

TASK4: ANALYSIS

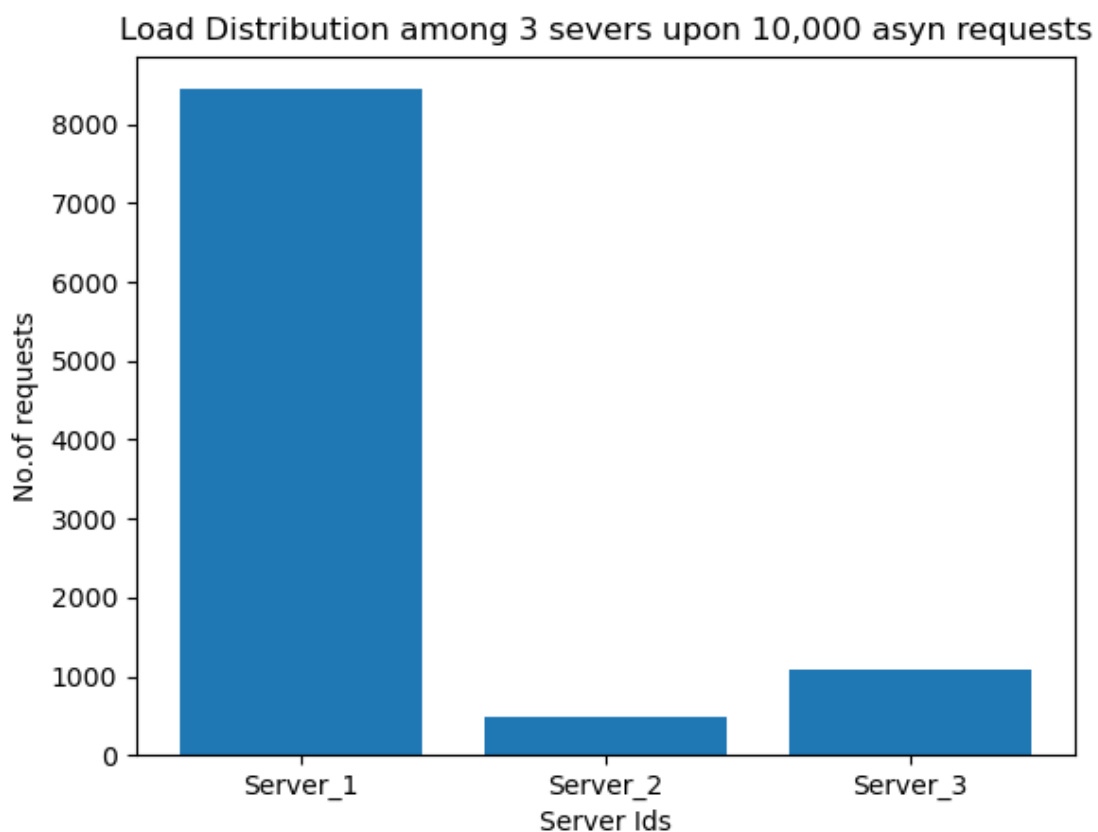
A-1) Launch 10000 async requests on $N = 3$ server containers and report the request count handled by each server instance in a bar chart. Explain your observations in the graph and your view on the performance.

Experiment Setup:

- Launched 10,000 asynchronous requests on a load balancer with $N=3$ server containers.
- Recorded the request count handled by each server instance.

Observations:

Load Distribution Bar Chart:



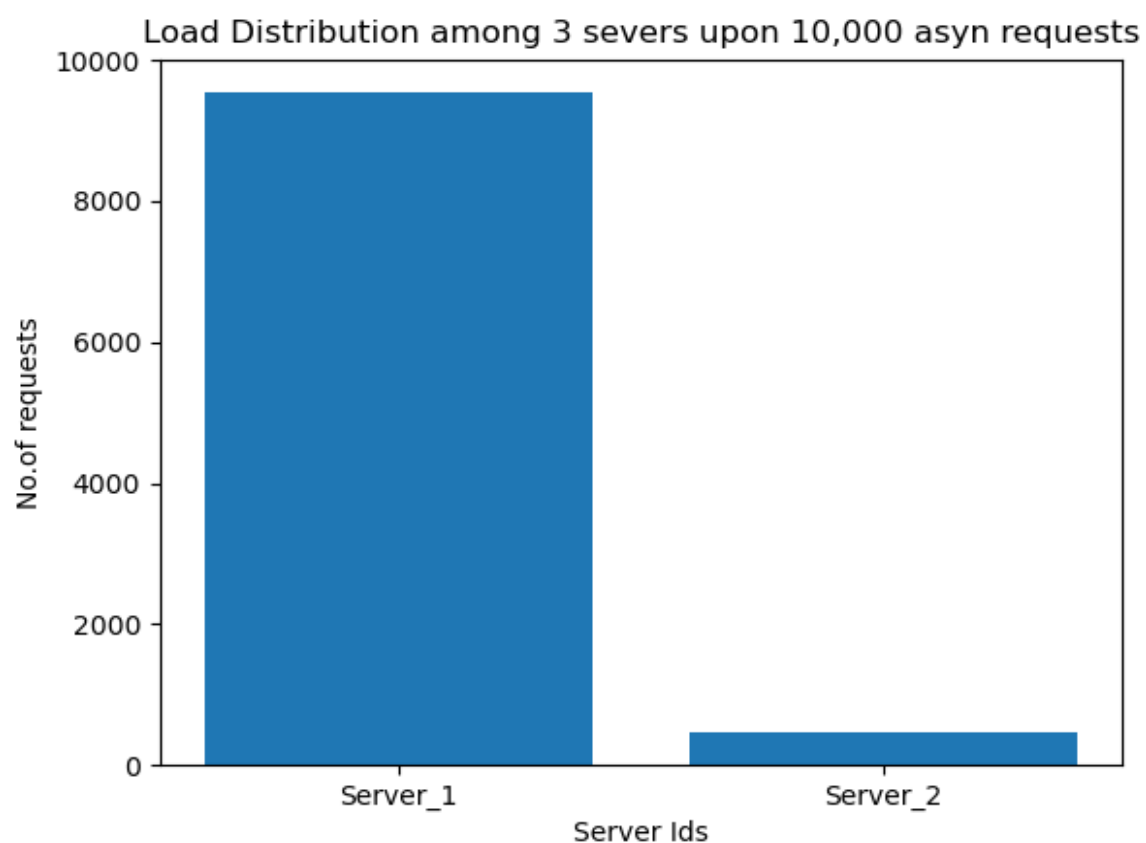
- **Server-1:** Handled 8445 requests
- **Server-2:** Handled 477 requests
- **Server-3:** Handled 1078 requests
- Server1 experiences a higher load compared to Server2 and Server3.

A-2) Next, increment N from 2 to 6 and launch 10000 requests on each such increment. Report the average load of the servers at each run in a line chart. Explain your observations in the graph and your view on the scalability of the load balancer implementation.

