# CMSC 621 PROJECT REPORT

A Distributed Web Service Framework

Submitted by - Rohan Gujarathi | Ajay Pal

## Contents

1. Abstract .......... 2
2. Introduction .......... 2
3. Architecture .......... 2
4. Working of the system .......... 4
   4.1 Web Services .......... 4
   4.2 Registry .......... 4
   4.3 Load Balancer .......... 5
   4.4 Client .......... 5
5. Assumptions .......... 6
6. Testing Strategy and Results .......... 6
7. Features .......... 8
8. Future Work .......... 9
9. Tools and Technologies Used .......... 9

References .......... 9

## 1. Abstract

The main goal of this project is to construct a distributed web-service system. This project will describe the infrastructure and architecture used to create the system. Furthermore, it will also describe how to discover the service and invoke them based on load distributed on the nodes. With certain assumptions involved, multiple sites will be hosting collection of web services

## 2. Introduction

This report discusses about the architecture of our distributed system, how we discover the service and how the load is balanced using load balancer. We start with explaining the architecture of the project and working of every component in the system. Next section discusses about the assumptions that we have followed by the testing strategy and the results. We also discuss the features of our system, tools and technologies used and the future scope of this project.

## 3. Architecture

Below is the diagram of our architecture

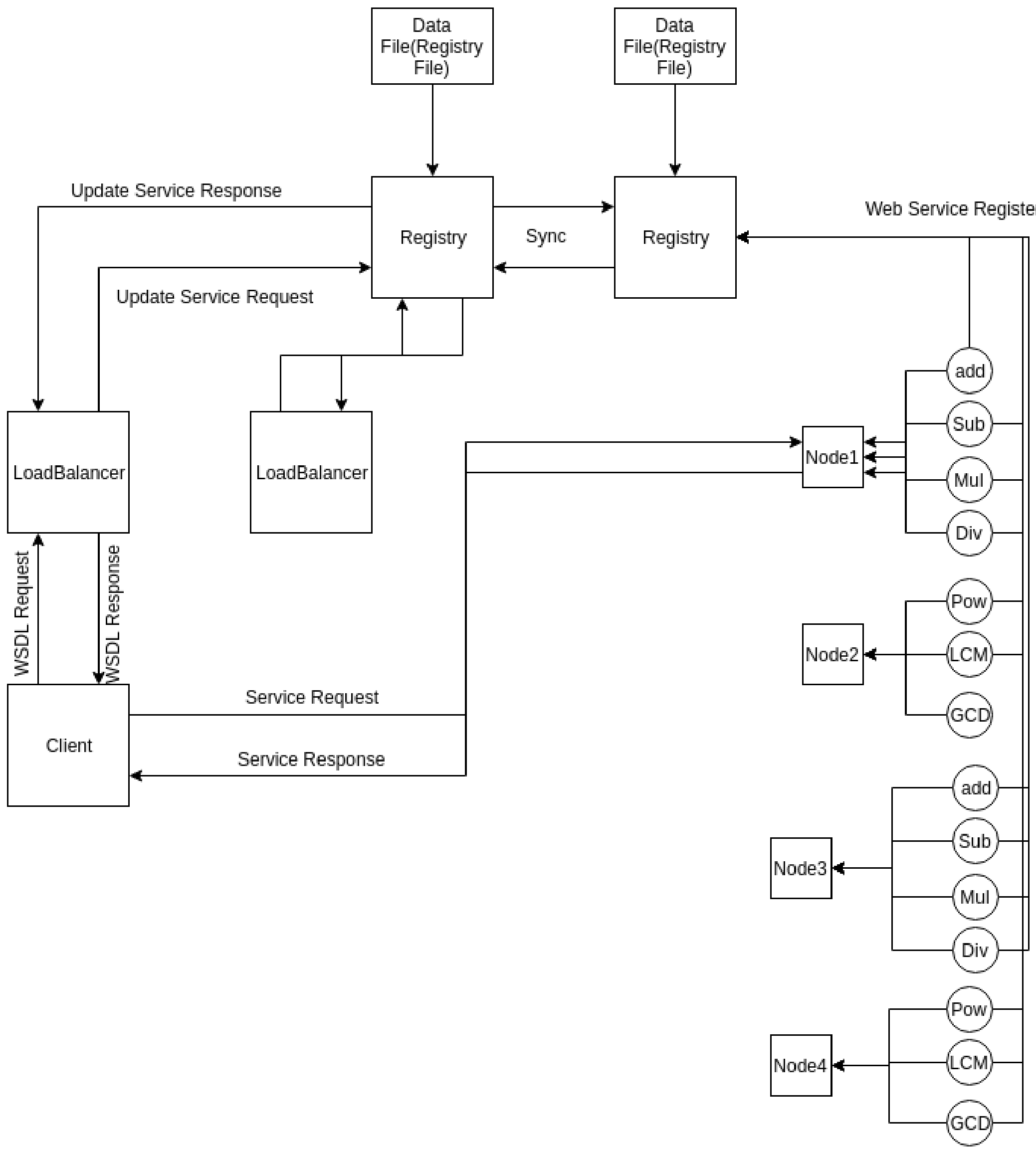

Fig 1. Architecture

1. All the services register to the registry
2. Load balancer contacts the registry and retrieves the data from registry
3. Client will contact the load balancer and retrieve the WSDL for the required service
4. Client will then contact the service using the WSDL received and get the required result.

# 4. Working of the system

## 4.1 Web Services

Below are the different web services which we have implemented

1. Add
2. Subtract
3. Multiply
4. Divide
5. Power
6. GCD
7. LCM

Different combination of all these services are deployed on multiple servers

All these services are SOAP based and are implemented using JAX-WS in Java

**Publishing Web Service:**

As soon as any web service is deployed on any server, it sends the following details to the web registry

1. Its name
2. Its WSDL
3. IP address and port on which it is running

If any of the registries are not up at that time, then service will give the error saying all the registries are down and will not allow to deploy itself

## 4.2 Registry

We have designed a replicated registry. There are two replicas of the registry available which can easily be increased based on the requirement.

Below is how the registry works:

1. Whenever a new web service application is created, it will first send its details to any of the registry which is currently working. It will send its details to the first registry it is able to connect to.
2. Registry will maintain this information in its map and will also store this data locally in to a file
