Transient analysis - Confidence intervals coding

Aurora Gensale s303535

I. Problem statement

Given the queuing simulator developed in *Lab01*, our goal was to compute the average delay, plot it, and add some routines to:

- detect the end of the transient in an automated manner;
- use the *"batch means"* technique to choose the number of batches needed to achieve the desired level of accuracy.

Furthermore, we must plot the average delay as a function of utilization while considering three different service time scenarios (see more in section I-A).

A. Assumptions

- *queuing system* is identical to the one developed in *Lab01*;
- *current time* is updated according to the event time scheduled and the simulation is stopped when an arbitrary max time, we refer to it as $MAXTIME$, is reached;
- delay times are defined as *"the time the client spends waiting in queue"*;
- we refer as utilization, u, the ratio between the arrival rate and the service rate;
- *arrival times* are exponentially generated.
- We consider three generation types for *service times*:
  - exponentially distributed with $mean = 1$
  - In a deterministic way, meaning that service times are always generated as $constants = 1$
  - Distributed according to an hyperexponential-2 distribution, see I-A1 for more, with $mean = 1$ and $std = 10$.

1) Hyperexponential distribution: the hyperexponential distribution is a continuous probability distribution whose probability density function is given by "the weighted sum of i exponentially distributed random variables". Fixing $i = 2$ and to be more precise its probability density function is:

$$f_{H2} = \sum_{i=1}^{2} f_{Y_i}(x)p_i \tag{1}$$

where f_{Y_i} is a exponentially distributed random variable with rate λ_i, while p_i is the probability that $H2$ will take on the form of the exponential distribution with rate λ_i.

B. Input parameters

- u : $[0.1, 0.2, 0.4, 0.7, 0.8, 0.9, 0.95, 0.99]$
- $MAXTIME = 100.000$

II. Main algorithms

We will look at how to generate hyper-exponential service times before explaining algorithms used to detect the end of the transient and perform batch means.

A. Hyper-exponential generation

Given $X \sim H2(p, \lambda_1, \lambda_2)$ we have that

$$E[X] = \frac{p}{\lambda_1} + \frac{1-p}{\lambda_2} \tag{2}$$

$$Var[X] = \frac{2p}{\lambda_1^2} + \frac{1-p}{\lambda_1^2} - E[X]^2 \tag{3}$$

We do not have the above-mentioned parameter in our scenario, but we do have target $mean = 1$ and $std = 10$, so we must solve the following linear system.

$$\begin{cases} E[X] = 1 \\ Var[X] = 100 \\ p = 0.5 \end{cases}$$

The system would have two equations and three unknown variables if we hadn't chosen $p = 0.5$, which means that we don't prefer one exponential generation over the other. With this assumption, we arrive at two solutions:

$$\lambda_1 = \frac{1}{6}, \lambda_2 = \frac{1}{8} \tag{4}$$

$$\lambda_1 = -\frac{1}{6}, \lambda_2 = -\frac{1}{8} \tag{5}$$

We excluded the negative one since those solutions express the expectation of a time quantity, hence we are able to generate instances from X using the following algorithm 1.

Algorithm 1: Hyper-exponential instance generation

```
Data: p, λ1, λ2
Result: x
u ← ~ U(0,1)
if u ≤ p then
  | x ← ~ Exp(λ1)
else
  | x ← ~ Exp(λ2)
end
return x
```

B. Idea behind detecting the end of the transient

In simulations, the *warm-up/initial transient* is the time needed for the system to reach its *steady-state* conditions after starting from a given initial condition,which is *start with an empty queue*.

To detect it, we started dividing our data in different windows, w_i, of length $u * 1000$,where u is the utilization considered. It was discovered experimentally that computing the length

in this manner allows the algorithm to work well with all service time distributions and all u. Then, for each window, we store $\min w_i$ and $\max w_i$ and compute the normalized ratio, $nr = \frac{\min w_i}{\max w_i}$.
If nr is still bigger than a threshold means that $\min w_i$ and $\max w_i$ are still to far, hence the curve still increasing so we need to remove w_i and consider the following window w_{+1}.
Instead, if nr is quite small means that $\min w_i$ and $\max w_i$ are similar, hence the curve started to be more stable, so we have reached the $steady - state$.
1 shows an output example of the above mentioned algorithm.

