Section 5.2: Next–Event Simulation Examples

Discrete-Event Simulation: A First Course

©2006 Pearson Ed., Inc. 0-13-142917-5
Two next-event simulation models will be developed

- Simple inventory system with delivery lag
- Multi-server service node
Two changes relative to sis2

- \textit{Uniform}(0,1) lag between inventory review and order delivery
- More realistic demand model

  - Demand instances for a single item occur \textit{at random}
  - Average rate is \( \lambda \) demand instances per time interval
  - Time between demand instances is \textit{Exponential}(1/\( \lambda \))
sis2 used an aggregate demand for each time interval, generated as an *Equilikel(y)*\((10,50)\) random variate

- Aggregate demand per time interval is random
- Within an interval, time between demand instances is constant
- Example: if aggregate demand is 25, inter-demand time is 0.04

Now using *Exponential*\((1/\lambda)\) inter-demand times

- Demand is modeled as an arrival process
- Average demand per time interval is \(\lambda\)
The simulation clock is $t$ (real-valued)

- The terminal time is $\tau$ (integer-valued)
- Current inventory level is $l(t)$ (integer-valued)
- Amount of inventory on order, if any, is $o(t)$ (integer-valued)
  - Necessary due to delivery lag
- $l(t)$ and $o(t)$ provide complete state description
- Initial state is assumed to be $l(0) = S$ and $o(0) = 0$
- Terminal state is assumed to be $l(\tau) = S$ and $o(\tau) = 0$
  - Cost to bring $l(t)$ to $S$ at simulation end (with no lag) must be included in accumulated statistics
Three types of events can change the system state

- A *demand* for an item at time $t$
  - $l(t)$ decreases by 1
- An inventory *review* at integer-valued time $t$
  - If $l(t) \geq s$, then $o(t)$ becomes 0
  - If $l(t) < s$, then $o(t)$ becomes $S - l(t)$
- An *arrival* of an inventory replenishment order at time $t$
  - $l(t)$ increases by $o(t)$
  - $o(t)$ becomes 0
Algorithm 5.2.1, initialization

- Time variables used for event list:
  - $t_d$: next scheduled inventory demand
  - $t_r$: next scheduled inventory review
  - $t_a$: next scheduled inventory arrival

- $\infty$ denotes impossible events

**Initialization Step of Algorithm 5.2.1**

- $l = S$; /* initialize inventory level */
- $o = 0$; /* initialize amount on order */
- $t = 0.0$; /* initialize simulation clock */
- $t_d = \text{GetDemand}()$; /* initialize event list */
- $t_r = t + 1.0$; /* initialize event list */
- $t_a = \infty$; /* initialize event list */
Main Loop of Algorithm 5.2.1

while \((t < \tau)\) {
    \(t = \min(t_d, t_r, t_a)\); /* scan the event list */
    if \((t == t_d)\) { /* process an inventory demand */
        /--;
        \(t_d = \text{GetDemand}()\);
    }
    else if \((t == t_r)\) { /* process an inventory review */
        if \((l < s)\) {
            \(o = S - l;\)
            \(\delta = \text{GetLag}()\);
            \(t_a = t + \delta;\)
        }
        \(t_r += 1.0;\)
    }
    else { /* process an inventory arrival */
        \(l += o;\)
        \(o = 0;\)
        \(t_a = \infty;\)
    }
}
Implements Algorithm 5.2.1

- \( t.\text{demand} \), \( t.\text{review} \) and \( t.\text{arrive} \) correspond to \( t_d \), \( t_r \), \( t_a \)
- State variables inventory and order correspond to \( l(t) \) and \( o(t) \)
- \( \text{sum.hold} \) and \( \text{sum.short} \) accumulate the time-integrated holding and shortage integrals
The single-server service node is extended to support multiple servers.

