Systems modelling and simulation

Assignment 2

Author: Kemal Cikota,
Armin Rezaie Valiseh
Teacher/examiner: Mauro Caporuscio
Semester: Fall 2024
Subject: Computer Science

Abstract

Instruction for running and executing code!
The code that was written to complete this assignment can be found in the root folder. The only prerequisite needed in order to run is to have the necessary libraries installed in the VScode workspace. Theese include matplotlib, numpy, datetime and pandas. IF not already in your workspace. Use the *Pip install *desired library** to download it.

The python script called A2-CODE is the solution for this problem and the main variables to change the config of the queueing system is defined as parameters in the init method

The **raw data** for each task can be found in the 'RAW DATA' folder. This just includes all of the data which has been printed in the VScode terminal.

Contents

1 Task 1: Simulation

1.1 Description of the problem

The system description describes a web application where we have three different services that customers can chose from where the web service is hosted on a cloud-based infrastructure composed of 4 VM's where the current configuration can handle 4 requests in parallel and buffer a total of 6 requests. The important thing to note here is that the servers are "shared" between the services. This means that the system is monolithic and we can describe the system using a single M/M/c/N = M/M/4/10 queue even though the different services have different service times we have one constant arrival rate.

The problem that we wish to discover is to find wether or not, changing the configuration by adding a VM in order to handle 5 requests in parallel and buffer a total of 5 requests (M/M/5/10). We will evaluate this by estimating, for both the current and proposed system the drop rate (the rate at which requests are lost) and the system performance in terms of utilization and response time.

1.2 Plan to solve the problem

The plan is to solve the problem by first finding the service rate and arrival rate for the system and then noting down the relevant performance metrics for a M/M/c/N queue. We then use this as a reference point when designing the simulation, which was done using a python script.

Since the arrival rate was already given for the system and is set to 34 requests per minute, we didn't have to calculate it. However, for the service rate, we had to first use the service times and convert them to service rate and then get it in a "per minute" unit and then take the weighted average of the service rates by using the distributions of the requests arrival for each service.

This gives us the following service rate:
Service Rate $= \left(\frac{1}{3} \times 60\right) \times 0.2 + \left(\frac{1}{7} \times 60\right) \times 0.7 + \left(\frac{1}{12} \times 60\right) \times 0.1 = 10.499$

Now when we have everything we need, we can get the theoretical performance metrics of both of the systems, which are as follows for the M/M/4/10 queue.

$$\rho = \frac{\lambda_e}{\mu c} = 0.779$$

$$P_0 = \left[1 + \sum_{n=1}^{c} \frac{a^n}{n!} + \frac{a^c}{c!} \sum_{n=c+1}^{N} \rho^{n-c}\right]^{-1} = 0.030$$

$$P_N = \frac{a^N}{c! c^{N-c}} P_0 = 0.038$$

$$L_Q = \frac{P_0 a^c \rho}{c!(1-\rho)^2}\left[1 - \rho^{N-c} - (N-c)\rho^{N-c}(1-\rho)\right] = 1.204$$

$$\lambda_e = \lambda(1 - P_N) = 32.699$$

$$w_Q = \frac{L_Q}{\lambda_e} = 0.037$$

$$w = w_Q + \frac{1}{\mu} = 0.132$$

$$L = \lambda_e w = 4.318$$

This is the performance metrics for the M/M/5/10 queue.

$$\rho = \frac{\lambda_e}{\mu c} = 0.640$$

$$P_0 = \left[1 + \sum_{n=1}^{c} \frac{a^n}{n!} + \frac{a^c}{c!} \sum_{n=c+1}^{N} \rho^{n-c}\right]^{-1} = 0.036$$

$$P_N = \frac{a^N}{c! c^{N-c}} P_0 = 0.012$$

$$L_Q = \frac{P_0 a^c \rho}{c!(1-\rho)^2}\left[1 - \rho^{N-c} - (N-c)\rho^{N-c}(1-\rho)\right] = 0.386$$

$$\lambda_e = \lambda(1 - P_N) = 33.582$$

$$w_Q = \frac{L_Q}{\lambda_e} = 0.011$$

$$w = w_Q + \frac{1}{\mu} = 0.107$$

$$L = \lambda_e w = 3.857$$

From these metrics it is important to distinguish what is relevant to us and what isn't. The most important metrics are ρ because this describes the utilization of the system, w because it describes the mean time in system which can be used to find the response times and P_N because it describes the probability that the system is full and can therefore be used to find the drop rates. Here we also make an assumption that the requests are dropped from the system if the total queue capacity is filled.

