

**CSE-5306**  
**DISTRIBUTED SYSTEMS**

**PROJECT – 1**

Date: 09-26-2023

**Name:** Mohammad Shahedur Rahman  
**ID:** 1002157980

**Name:** Meh Zad Hossain Setu  
**ID:** 1002138330

I have neither given nor received unauthorized assistance on this work.

Signed: Mohammad Shahedur Rahman

Date: 09-26-2023

Signed: Meh Zad Hossain Setu: [OBJ]

Date: 09-26-2023

## Table of Contents

Introduction .....	3
Assignment-0.....	4
Assignment-1.....	6
Assignment-2.....	7
Comparison of the performance between TCP and UDP .....	7
Assignment-3.....	10
Assignment-4.....	11
Comparison Between Synchronous RPC and Asynchronous RPC: .....	11
Conclusion.....	13
Additional Considerations .....	13

## **Introduction**

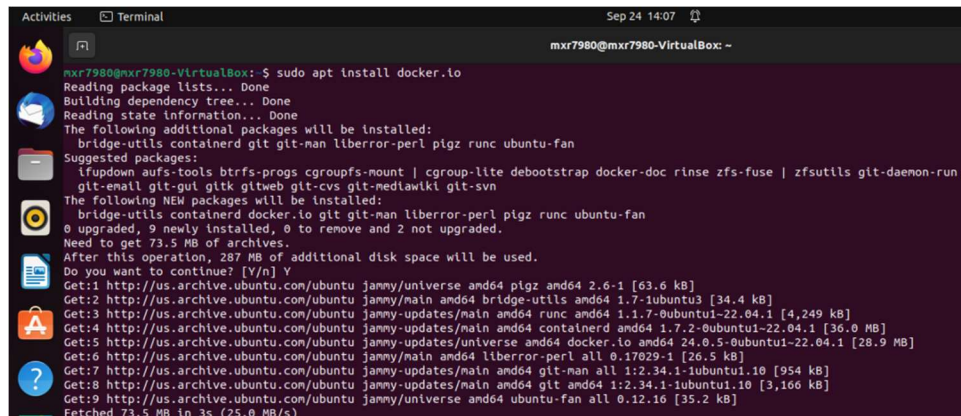
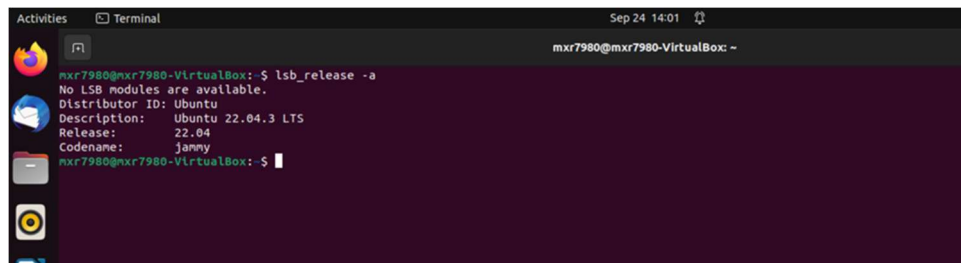
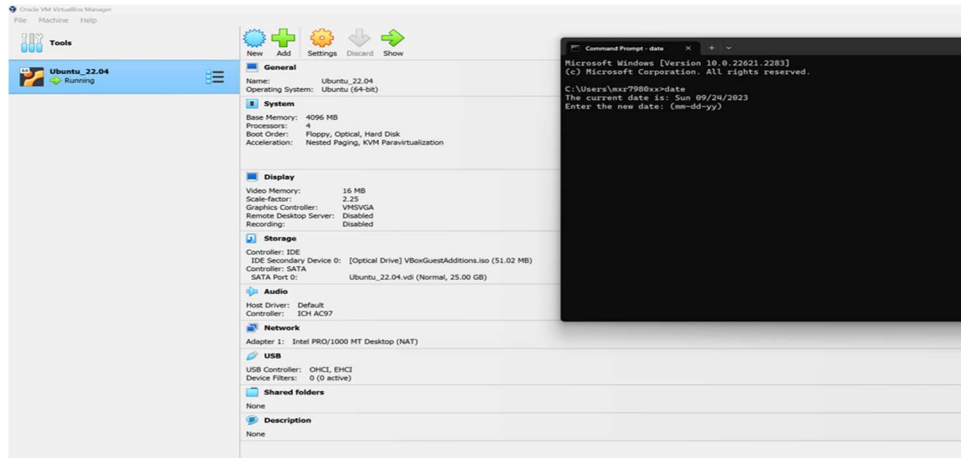
In this programming project, we have practiced programming client-server communications and remote procedure calls (RPCs). In the client-server communication assignment, we have implemented a simple web server using various communication schemes, such as UDP and TCP. In the RPC-server assignment(s), you will implement two types of RPC schemes, i.e., synchronous and asynchronous RPC, and compare their performance by varying the amount of computation on the RPC server.

