CS 744: Design and Engineering of Computing Systems
Autumn 2023

Project DECServer

- Soumik Dutta (23M0826)
- Kumar Ankur (23M0829)

Table of content:

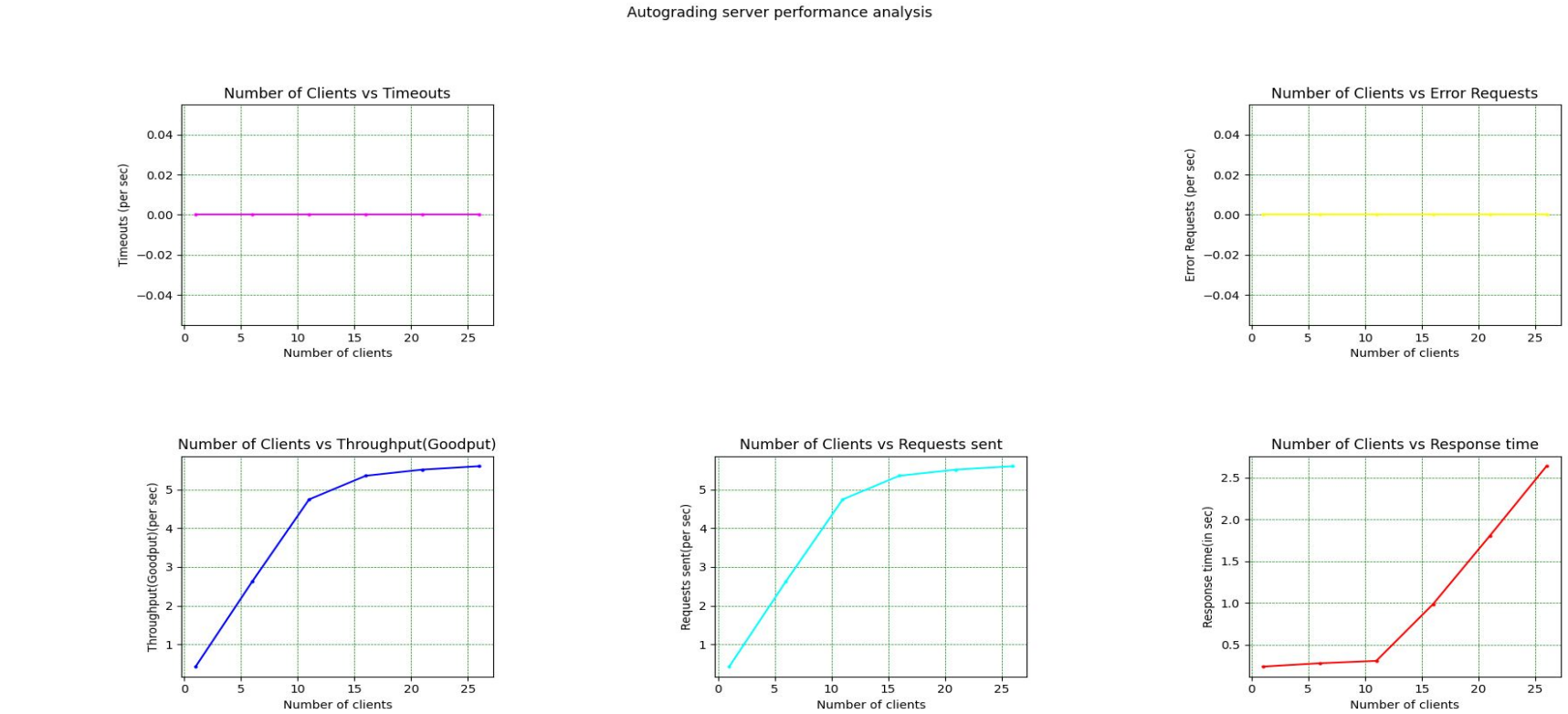
Autograder version-1

Architectural Design:

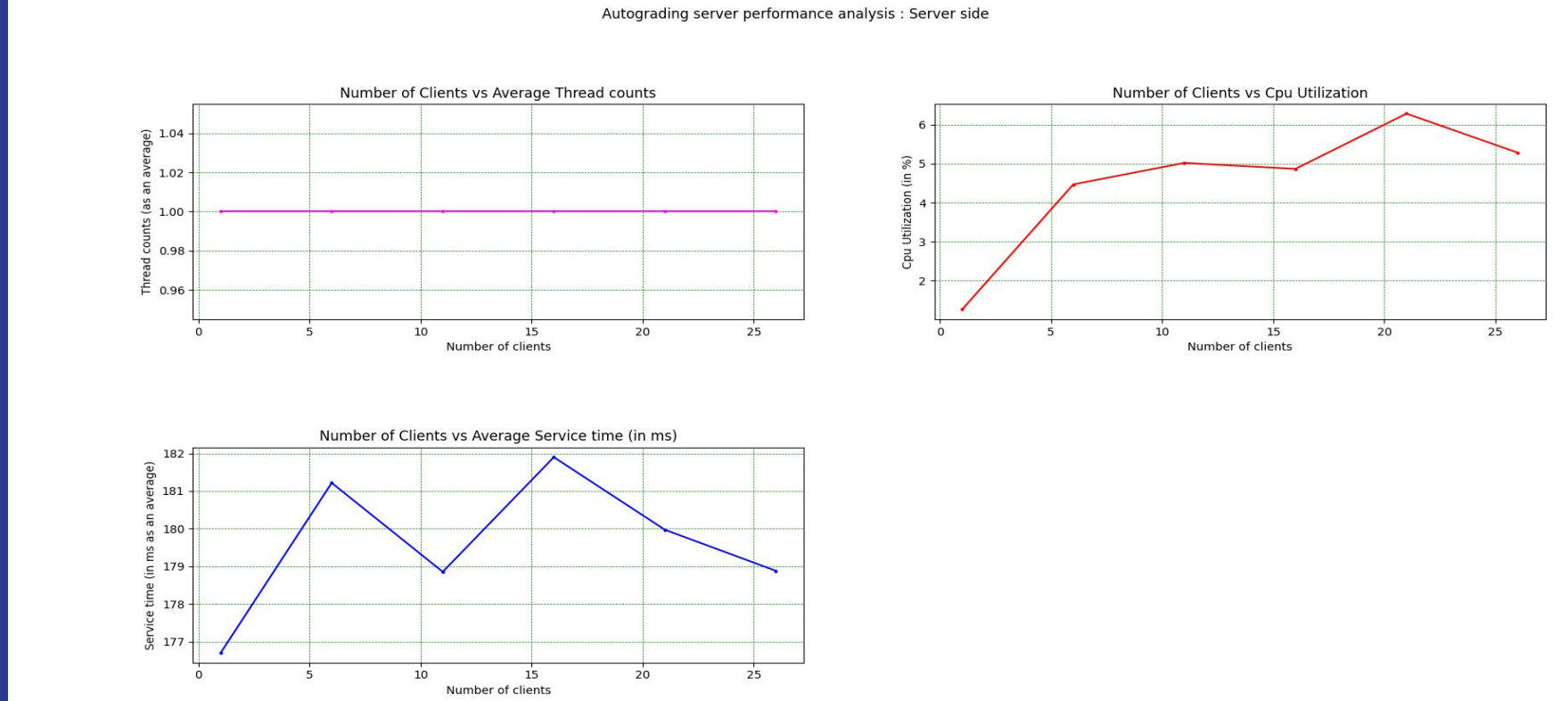
Client - This is a closed loop system in which client submits a request to the server and waits until it gets the grading response. If it does not get the response in the stipulated time then timeout will happen.
After getting the response the client will wait for the “think time” and again makes a new submission request.

Server - In this version the server is a single threaded server that accept client’s grading request and based upon the result of execution it sends either the error such as compilation error or runtime error or sends the output to the client.

Performance Test Analysis (Client Side)



Performance Test Analysis (Server Side)



