University of Victoria
Faculty of Computer Science
CSC 446
Operations Research: SimulationSimulation
Final Project Report
Term Spring 2023

Performance Study of Load-balancing with Random Choices

Pengfei Li
V00831098
Computer Science
lijasper@uvic.ca
April 11th, 2023

Yulei Jiang
V00019319
Computer Science
jzgzgd@uvic.ca
April 11th, 2023

Zexi Luo
V00929142
Computer Science
zexiluo@uvic.ca
April 11th, 2023

Table of Contents

I. Introduction

This project explores the effects of a first-in, first-out (FIFO) queue on a service-oriented problem where customers arrive randomly for service. The system is simulated using Python.

The objective of this project is to compare the performance of scheduling method RandMin with two baseline methods. The first method is Purely random, in which the load balancer randomly selects a server from a pool of servers and dispatches the incoming customer to the selected server. The second method is Round-robin, in which the load balancer uses a round-robin method to dispatch incoming customers. Finally, in the last part of this report, we analyze the model as a FIFO queue using the Purely random and Round-robin methods to compare their performance in terms of maximum workload, average workload, and average system time, as determined by our simulation results.

II. Simulation Model

We simulated a load balancer to dispatch customers to servers in a pool with a FIFO queue, creating a simple matching model. Each time a customer arrives, the load balancer performs the following three steps:

1. The load balancer randomly selects servers where selected servers have fewer customers than the existing servers' length. (e.g., if there are 10 existing servers, the load balancer will randomly select 8 servers from those 10)
2. The load balancer checks the selected servers' queue lengths.
3. The load balancer dispatches the customer to the server with the shortest queue length.

This project makes the following assumptions when customers arrive for service: all servers' service times follow an exponential distribution of parameter μ, customer arrivals follow the Poisson arrival process with a mean arrival rate of parameter λ, and we ignore the processing time of the load balancer. We use an M/M/c queue.

While running a simulation. The simulation runs with different combinations of x and y, x refers to a system variable and y refers to a specified performance measure. For each combination, run the simulation at least five times with different random seeds. After running the simulation, compare the final point estimation and the confidence interval for each performance metric, with a confidence level equal to 95%(α=0.05).
Figure II.I below shows our primary simulation model. We will adjust the number of servers in future work.

.

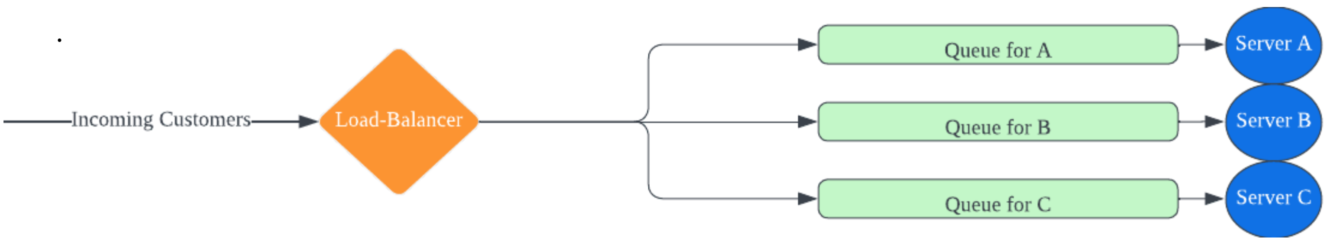

Figure II.I Simulation Diagram

III. Simulation Goal & Parameters

The simulation study aims to compare the effectiveness of the RandMin scheduling method against two baseline methods, round-robin and purely random, by analyzing their respective variance metrics. The primary performance indicators considered in this comparison are maximum workload, average workload, and average system time, with smaller values indicating superior performance. The simulation parameters have been set up in a comprehensive and fair manner to enable a thorough comparison of the three load balancing methods under varying server utilization levels and workload conditions. This approach will help us to identify the strengths and weaknesses of each method and determine their suitability for specific situations.

The simulations were conducted with different combinations of parameter x, which represents the service rate, and parameter y, which represents the number of selected servers.

x (Service Time)	y (Number of Selected Servers)
0.112	2
0.16	2
0.23	2
0.112	7
0.16	7
0.23	7

Table III.I combination of parameters

In the study, we set the capacity of the servers to be 10. This parameter remains constant throughout the study:

Number of servers = 10

Moreover, since the arrival rate and service rate are directly proportional, the arrival rate will also remain constant throughout the study:

Arrival rate = 1.6

During the simulation, we can obtain different sets of random numbers, which allows us to reduce bias and increase the reliability of the results. We are generating five different sets of random numbers:

Seed = 10, 20, 30, 40, 50

For each different seed number, the load balancer will randomly choose two different numbers of servers from 10 servers.

Number of chosen server = 2, 7

To investigate the impact on system performance, we will simulate scenarios with different numbers of customers where 100 arrived customers and 1000 arrived customers. Due to the time constraints, surveying 1000 customers is deemed as sufficiently large sample size for this study, larger customers could not be conducted.

Number of customers = 100, 1000

For this simulation, the queue lengths were configured to be unlimited.

To assist with the paper, the following formulas have been provided. These formulas help to calculate sample variance:

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(Y_i - \bar{Y})^2$$

Confidence interval (with $(1-\alpha)$-confident):

$$\bar{Y} \pm t_{\alpha/2,n-1}\frac{S}{\sqrt{n}}.$$

For evaluating absolute performance, we use the following equations:

$$\bar{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i.$$

$$V(\bar{Y}) = \frac{\sigma^2}{n}[1 + 2\sum_{k=1}^{n-1}(1 - \frac{k}{n})\rho_k].$$

IV. Methodology

IV.I How is your simulation built?

Our simulation is built using Python and the SimPy library, which is a popular library for modeling discrete event simulation systems. The main flow of the simulation can be divided into the following components:

i. Initialization
ii. Customer generation
iii. Load balancing strategies
iv. Customer processing
v. Performance evaluation

To provide a comprehensive understanding, the following is a detailed explanation of each component that has been included in the figure below.

i. Initialization:

The simulation starts by initializing the main parameters, such as the number of servers, the number of customers, the arrival rate of customers, the service rate of servers, and the load balancing strategies to be tested. The simulation also sets up an instance of the SimPy environment, which is responsible for managing the simulation time and scheduling events.

ii. Customer generation:

Customers are generated using a generator function that produces customers according to an exponential distribution, which is a common way to model random arrivals in queuing systems. Each customer is represented by an instance of the Customer class, which stores information about the customer's arrival time, departure time, and service time. The generator function schedules the arrival of each customer as an event in the SimPy environment using the env.timeout() function.

iii. Load balancing strategies:

The load balancer is represented by the LoadBalancer class, which has a method called choose_server() that implements different load balancing strategies. The strategies include:

- Round-robin (RR): The load balancer selects the next available server in a circular order.
- Purely random (PureRand): The load balancer selects a server at random from the pool of servers.
- Random min (RandMin): The load balancer selects a random sample of servers and picks the one with the minimum workload.

```python
def choose_server(self):
    if self.strategy == "RandMin":
        selected_servers = random.sample(self.servers, self.num_sample_servers)
        return min(selected_servers, key=lambda x: x.count)
    if self.strategy == "PureRand":
        return random.choice(self.servers)
    elif self.strategy == "RR":
        server = self.servers[self.current_server]
        self.current_server = (self.current_server + 1) % len(self.servers)
        return server
```

iv. Customer processing:

When a customer arrives, the load balancer selects a server using the chosen strategy and processes the customer. This is done by requesting a resource (server) in the SimPy environment and waiting for it to become available. Once the server is available, the load balancer schedules the service time of the customer using the env.timeout() function and updates the customer's departure time accordingly. The customer processing is modeled as a SimPy process, which allows multiple customers to be processed concurrently in the simulation.

```
def process_customer(self, customer):
    server = self.choose_server()
    with server.request() as req:
        self.customer_arrives(customer)
        yield req
        self.customer_departs(customer)

        service_time = random.expovariate(self.service_rate)
        customer.service_time = service_time
        yield self.env.timeout(service_time)
        customer.set_departure_time(self.env.now)
        self.departure_times.append(customer.departure_time)
        self.processed_customers.append(customer)
```

v. Performance evaluation:

After the simulation is completed, the performance of each load balancing strategy is evaluated by calculating the maximum and average queue lengths and the average system time, which are derived from the arrival and departure times of the processed customers. The results are then printed for comparison.

```
max_queue_length = max(load_balancer.queue_lengths)
avg_queue_length = sum(load_balancer.queue_lengths) / len(load_balancer.queue_lengths)
```

```
def avg_system_time(self):
    system_times = [customer.departure_time - customer.arrival_time for customer in self.processed_customers]
    avg_system_time = sum(system_times) / len(system_times) if system_times else 0
    return avg_system_time
```

Our simulation follows a straightforward flow that begins with initializing the parameters and the SimPy environment, generates customers based on an exponential distribution, processes customers using different load balancing strategies, and finally evaluates the performance of each strategy. The simulation is modular and easily extendable to include more load balancing strategies, additional performance metrics, or different customer arrival patterns.

IV.II How did you set up the simulation parameters?

To evaluate the performance of the RandMin algorithm against RR and PreRand under different utilization rates and customer volumes, we can compare the results for both high and low utilization levels.

IV. Analysis

We collected data from Round Robin(RR), Purely Random(PreRand), and RandMin, with the variable 'y' denoting 'd', where 'd' refers to randomly selected servers.

Table IV.I Service rate = 0.112, random selected servers = 2

RR						
X= 0.112 y = 2	100 Customers			1000 Customers		
Seed	Max queue length	Average queue length	Average system time	Max queue length	Average queue length	Average system time
10	33	18.35	28.62857	262	133.05	127.361
20	26	13.24	19.66523	364	186.15	185.32377
30	39	18.24	26.14616	346	173.72	177.63352
40	28	13.84	19.25827	277	138.39	135.0504
50	33	18.41	26.58187	337	173.4	173.65575
Mean:	31.8	16.416	24.05602	317.2	160.942	159.804888
Var:	4.53431362	2.3565279	3.845887643	40.18656	21.1682719	23.7752408
Confidence +/-	1.866460773	1.3455485	1.718942642	5.556528	4.03279171	4.27391176

PurRand						
X= 0.112 y = 2	100 Customers			1000 Customers		
Seed	Max queue length	Average queue length	Average system time	Max queue length	Average queue length	Average system time
10	35	19.77	34.21503	345	175.12	181.14294
20	40	21.23	38.00202	334	171.4	180.73315
30	35	17.83	30.4538	326	160.95	167.23919
40	35	16.58	30.20489	356	188.84	187.55488
50	34	17.06	29.06274	303	136.03	143.7172
Mean:	35.8	18.494	32.387696	332.8	166.468	172.077472
Var:	2.135415	1.747828367	3.299082045	18.01554	17.644309	15.64843526
Confidence +/-	1.280867	1.158810658	1.592060793	3.720376	3.6818443	3.467356912

RandMin						
X= 0.112 y = 2	100 Customers			1000 Customers		
Seed	Max queue length	Average queue length	Average system time	Max queue length	Average queue length	Average system time
10	33	16.26	25.81678	311	169.09	159.88124
20	39	17.93	30.82326	271	143.02	141.86391
30	36	15.56	27.05298	335	149.73	152.9531
40	23	9.1	16.86092	312	158.33	159.74392
50	42	16.68	30.23509	304	146.13	147.84226
Mean:	34.6	15.106	26.157806	306.6	153.26	152.456886
Var:	6.5299310	3.100397394	5.0147867	20.6358	9.4281408	6.95663767
Confidence +/-	2.2398420	1.5433761	1.9628599	3.98175	2.6913868	2.31186665

Table IV.II Service rate = 0.16, random selected servers = 2

RR						
X= 0.16 y = 2	100 Customers			1000 Customers		
Seed	Max queue length	Average queue length	Average system time	Max queue length	Average queue length	Average system time
10	18	9.19	15.46541	90	47.05	37.61205
20	28	15.32	16.93548	149	90.39	67.4018
30	31	14.02	15.53898	102	54.12	44.07721
40	18	7.76	10.58513	108	63.92	49.37391
50	23	11.05	14.91527	103	54.31	44.20443
Mean:	23.6	11.468	14.688054	110.4	61.958	48.53388
Var:	5.238320341	2.844780	2.1570764	20.1851	15.19475	10.144914
Confidence +/-	2.006130201	1.478384	1.2873476	3.93802	3.416724	2.7918191

PurRand						
X= 0.16 y = 2	100 Customers			1000 Customers		
Seed	Max queue length	Average queue length	Average system time	Max queue length	Average queue length	Average system time
10	32	16.81	21.36479	121	63.8	50.44232
20	27	17.07	20.77232	143	79.61	63.06443
30	19	9.29	13.68635	85	54.28	43.15902
40	25	11.07	14.61696	121	76.34	57.62854
50	28	14.58	21.82287	111	55.08	43.01051
Mean:	26.2	13.764	18.452658	116.2	65.822	51.460964
Var:	4.26145	3.10315710	3.53977243	18.7872	10.52078	7.92526970
Confidence +/-	1.80943	1.54406283	1.649114313	3.79922	2.843066	2.46757359

RandMin						
X= 0.16 y = 2	100 Customers			1000 Customers		
Seed	Max queue length	Average queue length	Average system time	Max queue length	Average queue length	Average system time
10	12	6.35	11.26491	80	52.19	40.09723
20	21	9.05	13.43538	61	30.53	27.24746
30	26	10.86	18.30967	84	49.28	39.04632
40	22	10.99	14.94285	101	56.38	43.62625
50	29	10.68	14.69183	87	51	42.63806
Mean:	22	9.586	14.528928	82.6	47.876	38.531064
Var:	5.7619441	1.763526013	2.2952794	12.9089	8.983795	5.87990510
Confidence +/-	2.1040090	1.164002798	1.3279474	3.14925	2.627199	2.12543705

Table IV.III Service rate = 0.23, random selected servers = 2

RR						
X= 0.23 y=2	100 Customers			1000 Customers		
Seed	Max queue length	Average queue length	Average system time	Max queue length	Average queue length	Average system time
10	12	4.39	6.43492	17	5.78	7.45898
20	14	7.24	8.76585	21	9.75	9.89009
30	8	4.59	5.94771	16	5.78	7.47252
40	7	3.7	5.88611	16	7.04	8.31751
50	12	4.76	7.16501	26	9.59	10.0052
Mean:	10.6	4.936	6.83992	19.2	7.588	8.62886
Var:	2.653299832	1.207055	1.0662745	3.86781	1.7618331	1.1214002
Confidence +/-	1.42776343	0.963001	0.9051021	1.72383	1.1634439	0.9282039

