SECOND PROJECT REPORT

Akash Reddy Gurram | HR69338
Team Mate: Naveen Yamparala | RS09174

Changes in Design and Design Justifications:

- We have created SOAP web services in JAVA as well as in Python. But we are inclined towards using python for the rest of the project.
- The reason for not using JAVA is that we are facing a huge learning curve and with the given time constraint, it might be difficult to accomplish the tasks.
- In the preliminary report we planned on using the RESTful architecture but, we found that REST web services use only WSDL 2.0 for service description and we were unable to find support to generate WSDL 2.0 for REST.
- SOAP supports different versions of WSDL (1.x & 2.0). Hence we have used SOAP and WSDL 1.1 to create web services.
- We are not using Chord protocol anymore for service discovery because of its complexity and time constraint. We are first implementing a centralized service discovery and load balancing, and then replicate them across multiple machines to avoid single point of failures and other drawbacks of centralized systems.

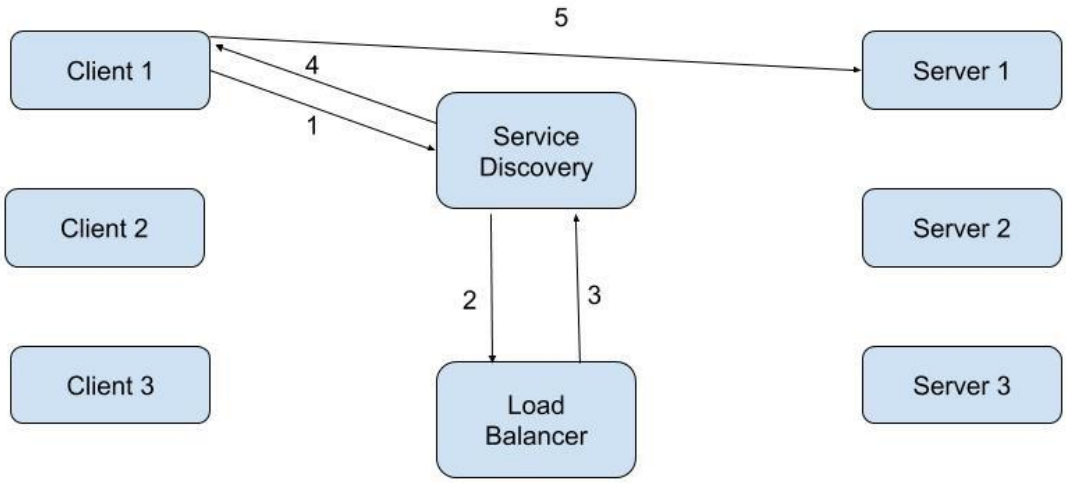
New System Specifications:

- We are using Python 2.7 to do the project.
- We have used *Flask* and *Spyne* frameworks. We used the built in server provided by flask called Flask Server to deploy web services. *Spyne* is used to make SOAP requests.
- *suds* library is used for the creation of client stub using WSDL provided by the SOAP service
- For service description we used WSDL 1.1 as professor agreed for using lower versions of WSDL for SOAP services.

New Assumptions:

- We are assuming that the service discovery and load balancer both know the addresses of the other servers that are running the web services.
- The clients are familiar with the address of the Service Discovery Server.
- The services provided by the web service are not rapidly changing.
- The service discovery server will contact all the web servers every few seconds for updating its service registry. This helps in reducing the latency of client requests.

Design Architecture:



1. Client sends a request to the Service Discovery Server. Service Discovery server and Load Balancer server have information about the services offered and the loads respectively.
2. Service Discovery Server filters the servers that provide a service and contacts the Load Balancer for the least loaded server among them.
3. Load balancer replies with a server that is best suitable for the job handling by considering statistics of the server like CPU/RAM usage, response time etc.
4. This is sent to the client from which the request is originated.
5. Client then contacts the Server that offers the service directly.

Progress to date:

- We have created web services for addition, multiplication and string reversal using SOAP protocol using flask and spyne frameworks of python.
- Using the WSDL file provided by the service, we have created a client stub to consume those services.
- We have replicated the servers and clients in our machine as well as other machines.
- We are in the process of implementing service discovery. We plan on doing this by repeatedly contacting the web servers for every few seconds and finding the services offered by them using the WSDL documents provided by the web servers.

Updated Timeline:

We finished the steps 1 and 2 on time as specified in the preliminary report. We are in the process of making the discovery. For service discovery we are making use of step 2. So they are being implemented in parallel. We are sticking to the preliminary timeline which is as follows

- Nov 12th – **Step 1**: First, we plan on creating a single client and server with the services specified in the design.

- Nov 15th – **Step 2**: We will replicate the services across different machines/servers.
- Nov 24th – **Step 3**: We will implement service discovery
- Dec 5th – **Step 4:** In the end, we will implement the load balancer.
- Dec 8th – **Step 5**: Test whether service discovery and load balancing are working as expected under high loads (many requests)
- Dec 10th – **Step 6**: We will try to make our sites participate in a block chain.

Scalability:

- Our design can accommodate scalability of both the servers and clients, because as per our assumption, if a new server joins the network, it will automatically get registered in the service registry as well as the load balancer.
- If time permits, when a new server is added to the network, we are planning to improve this by implementing automatic server registration and hence the service discovery by using techniques like flooding.
- Scalability of clients can be implemented since service discovery servers and load balancer accepts multiple requests.

Testing Strategy:

We did not explain testing strategy clearly in the preliminary report. Hence we are trying to explain it again.

- **Load Testing:** We can test the functionality of load balancer by giving fake loads for different services which can be given using the sleep( ) method and checking whether load balancing is being implemented as intended when there are a large number of client requests
- **Service Discovery Test:** We will disable few random services across random servers. Then we will check if the service discovery server is able to update its service registry accordingly. This can also be tested by debugging the service discovery server code.
- **Failure Testing:** This is mainly intended for service discovery and load balancer as they are centralized. We are going to intentionally take down these servers and check if the replicated servers are handling the requests or not.