CS681 Simulation Project Part-1

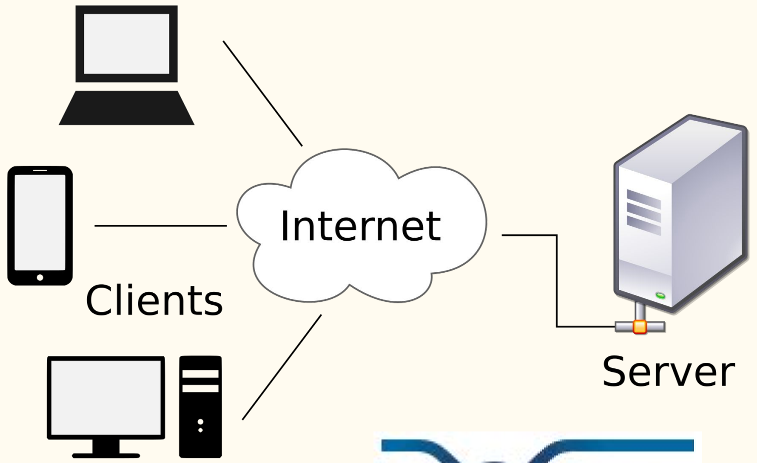
130070009 - Anand Dhoot
13D100032 - Anchit Gupta

The Model

BOINC - The system we are simulating

We consider the model of Volunteer Computing, specifically BOINC - Berkeley Open Infrastructure for Network Computing.

Through this project, we plan to implement various strategies for the server and client side scheduling using a Discrete Event Simulator and evaluate them based on various metrics to present a comparative study of the same.



System Abstraction

We use the following simplified model of an open queuing system -

Server

- Has a buffer of jobs which arrive at it from a poisson stream
- Receives requests from Client to allot a job to them
- Allocates a job, using some algorithm
- Discards jobs which have exceeded their deadline

Client

- Executes the jobs allotted by the server
- Implements its own scheduling algorithm
- Discards jobs which have exceeded their deadline

Key Scheduling Algorithms Implemented

Server (Job Selection)

- FIFO
- Shortest Job First
- Shortest Deadline first

Client

- FIFO
- FIFO - Round Robin
- Shortest Job First
- Shortest Deadline First

Server (Client Preference)

- First Come First Serve
- Fastest Client First
- Emptiest Client First

Assumptions

- Server-Client communication takes negligible time
- Clients are running only our jobs i.e they don’t have any extraneous load.
- Server Job Buffer accepts new jobs only if it has space i.e it doesn’t evict an accepted job in favor of a more “easier” job.
- All jobs have the same reward.
- Client’s always strive to fill all their buffer slots and send appropriate number of requests.
- No migration between clients

Implementation

Code Design (Class Structure, Data Structures)

We have made a C++ based Discrete Event Simulator. We have extensively used OOP (inheritance, virtual functions, etc..) to make the code efficient and easily extensible.

We use three main classes

- Clients
- Events
- Jobs

Key Data Structures used

- List of events for Discrete Event Simulator (Most recent event at head)
- Lists, Arrays for Job Buffers and Request Buffers (at Client/Server)

Execution Flow

In the main loop of our program,

- We handle the earliest event in the event list
- This in turn results in one or more subsequent events being pushed into the event list
- The event handlers in turn use the scheduling algorithms of the clients to schedule the next job

Key Events (& Handlers)

- Arrival of the job at the server - Serviced by adding the job to the job buffer at the server.
- Server Interrupt - Interrupt at the server to allocate some job from the buffer to a available client. This client is chosen by a suitable scheduling algorithm.
- Client Interrupt - This event happens at every Client CPU. This facilitates the job scheduling at individual CPUs, removing completed jobs from the client job buffer and if client job buffer not full, requests the server for more jobs.

