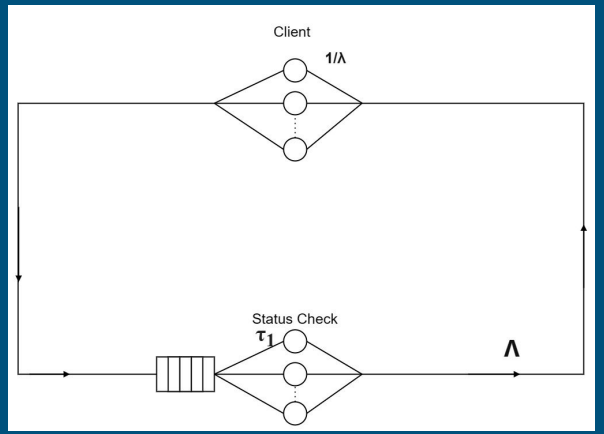
CS-744 Project Report

Submitted by,
Santhosh Kumar M (23d0369)
Goda Nagakalyani (214050010)

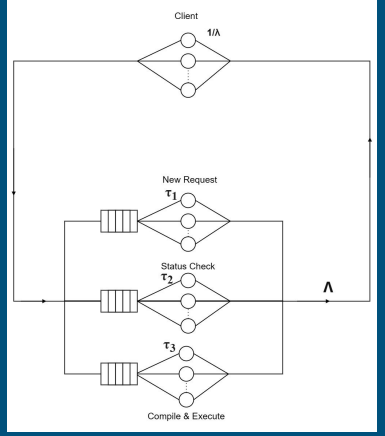
Implementation Design and Tool choices

- Programming language used - C
- Bash and awk scripting language for load testing
- Creating the files at server side using rand() function.
- Singly linked list queue using <sys/queue.h>



Challenges Faced

- Implemented closed loop system
  - Arrival rate limited to 10 requests per second
    - Otherwise TCP sockets was getting closed even before client is able to read the data (after server closes the socket)
- Load test validations
  - PASS case is used for all analysis and graphs

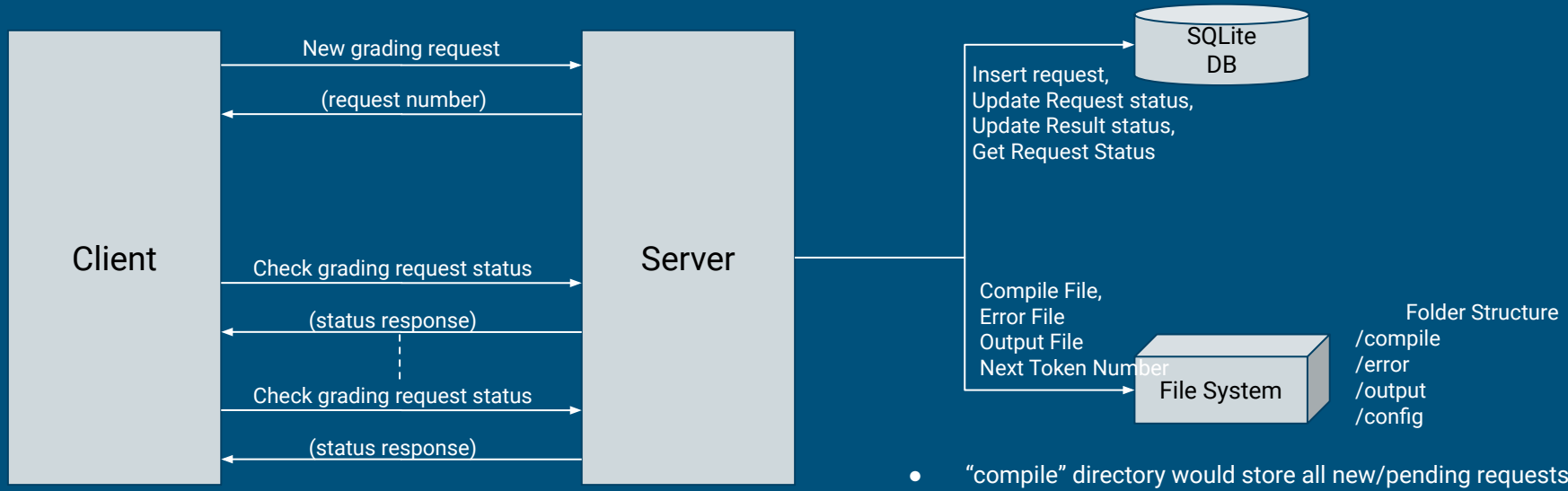


Autograder Version 4

Asynchronous Grading Server Architecture

High Level Architecture

Request ID	File Name	Request Status	Result Status
Sequential Number	<requestID>.c	New, WIP (work in progress), Completed	Pending, Pass, Compile Error, Output Error, Timeout Error



