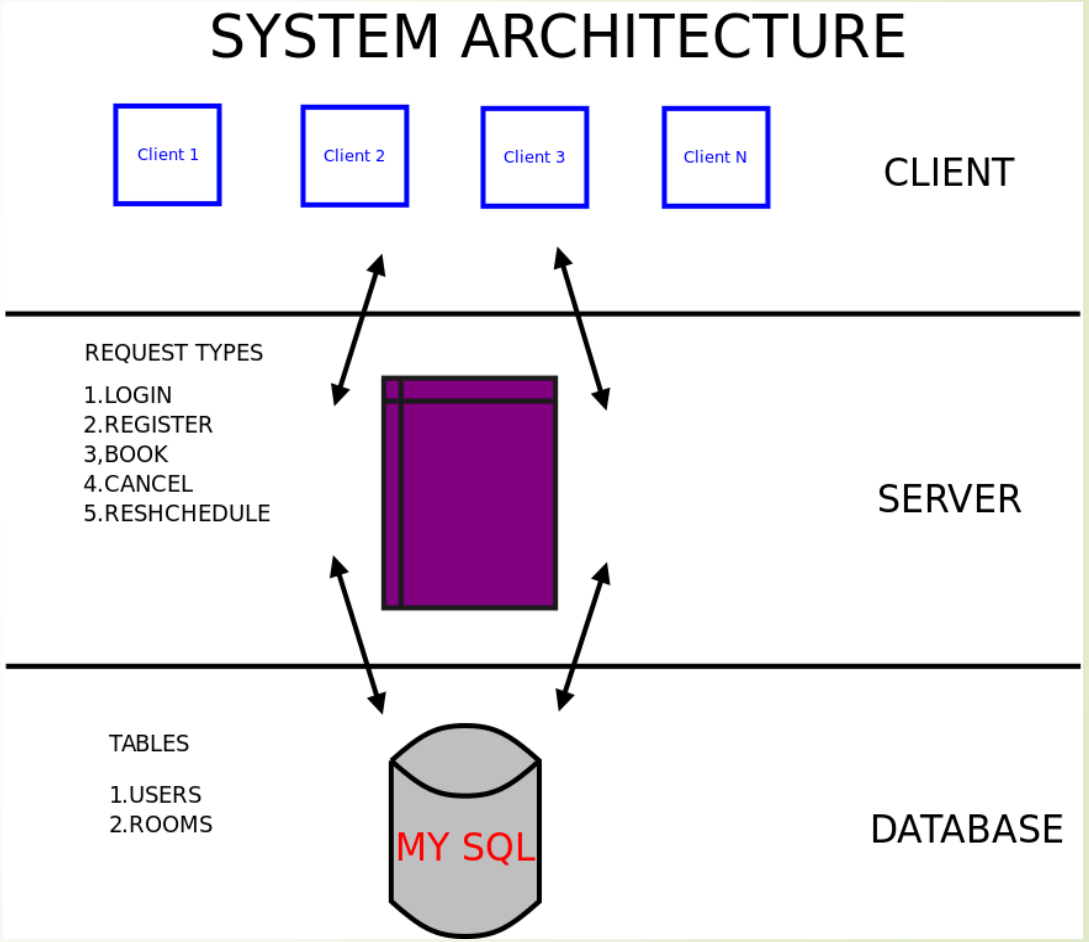
Room Booking System

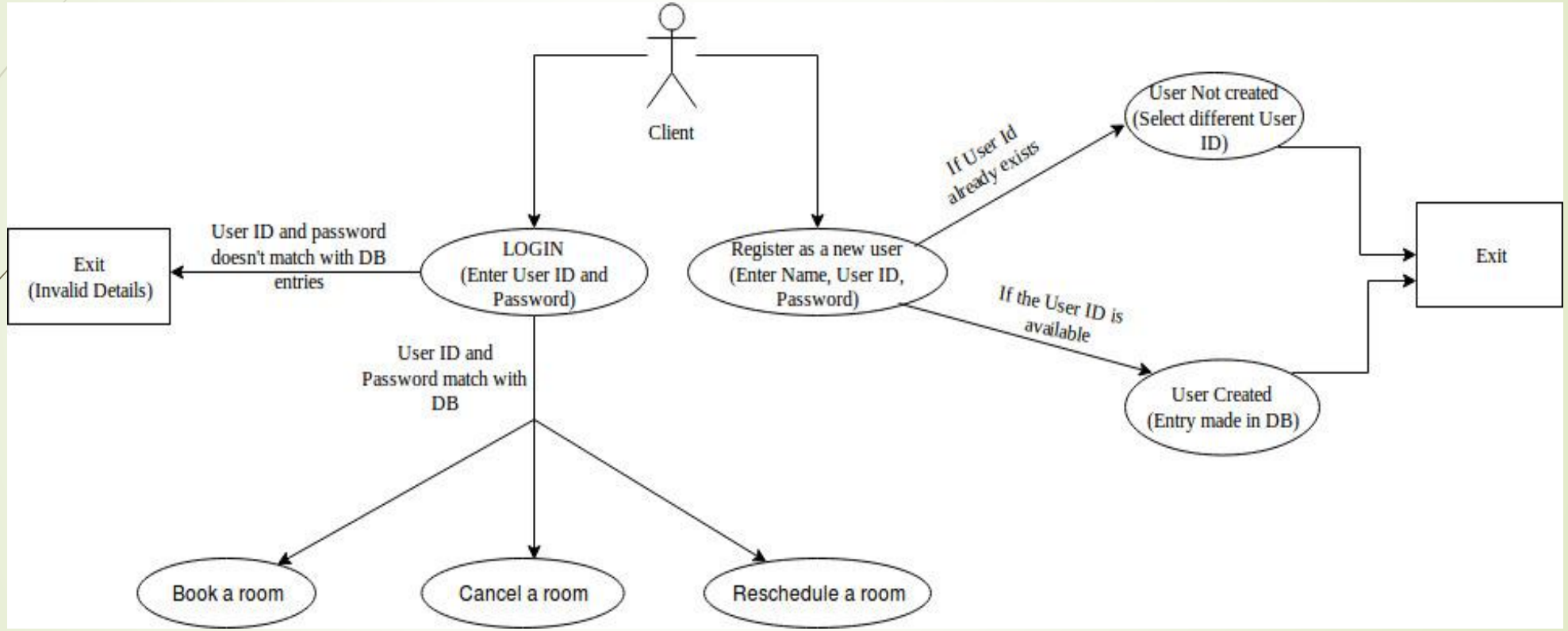
Bhagyesh Patil (16305R003), Ayush Goyal (16305R011)

System Architecture

- Multi-threaded.
- Front-end server:
  - Clients interact with it.
  - Processes their requests.
  - Interact with backend server.
- Back-end server:
  - Stores the information about users and rooms.



Flow Diagram



Phase 1: Constructing the system

