5.2.2 A MULTI-SERVER SERVICE NODE

As another example of next-event simulation we will now consider a *multi*-server service node. The extension of this next-event simulation model to account for immediate feedback, or finite service node capacity, or a priority queue discipline is left as an exercise. This example serves three objectives.

- A multi-server service node is one natural generalization of the single-server service node.
- A multi-server service node has considerable practical and theoretical importance.
- In a next-event simulation model of a multi-server service node, the size of the event list is dictated by the number of servers and, if this number is large, the data structure used to represent the event list is important.

Definition 5.2.1 A *multi-server service node* consists of a single queue, if any, and two or more servers operating *in parallel*. At any instant in time, the state of each server will be either *busy* or *idle* and the state of the queue will be either *empty* or *not empty*. If at least one server is idle, the queue must be empty. If the queue is not empty then all the servers must be busy.

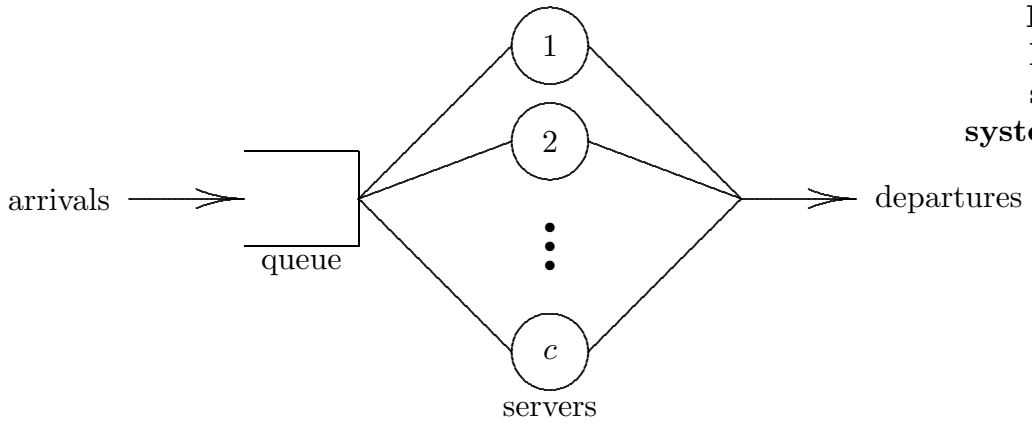

Figure 5.2.2. Multi-server service node system diagram.

Jobs arrive at the node, generally at random, seeking service. When service is provided, generally the time involved is also random. At the completion of service, jobs depart. The service node operates as follows. As each job arrives, if all servers are busy then the job enters the queue, else an available server is selected and the job enters service. As each job departs a server, if the queue is empty then the server becomes idle, else a job is selected from the queue to enter service at this server. Servers process jobs independently — they do not "team up" to process jobs more efficiently during periods of light traffic. This system configuration is popular, for example, at airport baggage check-in, banks, and roller coasters. Felt ropes or permanent dividers are often used to herd customers into queues. One advantage to this configuration is that it is impossible to get stuck behind a customer with an unusually long service time.

As in the single-server service node model, control of the queue is determined by the *queue discipline* — the algorithm used when a job is selected from the queue to enter service (see Section 1.2). The queue discipline is typically FIFO.

Server Selection Rule

Definition 5.2.2 A job may arrive to find two or more servers idle. In this case, the algorithm used to select an idle server is called the *server selection rule*.

There are several possible server selection rules. Of those listed below, the random, cyclic, and equity server selection rules are designed to achieve an equal utilization of all servers. With the other two server selection rules, typically some servers will be more heavily utilized than others.

- Random selection — select at random from the idle servers.
- Selection in order — select server 1 if idle, else select server 2 if idle, etc.
- Cyclic selection — select the first available server beginning with the successor (a circular search, if needed) of the last server engaged.
- Equity selection — select the server that has been idle longest *or* the idle server whose utilization is lowest.*
- Priority selection — choose the "best" idle server. This will require a specification from the modeler as how "best" is determined.

For the purposes of mathematical analysis, multi-server service nodes are frequently assumed to have *statistically identical, independent* servers. In this case, the server selection rule has no effect on the average performance of the service node. That is, although the utilization of the individual servers can be affected by the server selection rule, if the servers are statistically identical and independent, then the *net* utilization of the node is not affected by the server selection rule. Statistically identical servers are a convenient mathematical fiction; in a discrete-event simulation environment, if it is not appropriate then there is no need to assume that the service times are statistically identical.

