Jake Shapiro, Martin Glusker, Ned Searls, Matai Curzon
COSC-223 Probability and Computing
May 24, 2021

Investigating the Efficiency of Various Queuing Policies

Introduction

Our project examined how different queueing policies impact the efficiency of the queueing system as measured by mean response time. This problem is significant because queues are essential to the function of many systems in both our physical and digital environments. For example, what is the best way to serve customers in a coffee shop efficiently? Or perhaps more relevant to computer science, which jobs should a server execute first? We sought to understand what policies one might want to implement in a café or web server in order to get the best results. We focused primarily on two outcome variables: mean response time and the average jobs in the queue at any given time as our performance metrics, seeking to minimize both. We attempted to solve this problem by simulating a system that intakes jobs, places them in a queue, and then processes them according to the queuing policy we are testing. We tested several policies including: first-come-first-serve, last-come-last-serve, random, shortest first, and largest first. Consistent with our theoretical expectations of a greedy choice paradigm from the interval scheduling problem, we find that a shortest job first policy leads to the lowest mean response time.

Methods

We constructed an event-driven server simulator using the pseudo-code provided in class. Our simulator consists of two classes: Job and Server. Instances of our Job class each have immutable arrivalTime and serviceTime. These jobs are processed through the server, and we are able to calculate serviceTime based on its times of departure and arrival. The Server class

simulates a server and processes our Jobs. Our Server class has a linked list of Jobs which represent the queue and a linked list of Jobs that have been completed. Each instance of a Server also has a queuing policy that dictates how it selects the next Job to process. Finally, the Server class has variables for p and q to determine arrival rate and service times, it tracks the current job, and it keeps track of the time.

When our Server runs its simulation, it follows this pattern of advancing time and processing jobs:

```
Simulate:
    while (jobs completed < n)
        nextEvent = sooner of next arrival and next departure

        if (nextEvent is departure):
            advance to next departure time
            remove the completed job and update its variables
            put new job into service and update next departure time

        if (nextEvent is arrival):
            advance to next arrival time
            add new Job to queue according to policy
            update next arrival time
```

To perform our experiment, we created five queuing policies that our servers could follow. The five policies were *First-Come-First-Serve* (FCFS), *Last-Come-First-Serve* (LCFS), *Random* (R), *Shortest First* (SF), and *Longest First* (LF). The policies are described in the table below:

Policy Name	Description
First-Come-First-Serve	Add Jobs to the end of the linked list and remove from the front
Last-Come-First-Serve	Add Jobs to the front of the linked list and remove from the front
Random	Add Jobs at a random index in the linked list and remove from the front
Shortest First	Add Jobs to the linked list such that the shortest is at the front and they increase in size towards the back. Remove from the front
Longest First	Add Jobs to the linked list such that the longest is at the front and they decrease in size towards the back. Remove from the front

Arrival rates of jobs followed a geometric distribution, and we varied the parameter as an independent variable in our experiments. Jobs' service time also followed a geometric distribution with a parameter of .5.

In our experiment, for each server policy, we simulated 300,000 jobs for p values between .01 and .5, where p is the parameter of the geometric distribution that determines arrival rate. For each job in each trial, we tracked the service time. After running each trial, we calculated the mean response time by dividing the total service time by the number of jobs. Below, we present the mean response times for jobs in our simulate for each policy for a range of arrival rates determined by p values between .01 and .5.

Finally, we computed the variance of response times for jobs in servers for each policy with both p and q fixed at .5. We simulated 300,000 jobs on each server and recorded the response time for each. We then computed the variance of the response time for the 300,000 jobs in each server.

