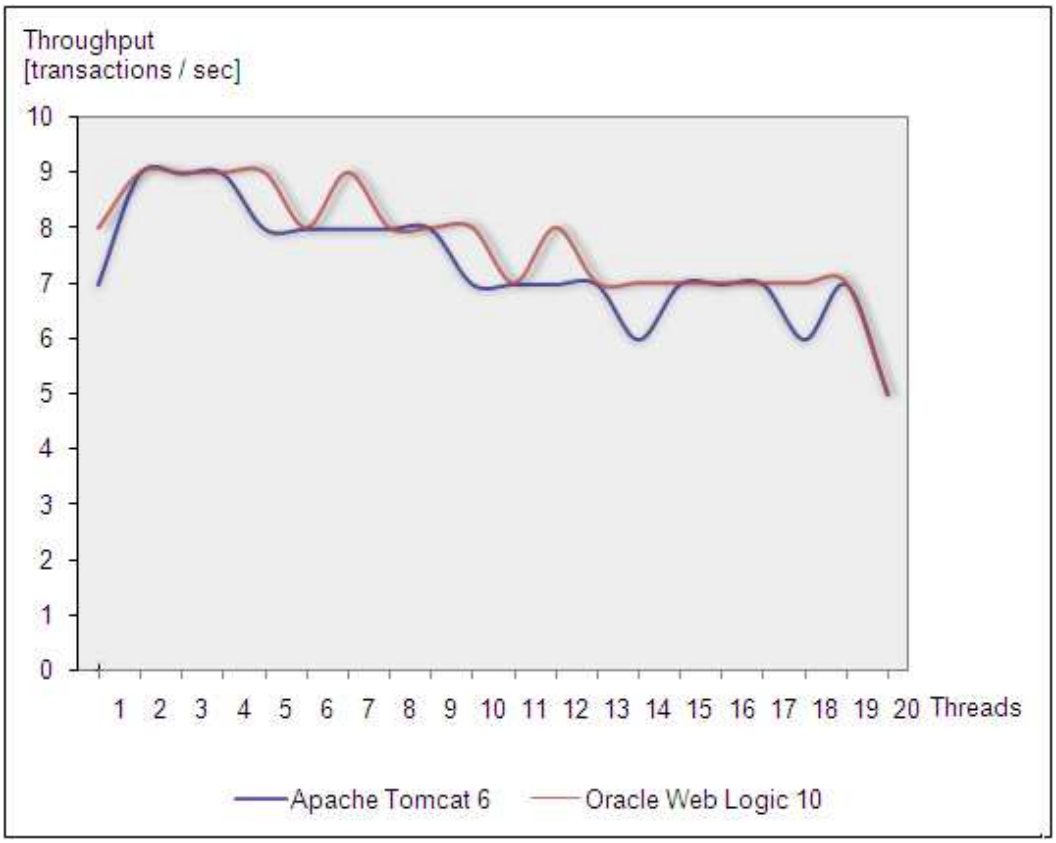

Figure 4.10 – Throughput for Test 6

Comparing to the corresponding test 4 with much lower message size (259 bytes) the average response time has been significantly increased to around 1000 ms (Table 4.6). The difference in performance between the servers is equal to 2,5% in favour of Apache Tomcat. To remind in the corresponding test with low payload size it was 2% difference. In this test the characteristics of throughput and average response time of the servers were similar. Both servers also didn't return any errors (Table 4.6). Change of message payload size didn't indicate any significant differences in two examined web servers (Figure 4.10).

To summarize amongst six tests that has been conducted, within three of them faster was Apache Tomcat and within remaining three faster was Oracle Web Logic. However difference in performance wasn't very big. In maximum it was 5% in test 2. In the first seconds of simple or burst tests where server was subjected to rapid load, Oracle Web Logic was responding faster and more stable. That is why got better results in the majority of tests where injection profile was big-bang. On the other hand in all ramp-up tests Apache Tomcat had better performance. With respect to simplicity of environment that servers had to deal with (single web service) in all tests servers were very stable and there wasn't any problem with overloading. Also throughput under maximum load of 35 or 40 threads didn't get much lower as we could expect. Tests in the next part of this chapter are conducted using more elaborated environments and will be also more focused on the stability under extremely high load including stress tests.

4.2 Web service dependent on other web services

Tests conducted in this section are aimed to compare the application servers performance in case when they are dealing with web service dependent on other web services. The detailed description of this environment is placed in Chapter 3.

4.2.1 Test 7: Simple 5 thread load test

Similar as in the test 1 of the environment consisted of single web service this load generator in this test uses five threads that simultaneously send requests to the server. The load is constant for the whole period of test.

The test configuration is as follows:

- Number of threads generating requests: 5
- Injection profile: big-bang
- Environment: web service dependent on other web services
- Test time: 120 seconds
- SOAP message payload size: 262 bytes

Table 4.10 – Results for Test 7

	Apache Tomcat	Oracle Web Logic
Average response time [ms]	81,498	205,634
Average throughput [transactions/sec]	62,211	24,083
Standard deviation of response times	31,778	57,640

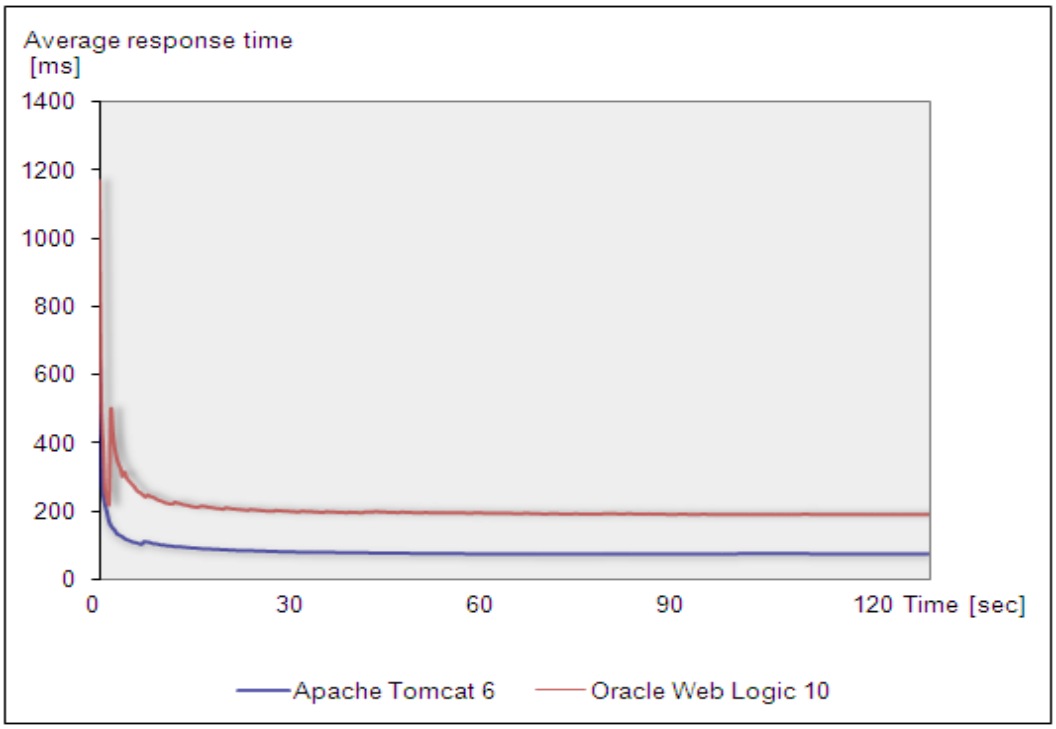

Figure 4.11 – Average response time for Test 7

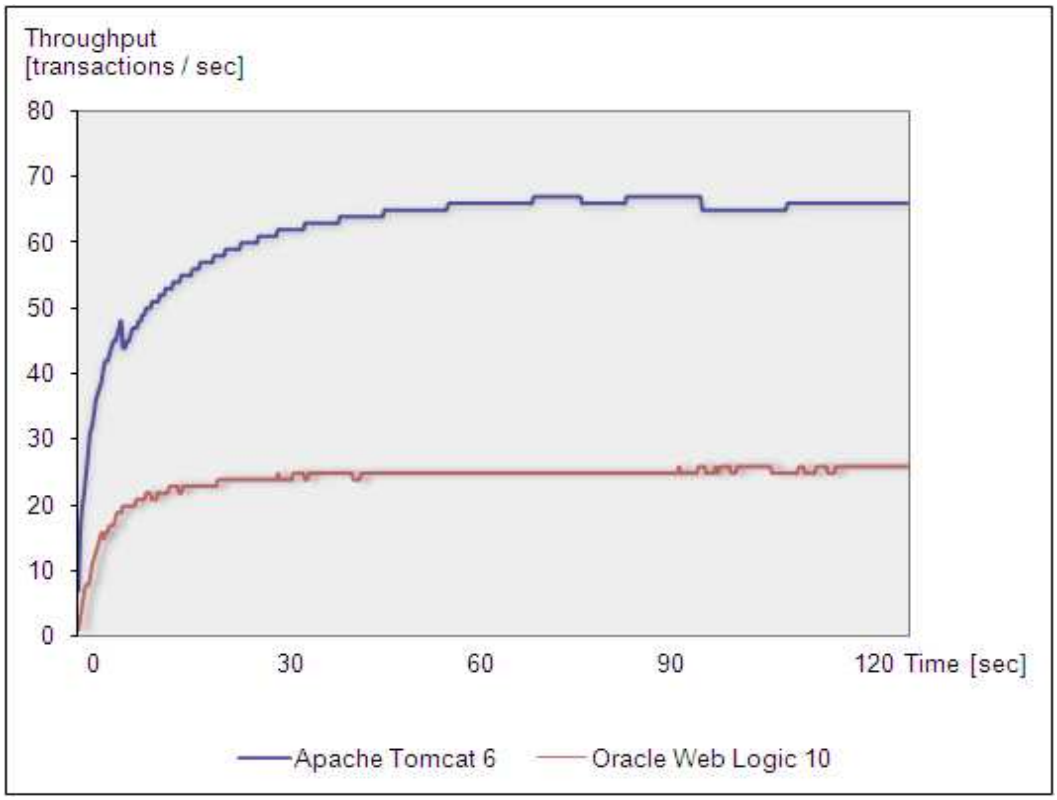

Figure 4.12 – Throughput for Test 7

Already the first of the tests conducted on more elaborated environment showed significant differences in performance (Table 4.7). Characteristics of average response times (Figure 4.11) are quite similar apart from the early beginning of the test where Oracle Web Logic had some anomalies. After that the response times became aligned and gradually decreased to certain level. The average response time of Apache Tomcat was 81,498 ms which is 153% faster than Oracle Web Logic (205,634 ms). Next experiments will try to show if this big difference will change under heavier load.