PurRand						
X= 0.23 y = 2	100 Customers			1000 Customers		
Seed	Max queue length	Average queue length	Average system time	Max queue length	Average queue length	Average system time
10	19	11.14	12.1794	33	16.73	14.29752
20	17	8.71	12.05209	29	15.91	13.85284
30	15	8.59	9.49676	46	25.74	20.21716
40	19	10.13	10.15575	39	18.15	14.60423
50	9	4.67	7.48818	25	11.42	10.68241
30	15.8	8.648	10.274436	34.4	17.59	14.730832
14.1421	3.70944	2.20173022	1.741897265	7.41889	4.655469	3.08104439
3.29625	1.68817	1.30060416	1.156842826	2.38744	1.891232	1.53855160

RandMin						
X= 0.23 y = 2	100 Customers			1000 Customers		
Seed	Max queue length	Average queue length	Average system time	Max queue length	Average queue length	Average system time
10	10	4.44	7.00643	18	8.13	8.90615
20	6	2.8	5.465617	19	6.24	7.70719
30	8	2.56	4.94511	18	6.48	7.94336
40	10	4.58	7.37438	25	8.04	8.62915
50	7	2.67	6.04613	28	11.66	11.56724
30	8.2	3.41	6.1675334	21.6	8.11	8.950618
14.1421	1.6	0.902441133	0.91235635	4.12795	1.9370906	1.37932676
3.29625	1.10872306	0.832669257	0.83723107	1.78086	1.2199388	1.02942964

Table IV.IV Service rate = 0.112, random selected servers = 7

RR						
X= 0.112 y = 7	100 Customers			1000 Customers		
Seed	Max queue length	Average queue length	Average system time	Max queue length	Average queue length	Average system time
10	33	18.35	28.62857	262	133.05	127.361
20	26	13.24	19.66523	364	186.15	185.32377
30	39	18.24	26.14616	346	173.72	177.63352
40	28	13.84	19.25827	277	138.39	135.0504
50	33	18.41	26.58187	337	173.4	173.65575
Mean:	31.8	16.416	24.05602	317.2	160.942	159.80488
Var:	4.53431362	2.356527	3.8458876	40.1865	21.16827	23.775240
Confidence +/-	1.866460773	1.345548	1.7189426	5.55652	4.032791	4.2739117

PurRand						
X= 0.112 y = 7	100 Customers			1000 Customers		
Seed	Max queue length	Average queue length	Average system time	Max queue length	Average queue length	Average system time
10	35	19.77	34.21503	345	175.12	181.14294
20	40	21.23	38.00202	334	171.4	180.73315
30	35	17.83	30.4538	326	160.95	167.23919
40	35	16.58	30.20489	356	188.84	187.55488
50	34	17.06	29.06274	303	136.03	143.7172
Mean:	35.8	18.494	32.387696	332.8	166.468	172.077472
Var:	2.135415	1.7478283	3.299082045	18.0155	17.64430	15.6484352
Confidence +/-	1.28086	1.15881065	1.592060793	3.72037	3.681844	3.46735691

RandMin						
X= 0.112 y = 7	100 Customers			1000 Customers		
Seed	Max queue length	Average queue length	Average system time	Max queue length	Average queue length	Average system time
10	36	13.15	22.2122	277	126.54	128.57799
20	34	17.82	20.93548	308	144.44	141.6698
30	23	10.83	17.52461	289	135.15	132.92797
40	34	14.34	23.00287	324	156.14	160.10388
50	31	16.9	27.474	353	176.84	187.53748
Mean:	31.6	14.608	22.229832	310.2	147.822	150.163424
Var:	4.5869379	2.531445437	3.2222246	26.7536	17.528117	21.5905822
Confidence +/-	1.8772604	1.394592659	1.5734067	4.53372	3.669701	4.07282046

Table IV.V Service rate = 0.16, random selected servers = 7

RR						
x= 0.16 y = 7	100 Customers			1000 Customers		
Seed	Max queue length	Average queue length	Average system time	Max queue length	Average queue length	Average system time
10	18	9.19	15.46541	90	47.05	37.61205
20	28	15.32	16.93548	149	90.39	67.4018
30	31	14.02	15.53898	102	54.12	44.07721
40	18	7.76	10.58513	108	63.92	49.37391
50	23	11.05	14.91527	103	54.31	44.20443
Mean:	23.6	11.468	14.688054	110.4	61.958	48.53388
Var:	5.238320341	2.844780	2.1570764	20.1851	15.19475	10.144914
Confidence +/-	2.006130201	1.478384	1.2873476	3.93802	3.4167241	2.7918191

PurRand						
x= 0.16 y = 7	100 Customers			1000 Customers		
Seed	Max queue length	Average queue length	Average system time	Max queue length	Average queue length	Average system time
10	32	16.81	21.36479	121	63.8	50.44232
20	27	17.07	20.77232	143	79.61	63.06443
30	19	9.29	13.68635	85	54.28	43.15902
40	25	11.07	14.61696	121	76.34	57.62854
50	28	14.58	21.82287	111	55.08	43.01051
Mean:	26.2	13.764	18.452658	116.2	65.822	51.460964
Var:	4.26145	3.1031571	3.53977243	18.7872	10.52078	7.9252697
Confidence +/-	1.80943	1.54406283	1.64911431	3.79922	2.843066	2.4675735

RandMin						
x= 0.16 y = 7	100 Customers			1000 Customers		
Seed	Max queue length	Average queue length	Average system time	Max queue length	Average queue length	Average system time
10	8	2.75	7.13447	114	51.31	39.77881
20	11	4.68	8.85662	79	32.94	28.36861
30	21	8.23	14.48629	54	17.54	17.26334
40	13	6.13	11.08188	52	29.41	24.76917
50	21	11.12	15.23224	54	25.77	23.87074
Mean:	14.8	6.582	11.3583	70.6	31.394	26.810134
Var:	5.3065999	2.891901796	3.1294498	23.8796	11.19407	7.41029353
Confidence +/-	2.0191624	1.490578524	1.5505903	4.28329	2.932628	2.38605689

Table IV.VI Service rate = 0.23, random selected servers = 7

RR						
x= 0.23 y = 7	100 Customers			1000 Customers		
Seed	Max queue length	Average queue length	Average system time	Max queue length	Average queue length	Average system time
10	12	4.39	6.43492	17	5.78	7.45898
20	14	7.24	8.76585	21	9.75	9.89009
30	8	4.59	5.94771	16	5.78	7.47252
40	7	3.7	5.88611	16	7.04	8.31751
50	12	4.76	7.16501	26	9.59	10.0052
Mean:	10.6	4.936	6.83992	19.2	7.588	8.62886
Var:	2.653299832	1.207055	1.0662745	3.86781	1.7618331	1.1214002
Confidence +/-	1.42776343	0.963001	0.9051021	1.72383	1.1634439	0.9282039

PurRand						
x= 0.23 y = 7	100 Customers			1000 Customers		
Seed	Max queue length	Average queue length	Average system time	Max queue length	Average queue length	Average system time
10	19	11.14	12.1794	33	16.73	14.29752
20	17	8.71	12.05209	29	15.91	13.85284
30	15	8.59	9.49676	46	25.74	20.21716
40	19	10.13	10.15575	39	18.15	14.60423
50	9	4.67	7.48818	25	11.42	10.68241
Mean:	15.8	8.648	10.274436	34.4	17.59	14.730832
Var:	3.70944	2.20173022	1.741897265	7.41889	4.655469	3.08104439
Confidence +/-	1.68817	1.30060416	1.156842826	2.38744	1.891232	1.53855160

RandMin						
x= 0.23 y = 7	100 Customers			1000 Customers		
Seed	Max queue length	Average queue length	Average system time	Max queue length	Average queue length	Average system time
10	4	1.19	3.92834	7	1.62	4.49803
20	3	1.18	3.90253	12	2.01	4.76313
30	3	1.57	4.12492	16	3.09	5.46952
40	1	1	3.62679	13	3.34	5.78714
50	6	1.63	4.75988	14	2.48	5.55563
Mean:	3.4	1.314	4.068492	12.4	2.508	5.21469
Var:	1.62480768	0.243852414	0.38039140	3.00665	0.6431609	0.49525456
Confidence +/-	1.11728526	0.432839233	0.54060298	1.51986	0.7029475	0.61684682

We will use two performance metrics in our analysis with tables above:

Absolute performance measures the actual performance of the system under different conditions, such as time spent in system for customer and change of queue by time, by directly comparing performance indicators, max of queue length, average of queue length and average time spent in system.