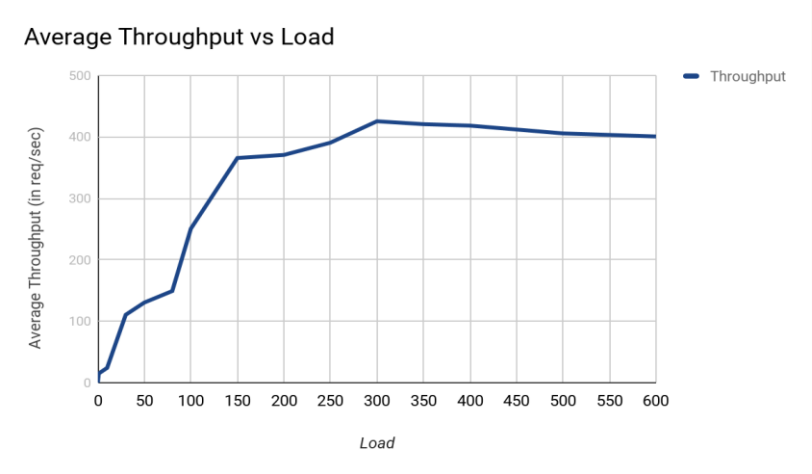
- Database used: MySQL.
- Types of requests: Register, Login, Book, Cancel, Reschedule.
- Critical Sections: Booking, rescheduling.
- User authentication required to enter the system.
- Booking and rescheduling successful if room is available on required date.
- Date check.
- Unique username.
- Room number given at the time of booking is used for Canceling and Rescheduling.