4.2.2 Test 8: Simple 15 thread load test

In this test the load was increased to 15 threads which is already quite big load for this environment. The load degree remains constant for the whole period of test.

The test configuration is as follows:

- Number of threads generating requests: 15
- Injection profile: big-bang
- Environment: web service dependent on other web services
- Test time: 180 seconds
- SOAP message payload size: 262 bytes

Table 4.11 – Results for Test 8

	Apache Tomcat	Oracle Web Logic
Average response time [ms]	215,7694	651,23
Average throughput [transactions/sec]	68,3	21,76
Standard deviation of response times	1,287	47,336
Number or error or missing responses	0	1

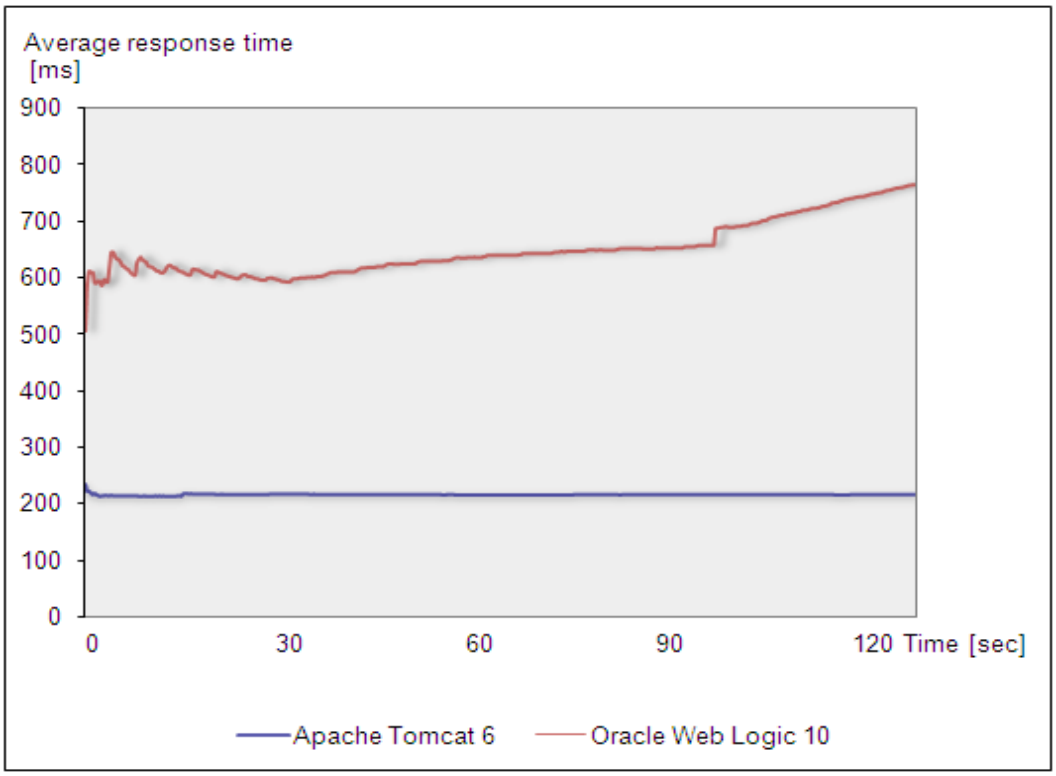

Figure 4.13 – Average response time for Test 8

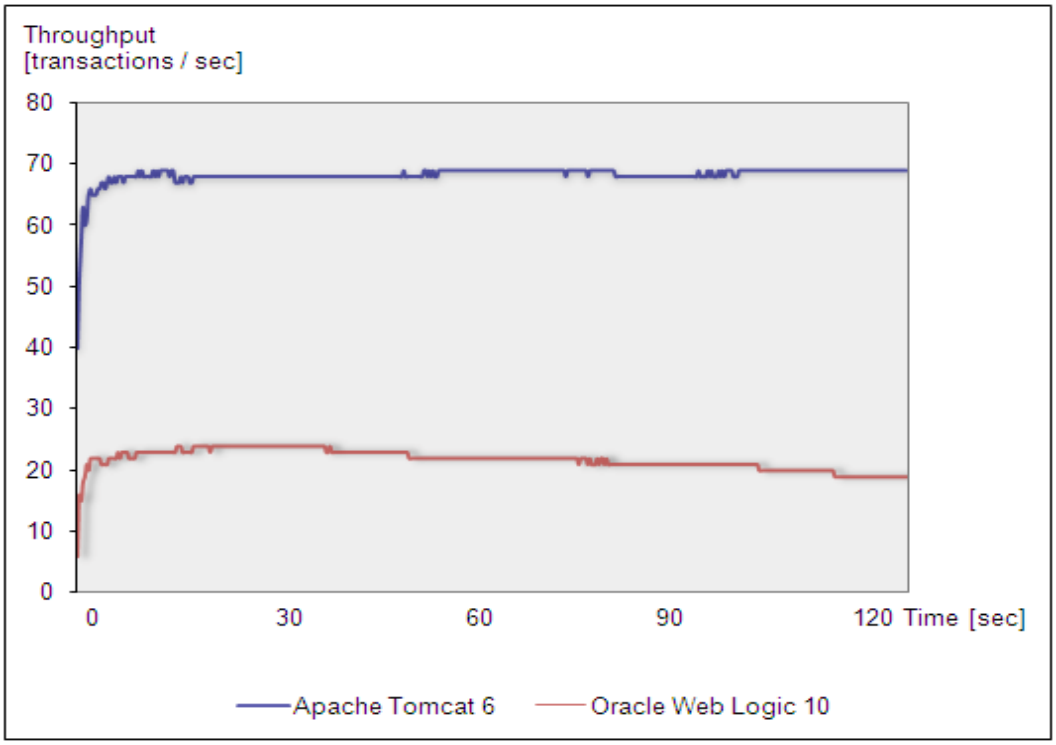

Figure 4.14 – Throughput for Test 8

Basing on the results from Table 4.8 average response time of Apache Tomcat was 202% greater than Oracle Web Logic. Comparing to Test 1 despite increasing the number of threads the throughput of Oracle Web Logic has decreased from 24,083 to 21,76 requests per second. On the other hand Apache Tomcat behaved in the opposite way. Throughput has been slightly increased from 62,211 to 68,3 requests per second. It means that during test 7, five threads made Oracle Web Logic fully loaded (but stable). In Figures 4.13 and 4.14 we can see that Apache Tomcat remains very stable which also confirms very little standard deviation of response times. Response times for Oracle Web Logic (Figure 4.15) are quite different and even worse has tiny upward trend. It is apparent symptom that this server is not able to deal with such a load in a long term.

4.2.3 Test 9: Simple 25 thread load test

The previous test 8 showed that 15 threads is already much for both servers especially to Oracle Web Logic. The current test is focused on even greater load in order to see how servers behave under extremely load of 25 threads.

