

# Load Balancer (भार संतुलन)

Distributed Computing Mini Project

01.11.2018

---

Nitish Jadia | 18MCMT13

Alok Kumar Verma | 18MCMT09

Mtech

Computer Science

University of Hyderabad

## Overview

This application is a primitive implementation of Load Balancing concept. Loadbalancer (भार संतुलन) distributes the incoming jobs across a group of backend servers, known as a server farm. This increases throughput and efficiency of our server.

Multiple clients can send request to obtain the highest prime number possible at given upper limit. The input request is sent to server and server enquires about the load on farm and assigns the task to the sub-server with minimum load in the farm. The server then forwards the output from the farm to the client.

## Goals

1. Distribute the requests in the farm depending on the load on sub-servers.
2. Support for multiple client requests.

## Specifications

1. JAVA SE 8 or higher. (Farm is required to calculate CPU load)
2. Only Linux systems are supported for farm. (CPU load is calculated using system calls which are exclusive to Linux)
3. IP addresses of server and farms must be known.

## Configuration used during application development and testing:

### I. Development Environment:

- A. zsh shell
- B. Eclipse for JAVA
- C. Java 10.0.2 2018-07-17
- D. Java(TM) SE Runtime Environment 18.3 (build 10.0.2+13)

### II. Platform:

- A. Ubuntu 18.04, CentOS 7

## Workflow:

The application is made up of three modules:

1. Client module - ClientPrime
2. Server module - PrimeServer
3. Farm module - FarmServer

The modules must be run in the following order only:

### I. Start Farm

- Start required number of sub-servers on different machines using **farm.jar**
- Punch in the IP address of the machine on which the farm will be running and define a port number.

*Starting farm at two different machines or on different ports:*

```
crysis in projectX/FarmServer/src on ʘ master  
> java FarmMain
```

```
This farm's IP: 192.168.0.7  
Port: 9999
```

```
<-----FarmServer is running----->  
<-----CPU request server also running----->
```

```
□
```

```
crysis in projectX/FarmServer/src on ʘ master took 42s  
> java FarmMain
```

```
This farm's IP: 192.168.0.7  
Port: 5555
```

```
<-----FarmServer is running----->  
<-----CPU request server also running----->
```

```
□
```

## II. Start Server

- Start Server using `server.jar` and input the IP address of the machine running the server.
- Fill the farm details such as number of farms, corresponding IP address and ports.
- Specify the port on which the server will listen on for the client to connect.

*Initializing server:*

```
crysis in projectX/PrimeServer/src on ʘ master
> java ServerMain
This server's IP: 192.168.0.7

Farm settings->

Number of Farms: 2
IP of farm 1 : 192.168.0.7
Farm 1's Port: 8888
IP of farm 2 : 192.168.0.5
Farm 2's Port: 5555
<====Farm registration with server done====>

List of sub-servers active:
Farm Number = 1-> IP: 192.168.0.7, Port: 8888
Farm Number = 2-> IP: 192.168.0.5, Port: 5555

Listening Port for Clients: 7777
<=====Server is running for client=====>
```