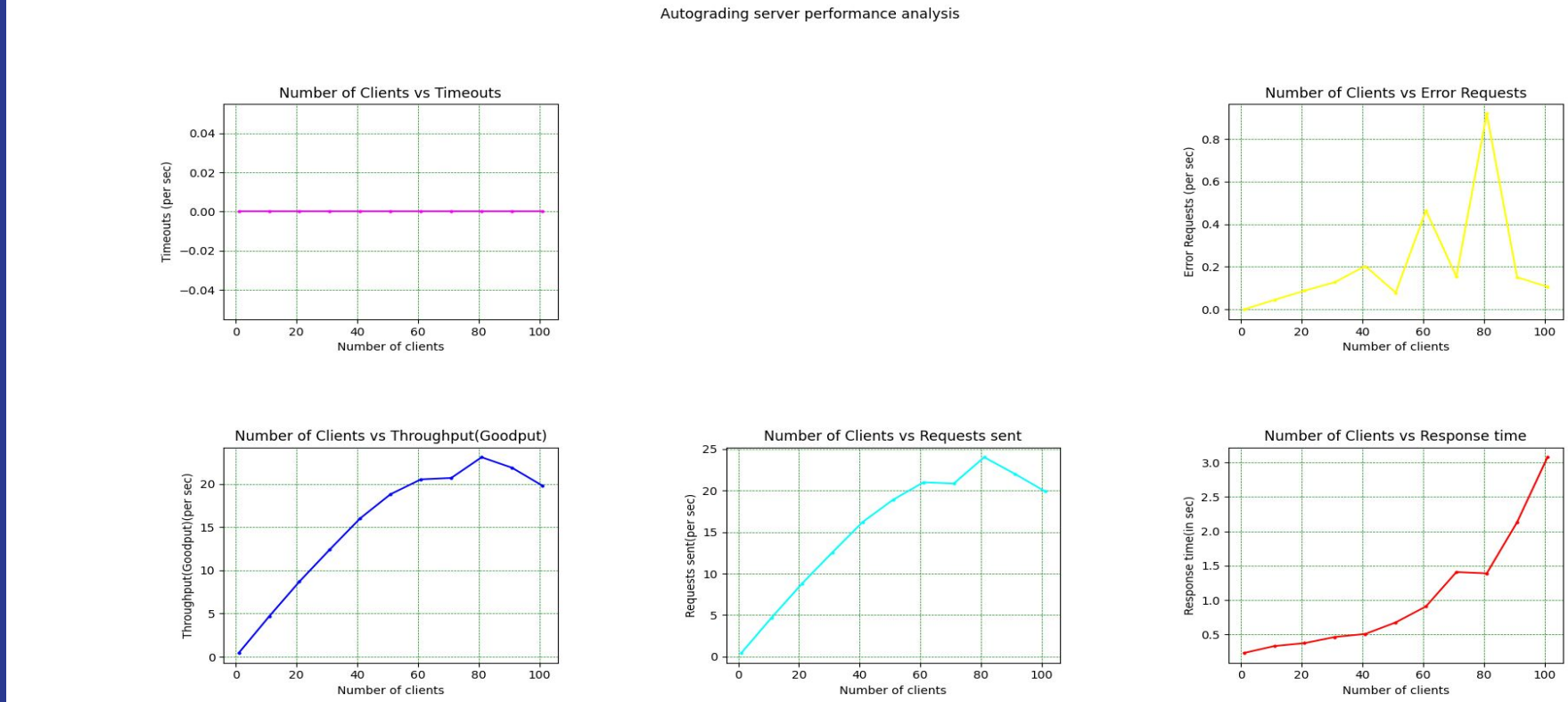
Autograder version-2

Architectural Design:

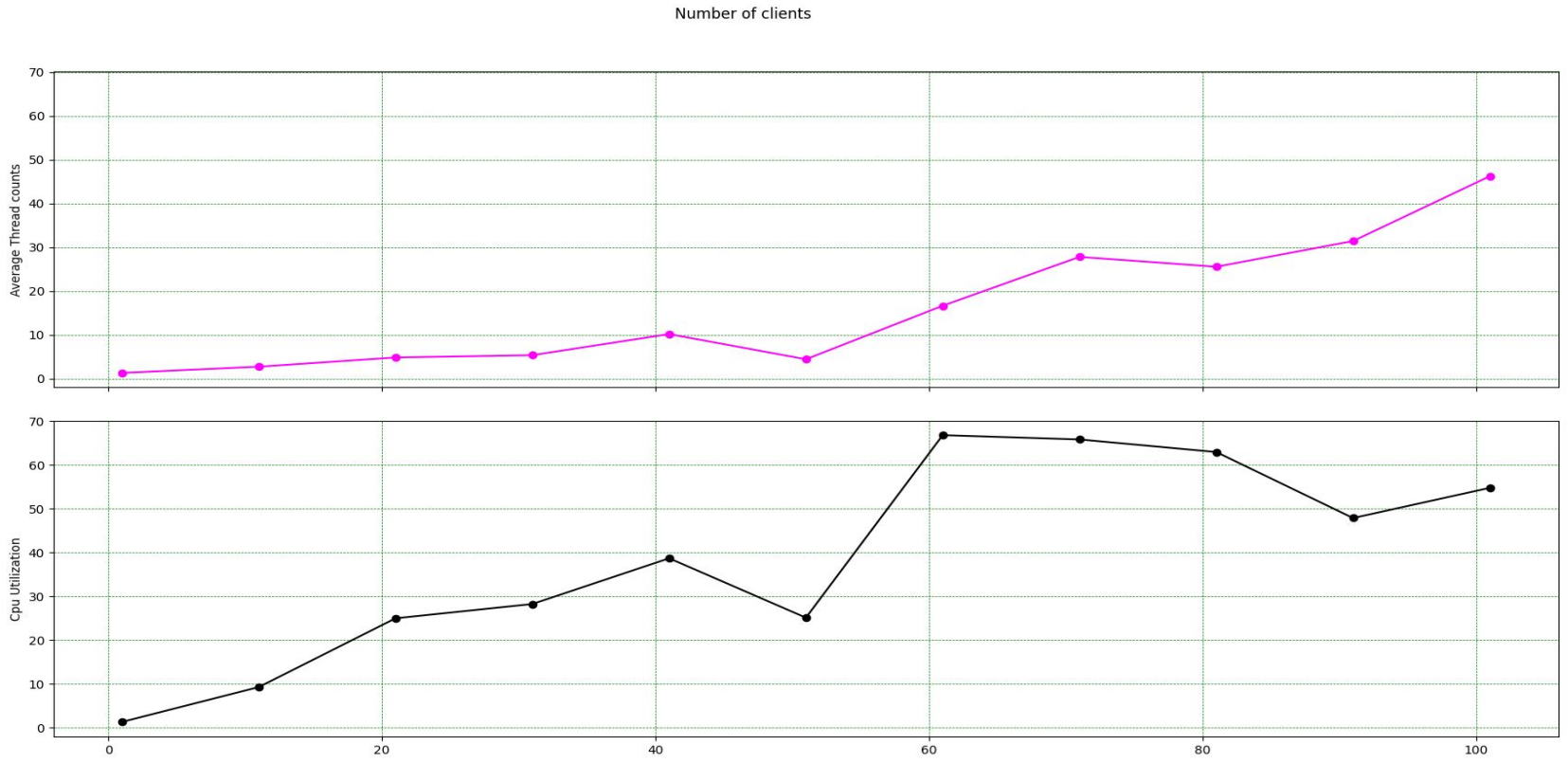
Client - This is a closed loop system in which client submits a request to the server and waits until it gets the grading response. If it does not get the response in the stipulated time then timeout will happen.
After getting the response the client will wait for the “think time” and again makes a new submission request.