- "compile" directory would store all new/pending requests
- All compiler errors would be stored in "error" directory
- All output errors would be stored in "output" directory
- All completed request would delete files from "compile" directory"
- "config" directory would contain the last request-count available

Client - Server Request-Response Flow (1/2)

Client

Application Request Header: (new Request)
- Request Type (4 bytes)
- Request Size (4 bytes)
- Program File

Application Response Header: (new Request)
- Request ID (4 bytes)

1. Initiate new program Grading Request

Program file →

2. Receive request number

← Grading request number

Server

1. On start-up, get the next request token number (from config file stored in filesystem)
2. Initialize queue and spawn worker threads
3. Get all pending requests from database and send the requests to queue

Server setup steps

4. Listen for new grading request or status check request
5. On new grading request
   a. Receive compile file (store file in filesystem)
   b. Insert new request into SQLite DB (Request status - New, Result Status - pending)
   c. Send request to request queue
   d. Return request number and Increment request count (update config file - to handle server restarts)

Worker Request Queue

Client - Server Request-Response Flow (2/2)

Client

Application Request Header: (Status Check)

- Request Type (4 bytes)
- Request ID (4 bytes)

Application Response Header: (Status Check Request)

- Request Status (4 bytes)
- Data Size (4 bytes)
- Output content

1. Check grading request status

Grading Request No

2. Receive request status

Request Status and Output

Server

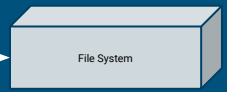
1. Receive grading request number
2. Check database for request status
3. Send request status
4. If completed, send result status and associated error files.

Worker

Worker Request Queue

1. Wait for request in Queue
2. Update Status to WIP in SQLite DB
3. Compile (output in “error” directory)
4. Execute (output in “output” directory)
5. Update request status to completed
6. Update result status (Pass, Compile Error, Output Error, Timeout Error)

loop



Design Alternatives (1/2)

Alternative 1 (Different Port)

- Separate Socket for New Request and Status Check
  - Two Server Threads
    - One thread listening for New Request
    - One thread listening for Status Check requests
  - Rationale:
    - Status Check requests would be too frequent (server thread would become bottleneck)
- Advantages:
  - Clean client design (New request on one port, status check on other port)
- Disadvantage:
  - For large programs, servers can become busy

Alternative 2 (Different Pool & Queues)

- Single Server, Two sets of (Queues and Thread Pool)
  - One Queue and Thread Pool
    - Focused on Status Check request handling
    - Other for Request handling
  - Rationale:
    - Status Check requests would be too frequent (server thread would become bottleneck)
- Advantage:
  - Able to handle more requests (scalable)
- Disadvantage:
  - Too many resources (more idle threads)

Design Alternatives (2/2)

Alternative 3 (Hybrid) - Proposed implementation

- New Request Thread
  - On receiving new grading request, create thread to receive program file and delete the
  - Rationale:
    - Enable receiving large programing files without impacting scalability
- Advantages:
  - Increased throughput without consuming too many resources.
- Disadvantage:
  - Overhead due to thread creation and deletion

Design Choices and Enhancements

Two worker Thread Pool

- Compile/Execute Thread Pool
  - Execution time is expected to be high. Hence more number of threads
- Status Response Thread Pool
  - Frequent status check requests is expected
  - Involves DB operation and result sharing (hence thread pool)
  - Small number of threads as the response time is expected to be smaller
- New Thread to receive files
  - We expect smaller number of submit requests, hence took the approach to spawn a thread

Version 5 - Design Considerations

- Design Options considered
  - Using “setrlimit”, setuid, chroot system calls
    - Re-inventing the wheel, hence dropped this approach
  - Using “safeexec as exece”
    - This is as good as using bash shell and would not problem status as required
  - Integrating safeexec as static library
    - Gives the benefits of the both the above choices