Relative performance compares the performance of RandMin relative to the two baselines, RR and PreRand, enabling us to determine which system is the most effective under max of queue length, average of queue length and average time spent in system.

IV.I Absolute performance

In this simulation, we used the max queue length, the average queue length for a single server, and the average time spent in the system for all customers as the performance indicators of our system. While the max queue length provides a single outcome, we do not measure the performance of this parameter because it has no calculated average value.

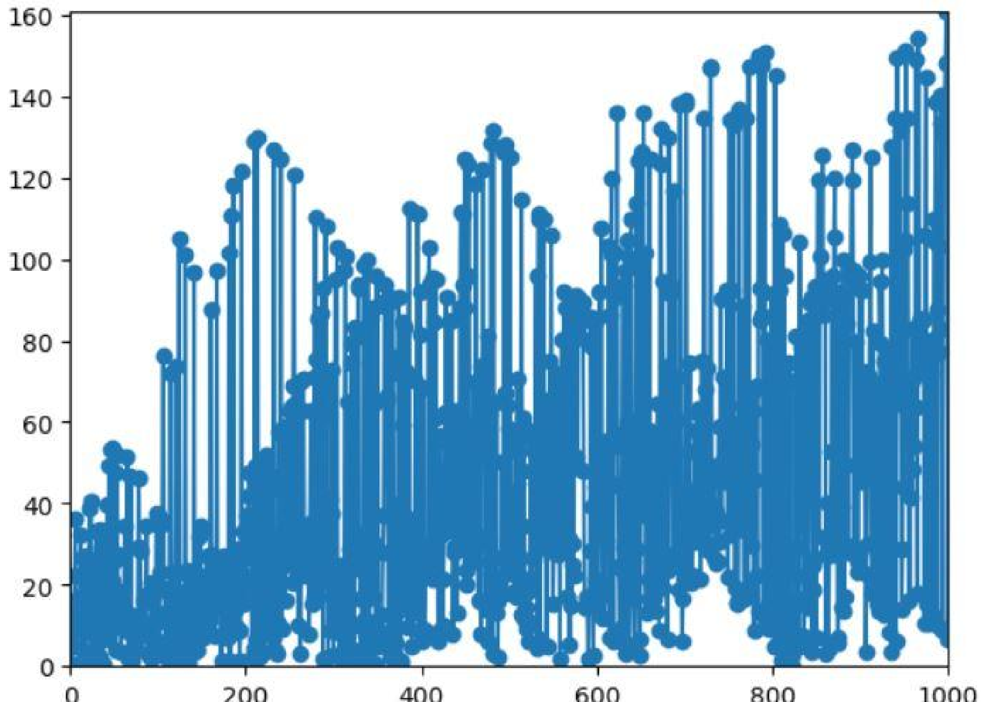

Figure IV.I.I Time spent in system for customers

For the average time spent in the system for all customers, shown in **Figure IV.I.I** above, after 200 customers, the system has an evenly spreaded range. This is because the first 200 customers may be influenced by the first few customers due to there being no waiting time for the first customer. In this case, we considered that customers after 200 are recognized as valid customer data for our system.

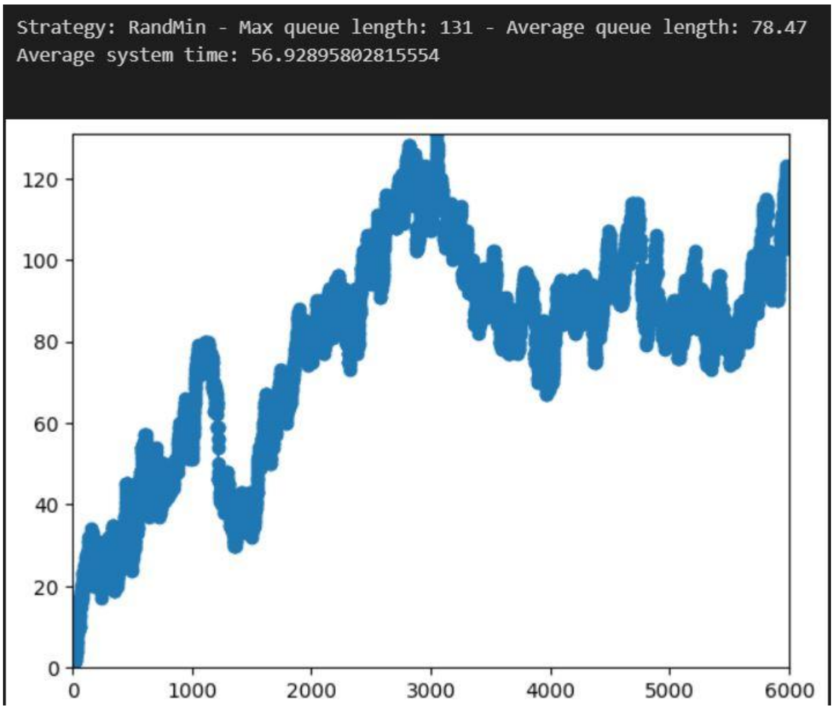

Figure IV.I.II Change of queue length by time

Due to the long warm-up period in our system, the first few thousand customers did not spend a significant amount of time in the queue. As a result, we excluded this unstable data and used the data from the first 1000 customers after 3000 as a reliable source for comparing the relative performance of the system.

As we can see in the tables for RandMin strategies, when server utilization becomes more and more efficient, the max of queue length, average queue length and average time spent in the system are all dropped significantly. When we give seven servers for the random selection, the system performs slightly better than the two server scenario. For the incoming customer, we find the interval overlapped between each other when there are 100 customers. But for 1000 customers, the relative confidence interval becomes smaller. We calculated the overall runs with the following performance data.

Mean: 19.1
Var: 4.23503

IV.II Relative performance

We have utilized the data from the tables presented above to produce the graphs below, which visually demonstrate the performance of each method. These graphs offer a clear representation of the results and enable us to compare the relative performance of RR, PreRand and RandMin across three different performance metrics: Max Queue Length, Average Queue Length and Average System Time.