Algorithm 2: End of the transient detection

Data: $w_i, \min w_i, \max w_i$
Result: delay without transient, end transient index
$nr \leftarrow \frac{\min w_i}{\max w_i}$
if $nr \geq threshold$ **then**
  delete all backwards data from this window
  break
end
return *cleaned data, idx*

C. Batch means method

Batch means is a method for having independent simulations; it is also useful for controlling output accuracy and reducing visualization complexity. Instead of utilizing various seeds, we split a single long run into non-overlapping subsequences, *batches*, and compute an estimation of the variable under study for each of them. Only once the warm-up transient has been eliminated could we carry out this practice.
Then, to determine the optimal number of batches, n, we dynamically compute the width of the 95% confidence interval, C.I., for each batch and decide whether additional batches are required. After found the ideal n, data are split in those groups and in the end batch's mean and 95% C.I. are returned.

III. Output Metrics

The end of the transient and batch means methods are performed by the simulator for each combination of u and *service times distributions*; those algorithms' outputs are then plotted.
Furthermore, the average delay (transient cleaned) in function of the utilization is shown with the corresponding 95% confidence interval.

IV. Results

In this section is reported an example of the simulator output, we show result for the following parameters combination:

- $n = 0.2$
- service time is exponentially generated

In 1 we can see the output of the transient detection algorithm while 2 shows batch means method results. Finally 3 shows the average delays (without transient) in function of u.

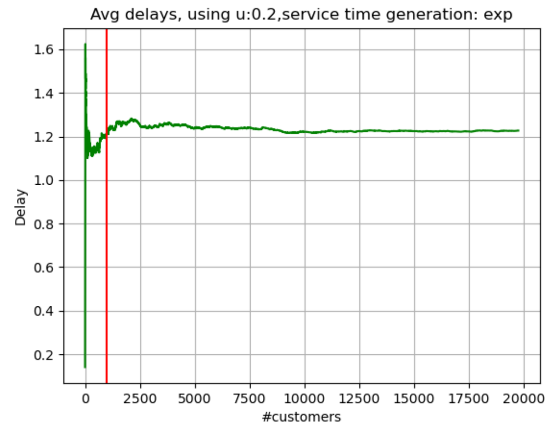

Fig. 1: End of the transient detection example with $u = 0.2$ and exponential service times

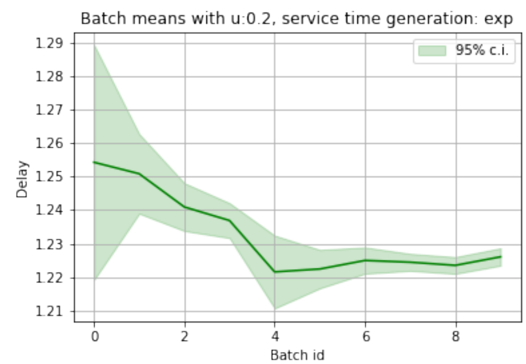

Fig. 2: Batched, transient cleaned delays with $u = 0.2$ and exponential service times

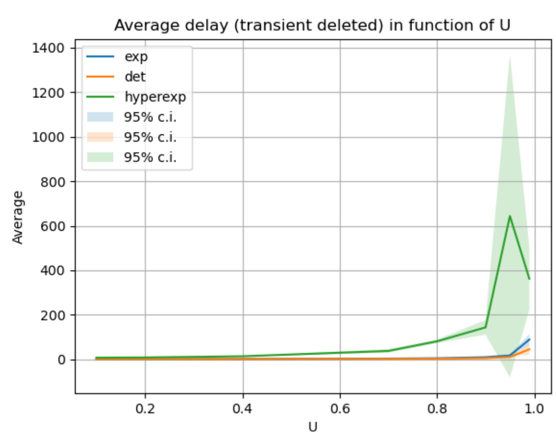

Fig. 3: Average delay (transient cleaned) in function of u, for all service times distributions, with 95% C.I.