

Lecture#18: Intro to I/O; Disk physical characteristics

Review -- 1 min

Scheduling:

General lessons:

- Separate mechanism from policy
- Know your goals
- Compare with ideal

Specific policies:

- FIFO – fair, potentially poor latency
- RR – fair, potentially poor latency
- SJF – optimal latency; potentially unfair; not implementable (usually)
- Multi-level feedback queue – heuristics to approximate SJF; often used in practice
- Fair scheduling: Lottery, STFQ

Outline - 1 min

Intro to I/O

performance: Log model: Overhead, latency, BW

Disks

Data layout

Preview - 1 min

File systems

- Performance -- data layout
- Performance/persistence -- naming
- Reliability -- transactions

Networks

- overview

- protocols
- RPC
- distributed file systems

Security

Lecture - 20 min

1. IO -- tie back to previous discussions

How does IO relate to memory and interrupts?

What is "I/O" -- input/output

So far we've talked about CPU and memory. Now we want to talk about IO

-- my laptop (March 2011): keyboard (x3), mouse, disk, display, ethernet, 802.11, Bluetooth, microphone, speaker, wireless microphone, touch pad (x2), status lights, printer, scanner, MP3 player, camera

-- Fortunately, a common way to deal with all of these

Low level I: Memory mapped IO

[simplified story:]

-- allocate a region of physical memory for each device

e.g., "whatever device is plugged into slot 7 of my PCI will get addresses 0x10007000 to 0x10007FFF"

-- Writes/reads to this rang of addresses get sent to device

-- "control registers"

--> device can treat these writes/reads as requests

e.g., write value v to address 0x10070F0 means "move disk arm to location v"

e.g., read from address 0x10070F4 returns current disk arm position

--> Can create arbitrary function calls by setting arguments and then "go"

2 modes

(1) **Programmed IO** -- read/write individual "control registers" one word at a time

(2) **DMA** -- hand an address, length to IO device; IO device reads/writes an array of bytes at that address

QUESTION: Should DMA get a physical address or virtual address?

QUESTION: If DMA gets a virtual address, what happens if OS changes mappings for that page?

typically "pin" pages shared by OS and devices

Low level II: Interrupts

IO device can interrupt kernel when something interesting happens

-- kernel handler runs and uses device driver to access IO device

-- hardware has multiple interrupt numbers so that different handlers can be called for different devices

Medium level: Device drivers

Specific control registers and "function calls" are hardware specific. Vendor produces a *device driver* which exports some higher-level procedural interface to OS

-- Possibly a standard interface e.g., "Standard IDE disk", "standard ethernet"

-- Possibly a standard interface + optional extensions/enhancements

-- possibly non-standard

High level: Standard abstractions

-- file system hides details of disk devices

-- sockets hides details of network devices

...

2. Terminal: keyboard and display

Terminal connects to computer via serial line; same concept applies to modem connections — type characters, get characters back to display.

e.g. RS-232 is bit serial: start bit, character code, stop bit
 Typical BW: 9600 baud (transmission rate in bits/sec) → 900 bytes/sec

Even though keyboards/displays reflect a relatively small rate of transfer, can still swamp CPU due to overhead of handling each byte

e.g. what if interrupt per byte
 10 users (vi/emacs) (or 10 people using modem)
 900 interrupts/sec per user
 overhead of handling interrupt 100usec
 → devote whole computer to handling interrupts

