A (quick) retrospect

COMPSCI210 Recitation
22th Apr 2013
Vamsi Thummala
## Latency Comparison

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache reference</td>
<td>0.5 ns</td>
<td></td>
</tr>
<tr>
<td>Branch mispredict</td>
<td>5 ns</td>
<td></td>
</tr>
<tr>
<td>L2 cache reference</td>
<td>7 ns</td>
<td>14x L1 cache</td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
<td>25 ns</td>
<td></td>
</tr>
<tr>
<td>Main memory reference</td>
<td>100 ns</td>
<td>20x L2 cache, 200x L1 cache</td>
</tr>
<tr>
<td>Compress 1K bytes with Zippy</td>
<td>3,000 ns</td>
<td></td>
</tr>
<tr>
<td>Send 1K bytes over 1 Gbps network</td>
<td>10,000 ns</td>
<td>0.01 ms</td>
</tr>
<tr>
<td>Read 4K randomly from SSD</td>
<td>150,000 ns</td>
<td>0.15 ms</td>
</tr>
<tr>
<td>Read 1 MB sequentially from memory</td>
<td>250,000 ns</td>
<td>0.25 ms</td>
</tr>
<tr>
<td>Round trip within same datacenter</td>
<td>500,000 ns</td>
<td>0.5 ms</td>
</tr>
<tr>
<td>Read 1 MB sequentially from SSD</td>
<td>1,000,000 ns</td>
<td>1 ms 4X memory</td>
</tr>
<tr>
<td>Disk seek</td>
<td>10,000,000 ns</td>
<td>10 ms 20x data center roundtrip</td>
</tr>
<tr>
<td>Read 1 MB sequentially from disk</td>
<td>20,000,000 ns</td>
<td>20 ms 80x memory, 20X SSD</td>
</tr>
<tr>
<td>Send packet CA-&gt;Netherlands-&gt;CA</td>
<td>150,000,000 ns</td>
<td>150 ms</td>
</tr>
</tbody>
</table>
Abstractions: Beauty and Chaos

- Context
- Component
- Connector
- Channel
- Event
- Entity
- Identity
- App
- Signature
- Attribute
- Label
- Principal
- Reference Monitor
- Subject
- Object
- Guard
- Service
- Module
Case Study: Unix

• Example program:
  ```
  cat compsci210.txt | wc | mail -s "word count" chase@cs.duke.edu
  ```

• Component: Executable program

• Context: Process that executes the component

• Connector: Pipes

• In general, an OS:
  – Sets up the context
  – Enforces isolation
  – Mediates interaction
Case Study: Unix protection

- Excerpt from “Notes on Security”:

  The Unix example exposes some principles that generalize to other systems. In general, all of the OS platforms we consider execute programs (or components, or modules) in processes (or some other protected context, or sandbox, or protection domain) on nodes linked by communication networks. A platform's protection system labels each running program context with attributes representing “who it is”, and uses these labels to govern its interactions with the outside world.
More on Protection

<table>
<thead>
<tr>
<th>Principal may do</th>
<th>Operation</th>
<th>on</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chase</td>
<td>Read</td>
<td>dFile</td>
<td></td>
</tr>
<tr>
<td>Alice</td>
<td>Pay invoice 4325</td>
<td></td>
<td>Account Q34</td>
</tr>
<tr>
<td>Bob</td>
<td>Fire three rounds</td>
<td></td>
<td>Bow gun</td>
</tr>
</tbody>
</table>

Authentication: Who sent a message?
Authorization: Who is trusted?
- Principal: Abstraction of “who”
- People: Chase, Alice
- Services: DeFiler

Principles for Computer System Design, Turing Award Lecture, 1983
Case Study: Android

- What is a component?
  - Types of components?
- What is an App?
- What is a Binder service?
- What is a Zygote?
  - Why does Android context needs just a fork() but not exec()?
- How does Android protection differs from Unix?
Prof. Chase slides
Concurrency

