CNG 476: Semester Coursework

SIMULATIONS AND ANALYTICAL MODELLING

Daniyal Selani, 2106268

5-1-2019

Introduction

The aim of this semester coursework was to apply the concepts learn during the lectures into a working project. The objectives were to implement a program to simulate an M/M/c/L queuing system (Poisson arrivals, exponentially distributed service times and c servers working in parallel). The simulations would assume a quasi-birth and death model with state independent mean arrival and service rates lambda and mu respectively. The simulation would then be tested against a suitable analytical model to determine the efficacy and correctness of the implemented simulation.

This document contains the design of the simulations, the hurdles faced in developments, and the design choices that were taken to overcome these hurdles. This document will also contain an analytical model, and the results of testing the implemented simulation against the analytical model.

Design and Development

Context

It is important to establish the context of the development and progress of the coursework. I was partnered with another student, and this student dropped the course midway during December. This meant that I had to restart development, as the other student had taken responsibility to do the initial designing of the project. Hence, there was a need for rapid result-oriented development for me to meet the set deadline.

Design Choices

System Type

For the required rapid development, a clear design direction was needed. It was decided to try to implement a network server, and specifically a VoIP (Voice over Internet Protocol) Servers. VoIP packets carry video and audio data for video conferencing applications like Skype. [1] This means that if a packet is dropped or lost, no attempt is made to recover it. For video conferencing applications, recovering old lost packets does not make sense as the application needs the most recent data.

System Design

Keeping this first decision in mind, the simulated system was designed with these features:

- Packets enter the system into a shared queue. This queue is always unbounded
- There are C servers in the system. Each server has its own queue
- Each server queue can be either unbounded or bounded to length L
- If there is any packet loss in the system due to breakdown or congestion, the packets are discarded and not recovered
- A server does not become active till it receives its first packet
- The system is not intelligent or adaptive, hence if a server breaks down, the system will still try sending the server packets when it can
- If server is broken down, the server queue will accept any forwarded packet from the system. But the server will reject the packet when the server queue forwards the packet to it

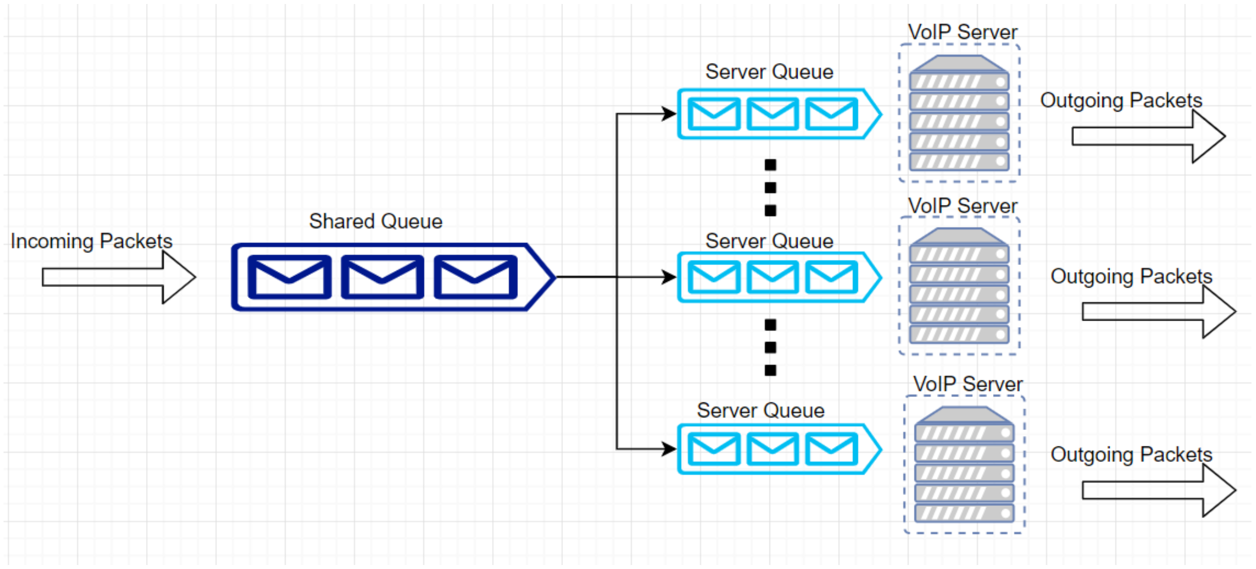


Fig 1: System Design

Simulation Design

The implemented program will simulate the following events:

- The arrival of packets
- Processing of packets
- Departure of packets
- Server breakdown
- Server repair

Even though the program simulates and generates all these events, because of the design approach taken due to the limitations faced, it can only iterate over the packet arrival event. This means that the only generated event that moves the simulation from one state to the next is packet arrival. Only after moving to the next state, all other events are generated, and the state of the system is updated accordingly.

The packets will have a Poisson arrival, with mean arrival rate lambda. The packets will be processed with an exponentially distributed service rate. Server breakdown follow a Weibull distribution with mean break-down rate b. Server repair will follow a lognormal distribution with repair rate r.

The simulation will take in user input for the following parameters:

1. n: The number of total packets the simulation will process
2. l: The queue bounded length (queue length can be left unbounded)
3. lbd: mean packet arrival rate
4. mu: mean packet service rate
5. b: average number of breakdowns per hour
6. r: average number of repairs per hour

