Lab Report
Transient queues

Francesco Pagano - 299266

I. Introduction

This laboratory activity was aimed at simulating long queues and removing the initial transient experienced in their delay's development. Specifically, the relation between average delay vs a fixed set of utilisation values is shown for three different distributions of the service time: deterministic (fixed at 1), exponential (generated through **numpy**) and hyperexponential (hardcoded exploiting its definition, more on this in next sections). The average delay used in the visualization has its transient removed by a custom algorithm and an automated batch means process regulates the output's accuracy.

II. Simulator's architecture

With the already presented scope in mind, the simulator is built upon an already existing single server queue implementation, which constitutes the backbone of the current simulator.

A. Assumption

The arrival time per customer has been implemented like a sampling from an exponential distribution with parameter strongly related to the service time's distribution. For the service time, as stated before, three different distributions have been used:

- Deterministic with fixed parameter 1.
- Exponential with parameter (expectation) 1.
- HyperExponential:

  1) HyperExponential Distribution: The hyperexponential distribution is a continous probability distribution with probability density function defined by:

$$f_X(x) = \sum_{i=1}^{N} f_{Y_i} p_i \tag{1}$$

where each $f_{Y_i}(x)$ is an exponential distributed random variable with parameter λ_i and p_i is the corresponding probability of sampling from the i-th distribution. This distribution features expectation and variance given by:

$$E[X] = \sum_{i=1}^{N} \frac{p_i}{\lambda_i} \tag{2}$$

$$Var[X] = \left[\sum_{i=1}^{N} \frac{p_i}{\lambda_i}\right]^2 + \sum_{i=1}^{N}\sum_{j=1}^{N} p_i p_j \left(\frac{1}{\lambda_i} - \frac{1}{\lambda_j}\right) \tag{3}$$

Simulating this variable with the **Inverse Transform** method is highly inefficient, since generating one sample requires solving a non-linear equation (code for this implementation is provided in the simulator's code but not actually used); therefore, the following algorithm has been used to generate each sample for this distribution.

```
input exponential parameter 1 lambda1
input exponential parameter 2 lambda2
input p
generate uniform sample u
if u >= p:
    return exponential(lambda1)
else:
    return exponential(lambda2)
```

Since it was required to work with a target mean and standard deviation, the following linear system has been solved to determine the distribution's parameters λ_1, λ_2 and p.

$$\begin{cases} \frac{p}{\lambda_1} + \frac{1-p}{\lambda_2} = 1 \\ \frac{2p}{\lambda_1^2} + \frac{2(1-p)}{\lambda_2^2} = 100 \\ p = const. \end{cases}$$

To maintain an unbiased sampling from one of the two distributions, p has been set to 0.5, resulting in:

$$\lambda_1 = \frac{1}{6} \tag{4}$$

$$\lambda_2 = \frac{1}{8} \tag{5}$$

Where, of course, the negative solutions have been discarded since they represent the expectation of a time-like quantity.

B. Input parameters

The simulator accepts as input the initial number of batches used to perform batch means and the maximum simulation time, respectively default at 10 and 10^5. Other parameters like the desired accuracy level, the threshold employed for the transient removal etc... have been experimentally tuned.

C. Main data structures and algorithms

The main data structure fundamental for the simulator's functioning is the Customer class, it is a simple wrapper over two main properties:

- Customer's arrival time.
- Customer's action type: arrival or departure.

from which the simulator can build all the needed quantities each time an events appears in the FES. For what concerns the main algorithms employed, the following were developed:

1) Transient removal: By plotting the delay experienced by a customer, it is clearly shown that a transient arises in the evolution process of the queue itself. The algorithm employed to remove this transient works as follows:

```
input starting array
input starting windows number
input window length (prop. to u)
input threshold (manually tuned)
divide array into #windows
for each window:
    compute min and max for each window
    if min/max > threshold:
        go to next window
        break
```

2) Batch Means: Computing batch means is a useful way of controlling the accuracy of the simulator's output and reduce the complexity of the required visualizations. The algorithm developed works as follow:

```
Run 1:
#find the right number of batches
input random threshold
input delays array
input number of batches = k
divide delays array into k batches
for batch in array batches:
   if batch_size > threshold:
       increase number of batches
       log batch width,
       break
Run 2:
#adjust threshold and perform batch means
set threshold = average batch width
recompute number of batches
#using new threshold
divide delays array in batches
compute batch's mean and 95% C.I.
```

D. *Output metrics*

The simulator outputs a picture for every transient identified for each possible utilisation and service time distribution combination possible and the output of the automated batch means algorithm. Eventually, the average delay in function of a fixed set of utilisation values (0.1, 0.2, 0.4, 0.7, 0.8, 0.9, 0.95, 0.99) along with their confidence interval is shown.

III. Results

In the following, an example or the simulator's output is presented. A likely output has been provided for each possible combination of (utilisation value, service time distribution), and they are available in the provided deliverables. Fig.1 shows that the HyperExponential distribution, featuring a high variance, isn't a suitable model for an m/m/1 queue. Fig.2 show the algorithm's output corresponding to Algorithms 1 and 2.

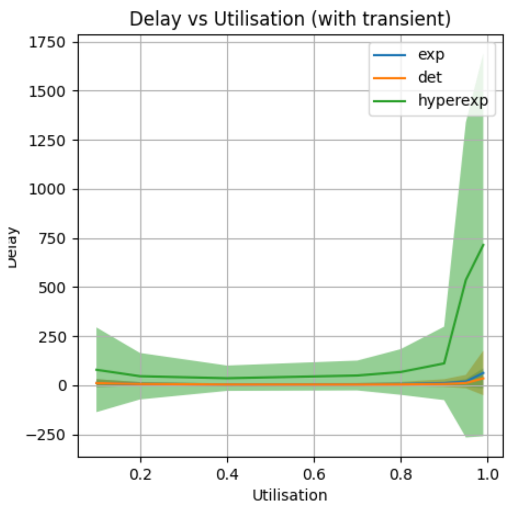
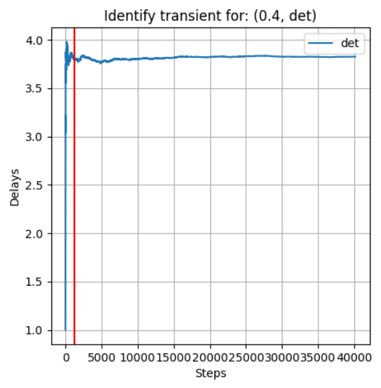
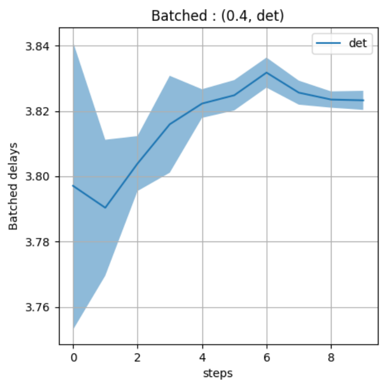


Fig. 1: Delays (not transient-cleaned) vs Utilisation for Deterministic, Exponential and HyperExponential sampling for service time.



(a) Transient identification for U=0.4 and deterministic generation of service time



(b) Batched, transient-cleaned delays for U=0.4 and deterministic generation of service time

Fig. 2: Transient identification (a) and batched output (b) of the presented custom algorithms.