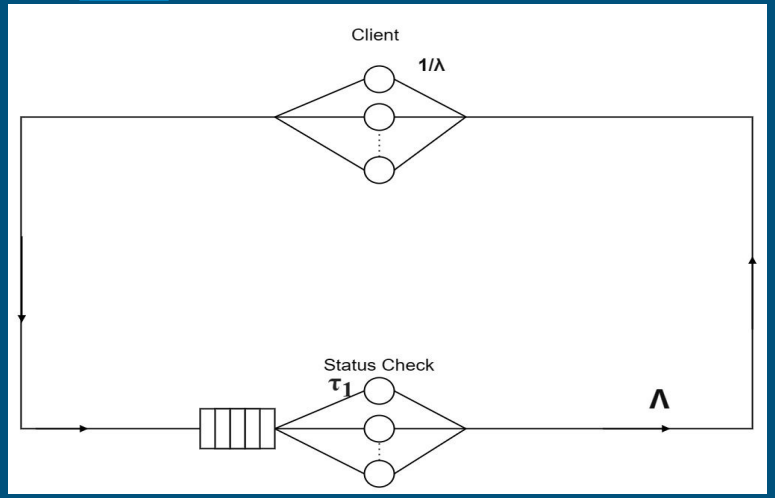
Version 5 - Implementation challenges and inefficiencies

- Safeexec design
  - Forks child process to execute and uses pipe to communicate
    - Autograder implements thread-pool.
    - New process is forked for every request received (compile, execute)
  - Safeexec programs required root privileges for certain system calls
    - seteuid(), chroot()
- Administration overhead
  - Setup users and autograder directory for each user
- Safeexec enhancements
  - –output file as additional argument
  - - number of threads limit support

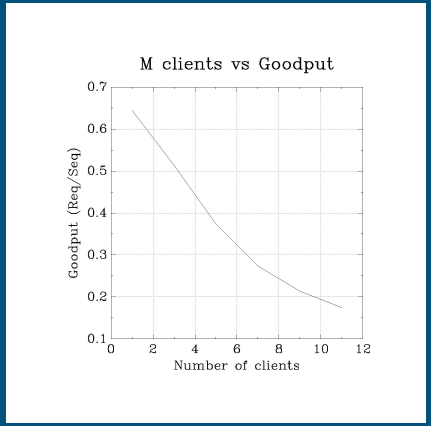
Comparative Performance Analysis

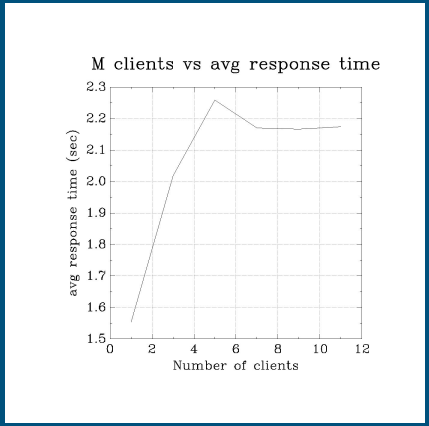
Server was executed on 8 core CPU machine

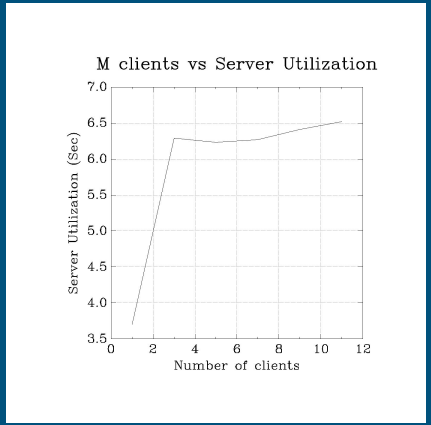
Version - 1

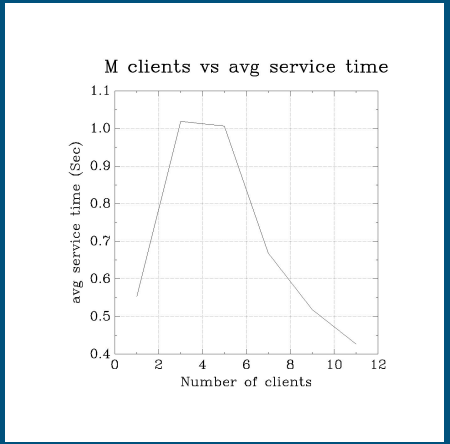


τ = 0.55 sec
λ = 10
Λ = 1.81
$\varrho = \tau * \lambda$ = 5.5(observed = 6.4)