- This is a natural generalization.
- Multi-server service nodes have both practical and theoretical importance.
- The event list size depends on the number of servers.
  - For large numbers of servers, the event list data structure becomes important.
- Extensions of the multi-server node (immediate feedback, finite capacity, non-FIFO) are left as exercises.
Definition 5.2.1: A multi-server service node consists of

- A single queue (if any)
- Two or more servers operating in parallel

At any instant in time,

- Each server is either busy or idle
- The queue is either empty or not empty
- If one or more servers is idle, the queue must be empty
- If the queue is not empty, all servers must be busy
When a job arrives
- If all servers are busy, the job enters the queue
- Else an idle server is selected and the job enters service

When a job departs a server
- If the queue is empty, the server becomes idle
- Else a job is removed from the queue, served by server

Servers process jobs independently
**Definition 5.2.2:** The algorithm used to select an idle server is called the *server selection rule*

- **Common selection rules**
  - Random selection: at random from the idle servers
  - Selection in order: lowest-numbered idle server
  - Cyclic selection: first available, starting after last selected (circular search may be required)
  - Equity selection: use longest-idle or lowest-utilized
  - Priority selection: choose the “best” idle server (modeler specifies how to determine “best”)

- Random, cyclic, equity: designed to achieve equal utilizations

- If servers are *statistically identical* and *independent*, the selection rule has *no effect* on average performance of the service node

- The *statistically identical* assumption is useful for mathematicians; unnecessary for discrete-event simulation
Servers in a multi-server service node are called service channels.

- \( c \) is the number of servers (channels).
- The server index is \( s = 1, 2, \ldots, c \).

\( l(t) \) denotes the number of jobs in the service node at time \( t \):
- If \( l(t) \geq c \), all servers are busy and \( q(t) = l(t) - c \).
- If \( l(t) < c \), some servers are idle.
- If servers are distinct, need to know which servers are idle.

For \( s = 1, 2, \ldots, c \) define:

\[ x_s(t) : \text{the number of jobs in service (0 or 1) at server } s \text{ at time } t. \]

The complete state description is \( l(t), x_1(t), x_2(t), \ldots, x_c(t) \).

\[ q(t) = l(t) - \sum_{s=1}^{c} x_s(t) \]
What types of events can change state variables $l(t), x_1(t), x_2(t), \ldots, x_c(t)$?

- An arrival at time $t$
  - $l(t)$ increases by 1
  - If $l(t) \leq c$, an idle server $s$ is selected, and $x_s(t)$ becomes 1
  - Else all servers are busy

- A completion of service by server $s$ at time $t$
  - $l(t)$ decreases by 1
  - If $l(t) \geq c$, a job is selected from the queue to enter service
  - Else $x_s(t)$ becomes 0

There are $c + 1$ event types
The initial state is an empty node

\[ l(0) = 0 \]
\[ x_1(0) = x_2(0) = \cdots = x_c(0) = 0 \]

The first event must be an arrival

The arrival process is turned off at time \( \tau \)

The node continues operation after time \( \tau \) until empty

The terminal state is an empty node

The last event is a completion of service

For simplicity, all servers are independent and statistically identical

Equity selection is the server selection rule

All of these assumptions can be relaxed
## Event List

A table of event types:

<table>
<thead>
<tr>
<th></th>
<th>t</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>t</td>
<td>x</td>
</tr>
<tr>
<td>1</td>
<td>t</td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>t</td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>t</td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>t</td>
<td>x</td>
</tr>
</tbody>
</table>

- Can be organized as an array of $c + 1$ event types.
- **Field t:** scheduled time of next occurrence for the event.
- **Field x:** current *activity status* of the event.
  - Superior alternative to using $\infty$ to denote impossible events.
  - For 0th event type, x denotes if arrival process is on or off.
  - For other event types, x denotes if server is busy or idle.
- For large c, consider alternate event-list structures (see section 5.3).
Program msq

Implements this next-event multi-server service node simulation model

- State variable $l(t)$ is number
- State variables $x_1(t), x_2(t), \ldots, x_c(t)$ are part of the event list
- Time-integrated statistic $\int_0^t l(\theta) d\theta$ is area
- Array sum records for each server
  - the sum of service times
  - the number served
- Function NextEvent searches the event list to find the next event
- Function FindOne searches the event list to find the longest-idle server (because equity selection is used)