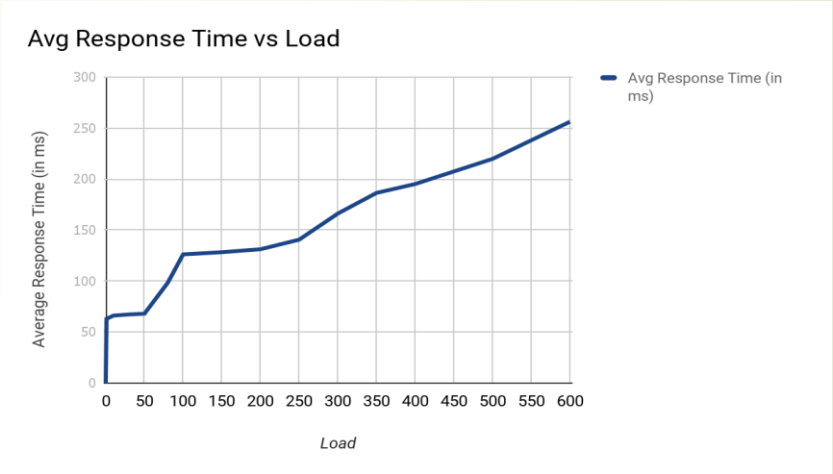
Phase 2: Load Testing

- Closed Loop testing. No think time between requests.
- Increased the load on the server by increasing the number of concurrent users.
- The load generator can be used to perform load testing of:
  - Book requests only.
  - Mixture of Book and Cancel requests.
- Identified bottleneck resource using htop and system monitor.
- **Changed the configuration file of the MySQL.**
- **Reduced the unnecessary locking of some instructions in the critical section.**
- Bottleneck: CPU consumption by MySQL.
- The system hits saturation when the number of users are 300 (for both requests).
- The capacity of the system when it hits bottleneck is around 415 requests/sec (for both requests).