1.3 Implementation

Before implementing the program, we defined an event graph in figure 1.1 in order to get a better understanding of how this system works in detail before attempting to implement it. This simple graph shows how the flow of arrivals increment the amount of customers present in the queue and services decrement the amount of available services and amount of customers in queue. When the customers have been serviced, the amount of available servers is once again incremented.

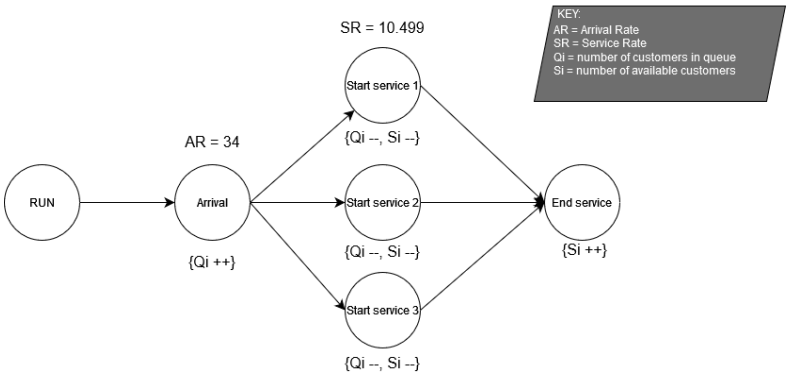

Figure 1.1: Event graph describing the system

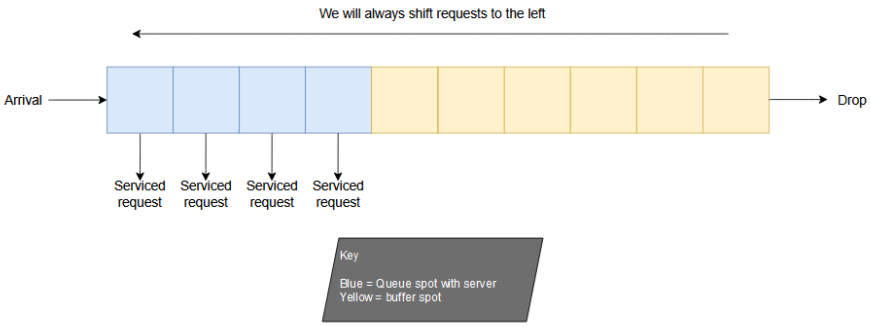

Figure 1.2: Figure describing the queueing system

Figure 1.2 shows the queue mechanism at a lower level complexity where we actually see how the queue itself is constructed. Understanding this is key to understanding how to write the logic for the code later on. We can see that the requests come from the left and in to the 4 parallel servers. From these parallel servers, requests can be served. As requests are served, they get shifted to the left because the yellow positions indicate a buffer spot. If the buffer spots get filled, requests will fail and be dropped from the other end of the queue.

When the system was implemented, we want to adhere to the same structure that we have been practicing on during the course, so we kept a lot of the notations from the various other exercises that we have completed previously.

The program starts by declaring the global variables like the clock, service rate, arrival rate and queue limit. All of these global variables are first used in a function called **"initialize()"** which is called every time the simulation is started. We also have a function called **Reset()** which is just there to reset the secondary clock which helps us simulate a steady-state system. More on this in the later chapters.

The first one is **"timing()"**. This function is responsible for handling the timing of the system and it is here where we identify the next event by finding the earliest point of time (timestamp) among the upcoming arrivals. The function advances the clock by setting the clock variable to the next scheduled event. The variable $next_event_type$ is set based on whether or not the next event is an arrival or a departure.

The system state is represented by several variables that all together provide different data for us to log. The most notable ones are however $clock$, which keeps track of the time. $serverStatus$, which is an array that keeps track of server state which can be either idle or busy. $buffer$, which is a list that holds requests waiting for a server when all servers are busy which also accurately represents the actual buffer of our queue.