The test configuration is as follows:

- Number of threads generating requests: 25
- Injection profile: big-bang
- Environment: web service dependent on other web services
- Test time: 180 seconds
- SOAP message payload size: 262 bytes

\

Table 4.12 – Results for Test 9

	Apache Tomcat	Oracle Web Logic
Average response time [ms]	375,720	4379,995
Average throughput [transactions/sec]	65,006	5,127
Number or error or missing responses	182	0

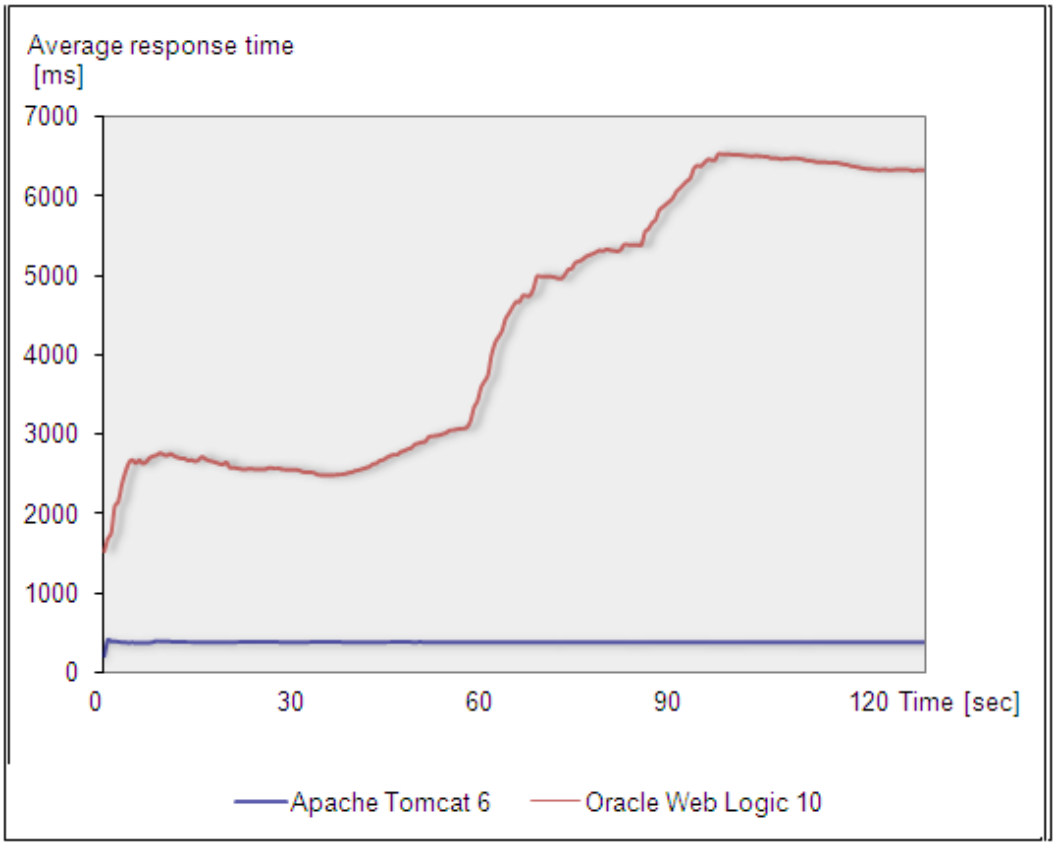

Figure 4.15 – Average response time for Test 9

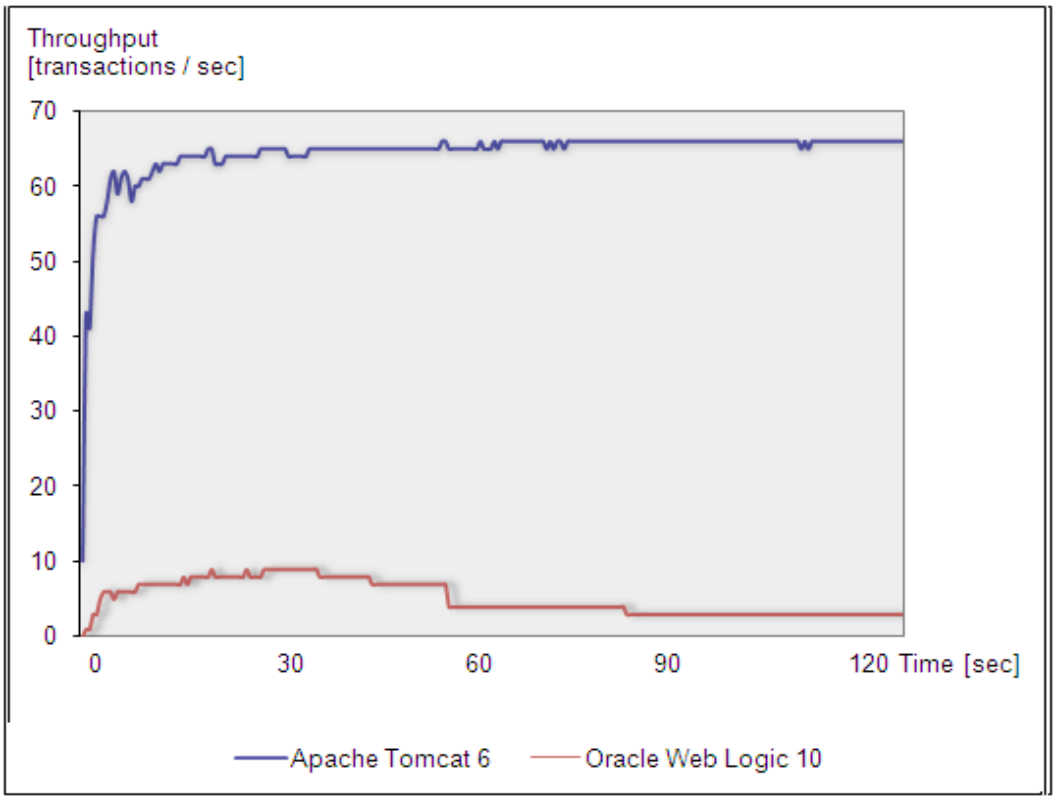

Figure 4.16 – Throughput for Test 9

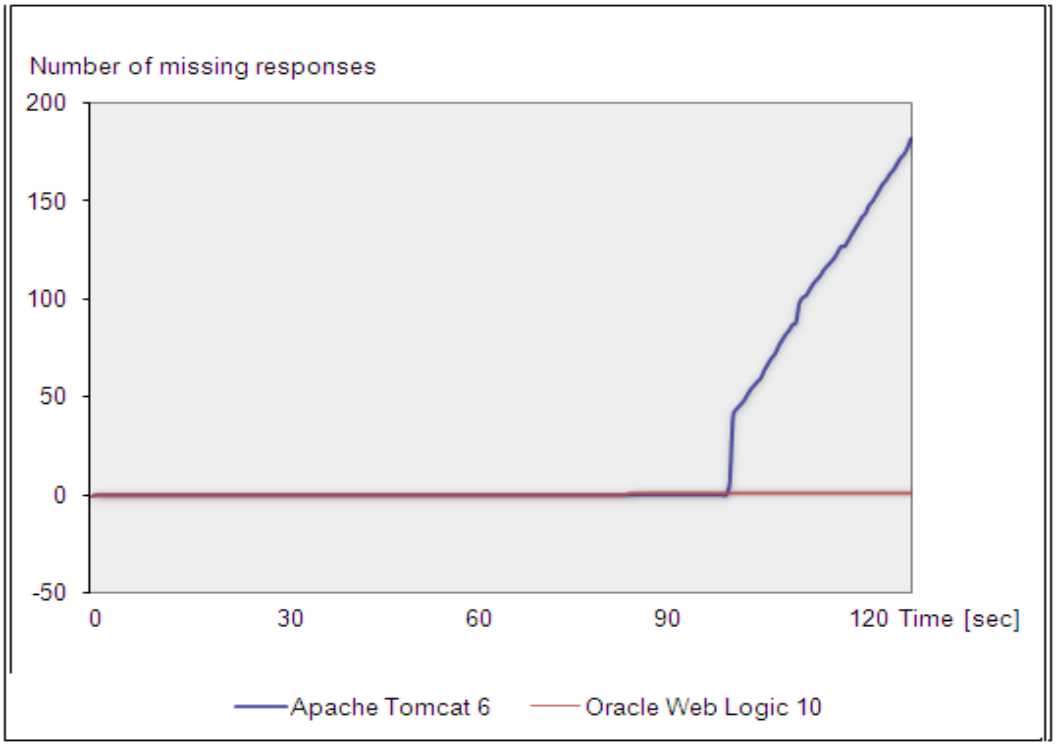

Figure 4.17 – Missing responses for Test 9

According to the results obtained in this test, the differences between both servers that showed previous test 8 are even bigger now. The figure 4.14 from the previous test of Oracle Web Logic average response time shows, that for 15 threads it has upwards trend. In current test for 25 threads it was much steeper and was constantly increasing. Server was not able to deal with such a load. Oracle Web Logic server was completely overloaded and couldn't even find a way to prevent from such a big load. For instance Apache Tomcat started refusing the excess of requests that could make it unstable and consequently elongate response time to enormous level as it was in case of Oracle Web Logic. The average response time of Apache Tomcat was aligned to approximately 375 ms.

4.2.4 Test 10: Ramp-up performance test

This is the first test of this environment where intensity of load is varied. It is aimed to examine the behavior of servers when load is being gradually increased from 1 to 30 threads used by load generator. According to previous test results (4.2.3 and 4.2.2) such a load is already very heavy.