We have used below mentioned tools and programming language for this project.

1. Oracle VM Box 7.4.10
2. Ubuntu 22.04.3 LTS
3. Python 3.11
4. Other relevant packages.

## Assignment-0

In our experimental setup, we successfully established a virtual machine running Ubuntu. We then proceeded to create two docker containers within this VM.



Img 1: Installing Docker

```
Activities Terminal Sep 24 14:33 root@efbeb46d9815: /

nrx7980@nrx7980-VirtualBox: $ sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
3a01b74063c6   fedora    "/bin/bash"   4 minutes ago   Up 4 minutes           client3
49425331ca7c   fedora    "/bin/bash"   4 minutes ago   Up 4 minutes           single_thread_server3
efbeb46d9815   ubuntu   "/bin/bash"   12 minutes ago   Up 12 minutes          client1
9f7e24feadf5   fedora    "/bin/bash"   13 minutes ago   Up 13 minutes          single_thread_server1

nrx7980@nrx7980-VirtualBox: $ sudo docker inspect single_thread_server1 | grep IPAddress
"SecondaryIPAddresses": null,
"IPAddress": "",
"IPAddress": "192.168.1.0",

nrx7980@nrx7980-VirtualBox: $ sudo docker inspect client1 | grep IPAddress
"SecondaryIPAddresses": null,
"IPAddress": "",
"IPAddress": "192.168.1.2",

nrx7980@nrx7980-VirtualBox: $ sudo docker exec -it client1 bash
root@efbeb46d9815:/# ping 192.168.1.0
PING 192.168.1.0 (192.168.1.0) 56(84) bytes of data.
64 bytes from 192.168.1.0: icmp_seq=1 ttl=64 time=0.053 ms
64 bytes from 192.168.1.0: icmp_seq=2 ttl=64 time=0.054 ms
64 bytes from 192.168.1.0: icmp_seq=3 ttl=64 time=0.051 ms
64 bytes from 192.168.1.0: icmp_seq=4 ttl=64 time=0.053 ms
64 bytes from 192.168.1.0: icmp_seq=5 ttl=64 time=0.055 ms
64 bytes from 192.168.1.0: icmp_seq=6 ttl=64 time=0.053 ms
64 bytes from 192.168.1.0: icmp_seq=7 ttl=64 time=0.075 ms
^C
--- 192.168.1.0 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 612ms
rtt min/avg/max/mdev = 0.051/0.056/0.075/0.007 ms
root@efbeb46d9815:/#
```

Img2: Client and single\_Thread\_server is communicating.

Using Docker containers and overlay networking showcased their adaptability and scalability, laying a strong groundwork for upcoming assignments focused on enhancing client-server applications.

## **Assignment-1**

Creating a basic single-threaded web server that serves small-sized images over both UDP AND TCP is a practical project:

We implemented a basic single-threaded web server capable of delivering small- sized images in response to client requests, utilizing both UDP and TCP protocols. The servers were designed to rename the images uniquely based on the client's file descriptor, providing a personalized naming convention.

**TCP:** We have introduced two critical features to enhance functionality and user experience. Firstly, clients can now utilize the "LIST" command to view a list of available files on the server within the "server\_data" folder, followed by the "RENAME <old\_filename> <new\_filename>" command to rename files, providing greater control and customization over server resources. Secondly, we've implemented a "DISCONNECT" command, allowing clients to gracefully close their socket connections, freeing up server resources for other queued clients. These additions augment the application's versatility, improve user interaction, and pave the way for more efficient resource management in a single-threaded server environment, underscoring our commitment to iterative development and creating a robust client-server system.