3. It will also send this newly received details to the all the other replicas of the registry, so that they are all in sync

4. If, by chance any of the replica is down at that time, registry stores this information in its buffer and keeps on pinging that replica every 5 seconds till that replica is up. When the replica is up, it will send the details to the replica and remove them from its buffer.
5. If a registry crashes it will lose all its data. But, since we are also storing this data in a file locally, registry will be able to recover this information and will be in sync with all other registries.

## 4.3 Load Balancer

The load balancer is also replicated, and it works in round robin fashion. Designing load balancer in the round robin fashion will reduce communication cost between the servers and the load balancer and load will be distributed equally as well assuming all the services takes almost same time to execute.

The load balancer in our system is connected to the client and it is necessary for it to always stay up to date

The working of load balancer is as follows:

1. As soon as the load balancer is up, it will update itself by contacting any of the replica of the registry fetching all the latest details of all the web services. This process is true for all the replicas of the load balancer.
2. Also, load balancer will keep its data updated by pinging the registries every 10 seconds and getting all the new data available with the registry.
3. Load balancer also maintains the state of each node.
4. Whenever the client contacts the load balancer, it will send the required WSDL to the client in round robin fashion. Since it maintains the state of each node, it knows which server was sent the last request and will send the WSDL to the client based on round robin method.

## 4.4 Client

Client works as follows

1. The client will display which services are available and user can choose any one of them based on the requirement.
2. Client connects to one of the load balancers.
3. Load balancer will send the WSDL of the required service to the client.
4. Once client receives the service WSDL, it will dynamically generate STUB for the service and will call it and fetch the required results.

## 5. Assumptions

1. One of the registries should always be up. If both registries go down, then the data available at load balancer will not be up to date. The load balancer will still have old data.
2. One of the load balancers should always be up since it is primary point of contact for client.
3. Not all the services are present at all the nodes
4. All the services with the same name will do same job.
5. Execution time of all the services are almost same.

## 6. Testing Strategy and Results

The arrangement for testing is as follows
2 load balancers
2 registries
4 nodes with multiple services

Below are the different testing strategies that we followed

1. **Basic testing**:
    - This is when all the registries and load balancers are up.
    - When the client sends the request, it will receive the required answer without any issue.
    - Also, addition of new service works successfully.