For N=2,

Load Distribution Bar Chart:

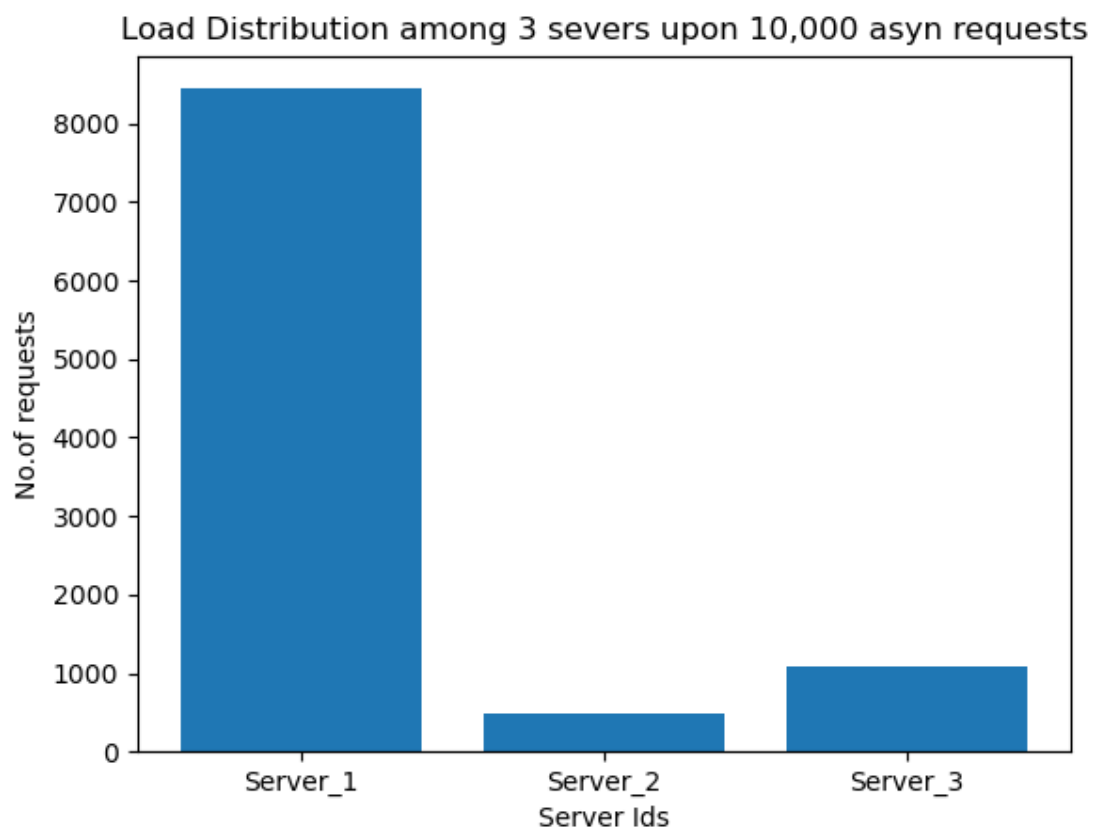
- **Server-1:** Handled 9537 requests
- **Server-2:** Handled 463 requests
- Server1 experiences a higher load compared to Server2.



For N=3,

Load Distribution Bar Chart:

- **Server-1:** Handled 8445 requests
- **Server-2:** Handled 477 requests
- **Server-3:** Handled 1078 requests
- Server1 experiences a higher load compared to Server2 and Server3.

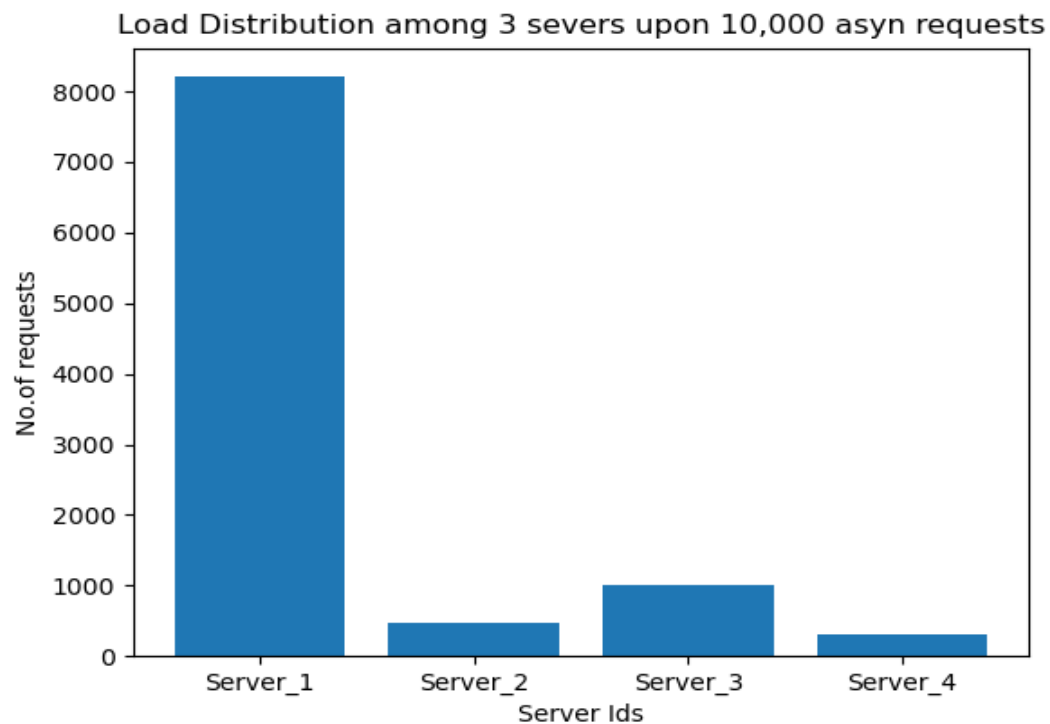


For N=4,

Load Distribution Bar Chart:

- **Server-1:** Handled 8207 requests
- **Server-2:** Handled 477 requests
- **Server-3:** Handled 1013 requests
- **Server-4:** Handled 303 requests

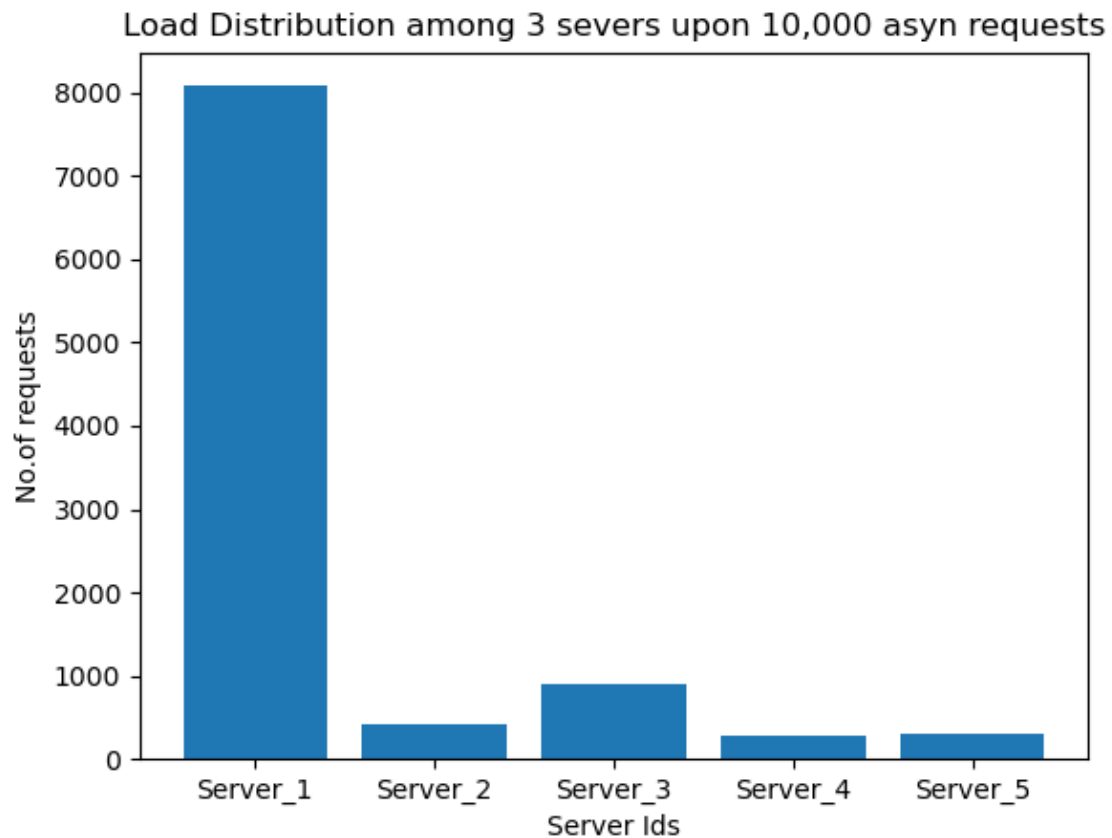
- Server1 experiences a higher load compared to Server2, Server3, and Server4, while Server2, Server3, and Server4 handle almost the same number of requests.