Simulation Program Flow

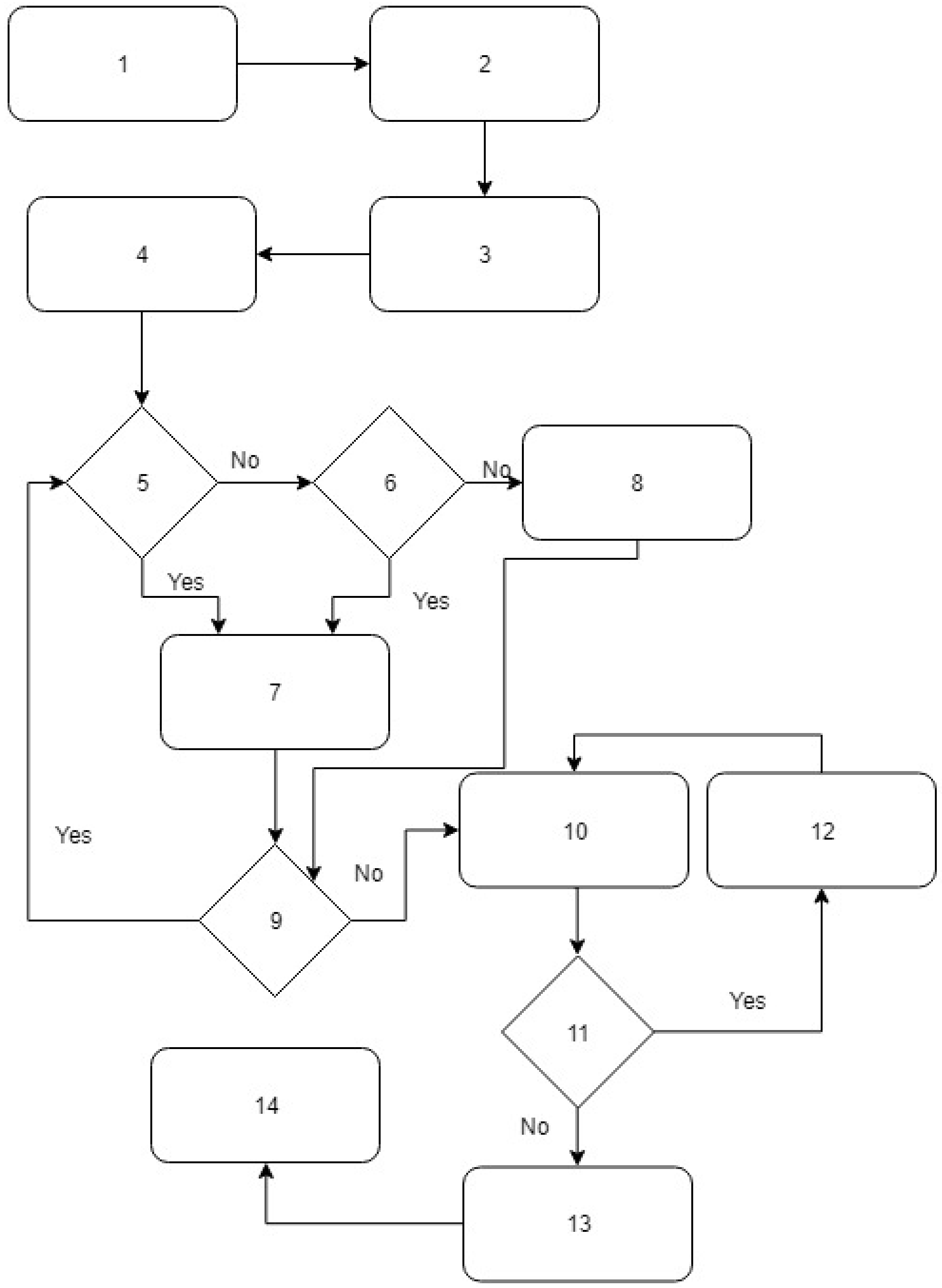


Fig 2: Program flowchart

1. Initialize simulation and enter simulation parameters
2. Initialize servers
3. Initialize server queue

4. Populate shared queue with packets
   Arrival time and service time for each packet is generated at this step. Each packet arrival and departure is an independent event.
5. Are there Idle servers in the system?
6. Does any server have capacity in its queue?
7. Enter the packet in the chosen server's queue, remove from shared queue
8. Discard packet and remove from shared queue
9. Any packets left in shared queue?
10. Iterate over every packet in every server
    Server breakdown and repair probability is calculated at this step.
11. Is server broken down?
12. Discard packet currently being processed
13. Process packet, calculate values (Wait time, start time, etc.)
14. Calculate statistics for each server, output result

Development and Implementation

The simulation program was implemented in python 3.6. Python was chosen as it is a language, I am familiar in, it is easy to work with, robust, and offers a lot of powerful libraries to enable rapid develop.

The libraries used are NumPy and matplotlib. NumPy provides a random sample from a uniform distribution, which is used to sample other distributions, as will be explained below. Matplotlib provided graphing abilities to observe the data.

Classes

An OOP based approach was used when implementing the simulation. It would not be very readable if a proper UML class diagram to represent the simulation was drawn, as it would be too large to fit into one page. Instead I will use a UML class diagram to show the overall structure and hierarchy of the program and then detail each class' methods and fields.

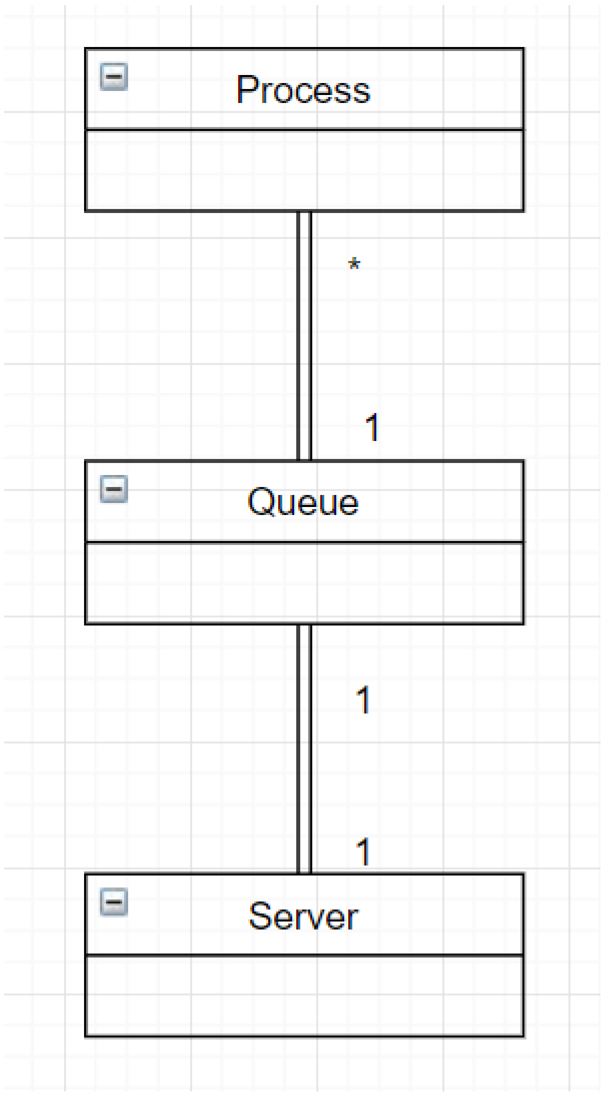


Fig 3: Class Diagram

Process

In the simulation each process is simulated as a packet

Fields:

- Float Arrival time (Generated by a function)
- Float Start time (The time at which the packet arrives at the server)
- Float Completion time (The time at which the packet exits the system after being processed)
- Float Service time (Generated by a function)
- Float Wait time (The total amount of time the packet waited in the server queue)
- Int ID (Each packet that arrives in the system is given a unique ID)
- Float Idle time (The amount of time the server was idle before the packet arrives at the server)
- Int Queue length (The length of the server queue at the time the packet arrived at the server queue)
- Int Server ID (The ID of the server that the packet was processed at)
- Bool Rejected (If True then the packet was rejected due to bounded queue limit or server breakdown)

Methods:

- Compute complete time
- Update process time (updates the service time of the packet after it is generated by the function)

Queue

Queue has a many to 1 relationship with Processes. Each Process can be only in 1 queue at any given time (shared or server). A queue can have many processes.

Fields

- Process Array Queue (contains all the processes that are supposed to be in the queue)

Methods

- Add Process (Adds process to queue)
- Remove Process (Removes process from queue)
- Reset Queue (Empties out the Queue)
- Process Queue (Populates the shared queue and generates event times for each process)
- Calculate (Calculates the event data for each process)

Server

Has a 1 to 1 relation with queue. A server can have only 1 queue. A queue does not need a server (shared queue), but at most can have 1 queue.