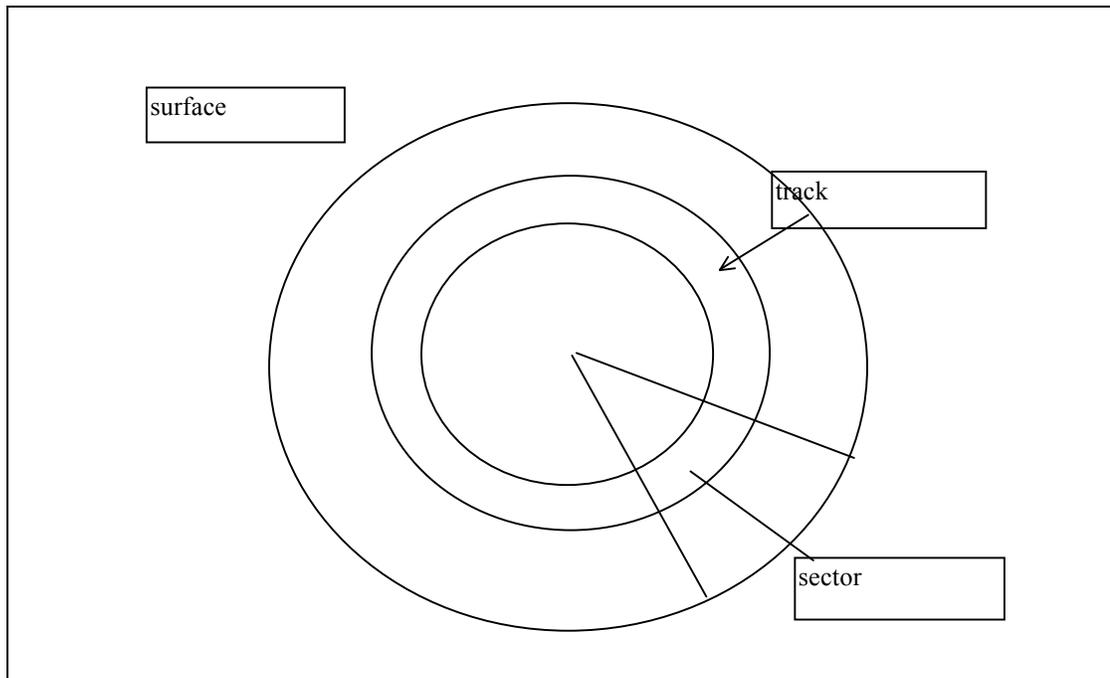
Alternative: use **Direct memory access (DMA)** to group bytes into blocks.

Instead of interrupt per byte, do block transfer, interrupt CPU when block is done

→ much higher transfer rates

Admin - 3 min

3. Disk Organization



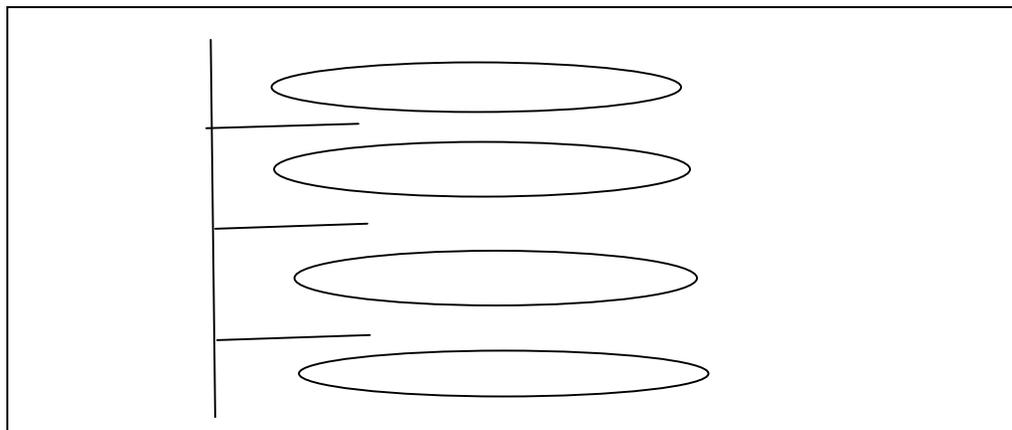
Disk **surface**: circular disk, coated with magnetic material

Tracks: concentric rings around disk surface, bits laid out serially along each track

Each track is split up into **sectors**: arc of track; also, minimal unit of transfer

Disks spin continuously

Disks organized as set of **platters** in a stack



Disk is read/write via a comb – 2 read/write “heads” at end of each “arm”

cylinder – corresponds to track on each surface

Disk operation is in terms of radial coordinates (not x, y, z) – move arm to correct track, wait for disk to rotate under head, select head, then transfer as it is going by

High-end disk today: Seagate Cheetah ST373405LC (March 2002)

	Seagate Cheetah ST373405LC (March 2002)	Seagate Cheetah NS ST3400755FC (March 2008)
Disk capacity	73 GB	400GB
# surfaces per pack	8	
# cylinders	29,549	
total # tracks per system	236,394	
#sectors per track	776 (avg)	
# bytes per sector	512 B	512B
# revolutions per minute	10000	10075 (2.98ms avg lat)
xfer rate	50-85 MB/s	97MB/s (max)
Avg seek time	5.1ms (r) 5.5ms (w)	3.9(r)/4.2(w)
1-track seek time	.4ms (r) .6ms(w)	.35(r)/.35(w)
full disk	9.4ms(r) 9.8ms(w)	
Extern xfer rate		4Gbit/s (500MB/s)
Cache size	4MB	16MB
Idle power	10W	8.1 W
Typical power		12.1W
MTBF	1,200,000 hours	1,400,000 hours
Annualized failure rate		.62%
Nonrecoverable read error per bit		10 ⁻¹⁶

4. Disk performance

4.1 Simple model

To read or write a disk block: **seek + rotation + transfer**

1. **seek**: position heads over cylinder (+ **select head**)

Time for seek depends on how fast you can move the arm.

