Layered Queueing Network Modeling of Software Systems

Building Security System (buffering)

Murray Woodside
5201 Canal Building
Two subsystems: CCTV storage, and door access control

hope to manage up to 100 cameras

Components, shown as UML with MARTE annotations:
Door Access Scenario

1. Poisson arrivals of Users 2/s

User puts card thru reader

2. Process card, check rights

3. Open door or record alarm

Requirement: 1 s response with 95% probability

4. Write log of events
1. **Trigger camera read events**
   (the N cameras were not modeled)

2. **Put the image in a buffer, send to database**

3. **Store image in database**

Buffer operations:
- get
- release

---

This object manages the resource Buffer

Buffer operations:
- get
- release
The LQN Model

Video Capture subsystem

- acquireLoop [1.8] → VideoController
- procOneImage [1.5, 0] → AcquireProc
- alloc [0.5, 0] → BufferManager
- bufEntry → Buffer
- getImage [12, 0] → AcquireProc2
- passImage [0.9, 0] → AcquireProc2
- storeImage [3.3, 0] → StoreProc
- releaseBuf [0.5, 0] → BufMgr2
- network [0, 1] → Network (infinite)

Door Access subsystem

- User rate=0.5/sec → Users
- readCard [1, 0] → CardReader
- admit [3.9, 0.2] → AccessController
- lock [0, 500] → Lock
- alarm [0, 0] → Alarm

Shared Resources

- CPU
- BufMgr2
- NetP
- Database (10 threads)
- Disk (2 threads)
Handling of Buffering

Each buffer must be emptied before it can be used for another camera

- Thus the buffer is a resource that could have a queue, which should be modeled as the pseudo-task Buffer

Model fragment without buffer

![Diagram showing the model fragment without buffer](image1)

How the buffer pool was modeled

![Diagram showing how the buffer pool was modeled](image2)

- real task BufferManager and pseudo-task Buffer
- forwarding call to a function of BufferManager
• The operations that require holding the buffer are executed by calls from the buffer pool pseudo-task

**Buffer**

—separate from the buffer manager task!!

—including the execution of the release operation by the buffer manager

—assumes the manager has a dedicated thread for release

• releaseBuf is executed by storeImage, or bufEntry
This model illustrates

• how we can model logical resources (the buffer pool)
• the use of forwarding
• the use of second phase to improve concurrency (later)
Results #1

Table 1. Simulation results for the base case

<table>
<thead>
<tr>
<th>Ncam</th>
<th>Average Response Time</th>
<th>Normalized Utilizations</th>
<th>Prob of Missing Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cycle (sec)</td>
<td>User (sec)</td>
<td>AcqProc</td>
</tr>
<tr>
<td>10</td>
<td>0.327</td>
<td>0.127</td>
<td>0.960</td>
</tr>
<tr>
<td>20</td>
<td>0.655</td>
<td>0.138</td>
<td>0.963</td>
</tr>
<tr>
<td>30</td>
<td>0.983</td>
<td>0.133</td>
<td>0.964</td>
</tr>
<tr>
<td>40</td>
<td>1.310</td>
<td>0.129</td>
<td>0.965</td>
</tr>
</tbody>
</table>

- Base case, one buffer, so one camera at a time
- Access-control responses are fine; the event rate was kept constant at 2/s.
- Camera polling becomes too slow between 20 and 30 cameras
  - try adding more buffers.
cameras fixed at 40, vary the number of buffers $N_{Buf}$
  - disappointing: the miss probability levels out above 9% at about 7 buffers.

StoreProc is apparently the new bottleneck: try an additional thread

<table>
<thead>
<tr>
<th>$N_{Buf}$</th>
<th>Average Response Time</th>
<th>Normalized Utilizations</th>
<th>Prob of Missing Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cycle (sec)</td>
<td>User (sec)</td>
<td>AcqProc</td>
</tr>
<tr>
<td>1</td>
<td>1.309</td>
<td>0.137</td>
<td>0.965</td>
</tr>
<tr>
<td>2</td>
<td>1.016</td>
<td>0.132</td>
<td>0.975</td>
</tr>
<tr>
<td>3</td>
<td>0.941</td>
<td>0.132</td>
<td>0.980</td>
</tr>
<tr>
<td>4</td>
<td>0.911</td>
<td>0.131</td>
<td>0.983</td>
</tr>
<tr>
<td>7</td>
<td>0.879</td>
<td>0.132</td>
<td>0.986</td>
</tr>
<tr>
<td>10</td>
<td>0.872</td>
<td>0.129</td>
<td>0.987</td>
</tr>
</tbody>
</table>
### Results #3

Success!
- Two threads on StoreProc combined with 4 buffers brings the miss probability down well within spec of 1 second for each

40 cameras, Nuser = 100 doors, Nbuf = 4 buffers

<table>
<thead>
<tr>
<th>No. of Store Proc</th>
<th>Average Response Time</th>
<th>Normalized Utilizations</th>
<th>Prob of Missing Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cycle (sec)</td>
<td>User (sec)</td>
<td>AcqProc</td>
</tr>
<tr>
<td>1</td>
<td>0.911</td>
<td>0.131</td>
<td>0.983</td>
</tr>
<tr>
<td>2</td>
<td>0.756</td>
<td>0.137</td>
<td>0.946</td>
</tr>
<tr>
<td>3</td>
<td>0.743</td>
<td>0.139</td>
<td>0.932</td>
</tr>
</tbody>
</table>
The saturated resource is AppProc
  - making multiplicity = 2 allows 50 cameras

The limitation is now at AcquireProc, due to a long service time
  - the time it takes to store the buffers is limiting
  - multithreading alone is not the answer

To allow an earlier start on the next camera, we can put the calls from AcquireProc into phase 2, with multithreading
  - early reply to VideoController moves the capture on to the next camera much earlier
  - allows the concurrent phase-2 Acquire tasks to run in parallel

Other adjustments are also possible
Results #4

- By using phase 2 at AcquireProc and various multiplicities we can get a capacity of 100 cameras.
  - Even more capacity can be found with StoreProc.
- Another exploration approach: set multiplicities at inf and see if specified delays are feasible at all, and what multiplicity is used (= utilization), then work down to specified delays.

100 cameras, Nuser = 100 doors, Nbuf = 10 buffers

<table>
<thead>
<tr>
<th>Multiplicity (Acquire, Buffer, Store, App. CPU)</th>
<th>Average Response Time</th>
<th>Normalized Utilizations</th>
<th>Prob of Missing Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cycle (sec)</td>
<td>User (ms)</td>
<td>Acquire Proc</td>
</tr>
<tr>
<td>2, 4, 2, 2</td>
<td>1.250</td>
<td>0.133</td>
<td>0.988</td>
</tr>
<tr>
<td>2, 10, 6, 3</td>
<td>0.837</td>
<td>0.132</td>
<td>0.988</td>
</tr>
<tr>
<td>3, 10, 6, 3</td>
<td>0.768</td>
<td>0.134</td>
<td>0.983</td>
</tr>
</tbody>
</table>