Fields

- Int ID (Each server has a unique ID)
- Float Total busy (The amount of time the server was busy during simulation)
- Float Total Idle (The amount of time the server was idle during simulation)

- Queue Q (The Queue feeds processes to the server)
- Float Server Utilization (The utilization of the server during simulation)
- Float Average Wait Time (The average wait time in queue of the packets that the server processed)
- Float Average Service Time (The average time it took a server to complete a process)
- Float Up time (The total amount of time the server was active)
- Float Start time (The time at which the server received its first process)
- Int Dropped (The amount of processes the server and the server queue dropped due to either server failure or queue limit being reached)
- Int Accepted (The amount of processes the server successfully processed)
- Float Throughput (The throughput of the server in processes/unit time)
- Float Average Queue Length (The mean queue length of the server queue during the simulation)
- Float Array Breakdowns (Records the time of each breakdown at the server)
- Float Array Repairs (Records the time of each repair at the server)
- Bool Broke Down (If true, server is broken down, if false server is functioning)

Methods

- Enqueue Process (Adds process to server queue and calculates the event data of process)
- Calculate Statistics (Calculate the usage statistics of server and server queue after simulation finished)
- Print statistics (Prints usage statistics of the server and server queue)
- Print Queue Data (Prints the data about each process in server queue)
- Export Server Data (Exports all relevant data to csv files)

Helper Functions

The following are helper functions which are employed in the simulation

- Choose Server (Sends process from shared queue to server queue depending on queue size)
- Export Data (Exports system statistical data to a csv file)
- Initialize server (initializes all the servers in the system)
- Random Generator (Generates values for events of the simulation)

Random Number Generation

Inverse Transform Technique

Producing exponentially distributed random numbers can be thought of as sampling points from an exponential distribution and finding the corresponding value of the sampled point. We will use inverse transform method to sample an exponentially distributed system. We will use these values for the Poisson arrival values of the packets in our simulations, as well as the service time values in our simulation.

A general proof and background of the inverse transform method is as follows:

In order to find the corresponding value of any point in a distribution mathematically, the distribution must be represented through an invertable function. That is to say, for any input point A in a distribution F, there must be only one output point B, the distribution function must be one to one.

A distribution can be represented through its pdf (Probability Density Function), however this cannot be used to sample a random point as the function is not always one to one, see the plot of a pdf of a normal distribution bellow.

Instead a cdf (Cumulative Density Function) should be used. A cdf (F(x)) of a distribution is given by :

$$F(x) = \int_{-\infty}^{x} f(t)dt$$

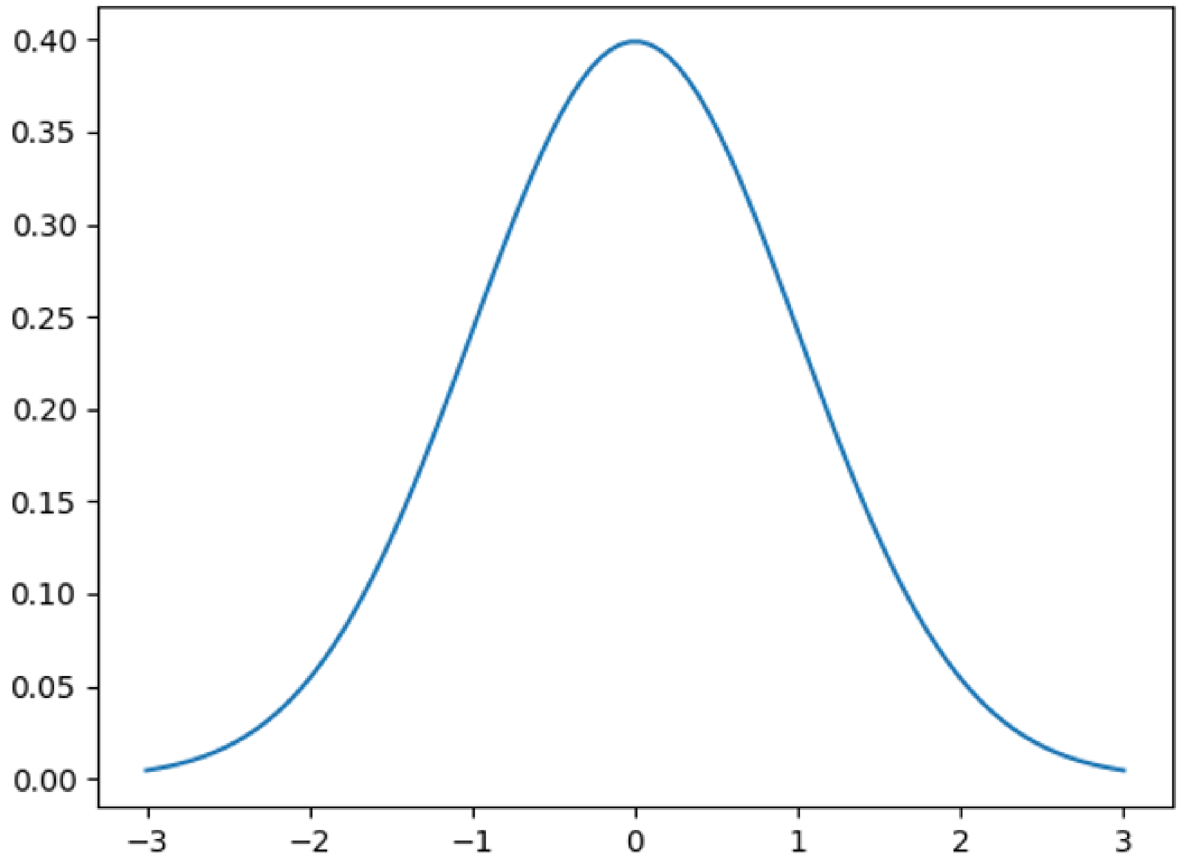


Fig 4: Normal Distribution PDF

Where f(t) is the pdf of the distribution. A cdf is always rising as it is the cumulative sum of all the points (values on the y axis) in a pdf from -inf to +inf. Since these points represent probabilities and a probability value can never be negative, this sum is always increasing. Hence the cdf is a function that is always rising and hence it has to be one to one for any distribution.

NOTE: There is more rigorous proof dealing with why cdf is used in sampling random variables, through a technique called inverse transform, that proof is too lengthy and out of scope to include. The proof provided is only intuitive and not mathematical

To sample from a cdf, simply take the inverse of the function for any point R (value on x axis), for a corresponding random variable (value on the y axis).

