Homework 4 and Queing Lab

Achille Salaün

18 janvier 2016

Problem 1

Question 1

Let set $\lambda = \lambda_n + \lambda_f$, $\mathbb{E}[S_n] = \frac{10Mbit}{10Mbps} = 1s$ and $\mathbb{E}[S_f] = \frac{5Mbit}{1Mbps} = 5s$.

So, the mean transmission time overall $\mathbb{E}[S]$ is equal to :

$$\mathbb{E}[S] = \frac{\lambda_n}{\lambda}\mathbb{E}[S_n] + \frac{\lambda_f}{\lambda}\mathbb{E}[S_f] = 2,14s$$

Thus, by definition, the utilization ρ of this system is equal to :

$$\rho = \lambda\mathbb{E}[S] = 0,75$$

Question 2

Because of the FCFS scheduling, the mean queuing time for requests doesn't depend on its origin (if the request is a far or near user request). Let derive it first :

$$\mathbb{E}[T_Q] = \rho\mathbb{E}[S_E] + \mathbb{E}[N_Q]\mathbb{E}[S] = \frac{\rho}{1-\rho}\mathbb{E}[S_E]$$

We have :

$$\mathbb{E}[S_n^2] = (1 + c_{v,n}^2)\mathbb{E}[S_n]^2 = 5s^2$$
$$\mathbb{E}[S_f^2] = (1 + c_{v,f}^2)\mathbb{E}[S_f]^2 = 50s^2$$

So :

$$\mathbb{E}[S^2] = \frac{\lambda_n}{\lambda}\mathbb{E}[S_n^2] + \frac{\lambda_f}{\lambda}\mathbb{E}[S_f^2] = 17,857s^2$$

We can now compute :

$$\mathbb{E}[S_E] = \frac{\mathbb{E}[S^2]}{2\mathbb{E}[S]} = 4,167s$$

Finally, the mean queuing time for requests is :

$$\mathbb{E}[T_Q] = 12,5s$$

In conclusion, the mean download time of files to near users is $\mathbb{E}[T_n] = \mathbb{E}[T_Q] + \mathbb{E}[S_n] = 13,5s$ and the mean download time of files to far users is $\mathbb{E}[T_f] = \mathbb{E}[T_q] + \mathbb{E}[S_f] = 17,5s$.

Question 3

And so the mean download time overall is $\mathbb{E}[T] = \mathbb{E}[T_Q] + \mathbb{E}[S] = 14,64s$.

Question 4

In the case of the same queue implementing the Processor-Sharing scheduling rather the First Come First Served one, the mean download time overall would be :

$$\mathbb{E}[T] = \frac{\mathbb{E}[S]}{1-\rho} = 8,56s$$

Problem 2

From the course, we get :

(01) $\leftrightarrow$ (e) : $\mathbb{E}[T]^{M/G/1/FCFS} = \frac{\rho}{1-\rho}\mathbb{E}[S_E] + \mathbb{E}_S$

(02) $\leftrightarrow$ (c) : $\mathbb{E}[T]^{M/G/1/PS} = \frac{\mathbb{E}[S]}{1-\rho}$

(03) $\leftrightarrow$ (e) : $\mathbb{E}[T]^{M/G/1/LCFS} = \frac{\rho}{1-\rho}\mathbb{E}[S_E] + \mathbb{E}_S$

(04) $\leftrightarrow$ (c) : $\mathbb{E}[T]^{M/G/1/PLCFS} = \frac{\mathbb{E}[S]}{1-\rho}$

(05) $\leftrightarrow$ (e) : $\mathbb{E}[T]^{M/M/1/FCFS} = \frac{\rho}{1-\rho}\mathbb{E}[S_E] + \mathbb{E}_S$

(06) $\leftrightarrow$ (c) : $\mathbb{E}[T]^{M/M/1/PS} = \frac{\mathbb{E}[S]}{1-\rho}$

(07) $\leftrightarrow$ (g) : *None of the above*

(08) $\leftrightarrow$ (a) : $\rho = \lambda\mathbb{E}[S]$

(09) $\leftrightarrow$ (c) : $\mathbb{E}[B]^{M/G/1/FCFS} = \frac{\mathbb{E}[S]}{1-\rho}$

