CSC2231: Making clusters faster

http://www.cs.toronto.edu/~stefan/courses/csc2231/05au

Stefan Saroiu
Department of Computer Science
University of Toronto
Administrivia

• **Next lecture: failures**
  - There are two papers assigned for reading
    • Oppenheimer’s study on causes of failures for Internet clusters
    • Intel Pittsburgh’s paper on failures on the wide-area
    • Read both!!!
    • Submit review for first paper only (Oppenheimer)
How to optimize performance
How to optimize performance

- **Step 1:** Find bottleneck in the system

- **Step 2:** Widen the bottleneck
How to optimize performance

• **Step 1: Find bottleneck in the system**
  – May be tough to find in complex/parallel systems
  – May depend on the workload
    • Scale, concurrency, popularity distribution
  – May change over time
    • Hardware trends, workload trends

• **Step 2: Widen the bottleneck**
  – Add more resources
  – Optimize current resource consumption
## Single machine Web server

<table>
<thead>
<tr>
<th>the app</th>
<th>CGI processing (if needed)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HTTP processing</td>
</tr>
<tr>
<td>threading</td>
<td>concurrency management (threads, events, select, …)</td>
</tr>
<tr>
<td>libC/runtime</td>
<td>socket abstractions</td>
</tr>
<tr>
<td>the OS</td>
<td>TCP/IP stack</td>
</tr>
<tr>
<td>the hardware</td>
<td>device drivers</td>
</tr>
<tr>
<td></td>
<td>NIC</td>
</tr>
</tbody>
</table>

- **NIC**: Network Interface Card
- **WAN**: Wide Area Network
Single machine Web server

- **the app** → CGI processing (if needed)
- **threading** → HTTP processing
- **libC/runtime** → concurrency management (threads, events, select, …)
- **the OS** →
  - socket abstractions
  - memory allocators
  - file abstractions
  - TCP/IP stack
  - VM
  - file system
  - device drivers
- **the hardware** →
  - NIC
  - CPU, $$, mem
  - WAN
  - disks

CSC2231: Internet Systems
Stefan Saroiu 2005
Packet processing path

• **1400 byte packet arrival costs on 1.7 GHz P4/Linux**
  – Device driver: 12us
  – TCP stack: 10us
  – User/kernel crossing: 1us
  – Extra copies: 0.3us

• **Max throughput:**
  – 550Mbps or roughly 10K web requests/sec
  – Upper bound (CPU is 100% utilized, nothing left for apps)

• **Probably not the bottleneck for Web servers**
Packet processing

• **Per-byte overhead:**
  – Cost scales with packet size
    • DMA between NIC/host
    • Memory copies (kernel/user space)
    • Data manipulation (checksums)
  – Solutions? Zero-copy networking, user-level networking, smart NICs

• **Per-packet overhead:**
  – Cost scaled with number of packets
    • Buffer allocation
    • Interrupt processing overhead
    • Data structure manipulation
  – Solutions? Optimize network stacks, OS architecture
Single machine Web server

- The app
  - CGI processing (if needed)
  - HTTP processing
  - Concurrency management (threads, events, select, …)
  - Socket abstractions
  - Memory allocators
  - File abstractions
  - TCP/IP stack
  - VM
  - File system

- Threading
  - Socket abstractions
  - Memory allocators
  - File abstractions

- LibC/runtime
  - Device drivers
  - NIC

- The OS
  - WAN
  - CPU, $$, mem
  - Disks

- The hardware
Single machine Web server

- The app
  - CGI processing (if needed)
  - HTTP processing
  - Concurrency management (threads, events, select, …)
  - Socket abstractions
  - Memory allocators
  - File abstractions
  - TCP/IP stack
  - VM
  - File system

- The OS
  - Device drivers

- The hardware
  - NIC
  - WAN
  - CPU, $$, mem
  - Disks

CSC2231: Internet Systems

Stefan Saroiu 2005
Socket abstractions

• **Pitfall: benchmarking on a LAN rather than WAN**
  - # of concurrent connections = f(latency, Xput)
  - State size is proportional to # of concurrent connections

• **Scaling to large number of concurrent connections**
  - initial select() was broken for long-lived connections

• **Handling long-lived, large transfers**
  - Provision socket buffers correctly
    - Only matters for high throughput connections
  - Any issues related to exceeding the 32bit TCP number space?
Single machine Web server

- **the app**: CGI processing (if needed)
- **threading**: concurrency management (threads, events, select, …)
- **libC/runtime**: socket abstractions, memory allocators, file abstractions
- **the OS**: TCP/IP stack, VM, file system, device drivers
- **the hardware**: NIC, CPU, $$, mem, disks
Single machine Web server

- The app
- HTTP processing
- CGI processing (if needed)
- Concurrency management (threads, events, select, …)
- Socket abstractions
- Memory allocators
- File abstractions
- TCP/IP stack
- VM
- File system
- Device drivers
- NIC
- WAN
- CPU, $$, mem
- Disks

CSC2231: Internet Systems
Stefan Saroiu 2005
Concurrency management

• A religious topic: threads vs. events
  – Threads
    • Easier to program
    • Easy to understand and exploit parallelism (multi-proc)
  – Events
    • Easier to program
    • Scheduling can be controlled and exploited
      – Not hidden in the thread scheduler or lock
    • Performance, scaling