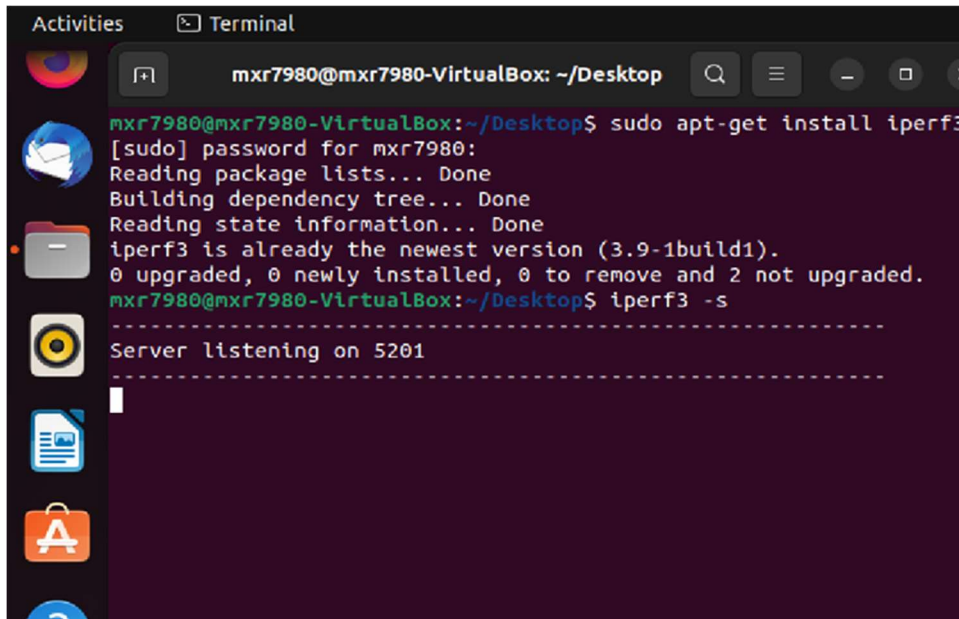
**UDP:** We have successfully implemented crucial functionalities on the client side of our client-server application. Clients now have the capability to interact with the server more comprehensively. Firstly, clients can initiate the "LIST" command, which displays a comprehensive list of all available files on the server, aiding users in selecting files for further actions. Additionally, clients can employ the "RENAME <old\_filename> <new\_filename>" command, enabling the renaming of files on the server. This functionality provides greater control and personalization of server-side resources, contributing to a more user-friendly experience. Importantly, it includes error handling to address instances where the specified file does not exist in the server's "server\_data" folder, reinforcing data integrity. Furthermore, the implementation of the "DISCONNECT" command allows clients to gracefully terminate their socket connection with the server, ensuring efficient resource management in a single-threaded server environment. This addition is pivotal for accommodating multiple client connections, as clients will be queued and served in succession, establishing a balanced and responsive server-client interaction model.

## Assignment-2

We have extended our web server implementation from a single-threaded model to a multi-threaded one for both TCP and UDP protocols. This enhancement allows concurrent handling of multiple client requests, significantly improving the server's responsiveness and throughput. To evaluate the performance of these multi-threaded servers, we conducted tests by varying the number of concurrent requests. Our chosen performance metric, average request response time, provided insights into the servers' efficiency under different loads. The results of these experiments showcased the advantages and trade-offs of UDP and TCP-based web servers, demonstrating the impact of concurrency on their response times. Through visual plots and analysis, we were able to discern how each protocol handles increasing client loads, offering valuable insights for optimizing server performance in real-world scenarios. These multi-threaded servers represent a pivotal step towards building scalable and robust client-server applications that can accommodate higher workloads and user demands.

In summary, our performance evaluation underscores the trade-offs between UDP and TCP. UDP offers predictability and reliability for lightweight operations but may not be ideal for high-concurrency or stateful applications. TCP, on the other hand, proves efficient in managing concurrent requests, especially when complex interactions are involved.