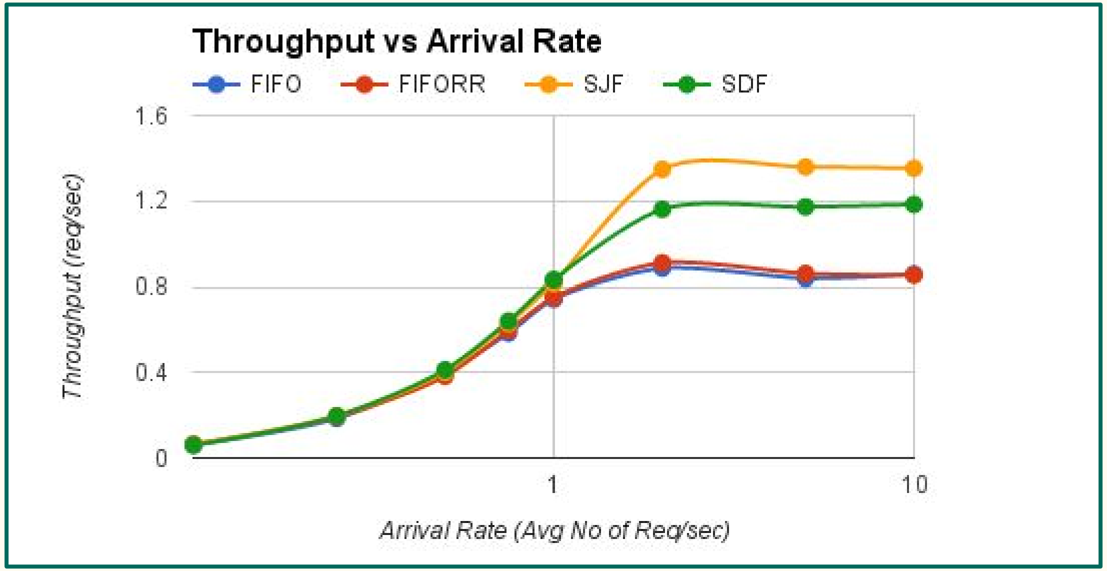
Observations

Throughput vs Arrival Rate (Client Algorithms)

We analyse how the throughput of various client algorithms vary as Arrival Rate increases. Generally, the throughput increases and saturates to system's capacity.

SJF and SDF perform better than the FIFO counterparts for nearly all arrival rates.

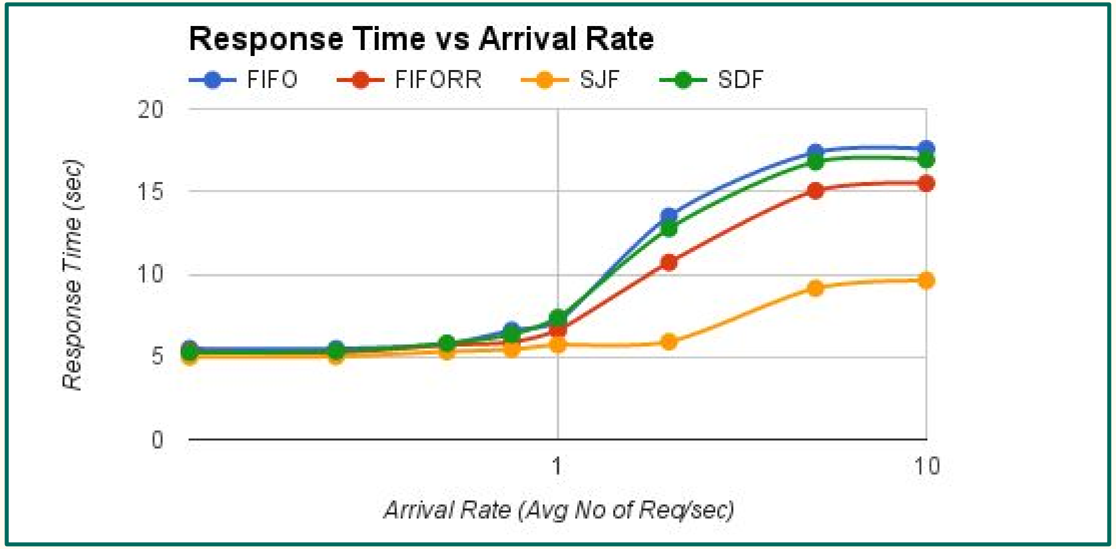
SJF performs the best because it eats up all the small jobs resulting in high throughput



Response Time vs Arrival Rate (Client Algos)

We expect that as Arrival Rate increases, the response time first increases and then saturates due to finite buffers.