Typically ~10-20ms to move all the way across disk

Typically 0.5-1ms to move 1 track (or select head on same cylinder)

“Average” typically 5-7ms to move 1/3 across disk

NOTE: potentially misleading/pessimistic – assumes **no locality**

2. **rotational delay** wait for sector to rotate underneath head

10000 RPM – 166 revolutions per second → 6 ms per rotation

3. **transfer** bytes

0.5 KB/sector * (100-1000)sectors/revolution * 166 Rev per second =
8-80 MB/s

(e.g. 10 MB/s → .5KB sector → 0.005ms)

overall time to do disk I/O

seek + rotational delay + transfer

Question: are seek and rotational

Seek and rotational delay are latency

transfer rate is BW

4.2 Caveats/Estimating performance

Modern disk drives more complex -- track buffer, sector sparing, etc.

see "Introduction to Disk Drive Modeling"

<http://www.hpl.hp.com/research/ssp/papers/IEEEComputer.DiskModel.pdf>

avg seek time seldom seen in practice

-- assumes no **locality**

-- assumes no **scheduling**

ty



Specifications	600GB ¹	450GB ¹	300GB ¹
Model Number	ST3600057SS ST3600957SS ² ST3600857SS ³ ST3600057FC ST3600957FC ^{2,4} ST3600857FC ^{3,4}	ST3450857SS ST3450757SS ² ST3450657SS ³ ST3450857FC ST3450757FC ^{2,4} ST3450657FC ^{3,4}	ST3300657SS ST3300557SS ² ST3300457SS ³ ST3300657FC ST3300557FC ^{2,4} ST3300457FC ^{3,4}
Capacity			
Formatted 512 KB/Sector (GB)	600	450	300
External Transfer Rate (MB/s)			
4Gb/s Fibre Channel	400	400	400
6Gb/s Serial Attached SCSI	600	600	600
Performance			
Spindle Speed (RPM)	15K	15K	15K
Average Latency (ms)	2.0	2.0	2.0
Seek Time Average Read/Write (ms)	3.4/3.9	3.4/3.9	3.4/3.9
Transfer Rate			
Internal (Mb/s, OD-ID)	1450 to 2370	1450 to 2370	1450 to 2370
Sustained (MB/s, 1000 x 1000)	122 to 204	122 to 204	122 to 204
Cache, Multisegmented (MB/s)	16	16	16
Configuration/Organization			
Disks/ Heads	4/8	3/6	2/4
Nonrecoverable Read Errors per Bits Read	1 sector per 10E16	1 sector per 10E16	1 sector per 10E16
Reliability Rating at Full 24x7 Operation (AFR)	0.55%	0.55%	0.55%
MTBF (hours)	1,600,000	1,600,000	1,600,000
Power Management			
Typical, Fibre Channel (W)	16.31	15.17	13.8
Typical, SAS (W)	16.35	14.6	12.92
Power Idle, Fibre Channel (W)	11.61	10.26	8.98
Power Idle, SAS (W)	11.68	10.1	8.74
Environmental			
Temperature, Operating (°C)	5 to 55	5 to 55	5 to 55
Temperature, Nonoperating (°C)	-40 to 70	-40 to 70	-40 to 70
Shock, Operating: 2ms (Gs)	60	60	60
Shock, Nonoperating: 2ms (Gs)	300	300	300
Acoustics Idle (dels—sound power)	3.6	3.6	3.6
Vibration, Operating: <400Hz (Gs)	1.0	1.0	1.0
Vibration, Nonoperating: <400Hz (Gs)	3.0	3.0	3.0
Encryption			
Self-Encrypting Drive Option ^{2,4}	Yes	Yes	Yes
FIPS Self-Encrypting Drive Option ^{2,4}	Yes	Yes	Yes
Physical			
Height (In/mm)	1.0/25.4	1.0/25.4	1.0/25.4
Width (In/mm)	4.0/101.6	4.0/101.6	4.0/101.6
Depth (In/mm)	5.76/146.52	5.76/146.52	5.76/146.52
Weight (lb/kg)	1.51/0.686	1.49/0.676	1.48/0.671
Warranty			
Limited Warranty (years)	5	5	5

¹ One gbyte, or GB, equals one billion bytes and one terabyte, or TB, equals one trillion bytes when referring to hard drive capacity.