Server - In this version the server is a multi-threaded server that spawns a thread for each client’s grading request without any upper limit and based upon the result of execution it sends either the error such as compilation error or runtime error or sends the output to the client.
The thread itself sends the response to the client

Performance Test Analysis (Client Side)



Performance Test Analysis (Server Side)



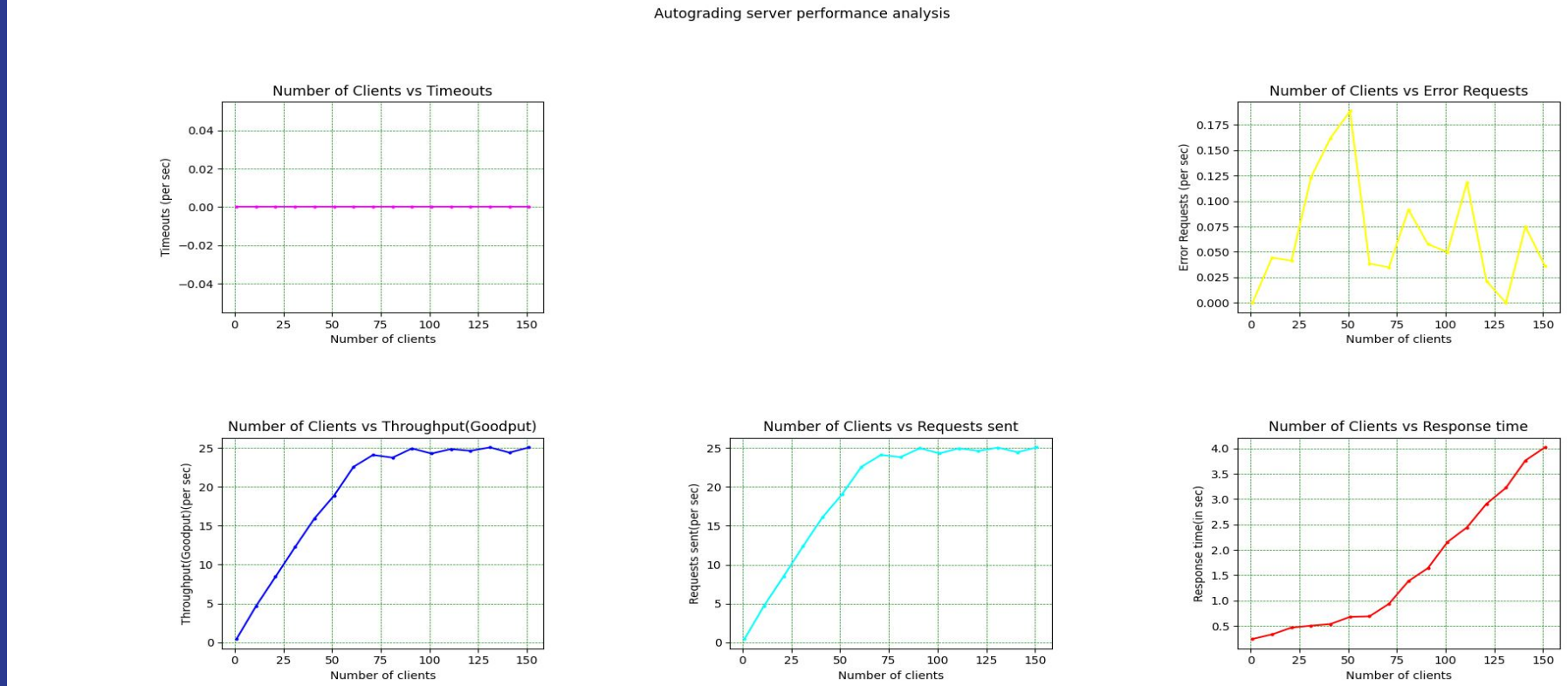
Autograder version-3

Architectural Design:

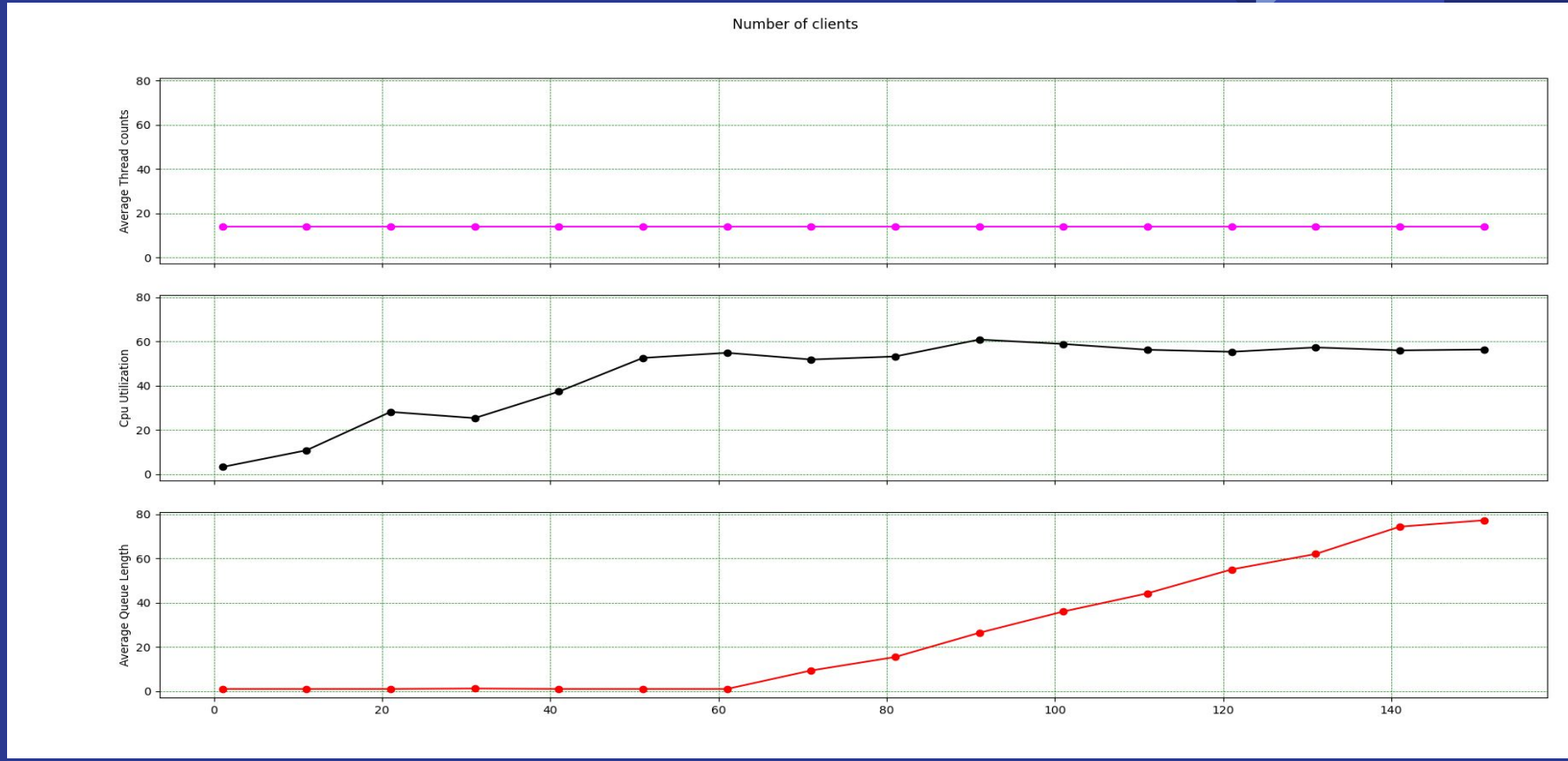
Client - This is a closed loop system in which client submits a request to the server and waits until it gets the grading response. If it does not get the response in the stipulated time then timeout will happen.
After getting the response the client will wait for the “think time” and again makes a new submission request.

Server - In this version the server is a multi-threaded server in which the main thread creates a thread pool of given size and the threads starts and wait for grading request.

Performance Test Analysis (Client Side)



Performance Test Analysis (Server Side)



