Scalability, Performance & Caching

Noah Mendelsohn
Tufts University
Email: noah@cs.tufts.edu
Web: http://www.cs.tufts.edu/~noah
Goals

- Explore some general principles of performance, scalability and caching
- Explore key issues relating to performance and scalability of the Web
Performance Concepts and Terminology
Performance, Scalability, Availability, Reliability

- **Performance**
  - Get a lot done quickly
  - Preferably a low cost

- **Scalability**
  - Low barriers to growth
  - A scalable system isn’t necessarily fast…
  - …but it can grow without slowing down

- **Availability**
  - Always there when you need it

- **Reliability**
  - Never does the wrong thing, never loses or corrupts data
Throughput vs. Response Time vs. Latency

- **Throughput**: the aggregate rate at which a system does work.
- **Response time**: the time to get a response to a request.
- **Latency**: time spent waiting (e.g. for disk or network).

We can **improve throughput** by:
- Minimizing work done per request.
- Doing enough work at once to keep all hardware resources busy…
- …and when some work is delayed (latency) find other work to do.

We can **improve response time** by:
- Minimizing total work and delay (latency) on critical path to a response.
- Applying parallel resources to an individual response…including streaming.
- Precomputing response values.
Know How Fast Things Are
Typical “speeds n feeds”

- **CPU (e.g. Intel Core i7):**
  - A few billions instructions / second *per core*
  - Memory: 20GB/sec (20 bytes/instruction executed)

- **Long distance network**
  - Latency (ping time): 10-100ms
  - Bandwidth: 5 – 100 Mb/sec

- **Local area network (Gbit Ethernet)**
  - Latency: 50-100usec (note microseconds)
  - Bandwidth: 1 Gb/sec (100mbytes/sec)

- **Hard disk**
  - Rotational delay: 5ms
  - Seek time: 5 – 10 ms
  - Bandwidth from magenetic media: 1Gbit/sec

- **SSD**
  - Setup time: 100usec
  - Bandwidth: 2Gbit/sec (typical) ← note: SSD wins big on latency, some on bandwidth
Making Systems Faster
Hiding Latency
Hard disks are slow
Handling disk data the slow way

The Slow Way

- Read a block
- Compute on block
- Read another block
- Compute on other block
- Rinse and repeat

Computer waits msec while reading disk → 1000s of instruction times!
Faster way: overlap to hide latency

The Faster Way

- Read a block
- Start reading another block
- Compute on 1st block
- Start reading 3rd block
- Compute on 2nd block
- Rinse and repeat

Buffering: we’re reading ahead...computing while reading!
Making Systems Faster
Bottlenecks and Parallelism
Amdahl’s claim: parallel processing won’t scale

1967: Major controversy …will parallel computers work?

“Demonstration is made of the continued validity of the single processor approach and of the weaknesses of the multiple processor approach in terms of application to real problems and their attendant irregularities. Gene Amdahl*”

Amdahl: why no parallel scaling?

“The first characteristic of interest is the fraction of the computational load which is associated with data management housekeeping. This fraction [...might eventually be reduced to 20%...]. The nature of this overhead appears to be sequential so that it is unlikely to be amenable to parallel processing techniques. Overhead alone would then place an upper limit on throughput of five to seven times the sequential processing rate. Gene Amdahl (Ibid)

In short: even if the part you’re optimizing went to zero time, the speedup would be only 5x.

$$\text{Speedup} = \frac{1}{(r_s + (r_p/n))}$$

where $r_s$ and $r_p$ are sequential/parallel fractions of computation

As $r_p/n \rightarrow 0$, Speedup $\rightarrow 1/r_s$
Web Performance and Scaling
Web Performance & Scalability Goals

- Overall Web Goals:
  - Extraordinary scalability, with good performance
  - Therefore...very high aggregate throughput (think of all the accesses being made this second)
  - Economical to deploy (modest cost/user)
  - Be a good citizen on the Internet

- Web servers:
  - Decent performance, *high throughput and scalability*

- Web clients (browsers):
  - Low latency (quick response for users)
  - Reasonable burden on PC
  - Minimize memory and CPU on small devices
What we’ve already studied about Web scalability…

- **Web Builds on scalable hi-perf. Internet infrastructure:**
  - IP
  - DNS
  - TCP

- **Decentralized administration & deployment**
  - The only thing resembling a global, central Web server is the DNS root
  - URI generation

- **Stateless protocols**
  - Relatively easy to add servers
Web server scaling

Browser

Web-Server
Application - logic

Reservation
Records

Data store
Stateless HTTP protocol helps scalability
Caching
There are only two hard things in Computer Science: cache invalidation and naming things.