The test configuration is as follows:

- Initial number of threads generating requests: 1
- Final number of threads generating requests: 30
- Injection profile: ramp-up
- Environment: web service dependent on other web services
- Test time: 300 seconds
- SOAP message payload size: 262 bytes

Table 4.13 – Results for Test 10

	Apache Tomcat	Oracle Web Logic
Average response time [ms]	152	579
Average throughput [transactions/sec]	57,9	20,2
Number or error or missing responses	344	0

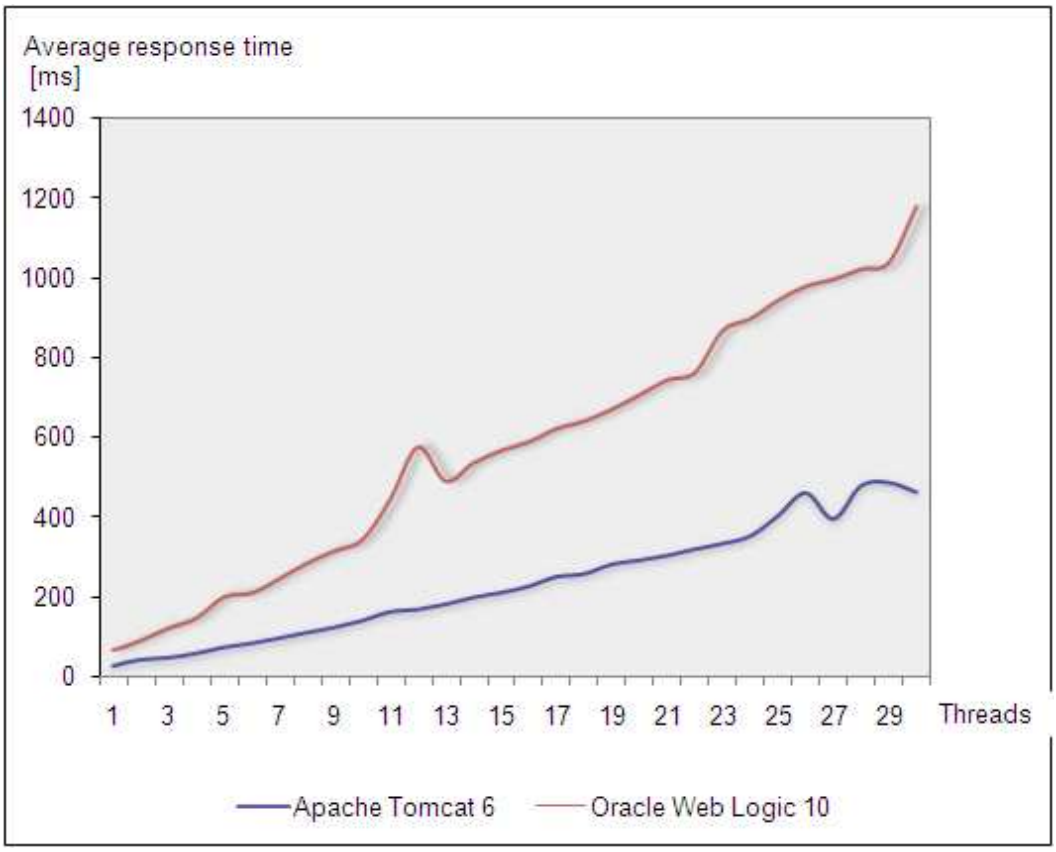

Figure 4.18 – Average response time for Test 10

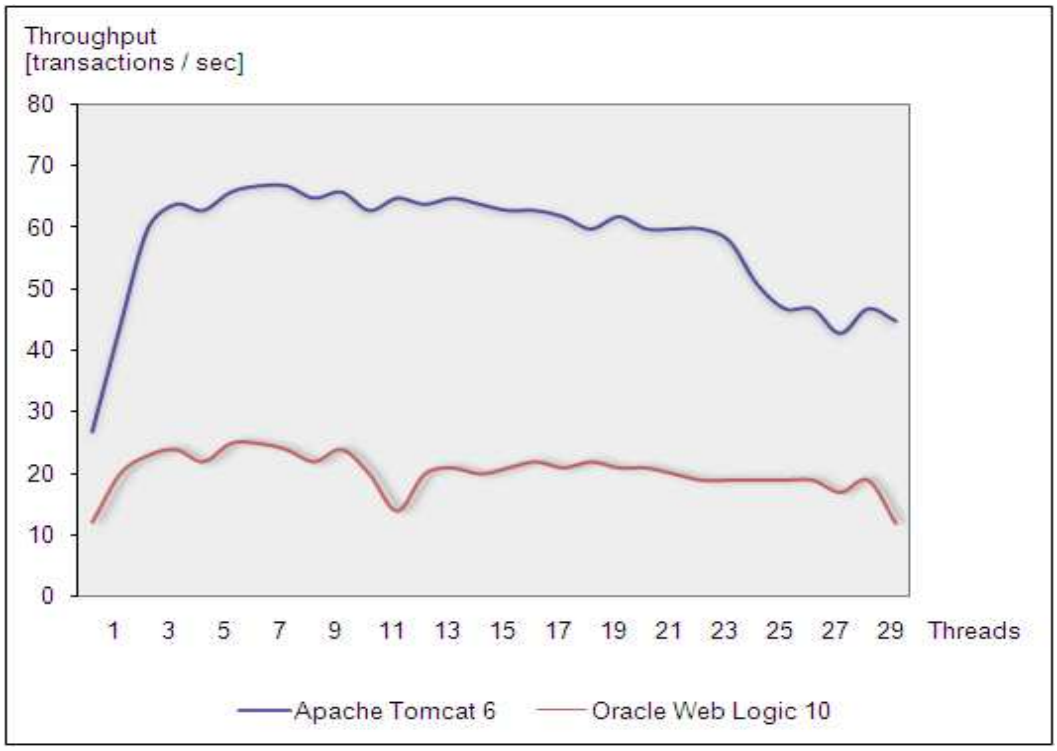

Figure 4.19 – Throughput for Test 10

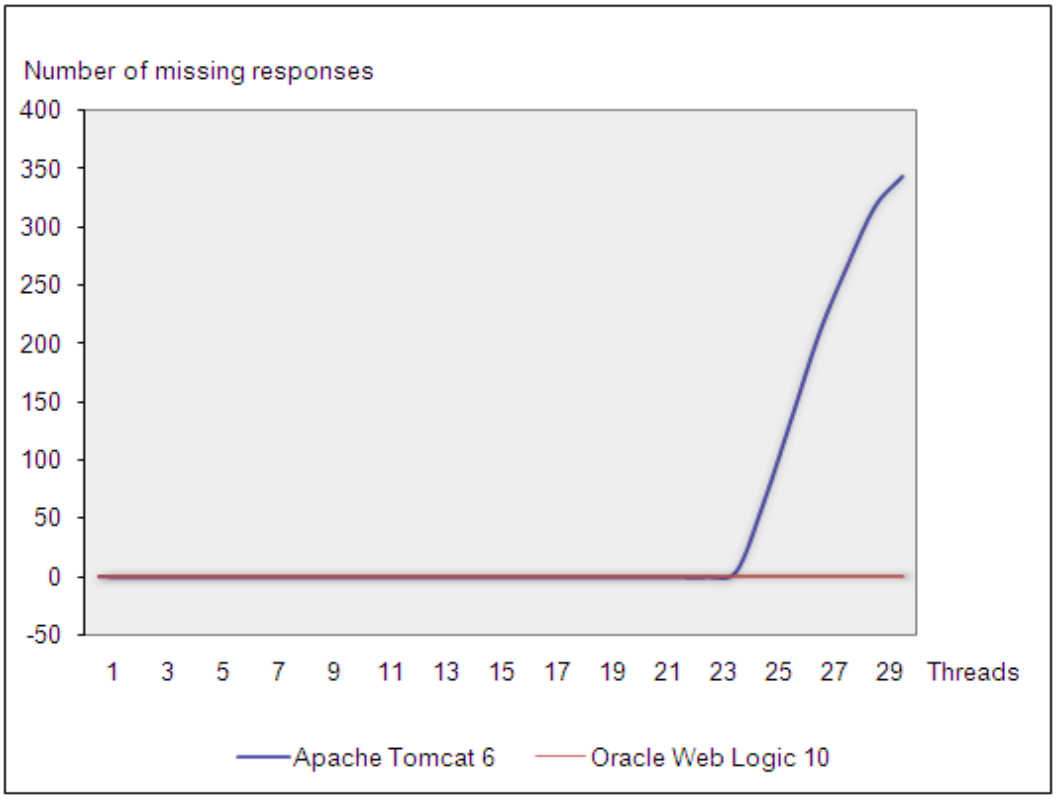

Figure 4.20 – Number of error or missing responses for Test 10

This test (Table 4.13) showed similar server behavior characteristics like the previous tests for this environment. Apache Tomcat is much faster (380%-550%), mainly because of the problems with handling higher load of Oracle Web Logic in second part of the test. Moreover Apache Tomcat was stable in each probe and all obtained results were very similar. On the other hand test results of Oracle Web Logic were quite diversified. Also according to figure 4.18 the average response time within single test was much differentiated for the given number of active threads. As it was already noticed in previous test (4.2.3) the interesting issue is related with refusing requests by Apache Tomcat when it's very loaded. Also when approximately 24 threads were generating requests, Apache Tomcat started to turning down pending requests. As it showed last test 4.2.3 it prevented from loosing stability and increasing response time under same degree of load. In the figure 4.19 we can see that corresponding value for 24 thread load is around 350 ms which is similar to the last test 4.2.3 (370 ms) when after some time Apache Tomcat was also refusing another requests. Admittedly the time of responses is getting higher quite rapidly but still it remains much shorter than Oracle Web Logic which was completely overloaded and unstable in this test.

4.2.5 Test 11: Ramp-up performance stress test

This test is focused even more on the behavior under extremely high load. That is why it starts already from 25 threads that generate the load.