2. **When one of the registries is down**:
    - Both the load balancers are up and one of the registries is down.
    - User was still able to retrieve the data.
    - Tried to add a new service in this case, and it was added successfully at registry and well as load balancer
    - We then started the same registry that was up and updated with the same data that another registry had.

3. **When all the registries are down:**
    - Both the load balancers were running, and both the registries were stopped.
    - The client was still able to work successfully
    - Since all the registries are down, no new services could be added
    - We then stopped one of the services and requested the same from client. The client was able to fetch the WSDL of that service from the load balancer but was not able to perform the required operation since the service was down. This means that the load balancer does not have the updated data in this case since the registries are down.

4. **When one of the load balancers is down:**

- Both the registries were up and only one load balancer was up
- Client was able to connect to the required service
- New service could be added without any issue and client was able to access that
- We then started the same load balancer which was down. It was able to fetch all the data from the registry and had the same data as other load balancer.

5. **When both the load balancers are down:**
    - Both the registries were up, and both the load balancers were down.
    - It was a complete failure for the client in this case since load balancer is the point of contact for the client

6. **Load Testing:**

Below is the basic setup for load testing

1. There are four nodes, each node has 3 service.
2. We have added sleep time of 10 seconds for all the services so that each service takes 10 seconds to complete

Different test cases are as follows

1. Client is sending 1000 requests parallelly. Client creates a thread for each request and send it to load balancer. After every 100 threads, client will wait for 5 seconds and then again continue to create requests.
   Below graph represents load on each node. We can see from the diagram that every node is having almost same number of requests.

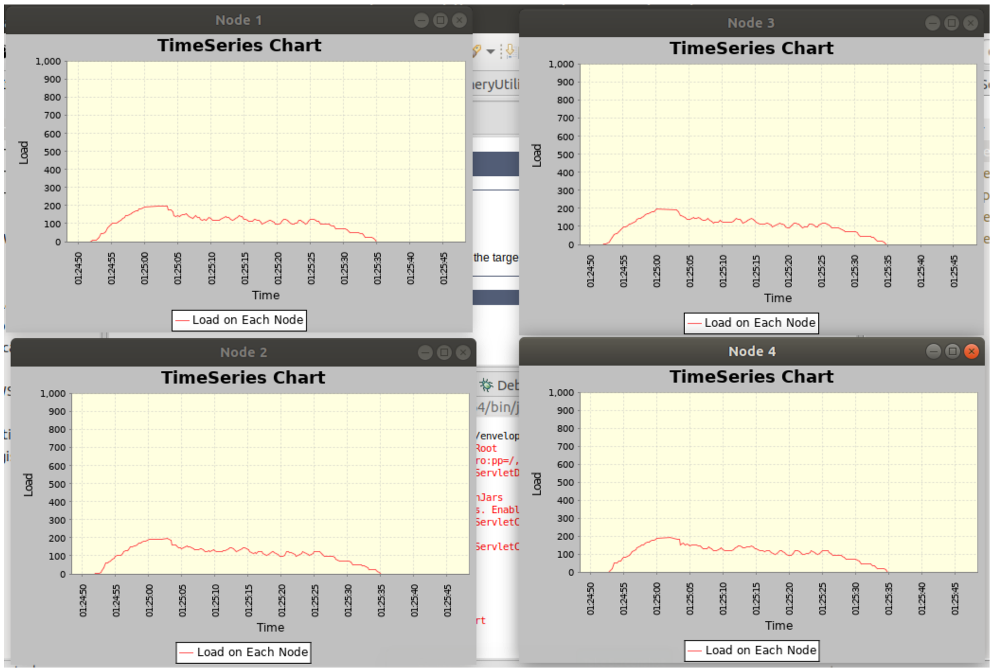

Fig.2 Load test

2. In second scenario, we again generated 1000 requests from the client, but this time the client sent all the threads at once and did not wait for some time after every 100 threads as we did in the previous case
   Our tomcat sever can handle only 200 requests at a time and what we observed in this scenario is that every node was running at its peak. 200 requests were assigned to every server after which it stopped taking any more requests until its load is reduced. But even in this case we could see that the load on all the nodes was nearly same