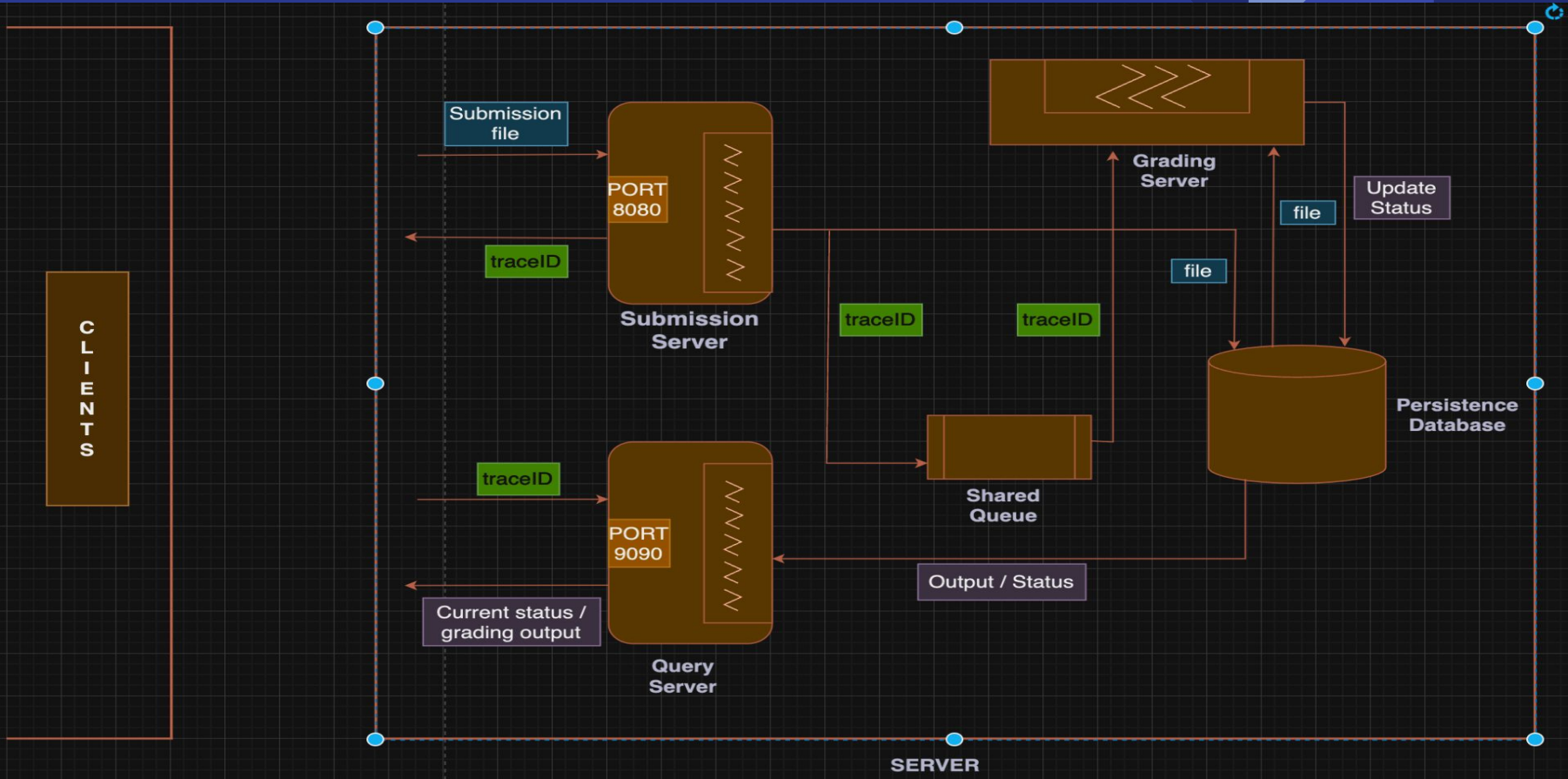
Autograder version-4

Architectural Design:

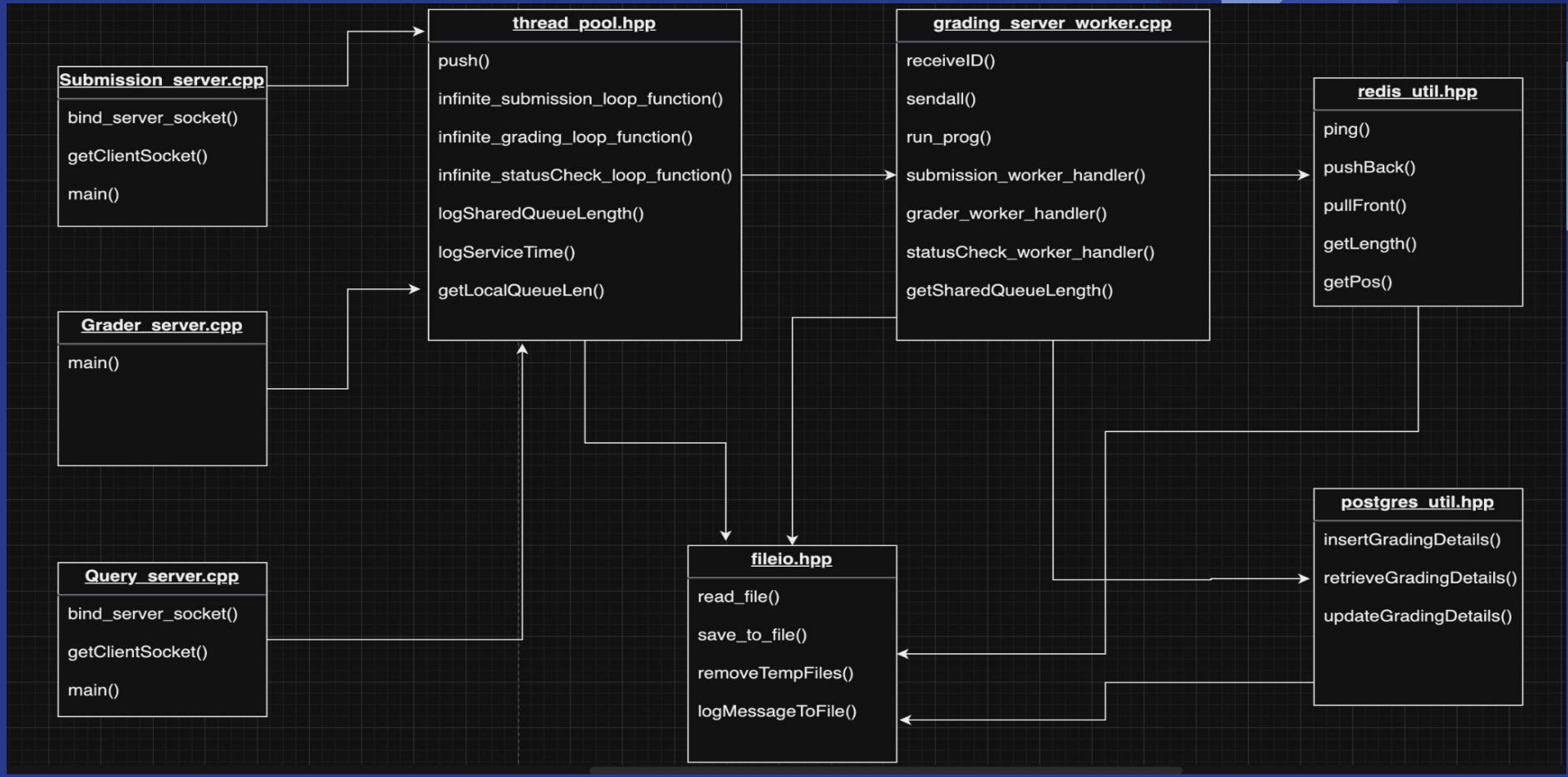
Client - We are using a closed loop system so after submitting a grading request client the client immediately gets a generalized response, then it keeps polling the server until it gets the actual grading response.
It waits for a think time, before submitting a new request to server

Server - This is an asynchronous server having three components namely submission server, query server and grader server. All works asynchronously and independently.