### III. Start client

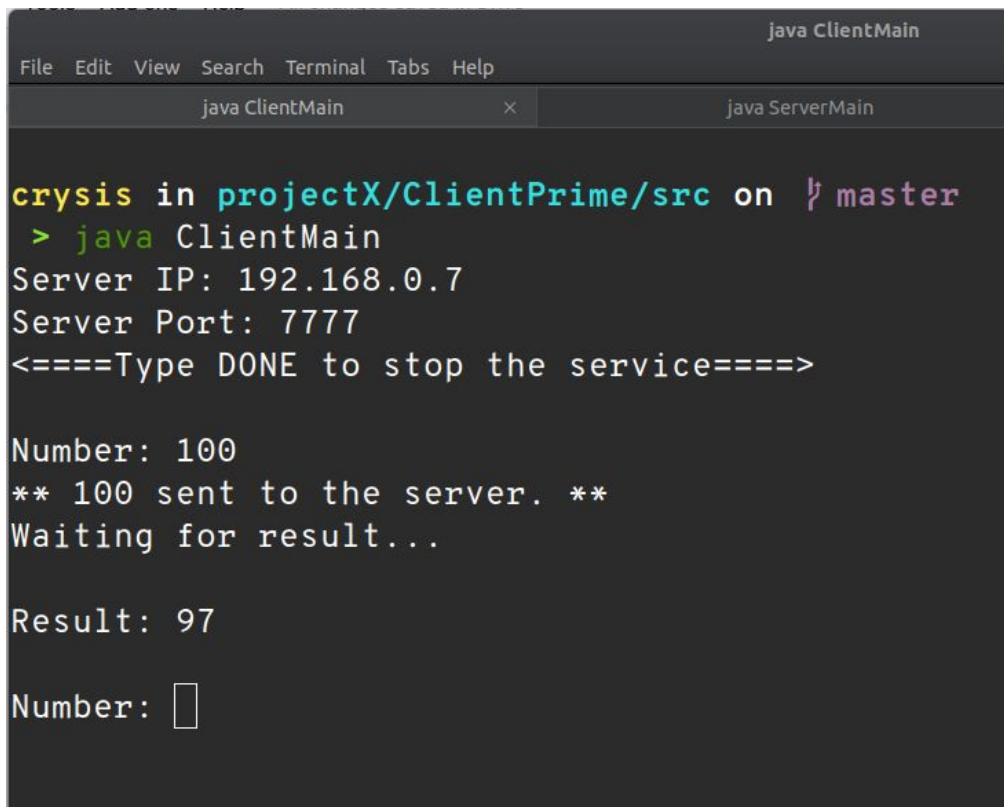
- Start the client and input the IP address and port of the server.
- Start giving input when asked and answer to the corresponding input will be provided.

*client started:*

```
crysis in projectX/ClientPrime/src on ʘ master
> java ClientMain
Server IP: 192.168.0.7
Server Port: 7777
<===Type DONE to stop the service===>

Number: 
```

*Input number 100 and we got our result:*



```
java ClientMain
File Edit View Search Terminal Tabs Help
java ClientMain x java ServerMain

crysis in projectX/ClientPrime/src on ʘ master
> java ClientMain
Server IP: 192.168.0.7
Server Port: 7777
<===Type DONE to stop the service===>

Number: 100
** 100 sent to the server. **
Waiting for result...

Result: 97

Number: 
```

The request 100 went to server and server forwarded the request to the farm having less CPU load.

Server:

```

java ServerMain
File Edit View Search Terminal Tabs Help
java ClientMain x java ServerMain x java FarmMain x

Farm settings->

Number of Farms: 1
IP of farm 1 : 192.168.0.7
Farm 1's Port: 8888
<====Farm registration with server done====>

List of sub-servers active:
Farm Number = 1-> IP: 192.168.0.7, Port: 8888

Listening Port for Clients: 7777
<=====Server is running for client=====>
<-----Connection accepted and new thread started ----->

Data received from client:100
Farm IP: 192.168.0.7, Port: 8888, CPU: 0.14
Farm allocation info:
Client IP: 192.168.0.7 got Farm at: 192.168.0.7, 8888
Data sent to client: 97

```

Farm:

```

crysis in projectX/FarmServer/src on master
> java FarmMain

This farm's IP: 192.168.0.7
Port: 8888

<-----FarmServer is running----->
<-----CPU request server also running----->

Message Received at farm is = cpu
CPUusages send by farm is =0.14

Message Received at farm is = 100
Prime number: 97

```



### Key point to observe in project:

1. This project support multiple client simultaneously because server is create thread for each request.
2. There is no restriction on number of farm .i.e. The number of farm can be increased to any number but before starting farm.
3. Target farm are dynamically chosen when the request come.

## Future Expansions:

1. Till now this project can generate prime number for the values in the range of integer , this can be expanded to BIG INTEGER class.
2. Till now farm can only generate prime number and we are planning to use this model to do some image processing or neural network task.
3. Till now we are using only cpu usage as a criteria for load distribution but we expand this to undertake some more criteria like proximity to client ,some more hardware configuration of farms etc.
4. One feature that need to be added is we want to take the farm list from a file so that we do not have to input the farm details every time.