1 Processor sharing queues

In general, if a service time S is served at constant rate $r > 0$, then the server spends S/r units of time to complete the service. For many queueing disciplines, such as FIFO, a server processes one job at a time. (And we usually keep $r = 1$ in our model for simplicity.) In some applications, however, this is neither possible nor desirable; it may make more sense to allow more than one job in service at the same time so that the server shares its resources among the jobs. *Processor sharing (PS)* at a server is a discipline for which all arriving customers enter service immediately (there is no line to wait in), but the service rate they receive is proportional to the number of customers in service: If there are n in service then each gets served at rate $1/n$. For example, if at a given time (denote this by $t = 0$) we find only customers 6 and 12 in service (denoted by C_6 and C_{12}) and C_6 has a remaining service time of 5 and C_{12} has a remaining service time of 7, then in the absence of any new arrivals, C_6 will complete service at time 10 since at rate 1/2 that is how long it takes to serve those 5 remaining units. At that point C_{12} (also having completed 5 units of its remaining 7) would have a remaining service time of 2, be in service alone, and so in the absence of any new arrivals would complete service 2 units of time later at time 12.

What complicates matters, however, is that there may be new arrivals when C_6 and C_{12} are in service causing the number in system to increase, thus slowing things down even more and so on. A typical customer, during their sojourn, shares the server with a randomly changing over time number of others, and thus their sojourn time is complicated to predict in advance. But small service times will get out quickly, which is one advantage of using PS. Under FIFO, for example, a small service time might have to wait behind very large ones, causing a huge delay for this small job and also keeping the queue level high. Web servers are a good example where PS makes sense; there tends to be great variability in service times (amount of time needed to service a web link request(s)), some jobs are small, some large. Moreover, the architecture of web servers makes it natural to share its processor, unlike a post office clerk (say), or a ATM machine which naturally work on jobs *one-at-a-time*.

1.1 *GI/GI/1 PS*

We will consider a GI/GI/1 PS single-server queue with iid service times S distributed as $G(x) = P(S \leq x)$, and iid interarrival times I distributed as $A(x) = P(I \leq x),\ x \geq 0$. To avoid having to deal with synchronization problems in our analysis (e.g., two customers departing at the same time, or an arrival and departure occurring at the same time) we will always assume that the distribution A has the property that $P(I > 0) = 1$ (equivalently $A(0) = 0$) and that either A or G has a density. For our first simulation, suppose that for some $T > 0$, we only wish to generate a copy of

$$X = \frac{1}{T}\int_0^T L(s)ds,$$

the average number in system over the first T time units, where $L(t)$ denotes the number of customers in the system at time t.

The simulation logic

T : Time at which the simulation is to stop.
t : time.
t_A : time of the next arrival.
t_S : time of the next service completion.
L : the number of customers in the system (service) at time t.
A : area up to time t, $\int_0^t L(s)ds$.

$R(1), \ldots, R(L)$: data list of the remaining service times of the L customers in service at time t *in ascending order*. $R(1)$ is thus the minimum value, and in the absence of new arrivals this service would be the next to be completed and would complete after $R(1) \times L$ units of time.

1. **Arrival is next event ($t_A < t_S$):**

   If $t_A \geq T$, then reset $A = A + L \times (T - t)$, give the output $X = A/T$ and stop. Otherwise:

   If $L = 0$, then reset $L = 1$, generate $S \sim G$ and set $R(1) = S,\ t = t_A,\ t_S = t + S$. Generate $I \sim A$, set $t_A = t + I$.

   If $L > 0$, then set $w = (t_A - t)/L$ (this is the amount of work processed on each of the L jobs), and reset $R(i) = R(i) - w,\ i = 1, \ldots, L,\ A = A + L \times (t_A - t)$;
   reset $L = L + 1,\ t = t_A$, generate $I \sim A$, set $t_A = t + I$. Generate $S \sim G$, set $R(L) = S$. Reorder the list of $R(i)$ values in ascending order $1 \leq i \leq L$. Reset $t_S = t + (R(1) \times L)$.

2. **Service completion is next event ($t_S < t_A$):**

   If $t_S \geq T$, then reset $A = A + L \times (T - t)$, give the output $X = A/T$ and stop. Otherwise:

   Reset $A = A + L \times (t_S - t)$, L=L-1;
   If $L = 0$, then reset $t = t_S,\ t_S = \infty$, whereas
   if $L > 0$, then shift the data list down by one: $R(1) = R(2), \ldots, R(L) = R(L + 1)$. Set $w = (t_S - t)/(L + 1)$, reset $R(i) = R(i) - w,\ i = 1, \ldots, L,\ t = t_S,\ t_S = t + (R(1) \times L)$.