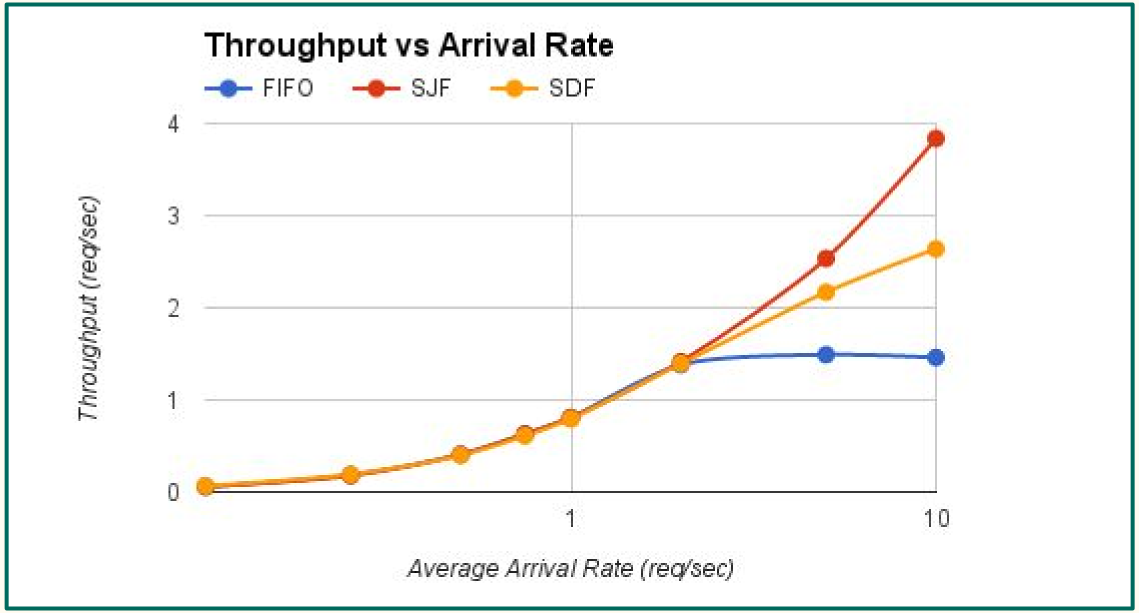
SJF having the highest throughput also has the lowest resp. Time as we measure response time for completed jobs only



Throughput vs Arrival Rate (Server Algorithms)

We analyse how the throughput of various algorithms vary as Arrival Rate increases. Generally, the throughput increases and saturates to system's capacity.

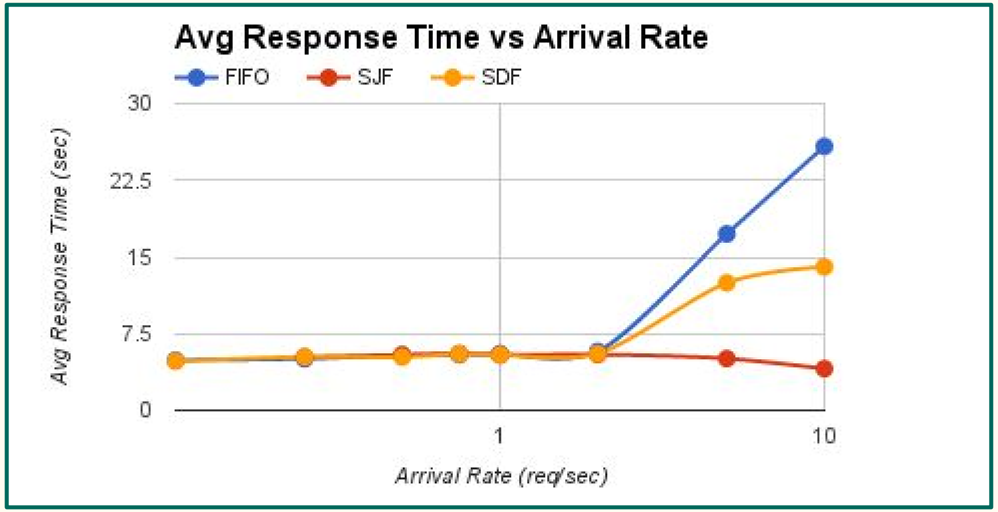
SJF again performs the best the reason being as arrival rate increases it has more number of jobs to chose from and hence more smaller jobs are scheduled



Response Time vs Arrival Rate (Server Algos)

We expect that as Arrival Rate increases, the response time first increases and then saturates due to finite buffer.

