Distributed Dependable Systems

Assignment 1 - Queueing Theory

Benzi Matteo
matteo.benzi97@gmail.com

Firpo Tiziano
tiziano@firpo.me

April 2021

1 Introduction

The goal of this assignment was to study and analyze how M/M/1 and M/M/n queue models behaves. The obtained results must be compared with the theoretical ones and make some consideration.

2 Background

M/M/1 (Kendall notation) queue is one of the simplest models to consider jobs in a queue. There is just one server and jobs arrives and are served in a memoryless fashion. Even if it is not like a real system it is much easier to study and to derive closed-form formulas.

M/M/1 queue has an average job arriving rate λ, then jobs are served with a rate μ. To be in equilibrium the queue needs to have $\lambda < \mu$, otherwise the number of jobs in the queue will keep growing.

M/M/n instead is a multiqueue implementation of M/M/1. The job distribution in the queues can be a random choice or following an algorithm like the market model, where new jobs will be append to the shortest queue.

3 Summary

The assignment was implemented in Python 3.8.2, we have used the deque, to behaves as a FIFO queue, data structure implemented in the library "collections" for the first part. In the second part of the assignment, due to memory problems, we have made a custom implementation of the FIFO queue based on numpy.

4 M/M/1

The main factors in play in this implementation are λ and μ, they provide the probability of job arriving and job serving in the executing queue. Dependant to those two variables is the serving time and the job queue length. To simplify the operations we will normalize both variables per $\mu = 1$. Given that we can calculate the theoretical expected results.

Formulas provides us that the average queue length is:

$$\frac{\lambda}{1-\lambda}$$

Given that, using the Little's Law we can find that the mean time spent by a job in the system is:

$$\frac{1}{1-\lambda}$$

Using those formulas we can retrieve the expected theoretical results, obtaining the following queue lengths results per lambda:

lambda	expected length
0.5	1.0
0.7	2.33
0.8	4.0
0.9	9.0
0.95	18.99
0.99	98.99

Table 1: Expected mean queue lengths per arrival rate.

And the following expected serving time per lambda:

lambda	expected time
0.5	2.0
0.7	3.33
0.8	5.0
0.9	10.0
0.95	19.99
0.99	99.99

Table 2: Expected mean serving time per arrival rate.

The obtained mean lengths by our discrete event simulator are the following:

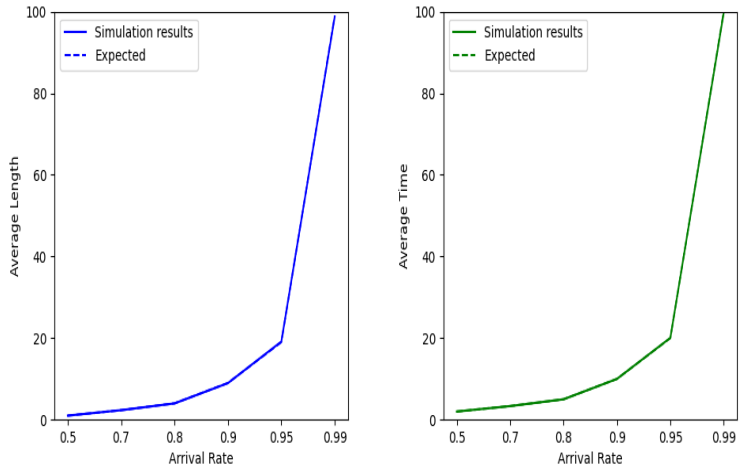
lambda	obtained length	variance
0.5	0.9959	± 0.012
0.7	2.3355	± 0.033
0.8	3.9955	± 0.0447
0.9	8.9598	± 0.0338
0.95	19.1412	± 0.2869
0.99	98.7658	± 4.6152

Table 3: Obtained mean queue lengths by the discrete event simulator.

While the obtained mean times are the following:

lambda	obtained time	variance
0.5	2.0	± 0.0023
0.7	3.3315	± 0.0057
0.8	4.9995	± 0.0159
0.9	9.9867	± 0.0469
0.95	20.0584	± 0.2846
0.99	99.7335	± 4.5366

Table 4: Calculated mean serving time by the discrete event simulator.