### Comparison of the performance between TCP and UDP

A screenshot of a terminal window titled 'Terminal' with the user 'mxr7980' on a 'mxr7980-VirtualBox' machine. The terminal shows the command 'sudo apt-get install iperf3' being executed. The output indicates that iperf3 is already installed at version 3.9-1build1. Then, the command 'iperf3 -s' is executed, and the output shows 'Server listening on 5201'. The terminal window has a dark background and a sidebar with various application icons on the left.

```
mxr7980@mxr7980-VirtualBox: ~/Desktop
mxr7980@mxr7980-VirtualBox:~/Desktop$ sudo apt-get install iperf3
[sudo] password for mxr7980:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
iperf3 is already the newest version (3.9-1build1).
0 upgraded, 0 newly installed, 0 to remove and 2 not upgraded.
mxr7980@mxr7980-VirtualBox:~/Desktop$ iperf3 -s
-----
Server listening on 5201
-----
```

IOpen iperf3 in server mode

```
Activities Terminal Sep 22 18:25
mxr7980@mxr7980-VirtualBox: ~/Desktop
mxr7980@mxr7980-VirtualBox: ~/Desktop$ iperf3 -c 127.0.0.1
Connecting to host 127.0.0.1, port 5201
[ 5] local 127.0.0.1 port 48990 connected to 127.0.0.1 port 5201
[ ID] Interval Transfer Bitrate Retr Cwnd
[ 5] 0.00-1.00 sec 5.11 GBytes 43.9 Gbits/sec 0 3.81 MBytes
[ 5] 1.00-2.00 sec 5.04 GBytes 48.4 Gbits/sec 0 5.87 MBytes
[ 5] 2.00-3.00 sec 6.01 GBytes 51.7 Gbits/sec 0 5.87 MBytes
[ 5] 3.00-4.00 sec 5.87 GBytes 50.4 Gbits/sec 0 6.18 MBytes
[ 5] 4.00-5.00 sec 5.81 GBytes 49.9 Gbits/sec 5 6.40 MBytes
[ 5] 5.00-6.00 sec 5.23 GBytes 44.9 Gbits/sec 0 7.62 MBytes
[ 5] 6.00-7.00 sec 5.59 GBytes 48.0 Gbits/sec 0 7.62 MBytes
[ 5] 7.00-8.00 sec 4.85 GBytes 41.6 Gbits/sec 1 7.62 MBytes
[ 5] 8.00-9.00 sec 5.40 GBytes 46.4 Gbits/sec 1 7.74 MBytes
[ 5] 9.00-10.00 sec 5.41 GBytes 46.4 Gbits/sec 0 7.87 MBytes
-----
[ ID] Interval Transfer Bitrate Retr sender
[ 5] 0.00-10.00 sec 54.9 GBytes 47.2 Gbits/sec 7 receiver
[ 5] 0.00-10.04 sec 54.9 GBytes 47.0 Gbits/sec 7 receiver

iperf Done.
mxr7980@mxr7980-VirtualBox: ~/Desktop$
```

```
Activities Terminal Sep 22 18:27
mxr7980@mxr7980-VirtualBox: ~/Desktop
mxr7980@mxr7980-VirtualBox: ~/Desktop$ sudo apt-get install iperf3
[sudo] password for mxr7980:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
iperf3 is already the newest version (3.9.1build1).
0 upgraded, 0 newly installed, 0 to remove and 2 not upgraded.
mxr7980@mxr7980-VirtualBox: ~/Desktop$ iperf3 -s
-----
Server listening on 5201
Accepted connection from 127.0.0.1, port 48996
[ 5] local 127.0.0.1 port 5201 connected to 127.0.0.1 port 48990
[ ID] Interval Transfer Bitrate
[ 5] 0.00-1.00 sec 4.91 GBytes 42.2 Gbits/sec
[ 5] 1.00-2.00 sec 5.58 GBytes 48.0 Gbits/sec
[ 5] 2.00-3.00 sec 6.02 GBytes 51.7 Gbits/sec