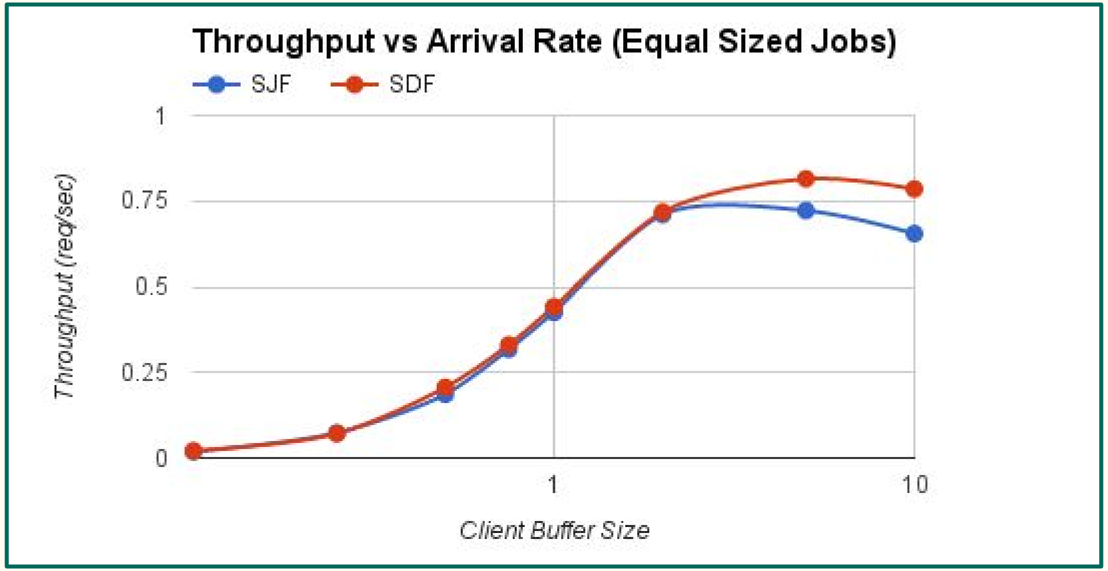
SJF saturates to a much smaller value as we calculate resp time only for completed jobs and SJF does the smaller jobs first.



Throughput vs Arrival Rate (Const Job Size)

We analyse a case where SDF(at client) performs better than the other algorithms.

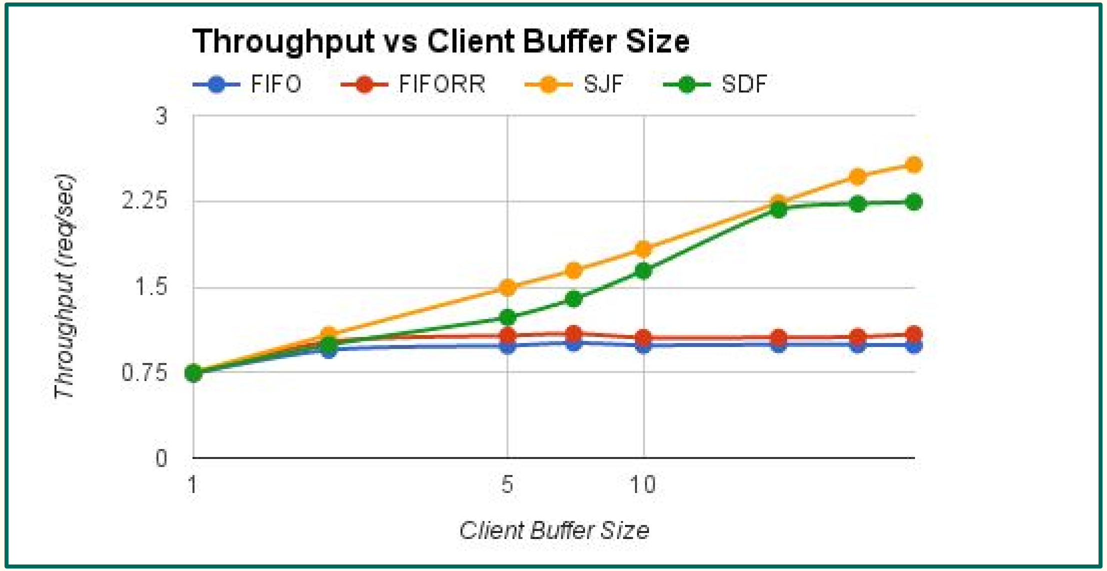
When the job length is const then SJF doesn't give any meaningful ordering of jobs and SDF does slightly better as expected, because now the deadline of a job is much more important than earlier.



