LNMB Course: Algorithmic Methods in Queueing Theory (AlQT)
Lecturers: Stella Kapodistria & Rob van der Mei

Homework Assignment 1 AlQT

Deadline: April 2, 2021

General information: This is the first assignment (of two) for the course "Algorithmic Methods in Queueing Theory" (AlQT). It requires the implementation of the approaches discussed during the course and it can be made in groups of at most two persons.

For the assignment it is required to submit a report with your findings and the original source code:

You need to create a single pdf file with the answers. Since the assignment requires programming, you can use your favourite programming language or mathematical software. All original source code should be included in its original format and should be submitted together with the pdf file of the report (i.e. in the same email as the solutions of the assignment). Zip all files together (pdf file and code files) and submit one single zipped file.

The assignment solutions should be submitted by email to s.kapodistria@tue.nl with the subject "Assignment 1 for AlQT" before 23:59 on April 2, 2021.

Don't forget to present your results in a clear, concise way and to document the code. Make sure to provide sufficient information for confirmation and replication of the results. Ensure that all used sources (such as books, articles, lecture notes or slides, code, etc.) are properly referred to and cited, that all graphs/tables have a caption explaining what is depicted and all axes are labeled. Lastly, but most importantly, create a cover for the report and include the names and affiliations of all the members of the group.

We wish you good luck and above all a lot of fun!

Stella and Rob

Exercise: Performance analysis of fork-join queues

In the big data era, more and more mainstream computing infrastructures become distributively deployed, and inevitably recruit the notion of task replication to facilitate the storing and to minimise errors in the processing of large-scale datasets. This notion of replication of tasks is usually modelled using the so-called fork-join queueing systems: In a fork-join queueing system, a job (upon arrival at a control node) is forked into n replica-tasks, and each replica-task is sent to a single node to be served. Results of finished replica-tasks are summarised at a central join node. When the job arrival rate λ is high, a replica-task may have to wait for service in the corresponding queue node in a first-come-first-serve order.

For the purpose of this exercise, we only consider the following two versions of fork-join queues: The (n, k)-*purging* one removes all the remaining replica-tasks of a job from all sub-queues and service stations once the k-th replica-task is served. See Figure 1 for a visualisation of the $(3, 1)$-purging fork-join queue. Contrary to the purging version, the *non-purging* one keeps queuing and executing all remaining replica-tasks, this can be in short referred to as the (n, n) fork-join queue.

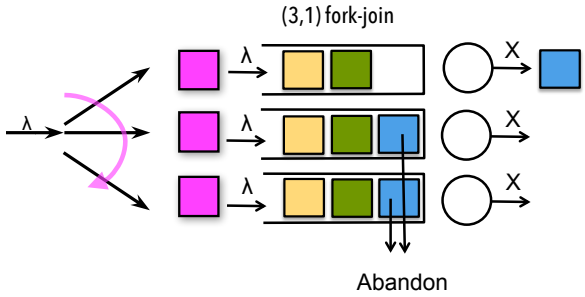

Figure 1: Visualization of the $(3, 1)$-purging fork-join queue (where X marks the random service time at the queue).

For the purpose of this exercise, consider the following basic fork-join models:

The $(2, 1)$ fork-join system: Each incoming job is forked into two replica-tasks and both replica-tasks join a separate first-come first-serve queue with a single server. Thus, there are two separate single-server queues, and one replica of a job enters the first queue and the other replica the other queue. When the service of the first one of the two replica-tasks is finished, the second one is canceled and abandons its queue immediately.

The $(2, 2)$ fork-join system: Each incoming job is forked into two replica-tasks and each replica-task joins a first-come first-serve queue with a single server. For this model, it is assumed that both replica-tasks need to be served for a job to be completed.

Assume that jobs arrive to the system according to a Poisson process at rate λ and upon arrival are forked into two replica-tasks, each joining a server that requires an exponentially distributed

service time at rate μ.

Note that the $(2,1)$ fork-join system can also be modelled as an $M/M/1$ queue with service rate 2μ. Furthermore, the $(2,2)$ fork-join system can be modelled as a two-dimensional stochastic process by considering the corresponding two queue lengths, i.e. $\{(X_1(t), X_2(t)),\ t \geq 0\}$, with $X_i(t)$ the number of customers in the queue (including the customer in service, if any) of the i-th server, $i = 1, 2$. Note that the two-dimensional stochastic process $\{(X_1(t), X_2(t)),\ t \geq 0\}$ is a Markov chain defined on $\mathbb{N}_0 \times \mathbb{N}_0$. This modelling renders the Markov chain to a homogeneous random walk in the quadrant.

Questions:

(a) Provide the stability condition, in terms of the system utilisation ρ, for the two fork-join systems.

(b) For the $(2,2)$ fork-join system briefly comment on the whether the 1) compensation approach, 2) power series approach, 3) kernel method would be successful and what are the bottlenecks you foresee.

(c) For the $(2,2)$ fork-join system, choose the most promising approach from Question (b) and compute, for a few selected values of the parameters, the equilibrium distribution, say π_{x_1,x_2}, $(x_1, x_2) \in \mathbb{N}_0 \times \mathbb{N}_0$.

(d) For the the $(2,2)$ fork-join system, write a fast (in terms of convergence) procedure to compute the expected response time (viz the expected time to complete a job from the time of arrival aka the sojourn time). Explain how/why your procedure converges fast to the correct result.

(e) Consider now, the (2,1) and the (2,2) fork-join systems with hyperexponentially service distributions, i.e., the service is with probability $p = 0.5$ exponentially distributed with rate $\mu_1 = 1/10$ and with probability $1 - p = 0.5$ is exponentially distributed with rate $\mu_2 = 1/20$. Compare these systems to the corresponding systems with exponentially distributed service times with rate $\mu = 1/15$. Which system is better? Intuitively motivate your results.