Results and Analysis

Response Time

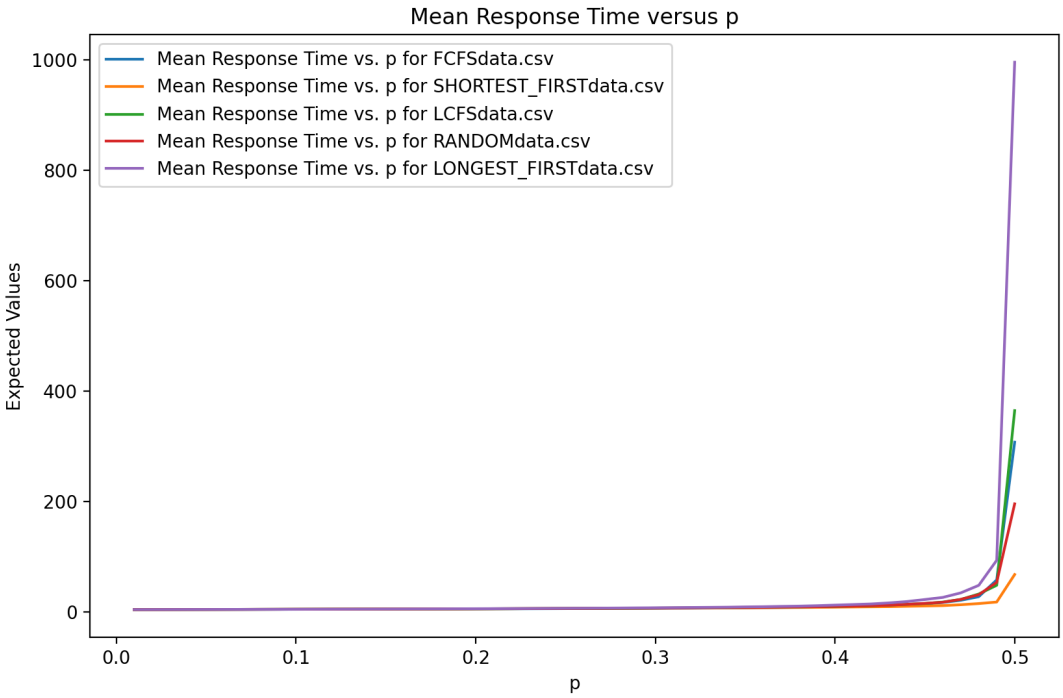
After running the simulations and graphing mean response time and mean number of jobs in the system at each point, we were able to compare the efficiency of each of our queuing policies.

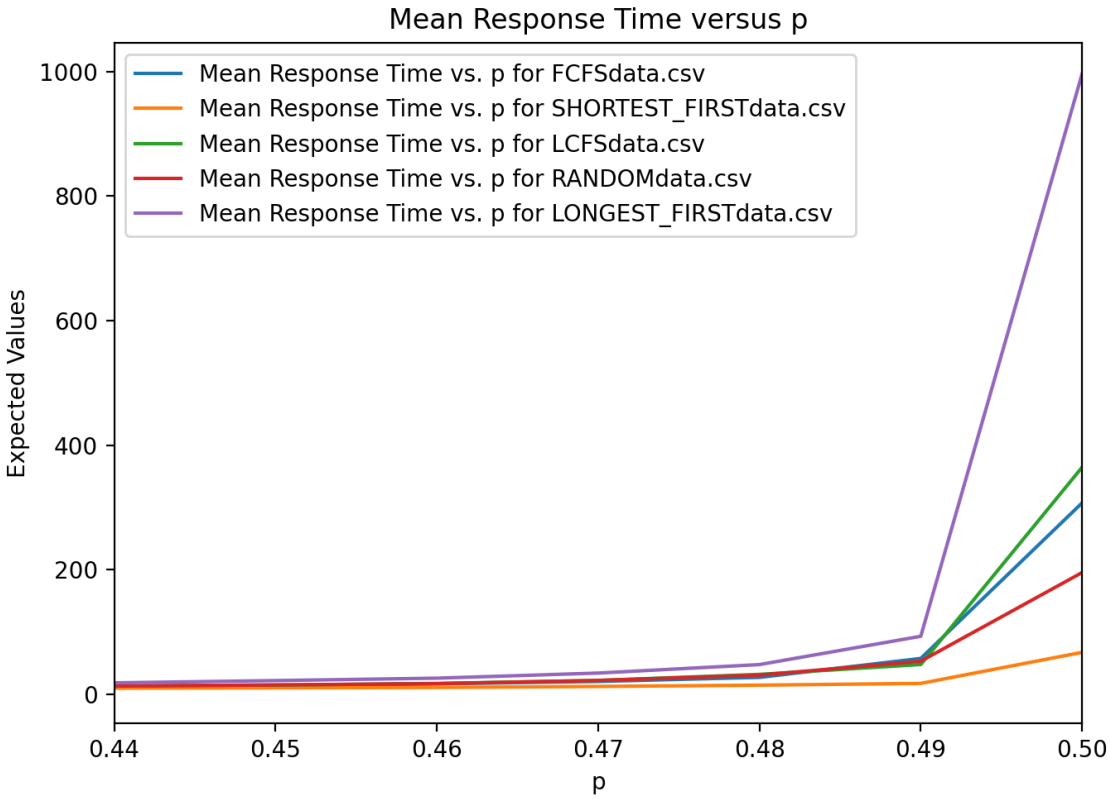
The five queuing systems are divided into three distinct levels of relative efficiency, with SF being the most efficient. Next were R, FCFS, and LCFS and LF was the least efficient queuing policy.