In a situation of equilibrium λ should be less then μ, otherwise the number of elements in the queue will keep growing.

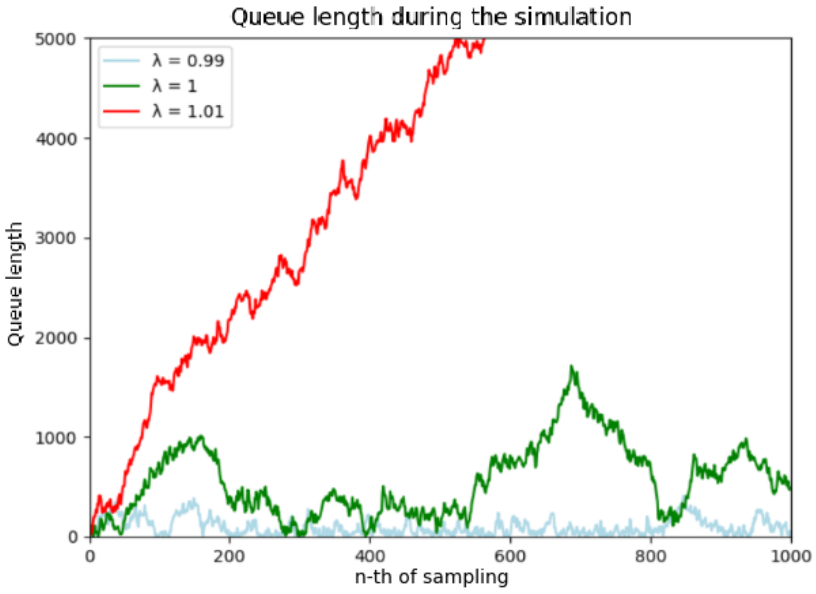


Figure 1: Queue lengths during a simulation with MAXT=1000000 and 1000 samplings

As we can see in this plot a small variation of λ can overload the queue.

5 M/M/n

In addition to λ and μ, M/M/n implementation has multiple queues. This means that the number of served jobs is increased by n times, to overcome this change with the theoretical results we need to multiply also the arrival rate. The job distribution is due to a random selection, given that we can compare the results of the simulation with table 1 and 2.

The average queue lengths of the simulator are the following:

lambda	obtained length
0.5	0.9999
0.7	2.3412
0.8	4.0134
0.9	9.0067
0.95	18.9796
0.99	99.4776

Table 5: Obtained mean queue lengths by the M/M/n simulator.

The average serving times of the simulator are the following:

lambda	obtained time
0.5	1.9994
0.7	3.3306
0.8	5.0003
0.9	10.0189
0.95	19.9785
0.99	100.5504

Table 6: Calculated mean serving time by the M/M/n simulator.

To obtain the results of these 2 tables the number of queues was set to 10.

6 M/M/n - Supermarket Model

Supposing now that we have the possibility to select more than one queue and choose the shortest between them, this algorithm is also known as supermarket model and will let us to distribute better the jobs to be served between the queues.

Mitzenmacher found that having d choices in the supermarket model provides an exponential improvement in the average time a job spends in the queue. In a system of i queues, having d choices, the expected time a job spends in the limiting supermarket system converges as t $\to \infty$ to:

$$T_d(\lambda) \equiv \sum_{i=1}^{\infty} \lambda^{\frac{d^i - d}{d-1}}$$

So we reproduced the table proposed in the paper:

Choices	λ	Simulation	Prediction
2	0.5	1.2673	1.2752
	0.7	1.6202	1.6281
	0.8	1.9585	1.9765
	0.9	2.6454	2.6589
	0.95	3.4610	3.4469
	0.99	5.9275	5.9936
3	0.5	1.1277	1.1299
	0.7	1.3634	1.3664
	0.8	1.3940	1.6001
	0.9	2.0614	2.093
	0.95	2.6137	2.5985
	0.99	4.4080	4.4058
5	0.5	1.0340	1.0384
	0.7	1.1766	1.1881
	0.8	1.3419	1.3535
	0.9	1.6714	1.6913
	0.95	2.0730	2.1165
	0.99	3.4728	3.4498

Table 7: Average Time in the Supermarket Model: 100 Queues.