The pdf for the exponential distribution is given by:

$$f(x) = \begin{cases} \lambda e^{-\lambda x}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

And accordingly, the cdf is :

$$F(x) = \int_{-\infty}^{x} f(t)dt = \begin{cases} 1 - e^{-\lambda x}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

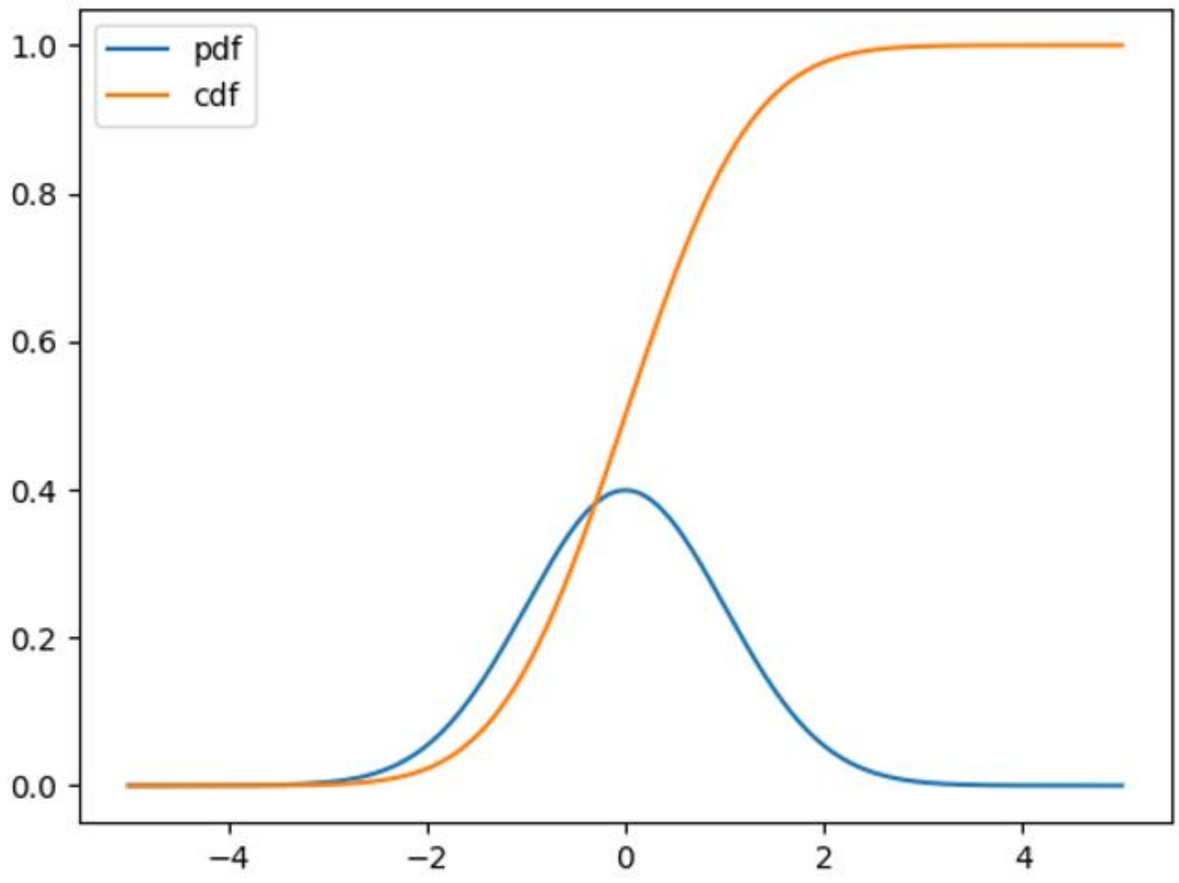


Fig 5: Normal Distribution PDF and CDF

To get a function to sample for any value X, solve F(x)= R in terms of x:

$$\begin{aligned}
1 - e^{-\lambda X} &= R \\
e^{-\lambda X} &= 1 - R \\
-\lambda X &= ln(1 - R) \\
X &= \frac{-1}{\lambda} ln(1 - R)
\end{aligned}$$

NOTE: In the above equations, the symbol ƛ denotes the inverse of the mean of the exponential distribution. That is, it denotes the frequency of the events that the distribution models. Higher the value of ƛ is, the more frequent the events will be. For the Exponential distribution, its mean Mu = 1/ ƛ

Because the distributions that we are dealing with are probabilistic, the sum of probabilities can never exceed 1. F(x) = R → 1 as x → infinity. Hence the number R must be within the range of (0, 1), and as the mathematical model requires the generation of random variables, the value R is to be chosen from a uniform distribution over the range (0, 1).

We can now use this sampling technique to produce a pdf and cdf for an exponential distribution and see if our technique is correct.

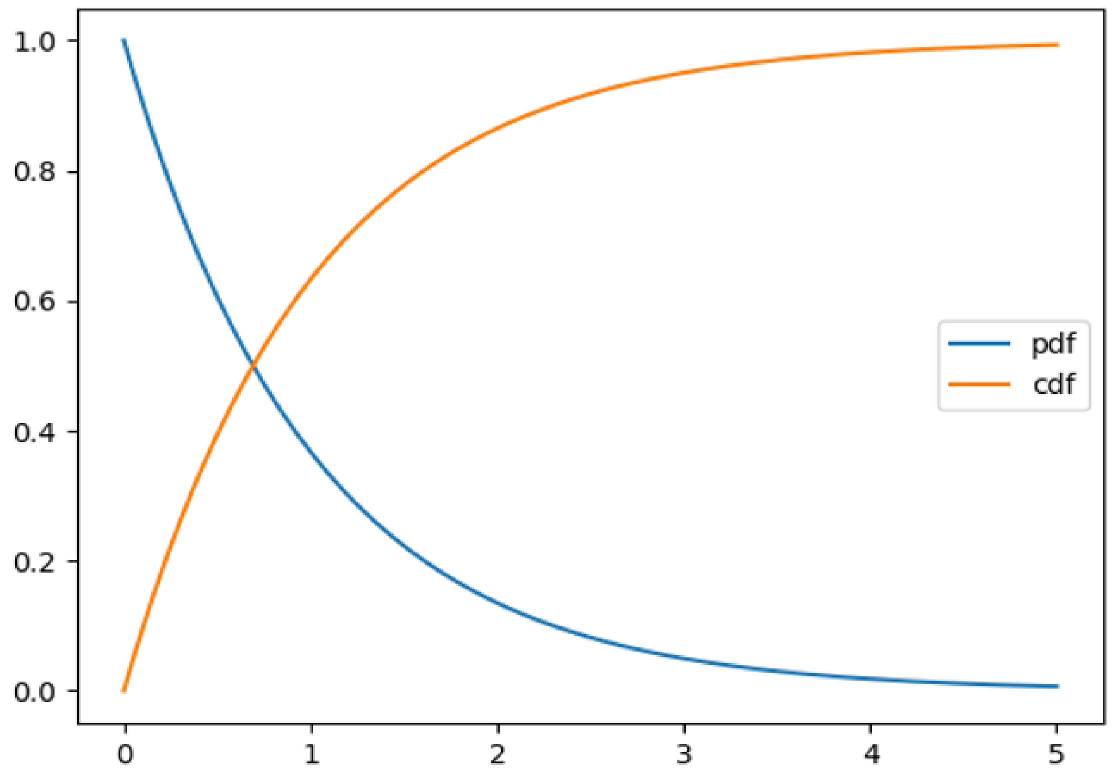


Fig 6: Exponential distribution pdf and cdf.

