CSE590IS
Making servers go fast

http://www.cs.washington.edu/education/courses/cse590is

Steve Gribble

Department of Computer Science and Engineering

UNIVERSITY OF WASHINGTON

Administrivia

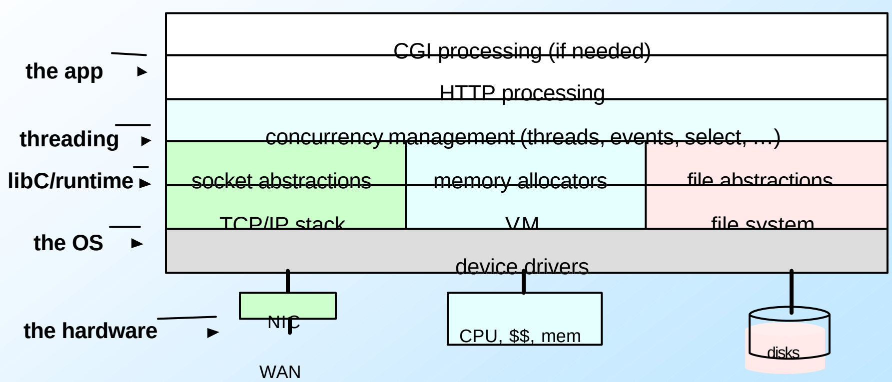
- **next on the hook: Paul**
  - coming up soon: Krishna, Sushant
- **some dates to take note of**
  - Feb 17: holiday
  - Feb 21: midterm out
  - Feb 28: midterm due

How to optimize performance

- **step 1: find the bottleneck of the system**
  - may be tough to find
    - obscured by parallelism/pipelining, multiple layers of abstraction
  - may depend on workload
    - scale, concurrency, popularity distributions
  - may change over time
    - hardware trends, workload trends, platform software
- **step 2: widen the bottleneck**
  - add more resources
  - make better use of resources: pipeline, parallelize, optimize algorithms
- **repeat as necessary**
  - but don't forget to stop when you're done…

Single machine web server

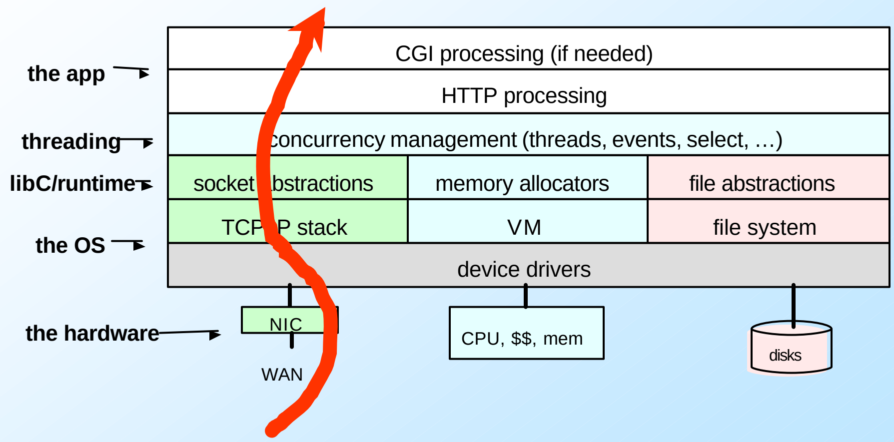
- **There are many potential bottlenecks:**



CSE590IS: Internet Systems/Services
©2003, Steven D. Gribble

Single machine web server

- **There are many potential bottlenecks:**



CSE590IS: Internet Systems/Services ©2003, Steven D. Gribble

Packet processing path

- **1400 byte packet arrival costs on 1.7 GHz P4 / Linux:**
  - device driver: 12 microseconds
  - TCP stack: 10 microseconds
  - user/kernel crossing: ~1 microsecond
  - extra copies: ~0.3 microseconds each
- **max throughput:**
  - ~550 Mb/s = 10,000 web requests/s = 1/5 of Yahoo
  - but now CPU is 100% utilized, no cycles left for apps
- **probably not the bottleneck for web servers…**