Throughput vs Buffer Size

We analyse how the throughput of various algorithms vary as the client buffer size varies.

FIFO, FIFORR show little variation and both SJF, SDF show a good amount of improvement.

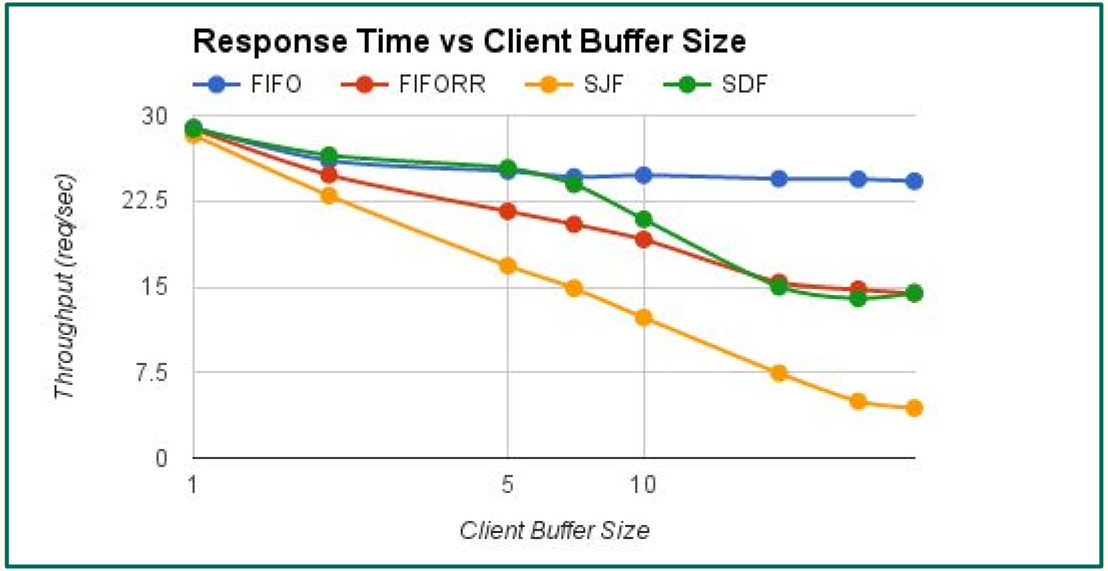


Response Time vs Buffer Size

We analyse how the Average Response Time of a job in the system changes as the client Buffer Size changes with a constant server buffer size.

We expect Response Time to decrease as Client Buffer Size increases as the clients can schedule jobs better, rather than jobs being left in the server buffer.

We see that SJF gives the best performance in this case.



