Project Report

Ashlesha Bansode, Ashish Kumar.
Computer Science and Electrical Engineering
University of Maryland, Baltimore County
ashbansode@umbc.edu , akumar5@umbc.edu
December 14, 2018

Abstract

To handle large web requests from the user, large number of servers are required which depends on the number of users. Due to the increase in the number servers a Load Balancer is required which receives the web traffic and distributes the request among the server. Load Balancer can be implemented using both hardware and software depending upon the requirement. The purpose of this project is to implement a simple Load Balancer using the least connection algorithm and identify connection rate, response time, throughput and identify the failed connection.

Keywords—Web-Services, Apache Axis 2, Load Balancer, Service Directory, WSDL, Least Connection Algorithm.

INTRODUCTION

This project will build a distributed system using apache axis2 as web service, apache tomcat for creating different servers and load balancer for distributing traffic across the cluster of servers to improve responsiveness and distribute the load on the least utilized server. The main task of the load balancer is to minimize response time, optimize resource use, maximize throughput, and avoid overload of any single resource. There are many ways to implement web-services according to the requirement of the user.

Our implementation is a simple, scalable system with two Load Balancer to avoid single point of failure. In case if there was only a single load balancer there is a chance of single point of failure when it stops working. The motive of the project was to implement this on a small number of web-services request, we decided to deal with six number of servers where two servers acts as a Load Balancer and the rest four are for performing the task depending on receiving the request from the load balancer and the number of tasks stacked on each server queue.

SYSTEM DESIGN

The design of this project has been kept simple but robust and scalable at the same time. There are six servers created in total in which two of them acts as a Load Balancer and rest as a computational server. Each request from the client is sent to the service directory which has the details about each server and the operations that each server can perform. The servers capable of performing the task are list together and sent to the load balancer. Load Balancer finds the server with least connection and add the task into the queue of the server. The decided server performs the task and sends back the output to the client.

IMPLEMENTATION

The request from any user first goes to the service directory. It is a shared infrastructure for managing, administering and storing all the information about each server and the operations or task that can be executed by each server. We used the Apache Axis2 framework to add Web services interfaces to Web applications and used its flexibility of choosing between the REST and SOAP API. Although the whole project only implements the SOAP API.

Generated WSDL (Web Services Description Language) file which contains the set of definitions that describes a Web Service and information needed to access and use a Web Service. It describes what the web services does, how it communicates and where it resides.

Implemented java web services client and created stubs by using proxy objects to access remote services. At request time, the proxy accepts the Java method call and then translates to xml message. At response time, the proxy receives the xml message translates to java objects and returns result to client application.

Created local instances of tomcat server and deployed Java web services server on the tomcat for initial stag of testing and later remote server were create on different system and web requests were handled by remote servers.

There are four operations defined inside the service directory. Each server can perform few numbers of tasks but not all of them. The task each server can be perform is defined inside the service directory. While a request is made from the user, the request goes to the service directory.

Service directory has the information of the servers that are capable of performing each action as each server cannot perform all the operations. Service directory creates a list of the servers that can perform the action and send it to the lad balancer.

Load Balancer improves the distribution of workload and assign the task to the least occupied server among all the servers actives during a particular instance of time. If all the servers in the list sent by the service directory are busy performing other tasks, load balancer finds the server with least load and add the task in server's queue. The task that can be performed are Addition, Subtraction, Multiplication, Division and each server can only perform the task which are present in their service directory.

Client must select which operation to perform among four of them and input two numbers. This client request is sent to the server which has load balancer and service directory. We have two load balancers implemented. It checks which load balancer is up and running and sends the request to that load balancer and service registry.

Service directory picks the list of servers which has the desired operation and then checks if all those selected servers are up and running. It sends these lists of up and running servers to the load balancer. Load balancer receive the list of all the up and running servers with the desired operation.

Using these servers list, it checks which server has minimum load based on the least connection first algorithm. After selecting the server with minimum load, it sends the request to that server. The server processes that request and sends the result back to the load balancer. Ultimately this load balancer responds to the client with the result.

TESTING STRATEGY

There were various testing strategies used from initial phase of development to the final stage. Servers were intentionally stopped to check the smooth flow of the task to another working server. Two load balancers were created to avoid any single point of failure, loader balancers are the single gateway between the client and the server, so any problem between them can cause the whole process from working.

Load performance was closely measured, if a server has requests in its queue and other servers are available to perform the task then the load balancer should assign the task to severs with the least number of items in its queue, this also tests the proper implementation of the Least connection algorithm.