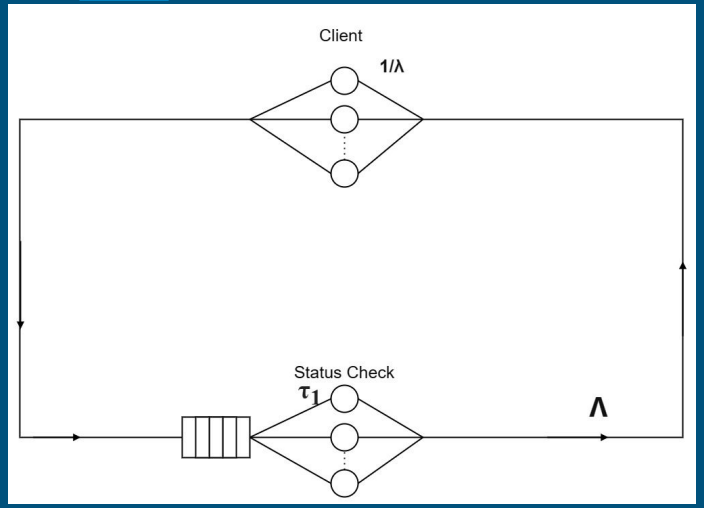




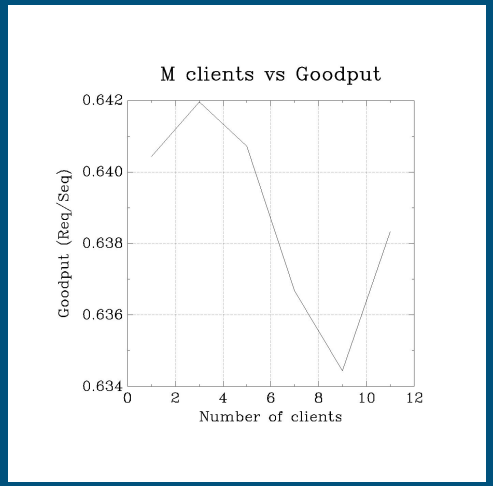


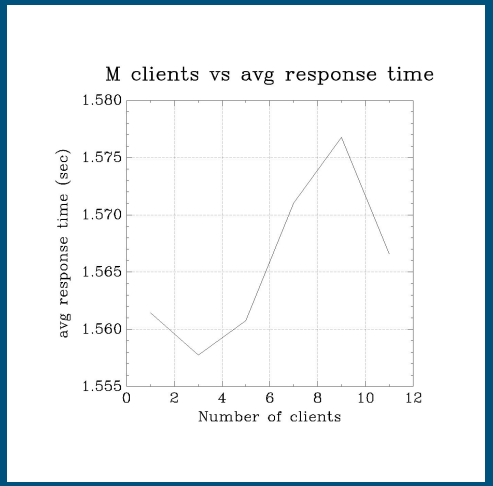


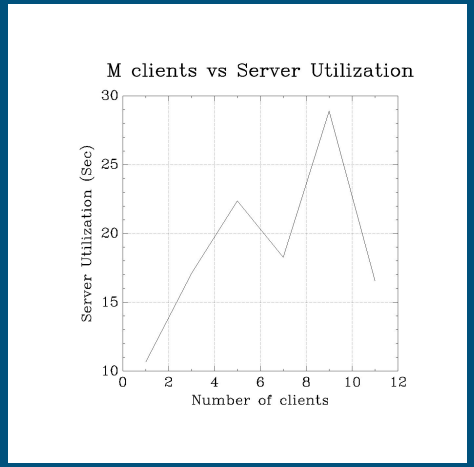
Version - 2

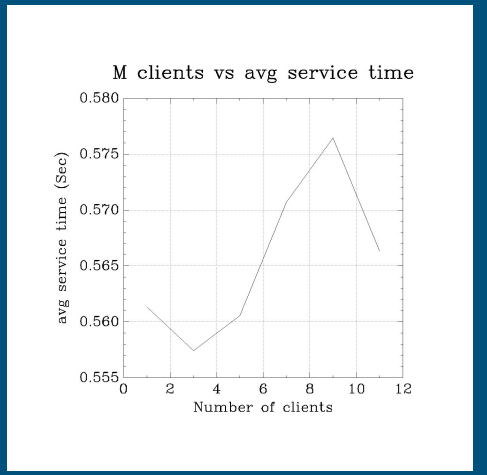


τ = 0.56 sec
λ = 10
Λ = 1.71
$\varrho = \tau * \lambda$ = 44.8 (observed = 28)