² Self-Encrypting Drive model. Requires TCG-compliant host or controller support. Not available in all countries.

³ FIPS Self-Encrypting Drive model. Requires TCG-compliant host or controller support. Not available in all countries.

⁴ Self-Encrypting Drives and FIPS Self-Encrypting Drives are only available in the channel in 6Gb/s SAS.

Suppose 1000 pending 512 byte read requests, randomly distributed across Seagate Cheetah ST3600057FC

-- FIFO --> random seek + random rotate + xfer
 $3.4\text{ms} + 2\text{ms} + 512/150\text{MB/s} = 5.4\text{ms}$

-- SJF --> shortest job first
 NOT IDEAL
 problem with SJF -- starvation

-- SCAN/CSCAN/Elevator -- widely used
QUESTION what would you estimate? (strict track-track elevator; elevator + rotational knowledge)

strict: 1000 requests; 8 platters -- average request is .1% of distance across disk + head switch

smart:

Suppose 1000 pending 512 byte read requests, randomly distributed across 1GB file on Seagate Cheetah ST3600057FC and CSCAN scheduler

...now data is spread across

4.3 Sequential >> random

Random access

avg seek ~4ms

- ½ rotation ~3ms
- transfer ~0.02 ms
- about 7ms to fetch/put data; mostly seek/rotation
- 73 KB/s for random sector reads

Sequential access

What if next sector on same track? Then no seek, no rotation delay
 → 50-200 MB/s

Key to using disk effectively (and therefore to everything in file systems) is to minimize seek and rotational delay(1000x difference)

Simple example: do random read/writes of varying size; assume
 ~10ms avg seek + rot (no locality) and 100MB/s bandwidth

$$\text{EffectiveBW}(\text{size}) = \text{size} / (10\text{ms} + \text{size} / \text{BW})$$

(Note: for an access larger than 1 track, the 100MB/s bandwidth term neglects ~.5ms per rotation to resettle on next track; e.g., this simplification -- if a rotation is 6ms, then this simplification is about 10%)

Size	Time	EffectiveBW
.5KB	10ms	50KB/s
1KB	10.01ms	100KB/s
10KB	10.1ms	1MB/s
100KB	11ms	10MB/s
1MB	20ms	50MB/s
10MB	110ms	90MB/s

3 orders of magnitude better BW to read 1MB than to read 1 sector

“The secret to making disks fast is to treat them like tape” (J. Ousterhout)

5. Reliability

[[INSERT LAST PART OF LEC22.DOC HERE]]

6. Technology trends

1. Disks getting smaller for similar capacity
smaller → disk spins faster (less rotational delay, higher BW)
smaller → less distance for head to travel (faster seeks)
smaller → lighter weight (for portables)
2. disk data getting denser (more bits/square inch; allows smaller disks w/o sacrificing capacity)
Tracks closer together → faster seeks
3. Disks getting cheaper (per MB) (2x/year 1991-2005; 7x/5yr 2006-2011)

1983: 44MB disk \$4395 [byte magazine ad]
2006: 300GB EIDE disk \$94 [pricewatch.com]
2008: 500GB SATA disk \$82 [pricewatch.com]
2011: 2TB Western digital disk \$89 [buy.com]

4. Disks getting (a little) faster
seek, rotation – 5-10%/year (2-3x per decade)
bandwidth – 20-30%/year (~10x per decade)

Overall – disk density (\$/byte) improving much faster than mechanical limitations (seek, rotation)

Key to improving density: get head close to surface

Heads are spring loaded, aerodynamically designed to fly as close to surface as possible (also, lightweight to allow for faster seeks)

What happens if head contacts surface? Head crash – scrapes off magnetic material (and data)

6.1 Form factors

Why not read from multiple platters in parallel?

Why don't disks improve performance by having multiple independent arms?