If there are task present in the server's queues and the server stops working, all the task in the queues must to be transferred back to the load balancer so that they can be assigned to the server which are up and running effectively.

Load testing were done which determine how a application performs based on a certain volume of users, generally it increases the volume of requests from the client in the specified duration of time.

Stress testing where done which is similar to load testing, except they are designed to test system performance with maximum capacity of requests and job. Basically, stress identifies the time when the system fails or crash and verifies if system continues to work at maximum capacity.

Average response time for different length of requests were measured and behavior of system was analyzed and checked fluctuations with less load and more load.

EXPERIMENTS / RESULTS

The web services were experimented with different strategies and few of them are mentioned below.

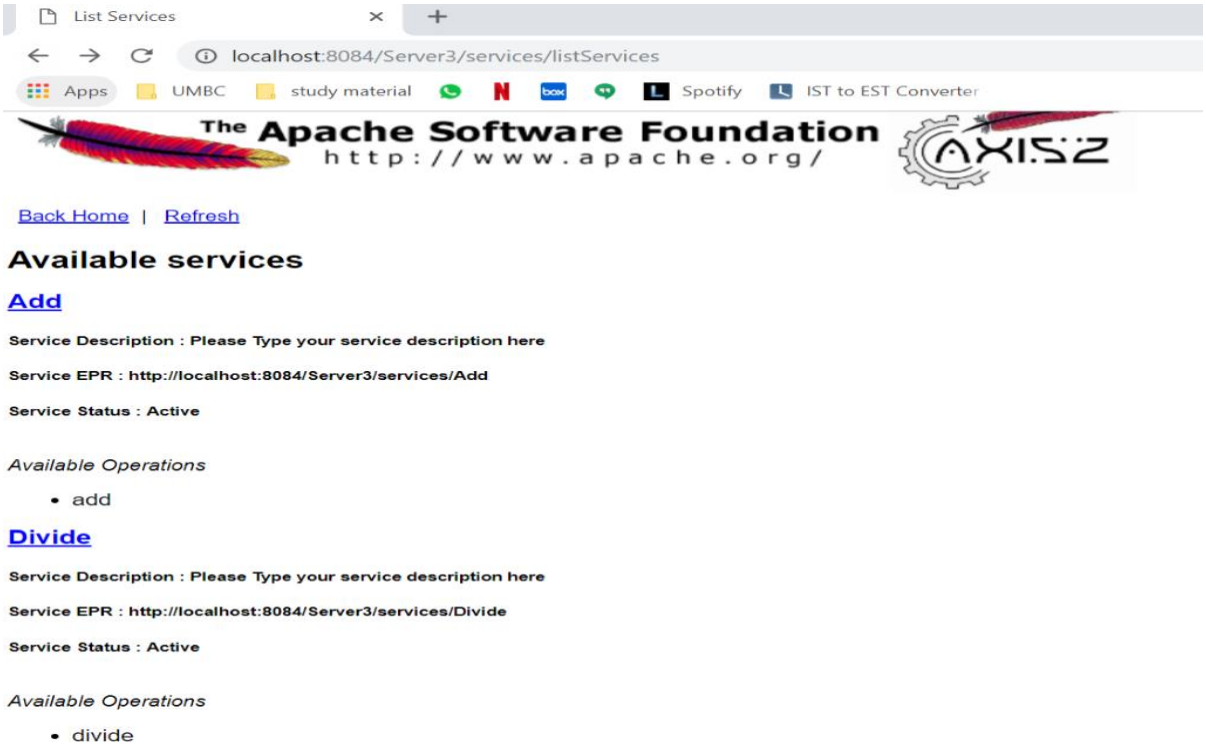

Fig: Services deployed on Apache Axis2.

