Distributed Web Services System

Vineeth Reddy Anugu, Manogna Teja Pidikiti
Computer Science and Electrical Engineering
University of Maryland, Baltimore County
vanugu1, mpidiki1@umbc.edu
December 14, 2018

***Abstract*—In the distributed web services system we create, access and invoke web services in a distributed infrastructure. Multiple sites are supposed to host multiple web services. Each web service in a server has some load. We provide a service discovery mechanism for a client to ask for the service that it is looking for. Then we discover all the sites where the required web service is hosted and return the server with minimum load imposed on it, thus implementing load balancing through round robin.**

***Keywords*—Service discovery, load balancing, Request, Response, service, load, web service, distributed system.**

I. Introduction

Distributed system is a collaboration of individual components communicating with each other over a middleware layer. In this context, the components include basically the client/server network. Web services are the services which are offered from one component to the other over the Internet. To implement these web services in the distributed system, SOAP or RESTful architecture is used. The components used in the distributed system are written in Java using the spring boot framework and deployed on tomcat servers.

This report is structured as follows: Section 2 provides the overall design of the distributed system created and the different methodologies used for that. Section 3 states the assumptions made which help in going through the distributed system without any conflicts. Section 4 gives the insight into what has been done to develop a web services distributed system, including the service discovery and the load balancing part. Section 5 tells the different testing strategies used to test the complete working of the system. Section 6 analyses on the graphical notion of the working and the respective results. Section 7,8 summarizes and outlines the likely future work on this distributed system.

II. Design

In this distributed system we provide 4 services, namely Addition, Subtraction, Multiplication, Division. We deploy multiple servers which contain various services such as Addition and Subtraction in Server1, Multiplication and Division in Server2, Addition and Multiplication in Server3, and Division and Subtraction on Server4. Additionally, Server1 is also responsible for maintaining a service registry in the form of a hash map and also perform load balancing. Server1 must be initialized first as it maintains the hash map/registry. Each server sends a request called AliveRequest to Server1 when it starts, containing information such as the IP, port of that server and the services that the server provides. When Server1 receives these AliveRequests it stores the data received from these messages in the hash map so that these are utilized when needed. Similarly, if a server goes down, it sends a DeadRequest to Server1 so that the hash map is updated accordingly and their loads are reset.

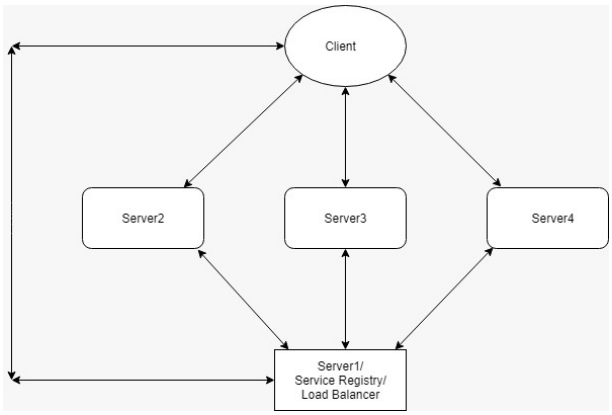

Figure 1. Architecture

Once all the servers are up and running, a client will establish an HTTP connection to Server1 (Registry) and send a WhichRequest where the client specifies the type of service that it requires and the load that it will impose on the server. When this request is sent to Server1 it takes the name of the service sent by the client and retrieves the IP and port of the servers that are able to provide that service and sends it as a response called WhichResponse back to the client. Once the information of the servers which provide the required service is retrieved, they are compared based on the load that has been imposed on them, if any and the server with minimum load on it is sent back to the client. The load that is retrieved from the WhichRequest is added to the chosen server's load in the hash map.

When the client receives the WhichResponse from Server1 which contains IP and port of the server containing the requested service, a new Service Request (Add, Minus, Mul, Div) is sent to the server with the obtained IP and port containing the input required to perform the needed service. This Request is accepted by the respective server which then, performs the service that has been requested by the client and sends back the result as Response.

III. Assumptions

- Each server has a minimum bandwidth.

- The network is reliable and stable.
- We store data of all servers in a service registry.
- The network is secure.
- All services impose the same amount of load.
- Each server and client must know the IP and port of the registry server (Server1).