(10) $\leftrightarrow$ (c) : $\mathbb{E}[B]^{M/M/1/FCFS} = \frac{\mathbb{E}[S]}{1-\rho}$

(11) $\leftrightarrow$ (b) : $\mathbb{E}[S_E] = \frac{\mathbb{E}[S^2]}{2\mathbb{E}[S]}$

(12) $\leftrightarrow$ (b) : $\mathbb{E}[W]^{M/G/1/FCFS} = \frac{\mathbb{E}[S^2]}{2\mathbb{E}[S]}$

Queuing Lab

Overview

I did this lab thanks to a project, coded in Java, three teammates and I have realized. The goal of this project was to implement from scratch a simulator of discrete events with a scheduler. Our main concern in this project was to be as general as possible : **you should be able to design any network of M/M/1/FCFS, M/G/1/FCFS and M/G/1/SJF you want to.** However, since I focused on answering to the lab's questions, I didn't take care of, for example, how an item chooses an exit rather than an other one : it may be some mistakes despite the code run for networks more complicated than the system [Source → Queue → Pit].

I have reorganized the lab in 4 packages :

`theory :` There is the class `Theory` which contains static methods returning the theoretical expected time spent in the queue (queuing and being served), for M/M/1, M/G/1/FCFS and M/G/1/SJF queues.

`scheduler :` Here are all the classes linked to the scheduler. They allow to build networks and then make the scheduler run by using the notion of events. Using this package leads to really verbose results, explaining what happens in each event.

`schedulerNoText :` This package is a copy of the previous one. However, all the `System.out.println("...");` has been removed in order to soar the run speed of the scheduler.

`mains :` Here are the three main classes I ran. Two are really interesting : `Main` which gives a verbose history while running the scheduler for a given ρ, and `MainNoText` which has been used for getting the results presented a bit later in this report.

Scheduler

A scheduler is a list of events. An event is an object containing the following pieces of information : who ? what ? when ? from where ? to where ? Recursively, the scheduler will execute the method `nextEvent()`, that is to say : looking for the next event, calling the actor supposed to execute the event, asking to this actor to do what he has to do.

Actor

Since the actor has been asked to execute the task he has to do with `execute()`, he will first look for which kind of action he's supposed to do.

The systems we can study are items moving inside networks of cases (boxes). Both cases and items are actors.

Cases

They are the canvas of the system. Cases design a network, pointing on other cases. Thus, the list of cases pointing on me are my entrances and the list of cases I'm pointing on are my exits. A case A is pointing to a case B thanks to

the command : `A.linkExit(B) ;` Cases contains Items which moves from case to case. There are basically three main kinds of cases : sources, queues and pits :

`Source :` Typically, sources create new items. Actually there is two actions that a source can execute : `generate()` and `stop()`. The method `generate()` creates a new item, stores it in the case and plans the date when the item will go from the case following an exponential law. As soon as the item leave, a new generation begins. Thus, since the time between departures from the source is exponential : the law of arrival in the system is a Poisson law. `stop()` ends the simulation emptying the scheduler. If the sources are the only one to be able to do such an action, it's only because they are the only one to be absolutely necessary to run a simulation : thus, we can be sure that there will always be an actor able to stop the system.

Queues : Three kinds of queues have been implemented here : M/M/1, M/G/1/FCFS and M/G/1/SJF. The queue's main role is to `serve()` items. A Boolean `serving` has been implemented in the class `Case` in order to allow only one object at the same time to be served by the queue. Serving items is just defining the item which is the first in the case's list of items and then planning the moment when this item will leave the queue. After it left, the queue has been freed and so can serve again.

`Pit :` Pits get items and then destroy them executing the action `flush()`

Item

It just goes from case to case, executing `go()` : it first checks if the case it just left was a queue (if it is the case, it has to free the queue), then, after having updated its attributes, it looks where it arrives : if it is a pit, `flush()` is asked, if it's a queue and I can be served without waiting, I don't wait and ask to be served.

MainNoText

Despite I encourage you to run `Main` for a better understanding of what happens during the simulation, the tests have been done with the `MainNoText`. Both of them ran with a duration of 19E9 events (close to the highest number of events the program could run correctly, I wanted to get an accuracy as high as possible).