To provide a better understanding of the figures below, it's important to note that:

- Blue dots represent when utilization = 0.7
- Orange dots represent when utilization = 1
- Green dots represent when utilization = 1.3
- The first dot across all colors corresponds to the Round Robin (RR)
- The second dot across all colors corresponds to the PreRandom(PreRan)
- The third dot across all colors corresponds to the RanMin

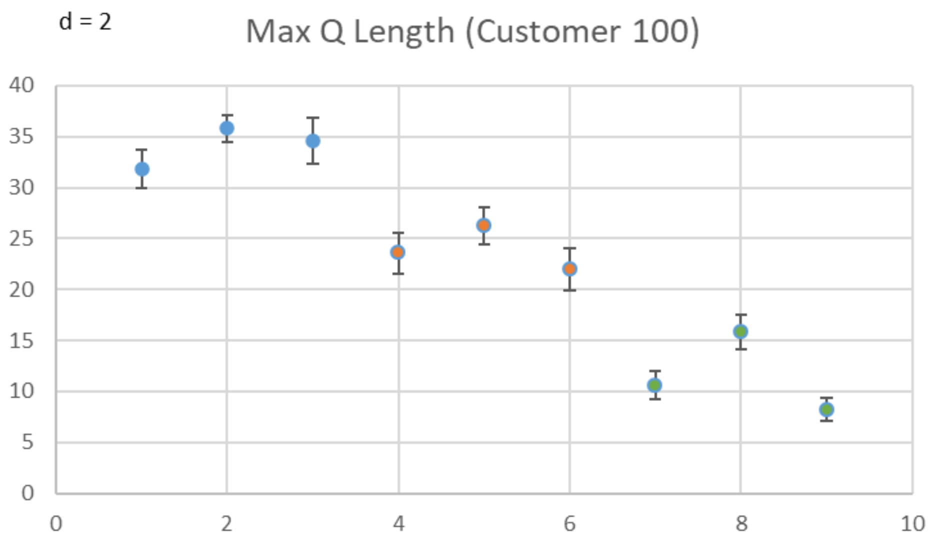

Figure IV.II.I Max Queue Length with 100 customers where d = 2

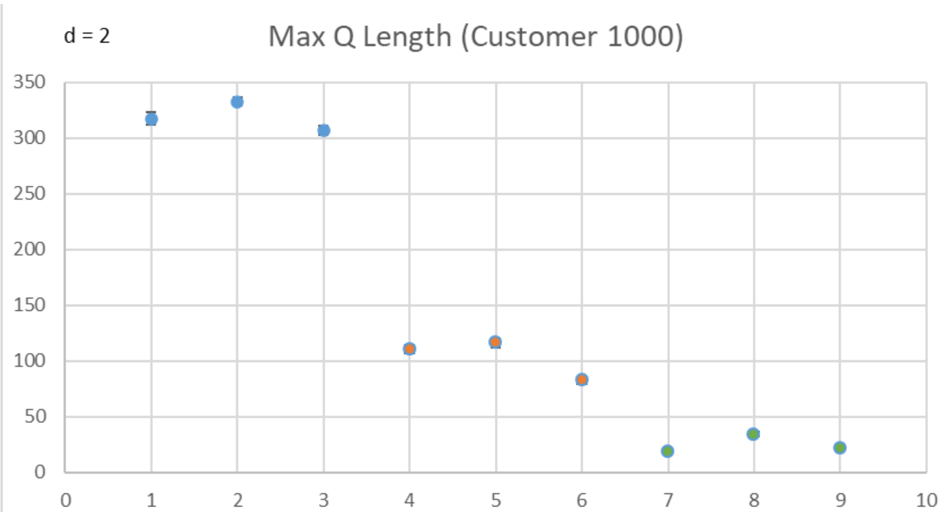

Figure IV.II.II Max Queue Length with 1000 customers where d = 2

In **Figure IV.II.I**, which represents a system with 100 customers, it can be observed that when the utilization rate is at 1.3, both Round Robin (RR) and Purely Random (PreRand) overlap with RandMin, making it difficult to discern their relative performance. However, as the utilization rate decreases, RR continues to overlap with RandMin, whereas PreRand exhibits a higher maximum queue length than RandMin.

When the number of customers is increased to 1000, as shown in **Figure IV.II.II**, RR still overlaps with RandMin, while PreRand exhibits a higher maximum queue length than RandMin across all utilization rates. Similarly, as utilization decreases, RR continues to overlap with RandMin, and PreRand remains larger than RandMin.

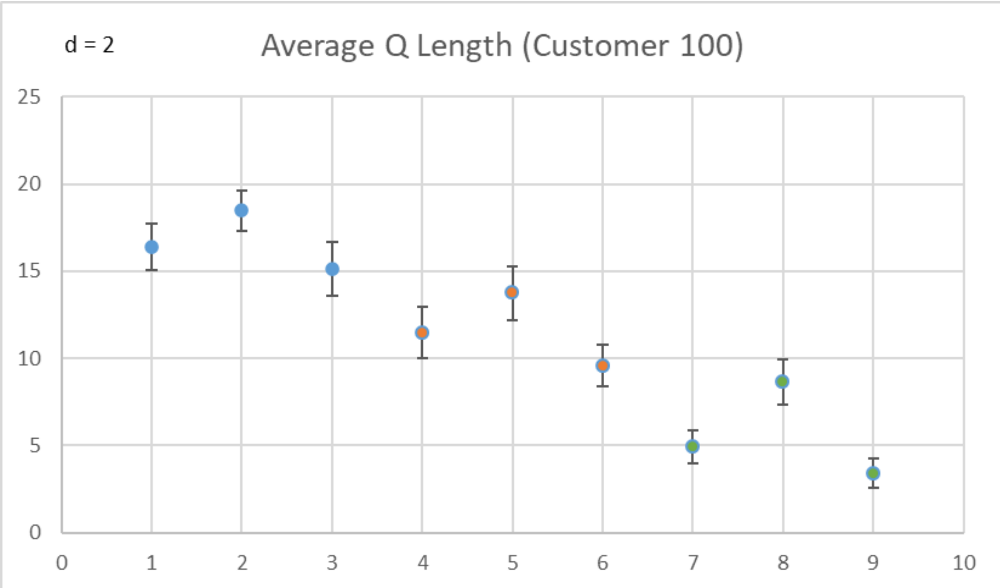

Figure IV.II.III Average Queue Length with 100 customers where d = 2

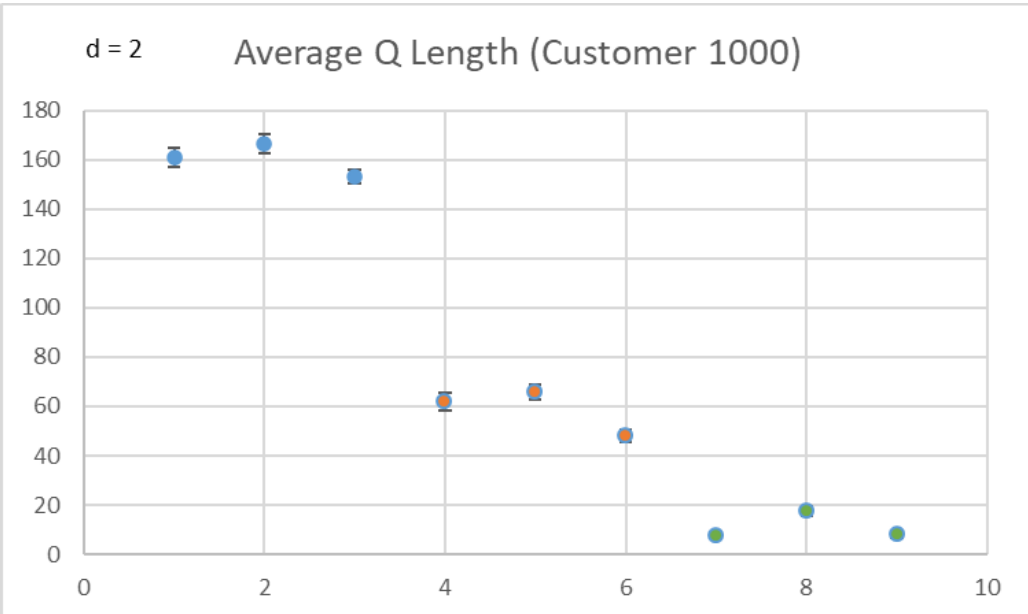

Figure IV.II.IV Average Queue Length with 1000 customers where d = 2

In terms of the average queue length, we can observe similar patterns for the Round Robin (RR), Purely Random (PreRand), and RandMin algorithms in **Figure IV.II.III** for a system with 100 customers, where both RR and PreRand overlap with RandMin as seen in **Figure IV.II.I**.