The test configuration is as follows:

- Initial number of threads generating requests: 25
- Final number of threads generating requests: 50
- Injection profile: ramp-up
- Environment: web service dependent on other web services
- Test time: 300 seconds
- SOAP message payload size: 262 bytes

Table 4.14 – Results for Test 11

	Apache Tomcat	Oracle Web Logic
Average response time [ms]	510,75	1348
Average throughput [transactions/sec]	47,5	16,9
Number or error or missing responses	1392	0

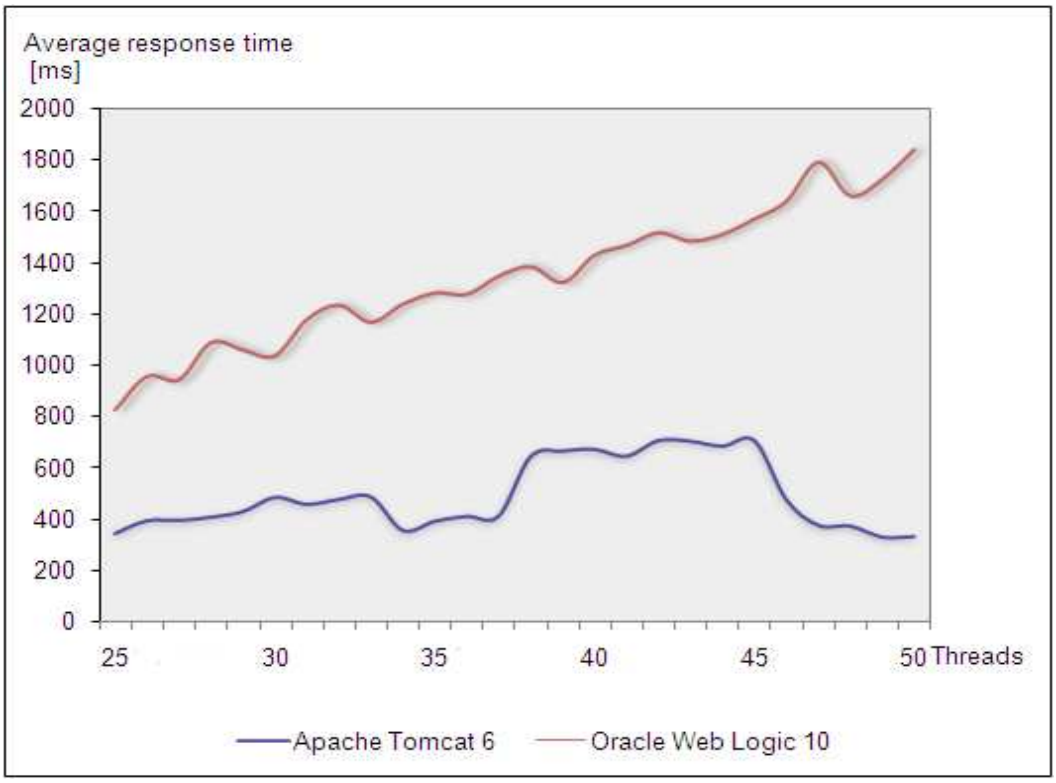

Figure 4.21 – Average response time for Test 11

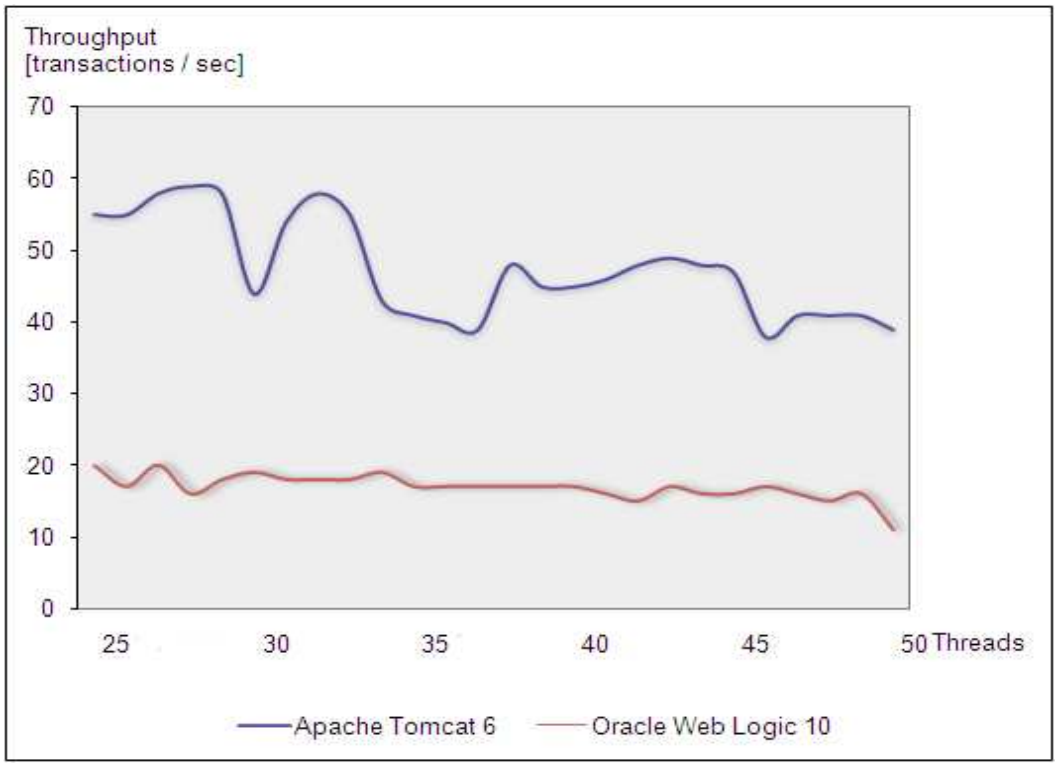

Figure 4.22 – Throughput for Test 11

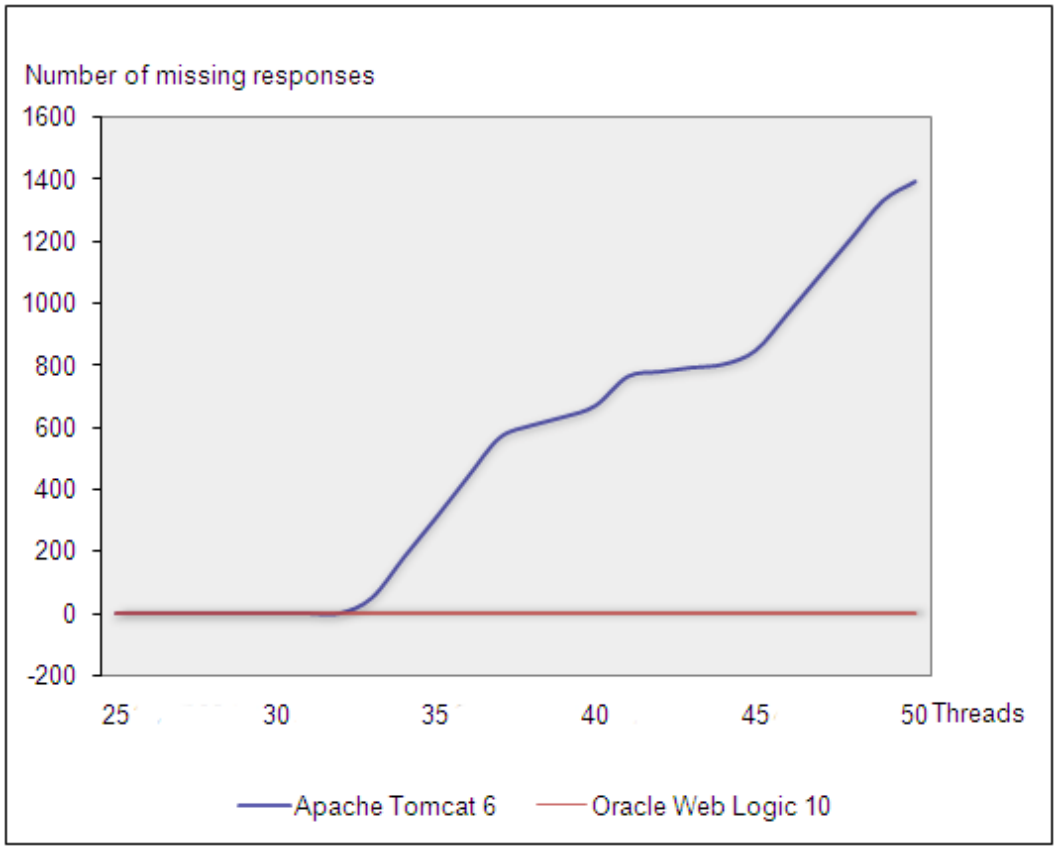

Figure 4.23 – Number of error or missing responses for Test 11

This test only confirms the results and conclusions from two previous tests (4.2.4 and 4.2.3). Difference in performance between the servers turned out to be smaller than in previous tests but again Apache Tomcat was faster 164% (Table 4.14). In the figure 4.22 we can see that this time response times of Apache Tomcat were more diversified but eventually in last 20 seconds of test (from 45th thread) got aligned and stabled again.

4.2.6. Test 12: Burst load test

This test is to examine the behavior of servers when they have to deal with rapidly changing very high load (burst test). After 10 seconds of idle server state, then all 35 threads immediately start sending requests to the server. This number of thread as showed last tests makes server overloaded. After 40 seconds the server is becoming idle again for 10 seconds, and then the load is generated again. The period when server is totally idle is not shown at the charts.