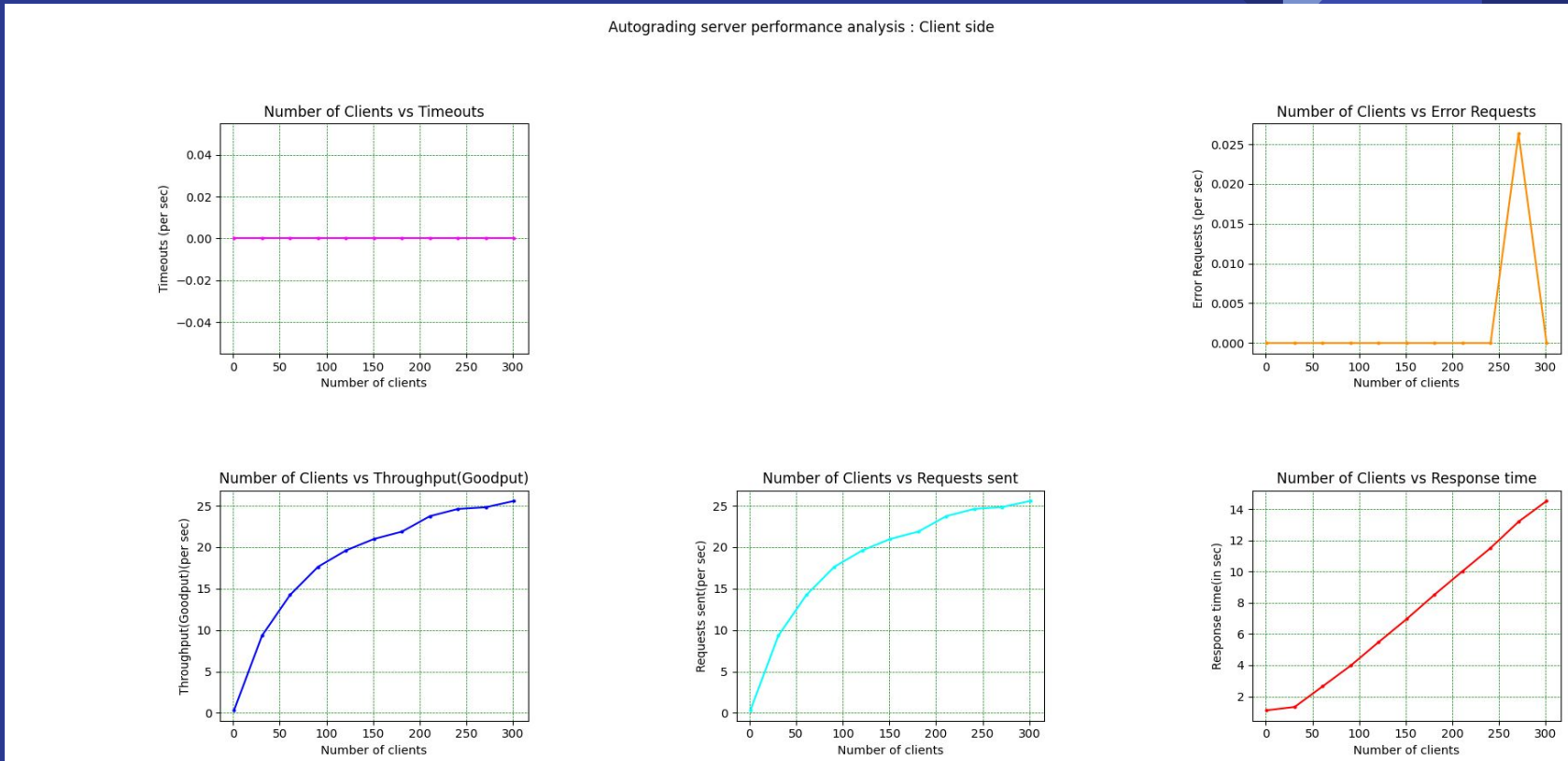
Architecture diagram



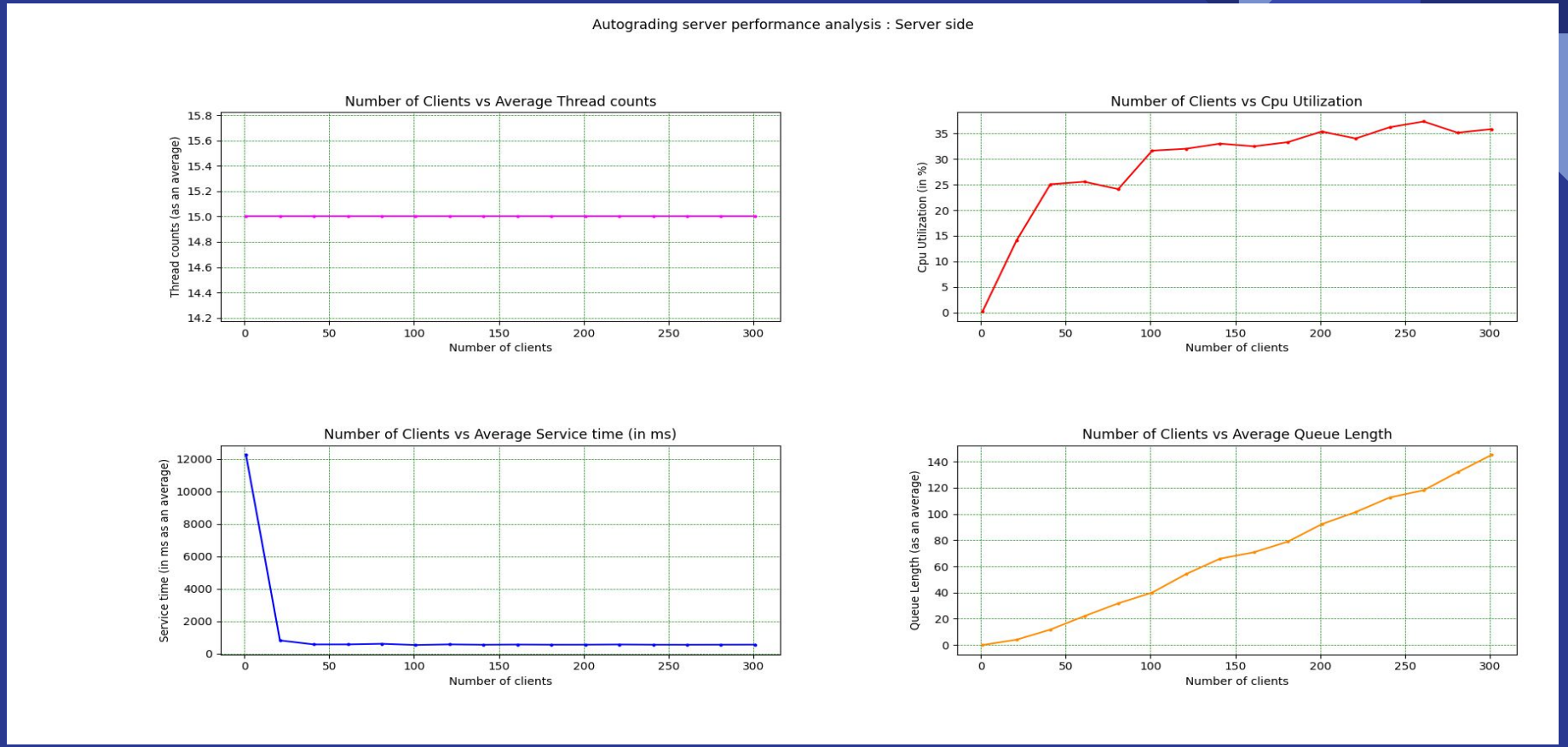
Class Diagram



Performance Analysis (Client Side)



Performance Analysis (Server Side)



Postgres and Redis Implementation

Redis - Redis is an open source no-sql database which we are using as a shared queue.

Postgres - Postgres is an open source relational database which we are using to store historical data.
This class holds the structure of the database entry which in turn holds all the info regarding submission request.

```
struct GradingDetails {
  std::string trace_id;
  std::string progress_status;
  std::string submitted_file;
  std::string lastupdated;
  std::string grading_status;
  std::string grading_output;
};
```

Unique ID generation

Unique id is generated using the following function:

```
long long Thread_pool::getUniqueId()
 {
        std::random_device rd;
        std::mt19937 gen(rd());
        std::uniform_int_distribution<long long int> dist(0, std::numeric_limits<long long int>::max());
        long long id = dist(gen);
        return id;
 }
```

Here, random_device instance ‘rd’ is used to seed , i.e, provide an initial value to the random number generator.The random number generator algorithm used here is the Mersenne Twister algorithm which is seeded with ‘rd’.With this algorithm the probability of repetition is incredibly low, likely well below 0.0000000001% or even lower

Then we created a uniform distribution for long long integers, ranging from 0 to the maximum representable value of a long long int to ensure an equal probability of generating any value within the specified range.

Finally the random number is generated using the previously defined distribution and the Mersenne Twister generator.

Reason of selecting this architecture

Microservice architecture - Instead of monolithic architecture we are using microservice architecture. We are using 3 separate server for submission, querying and grading and each server can be independently deployed and can be scaled individually depending on the load.