When the number of customers is increased to 1000, as depicted in **Figure IV.II.IV**, RandMin outperforms both RR and PreRand in terms of average queue length at a utilization rate of 1.3. As the utilization rate decreases, RR overlaps with RandMin, and RandMin continues to have a lower average queue length than PreRand.

Interestingly, we find that the average queue length and maximum queue length exhibit similar patterns for both 100 and 1000 customers, as shown in the figures above.

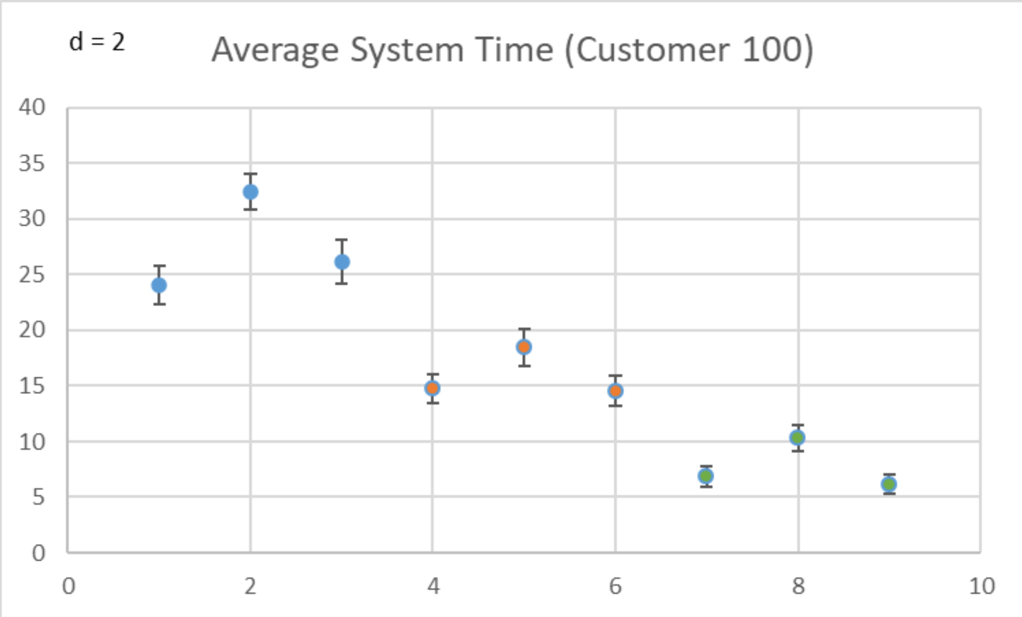

Figure IV.II.V Average System Time with 100 customers where d = 2

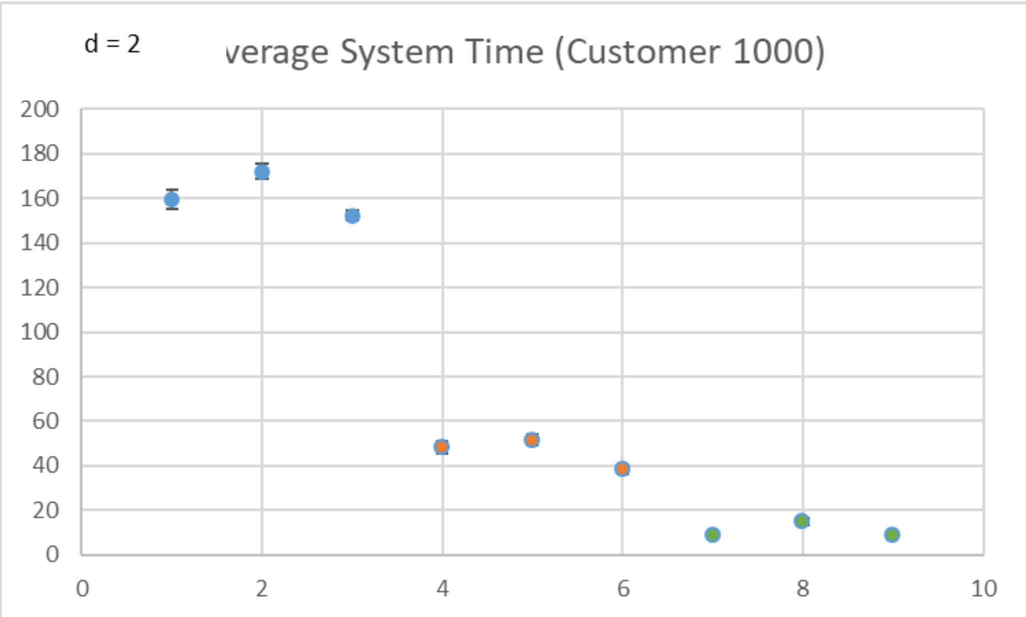

Figure IV.II.VII Average System Time with 1000 customers where d = 2

When considering the Average System Time for a system with 100 customers, we can observe in **Figure IV.II.V** that RR overlaps with RandMin, while PreRand exhibits a higher

Average System Time than RandMin. These results remain consistent as the utilization rate decreases.

In **Figure IV.II.VII**, which represents a system with 1000 customers, we find that the same pattern holds when the utilization rate is at 1.3. However, as the utilization rate decreases, RandMin starts overlapping with both RR and PreRand in terms of Average System Time.