The test configuration is as follows:

- Number of threads generating requests: 35
- Injection profile: periodical big-bang
- Burst time 40 seconds
- Time of break between bursts: 10 seconds
- Environment: web service dependent on other web services
- Test time: 180 seconds
- SOAP message payload size: 262 bytes

Table 4.15 – Results for Test 12

	Apache Tomcat	Oracle Web Logic
Average response time [ms]	582,2	1182,4
Average throughput [transactions/sec]	56,027	22,873
Number or error or missing responses	6	0

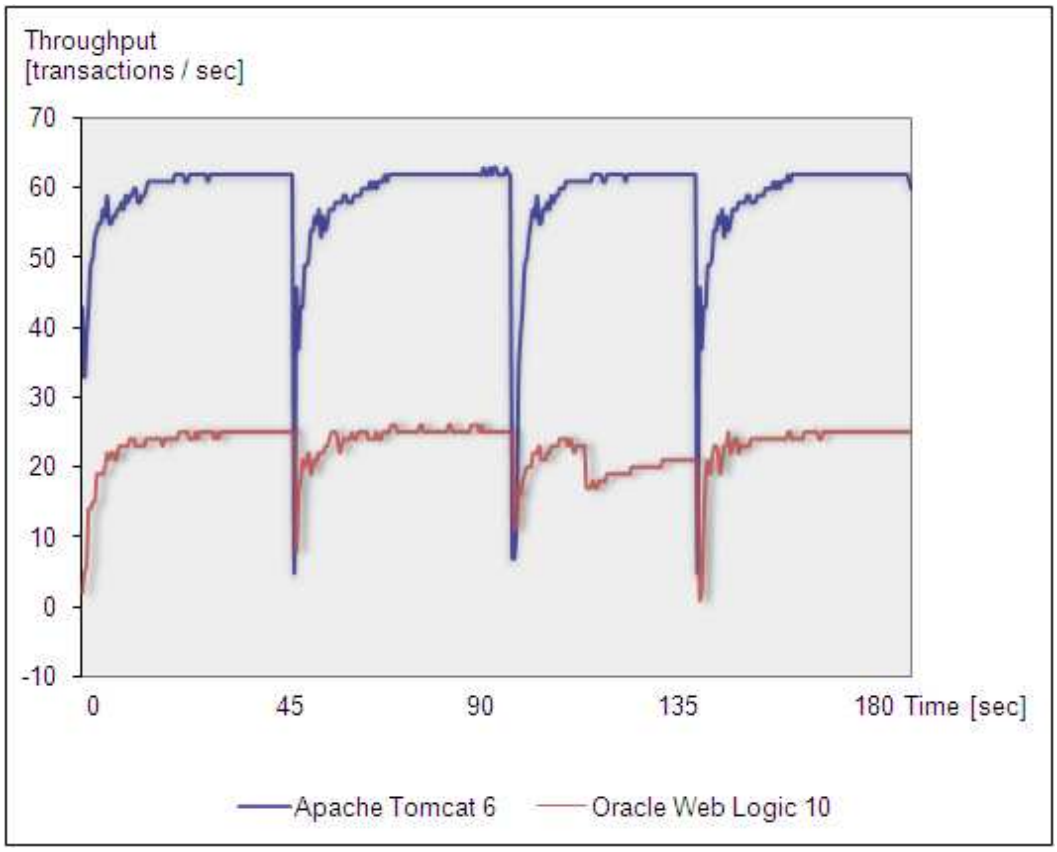

Figure 4.24 – Throughput for Test 12

In case of Oracle Web Logic application server we can observe some anomalies (figure 4.25) but the load within this test didn't cause any greater problems for the servers. Apache Tomcat didn't respond only for six requests which is meaningless. Such a small number of error responds (comparing to test 11) is most probably caused by 10 seconds breaks between the load bursts. During this time servers could attend to all pending requests. The overall performance (Table 4.15) was again better in Apache Tomcat case (102%).

4.3 Very complex SOA environment

Tests conducted in this section are aimed to compare the performance of both web servers in case when they are handling very complex SOA environment.

4.3.1 Test 13: Simple 1 thread load test

Since servers deal with extremely elaborated SOA environment the number of threads that generate requests should be much lower than in last two simpler environments. This test compares performance for just 1 thread.

The test configuration is as follows:

- Number of threads generating requests: 1
- Injection profile: big-bang
- Environment: a web service relies on multiple web services
- Test time: 180 seconds
- SOAP message payload size: 260 bytes

Table 4.16 – Results for Test 13

	Apache Tomcat	Oracle Web Logic
Average response time [ms]	199,797	469,980
Average throughput [transactions/sec]	4,722	1,908
Number or error or missing responses	0	0

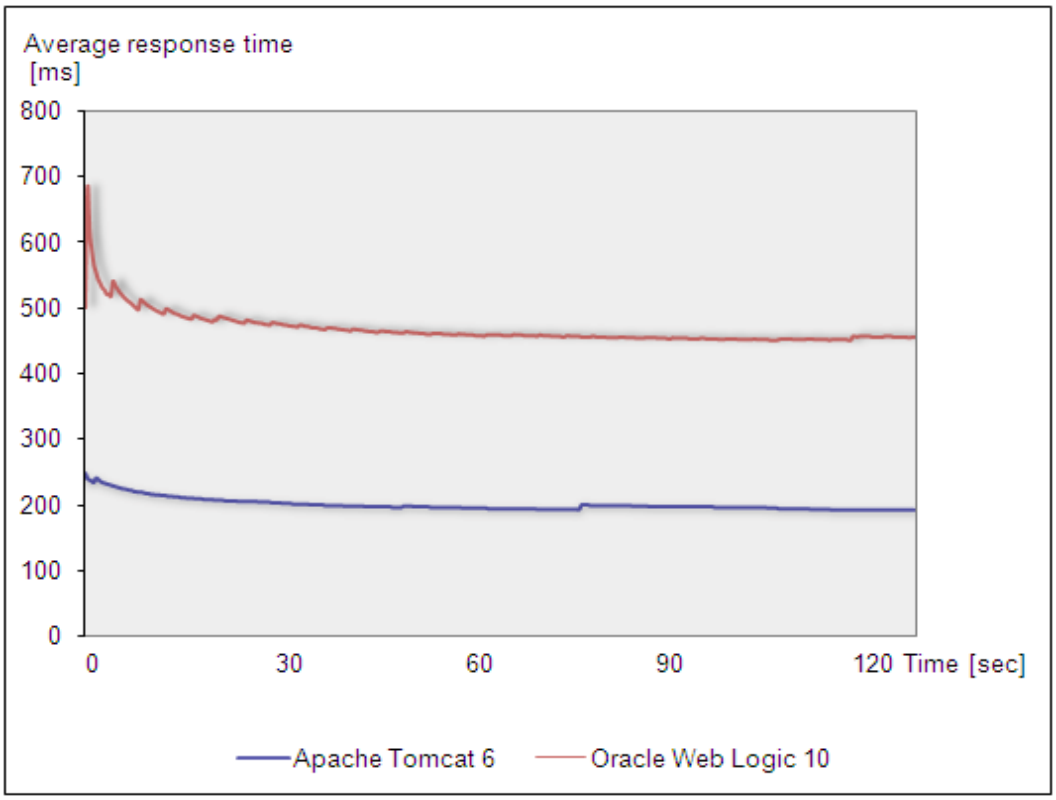

Figure 4.25 – Average response time for Test 13

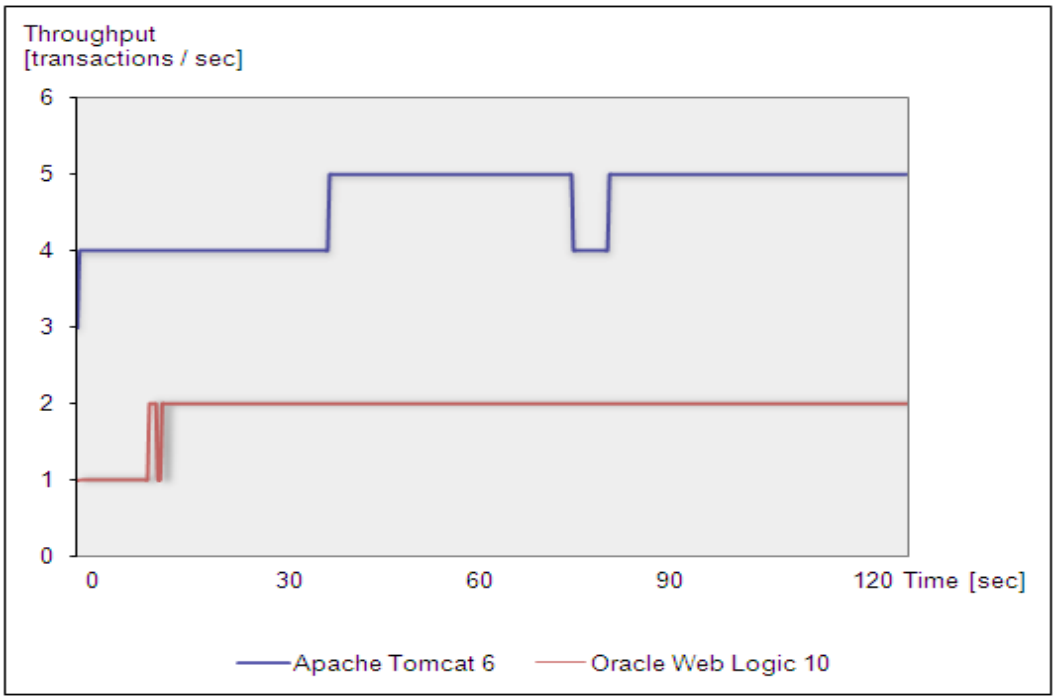

Figure 4.26 – Throughput for Test 13

This test showed that even if there is just one thread (no parallel requests) it took relatively lot of time to generate response (Table 4.16). For Apache Tomcat it was on average 200 ms. Oracle Web Logic similar as in previous tests for the environment consisted of web service and other depending web services (description in chapter 3) was slower for approximately 135% (469,980 ms). Both servers were stable which is understood for just 1 thread (Figure 4.26).

4.3.2. Test 14: Ramp-up performance test

During this test the number of threads generating request is constantly increasing from 1 to 3. This test is aimed to compare performance of servers for this kind of SOA environment.