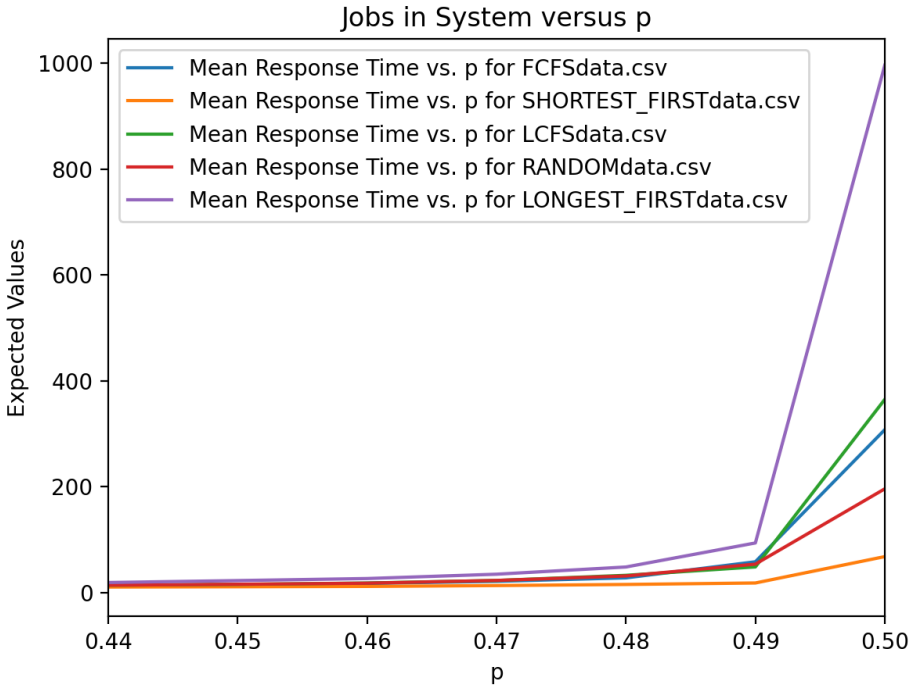
The difference in efficiency was most observable when p got to values closest to .5. This makes sense because when jobs arrive more frequently, they are more likely to spend time in the queue. This is what makes the efficiency of the queue relevant. If the jobs never have to wait, then the way the queue is structured is irrelevant.

While mean response time for the policies was similar with p values up to .47, by the time they reached .5, the difference between the best system, SF, and the worst, LF was 929 time units (996 versus 67). The other three policies ranged from 195 to 364.

The number of jobs in the system followed similar patterns. Again, it makes sense that the differences only begin to appear when jobs are forced to spend more time in the queue. The data are graphed below:


Mean Response Time versus p
Mean Response Time vs. p for FCFSdata.csv
Mean Response Time vs. p for SHORTEST_FIRSTdata.csv
Mean Response Time vs. p for LCFSdata.csv
Mean Response Time vs. p for RANDOMdata.csv
Mean Response Time vs. p for LONGEST_FIRSTdata.csv
Expected Values
p
0
200
400
600
800
1000
0.0
0.1
0.2
0.3
0.4
0.5


Mean Response Time versus p
Mean Response Time vs. p for FCFSdata.csv
Mean Response Time vs. p for SHORTEST_FIRSTdata.csv
Mean Response Time vs. p for LCFSdata.csv
Mean Response Time vs. p for RANDOMdata.csv
Mean Response Time vs. p for LONGEST_FIRSTdata.csv
Expected Values
p
0
200
400
600
800
1000
0.44
0.45
0.46
0.47
0.48
0.49
0.50



First, we found LCFS, FCFS, and R to perform nearly identically. The reason for this is that they are all essentially randomized queuing policies. Because the service times of jobs are independent of when they arrive, any queuing system based on arrival time will not perform better than random. Arrival time provides us no information we can base our queuing decisions on, LCFS and FCFS perform no better than random.