From the results, it indeed is. We will use this technique to sample values for breakdown and repair events as well

Weibull Distribution

The Weibull distribution has the probability density function (pdf)

$$f(x) = \frac{\beta}{\alpha}\left(\frac{x}{\alpha}\right)^{\beta-1} e^{-\left(\frac{x}{\alpha}\right)^{\beta}}$$

for $x \geq 0$. Here $\beta > 0$ is the **shape** parameter and $\alpha > 0$ is the **scale** parameter.
The cumulative distribution function (cdf) is

$$F(x) = 1 - e^{-\left(\frac{x}{\alpha}\right)^{\beta}}$$

The inverse cumulative distribution function is $I(p)$ =

$$\alpha(-\ln(1-p))^{1/\beta}$$

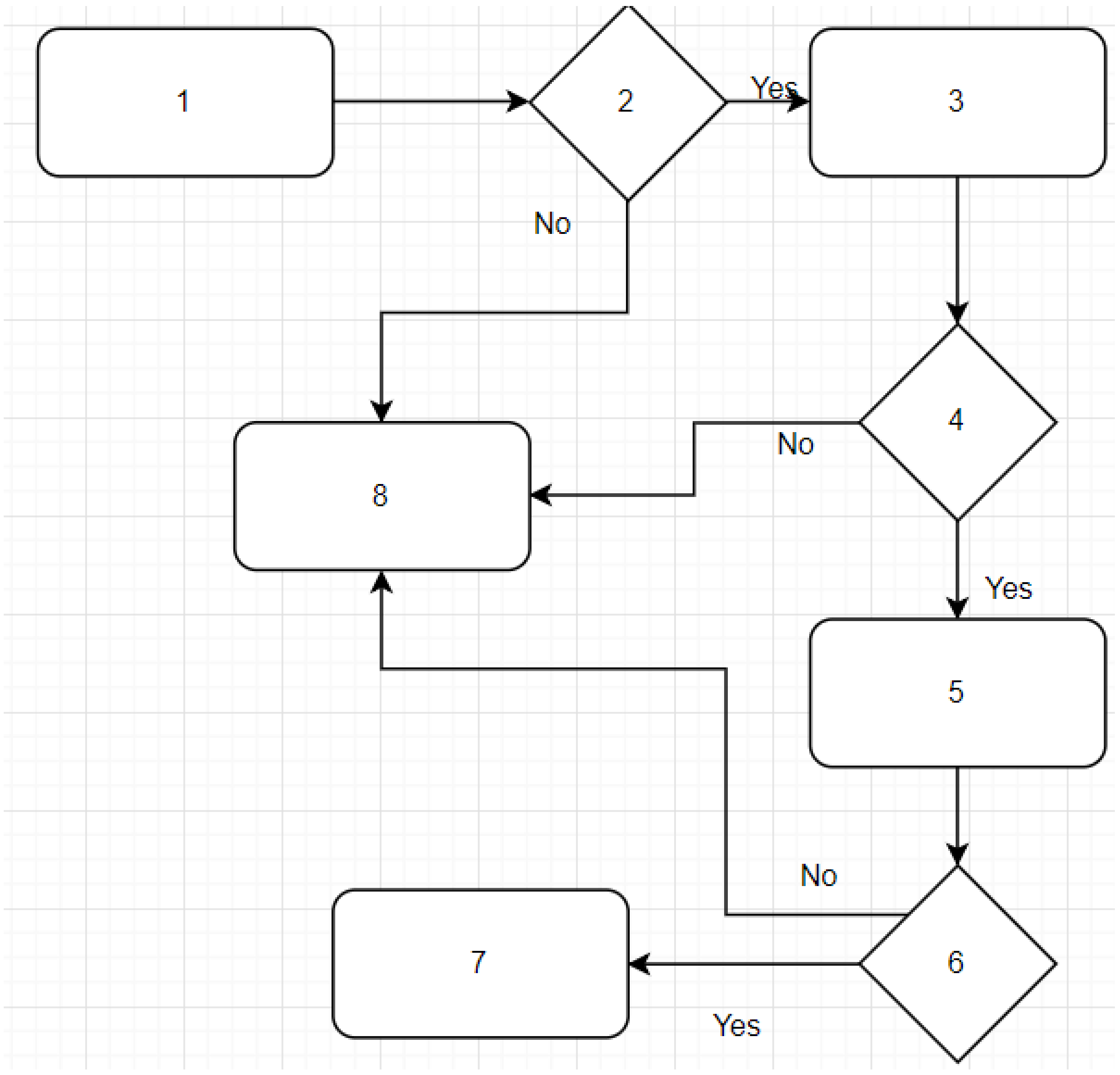
Hence, we will use the inverse transform method by using I(p) to sample values for the breakdown and repair event generation.

Breakdowns and Repairs

In the simulation flowchart, at item (10) and (11), the breakdown and repair events were generated and implemented. In this section the implementation will be explained in more detail.

The first important note is that breakdown and repairs are not independent event. A breakdown cannot happen if the server is already broken down, and a server cannot be repaired if it is not broken down. Hence, these 2 events cannot be generated and populated in a queue at the beginning of the simulation just like packet arrival and service time were. Each event must be generated every time the simulation iterates and transitions from one state to the next.

Furthermore, the coursework defines the 2 user defined values for breakdown and event generation. Breakdowns/hour and repairs/hour. However, since the unit time used in the simulation is seconds, these values have to be converted to breakdowns/second and repairs/second.



1. Packet arrives to server queue
2. Is breakdown and repair enabled?
3. Generate Breakdown event
4. Is Server broken down?
5. Generate repair event
6. Is server still broken down?
7. Discard packet
8. Send packet to next step

Outputs

The simulation outputs 4 kinds of results as screen output and as csv files. CSV files are chosen because they are easily readable and can be opened using either Google Sheets or Microsoft Excel. The program outputs:

1. Raw data, which is simply the event data of every packet that arrived in the system
2. Observed system data, which is the observed statistics of the activity of the system during simulation
3. Calculated data, which is the data about the activity of the system calculated using analytical modelling (As detailed below)
4. Loss factor data, which is the data that is used for comparison between analytical modelling and the simulation (As detailed below)

Additionally, the system outputs a graph of the loss factor data. The other data is not graphed as to reduce visual clutter when running the simulation. Loss factor is graphed because it immediately tells if the simulation and analytical model had similar results.

The data is stored in 3 type of csv files

1. Raw_data : This stores the activity of each packed that arrives at the system, regardless if that packet was discarded or accepted by the server
2. Raw_server_n_data: This stores the activity of each packet that is processed by the nth server
3. System_data: Stores the observed and calculated statistics from the simulation and analytical model, as well as loss factor data

Analytical Model

To get a sense of the correctness of my simulation, I compared my results to an Analytical model. The analytical model was derived from Little's Law and basic queueing theory.