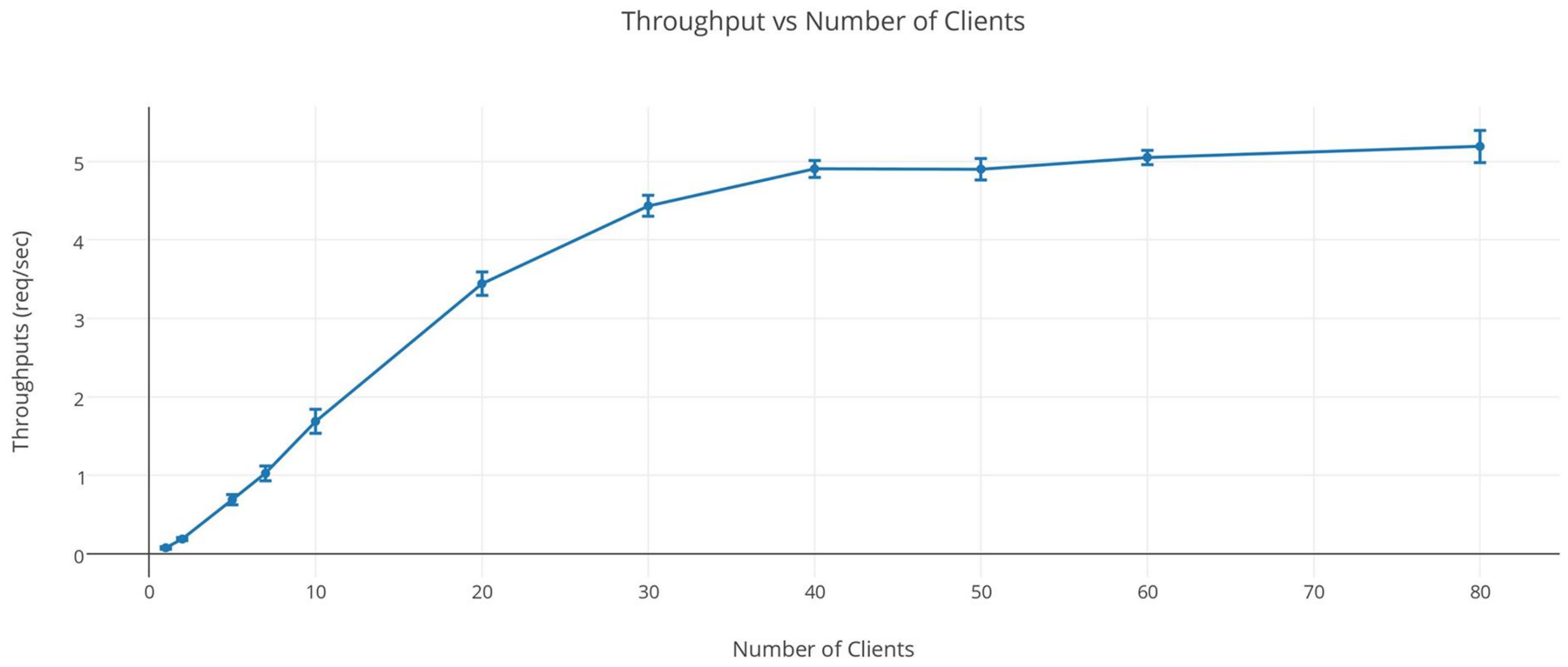
Throughput vs # Clients (99% confidence bars)

We analyse how the Throughput varies as the Number of Clients in the system increases..

We expect Throughput to increase with number of clients before saturating.

The confidence Intervals are shown as error bars for each value

Throughput vs # Clients (99% confidence bars)