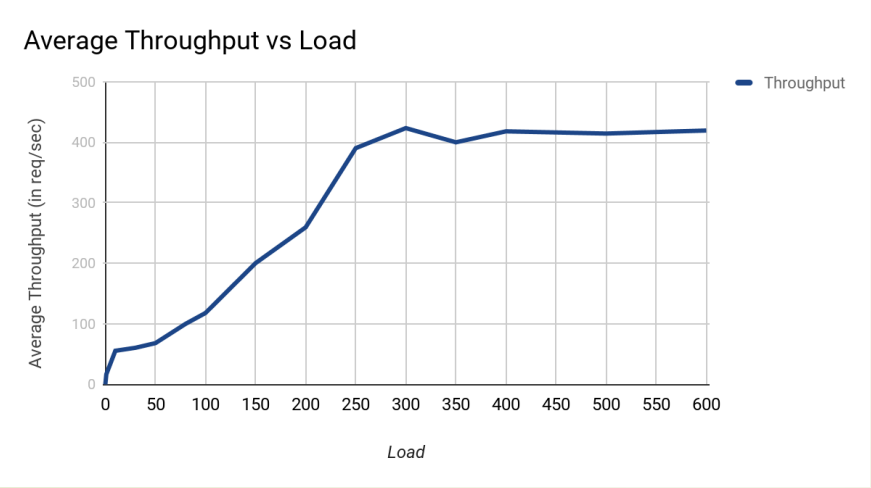
Obtained Graphs in Phase 2

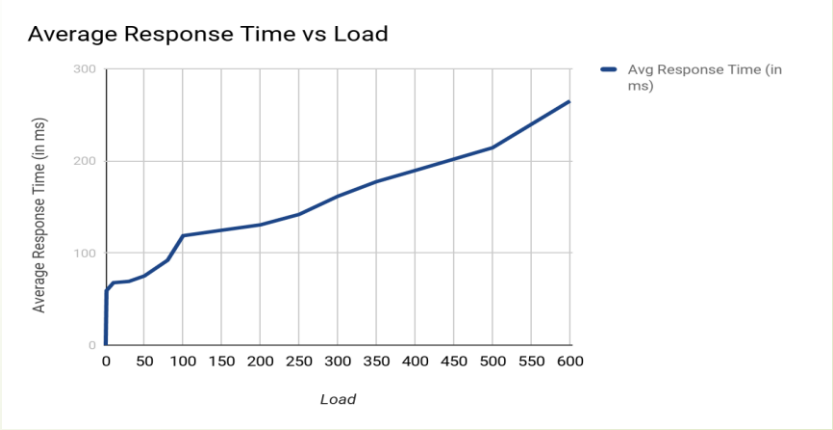
- Book Request Only: (Run time of experiment = 4 minutes)





- Mixture of Book (70%) and Cancel Requests(30%):