For the coursework, I focused on the following performance variables to compare between my analytical model and simulations:

1. Mean service time (mu)
   Given by user in both analytical model and simulation
2. Mean inter arrival time (lambda)
   Given by user in both analytical model and simulation
3. Number of servers in system (c)
   Given by user in both analytical model and simulation
4. The average server utilization (ro)
   Simulation: Calculated by averaging the capacity of all the servers during the simulations
   Analytical Model: ro = lambda/(mu*c)
5. The average queue length (lq)
   Simulation: Calculated by averaging the mean queue length of all the servers in the system
   Analytical Model: lq = (ro^2)/(1-ro)
6. The average waiting time in queue (Wq)
   Simulation: Calculated by taking the mean waiting time in queue of all packets that were processed by the system (The packets that were discarded and not processed were not considered.
   Analytical Model: Wq = lq/lambda
7. Response time (The average time in system) (W)
   Simulation: Calculated by taking the mean waiting time in the system of all packets that were processed by the system (The packets that were discarded and not processed were not considered.
   Analytical Model: W = Wq + (1/mu)
8. Throughput (Clients served per unit time) (mu)
   Simulation: Calculated by finding the mean of the number of packets a server has processed
   Analytical Model: mu

The state probability for state k (Pk), of the system can be calculated with the formula:

Pk = (1- ro)*(ro^k)

Additional Notes

The analytical model used assumes that lambda < mu *c, i.e. the system utilization will always be less than 1. If these parameters are not met, then the analytical model produces faulty results that cannot be used for comparative analysis

The analytical model also does not compute data regarding the breakdown and repair of the servers. This is because this was not in the scope of the project and not asked from this coursework either.

Results

Testing Approach

The simulation will be tested in two stages. First the simulation's correctness needs to be verified. I.E to see if that the simulation provides the correct results when compared to the analytical model. If so, then under which parameters does the simulation provide the most correct results.

If the simulation achieves an acceptable level of correctness, only then will the simulation be tested using the guidelines provided in the coursework paper. If not, then the design of the simulation needs to be reevaluated

The testing results will be represented in tabular and graphical form wherever possible. All the graphics will be plotted using matplotlib. Every time the simulation is run, all raw data is saved into different csv files, I will not be showing the contents of every file, but only the most relevant data.

Loss Factor

There is a need to find a normalized way to compare the results between the analytical model and simulation in a measurable way. We know that both methods will produce some results when given input parameters. We can use a formula to calculate the difference between each of these results. This difference will be called the loss factor. The loss factor will always > 0, and the lower the loss factor, the more similar the outputs between the analytical model and simulation.

The loss factor will be calculated for:

1. Mean Server Utilization (ro)
2. Mean Wait Time in Queue (Wq)
3. Mean System Response Time (W)
4. Mean Queue Length (Lq)
5. System Throughput (mu)

Finally, there will be a mean loss factor which will average the loss factor values of all of the above

Loss factor formula:

((calculated variable – observed variable)^2)^0.5

The calculated variable is the variable result from the analytical model

The observed variable is the variable result from the simulation

Correctness Testing

First, we will test the simulation to check its correctness using loss factor.

We will fix certain input parameters and vary others, and we will do 3 testing runs initially. If the desired results are not achieved, then parameters will be tweaked and the reason for the undesired result will be deduced.

Run 1

Parameters

Lambda = 0.5

Mu = 1

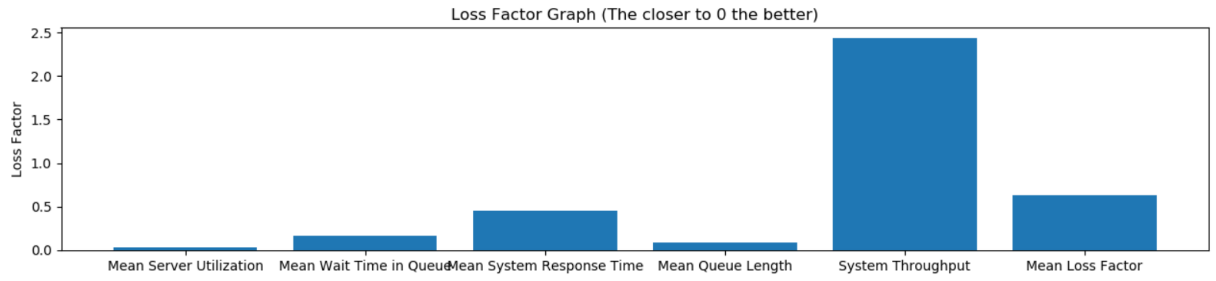
C = 2

L = infinite

N = 10

No breakdown and repair

Results



server id	utilization	average service time	average wait time in queue	average queue length	number of packets processed	number of packets dropped	server uptime	server throughput
0	25.28111	0.765409	0	0	5	0	15.13796	3.027593
1	17.11199	0.651325	0	0	5	0	19.21475	3.84295
Observed System Statistics								
Mean Server Utilization	Mean Average Service Time	Mean Average Wait Time in Queue	Mean System Response Time	Average Mean Queue Length of server queues	Average number of packet process	Average number of packets dropped		
21.19655	0.708367	0	0.708367	0	5	0		
Calculated Statistics								
Mean Server Utilization	Mean Queue Length	Mean Average Wait Time in Queue	Mean System Response Time	Average System Throughput				

25	0.083333	0.166667	1.166667	1				
Loss Factors								
Mean Server Utilization	Mean Wait Time in Queue	Mean System Response Time	Mean Queue Length	System Throughput	Mean Loss Factor			
0.038034	0.166667	0.4583	0.083333	2.435271	0.636321			

Run 2

Parameters

Lambda = 0.5

Mu = 1

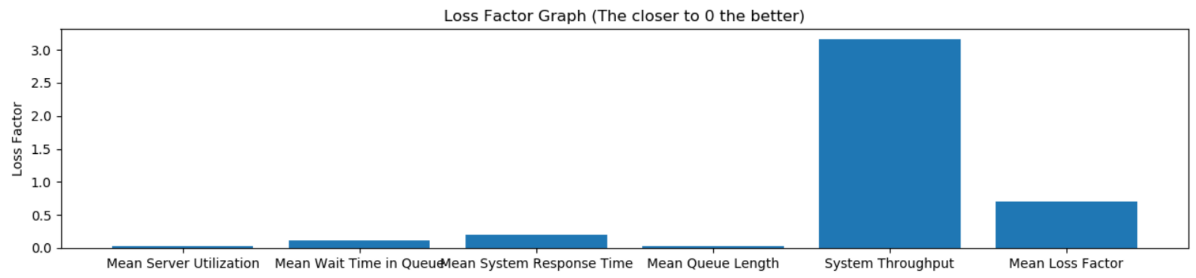
C = 2

L = infinite

N = 100

No breakdown and repair

Results



server id	utilization	average service time	average wait time in queue	average queue length	number of packets processed	number of packets dropped	server uptime	server throughput
0	23.02698	1.013607	0.055186	0.042553	47	0	206.2324	4.387924
1	20.76852	0.816133	0.052536	0.056604	53	0	208.2721	3.929663

Observed System Statistics							
Mean Server Utilization	Mean Average Service Time	Mean Average Wait Time in Queue	Mean System Response Time	Average Mean Queue Length of server queues	Average number of packets processed	Average number of packets dropped	System Throughput
21.89775	0.91487	0.053861	0.96873	0.049578	50	0	4.158794

Calculated Statistics				
Mean Server Utilization	Mean Queue Length	Mean Average Wait Time in Queue	Mean System Response Time	Average System Throughput
25	0.083333	0.166667	1.166667	1