CSE590IS: Internet Systems/Services ©2003, Steven D. Gribble

Packet processing to the extreme

- **Two kinds of overhead: per-byte, per-packet**
  - **per-byte**: cost scales with size of packet
    - DMA between NIC/host
    - memory copies within host (e.g., copy across kernel boundary)
    - data manipulation (e.g., checksums)
  - solution? zero-copy networking, user-level networking, smart NIC
    - get OS out of way, DMA from device to user-level
  - **per-packet**: cost scales with # of packets
    - buffer allocation/deallocation
    - interrupt processing overhead
    - data structure manipulation (Mogul & Banga)
  - solution? optimized networking stacks, OS architectures

Socket abstractions

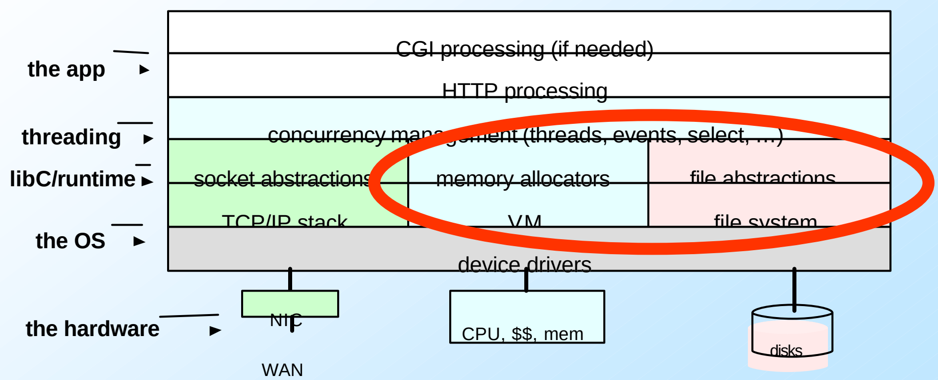
- **pitfall: benchmarking on a LAN instead of on a WAN**
  - WAN has 1000x higher latency
    - # concurrent connections = throughput x latency
    - amount of live state proportional to # concurrent connections
    - bandwidth-delay product is much higher
- **scaling to large # of concurrent connections**
  - Mogul & Banga paper: don't use linear data structures!
    - fancy select( ), socket allocator: still matters today
- **handling large BxD products**
  - provision socket buffers correctly
    - only matters for high throughput connections (video?)
    - not an issue for most servers: transfers are short, client BW is limited
    - running out of 32 bit sequence number space for TCP

Concurrency management

- **a religious topic: processes vs. threads vs. events**
  - thread fanatics
    - much easier to program
    - parallelism easier to find and exploit (SMPs)
    - performance is perfectly fine, thank you
  - event fanatics
    - much easier to program
    - scheduling is easier to control and exploit
      - not hidden in thread scheduler, or lock structure
    - performance, scaling properties are much better than threads
  - process people
    - who cares about this stuff, get a life!
- **my take on it (for servers)**
  - threads/processes work great, and this isn't the real bottleneck in most systems, so let's move on

Single machine web server

- **There are many potential bottlenecks:**



Pipeline servers: L1/L2 cache

- **claim: instructions-per-cycle (IPC) is low on servers**
  - blame threads for hurting I-cache performance
    - thread scheduler jumps between unrelated basic blocks
  - instead, break server into computational "stages"
    - execute all tasks in one stage before moving on to next
- **does it work?**
  - yes, but performance becomes very fragile
    - OS gets in the way
    - d-cache matters too
    - working set size of stage must be perfectly sized
  - payoff in practice is minimal
    - 5-10% improvement (1 month of moore's law)