IV. Implementation

First, we created 4 different Spring Applications as maven projects with SOAP Web Services and wsdl4j dependencies added to their pom.xml. We used the spring boot framework in java to create these spring web service applications. Next, we added the jaxb2-maven-plugin to the pom.xml. We then wrote .xsd file detailing the XML schema of the requests and responses being sent and received by these servers. The jaxb2 maven plugin automatically creates the classes for these requests and responses based on the .xsd file created. We created the configuration file detailing where the server is to be deployed and an Endpoint which contains the logic for the web services.

In the case of the registry server (Server1), we also wrote the logic to handle the Alive, Dead and WhichRequests. Next, we set up init() and destroy() methods which basically created an HTTP connection to send the Alive and DeadRequests to the registry server on starting and ending of the server respectively, so that the hash map is adjusted according to the server availability. We then established an http connection to the service registry from the client to post the WhichRequest containing a SOAP message which was written based on the XML schema of that WhichRequest. Then it resulted in a WhichResponse which was parsed to retrieve the data from the body of the response XML message. This data included the IP and port of the target server which contains the required service. Now, using this data another SOAP message was created which is the service request. This request was sent along with, the input required to perform the web service, and was posted on the destination server.

Load Balancing: For load balancing, we implemented a round robin technique where services were spread out to various servers. This was done by maintaining a counter for each server in the hash map and incrementing when a request was sent to that server. When a WhichRequest was received, the registry first found out all the servers that perform the given service and then compared the load counter that each server has, and return the server with the lowest count. This makes sure that various service executions were spread out over several servers instead of a single server being used consecutively.

V. Testing Strategy

- **Unit Testing**: First, we tested each web service by calling them one after the other and see if the results were accurate to what that service had to offer. We checked to see if all the servers sent an AliveRequest on startup and similarly a DeadRequest upon termination. Next, we tested if the WhichRequest was working correctly by asking for the services offered in the system and checking whether they return the relevant IP and port. After that, we ran all four services that are offered i.e. Addition, Subtraction, Multiplication and Division and checked for their results to see if they add up.
- **Scale/Load Testing**: The next order of testing we put the system through was load testing. We ran various services provided to see which server it would be directed to by the registry based on load. It was clear through our testing that the registry always chose the least utilized (loaded) server among the available relevant servers that provide the required service using a round robin like technique. Once a server was utilized for performing a service, it is used again after all the other alternative servers (for required service) have been put to use.
- **Resilience Testing**: Finally, we tested how our system adapts to random terminations and addition of servers to the system. We tried multiple combinations, like running all 4 servers at once, then terminating a few servers to make them unavailable. Next, we tried adding more servers while the system is already performing actions with the currently available number of servers. This shows that how our system facilitates for the various servers coming in and out of the system and how it affected the service discovery and load balancing.

VI. Results

We experimented by running various service discovery requests from our client and plotted their response times as shown in Figure 2. During this experiment we ran our discovery requests for various services while changing the topology of our web service network i.e. changing the number of servers that are running when the given requests are made, consistently. We observed that, irrespective of the scenario, the response times of any requests always lie in the range of 155-200ms range and didn't really show us any noteworthy effects of the changing topology as well as the type of service that was requested.

Next, we tested if our system was spreading out the load of services equally among all the servers. For example, when we ran the 'add' service, it first was executed by the server on port 8082. Whereas the next time around, it was executed on server with port 8083 if its load history is lower than port 8082. We tested this by sending 60 service requests(15 of each) and they were distributed evenly among servers based on the service type that was requested.

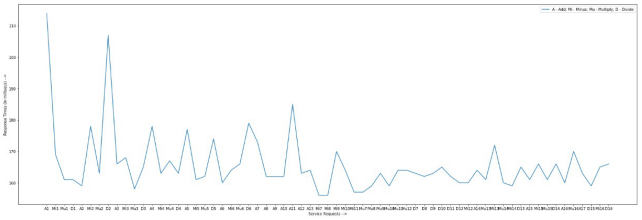

Figure 2: Response Times

VII. Conclusion and Future Work

SOAP and the spring boot framework was used to develop and coordinate the web services used in the distributed system. Service discovery and load balancing were implemented using a central registry which is a hash map, which stores the

information of every server along with their respective dynamic loads.

To improve this system further, a dynamic load balancing system besides using a round robin like technique can be implemented where the load will be distributed in real time, while the services are being performed and removed later when the service is executed.

ACKNOWLEDGMENT

Thanks to my teammate for his time and support.