• All this makes sense only…
  – If the bottleneck is due to threads/events (unlikely)
Pipeline servers: L1/L2 cache

• **Claim: instructions-per-cycle is low on servers**
  – Threads hurt l-cache performance
  – Idea: re-architect software into computational stages
    • Execute each task repetitively in a stage

• **Problems:**
  – Quite a drastic change in architecture
  – Working set size of stage must align well with l-cache size
  – Performance pay-off is minimal
    • 5-10% improvement (1 month of Moore’s law)
Single machine Web server

- **the app**
  - CGI processing (if needed)
  - HTTP processing

- **threading**
  - concurrency management (threads, events, select, …)

- **libC/runtime**
  - socket abstractions
  - memory allocators
  - file abstractions

- **the OS**
  - TCP/IP stack
  - VM
  - file system
  - device drivers

- **the hardware**
  - NIC
  - WAN
  - CPU, $$, mem
  - disks
Single machine Web server

- **the app**: CGI processing (if needed)
- **threading**: concurrency management (threads, events, select, …)
- **libC/runtime**: socket abstractions
- **the OS**: TCP/IP stack, memory allocators, file abstractions, VM, file system
- **device drivers**: NIC, CPU, $$, mem, disks
- **the hardware**: WAN

CSC2231: Internet Systems

Stefan Saroiu 2005
Memory management

• **Cache and VM performance:**
  – Memory allocation research:
    • Efficient layout to avoid VM pressure
    • Parallelize to avoid becoming bottleneck on SMPs
    • Stack layout matters also

• **Issues:**
  – This machinery is very well-hidden
    • Hard to expose or take advantage of it
Disks

• **If you move the disk arm, it will be your bottleneck**
  – Seek: 5ms
    • 16 millions cycles on a 3GHz machine
    • 500 Kilo-bytes of throughput over a Gb link

• **Ideas?**
Disks

- If you move the disk arm, it will be your bottleneck
  - Seek: 5ms
    - 16 millions cycles on a 3GHz machine
    - 500 Kilo-bytes of throughput over a Gb link

- Ideas?
  - Buy lots of memory to cache disk
  - Avoid writes, or use logging to write sequentially
  - Avoid reads, or read more data and cache (just in case)
    - Clever layout
  - Batch reads and writes
  - Buy lots of disks
Single machine Web server

- CGI processing (if needed)
- HTTP processing
- concurrency management (threads, events, select, …)
- socket abstractions
- memory allocators
- file abstractions
- TCP/IP stack
- VM
- file system
- device drivers
- NIC
- WAN
- CPU, $$, mem
- disks

CSC2231: Internet Systems

Stefan Saroiu 2005
Higher-level Issues

• **Overload management**
  – If offered load exceeds your capacity, what happens?
  – Need to reject load early, otherwise you’ll livelock
    • Admission control outside server (L4 switch)
    • Switch to polling (instead of interrupts) on high load
    • Reject early in the TCP stack

• **Differential quality of service**
  – Service only high-priority requests
Latency vs. Throughput

• **Harchol-Balter: optimizing order of request handling**
  – Network stacks and servers are “fair”
    • Each connection is processed at an equal rate
  – Not optimal if we want to minimize average latency
    • Or minimize amount of state in a server
  – Instead: process connections with SRJF
    • Doesn’t matter under light load
    • Matters as approach capacity (10x latency at 90% load)

• **Issues:**
  – How do you estimate the “length” of a connection
  – Starvation of long jobs
HTTP Mambo-Jambo

- HTTP is broken in many ways
  - Many small connections (HTTP 1.0)
    - Overhead of establishing TCP connection is bad
    - Persistent connections helped
  - Chatty, untokenized wireline protocol
    - Headers account for 5-700 bytes / object
    - Irrelevant for wired servers/clients
    - Matters more for wireless
Clusters

- **Increase performance:**
  - Replicate:
    - Load-balancing: avoid any replica from becoming bottleneck
    - Mitzenmacher:
      - State information is good enough
      - Goal: avoid worst-case (and not achieve optimal)
      - Sample two or three, pick best
  - Partitioning:
    - LARD
Discussion

- **Low-bandwidth last-hop:**
  - We know how to make server faster, but …
  - The real bottleneck is low bandwidth on the last mile
  - Solutions?
Discussion

• **Low-bandwidth last-hop:**
  – We know how to make server faster, but …
  – The real bottleneck is low bandwidth on the last mile
  – Solutions?
    • Better compression
      – Content adaptation
      – Content hashcaches
    • Latency-hiding with pipelined rendering/streaming
      – Works well for the Web
    • Latency-hiding with aggressive prefetching
      – Every bit of unused bandwidth is a missed opportunity
      – ISPs hate this
Discussion

• **Content is getting bigger**
  
  – Web: 4-6KB
  – P2P: audio 4MB, video 1GB

  – Other forms of distribution?
Discussion

• **Content is getting bigger**
  - Web: 4-6KB
  - P2P: audio 4MB, video 1GB
  - Other forms of distribution?
    • Sneaker-net
    • Satellites/TV cable/HDTV
  - Any new server issues?