Outline

• Simulation of discrete event systems
• Choosing the number of servers
• State-dependent service rate
Simulation of discrete event systems (DES)

• DES is different from the CVS (continuous variable system)
  – CVS: satellite orbit, rocket trajectory, chemical reaction process
  – DES: complex man-made system

• DES
  – System dynamic is driven by events
  – Difficult to obtain closed-form solution, use simulation to evaluate its performance
Generating random variable

• Basic: Generate a basic random number
  • Hardware generator, observe white noise
  • Pseudo random number in uniform distr. U(0,1)
    – Congruential method, widely used in computer, recursively
      \[ r_{n+1} = (kr_n + a) \mod m, \quad r_0 \text{ is called seed} \]
      » In Matlab, command: rand();

• More: Generate a general distribution
  – Inverse-Transform method
  – Accept-Reject method
Inverse-Transform method

• Obtain r.v. X obeying cdf \( F(x) \)
  – Generate \( u \sim U(0,1) \)
  – Return \( X = F^{-1}(u) \)

• Example
  – Exponential distr.
    • \( u \sim U(0,1) \)
    • \( X = -\ln(u)/\lambda \)

• Question
  – What if the closed form \( F(x) \) is not available?
    • E.g., normal distribution

We know \( \Pr\{U<F(x)\}=\Pr\{X<x\} \). Since \( \Pr\{U<F(x)\}=F(x) \), we have \( \Pr\{X<x\}=F(x) \). /endproof
Accept-Reject method

- For the case $F(x)$ is unknown, only know $f_X(x)$

- Construct a similar r.v. $Y$ s.t. $f_Y(s) > 0 \iff f_X(s) > 0$ and $F_Y(s)$ is known
  - Find a constant $c$ such that $\frac{f_X(s)}{f_Y(s)} \leq c, \forall s, \text{s.t. } f_X(s) > 0$
  - Generate a sample $s$ of $Y$
  - With prob. $\frac{f_X(s)}{cf_Y(s)}$, return $X=s$ (accept it);
    otherwise, reject $s$ and return to step 2

Li Xia, Tsinghua Univ.
Event scheduling approach

• Concentrate on events and their effect on system state
• Future events are ordered according to event time in an event list
  – (event, time) is stored in event list, for each type of event
• Simulation time is kept in clock variable
Simulation program skeleton

• Initialization
  – initialize clock to zero
  – initialize state variables, sets, and statistical counters
  – initialize event list (with known future events)

• Main loop (repeat until condition for terminating simulation is met)
  – determine the most imminent event and remove it from the event list (suppose this event is of type $i$)
  – advance clock to the time of this event
  – invoke event routine for type $i$
Simulation program skeleton

• Event routine (a separate routine for each event type)
  – update state variables and sets
  – update statistical counters
  – when required, add future events to event list

• Report generator
  • invoked when simulation is terminated
  • compute and output performance measures of interest
Example - Single Server Queue

Infinite population model
(open model)

Arrival → Queue → Server → Departure
## Example - Single Server Queue

<table>
<thead>
<tr>
<th>Entities</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>service rate</td>
</tr>
<tr>
<td>customer</td>
<td>service requirement</td>
</tr>
<tr>
<td>system</td>
<td>interarrival time</td>
</tr>
</tbody>
</table>

**Note:** service time = service requirement/service rate

### State variables:
- status of server (busy/idle)
- number of customers in system
Example - Single Server Queue

Events

arrival           start service           departure

Activities

waiting → receiving

in queue          service

Set

customers in queue
Single Server Queue Model

• Assumptions
• interarrival times are independent of system state (similarly for service times)
• interarrival times are independent of each other and have identical probability distribution (similarly for service times)
• FCFS scheduling
• system is empty at time zero
• arrival of first customer occurs after the first interarrival time
• simulation terminates when the \(m\)-th customer starts service
Single Server Queue Model

- Input parameters
  - interarrival time distribution
  - service time distribution

- Performance measures of interest
  - mean waiting time, $\bar{W}$
  - mean number of customers in system, $\bar{N}$
Single Server Queue Model

- State variables
  - $status =$ server status (busy or idle)
  - $n =$ number of customers in system

- Statistical counters
  - $nw =$ number of waiting times accumulated
  - $sw =$ sum of accumulated waiting times
  - $sa =$ sum of accumulated areas (for calculating $\bar{N}$)
  - $last\_event =$ time of last event when accumulating area
Single Server Queue Model