Each results was calculated with a MAXT of 100'000, avoiding and the first 10'000 steps to let the queues to be in equilibrium.

Plotting a comparison on the average time spent by a job in the system between the random queue choice and the supermarket model is clear the improvement that the last method lead to:

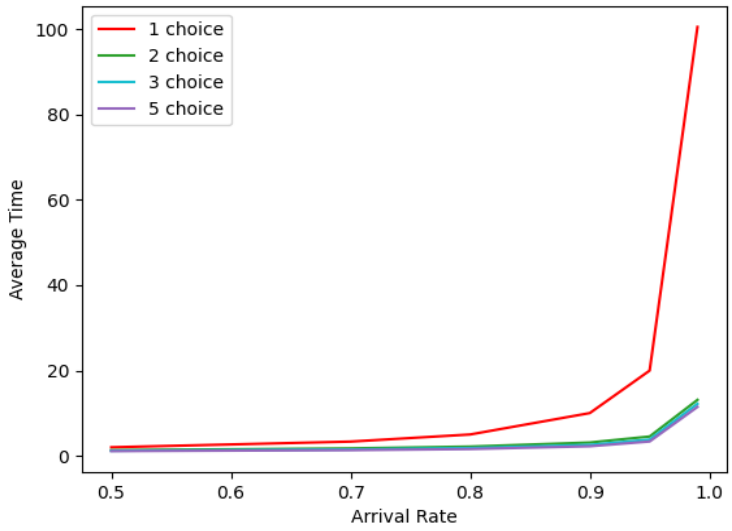


Figure 2: To generate the plot were used 10 queues, MAXT=10000000 and 10000 samplings

This demonstrate that in a simple dynamic load balancing model allowing customers to choose between the shortest of two queues leads to an exponential improvement over distributing customers uniformly at random.

Also looking at the average queue lengths the comparison highlight the improvements of the supermarket model:

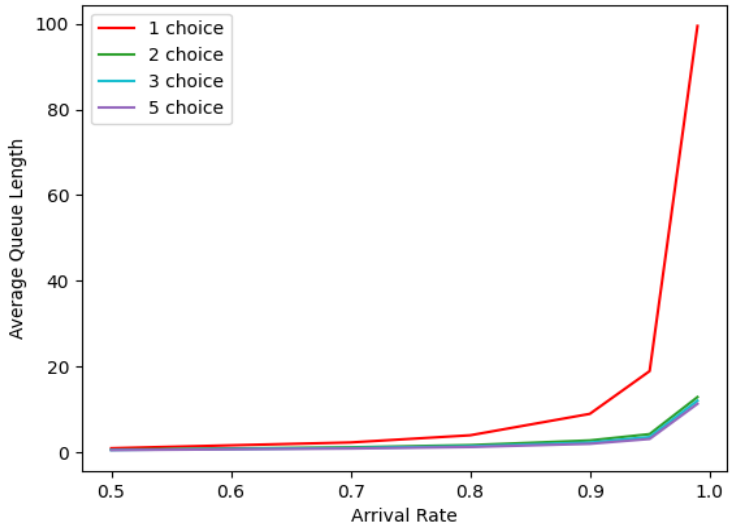


Figure 3: To generate the plot were used 10 queues, MAXT=10000000 and 10000 samplings

Choosing randomly more then one server is not always possible, since keeping updated the queue lengths among distributed servers requires more effort. A practical trade-off in these situations could be to choose between local servers, implementing a locality feature.