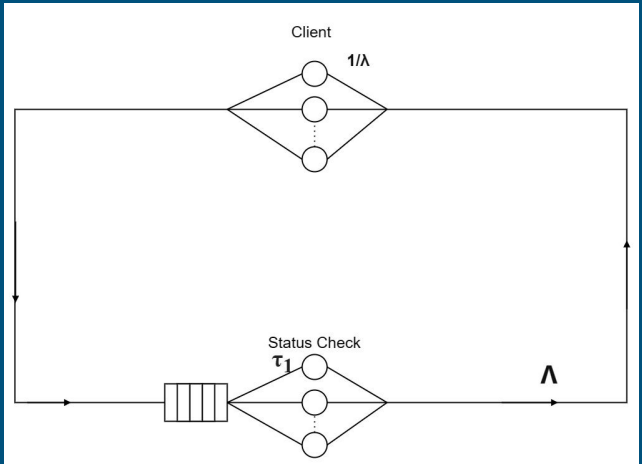




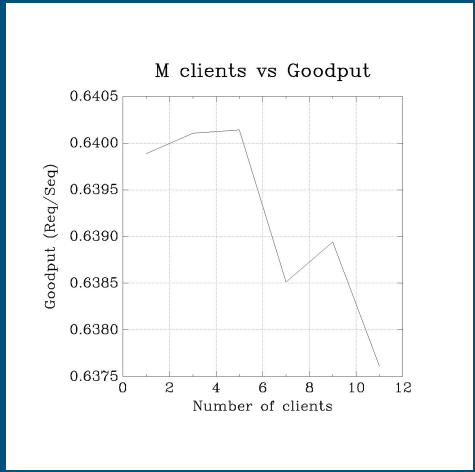


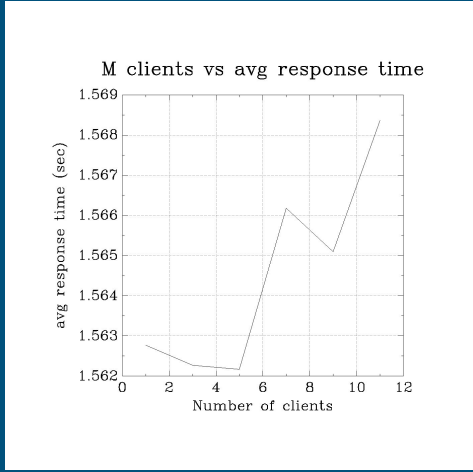


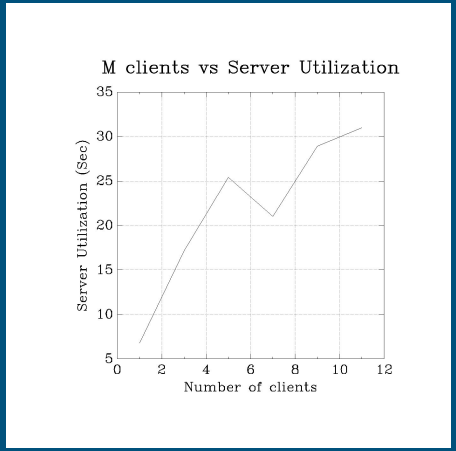
Version - 3

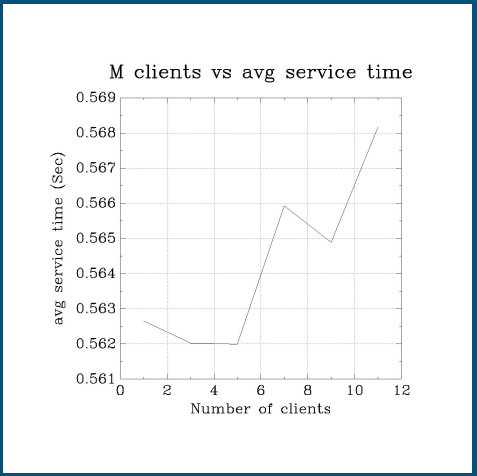


τ = 0.56 sec
λ = 10
Λ = 1.71 theoretical. (Observed = 0.63)
$\varrho = \tau * \lambda$ = 44.8 (observed = 33)
c = 8