The events which are defined in the event graph are mainly processed in the functions $Arrival()$, $Depart()$ and $refreshStatistics$. In $Arrival()$, a new request arrives and if there is an available server, the request is immediately assigned to a server and a server completion time is added. But if all servers are busy but there's space in the buffer, the request is queued. If both the servers and buffers are full, then the request is dropped and the counter $numoOfCustomersNotServed$ is incremented. In $Departure()$, when a server completes a request, the first entry in the $servers$ is removed. If there are requests in the buffer, then the earliest one is taken and it's delay is also calculated. the $updateTimeAvgStats()$ function is mainly responsible for keeping track of and logging all necessary data after any kind of event. Here we keep track of all of the necessary statistics like number of customers served, total queue length, server utilization, drop rate and average response time.

2 Task 2: Experiments

2.1 initial output from implementation

In this initial experiment, we just want to test-run the program that has been developed by checking if the utilization, drop rate and response time adheres to the theory. The assumptions we have made is that this system is a steady-state system which doesn't terminate because of some specified time rule, we instead collect timestamps during runtime. For this system, this is the most realistic as we want the system containing the webservices to run without stopping.

We implemented the steady-state system by running it for 365 days and using the timing method to catch the data from each day and retrieve a timestamp while still preserving the state of the system, this way we make sure that the system isn't terminating and we don't have an explicit stopping rule that resets the system after each day to preserve the steady-state properties. The only addition is that we didn't catch the data for the first 10 days in order to take the initialization bias in to account.

From running the created program for 365 days, we get the following output when running the program for the initial configuration where we have 4 VMs and a buffer with a capacity of 6 requests:

```
Average response time (w) : 0.1370080733543464
Average server utilization (rho) : 0.7853450053921138
Average drop rate: 1.022528538812785
```

Figure 2.3: Initial output from system with initial configuration

By recalling the performance metrics from chapter 1.2, we can see that the model comes very close to the theoretical calculation. The response time and server utilization which are ρ and w from theory come very close.

And we get the following output when running the proposed model with 5 VM's and a buffer with a capacity of 5 requests.

```
Average response time: 0.1079828342945613
Average server utilization: 0.642741999150526
Average drop rate: 0.2688184931506849
```

Figure 2.4: Initial output from system with proposed configuration

Here we can see that we also get very close to theory which further strengthens the initial validity of our model by just "eyeballing" it. Further validation and output analysis will be made in the next subsections.

Raw data available in outputMM410.txt and outputMM510.txt

2.2 Confidence interval and amount of repitions calculation

In order to calculate the confidence interval, we used the following formula:

$$\text{Confidence Interval} = \bar{X} \pm z \cdot \frac{\sigma}{\sqrt{N}}$$

where $\bar{X}$ is the sample mean of the data, z is the z-value which is set to 1.96 for 95% confidence, σ is the standard deviation and N is the sample size.

The variance, that was calculated in a **getMV()** method was calculated in the following way:

$$\text{Variance} = \frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N-1}$$

From this, we could calculate the amount of repetitions needed to obtain 95% confidence with a 10% error margin. This means, how many repitions do we need to do in order to say with 95% certainty that the data is aligned with the theory:

$$R = \left(\frac{z \cdot \sigma}{E}\right)^2$$

Where z is the z-value corresponding to the desired confidence level, which is once again 1.96 for 95% confidence, σ is the standard deviation and E is the desired margin of error, which we have set to 10% for this experiment.

Conclusively, we could find that the variance, confidence interval and the amount of repitions needed to obtain 95% confidence with a 10% error margin for the M/M/4/10 queue from the initial system configuration was the following:

```
Average response time: 8.223193659207517
Average server utilization: 78.53782044716252
Average drop rate: 1.022846270928463


Variance response time: 0.007934743659010415
Variance server utilization: 0.19240028453277874
Variance drop rate: 0.00430061809121455


Confidence interval response time: 8.223193659207517 +- 0.010537278470706268
Confidence interval server utilization: 78.53782044716252 +- 0.05188775275243273
Confidence interval drop rate: 1.022846270928463 +- 0.0077576011266524605


Considering 95% confidence with error margin of 10% we get
Repetitions for response time: 3.0482111240454404
Repetitions for server utilization: 73.91249330611225
Repetitions for drop rate: 1.6521254459209809
```

Figure 2.5: Conf.int, variance and amount of repititions for the initial configuration