States

In the queuing theory literature, the parallel servers in a multi-server service node are commonly called *service channels*. In the discussion that follows,

- the positive integer c will denote the number of servers (channels);
- the server index will be $s = 1, 2, \ldots, c$.

* There is an ambiguity in this server selection rule in that idle time can be measured from the most recent departure or from the beginning of the simulation. The modeler must specify which metric is appropriate.

As for a single-server node, the state variable $l(t)$ denotes the number of jobs in the service node at time t. For a multi-server node with distinct servers this single state variable does not provide a complete state description. If $l(t) \geq c$, then all servers are busy and $q(t) = l(t) - c$ jobs are in the queue. If $l(t) < c$, however, then for a complete state description we need to know which servers are busy and which are idle. Therefore, for $s = 1, 2, \ldots, c$ define

$$x_s(t) : \text{the number of jobs in service (0 or 1) by server } s \text{ at time } t,$$

or, equivalently, $x_s(t)$ is the state of server s at time t (with 0 denoting idle and 1 denoting busy). Finally, observe that

$$q(t) = l(t) - \sum_{s=1}^{c} x_s(t),$$

that is, the number of jobs in the queue at time t is the number of jobs in the service at time t minus the number of busy servers at time t.

Events

The $c+1$ state variables $l(t), x_1(t), x_2(t), \ldots, x_c(t)$ provide a complete state description of a multi-server service node. With a complete state description in hand we then ask what types of events can cause the state variables to change. The answer is that if the servers are distinct then there are $c + 1$ event types — either an arrival to the service node or completion of service by one of the c servers. If an *arrival* occurs at time t, then $l(t)$ is incremented by 1. Then, if $l(t) \leq c$ an idle server s is selected and the job enters service at server s (and the appropriate completion of service is scheduled), else all servers are busy and the job enters the queue. If a *completion of service* by server s occurs at time t then $l(t)$ is decremented by 1. Then, if $l(t) \geq c$ a job is selected from the queue to enter service at server s, else server s becomes idle.

The additional assumptions needed to complete the development of the next-event simulation model at the specification level are consistent with those made for the single-server model in the previous section.

- The initial state of the multi-server service node is empty and idle. Therefore, the first event must be an arrival.
- There is a terminal "close the door" time τ at which point the arrival process is turned off but the system continues operation until all jobs have been completed. Therefore, the terminal state of the multi-server node is empty and idle and the last event must be a completion of service.
- For simplicity, all servers are assumed to be independent and statistically identical. Moreover, equity selection is assumed to be the server selection rule.

All of these assumptions can be relaxed.

Event List

The event list for this next-event simulation model can be organized as an array of $c + 1$ event types indexed from 0 to c as illustrated below for the case $c = 4$.

Figure 5.2.3. Event list data structure for multi-server service node.

	t	x	
0	t	x	arrival
1	t	x	completion of service by server 1
2	t	x	completion of service by server 2
3	t	x	completion of service by server 3
4	t	x	completion of service by server 4

The `t` field in each event structure is the scheduled time of next occurrence for that event; the `x` field is the current *activity status* of the event. The status field is used in this data structure as a superior alternative to the ∞ "impossibility flag" used in the model on which programs `ssq3` and `sis3` are based. For the 0^{th} event type, `x` denotes whether the arrival process is on (1) or off (0). For the other event types, `x` denotes whether the corresponding server is busy (1) or idle (0).

An array data structure is appropriate for the event list because the size of the event list cannot exceed $c + 1$. If c is large, however, it is preferable to use a variable-length data structure like, for example, a linked-list containing events sorted by time so that the next (most imminent) event is always at the head of the list. Moreover, in this case the event list should be partitioned into busy (`event[e].x = 1`) and idle (`event[e].x = 0`) sublists. This idea is discussed in more detail in the next section.

Program msq

Program `msq` is an implementation of the next-event multi-server service node simulation model we have just developed.

- The state variable $l(t)$ is `number`.
- The state variables $x_1(t), x_2(t), \ldots, x_c(t)$ are incorporated into the event list.
- The time-integrated statistic $\int_0^t l(\theta)\, d\theta$ is `area`.
- The array named `sum` contains structures that are used to record, for each server, the sum of service times and the number served.
- The function `NextEvent` is used to search the event list to determine the index `e` of the next event.
- The function `FindOne` is used to search the event list to determine the index `s` of the available server that has been idle longest (because an equity selection server selection rule is used).