Simple Web Server: Bottlenecks

Murray Woodside
Department of Systems and Computer Engineering
Carleton University, Ottawa, Canada

cmw@sce.carleton.ca
www.sce.carleton.ca/faculty/woodside.html
LQN for a Web server

- Server has entry demand 0.005 sec
  - can be multithreaded
- Net delay represents total net delays that block a server thread in a response

N Users with a thinking time of 5 sec.
Bottleneck in the web server...

- is a saturation point that causes it to run slowly
  - a saturated resource that limits the throughput

- in a flat resource architecture one resource is saturated, the rest are underutilized at that throughput

- in a layered architecture several resources may be saturated
  - resources above the bottleneck have increased holding times due to pushback
Throughput saturation in the web server

\( f \) (throughput)

- \( M=30 \) threads
- \( M=100 \) threads
- \( M=300, 500, 1000 \) threads

\( N \) users

line of good response delays, no saturation

...or
Bottleneck in a web server: use of threads

N Users with a thinking time of 5 sec.

<table>
<thead>
<tr>
<th>N users</th>
<th>500</th>
<th>500</th>
<th>500</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>M threads</td>
<td>10</td>
<td>30</td>
<td>100</td>
<td>inf</td>
</tr>
<tr>
<td>X server</td>
<td>.512</td>
<td>.52</td>
<td>.52</td>
<td>.52</td>
</tr>
<tr>
<td>f thruput</td>
<td>19.5</td>
<td>58.2</td>
<td>90.6</td>
<td>90.6</td>
</tr>
<tr>
<td>W user wait</td>
<td>20.6</td>
<td>3.6</td>
<td>0.51</td>
<td>0.5</td>
</tr>
<tr>
<td>U server</td>
<td>10</td>
<td>30</td>
<td>47</td>
<td>47</td>
</tr>
<tr>
<td>U net</td>
<td>9.7</td>
<td>29.1</td>
<td>45.3</td>
<td>45.3</td>
</tr>
<tr>
<td>U CPU</td>
<td>.097</td>
<td>.29</td>
<td>.45</td>
<td>.45</td>
</tr>
</tbody>
</table>
Pattern around the bottleneck

- users are always “busy” (waiting or “thinking”)
  - saturated in a sense
- server is saturated
- devices and lower servers are unsaturated

....with sufficient server threads, the server is unsaturated and the devices too... this is the ideal
Insight: Pattern for a “Software Bottleneck”

- a saturated server
- but.... a saturated server pushes back on its clients
  - the long waiting time becomes part of the client service time!!
  - result is often a cluster of saturated tasks above the bottleneck
- thus: the “real” bottleneck is the “lowest” saturated task
  - its servers (including its processor) are not saturated
  - some or all of its clients are saturated
Hourglass pattern shows saturation behaviour

**above:** tasks above the bottleneck are saturated because of pushback delays
- there must be sufficient numbers to build a queue

**below:** tasks below are unsaturated because the bottleneck throttles the load
- typically their load is spread across several resources
Recognizing the “real” bottleneck

- a saturated task with unsaturated servers and host
- look at resource utilizations
- look for a step downwards in utilization, in descending the heirarchy:
  - sat
  - sat
  - *sat: bottleneck*
  - unsat
  - unsat
“Next bottleneck”

- if the capacity of bottleneck T1 can be increased
  - then lower task T2 with the max utilization UT2 is the *next bottleneck*
  - strength measure is UT1 / UT2
  - processor or server “support”

- the potential throughput increase
  - will raise UT2 to unity and saturate T2
  - is bounded in ratio by the strength measure

- in practice the utilization of T2 may increase more rapidly with throughput, and T2 saturate at a lower throughput

- IEEE TSE paper 1995
Mitigation of a bottleneck (Peter Tregunno)

(1) provide additional resources at the bottleneck
   - for a software server, provide *multiple threads*
     - some “asynchronous server” designs provide unlimited threads
   - *replicated* servers can split the load and distribute it, but give them each a processor
   - for a processor, a *multiprocessor* (or faster CPU)

(2) reduce its service time to make it *faster*:
   - reduced host demand (tighter code)
   - reduced requests to its servers
   - parallelism, optimism
   - less blocking time (phase 1 time) at its servers

(3) divert load away from it
Use additional resources...

- a resource may be given additional (M servers)
  - multiprocessor
  - multithreaded task
- a (rough) rule of thumb for M, based on potential needs for concurrency at a task T1:
  \[ M = \min \left\{ \left( 1 + \text{sum of resources of servers of T1} \right), \left( \text{sum of clients of T1} \right) \right\} \]

- increase the capacity of the bottleneck resource
  - holding time drops, throughput increases
  - *lower* resources see more load and also more waiting
    - their utilization increases (bottleneck can move down to the “next bottleneck”)
- however, a *higher* resource may also remain saturated due to higher throughput
  - bottleneck can move up, to a destination difficult to predict.
Comments on additional resources... e.g. increasing threading levels

- Useful with a strong software bottleneck
- Potential throughput at bottleneck $\leq f_b \times B_b$
  - $f =$ throughput
  - $B =$ ratio of utilizations (relative to saturation) at the bottleneck, to its highest utilized server.
  - $B > 1$ at a bottleneck
- Optimal threading level is usually found through experiment
  - first rule of thumb is to use the sum of threads or multiplicities of its servers
  - second rule, increase multiplicity by factor $B$ (to provide the additional throughput)
- Cost is usually minimal (low overhead), unless software design is explicitly singlethreaded
Comments on replication of task & processor

- meaning, add more hardware…
  - Useful with a weak processor supported software bottleneck (threading helps strong bottlenecks)
  - Reduction in utilization of the bottleneck task proportional to $p/n$ (where $p$ is the percentage of total service time that a task spends blocked due to processor contention, and $n$ is the number of processors added)
  - Only effective when processor contention is high

- other ways to increase resource accessibility: more read access, less exclusive access
Comments on *reducing processing demands*

- ... write faster code…
- Only applicable for processor supported software bottlenecks
- The utilization gain is only proportional to the reduction in total processing demands
- For a strong server supported software bottleneck, the underlying problem is blocking, not slow software at the bottleneck.
Other ways to reduce holding time

- anticipation (prefetching)
- other optimistic operations
- parallelism in a server
- asynchronous operations
Comments on *decreasing interactions*

- for example, batching multiple requests
  - if synchronous requests can be bundled together - server still has to be the same amount of work, but $n$ times less waiting (waiting for rendezvous acceptance) required at the client
- effective when bottleneck is weak (long rendezvous delays are a product of high server utilizations, high server utilization = weak bottleneck)
Papers on the research

- “The Layered Queueing Tutorial”, available at www.layeredqueues.org
Papers (2)

- other papers on layered queueing by Perros, Kahkipuro, Menasce, and many others (see www.layeredqueues.org).