Loss Factors					
Mean Server Utilization	Mean Wait Time in Queue	Mean System Response Time	Mean Queue Length	System Throughput	Mean Loss Factor
0.031023	0.112806	0.197936	0.033755	3.158794	0.706863

Run 3

Parameters

Lambda = 0.5

Mu = 1

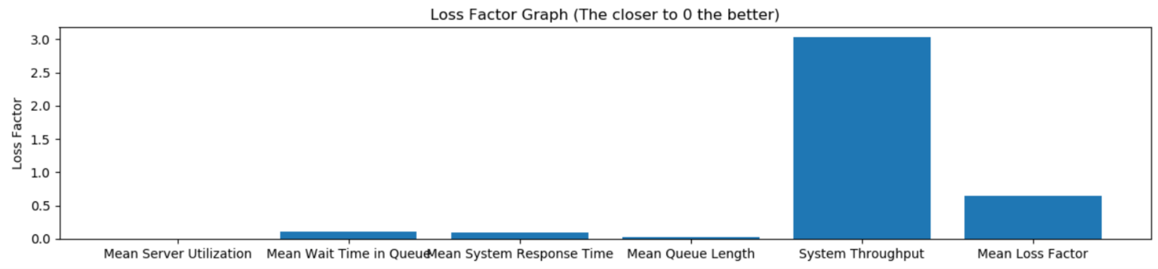
C = 2

L = infinite

N = 1000

No breakdown and repair

Results



server id	utilization	average service time	average wait time in queue	average queue length	number of packets processed	number of packets dropped	server uptime	server throughput
0	25.22928	0.997291	0.068484	0.123529	510	0	2014.222	3.949455
1	24.82099	1.021371	0.048396	0.093878	490	0	2016.324	4.114947

Observed System Statistics							
Mean Server Utilization	Mean Average Service Time	Mean Average Wait Time in Queue	Mean System Response Time	Average Mean Queue Length of server queues	Average number of packets processed	Average number of packets dropped	System Throughput
25.02514	1.009331	0.05844	1.067771	0.108703	500	0	4.032201

Calculated Statistics				
Mean Server Utilization	Mean Queue Length	Mean Average Wait	Mean System	Average System Throughput

		Time in Queue	Response Time	
25	0.083333	0.166667	1.166667	1

Loss Factors					
Mean Server Utilization	Mean Wait Time in Queue	Mean System Response Time	Mean Queue Length	System Throughput	Mean Loss Factor
0.000251	0.108226	0.098896	0.02537	3.0322	0.652989

Discussion

If the loss factor for system throughput is ignored. The loss factor indeed produces a downward trend as n (the number of packets passed into the system) increases. This confirms that the simulation indeed does produce correct results when compared to the analytical model. Additionally, all the values of the tests seem to be consistent. For example: Total number of packets dropped + total number packets processed = number of packets that entered the system. This is true always during the results of the simulation.

The abnormality of the loss factor of the system throughput cannot be explained through randomness as the loss factor for all other variables is trending to 0. The only explanation can be that the throughput for the observed statistics of the simulation are calculated incorrectly. But as this calculation is done after the simulation ends, it does not affect the correctness of the simulation.

We can now proceed to the next stage of the testing.

Coursework Testing

For this stage of the testing, I will use a bounded queue and vary the parameters of the runs more widely. I will also run a test in which ro > 1 and breakdown and repairs enabled. I will not be able to compare the results of these runs with the analytical model, as the analytical model does not provide correct results for ro > 1 and breakdown and repairs enabled.

For all runs I will keep n = 1000, as it provided me with the best results when testing for correctness.

Run 1

Parameters

Lambda = 0.9

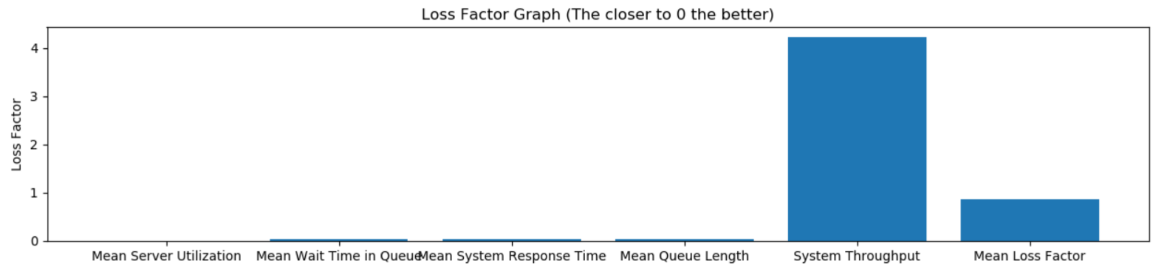
Mu = 1.1

C = 5

L = 1000

N = 1000

No breakdown and repair



server id	utilization	average service time	average wait time in queue	average queue length	number of packets processed	number of packets dropped	server uptime	server throughput
0	16.83897	0.891826	0	0	201	0	1060.806	5.277644
1	15.84335	0.831747	0	0	203	0	1063.474	5.238787
2	17.09496	0.915714	0	0	199	0	1064.934	5.35143
3	16.75535	0.902347	0	0	198	0	1066.314	5.385426
4	18.49372	0.991009	0	0	199	0	1064.981	5.351661

Observed System Statistics							
Mean Server Utilization	Mean Average Service Time	Mean Average Wait Time in Queue	Mean System Response Time	Average Mean Queue Length of server queues	Average number of packets processed	Average number of packets dropped	System Throughput
17.00527	0.906529	0	0.906529	0	200	0	5.320989

Calculated Statistics				
Mean Server Utilization	Mean Queue Length	Mean Average Wait Time in Queue	Mean System Response Time	Average System Throughput
16.36364	0.032016	0.035573	0.944664	1.1

Loss Factors					
Mean Server Utilization	Mean Wait Time in Queue	Mean System Response Time	Mean Queue Length	System Throughput	Mean Loss Factor
0.006416	0.035573	0.038135	0.032016	4.220989	0.866626

Run 2

Parameters

Lambda = 9.99

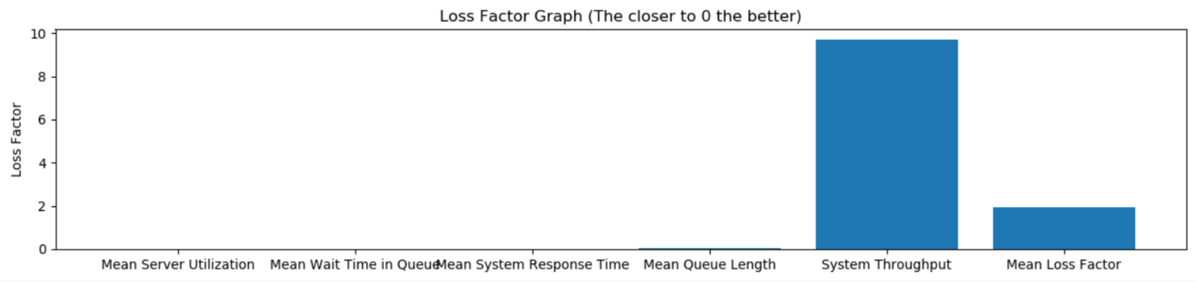
Mu = 10

C = 3

L = 100

N = 1000

No breakdown and repair



server id	utilization	average service time	average wait time in queue	average queue length	number of packets processed	number of packets dropped	server uptime	server throughput
0	31.97436	0.095051	0.005729	0.132931	331	0	98.39688	0.297272
1	32.24791	0.09634	0.007877	0.114804	331	0	98.81383	0.298531
2	32.27004	0.094419	0.011385	0.183432	338	0	98.65769	0.291887