For N=5,

Load Distribution Bar Chart:

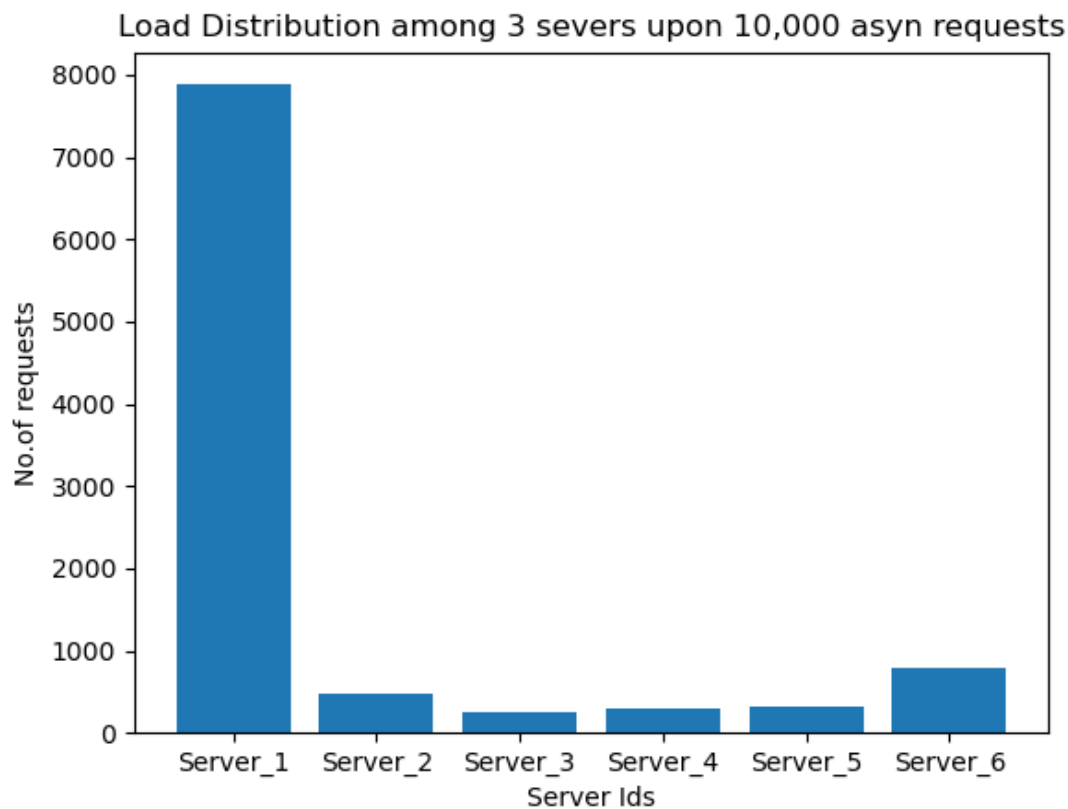
- **Server-1:** Handled 8081 requests
- **Server-2:** Handled 426 requests
- **Server-3:** Handled 907 requests
- **Server-4:** Handled 290 requests
- **Server-5:** Handled 296 requests
- Server1 experiences a higher load compared to Server2, Server3, and Server4, while Server2, Server4 and Server5 handle almost the same number of requests and bit lesser than Server3.



For N=6,

Load Distribution Bar Chart:

- **Server-1:** Handled 7881 requests
- **Server-2:** Handled 484 requests
- **Server-3:** Handled 248 requests
- **Server-4:** Handled 294 requests
- **Server-5:** Handled 311 requests
- **Server-6:** Handled 782 requests
- Server1 experiences a higher load compared to remaining servers, while Server2, Server3, Server4 and Server5 handle almost the same number of requests and bit lesser than Server6.



A-3) Test all endpoints of the load balancer and show that in case of server failure, the load balancer spawns a new instance quickly to handle the load.

1) Endpoint (/add, method=POST):

This endpoint adds new server instances to the load balancer, scaling up to accommodate the growing number of clients in the system. The endpoint expects a JSON payload specifying the number of new instances and their preferred hostnames in a list.

For example, to add 3 new servers named 's1', 's2', and 's3', you can make a POST request using Postman with the following URL:

`http://127.0.0.1:5000/add?n=3&replicas=['s1', 's2', 's3']`

POST http://127.0.0.1:5000/add?n=3&replicas=["s1","s2","s3"] Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

<input checked="" type="checkbox"/> Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> n	3			
<input checked="" type="checkbox"/> replicas	["s1","s2","s3"]			
Key	Value	Description		

```

1 {
2   "message": {
3     "N": 6,
4     "replicas": [
5       "s3",
6       "s1",
7       "s2",
8       "server3",
9       "server2",
10      "server1"
11    ]
12  },
13  "status": "Successful"

```

Hence 3 new servers are added.