Question 1

Actually, every item is generated with a given time of service (98% of the items come with a 1 second service time, the other 2% have 201 seconds service time). When a queue serves an item it just gets this time and uses it for planning the moment where the item will leave the queue.

Theoretical predictions

Following the *Pollaczek-Khinchin formula* :

$$\mathbb{E}[T]^{M/G/1/FCFS} = \frac{\rho}{1-\rho}\frac{\mathbb{E}[S^2]}{2\mathbb{E}[S]} + \mathbb{E}[S]$$

where $\mathbb{E}[S] = 5s$ and $\mathbb{E}[S^2] = 809s^2$.

Experimental results

ρ	**Theory**	**Experiment**
0.1	13.995	13.999
0.2	25.240	35.225
0.3	39.701	39.671
0.4	58.924	58.933
0.5	85.930	85.900
0.6	126.38	126.35
0.7	193.60	193.77
0.8	328.51	328.60
0.9	727.2	733.1

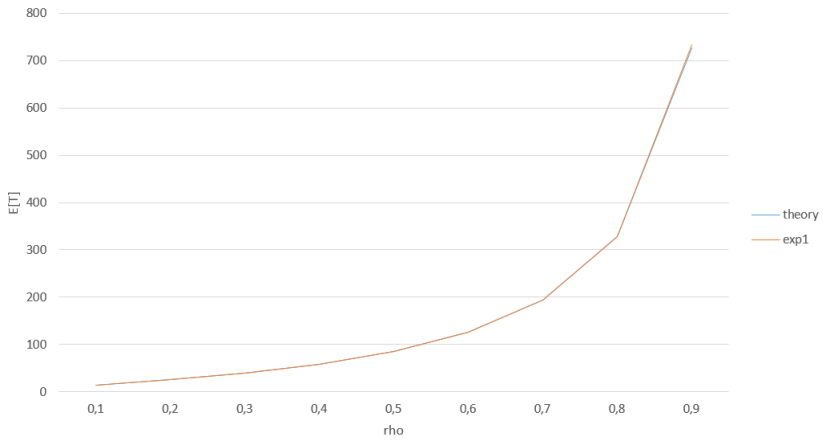

FIGURE 1 – M/G/1/FCFS

Conclusion

The simulation perfectly fits the theoretical expectations : the biggest error (for highest ρ) is here strictly smaller than 1%.

Question 2

The only difference is in the implementation of the method `getFirst()` : it splits the items between those which need small time to be served and the others which need long time to be served. The order of arrival is kept in these two sub-lists. The queue will first look for a small "jobs", if this list is empty, it looks for a big one.

Theoretical predictions

Let first derive the mean time of waiting (queuing and service) of small jobs :

$$\begin{aligned}\mathbb{E}[T_Q^s] &= \rho\mathbb{E}[S_E] + \mathbb{E}[N_Q^s]\mathbb{E}[S_s] \\ &= \rho\mathbb{E}[S_E] + \mathbb{E}[T_Q^s](0.98\lambda\mathbb{E}[S_s])\end{aligned}$$

Thus :

$$\mathbb{E}[T_Q^s](1 - 0.98\lambda\mathbb{E}[S_s]) = \rho\mathbb{E}[S_E] \tag{1}$$

Let now derive the mean time of waiting (queuing and service) of long jobs :

$$\mathbb{E}[T_Q^l] = \rho\mathbb{E}[S_E] + \mathbb{E}[N_Q]\mathbb{E}[S] + \mathbb{E}[T_Q^l](0.98\lambda\mathbb{E}[S_s])$$

So we get :

$$\mathbb{E}[T_Q^l](1 - 0.98\lambda\mathbb{E}[S_s]) = \rho\mathbb{E}[S_E] + \rho\mathbb{E}[T_Q] \tag{2}$$

In the equation (3) $\leftarrow 0.98 \times (1) + 0.02 \times (2)$:

$$\mathbb{E}[T_Q](1 - 0.98\lambda\mathbb{E}[S_s]) = \rho\mathbb{E}[S_E] + 0.02\rho\mathbb{E}[T_Q] \tag{3}$$

Let set $\rho^\star = (0.98\frac{\mathbb{E}[S_s]}{\mathbb{E}[S]} + 0.02)\rho$.
Thus, (3) is equivalent to :

$$\mathbb{E}[T_Q] = \frac{\rho}{1-\rho^\star}\frac{\mathbb{E}[S^2]}{2\mathbb{E}[S]}$$

In conclusion,

$$\mathbb{E}[T]^{M/G/1/SJF} = \frac{\rho}{1-\rho^\star}\frac{\mathbb{E}[S^2]}{2\mathbb{E}[S]} + \mathbb{E}[S]$$

where $\mathbb{E}[S] = 5s$ and $\mathbb{E}[S^2] = 809s^2$.