The most efficient queuing system that minimized mean response time was SF. This makes sense based on the nature of the problem we are solving. Finding the most efficient queuing policy is a problem that falls into the greedy choice paradigm. An optimal scheduler to minimize response time should make the greedy choice of processing the shortest remaining job. This has the effect of minimizing the response time for each job and results in the lowest overall response time. We want the response times of jobs that we complete to be as small as possible. To that end, we should process the shortest jobs first, and we can ensure that each response

time we add to our list of completed jobs is the minimum that we can add. This is what our policy does by placing the jobs in order of size with the smallest at the front of the queue. We always ensure we are processing the smallest possible job and adding the smallest possible response time to our list of completed jobs.

The worst policy was LF. This is simply the opposite of the best policy, and it achieves the opposite goal of SF. It maximizes response times by always choosing the job that will contribute the longest time to our list of completed jobs.

Overall, the best policy was SF because it follows the greedy choice paradigm for job scheduling. The random policies performed relatively well but not optimally, and LF was the worst because it does exactly the opposite of the best policy. We also found that because the length of the job is independent of all other variables, any system that seeks to optimize response time must evaluate the ordering of the queue based on job length. In this simulation, there is no proxy we can use that might provide information as to the length of a job and its optimal position in a queue. If, for example, there was a relationship between arrival time and job size, we would likely see different results.

Finally, we want to acknowledge that SF provides no guarantees of individual jobs having a small service time. The only metric we measured was mean service time, and there are applications where a server would want to minimize individual service times, rather than the mean.

Variability

Lastly, we examined the variance of the response time for jobs with each policy. Our data are shown below (values have been divided by 100,000 to increase readability):

Empirical Variability

FCFS	0.257
LCFS	186.394
RANDOM	167.716
LONGEST_FIRST	230.552
SHORTEST_FIRST	24.98

FCFS has the lowest variance. In this policy, all jobs tend to spend similar amounts in the queue, because each one enters at the end and occupies each spot in the queue before being processed. LCFS, the opposite of that, had high variance because some jobs enter the queue and get processed immediately, whereas others get pushed to the back by additions to the front and spend large amounts of time. Random seems to mimic LCFS. FCFS is faster because we guarantee that a job that arrives before another will be processed first. In the other two, however, it is theoretically possible for the first job to arrive to be the last to be processed. FCFS guarantees that a job will be processed within some reasonable amount of time, but the other systems do not guarantee that.

Finally SF had significantly lower variance that LF. Because we know that their response times are very different, we also calculated the Coefficients of Variance, which were 1,118 and 2,725 for SF and LF, respectively. SF is both more efficient and more consistent than LF as a queuing system.

Conclusion

We analyzed different queueing policies and their relative efficiencies. We hypothesized that of the five policies we tested (First come first served, last come first served, random, shortest job first, longest job first), shortest job first would be the most efficient, and longest job first would be the least efficient. Our results confirmed our hypothesis. Shortest job first was the most efficient policy, and longest job first was the least efficient. The remaining three policies all performed similarly between those two. Further, we found that FCFS, LCFS, and random policies all behaved almost identically. This makes sense because the job size for each job was independent of all other variables, essentially making FCFS, LCFS, and random perform as a random policy would. Because arrival time had no effect on job size, ordering our queue based on arrival time should not have been any different from random. We believe that because job size is independent of other variables, policies that do not consider job size will perform as if they were random because no other information we have about a job can provide us insight into the response time for that job. We found that in a scenario where we are aware of the job size before it is processed, the most efficient policy of those we considered is the shortest-job-first policy. Finally, we recognize one limitation this finding presents. While we minimize mean response time, this policy makes no guarantees that any individual job has a low response time, so it may not be optimal in all applications. The scope of this paper was to investigate which of our policies was efficient in terms of mean response time, and we concluded that it was the shortest job first policy.