The test configuration is as follows:

- Initial number of threads generating requests: 1
- Final number of threads generating requests: 3
- Injection profile: ramp-up
- Environment: a web service relies on multiple web services
- Test time: 300 seconds
- SOAP message payload size: 260 bytes

Table 4.17 – Results for Test 14

	Apache Tomcat	Oracle Web Logic
Average response time [ms]	247,769	611,404
Average throughput [transactions/sec]	7,217	2,506
Number or error or missing responses	0	0

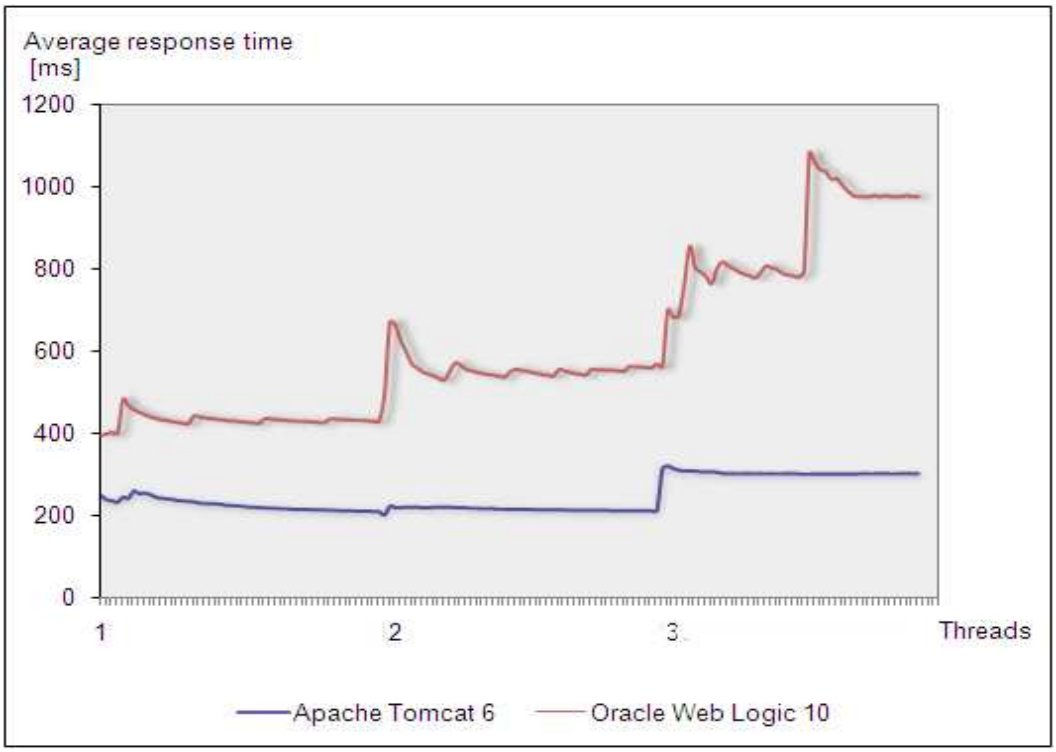

Figure 4.27 – Average response time for Test 14

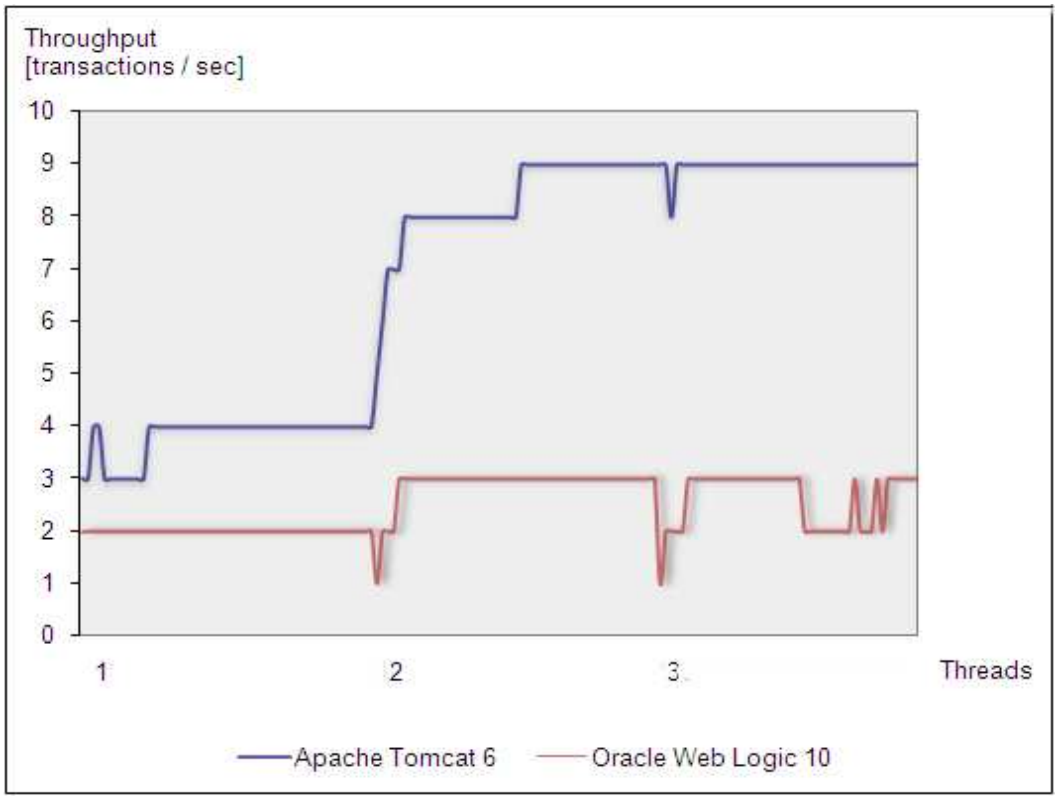

Figure 4.28 – Throughput for Test 14

This test results another time confirms that Apache Tomcat is much faster with complex environments (Table 4.17). Also in this case the average response time was 147% lower than Oracle Web Logic. Moreover according to Figure 4.27 and 4.28 after the second thread joined Apache Tomcat's average response time hardly increased but in Oracle Web Logic case this difference was bigger. After third thread in both servers response time has increased significantly but Oracle Web Logic was less aligned.

CHAPTER 5

CONCLUSIONS

The main objective of this thesis is to compare performance of two popular application servers Apache Tomcat and Oracle Web Logic server in terms of Service Oriented Architecture applications. For this reason three different SOA-based applications has been implemented. All tests were conducted using SoapUI as a load generator and the main testing tool as well. The main types of performance tests used in experiments are load test with a constant load degree, load test with increasing load (ramp-up), and burst load test with diversified load degree. The results were analyzed against average response time (latency), throughput (requests per second), and number of error or lacking responses. All measurements were conducted under different load. For verifying correctness of responses, responsible was SoapUI.

Performance load tests for which all results are available in Chapter 4 were divided into three main parts. Each part represents one SOA environment with different degree of complexity, which was deployed to the servers.

The first SOA environment consists of just a single self-dependent web service. It was necessary to use many threads that were generating load in order to make server overloaded. Nevertheless amongst all six tests that were conducted for this

environment, none of them was able to make server significantly overloaded or force server to turning down or responding by incorrect responses. Even when 40 threads were used to generate requests, both application servers were stable. In most of the simple load tests where the constant number of threads was used, faster was Oracle Web Logic. However differences weren't really big. It wasn't more then 1,1,%. The chart (Figure 4.6 and 4.5) for test 4.1.3 of 30 thread load shows that throughput is unaligned only at the beginning of the test. After approximately 10 seconds it keeps constant level aligned to about 720 transactions per second. Both servers don't have either upward or downward trend which means that they are able to handle this load in a long term perspective. It applies to all kinds of experiments that were conducted for the environment of single web service.

While in simple load tests (Subsections 4.1.1, 4.1.2, 4.1.3) at the beginning all threads simultaneously were starting to generate requests, in test 4.4 the profile injection was smooth. In the tests with ramp-up injection profile the situation became different. In this case Oracle Web Logic server needed more time on average to generate a response SOAP message. Test 4.1.4 starts from just one thread and ends up with 40 threads that are simultaneously generating a load. The change of the injection profile was sufficient to cause that Oracle Web Logic wasn't faster at this time. However the difference was just around 2%. Despite quite high load in the end of the test (35-40 threads) both servers behaved stably. It's easy to conclude that for application where rapid-type injection profiles occur Oracle Web Logic may be a better choice as an application server.

Test 4.1.5 aims to increase the frequency of injections and the impact of load injection towards the final test result. The main goal was to examine the behavior of servers when they have to handle rapidly changing load. The rule that Oracle Web Logic server is better in handling this kind of diversified load, proved to be true also in this test (4.1.5). Again the differences in average response time weren't very big. Overall it was approximately just 0, 9%.

At first impression the characteristics of average response times (figure 4.9 and figure 4.10) look quite similar. In both of them it is difficult to observe any anomalies. But after more careful observation of figure 4.10 it is possible to see that Oracle Web Logic Application Server for every subsequent burst of load needed less time to generate corresponding responses (except the last 4^{th} burst). This pattern does not apply to Apache Tomcat.

In the last test for the simple SOA environment consisted of single self-dependent web service the size of payload size of SOAP XML request message was significantly increased to 129 890 bytes. In real world such big request messages aren't very common but it can happen in case of SOAP response message from server to a client. In order to examine performance of servers for increased message payload size, the load test with ramp-up injection profile was applied (test 4.1.6). The conducted test doesn't bring any lurid results. The difference in performance between the servers is equal to 2, 5% in favour of Apache Tomcat. To remind in the corresponding test with low payload size it was 2% difference. The conclusion from this test is that XML message payload size doesn't have any significant impact which could distinguish both application servers. It just doesn't favour one of them more than the other.