To implement this feature in our discrete event simulator we decided to divide the queues in groups, each one of the size up to the number of choices in play. Then choose the shortest queue among them. The results are the followings:

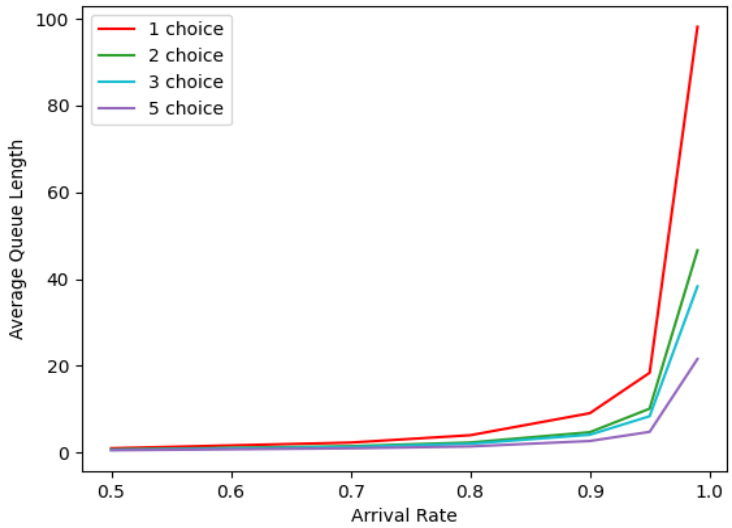


Figure 4: Average time spent by a job implementing the job selection with the locality feature

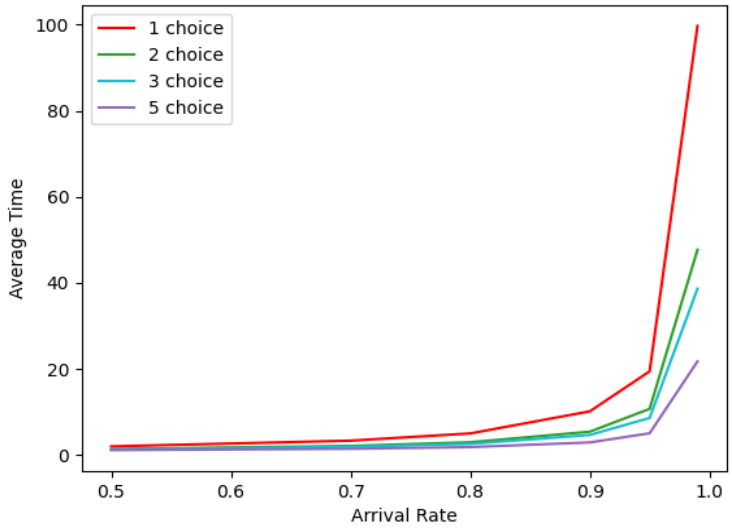


Figure 5: Average length spent by a job implementing the job selection with the locality feature

Since the group is chosen randomly this case is like to have N different independent systems, where N is number of groups. In fact, those results are the same of having a system of d queues with d choices. Anyway the global performance are improved.

In his studies, Mitzenmacher, states also that queue lengths for random queues are the same as the single-server case, while with the power of d choices, this number becomes:

$$\lambda^{\frac{d^i - d}{d-1}}$$

The results of our simulator are the followings:

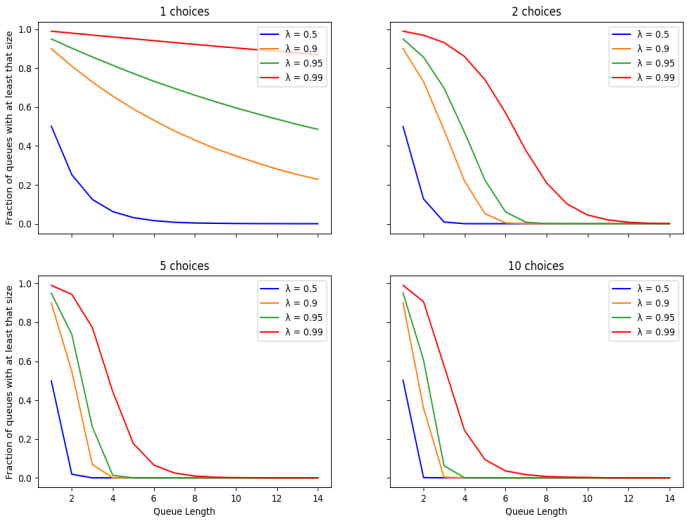


Figure 6: To generate these plots were used 100 queues, MAXT=1000000 and 1000 samplings