Innovator's dilemma case study: form factor server->mini->desktop... kills old guard

Low-end product initially worse than high end product

→ Incumbents stick with high end

But low end product has volume

→ low end supplants high end

2008: In 10 years, will we still have magnetic disk or will solid state drives supplant? Seagate says we'll have both (2008). But the scenario has some resemblance to past disruptive technologies in this market...

7. Solid state drives

Different physical characteristics:

(1) no moving metal --> good random access

BUT

(2) write in 2 phases

(1) Clear group of pages (SLOW)

(2) Write individual pages (fast)

(3) Low power (no moving metal)

(4) Wear out (--> wear leveling)

	Spinning	SSD	
Cost (2010)	\$.10/GB	\$2/GB	(<10:1)
Random read/s	~100/s	~10K/s	(>100:1)
Random writes	~100/s	~200-1000/s (*)	(~= or better)
Read BW	~50MB/s	~50MB/s	(~=)
Power (active)	~10W	<1W	
Random read/s/w	~10/s/W	~10K/s/W	(>1000:1)
Sales volume	Lots	LOTS	

(* To write a flash drive you first clear a group of pages (128KB-256KB) and then write individual sectors. Clear group is slow. Write

sector is pretty fast. Bad design: write one sector --> need to re-write many sectors from block to new location --> slow. Newer systems keep spare sectors around and use something like a log structured file system and can have pretty good write performance (though still not as good as read).

Conclude: If you are capacity-bound, buy spinning. If you are IO bound or power or form-factor bound, buy SSD

Supplanting spinning drives in some markets (palmtop, smart phone, camera, music player)

Competing with spinning drives in some markets (laptop -- low power, small size v. capacity; server -- high IOPS/\$ v. high capacity/\$)

8. Data layout on (spinning) disk

2 driving forces

1) technology: avoid seeks, rotation (last time)

2) workloads:

How do users access files?

1. Sequential access – bytes read in order (give me the next X bytes, then give me the next)
2. Random access - read/write elements out of middle of array (give me bytes j-k)

How are files typically used?

1. Most files are small (e.g. .login, .c files)
2. Large files use up most of the disk space
3. Large files account for most of the bytes transferred to/from disk

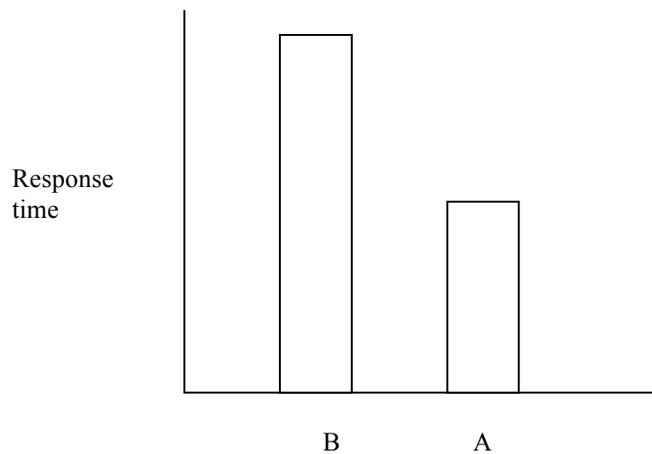
Bad news: need everything to be efficient

- Need small files to be efficient since lots of them

- need large files to be efficient, b/c most of the disk space, most of the I/O due to them

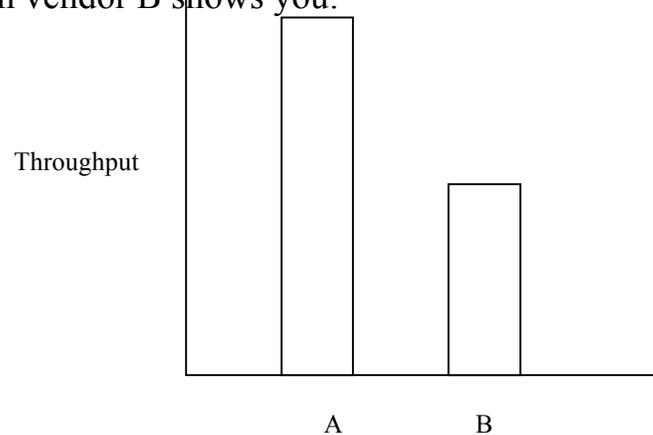
9. Understanding IO Performance

You need to buy a new RAID (Redundant array of inexpensive disks) system for your transaction server. Vendor A shows you this graph:



“Buy our system. It is twice as fast as B’s”

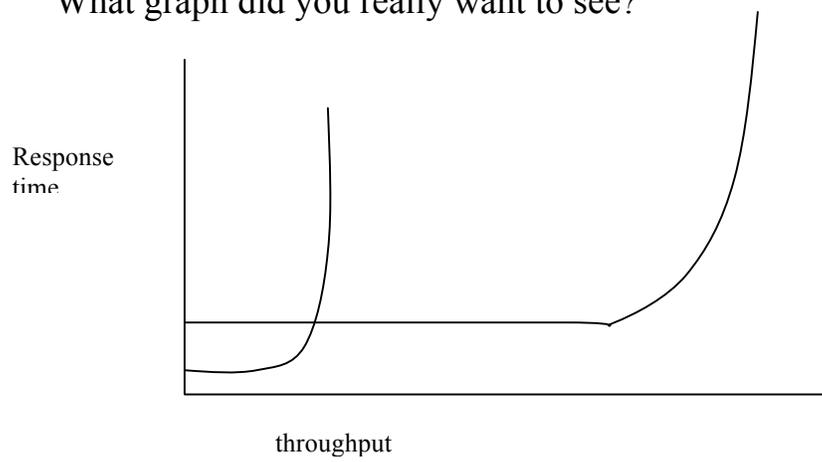
But then vendor B shows you:



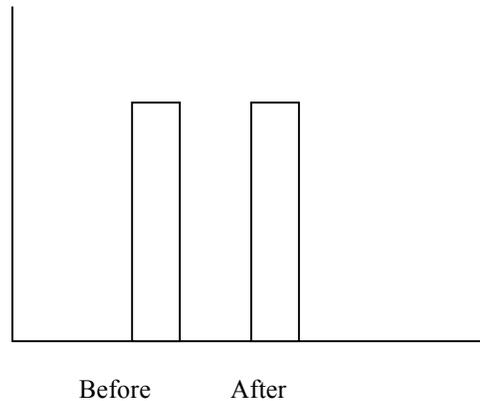
“Buy our system – it is twice as fast as A’s”

What should you buy? Who is lying?

What graph did you really want to see?



You buy one of them because it is faster than what you have. You plug the new RAID into your server and keynote measures the performance your users receive:



Why didn't the \$100K you just spent on faster hardware speed things up? How can you avoid making this mistake at your next job?

Two key ideas –

- (1) queuing theory
- (2) pipelining in IO systems, LogP model

10. A little queuing theory

Question: when should you buy a faster computer?

One approach – buy when it will pay for itself in improved response time

Queuing theory allows you to **predict** how response time will change as a function of hypothetical changes in # users, speed of CPU, speed of disk, etc

Might think you shouldn't buy a faster X when X has spare capacity (utilization of X < 100%), but for most systems, response time goes to infinity as utilization goes to 100%

How does response time vary with # users?

Worst case: all users submit jobs at same time. Thus response time gets linearly worse as add extra users, linearly better as computer gets faster

Best case: each user submits job after previous one completes. As increase #users, no impact on response time (until system completely utilized)

What if we assume users submit jobs randomly and they take random amounts of time. Possible to show mathematically:

$$\text{response time} = \text{service time} / (1 - \text{utilization})$$

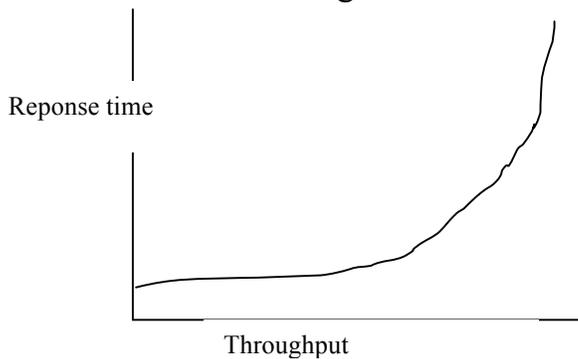
fine print – exponential distribution

10.1.1 Bottom line

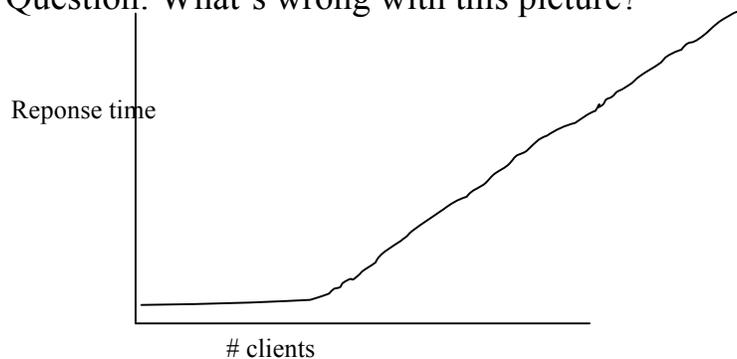
To measure system, plot throughput v. response time

Measuring response time at just one load level could be very misleading.