The second part of experiments was focused on more complex SOA environment. It is not just a single web service but few of them, collaborating with each other. The detailed description is given in Chapter 3.

Likewise in the first part of chapter 4, experiments for this environment began from a series of simple load tests with the constant load of 5, 15, and 25 threads. It is important to take into account significantly higher complexity of this environment. In the first's experiments, 5 or 15 threads make server already very loaded unlike to tests in the previous simple SOA environment.

All three load tests with a constant load degree (Subsections 4.2.1, 4.2.2, and 4.2.3) explicitly proved that Oracle Web Logic application server is not very swift with handling complex SOA-based applications. Apache Tomcat in all cases was much faster. In the test 4.2.2 which used 15 threads to generate a load, the throughput became slightly smaller (10, 5%) in comparison to the first test 4.2.1 of 5 threads. In contrary Apache Tomcat in the test in Subsection 4.2.2 (15 threads) reached its maximum throughput. The difference of performance in the first in Subsection 4.2.1 test was equal to 153% in favour to Apache Tomcat. This is already very big difference and under heavier load it became even bigger.

Load test with ramp-up injection profile (Subsection 4.2.4) which starts with 1 thread and ends up with 30 threads generating a load, showed similar correlation. Apache Tomcat was much faster (280%). Moreover Apache Tomcat was stable in each probe and all obtained results were very similar. On the other hand test results of Oracle Web Logic were quite diversified. Average response time was from 579 ms to 844,77 ms. This kind of behavior of Oracle Web Logic has been already noticed in burst test in Subsection 4.1.5.

Another issue is related with error responses. When approximately 24 threads were generating requests, Apache Tomcat started turning down pending requests. The average response time of Apache Tomcat for certain load level is increasing quite vastly, it still remains much shorter than Oracle Web Logic which was completely overloaded and unstable in this test.

Another ramp-up load test (Subsection 4.1.6) was intended to be stress performance test which is to show what happens under extremely high load. The results and behavior patterns were similar as in previous test in Subsection 4.1.5. Both servers had some problems with such a heavy load, but again Oracle Web Logic didn’t return any error responses while Apache Tomcat returned 1392 during the whole test.

Burst test in Subsection 4.2.6 only confirmed the patterns that were observed in previous tests. The only difference between previous stress tests is number of error responses which this time didn't occur. Most probably it was caused by 10 seconds breaks between the bursts of load. Again the overall performance was again better in case of Apache Tomcat (102%).

In the last part experiments were conducted under extremely elaborated SOA environment. The detailed description is placed in chapter 3. Because of the complexity of the environment, number of threads that generate requests was lower than in last two simpler environments.

All tests conducted on this environment explicitly showed that similarly like in previous part (Section 4.2) Oracle Web Logic is definitely slower than Apache Tomcat application server. It applies to all tests, which were conducted on this complex SOA environment. Apache Tomcat had 135% and 147% shorter average response times.

To summarize, when servers were handling simple web service, both application servers had very similar performance. Usually Oracle Web Logic was a few percents faster where the degree of load was changing fast. In smooth tests without ant rapid load bursts Apache Tomcat had better performance.

For more elaborated SOA environments the difference in performance was significant. The higher load was the bigger difference was in performance between Oracle Web Logic and Apache Tomcat application server.

Apache Tomcat was not only faster in all conducted tests but also more stable, and behaved more predictably. Oracle Web Logic under high load was sometimes losing stability and the times of response were becoming very scattered. Oracle Web Logic application server even under moderate or high load (comparing to Apache Tomcat) during the simple load test (constant load degree) had downward

trend of throughput. It is very bad sign in long term perspective work. Heavy load stress tests showed that Apache Tomcat is not able to handle all the requests. For this reason turns some of them down. Nevertheless even under extremely heavy load Apache Tomcat is relatively stable. Oracle Web Logic application server tries to response to every request but on the other hand its performance in this kind of load is very poor comparing to Apache Tomcat.

Regarding to the architecture of application server its middle tier is composed mainly of http web server and EJB server, which includes also implementation of J2EE specification. These two components have the main influence on overall performance of two compared application servers. The tests showed that the way of implementing J2EE specification can be crucial for overall application server performance. Moreover Oracle Web Logic application server needs much more memory. It could be one of the reasons of relatively low Oracle Web Logic performance in complex SOA environments. Another reason is implementation of J2EE specification supported by Oracle Web Logic. The HTTP server delivered by Oracle Web Logic should be excluded as a reason of lower performance. As it showed load tests for the simple Web Service, the performance for this environment was similar for two compared application servers. It means that both HTTP servers as a single entities have most likely similar performances. In test for the complex environments important issue is also related with reliability. The way of dealing with a very high load was different in two compared application servers. The implementation offered by Apache Tomcat was faster but under high load server was denying oncoming requests. Therefore the client couldn’t use the service at that moment while Oracle Web Logic was always handling requests.

5.1 Future research opportunities

The main goal of conducted experiments in this thesis was to compare the performance of two application servers Oracle Web Logic and Apache Tomcat in different SOA applications (environments).

There are other factors that may have impact on the performance of the servers. One of them is operating system. Tests conducted within this thesis weren't aimed to compare server performance also against operating systems. All of them were carried out under Windows XP. It is possible that one application server has better performance under certain operating system while second server behaves in the opposite manner. This can be the subject of another research focused on this issue.

All conducted experiments in this thesis were based on the server with Intel Xeon 2 core processor. Nowadays we can observe the trend of increasing number of cores rather than increasing CPU clock rate. That's why application servers can be also compared against scalability. It could point out which server can assure more efficient work, while computing resources (number of cores) are being increasing. In this research could be also included different operating systems. Scalability is important especially in commercial services which have to keep their customers satisfied and leave the door open for an expected growth and expansion.

Another issue of concern related with scalability can be problem of distributed work load among multiple instances of application servers, run on multiple or a single physical server. Incoming requests based on load balancing algorithms can be routed to other server instances in a certain clusters.

REFERENCES

[IBM2009] IBM Smart SOA solutions September 2009 White paper, *Making sense of SOA and today's IT innovations*

[Endrei2004] Mark Endrei, Jenny Ang, Ali Arsanjani, Sook Chua, Philippe Comte, Pål Krogdahl, Min Luo, Tony Newling (April 2004) *Patterns: Service-Oriented Architecture and Web Services IBM*

[Josuttis2007] Nicolai M. Josuttis; O'Reill (2007), *SOA in Practice*

[Molyneaux2009] Ian Molyneaux Beijing (2009), *The Art of Application Performance Testing*

[Priyanka2008] Priyanka Mane , Budhaditya Das, Aztecsoft Ltd.(2008); *Scalability Factors of JMeter In Performance Testing Projects*

[Brittain2009] A. Van Abs, Jason Brittain, (2009); *Migrating Applications From Oracle WebLogic to Apache Tomcat*

[Salter2008] David Salter, Frank Jennings (2008) *Building SOA-Based Composite Applications Using NetBeans IDE 6*

[Zambon2007] Giulio Zambon, Michael Sekler (2007) *Beginning JSP JSF and Tomcat Web Development*

[Gao2007] Tom Yuan Gao (2007), *The Complete Reference To Professional SOA With Visual Studio 2005*

[Newcomer2005] Newcomer, E. and Lomow, G. (2005). *Understanding SOA with Web Services.*

[Rosen2008] Mike Rosen, Boris Lublinsky, Kevin T. Smith, Marc J. Balcer (2008), *Applied SOA Service Oriented Architecture and Design Strategies*

[Lawler2008] James P. Lawler,H. Howell-Barber (2008) *Service-Oriented architecture SOA Strategy, Methodology and Technology*

[AppServ2010] Application server definition, Last Date Accessed: 03/07/2010, Url: http://www.service-architecture.com/application-servers/articles/application_server_definition.html

[W3C2010] W3C, Web Service Description (WSD), Last Date Accessed: 03/07/2010,
Url: http://www.w3.org/2004/Talks/1117-hh-wsdl20/slide6-0.html

[OracleCorp2009] Oracle Corporation (2009) Oracle Fusion Middleware Introduction to Oracle WebLogic Server, Last Date Accessed: 03/07/2010,
Url: http://sqltech.cl/doc/oas11gR1/web.1111/e13752.pdf

[Poskanzer2010] Jef Poskanzer Web Server Comparisons, Last Date Accessed: 03/07/2010,
Url: http://www.acme.com/software/thttpd/benchmarks.html

[Nichols2010] Linux Is The Web Server's Choice, Last Date Accessed: 03/07/2010, Url:
http://pauillac.inria.fr/~lang/hotlist/free/bench/0_4537_2196115_00.html

[LitespeedTech2010] Web Server Performance Comparison, Last Date Accessed: 03/07/2010, Url: http://www.litespeedtech.com/web-server-performance-comparison-litespeed-2.1-vs.html

[SunJava2010] Sun Java System Web Server Performance Benchmarks, Last Date Accessed: 03/07/2010,
Url: http://www.sun.com/software/products/web_srvr/benchmarks.xml

[Zeuscat2010] Apache Webserver Benchmarks, Last Date Accessed: 03/07/2010,
Url: http://www.zeuscat.com/andrew/work/aprbench/

[Pulier2006] Eric Pulier, Hugh Taylor (2006) *Understanding Enterprise SOA*