Remark 1.1 In the special case when arrivals are Poisson (at rate λ), the M/G/1 PS model, it is known that the stationary distribution for number in system has a geometric distribution and is insensitive to the service time distribution G except through its mean $E(S) = 1/\mu$. $P(L = n) = \rho^n(1 - \rho),\ n \geq 0$, where $\rho = \lambda/\mu < 1$. Thus $l = E(L) = \rho/(1 - \rho)$ and from $l = \lambda w$, we obtain $w = E(S)/(1 - \rho)$. We would not then need to simulate this model if we wanted to estimate steady-state properties of L or mean sojourn time. But whereas the mean sojourn time is insensitive to G, the distribution of stationary sojourn time is very sensitive to G (even second moments are). Moreover, when arrivals are not Poisson, no general closed formulas exist even for l, and simulation becomes an important estimation method.

1.2 Simulating sojourn times

Suppose now that we wish to generate, for a given N, the sojourn times of the first N customers, $W_1, \ldots, W_N$. (We are referring to the first N arriving customers, $C_1, \ldots C_N$, not the first N customers *to depart*.) In this case, we need to keep track of which customers are in service all

throughout their sojourn time, unlike our previous simulation where we did not need to know who the customers in service were. Also, the simulation typically will yield the sojourn times of more than N customers because the N customers $C_1 - C_N$ might not depart until after other customers $C_{N+1}, \ldots, C_{N+k}$ (say) depart first.

We need to keep a record of $A(j)$ = the arrival time of C_j and be capable of collecting $D(j)$ = the departure time of C_j, so that we can compute $W_j = D(j) - A(j),\ j = 1, 2, \ldots N$. To this end, we need to accompany R with the list of customer indexes. $IN(i)$ = the index of $R(i)$: If $IN(i) = j$, then C_j is the customer in service with remaining service time $R(i)$. It is very important to realize that typically more than N customers will depart before we are able to collect $D(1) - D(N)$. For example, suppose $N = 1$ and C_1 arrives with a large service time $S_1 = 20$. Suppose further that 10 new customers all arrive soon after C_1 but have small service times of length 1. Then customers $C_2 - C_{11}$ will depart before C_1. In the absence of any new arrivals then, we would have collected the sojourn times of the first eleven customers when the simulation stops.

The simulation logic

N : Specifies that we want the sojourn times of the first N arriving customers.
t : time.
t_A : time of the next arrival.
N_A : The number of arrivals by time t.
t_S : time of the next service completion.
L : the number of customers in the system (service) at time t.

$R(1), \ldots, R(L)$: data list of the remaining service times of the L customers in service at time t *in ascending order*. $R(1)$ is thus the minimum value, and in the absence of new arrivals this service would be the next to be completed and would complete after $R(1) \times L$ units of time.

$IN(1), \ldots, IN(L)$: data list of the index of each of the L customers in service at time t. $IN(i)$ is the index of the customer with remaining service time $R(i),\ i = 1, \ldots, L$.

CS: the number of customers from among the first N that have completed service. As soon as $CS = N$, the simulation stops.

1. **Arrival is next event ($t_A < t_S$):**

   Reset $N_A = N_A + 1$. If $L = 0$, then reset $L = 1$, generate $S \sim G$ and set $R(1) = S,\ IN(1) = N_A,\ A(N_A) = t_A,\ t = t_A,\ t_S = t + S$. Generate $I \sim A$, set $t_A = t + I$.

   If $L > 0$, then set $w = (t_A - t)/L$ (this is the amount of work processed on each of the L jobs), and reset $R(i) = R(i) - w,\ i = 1, \ldots, L$;
   reset $L = L + 1,\ t = t_A$, generate $I \sim A$, set $t_A = t + I$. Generate $S \sim G$, and fit this new value into the correct position of the R list (shifting the others appropriately) and record its index as N_A. Reset $t_S = t + (R(1) \times L)$.

   The "fit this new value" above can be carried out as follows: Case 1: $S < R(1)$ in which case shift $R(L) = R(L-1), \ldots, R(2) = R(1)$ and $R(1) = S$; $IN(L) = IN(L -$

1), \ldots, IN(2) = IN(1) and IN(1) = N_A.
Case 2: S > R(L-1) in which case R(L) = S and IN(L) = N_A.
Case 3: R(i-1) < S < R(i) for a unique 2 \leq i \leq L-1 in which case shift R(L) = R(L-1), \ldots, R(i+1) = R(i) and R(i) = S; IN(L) = IN(L-1), \ldots, IN(i+1) = IN(i) and IN(i) = N_A.

2. **Service completion is next event ($t_S < t_A$):**

   $D(IN(1)) = t_S$.
   If $IN(1) \leq N$, then $CS = CS + 1$.
   If $CS = N$, then give the output $W(j) = A(j) - D(j),\ j = 1, \ldots, N$ and stop. Otherwise:

   $L = L - 1$.
   If $L = 0$, then reset $t = t_S,\ t_S = \infty$, whereas
   If $L > 0$, then shift the data lists down by one: $R(1) = R(2), \ldots, R(L) = R(L+1)$ and $IN(1) = IN(2), \ldots, IN(L) = IN(L+1)$;
   $w = (t_S - t)/(L+1)$, reset $R(i) = R(i) - w,\ i = 1, \ldots, L,\ t = t_S,\ t_S = t + (R(1) \times L)$.