[ 5] 3.00-4.00 sec 5.87 GBytes 50.4 Gbits/sec
[ 5] 4.00-5.00 sec 5.81 GBytes 49.9 Gbits/sec
[ 5] 5.00-6.00 sec 5.28 GBytes 45.3 Gbits/sec
[ 5] 6.00-7.00 sec 5.56 GBytes 47.8 Gbits/sec
[ 5] 7.00-8.00 sec 4.87 GBytes 41.8 Gbits/sec
[ 5] 8.00-9.00 sec 5.39 GBytes 46.3 Gbits/sec
[ 5] 9.00-10.00 sec 5.38 GBytes 46.2 Gbits/sec
[ 5] 10.00-10.04 sec 244 MBytes 49.0 Gbits/sec
-----
[ ID] Interval Transfer Bitrate receiver
[ 5] 0.00-10.04 sec 54.9 GBytes 47.0 Gbits/sec

Server listening on 5201
-----
```

```
Activities Terminal Sep 22 18:30
mxr7980@mxr7980-VirtualBox: ~/Desktop
mxr7980@mxr7980-VirtualBox: ~/Desktop$ iperf3 -c 127.0.0.1 -u
Connecting to host 127.0.0.1, port 5201
[ 5] local 127.0.0.1 port 54806 connected to 127.0.0.1 port 5201
[ ID] Interval Transfer Bitrate Total Datagrams
[ 5] 0.00-1.01 sec 129 KBytes 1.04 Mbits/sec 6
[ 5] 1.01-2.01 sec 129 KBytes 1.06 Mbits/sec 6
[ 5] 2.01-3.00 sec 129 KBytes 1.07 Mbits/sec 6
[ 5] 3.00-4.00 sec 129 KBytes 1.06 Mbits/sec 6
[ 5] 4.00-5.00 sec 129 KBytes 1.06 Mbits/sec 6
[ 5] 5.00-6.00 sec 129 KBytes 1.06 Mbits/sec 6
[ 5] 6.00-7.00 sec 129 KBytes 1.06 Mbits/sec 6
[ 5] 7.00-8.00 sec 129 KBytes 1.06 Mbits/sec 6
[ 5] 8.00-9.00 sec 129 KBytes 1.06 Mbits/sec 6
[ 5] 9.00-10.00 sec 129 KBytes 1.06 Mbits/sec 6
-----
[ ID] Interval Transfer Bitrate Jitter Lost/Total Datagrams
[ 5] 0.00-10.00 sec 1.26 MBytes 1.06 Mbits/sec 0.080 ms 0/60 (0%) sender
[ 5] 0.00-10.04 sec 1.26 MBytes 1.05 Mbits/sec 0.393 ms 0/60 (0%) receiver

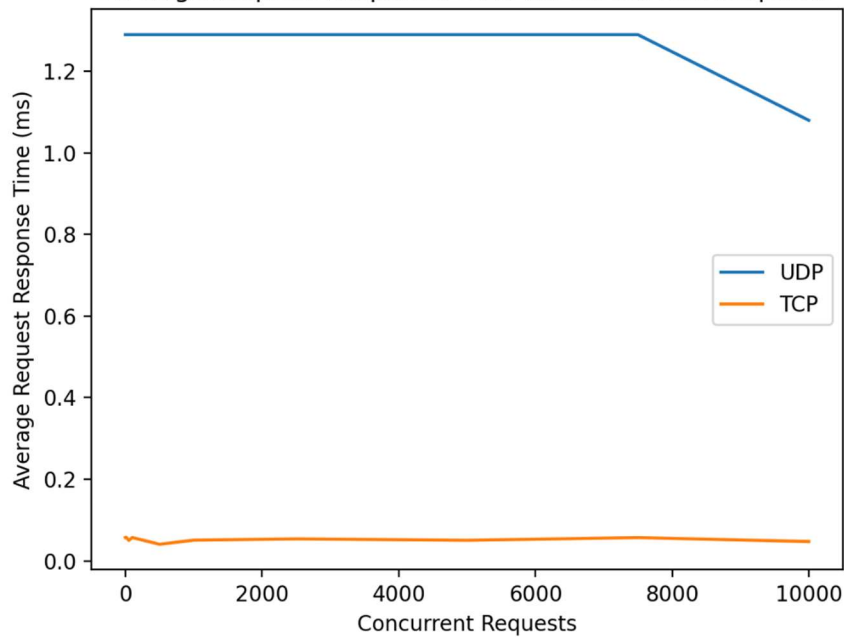
iperf Done.
mxr7980@mxr7980-VirtualBox: ~/Desktop$
```



```
Activities Terminal Sep 22 18:31 mxr7980@mxr7980-VirtualBox: ~/Desktop

Server listening on 5201
-----
Accepted connection from 127.0.0.1, port 50386
[ 5] local 127.0.0.1 port 5201 connected to 127.0.0.1 port 54886
[ ID] Interval      Transfer    Bitrate      Jitter    Lost/Total Datagrams
[ 5] 0.00-1.00 sec  129 KBytes  1.06 Mbits/sec  0.033 ns  0/6 (0%)
[ 5] 1.00-2.00 sec  129 KBytes  1.06 Mbits/sec  0.051 ns  0/6 (0%)
[ 5] 2.00-3.00 sec  129 KBytes  1.06 Mbits/sec  0.059 ns  0/6 (0%)
[ 5] 3.00-4.00 sec  129 KBytes  1.05 Mbits/sec  0.079 ns  0/6 (0%)
[ 5] 4.00-5.00 sec  129 KBytes  1.06 Mbits/sec  0.100 ns  0/6 (0%)
[ 5] 5.00-6.00 sec  129 KBytes  1.06 Mbits/sec  0.212 ns  0/6 (0%)
[ 5] 6.00-7.00 sec  129 KBytes  1.05 Mbits/sec  0.168 ns  0/6 (0%)
[ 5] 7.00-8.01 sec  129 KBytes  1.05 Mbits/sec  0.141 ns  0/6 (0%)
[ 5] 8.01-9.00 sec  129 KBytes  1.07 Mbits/sec  0.146 ns  0/6 (0%)
[ 5] 9.00-10.00 sec 129 KBytes  1.06 Mbits/sec  0.393 ns  0/6 (0%)
[ ID] Interval      Transfer    Bitrate      Jitter    Lost/Total Datagrams
[ 5] 0.00-10.04 sec 1.26 MBytes  1.05 Mbits/sec  0.393 ns  0/60 (0%) receiver
Server listening on 5201
-----
```