Resilient to reboot - If the grading server gets down, still submission server can accept grading request and query server can accept query. And with every request these servers update the status of the grading request in the postgre database.
In case all the servers gets down, then after reboot , data is still present as we are using redis and postgres as cloud service

Future Enhancements

- Application can be deployed on Docker for autoscaling.
- Both redis and postgres can be converted to clustered architecture where data loss can be prevented even if a node gets shut down

Load testing

We are using bash script and python script for load test generation and to plot the graph.
loadtest.sh - This bash script is used on the client side simulating clients and hence generates multiple clients.

plot_client_stats.py - This python script is used on the client side to generate loads of different size and client side plot.

plot_server_stats.py - This python script is also used on client side to generate plots using the log files send by the server to the client.

server_snapshot.sh - This bash script runs on the server side and logs CPU utilization, Average Number of Threads in log file and send it to server.

Comparison between graphs

Version1: As it is a single threaded server , it can serve very little request at a time, its cpu utilization is very low hence resource remains under utilized.

Version2: As it is a multithreaded server, service rate gets higher as we can leverage full cpu by multithreading, but its limitation is that as there is no upper limit on threads, after a time thread creation overhead slows the process.

Version3: Using thread we bound the upper limit of thread minimizing the thread creation overhead but still timeout happens as client waits for the grading response.

Version4: Async characteristics of this server minimizes timeout and gives the best performance.

Glimpse of version 5

Timeout at server side - If any client tries to send a malicious code which runs infinitely, then we have implemented a mechanism at server side so that each submission request can have a maximum stipulated time inside gcc.