If n value (number of servers to be added) is more than the no.of hostnames then we might get error:

HTTP http://127.0.0.1:5000/add?n=1&replicas=["s1","s2","s3"]

POST http://127.0.0.1:5000/add?n=1&replicas=["s1","s2","s3"]

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

<input checked="" type="checkbox"/> Key	Value
<input checked="" type="checkbox"/> n	1
<input checked="" type="checkbox"/> replicas	["s1","s2","s3"]
Key	Value

body Cookies Headers (5) Test Results Status: 400

Pretty Raw Preview Visualize JSON

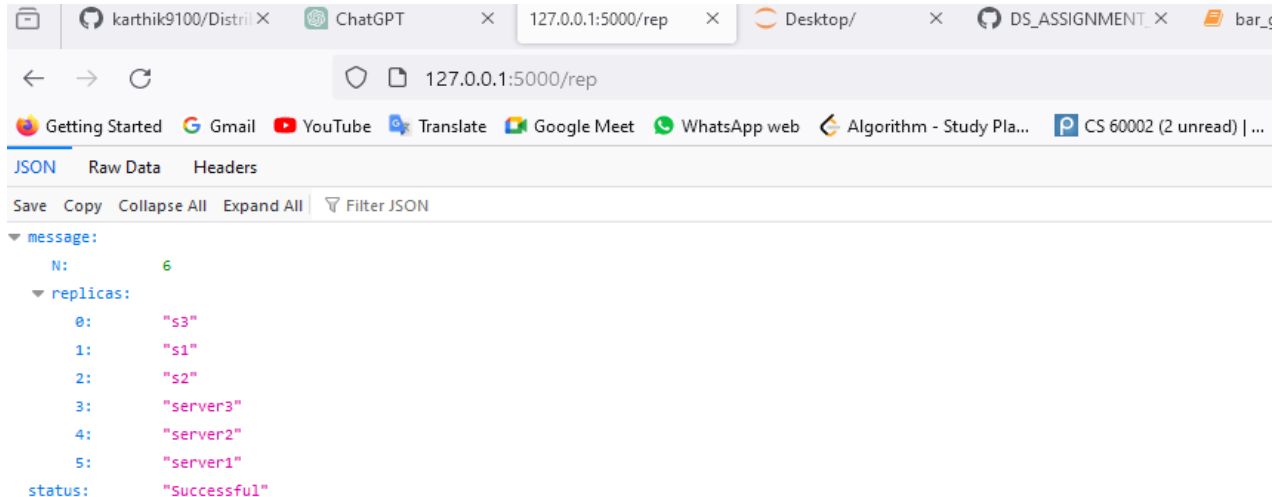
```

1 {
2   "message": "<Error> Length of hostname list is more than newly added instances",
3   "status": "Faliure"
4 }

```

2) Endpoint (/rep, method=GET):

This endpoint only returns the status of the replicas managed by the load balancer.

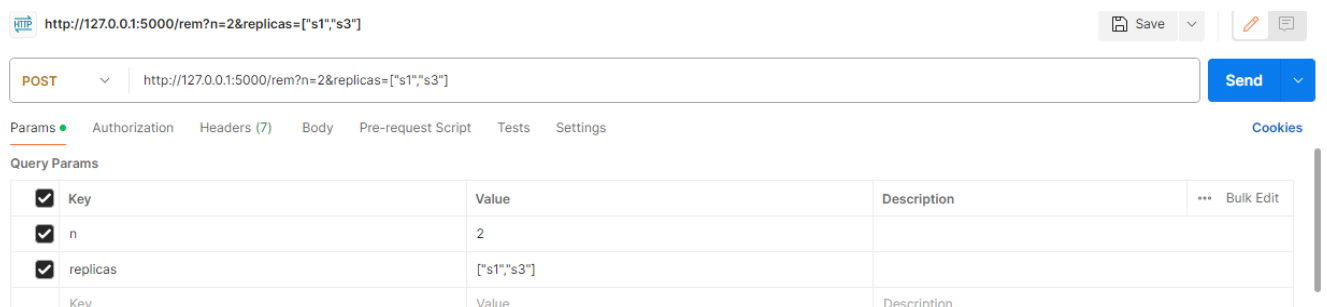


3) Endpoint (/rm, method=DELETE):

This endpoint removes server instances in the load balancer to scale down with decreasing client or system maintenance. The endpoint expects a JSON payload that mentions the number of instances to be removed and their preferred hostnames.

For example, to remove servers let's say 's1' and 's3', i made a POST request using Postman with the following URL:

[http://127.0.0.1:5000/rem?n=3&replicas=\['s1', 's3'\]](http://127.0.0.1:5000/rem?n=3&replicas=['s1', 's3'])

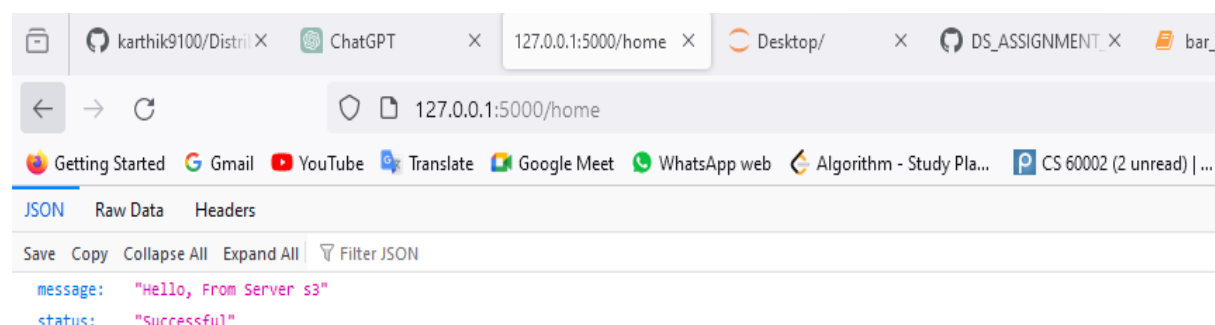


Here, we can see that S1 and S3 got removed.

```
Pretty Raw Preview Visualize JSON ↕
1  {
2    "message": {
3      "N": 4,
4      "replicas": [
5        "s2",
6        "server3",
7        "server2",
8        "server1"
9      ]
10   },
11   "status": "Successful"
12 }
```

4) Endpoint (/<path>, method=GET):

Request in this endpoint gets routed to a server replica as scheduled by the consistent hashing algorithm of the load balancer.



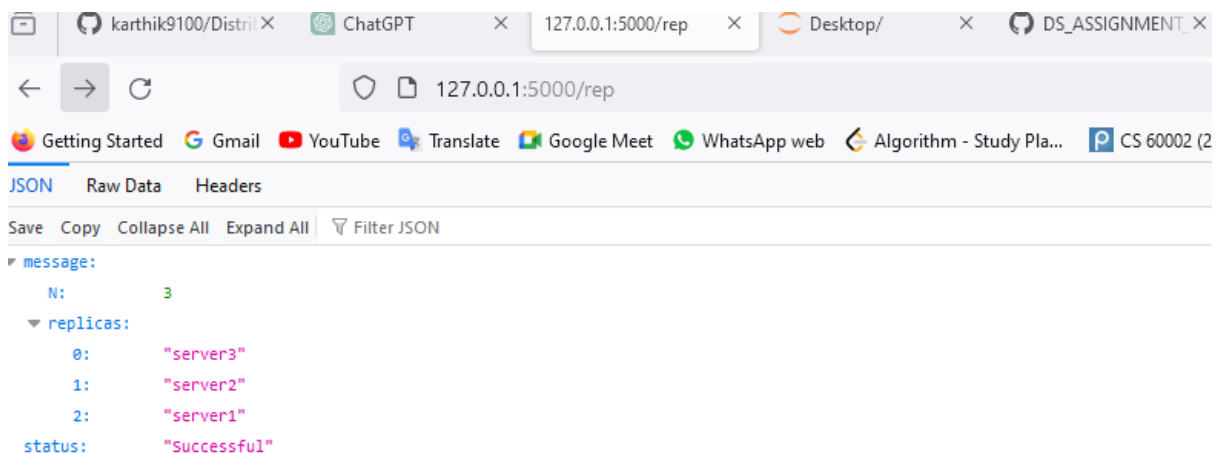
The screenshot shows a web browser window with multiple tabs. The active tab is titled '127.0.0.1:5000/home'. The address bar shows the URL '127.0.0.1:5000/home'. Below the address bar, there are several bookmarks including 'Getting Started', 'Gmail', 'YouTube', 'Translate', 'Google Meet', 'WhatsApp web', 'Algorithm - Study Pla...', and 'CS 60002 (2 unread) | ...'. The browser's developer tools are open, showing the 'JSON' tab. The JSON response is as follows:

```
message: "Hello, From Server s3"
status: "Successful"
```