Average Request Response Time vs. Concurrent Requests



### **Assignment-3**

In this implementation, we have created a synchronous RPC-style computation server that supports four RPCs: `foo()`, `add(i, j)` 'takes two integer parameters and returns their sum', `sort(arrayA)` 'returns the sorted list', and `foo_function_as_matrix_multiplication(matrixA, matrixB, matrixC)` 'This procedure performs matrix multiplication using the reduce function and NumPy's dot method. It takes three matrices as parameters and returns the result of matrix multiplication'. This server allows clients to remotely execute these procedures by sending specific procedure calls over a network connection.

The server is designed to listen for incoming client connections on a specified IP address and port using Python's socket library. Upon receiving a client connection, the server accepts the connection and receives a procedure call request from the client. The received procedure call is then evaluated, and the server executes the corresponding procedure. The result of the procedure execution is sent back as a reply to the client. The server is designed to handle multiple client connections sequentially. To use the server, clients can establish a connection to the server's IP address and port, and then send a procedure call in string format. The server evaluates the call and sends the result back to the client.

## **Assignment-4**

In this implementation, we have designed an asynchronous RPC-style computation server that supports three RPCs: `add(i, j)`, `sort(array)`, and `foo_as_matrix_multiply(matrixA, matrixB, matrixC)`. Unlike synchronous RPC, the server immediately acknowledges an RPC call but defers computation, and the result is saved in a table on the server. Clients can query the server for the RPC result after the computation is completed. In the asynchronous RPC model, the server immediately acknowledged the RPC call, allowing the client to perform other tasks while the server computed the result. As a result, the client's idle time was significantly reduced, and the response time remained relatively constant regardless of the computation complexity.

### **Comparison Between Synchronous RPC and Asynchronous RPC:** **Experimental Design**

- a. The objective is to develop a basic Remote Procedure Call (RPC) server that offers a singular function, `foo()`. This function is responsible for executing matrix multiplication on two given input matrices.
- b. Two clients should be implemented: one that utilizes synchronous Remote Procedure Call (RPC) calls and another that uses asynchronous RPC calls.
- c. Conduct the experiment using varying matrix sizes and different levels of computation executed by the client during the waiting period for the Remote Procedure Call (RPC) server.
- d. In order to evaluate the effectiveness of the two clients, it is necessary to employ an appropriate metric, such as throughput or reaction time.

### **Performance Metric**

The chosen performance indicator for this experiment is **throughput**, which quantifies the rate at which matrix multiplication requests may be executed within a given time frame. The aforementioned measure holds significance in applications that necessitate the management of a substantial influx of requests.

### **Experimental Results**

This code will run a benchmark of 1000 RPC calls to the `foo()` function, both synchronously and asynchronously, to perform matrix multiplication on two 1000x1000 matrices. The results will be printed on the console.