Often see something like this:



Question: What's wrong with this picture?



Common mistake: measure “offered load” rather than “throughput”

10.2 ~~LogP model (moved to lec24)~~

Your system really is client + NW + front end + back end + raid...

- Pipelining of computation and networks (I/O) really important for understanding performance
- Understanding pipelining more complex than CPU pipeline: variable stage depth, different “functional units”, etc.
- LogP model is useful pipelining model for I/O

10.2.1 Performance != Bandwidth

- Performance: “how fast is your network?” “how fast is your disk?”— people tend to answer in terms of bandwidth
- bandwidth is the MIPS of I/O
- In architecture, MIPS is one of three factors (cycles per instruction, instruction count, instructions per second)— only looking at one is misleading

Similar issue for I/O

Suppose I have a 100Mbps and 1000Mbps network. Is second network 10x faster?

Not if I use it to do a “remote read” (50 byte request, 50 byte response)

Graph: (lab) 510us (100Mbps), 501us (1000 Mbps)

(Graph: fixed portion + variable portion...)

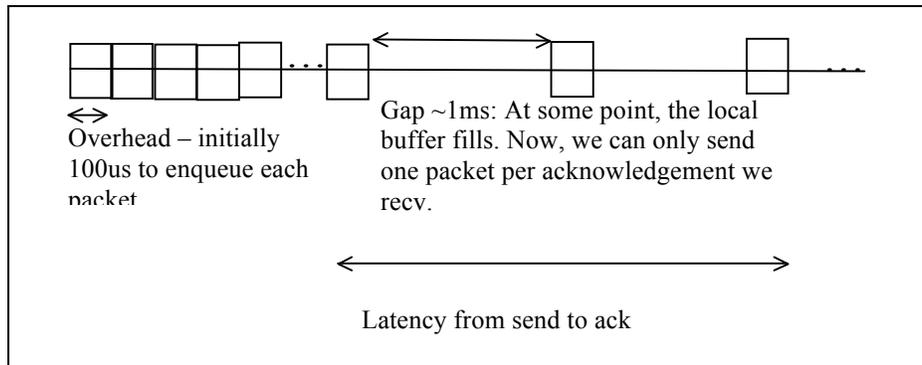
— Cross-country: 50.5ms (10Mbps), 50.5ms (100Mbps)

What’s going on?

How to understand I/O performance LogP:

- **Latency from X to Y:** real time from X to Y. Delay to initiate 1 byte operation (can be overlapped). “Latency” alone is ambiguous— must always specify latency of what to what? (E.g., network hardware latency— latency from when bit appears on one end of wire until it is taken off other end) [Delay from starting first to exiting last pipeline stage]
- **Overhead:** bottleneck time to initiate operation (can’t be overlapped) [Time consumed by first pipeline stage]
- **Bandwidth:** (“gap”) bottleneck rate of large number of operations once initiated [Time consumed by slowest pipeline stage]

E.g., suppose you open a TCP connection and start sending 1KB messages to another node on a 10Mbit/s Ethernet



1) What is "bottleneck rate"? (for overhead, BW)

The only tricky thing about this is that you have complex pipelining models (e.g., a disk request "occupies" CPU, bus, scsi controller, scsi bus, disk arm)

Which one is the bottleneck depends on configuration (how many disks? How many SCSI busses? How fast CPU?)

Which one is the bottleneck depends on how question is asked:

E.g., "For a Seagate Barracuda 5100 disk, what is the average overhead per 1-sector disk request?" v. "For a Dell Dimension 5100, what is the overhead per 1-sector disk request?" The first is asking how long a disk seek and rotation take; the second is asking how long the CPU is busy to set up a request.

Need to consider: What bottleneck is the question asking about?

For throughput, steady state bottleneck is the same in both cases.