- **Sets**
  - `event_list`
  - `queue`

- **Event types**
  - type 1: `arrival`
  - type 2: `start_service`
  - type 3: `departure`
Single Server Queue Model

- **Initialization**
  - $clock = 0$
  - $status = idle$
  - $n = 0$
  - $nw = sw = 0$
  - $last_event = 0$
  - $sa = 0$
  - initialize $queue$ to empty
  - initialize $event\_list$ to empty
  - determine $inter\_t$, the first interarrival time
  - schedule an arrival event to occur at $clock + inter\_t$
Single Server Queue Model

- Main loop (repeat until condition for terminating simulation is met)
  - determine the most imminent event and remove it from the event list (suppose this event is of type $i$ and occurs at time $t$)
  - $clock = t$
  - $sa = sa + (clock - last_event) \ast n$
  - $last_event = clock$
  - invoke event routine for type $i$
Single Server Queue Model

*arrival* event – type 1

- determine *inter_t*, the interarrival time between the current and next arrivals
- schedule an arrival event to occur at *clock + inter_t*
- \( n = n + 1 \)
- enter arriving customer to end of *queue*, and save its time of arrival (given by *clock*)
- if *status* is idle, invoke routine for *start_service* event
Single Server Queue Model

- `start_service` event – type 2
  - remove customer from front of queue, and retrieve time of arrival (`t_arrival`)
  - `nw = nw + 1`
  - `sw = sw + (clock - t_arrival)`
  - if `nw = m` (condition for terminating simulation), exit main loop
  - `status = busy`
  - determine `serv_t`, the service time of customer
  - schedule a `departure` event to occur at `clock + serv_t`
Single Server Queue Model

- **departure event – type 3**
  - $n = n - 1$
  - $status = idle$
  - if $n > 0$, invoke event routine for `start_service` event

- **Report generator**
  - mean waiting time $\overline{W} = sw / nw$
  - mean number of customers in system $\overline{N} = sa / clock$
  - output results
Single Server Queue Model

Cj - customer j

number of customers in system

departure

start service

arrival

Li Xia, Tsinghua Univ.
Notation:

A - arrival event,  D - departure event
(Cj, x) - customer j in queue, time of arrival of this customer is x
n - number of customers in system

<table>
<thead>
<tr>
<th>clock</th>
<th>event</th>
<th>status</th>
<th>n</th>
<th>event list</th>
<th>queue</th>
<th>nw</th>
<th>sw</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>--</td>
<td>idle</td>
<td>0</td>
<td>A at 1</td>
<td>empty</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>busy</td>
<td>1</td>
<td>A at 3, D at 4</td>
<td>empty</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>busy</td>
<td>2</td>
<td>D at 4, A at 6</td>
<td>(C2, 3)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>D</td>
<td>busy</td>
<td>1</td>
<td>A at 6, D at 9</td>
<td>empty</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>busy</td>
<td>2</td>
<td>D at 9, A at 11</td>
<td>(C3, 6)</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>D</td>
<td>busy</td>
<td>1</td>
<td>D at 10, A at 11</td>
<td>empty</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>D</td>
<td>idle</td>
<td>0</td>
<td>A at 11</td>
<td>empty</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>A</td>
<td>busy</td>
<td>1</td>
<td>A at 15, D at 17</td>
<td>empty</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

mean waiting time = sw/nw = 1.0
Homework-simulation

• Use Matlab to simulate various queues
  – c entities of M/M/1
  – M/M/c
  – M/M/1 with rate times c
  – Consider different interarrival distributions
    • Erlang, $c_v=0.5$
    • Exponential, $c_v=1$
    • Hyper-exponential, $c_v=3$
Choosing the number of servers

- Operation management of queuing systems
  - Determine the service rates
  - Determine the number of servers

- Balance of service quality and operating cost
  - Service rate control problem (continuous variable)
  - Choosing the number of servers (discrete variable)
Choosing the number of servers

• For stable, we have
  \[ c = r + \Delta, \]  
  \( r \) is called \textit{offered load}

• Three approaches to determine \( c \)
  – Quality domain
    • Keep a constant traffic intensity \( \rho \)
  – Quality and efficiency domain
    • Keep a constant queuing probability, \( P_Q \leq \alpha \) (Erlang-C formula)
  – Efficiency domain
    • Keep a constant \( \Delta \)
An example

- An M/M/c queue with offered load $r=9$, $c=12$ servers, traffic intensity $\rho=0.75$
- If increase the offered load to $r=36$