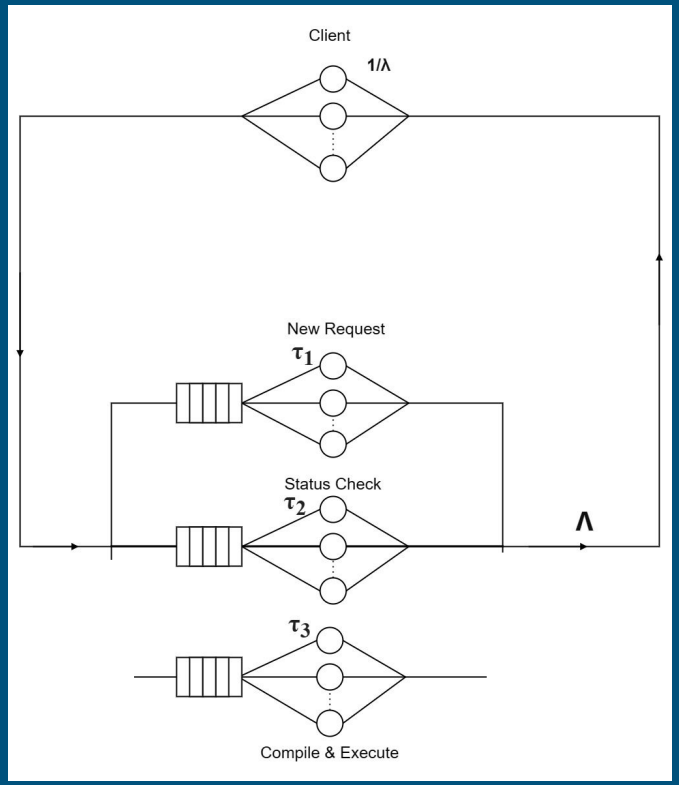


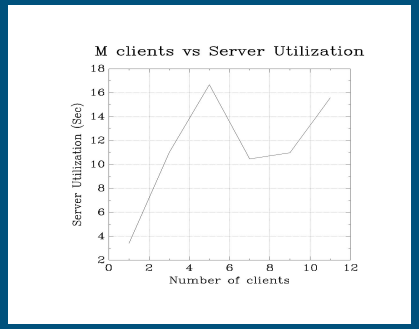


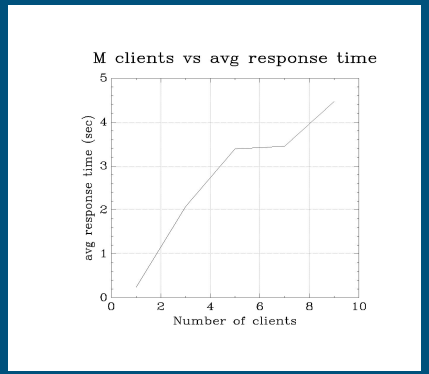


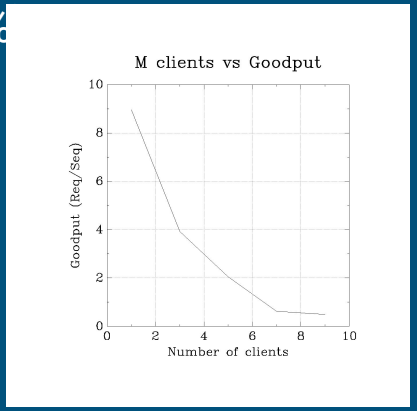
Version - 4

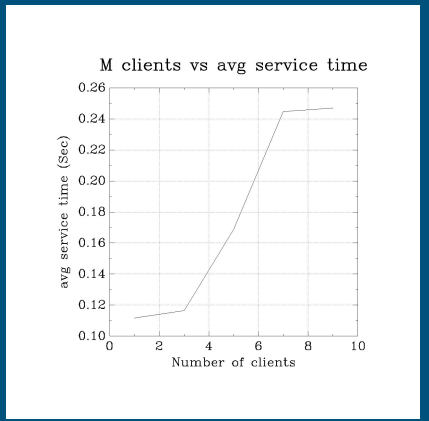
- τ = 0.11 sec (τ_1 + τ_2)
- λ = 10
- Λ = 8.96 (theoretical = 9.09)
- ϱ = τ * λ = 44.8 (observed = 17.8%











Thank You

Would request Feedback and Suggestions