SPAWNING:

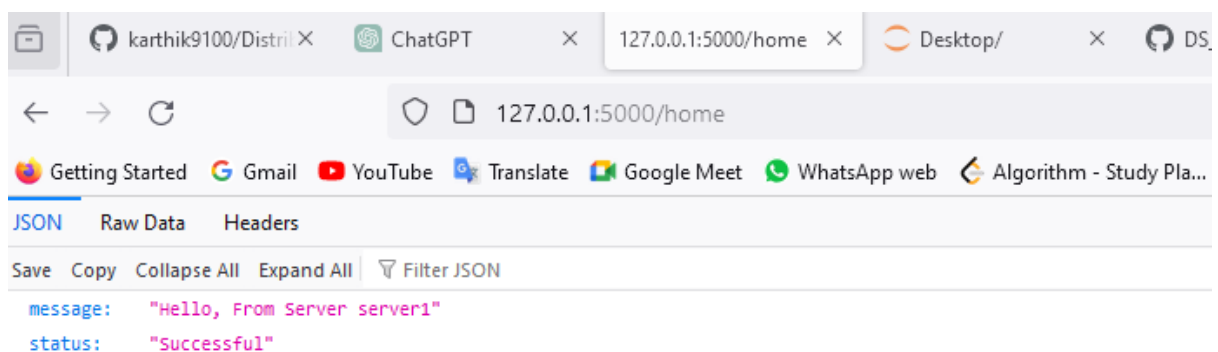
In case of server failure, the load balancer spawns a new instance quickly to handle the load.

/rep: Currently 3 servers are there.



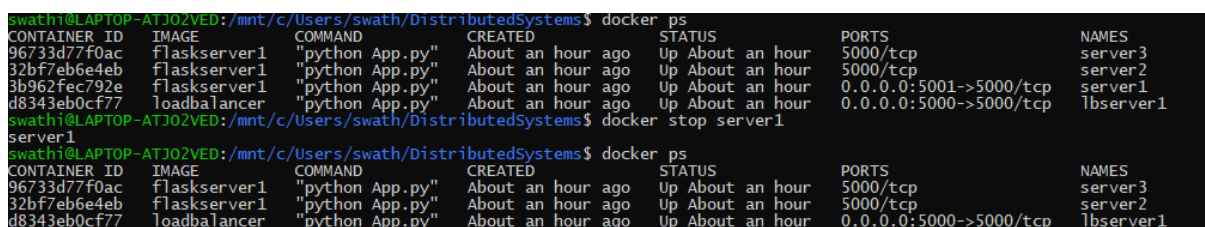
```
{
  "message": {
    "N": 3,
    "replicas": {
      "0": "server3",
      "1": "server2",
      "2": "server1"
    },
    "status": "Successful"
  }
}
```

/home: I placed one request, and it was assigned to Server1.



```
{
  "message": "Hello, From Server server1",
  "status": "Successful"
}
```

Now, I am manually stopping Server1 and putting the request again.

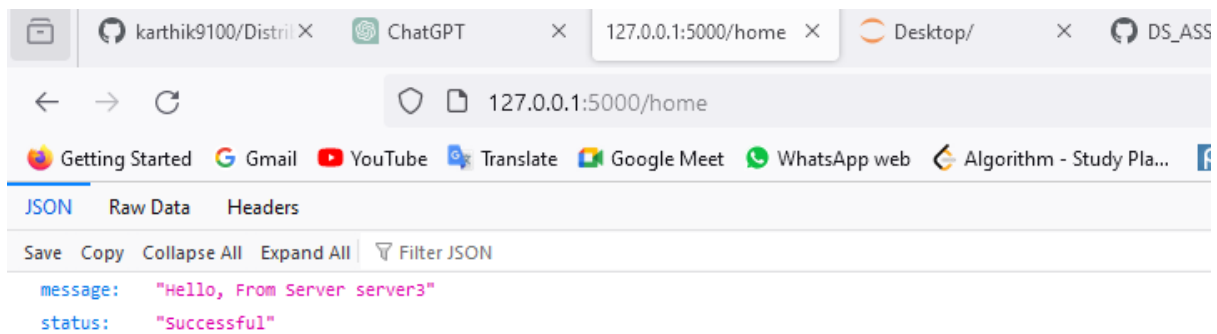


```
swathi@LAPTOP-ATJ02VED:/mnt/c/Users/swath/DistributedSystems$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
96733d77f0ac   flaskserver1   "python App.py"         About an hour ago    Up About an hour    5000/tcp                           server3
32b7eb6e4eb    flaskserver1   "python App.py"         About an hour ago    Up About an hour    5000/tcp                           server2
3b962fec792e    flaskserver1   "python App.py"         About an hour ago    Up About an hour    0.0.0.0:5001->5000/tcp              server1
d8343eb0cf77    loadbalancer   "python App.py"         About an hour ago    Up About an hour    0.0.0.0:5000->5000/tcp              lbserver1

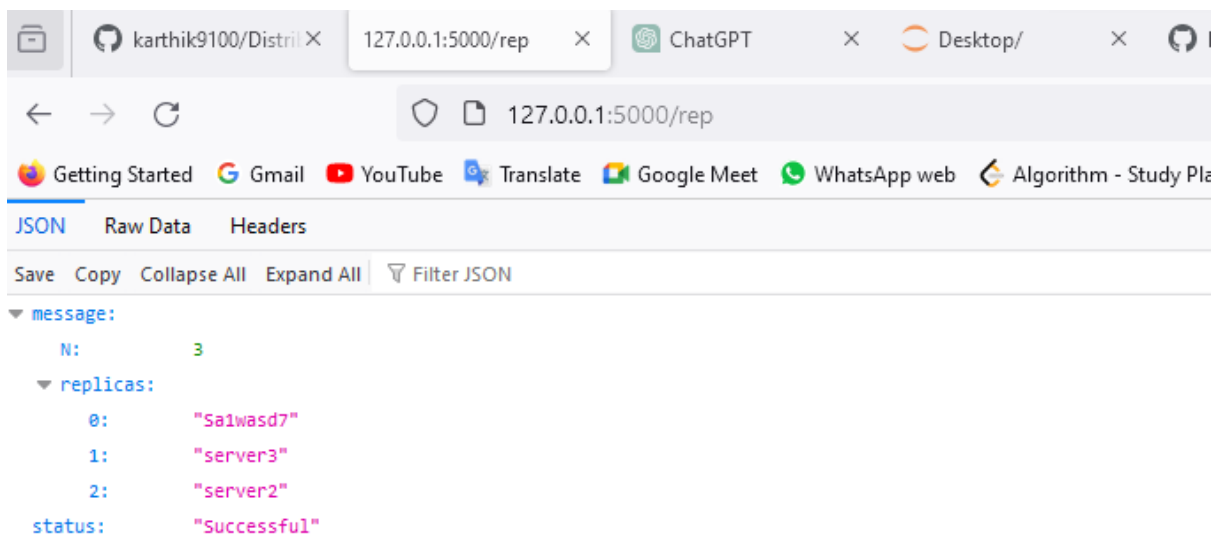
swathi@LAPTOP-ATJ02VED:/mnt/c/Users/swath/DistributedSystems$ docker stop server1
server1

swathi@LAPTOP-ATJ02VED:/mnt/c/Users/swath/DistributedSystems$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
96733d77f0ac   flaskserver1   "python App.py"         About an hour ago    Up About an hour    5000/tcp                           server3
32b7eb6e4eb    flaskserver1   "python App.py"         About an hour ago    Up About an hour    5000/tcp                           server2
d8343eb0cf77    loadbalancer   "python App.py"         About an hour ago    Up About an hour    0.0.0.0:5000->5000/tcp              lbserver1
```

Now, we can see that newly requested request is mapped to different server.