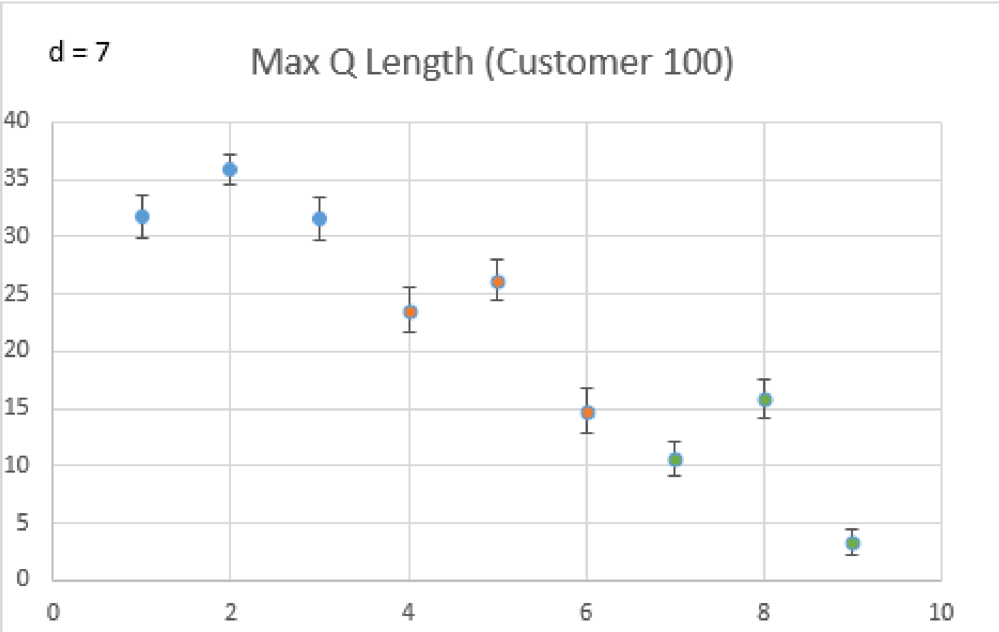

Figure IV.II.VIII Max Queue Length with 100 customers where d = 7

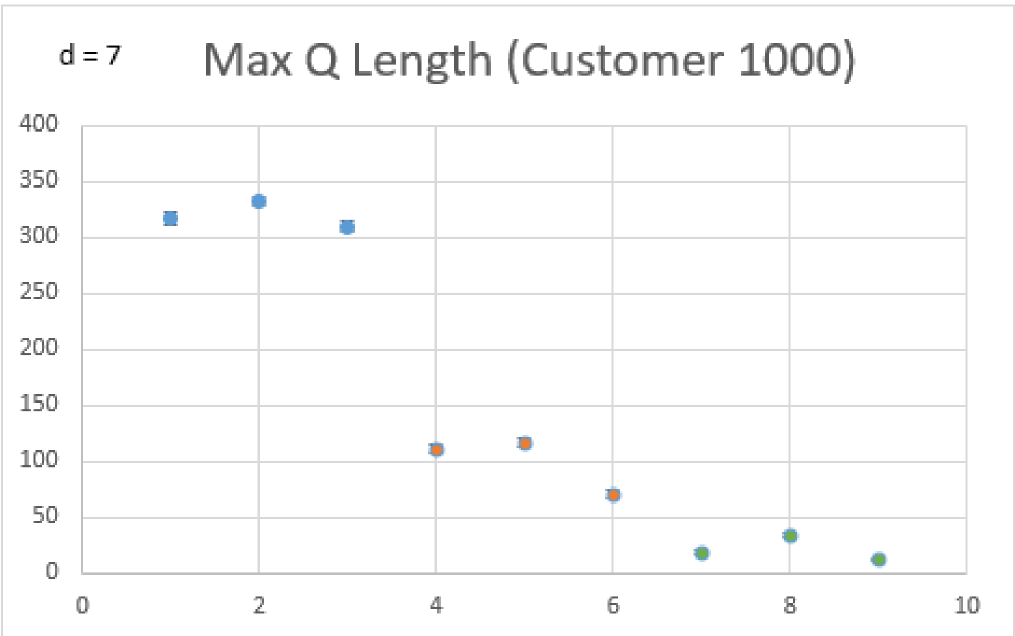

Figure IV.II.IX Max Queue Length with 100 customers where d = 7

With 100 customers, when the utilization is around 1.3, you can see that the pure random method has the most customers in the queue.However, when the utilization is less than one, the max queue length for the Randmin method is significantly smaller than the other two methods.

When there are 1000 customers, it can be observed that during times when the server is extremely busy or extremely idle, PureRandom always holds the most customers in the queue, and there is barely any noticeable difference between the number of customers holding in queue RR and Randmin.

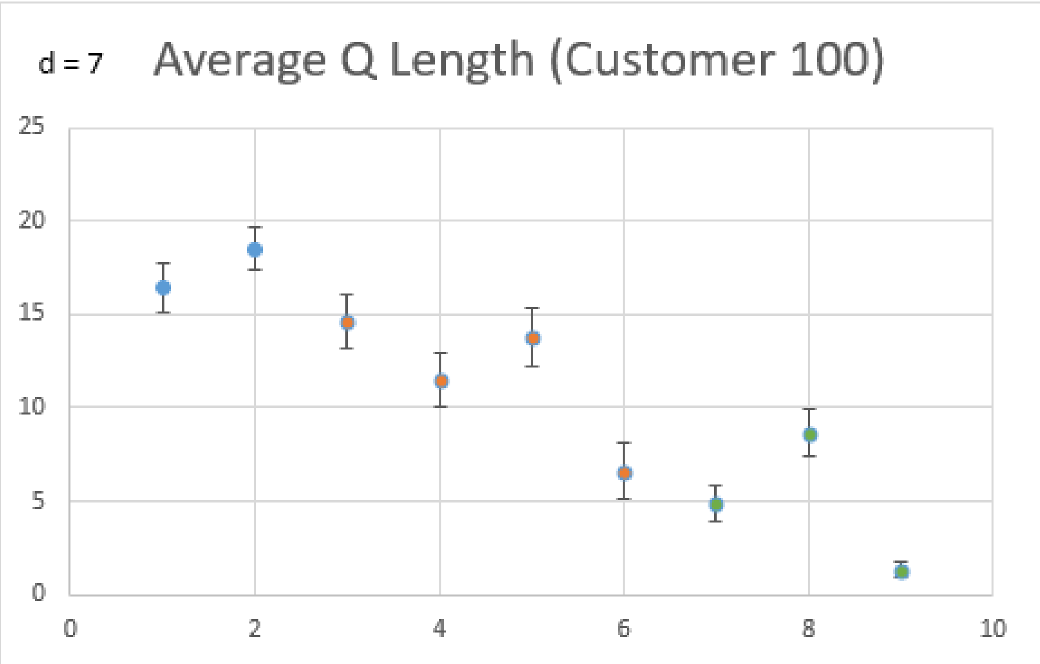

Figure IV.II.X Average Queue Length with 100 customers where d = 7

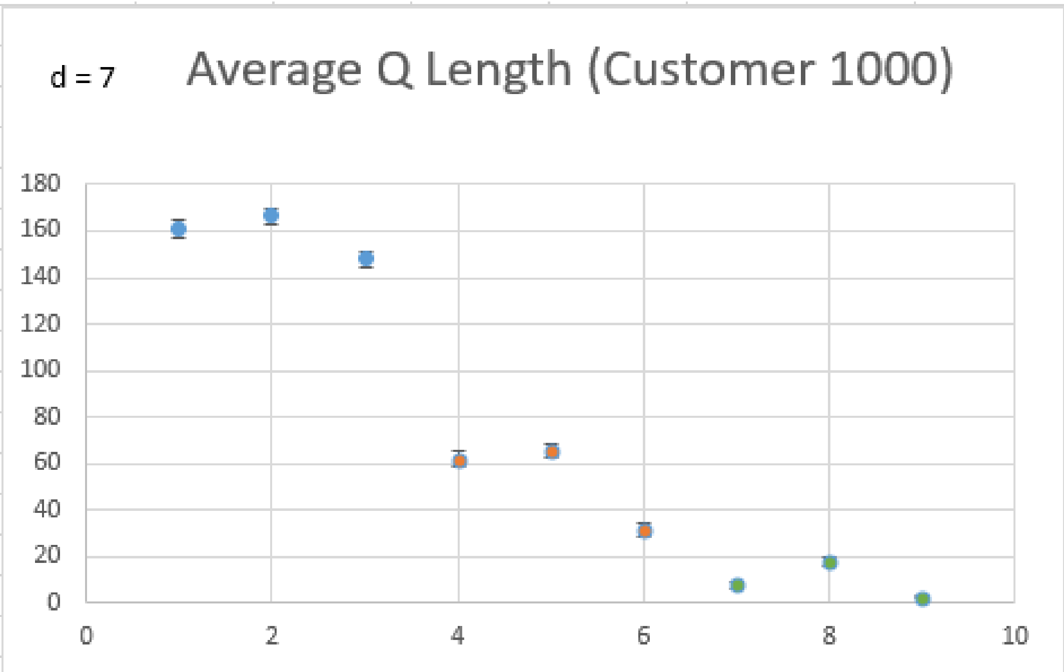

Figure IV.II.XI Average Queue Length with 1000 customers where d = 7

When d = 7, the Average Queue Length graph provides a comparable understanding of the system's performance as the Max Queue graph. Both graphs depict how well each method handles the customers under varying server loads. In this scenario, the Average Queue Length and Max Queue values seem to be closely related, which indicates that the insights gained from analyzing one can be largely applied to the other.