-- Phil Karlton
Why does caching work at all?

- **Locality:**
  - In many computer systems, a small fraction of data gets most of the accesses
  - In other systems, a slowly changing set of data is accessed repeatedly

- **History: use of memory by typical programs**
  - Denning’s *Working Set Theory*
  - Early demonstration of locality in program access to memory
  - *Justified paged virtual memory with LRU replacement algorithm*
  - Also indirectly explains why CPU caches work

- **But…not all data-intensive programs follow the theory:**
  - Video processing!
  - Many simulations
  - Hennessy and Patterson: *running vector (think MMX/SIMD) data through the CPU cache was a big mistake in IBM mainframe vector implementations*


Also 2008 overview on locality from Denning:
Why is caching hard?

Things change

Telling everyone when things change adds overhead

So, we’re tempted to cheat...
...caches out of sync with reality
CPU Caching – Simple System

CPU

Memory

Read request

Read data

Read data

Read request

Read data CACHE
CPU Caching – Simple System

Life is Good
No Traffic to Slow Memory

Repeated read request

Read data

Read data CACHE

Memory
CPU Caching – Store Through Writing

Everything is up-to-date…
…but every write waits for slow memory!
CPU Caching – Store In Writing

Write request

CPU

Write data CACHE

Memory

The write is fast, but memory is out of date!
CPU Caching – Store **In** Writing

If we try to write data from memory to disk, the wrong data will go out!
Cache invalidation is hard!

We can start to see why cache invalidation is hard!
Multi-core CPU caching

Write request

CPU → CACHE

Coherence Protocol

CACHE → CPU

CPU → CACHE

CACHE → MEMORY

MEMORY
Multi-core CPU caching

Write request

Data → CACHE

Coherence Protocol

CACHE → CPU

Read request

Data

CACHE → CPU

Memory
Multi-core CPU caching

A read from disk must flush all caches
Consistency vs. performance

- Caching involves difficult tradeoffs
- Coherence is the enemy of performance!
  - This proves true over and over in lots of systems
  - There’s a ton of research on weak-consistency models...
- Weak consistency: let things get out of sync sometimes
  - Programming: compilers and libraries can hide or even exploit weak consistency
- Yet another example of leaky abstractions!
What about Web Caching?

Note: update rate on Web is mostly low – makes things easier!
Browsers have caches

Browser
Usually includes a cache!

Web Server

E.g. Firefox
Browser cache prevents repeated requests for same representations

E.g. Apache
Browsers have caches

Browser
Usually includes a cache!

E.g. Firefox
Browser cache prevents repeated requests for same representations...even different pages share images stylesheets, etc.

Web Server

E.g. Apache
Web Reservation System

Many commercial applications work this way.
HTTP Caches Help Web to Scale
Web Caching Details
HTTP Headers for Caching

- **Cache-control: max-age:**
  - Server indicates how long response is good

- **Heuristics:**
  - If no explicit times, cache can guess

- **Caches check with server when content has expired**
  - Sends ordinary GET w/validator headers
  - Validators: Modified time (1 sec resolution); Etag (opaque code)
  - Server returns “304 Not Modified” or new content

- **Cache-control: override default caching rules**
  - E.g. client forces check for fresh copy
  - E.g. client explicitly allows stale data (for performance, availability)

- **Caches inform downstream clients/proxies of response age**

- **PUT/POST/DELETE clear caches, but...**
  - No guarantee that updates go through same proxies as all reads!
  - Don’t mark as cacheable things you expect to update through parallel proxies!
Summary
Summary

- We have studied some key principles and techniques relating to performance and scalability
  - Amdahl’s law
  - Buffering and caching
  - Stateless protocols, etc.

- The Web is a highly scalable system w/most OK performance

- Web approaches to scalability
  - Built on scalable Internet infrastructure
  - Few single points of control (DNS root changes slowly and available in parallel)
  - Administrative scalability: no central Web site registry

- Web performance and scalability
  - Very high parallelism (browsers, servers all run in parallel)
  - Stateless protocols support scale out
  - Caching