And also it spawns the new server.



Server “Sa1wasd7” was added newly in place of server1.

A4) Modify the hash functions $H(i)$, $\Phi(i, j)$ and report the observations from (A1) and (A2).

New hash functions are:

$$H(i) = i^2 + 2^i + c,$$

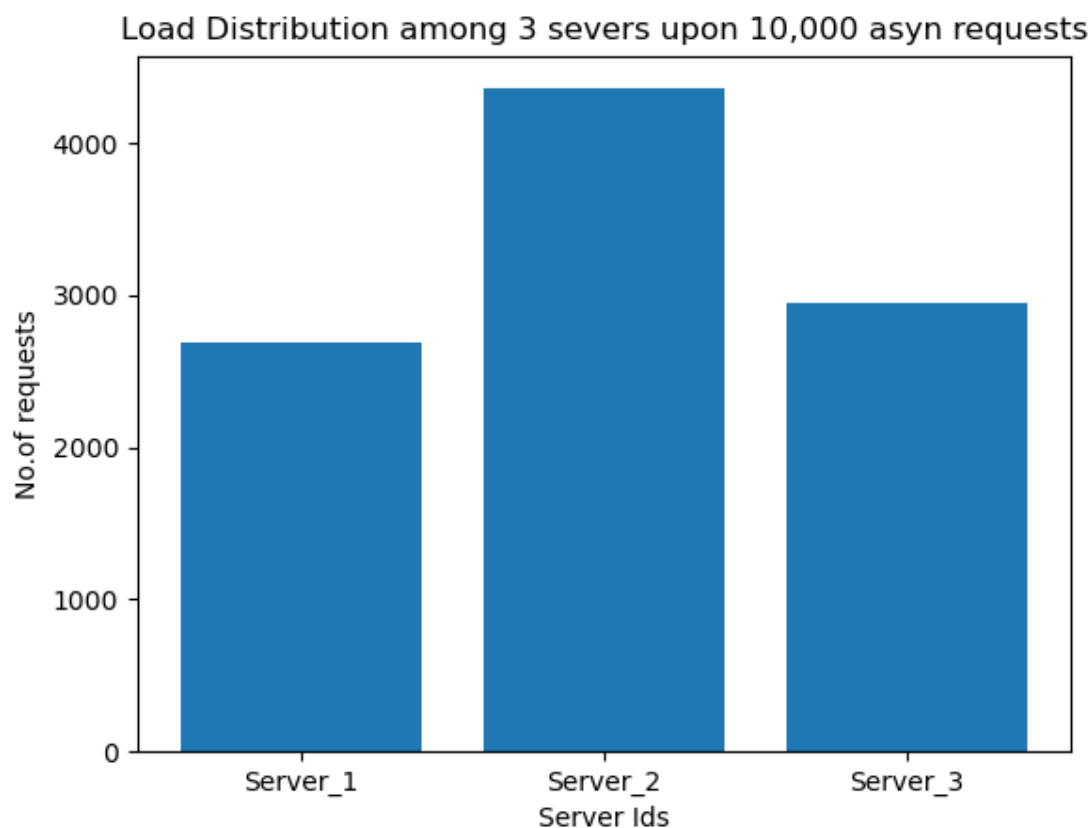
$$\Phi(i, j) = i^2 + j^2 + 2^j + k,$$

Where c, k are constants and randomly taken from range 1 to 1000.

A1) Launched 10,000 asynchronous requests on a load balancer with N=3 server containers. Recorded the request count handled by each server instance.

Observations:

Load Distribution Bar Chart:



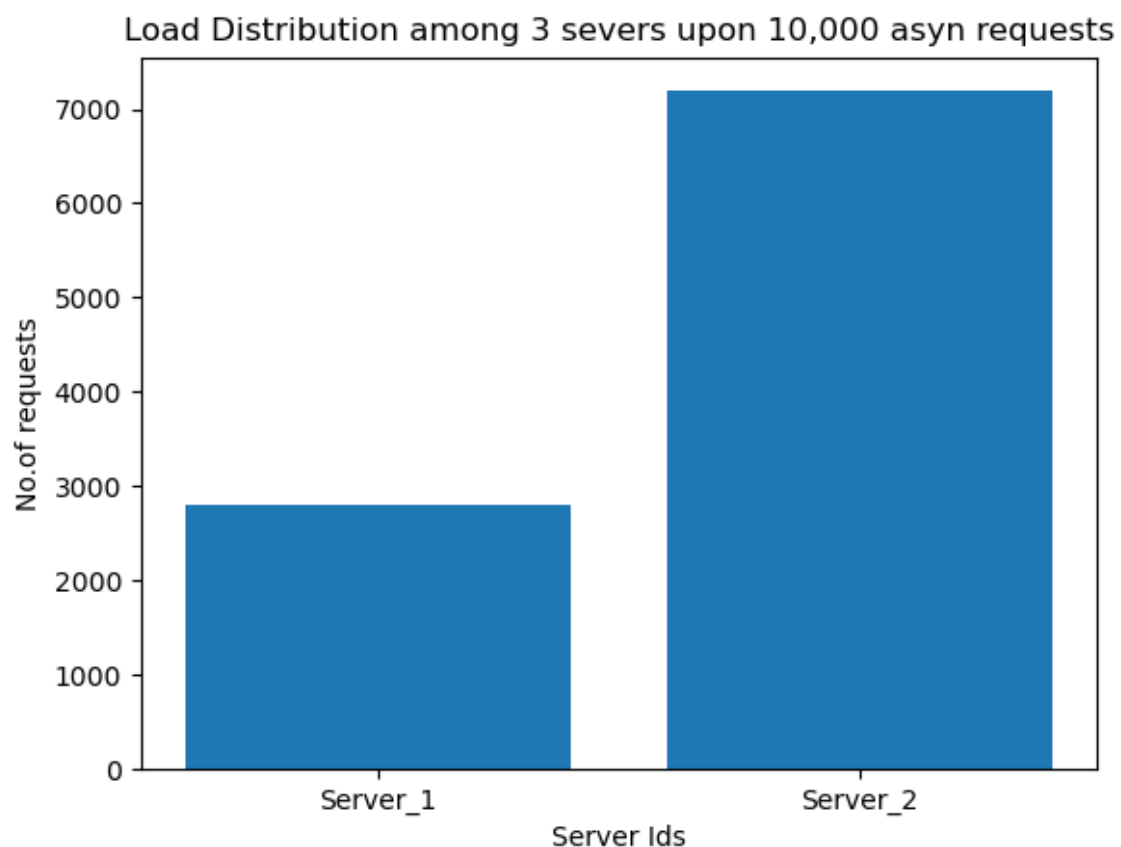
- **Server-1:** Handled 2693 requests
- **Server-2:** Handled 4357 requests
- **Server-3:** Handled 2950 requests
- Server2 experiences a bit higher load compared to Server1 and Server3.
- Server1 and Server3 loads are almost similar.
- One more observation is that, Load is better distributed among 3 servers after changing the hash function. (In comparison to older bar graph).