<table>
<thead>
<tr>
<th>Domain</th>
<th>$c$</th>
<th>$\rho$</th>
<th>$P_Q$</th>
<th>$\Delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>12</td>
<td>0.75</td>
<td>0.266</td>
<td>3</td>
</tr>
<tr>
<td>1. Quality domain</td>
<td>48</td>
<td>0.75</td>
<td>0.037</td>
<td>12</td>
</tr>
<tr>
<td>2. Quality and efficiency domain</td>
<td>42</td>
<td>0.86</td>
<td>0.246</td>
<td>6</td>
</tr>
<tr>
<td>3. Efficiency domain</td>
<td>39</td>
<td>0.92</td>
<td>0.523</td>
<td>3</td>
</tr>
</tbody>
</table>
Comparison of 3 approaches

• Quality domain
  – More expense of server cost, low efficiency
  – $r \to \infty$, $P_c \to 0$

• Efficiency domain
  – Least expense of server cost, bad quality
  – $r \to \infty$, $P_c \to 1$

• Quality and efficiency domain
  – Balance between quality and efficiency
An approximate formula, square root law

- **Square root law**
  - The number of excess servers should increase with the square root of the offered load in order to keep the same queueing probability $P_Q$

  \[ c \approx r + \beta \sqrt{r}, \quad \text{or} \quad \Delta \approx \beta \sqrt{r} \]

  - The previous example, $r \rightarrow 4r$, $\Delta \rightarrow 2\Delta$, the 2\textsuperscript{nd} result

- **Applied to practical engineering**
  - # of staffs in a call center, scheduling of call center
Theorem of square root law

Given an M/M/c with \( r = \frac{\lambda}{\mu} \) (offered load or resource requirement), let \( c \) be the least number of servers to ensure the queueing probability \( P_Q < \alpha \), then

\[
c \approx r + \beta \sqrt{r}
\]

where \( \beta \) is the solution to the equation

\[
\frac{\beta \Phi(\beta)}{\phi(\beta)} = \frac{1-\alpha}{\alpha}
\]

where \( \phi(.) \) and \( \Phi(.) \) are PDF and CDF of a standard normal distribution (Halfin and Whitt, 1981)

<table>
<thead>
<tr>
<th>( \alpha = 0.8 )</th>
<th>( \alpha = 0.5 )</th>
<th>( \alpha = 0.2 )</th>
<th>( \alpha = 0.1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \beta = 0.173 )</td>
<td>( \beta = 0.506 )</td>
<td>( \beta = 1.06 )</td>
<td>( \beta = 1.42 )</td>
</tr>
</tbody>
</table>
Theorem of square root law
(another form)

• Consider a sequence of M/M/c queues indexed by \( n = 1, 2, ..., \) with \( n \) servers and \( r_n \),

\[
\lim_{n \to \infty} C(n, r_n) = \alpha, \quad 0 < \alpha < 1
\]

if and only if

\[
\lim_{n \to \infty} \frac{n - r_n}{\sqrt{n}} = \beta, \quad \beta > 0
\]

where \( C(c, r) \) is the Erlang-C formula, and

\[
\alpha = \frac{\phi(\beta)}{\phi(\beta) + \beta \Phi(\beta)}
\]

where \( \phi(.) \) and \( \Phi(.) \) are PDF and CDF of a standard normal distribution (Halfin and Whitt, 1981)
Application of square root law

• Specify the values of $\beta$
  – Use formula to determine $\beta$ according to $\alpha$ value
  – Approximation: $\beta$ equals $(1-\alpha)$ quantile of the standard normal distribution (Kolesar and Green, 1998)

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$\beta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8</td>
<td>0.173</td>
</tr>
<tr>
<td>0.5</td>
<td>0.506</td>
</tr>
<tr>
<td>0.2</td>
<td>1.06</td>
</tr>
<tr>
<td>0.1</td>
<td>1.42</td>
</tr>
</tbody>
</table>

• If we want to keep the queueing probability $P_Q < 20\%$, then the least # of servers should be

$$c = r + \sqrt{r}$$

A Simple Rule in Practice
State-dependent service rate

• Service rate depends on system state
  – Crowded, speed up; otherwise, low speed

• A model with 2 kinds of service rates
  – Similar analysis, balance equation, steady-state distribution
  – See pp.91-92 of Gross’ book (optional)

• Question
  – If consider the cost of service rates and reward of service completion, how to control the service rate? Called service rate control problem