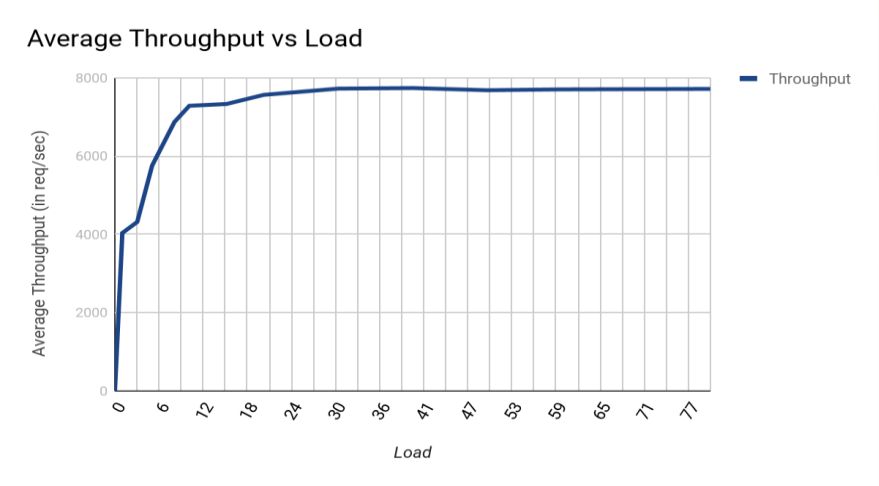


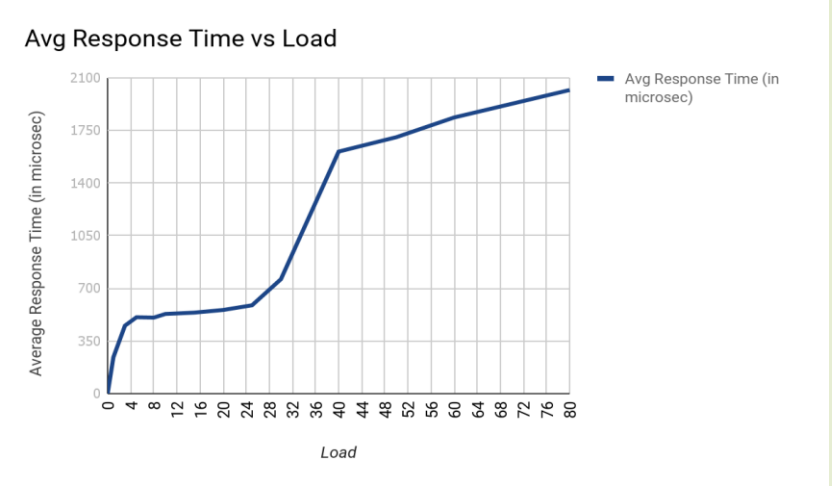
Phase 3: Optimizing System Performance

- First approach:
  - Moved the MySQL database to In-memory.
  - Increased the performance (less as compared to Redis).
- Second approach:
  - Used **Redis** (In-memory DB, stores in key-value pair).
  - Data read and written to RAM.
  - User requests are fulfilled faster as compared with MySQL, so we get high throughput and low response time.
  - Used redis hashes to store each tuple.
  - Bottleneck: CPU consumption by Front-end server.
  - The system hits saturation when the number of users are 25 (for both requests).
  - The capacity of the system is around 7700 req/sec (for book only) and 9450 req/sec (for mixture of book and cancel).

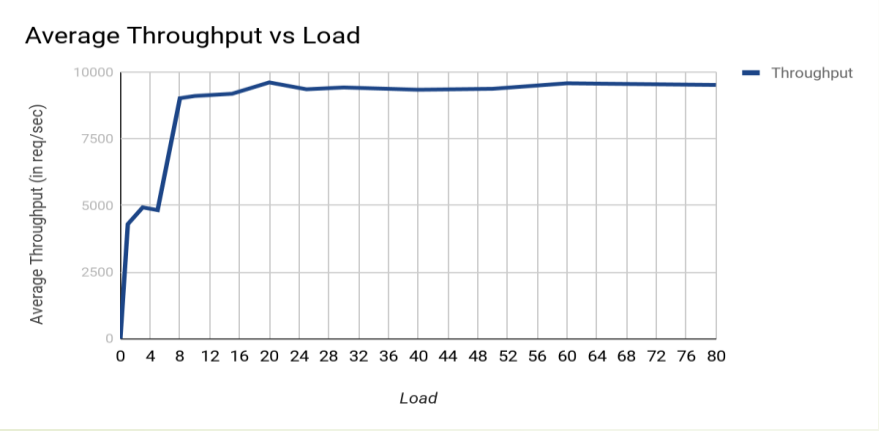
Obtained Graphs in Phase 3

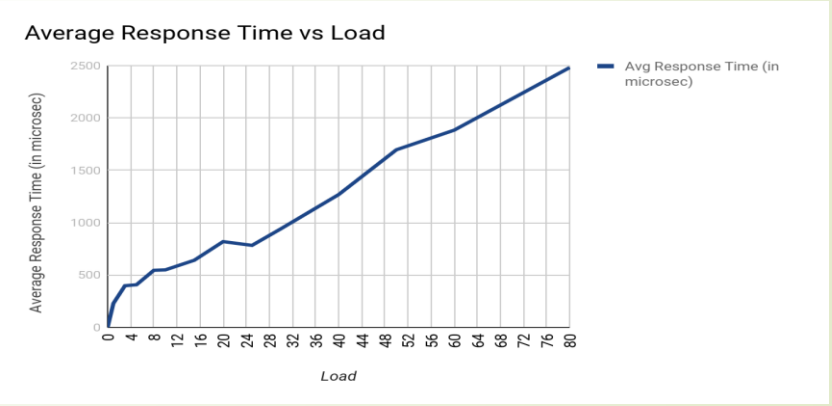
- Book Request Only: (Run time of experiment = 4 minutes)





- Mixture of Book (70%) and Cancel Requests(30%):





Comparison between Phase 2 and 3

- **The throughput is increased drastically in Phase 3**
  - As Redis handles requests directly from RAM which takes less time compared to insertion and removal from disk.
  - RAM is fast for I/O and disks are comparatively slow.
  - System hits saturation with less number of users.
- **The response time is decreased drastically in Phase 3**
  - As the number of requests being fulfilled are increased by a large number as insertion is done in RAM which takes very less time.
  - In case of MySQL response time was hundreds of milliseconds while in redis response time is hundreds of microseconds.
- **Understandings**:
  - Try to use minimal locking as it decreases performance drastically.

Thank You.