A-2) Next, increment N from 2 to 6 and launch 10000 requests on each such increment. Report the average load of the servers at each run in a line chart. Explain your observations in the graph and your view on the scalability of the load balancer implementation.

For N=2,

Load Distribution Bar Chart:

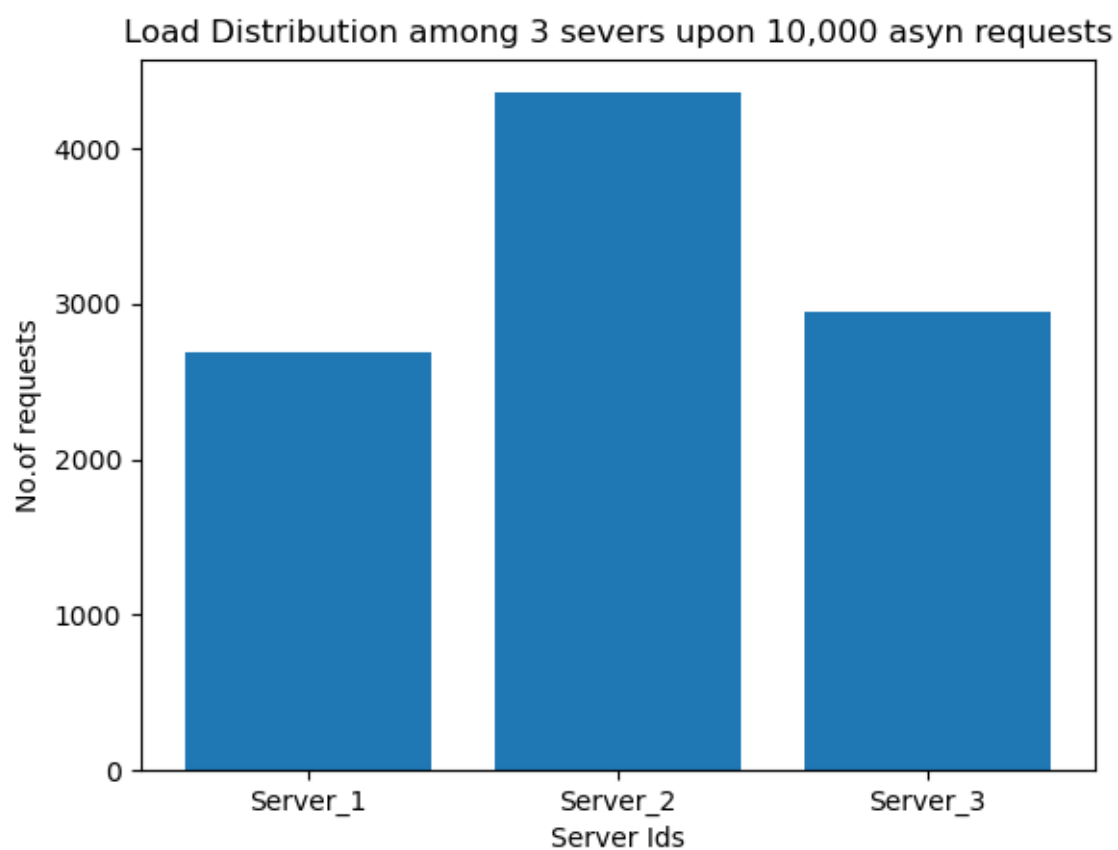
- **Server-1:** Handled 2810 requests
- **Server-2:** Handled 7190 requests
- Server2 experiences a higher load compared to Server1.



For N=3,

Load Distribution Bar Chart:

- **Server-1:** Handled 2693 requests
- **Server-2:** Handled 4357 requests
- **Server-3:** Handled 2950 requests
- Server2 experiences a bit higher load compared to Server1 and Server3.
- Server1 and Server3 loads are almost similar.
- But it is well distributed.

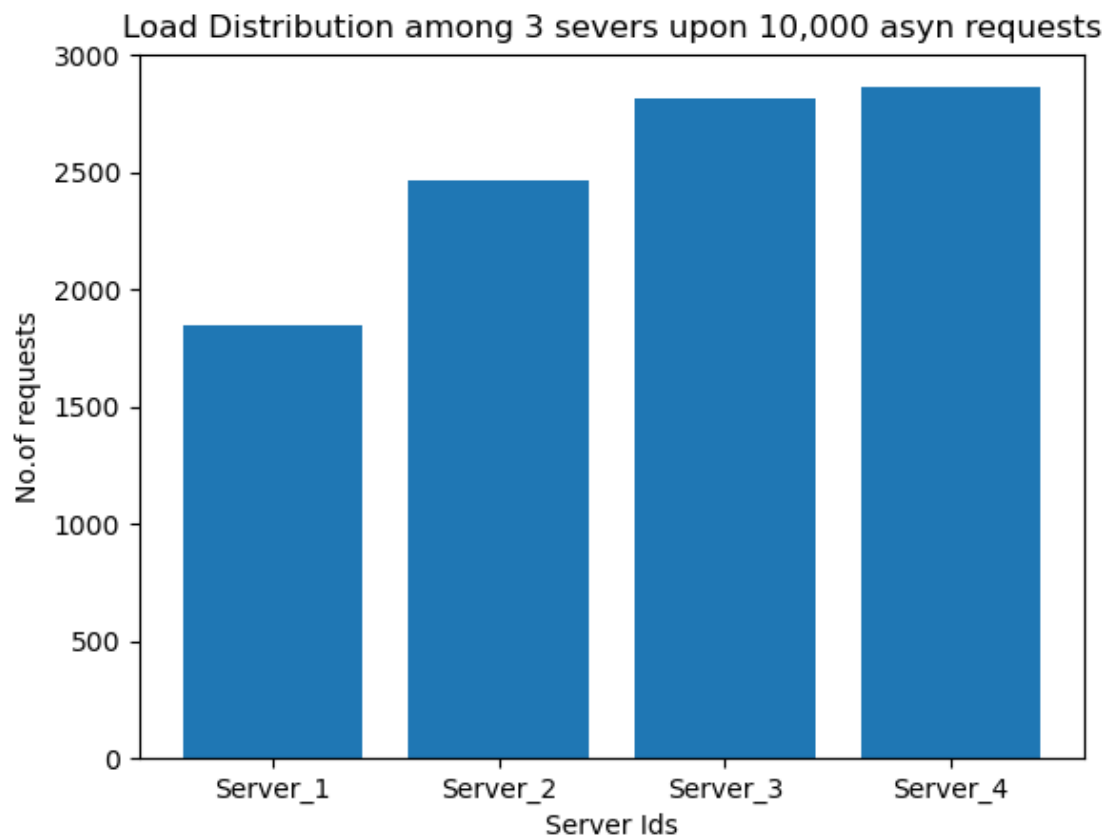


For N=4,

Load Distribution Bar Chart:

- **Server-1:** Handled 1849 requests
- **Server-2:** Handled 2468 requests
- **Server-3:** Handled 2819 requests

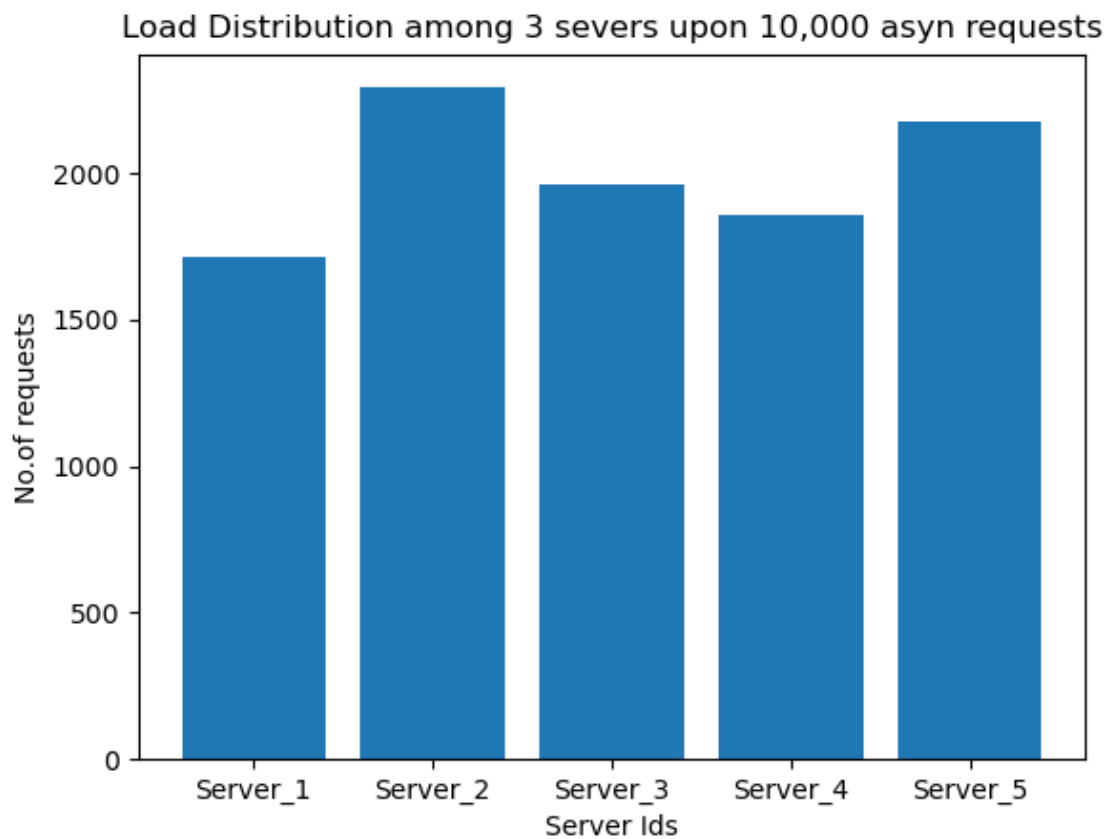
- **Server-4:** Handled 2864 requests
- Here all servers has the same load.



For N=5,

Load Distribution Bar Chart:

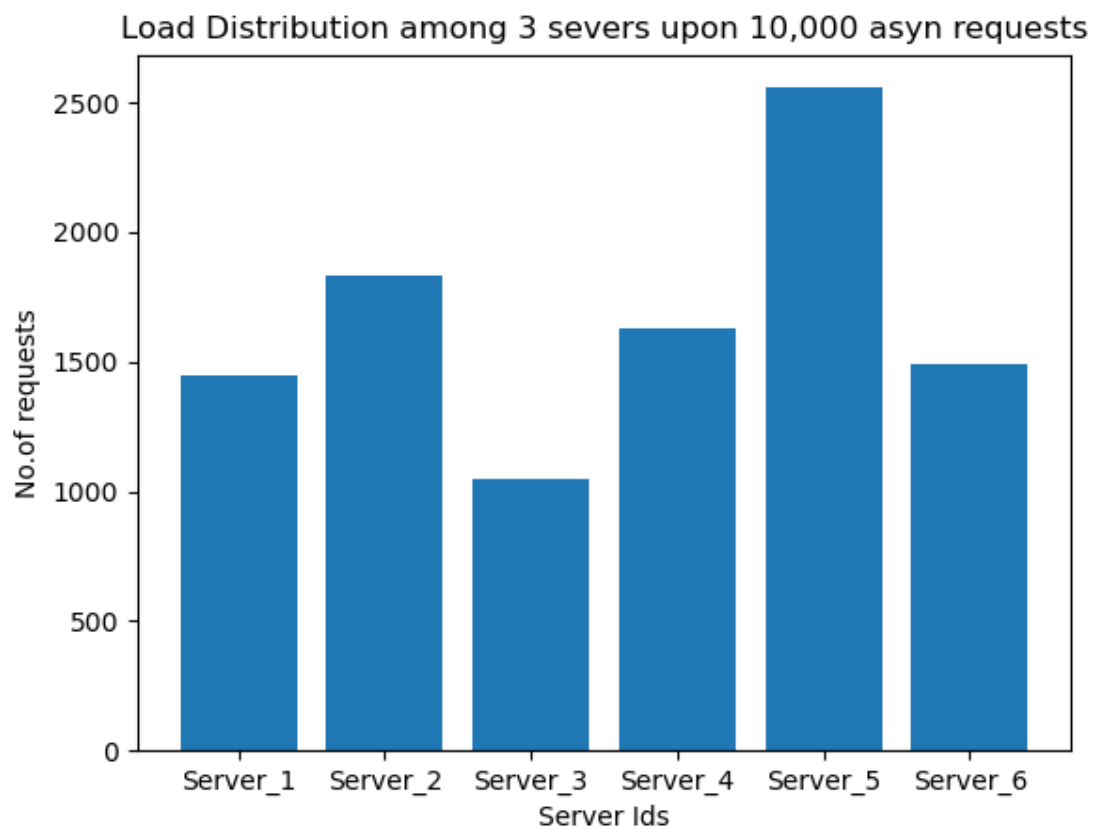
- **Server-1:** Handled 1715 requests
- **Server-2:** Handled 2292 requests
- **Server-3:** Handled 1960 requests
- **Server-4:** Handled 1857 requests
- **Server-5:** Handled 2176 requests
- Here all servers has the same load. And almost equally distributed, that means load has well distributed among all servers.



For N=6,

Load Distribution Bar Chart:

- **Server-1:** Handled 1447 requests
- **Server-2:** Handled 1833 requests
- **Server-3:** Handled 1048 requests
- **Server-4:** Handled 1626 requests
- **Server-5:** Handled 2558 requests
- **Server-6:** Handled 1488 requests
- Server5 experiences a bit higher load compared to other servers. Upon Remaining servers load is almost equally distributed.



⇒ Finally, through observation, what I conclude is that the changed hash function works well. This is evident as, upon using those hash values, the load is evenly distributed among all servers. (we can witnessed through above bar graphs).