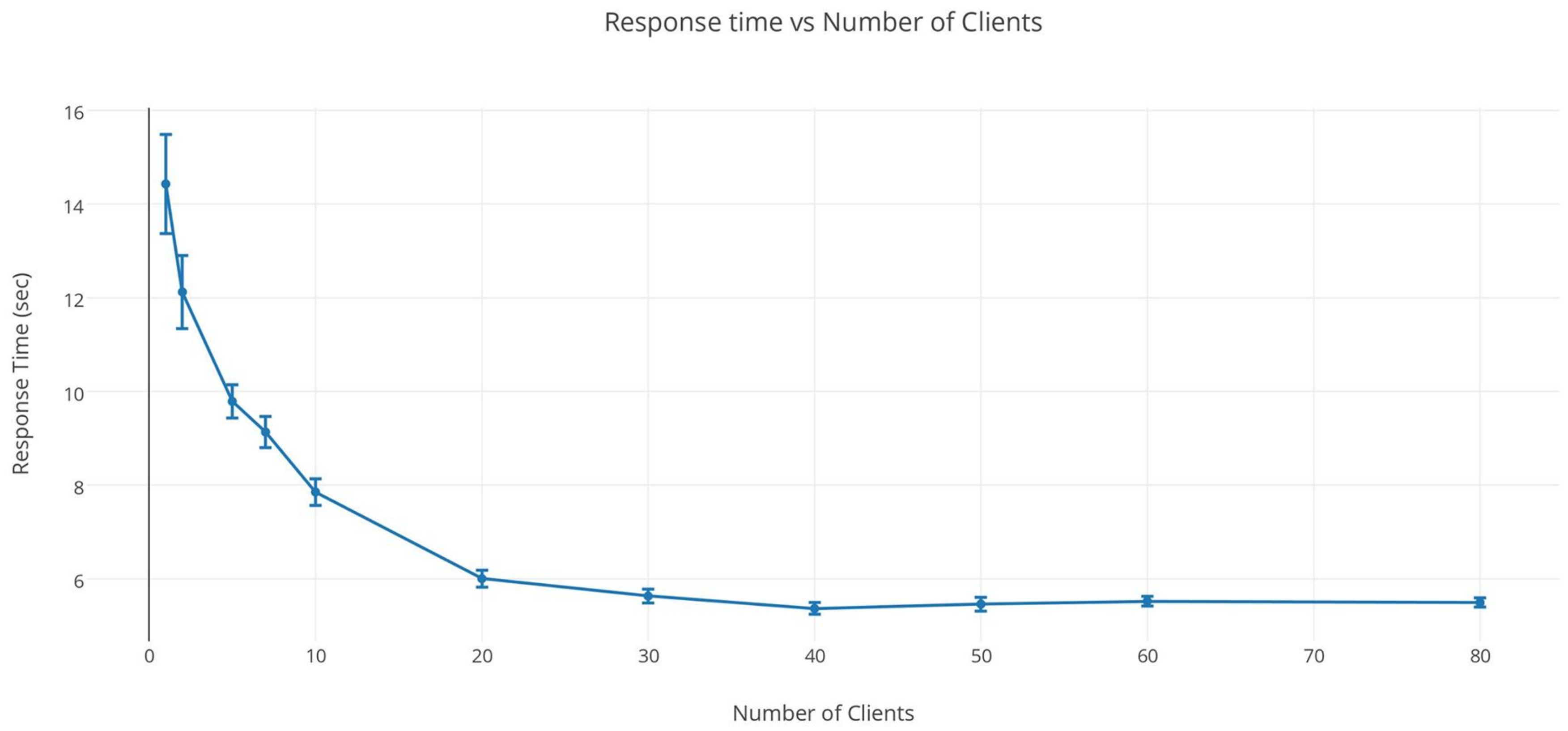
Resp. time vs # Clients (95% confidence bars)

We analyse how the Response Time varies as the Number of Clients in the system increases..

Response Time is expected to decrease as number of clients increases and subsequently saturates.

The confidence Intervals are shown as error bars for each value

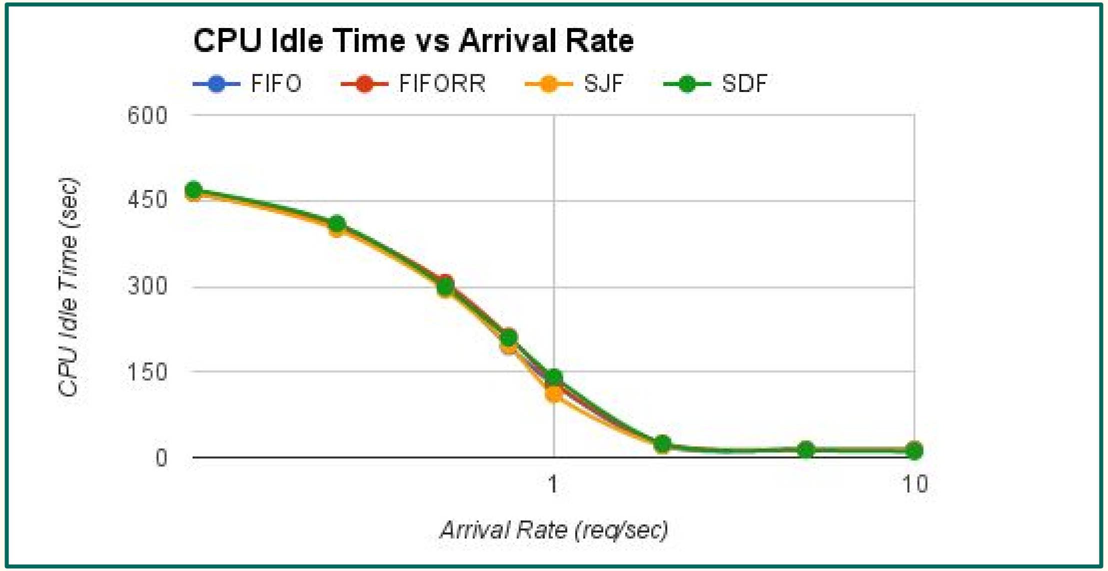
Resp. time vs # Clients (95% confidence bars)