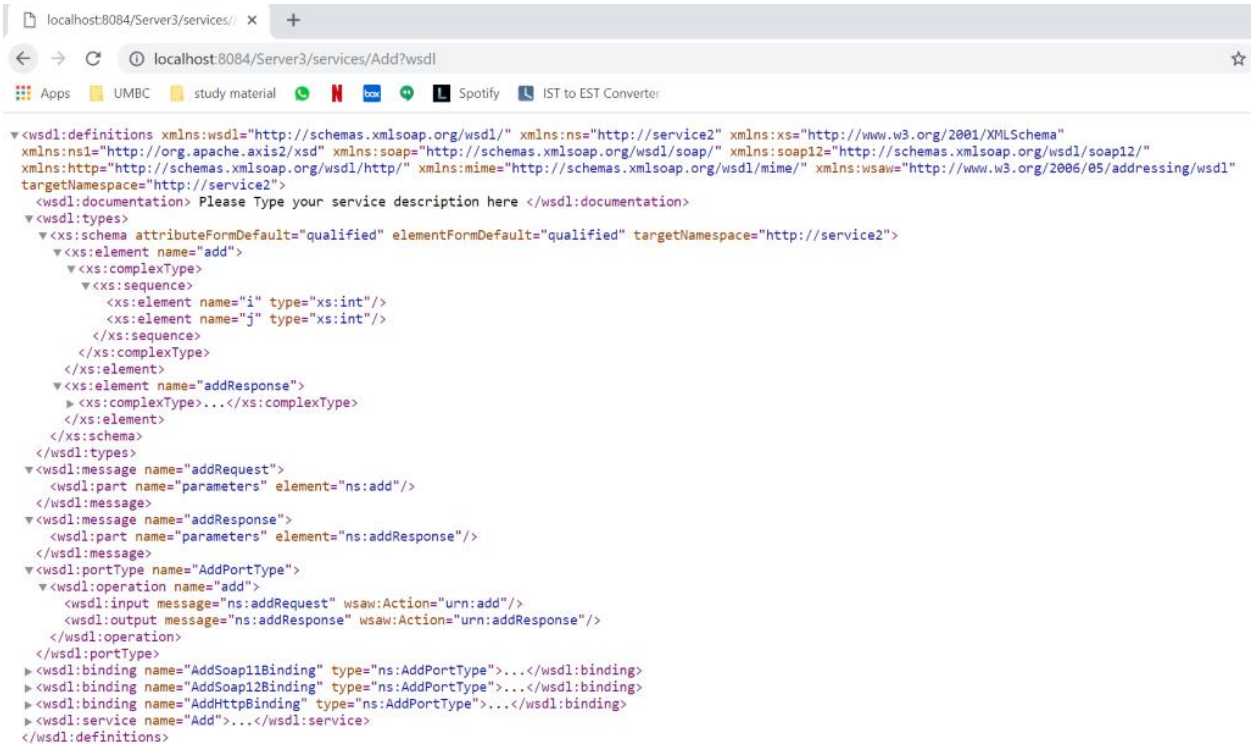

Fig: WSDL for Addition.

The below picture depicts the operations available on different server, Load Balancer selected for use and identification of server with least load which is capable of performing the operation.

Fig: Output after performing the selected operation.

REFERENCE

1. Dr. Cesare Pautasso, Computer Science Department Swiss Federal Institute of Technology (ETHZ), *Distributed Systems Web Services*. Retrieved from https://www.vs.inf.ethz.ch/edu/WS0405/VS/VS-050117.pdf

2. AVI Networks. *An Introduction to Load Balancer*. Retrieved from https://avinetworks.com/what-is-load-balancing/

3. Chris Richardson, October 12, 2015, *Service Discovery in a Microservices Architecture.* Retrieved from https://www.nginx.com/blog/service-discovery-in-a-microservices-architecture/

4. The Apache Software Foundation, November 16, 2017, *Apache Axis2/Java – Next Generation web-services.* Retrieved from http://axis.apache.org/axis2/java/core/

5. Angelo's Blog, January 23, 2011, *JAX-WS with Apache CXF.* Retrieved from https://angelozerr.wordpress.com/

6. Daniel E. Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith, *Maglev: A Fast and Reliable Software Network Load Balancer,*

7. Wikipedia, *Load Balancing Computing,* Retrieved from https://en.wikipedia.org/wiki/Load_balancing_(computing)

8. Wikipedia, *Directory Service*, Retrieved from https://en.wikipedia.org/wiki/Directory_service

9. Eclipse Documentation, *CXF- Web services*, Retreived from *https://help.eclipse.org/mars/index.jsp?topic=%2Forg.eclipse.jst.ws.cxf.doc.user%2Ftasks%2Fcreate_cxf_project.html*

10. Apache CXF, *DynamicClientFactory and JaxWsDynamicClientFactory,* Retrieved from http://cxf.apache.org/docs/dynamic-clients.html

11. Haseem Moqaddam, June 21, 2016, *Load Balancing with Apache Camel* .Retrieved from https://www.javacodegeeks.com/2016/06/load-balancing-apache-camel.html

12. MuleSoft, *Tomcat Clustering.* Retrieved from https://www.mulesoft.com/tcat/tomcat-clustering

13. Gobinath Loganathan, March 09, 2015, Retrieved from https://www.javahelps.com/2016/04/apache-axis2-hello-world-using-eclipse.html