- Mutual exclusion
  - Lock/mutex; too much milk
- Monitor
  - CV + mutex; scheduling threads; ping-pong
- Semaphore
  - Numeric resources; producer-consumer soda example
- EventBarrier
  - Scheduling in phases/batches; Elevator

- Implement one primitive in terms of the other
  - E.g., Implement a Semaphore using only a monitor
Performance

- Single node OS
  - Latency/Response time
  - Throughput

- Internet Scale systems
  - Consistency
  - Availability
  - Partition Tolerance
  - Incremental scalability
Servers Under Stress

- Ideal
- Overload
- Thrashing
- Collapse

Response rate (throughput)

Request arrival rate (offered load)

saturation

Response time

[ Von Behren ]
Decompose service into **stages** separated by **queues**

- Each stage performs a subset of request processing
- Stages internally event-driven, typically nonblocking
- Queues introduce execution boundary for isolation and conditioning

Each stage contains a **thread pool** to drive stage execution

- However, threads are not exposed to applications
- Dynamic control grows/shrinks thread pools with demand
  
  ▶️ *Stages may block if necessary*

Best of both threads and events:

- Programmability of threads with explicit flow of events
Crypto: Concept checkers

• What is the basic assumption that cryptography relies on?
• What is a hash/fingerprint/digest?
• What is a digital signature?
• Symmetric vs Asymmetric crypto
• What is a nonce?
• What is a security/treat model?
• Type of attacks and defenses
### Response Time Distribution - 1024 Clients

- **Graph Description**
  - Shown are response time distributions for Apache, Flash, and SEDA systems.
  - Apache has 150 connections, Flash has 506 connections, and SEDA has 1024 connections.
  - Long tail in Apache & Flash due to TCP SYN backoff.
  - Note log scale.

<table>
<thead>
<tr>
<th></th>
<th>SEDA</th>
<th>Flash</th>
<th>Apache</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean RT</td>
<td>547 ms</td>
<td>665 ms</td>
<td>475 ms</td>
</tr>
<tr>
<td>Max RT</td>
<td>3.8 sec</td>
<td>37 sec</td>
<td>1.7 minutes</td>
</tr>
</tbody>
</table>

- **Key Points**
  - **SEDA yields predictable performance** - Apache and Flash are very unfair.
  - "Unlucky" clients see long TCP retransmit backoff times.
  - Everyone is "unlucky": multiple HTTP requests to load one page!
80% of the requests have response time $r$ with $x_1 < r < x_2$.

“Tail” of 10% of requests with response time $r > x_2$.

90% quantile

50% median

10% quantile

$x_1$ $x_2$

What’s the mean $r$?

A few requests have very long response times.

Understand how the mean (average) response time can be misleading.
SED A Lessons

• Means/averages are almost never useful: you have to look at the distribution.
• Pay attention to quantile response time.
• All servers must manage overload.
• Long response time tails can occur under overload, and that is bad.
• A staged structure with multiple components separated by queues can help manage performance.
• The staged structure can also help to manage concurrency and simplify locking.
Fischer-Lynch-Patterson (1985)

• No consensus can be guaranteed in an asynchronous system in the presence of failures.
• **Intuition**: a “failed” process may just be slow, and can rise from the dead at exactly the wrong time.
• Consensus **may** occur recognizably, rarely or often.

Network partition  
Split brain
CA: available, and consistent, unless there is a partition.

AP: a reachable replica provides service even in a partition, but may be inconsistent.

CP: always consistent, even in a partition, but a reachable replica may deny service if it is unable to agree with the others (e.g., quorum).

C-A-P choose two
Coordination in Distributed Systems

• Master coordinates, dictates consensus
  – e.g., lock service
  – Also called “primary”

• Remaining consensus problem: who is the master?
  – Master itself might fail or be isolated by a network partition.
  – Requires a high-powered distributed consensus algorithm (Paxos).