CPU Idle Time vs Arrival Rate

We analyse how the CPU Idle Time varies as Arrival Rate increases. We expect to see Idle time decreasing as there are more and more jobs in the system.

The trend is as expected, with all four Client Scheduling algorithms behaving in a similar fashion.



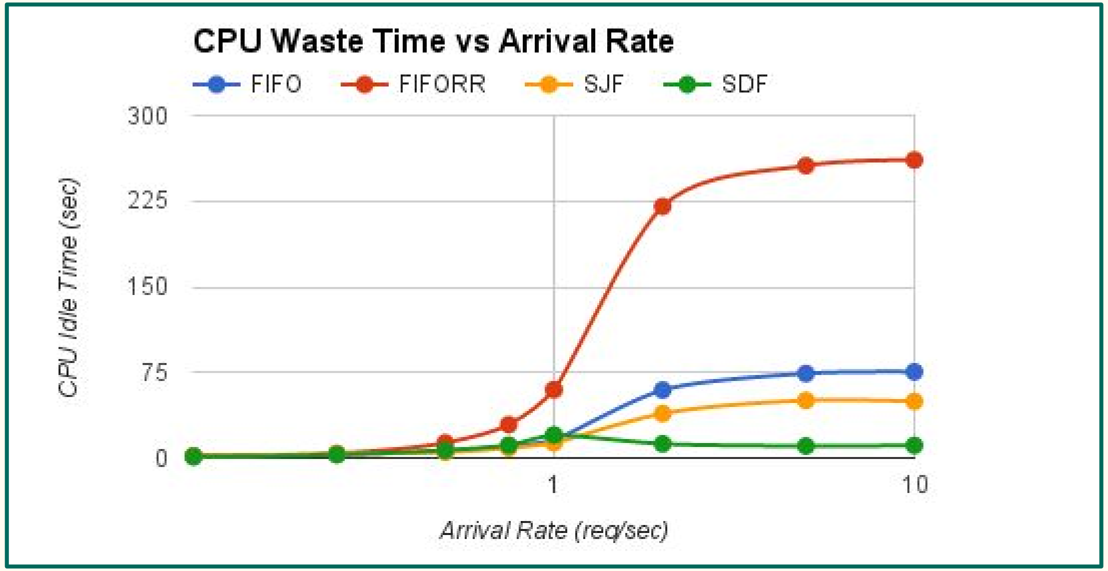
CPU Waste Time vs Arrival Rate

We analyse how much time the CPU spends on jobs that eventually end up exceeding their deadlines.

We expect this to be highest for FIFORR and FIFO algorithms.

SDF should give the least CPU Waste Time and SJF should perform better than FIFO, but worse than SDF.

This shows SDF is the most “efficient” algorithm from the client perspective



Conclusions

Scheduling Algorithms

- SJF performs better(both at server,client) when there is a decent amount of variability in the job lengths
- SDF performs slightly better with constant job lengths.
- SDF has the lowest wasted CPU cycles and hence is more resource efficient from the client perspective.
- SDF/SJF benefit from bigger buffer sizes
- Besides this our implementation also verifies the general trends of throughput/response time vs arrival rate