Experimental results

ρ	**Theory**	**Experiment**
0.1	13.269	13.267
0.2	21.911	21.926
0.3	30.952	30.999
0.4	40.420	40.578
0.5	50.348	50.775
0.6	60.768	61.631
0.7	71.718	73.662
0.8	83.240	87.879
0.9	95.380	109.210

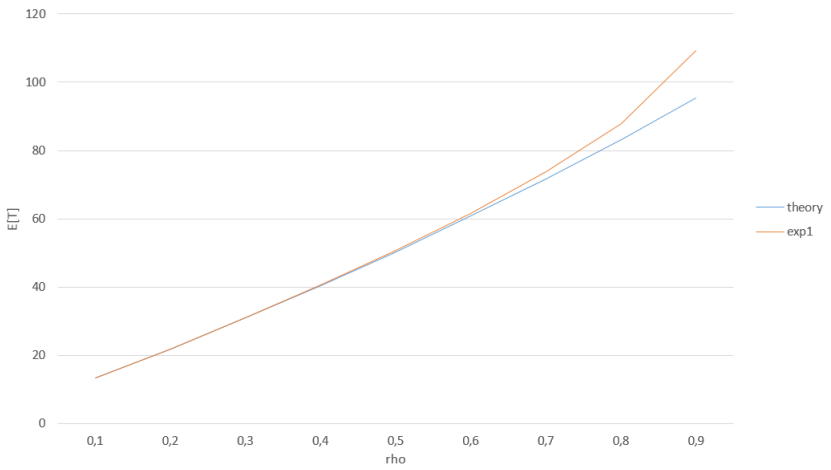

FIGURE 2 – M/G/1/SJF

Conclusion

The experiment fits here a bit less to the theory than it fitted in the first case : as soon as ρ is greater than 0.6 / 0.7, the error rate exploded from around to 1% to 14% (for highest ρ). Actually, I don't really understand why there is such an explosion : I ran twice the simulation and got "exactly" the same result. The problem is not a problem of convergence. Thus it could a problem laying either in the way I coded the program or in the way I built my theoretical predictions. First question validated all the code for a M/G/1/FCFS queue : so, if it is really a coding mistake, the error can only be in the scheduling method `getFirstItem()` (but I can't understand where). Or maybe it's a theoretical mistake. Nonetheless, why the theory would be validated for low utilization but not high one ?

However, we can notice that the SJF scheduling is far more effective than the FCFS one.

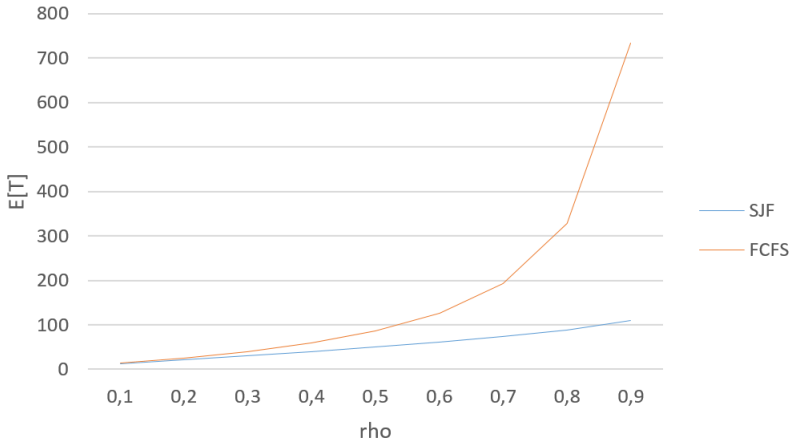

FIGURE 3 – Comparison between M/G/1/FCFS and M/G/1/SJF