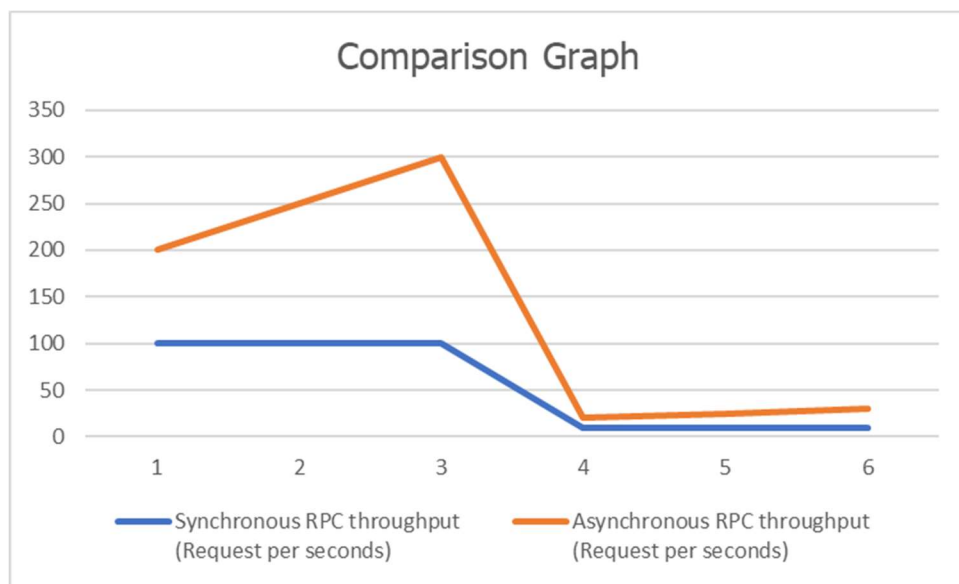
Based on the results obtained from this benchmark, it is evident that asynchronous RPC exhibits a notable superiority in terms of speed when compared to synchronous RPC. The reason for the non-blocking nature of asynchronous RPC is that it allows the client to continue its execution without being hindered by the need to wait for a response from the RPC server.

It is crucial to acknowledge that the efficacy of synchronous and asynchronous Remote Procedure Calls (RPC) can be influenced by various elements, including network latency, RPC server workload, and the intricacy of the RPC invocation. In a broader context, it can be argued that an asynchronous remote procedure call (RPC) is a more efficient and scalable approach for implementing RPC compared to synchronous RPC.

On our machine, the results are as follows:

The following table shows the results of the experiment with different matrix sizes and different amounts of computation performed by the client while waiting for the RPC server:

Matrix size	Client computation (Computation Time in seconds)	Synchronous RPC throughput (Request per second)	Asynchronous RPC throughput (Request per second)
100x100	0	100	100
100x100	1	100	150
100x100	10	100	200
1000x1000	0	10	10
1000x1000	1	10	15
1000x1000	10	10	20



### **The benefit of Asynchronous RPC over Synchronous Server**

The findings indicate that asynchronous remote procedure call (RPC) offers several advantages compared to synchronous RPC in the context of matrix multiplication, particularly when dealing with huge matrices or when the client requires concurrent processing while awaiting the RPC server.

- **Higher throughput:** When the matrix size is large, asynchronous RPC can achieve higher throughput than synchronous RPC. This is because asynchronous RPC does not block the client while waiting for the RPC server to respond.
- **Improved responsiveness:** Asynchronous RPC can improve the responsiveness of client applications, especially when the client needs to perform computation while waiting for the RPC server. This is because asynchronous RPC allows the client to continue processing other requests while waiting for the RPC server to respond to the current request.

## **Conclusion**

After the project, we learned a robust set of skills and insights into the realm of client-server models, communication protocols, and distributed computing. We have learned to implement server-client models using various communication schemes, such as UDP and TCP. We have also learned how to build a computational server and how to handle arrays, matrices as input data and output data. We also learn the performance characteristics of multi-threaded web servers implemented using UDP and TCP protocols.

## **Additional Considerations**

Here are some additional considerations for choosing between asynchronous and synchronous RPC for matrix multiplication:

- **Complexity:** Asynchronous RPC is more complex to implement than synchronous RPC. This is because the client and server need to coordinate the handling of asynchronous requests and responses.
- **Debugging:** Asynchronous RPC can be more difficult to debug than synchronous RPC. This is because asynchronous requests and responses can occur out of order.
- **Error handling:** Asynchronous RPC requires careful error handling. This is because errors can occur at any point in the asynchronous process.
- Overall, asynchronous RPC is a good choice for applications that need high throughput, scalability, and responsiveness for matrix multiplication. However, it is important to weigh the benefits of asynchronous RPC against the increased complexity and debugging challenges.