Observed System Statistics							
Mean Server Utilization	Mean Average Service Time	Mean Average Wait Time in Queue	Mean System Response Time	Average Mean Queue Length of server queues	Average number of packets processed	Average number of packets dropped	System Throughput
32.1641	0.09527	0.00833	0.1036	0.143722	333.3333	0	0.295896

Calculated Statistics				
Mean Server Utilization	Mean Queue Length	Mean Average Wait Time in Queue	Mean System Response Time	Average System Throughput
33.3	0.16625	0.016642	0.116642	10

Loss Factors					
Mean Server Utilization	Mean Wait Time in Queue	Mean System Response Time	Mean Queue Length	System Throughput	Mean Loss Factor
0.011359	0.008311	0.013041	0.022528	9.704104	1.951869

Run 3

Parameters

Lambda = 3

Mu = 0.5

C = 1

L = 10

N = 1000

No breakdown and repair

Results

server id	utilization	average service time	average wait time in queue	average queue length	number of packets processed	number of packets dropped	server uptime	server throughput
0	99.93849	0.994526	9.269616	9.244318	352	648	350.2888	0.995139

Observed System Statistics							
Mean Server Utilization	Mean Average Service Time	Mean Average Wait Time in Queue	Mean System Response Time	Average Mean Queue Length of server queues	Average number of packets processed	Average number of packets dropped	System Throughput
99.93849	0.994526	9.269616	10.26414	9.244318	352	648	0.995139

Run 4

Parameters

Lambda = 0.9

Mu = 1.1

C = 5

L = 1000

N = 1000

Breakdowns/hour = 20

Repairs/hour = 30

Results

server id	utilization	average service time	average wait time in queue	average queue length	number of packets processed	number of packets dropped	server uptime
0	12.4336	0.89117	0	0	147	52	1045.447
1	9.120678	0.852897	0	0	113	86	1051.046
2	11.54294	0.865103	0.000577	0.007092	141	62	1055.967
3	9.247464	0.880539	0.001351	0.009009	111	85	1056.249
4	10.73738	1.071843	0	0	106	97	1058.129

Observed System Statistics							
Mean Server Utilization	Mean Average Service Time	Mean Average Wait Time in Queue	Mean System Respons e Time	Average Mean Queue Length of server queues	Average number of packet processed	Average number of packets droppe d	System Throughpu t
10.61641	0.91231	0.000386	0.912696	0.00322	123.6	76.4	8.680081

Run 5

Parameters

Lambda = 9.99

Mu = 10

C = 3

L = 100

N = 1000

Breakdowns/hour = 20

Repairs/hour = 30

Results

server id	utilizatio n	average service time	average wait time in queue	average queue length	number of packets processe d	number of packets droppe d	server uptime	server throughpu t
0	16.66342	0.10525 8	0.00247 3	0.07831 3	166	156	104.434 3	0.629123
1	26.2466	0.10316 8	0.00392 7	0.08988 8	267	67	104.866 7	0.392759
2	21.62005	0.10366 6	0.00405 2	0.10958 9	219	125	103.437 3	0.472316

Observed System Statistics							
Mean Server Utilization	Mean Average Service Time	Mean Average Wait Time in Queue	Mean System Response Time	Average Mean Queue Length of server queues	Average number of packet processed	Average number of packets dropped	System Throughput
21.51002	0.104031	0.003484	0.107515	0.092597	217.3333	116	0.498066

Run 6

Parameters

Lambda = 3

Mu = 0.5

C = 1

L = 10

N = 1000

Breakdowns/hour = 20

Repairs/hour = 30

Results

server id	utilization	average service time	average wait time in queue	average queue length	number of packets processed	number of packets dropped	server uptime	server throughput
0	99.86543	1.876666	16.59349	8.962567	187	813	350.9365	1.876666

Observed System Statistics							
Mean Server Utilization	Mean Average Service Time	Mean Average Wait Time in Queue	Mean System Response Time	Average Mean Queue Length of server queues	Average number of packet processed	Average number of packets dropped	System Throughput
99.86543	1.876666	16.59349	18.47015	8.962567	187	813	1.876666

Discussion

The first observation is that the loss factor of the throughput increases the more that mu varies from the value 1. This confirms the belief that there is an error in the calculation of the simulation's throughput value. However, since it is calculated after the simulation finished, it does not affect the correctness of the simulation.

The second noteworthy observation comes from comparing the runs with breakdown and repair disabled, to the runs with breakdown and repair disabled. Runs 1 and 4, 2 and 5, 3 and 6, all have the same parameters except for breakdown and repairs being enabled or disabled

The observed pattern is that whenever the simulation goes from no breakdown and repair, to breakdown and repair enabled:

- The server utilization experiences an appreciable drop
  This is because there are less packets reaching the server
- Mean service time increases
  This is because the design of the system is such that even packets which will eventually be dropped by the broken-down server will still be in the server queue, increasing the service time of the packets that are processed by the server
- Mean queue length decreases
  Because packets that are rejected don't experience any service time, they exit the system quickly, decreasing the mean queue length of the server queue
- Increased throughput
  Because the server breaks down, it rejects a lot of packets, which mean it does not have to be active for longer. Even though the server has to process less packets, the affect of the decreased time the server is active is greater. Since throughput = number of packets processed/time active, throughput increases

Conclusion

This semester coursework provided hands on experience into how a simulation works and how it is to be implemented and tested. The lessons learned is that simulation is a powerful tool but without an analytical model to build from or to test against, simulations can never be fully trusted. The results gotten from the simulation revealed how different factors can affect the over all performance (Queue length and repairs and breakdowns).

The unique approach due to the limitations I faced also gave me opportunity to apply ingenuity. I was forced to take an approach and a perspective that others may have not taken. I took a results-based approach, in which the most important thing was to get the simulation to provide correct results, and giving the implementation of the simulation itself a lower priority.

The future improvements that could be made for the simulation is to iterate the simulation over every event generated rather than just packet arrival. Also fix the issue of the throughput time values of the simulation. Lastly, I would not choose a project partner that will drop the course in the 2nd week of the 2 month of the semester. This caused me to submit this report almost 12 hours after the deadline.

The code of the simulation is enclosed within the file submitted and can also be found on:

https://github.com/daniyal9538/cng-476-coursework

References

- http://web.mst.edu/~gosavia/queuing_formulas.pdf
- https://www.swisseduc.ch/compscience/infotraffic/queuetraffic/docs/queuetraffic_introduction.pdf
- https://www.weibull.com/hotwire/issue14/relbasics14.htm
- https://www.mathworks.com/help/stats/wblcdf.html
- http://www.math.utah.edu/~lzhang/teaching/3070summer2009/Daily%20Updates/lectures/sec4_5.pdf