And also for the M/M/5/10 queue which was the proposed and improved configuration of the system:

```
Average response time: 6.479473634113903
Average server utilization: 64.26455858417256
Average drop rate: 0.2692332572298327


Variance response time: 0.0022019274605953433
Variance server utilization: 0.14656426167001124
Variance drop rate: 0.000794346674037742


Confidence interval response time: 6.479473634113903 +- 0.005550901370155719
Confidence interval server utilization: 64.26455858417256 +- 0.04528726038486361
Confidence interval drop rate: 0.2692332572298327 +- 0.003334011148018263


Considering 95% confidence with error margin of 10% we get
Repetitions for response time: 0.8458924532623071
Repetitions for server utilization: 56.30412676315151
Repetitions for drop rate: 0.30515621829833894
```

Figure 2.6: Conf.int, variance and amount of repititions for the proposed configuration

2.3 Discussion on the calculated results

As we can see from the data calculated, we see that when we run the system with both configurations during 365 days and also taking initialization bias in to consideration, we need to take the roof function of each calculated number for the repetition and round it up to the upper whole number. We can see that we need 74 repetitions to be able to say with 95% confidence with 10% error margin that our data is statistically significant since we ran the simulation for 365 days. We also need 56 repetitions for the proposed configuration. Because we run the simulation for 365 days we far exceed this criteria and can say that the data we have collected is statistically significant on the 95% confidence level for both configurations. However, for the response time and drop rate, the amount of repetitions are much lower. This has been discussed with numerous other people and the code has been verified multiple times so that just makes us believe that the data actually varies that little, which tells us that the drop rate and response time are much more stable than the utilization. One thing that we did notice is that there are drastic differences in amount of repetitions based on what units we used for the response time and drop rate (i.e. if we calculate it in minutes or seconds). It is however important to notice that a low variance and low amount of repetitions does not mean that the model is inherently bad. Rather, it means that there could be a small calculation mistake somewhere.

3 Task 3: Output analysis

3.1 Sample average and variance for each performance metric

As we cab see from section 2.2, we managed to calculate the sample average and sample variance from each performance metric which is Utilization, response time and drop rate for the system with the initial and proposed server configuration. Keep in mind that for this summary i calculate the utilization in decimals and i keep the response time and drop rate in *minutes* as our arrival rate and service rate is in minutes so this will make the validation easier in the next subsection because we will compare it to theory.

Performance metrics for the initial configuration (M/M/4/10 queue):

$$\text{Average response time} = 0.13705$$

$$\text{Average server utilization} = 0.785378$$

$$\text{Average drop rate} = 1.02284$$

$$\text{variance response time} = 0.007934$$

$$\text{variance server utilization} = 0.1924$$

$$\text{Variance drop rate} = 0.0043$$

Performance metrics for the proposed configuration (M/M/5/10 queue):

$$\text{Average response time} = 0.10799$$

$$\text{Average server utilization} = 0.64264$$

$$\text{Average drop rate} = 0.269233$$

$$\text{variance response time} = 0.0022$$

$$\text{variance server utilization} = 0.1466$$

$$\text{Variance drop rate} = 0.00080$$

From this we can conclude that the performance metrics yield better result for the proposed system. The response time is lower, indicating a faster system that can handle requests faster on average. The server utilization is also significantly lower, indicating that the proposed system operates with less stress and can handle the same amount of requests easier. The drop rate is also significantly lower, this is important as this was the owners main motivation for implementing this configuration.

3.2 Output validation

From the means given in subsection 3.1, we can compare it to the values from the theory that was given in subsection 1.2 in order to get the error for each performance metric which will aid us in validating the model:

This is the errors for the performance metris for the initial M/M/4/10 queue:

$$\text{Error in response time} = \hat{w}_M = |\hat{w}_M - w_T| = |0.13705 - 0.132| = 0.00505$$

$$\text{Error in utilization} = \hat{\rho_M} = |\hat{\rho_M} - \rho_T| = |0.785378 - 0.779| = 0.006378$$

This is the errors for the performance metris for the proposed M/M/5/10 queue:

$$\text{Error in response time} = \hat{w}_M = |\hat{w}_M - w_T| = |0.10799 - 0.107| = 0.00099$$

$$\text{Error in utilization} = \hat{\rho_M} = |\hat{\rho_M} - \rho_T| = |0.64264 - 0.640| = 0.00264$$

As we can see from these errors, they are very low, which strengthens the validity of our model and also strengthens the previous hypothesis that the proposed server configuration which involves having 5 VMs and 5 queue spots for the buffer would make an improvement on the utilization, response time and also the drop rate, which were the main main motivation and argument behind this upgrade.