For overhead, first stage overhead differs.

How does overhead differ from latency?

Overhead: resource usage

Latency: real-time end-to-end delay

How would you measure latency of a network request?

How does overhead differ from bandwidth?

How would you measure overhead of sending a packet?

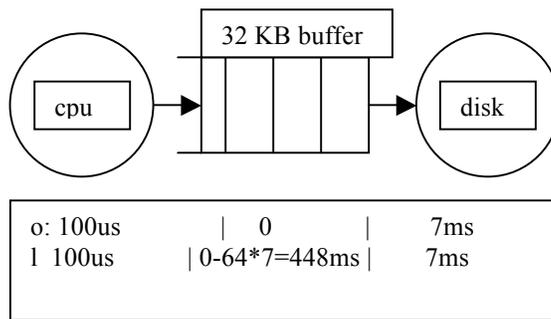
How would you measure bandwidth of a network?

2) Batching

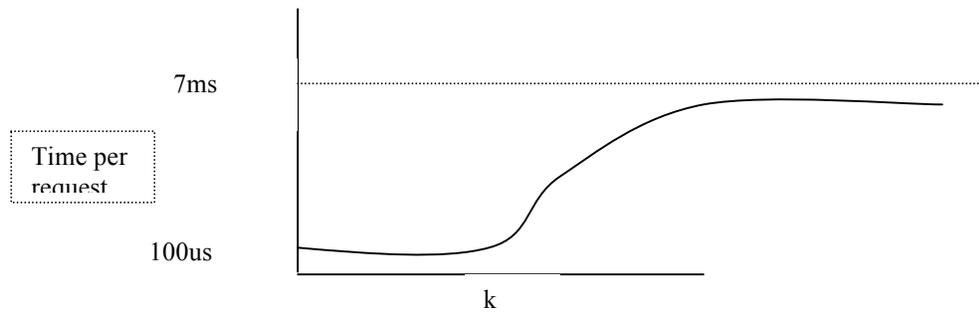
General rule of thumb: OS provides abstraction of byte transfers, but batch into block I/O for efficiency (pro-rates overhead and latency over larger unit)

Example

- Suppose CPU takes 100us of processing to issue one 512 byte write request
- Each request is to a random sector on disk
- Disk has parameters as above (4ms avg seek, 3ms $\frac{1}{2}$ rot, transfer .02ms)
- 32KB write buffer on disk (producer/consumer bounded buffer)
- Writes are issued asynchronously (CPU can issue $k+1$ as soon as k is in write buffer)



(1) Suppose CPU issues k back-to-back requests, when does CPU complete?



~~(2) When does first write to disk complete at the disk?~~

~~(e.g., latency from when first write starts at CPU until done at disk?)~~

7.1ms

~~(3) Suppose there are 500 writes in a burst, when does the last write complete at the disk?~~

~~100us + 500 * 7ms~~

10.3 Case study: sector layout

What is the fastest way to lay out a sequential file on disk

answer 1:

a series of sequential sectors on a track

problem (in old systems)

read sector 1

process sector 1

read sector 2 -- whoops, sector 2 is already past

wait 1 rotation

read sector 2

...

→ N rotations to read N blocks

→ BW for sequential read is 512 bytes/rotation = 100KB/s

answer 2: (in old systems)
 skip 1 sector (or 2 sectors) between sequential blocks
 → 2 rotations to read N blocks

answer 3: (modern systems)
 track buffer -- on-disk cache
 read entire sector into track buffer
 in parallel (once sector 1 arrives...) read sector 1 (from track)
 then read sector 2
 ...
 → 1 rotation to read N blocks

Moral: OS designer needs to understand physical properties of disk

Latency, overhead, bandwidth:

From disk -- what is overhead for a 1-sector read?

what is latency for a 1-sector read?

what is bandwidth term for a series of 1-sector reads to random blocks on disk?

From CPU/memory system

- what is overhead for a 1-sector read
- what is latency for a 1 sector read
- what is BW term for 1-sector reads to random blocks on disk?

Be careful: What is end-to-end average bandwidth for a 1-sector read (people phrase this question to mean end-to-end bytes/sec including latency and overhead)

Lecture - 23 min

Physical reality	Desired abstraction
disks are slow	fast access to data
sector addresses (“platter 2, cylinder 42, sector 15”)	named files, directories
write 1 sector at a time	atomic writes, transactions