Memory management

- **cache and VM performance might matter too**
  - memory allocator research
    - make more efficient use of physical memory to avoid VM pressure
    - parallelize to avoid becoming a bottleneck on SMPs
    - avoid artificial conflicts in caches due to integral page size layout
      - stack layout matters too
- **my take on this stuff**
  - we've been successful at hiding all of this machinery
    - but, not at all easy for app writers to optimize for this, or worse, to decide if optimizing for this matters…
  - thankfully, in most cases, I/O or processor is the bottleneck
    - cheap to overprovision memory to help make sure of this

Disks

- **if you move the disk arm, it will be your bottleneck**
  - seek: 5 ms
    - 10 million cycles, or 100 Mb/s of network throughput
    - seek bandwidth: 1 MB/s per disk
- **so what can we do?**
  - buy lots of memory to cache disk
  - avoid writes, and if use them, use logging to go sequential
  - avoid seeks on read, but if must, read >2MB after each seek
    - clever layout
  - coalesce reads from multiple connections by delaying
  - ultimately, buy lots of disks (clusters, disk arrays)

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) at high load
    - lazy-receiver-processing: reject early in TCP stack, interrupt costs accounted to destination process
- **differential quality-of-service**
  - if approaching capacity, service "high priority" connections
    - early demultiplexing so can associate packets with consumers

Latency vs. throughput

- **Harchol-Balter: optimizing the 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 live state in a server
  - instead: process connections with SRJF
    - doesn't matter under light load
    - matters a lot as approach capacity (10x latency at 90% load)
- **problems**
  - how do you estimate "length" of connection?
    - size of document x BW to end host
  - starvation of long jobs: why not just reject them?

Protocol optimization

- **HTTP is a really horrible protocol**
  - many small connections
    - overhead and latency of establishing TCP connection is bad
    - persistent connections helped
  - chatty, untokenized wireline format
    - typically 500-700 bytes per object in headers
    - irrelevant for wired servers/clients
    - matters more for wireless
      - pay-per-byte, content is much smaller
      - WAP fiasco

What about dynamic content?

- **most optimization papers deal with static web pages**
  - but increasing fraction of content is dynamically generated
- **what can we do?**
  - make CGI frameworks faster (“fast-CGI”)
  - make app logic faster
    - hard to generalize
  - punt and throw money at it (clusters)
  - offload costs to the client
    - edge-side includes (cache fragments, reassemble at clients)
    - push applets/data all the way to clients

Clusters

- **increase performance by replicating bottleneck resource**
- **introduces new issues**
  - load-balancing: avoid any replica from becoming bottleneck
    - how up-to-date must load information be?
    - Mitzenmacher:
      - stale information is good enough
      - real job is to avoid worst-case, rather than get to best-case
      - sample two or three, pick best
  - distributing working set rather than replicating it
    - LARD: partition working set
      - aggregate memory/disk scales with # of nodes

Discussion topics

- **assume that server/cluster performance issues are solved; what remains?**

Hotspots

- **sudden rise in popularity of a server (/. effect)**
  - dilemma
    - unlikely to happen any given server, so nobody provisions for it
    - most clients see them, so somebody ought to provision for it
- **many proposed solutions**
  - spill content to clients to absorb loads (padmanaban)
  - have servers cooperate
    - hash-signature of content means trust isn't issue
  - rent-a-server / CDN

Low-bandwidth last hops

- **the edge of the network isn't getting any faster**
  - and is the bottleneck for many systems [e.g., p2p]
  - limited number of tricks here…
    - better compression
      - lossy compression (distillation)
      - content hashcaches (exploit redundancy across objects)
    - latency-hiding with pipelined rendering / streaming
      - turned out to matter a lot for web page design
    - latency-hiding with aggressive prefetching
      - goal: 100% link utilization all the time
      - servers and ISPs hate this

Content is getting bigger

- **web: 4-6KB objects**
- **P2P: audio: 4MB, video: 1GB**
  - no part of the Internet is ready for this
    - server links, backbones, client links

- **not at all clear what to do here**
  - find something other than Internet to push the content?
    - snail-mail, sneakernet
  - one lever is that the content is immutable
    - satellites/cable/multicast to carousel most popular content