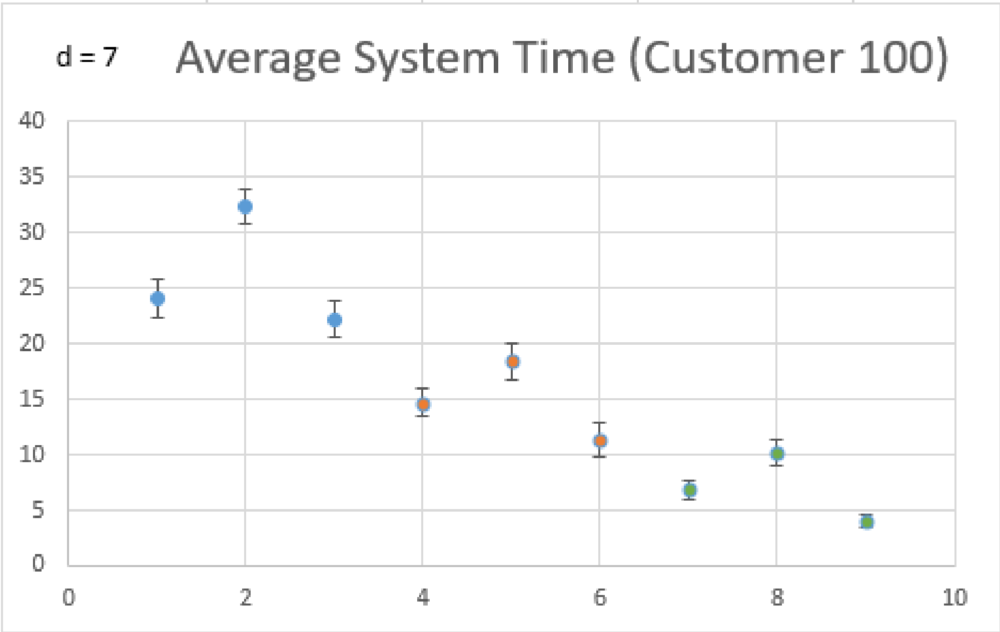

Figure IV.II.XII Average System Time with 100 customers where d = 7

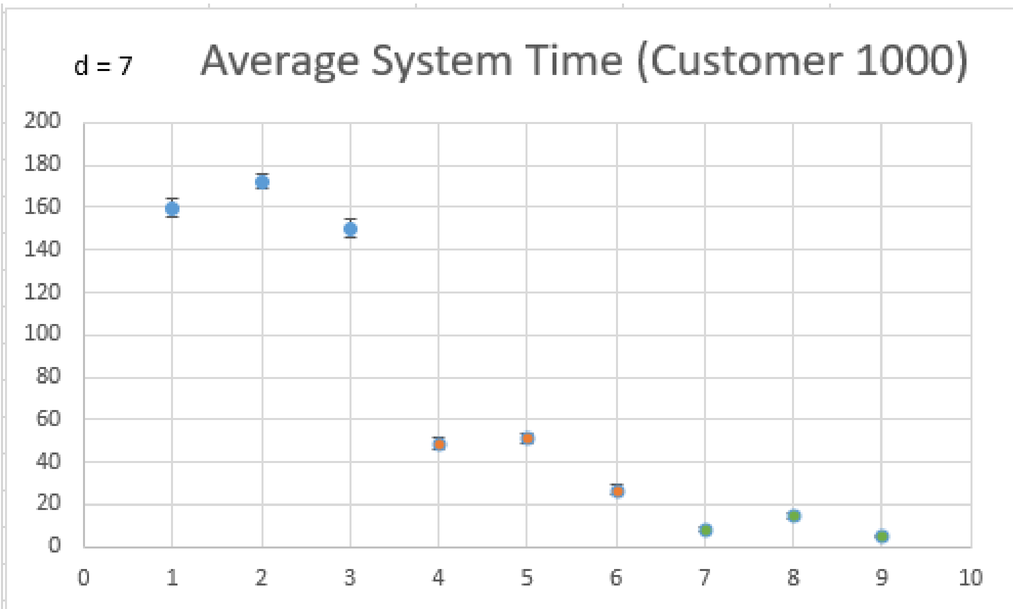

Figure IV.II.XIII Average System Time with 1000 customers where d = 7

In **Figure IV.II.XII** and **Figure IV.II.XIII**, we can see that regardless of whether the number of customers is 100 or 1000, the average system time for Randmin is generally the lowest. It is only when the number of customers is 1000 and the utilization is equal to 0.7 that it becomes difficult to distinguish the difference between RR and Randmin. This indicates that Randmin tends to outperform the other methods in terms of average system time, except for specific scenarios where its advantage is less pronounced.

IV.III The importance of choosing the value of d.

The choice of an appropriate value for d is crucial for the efficiency of the Randmin algorithm. When an excessively small value of d is chosen, the performance of the algorithm might be significantly weakened. Sometimes, it can be difficult to discern any differences between the Randmin method and the RR method. In certain situations, the Randmin algorithm may even be less efficient than the RR method. Furthermore, when the number of customers is small, the uncertainty in the results of max queue length and average queue length brought by the Randmin algorithm can be quite significant.

V. Conclusion

In summary, the analysis of our data and figures indicates that the RandMin algorithm is overall a better option for managing queues in systems with varying numbers of customers and utilization rates, as it generally outperforms Round Robin(RR) and Purely Random(PureRand) in terms of maximum queue length, average queue length, and average system time. The RR algorithm sometimes tends to overlap with RandMin in these metrics, especially when the utilization rate is low, while PureRand consistently performs worse. However, the choice of an appropriate value for d is crucial for the efficiency of the RandMin algorithm, as an excessively small value might significantly weaken its performance or even be less efficient than the RR method. Both the average queue and maximum queue length exhibit similar patterns, indicating that insights from one can be largely applied to the other. and One aspect that should be highlighted is that the superiority of the RandMin algorithm becomes less discernible under specific conditions, such as when the system hosts 1000 customers and operates at a utilization rate of 0.7.

V.I Further Improvements

In this model we have already analyzed as many variables as possible. We have intotal 18 tables of data and 12 charts for analysis. In order to improve our simulation results, we might need to add more comparison groups under the same service rate with different arrival flows. However, that would be at least two times of the workload which we have already done in this project. For the calculation part, we might use the actual confidence interval equation for a more accurate result. Here, we might just simply conclude what we already have right now.