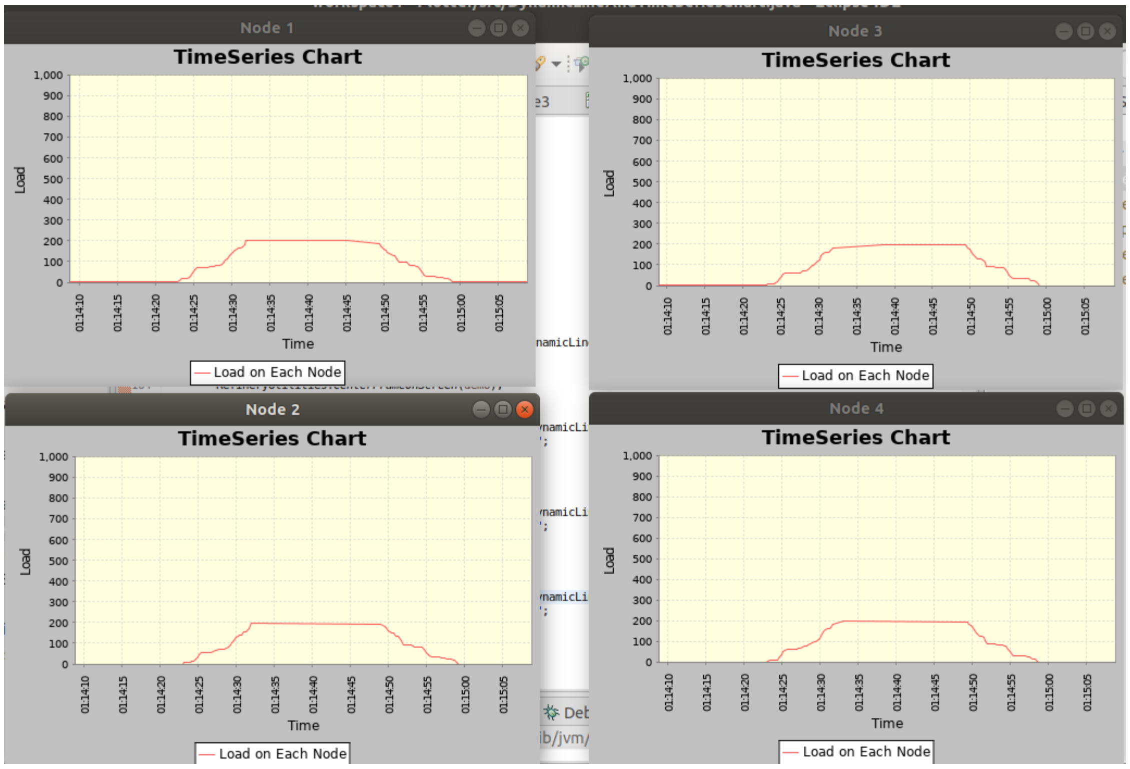

Fig 3. Load test

## 7. Features

1. Every service is a separate web application and it can be deployed separately on any number of different servers just by changing its ip address and port in the properties file.
2. We have used dynamic SOAP client instead of stubs, so that every entity will be independent of other entity.
3. Registry, load balancer and service are all implemented using SOAP web services

4. We have only two replicas of load balancer and registry in this project, but it can be increased to any number without changing any piece code. Only the properties file needs to be changed with the correct IP addresses and port numbers.
5. If any registry or load balancer crashes, whenever it starts working again, it will have the same data as all its other replicas.

## 8. Future Work

Currently, we are using centralized service discovery with replication which can be improved by using Chord or other distributed service discovery algorithms. Even though, this is going to make the framework less centralized, it will degrade the lookup time for the service. For load balancing, we can use a better centralized load balancing or distributed load balancing. The current design of framework only allows SOAP web services to register themselves which can be extended to allow REST web service.

## 9. Tools and Technologies Used

Different technologies used in this project are

1. Java 1.8
2. SOAP web services
3. Tomcat 9 web server
4. SOAP UI
5. Eclipse IDE

## References

http://speakingjava.blogspot.com/2013/10/dynamic-soap-service-client.html

https://www.membrane-soa.org/soa-model-doc/1.4/java-api/parse-wsdl-java-api.htm

https://docs.oracle.com/javaee/6/tutorial/doc/bnayl.html

https://kemptechnologies.com/load-balancer/load-balancing-algorithms-techniques/

https://www.javatpoint.com/multithreading-in-java

http://www.jfree.org/jfreechart/download.html