Lecture #27

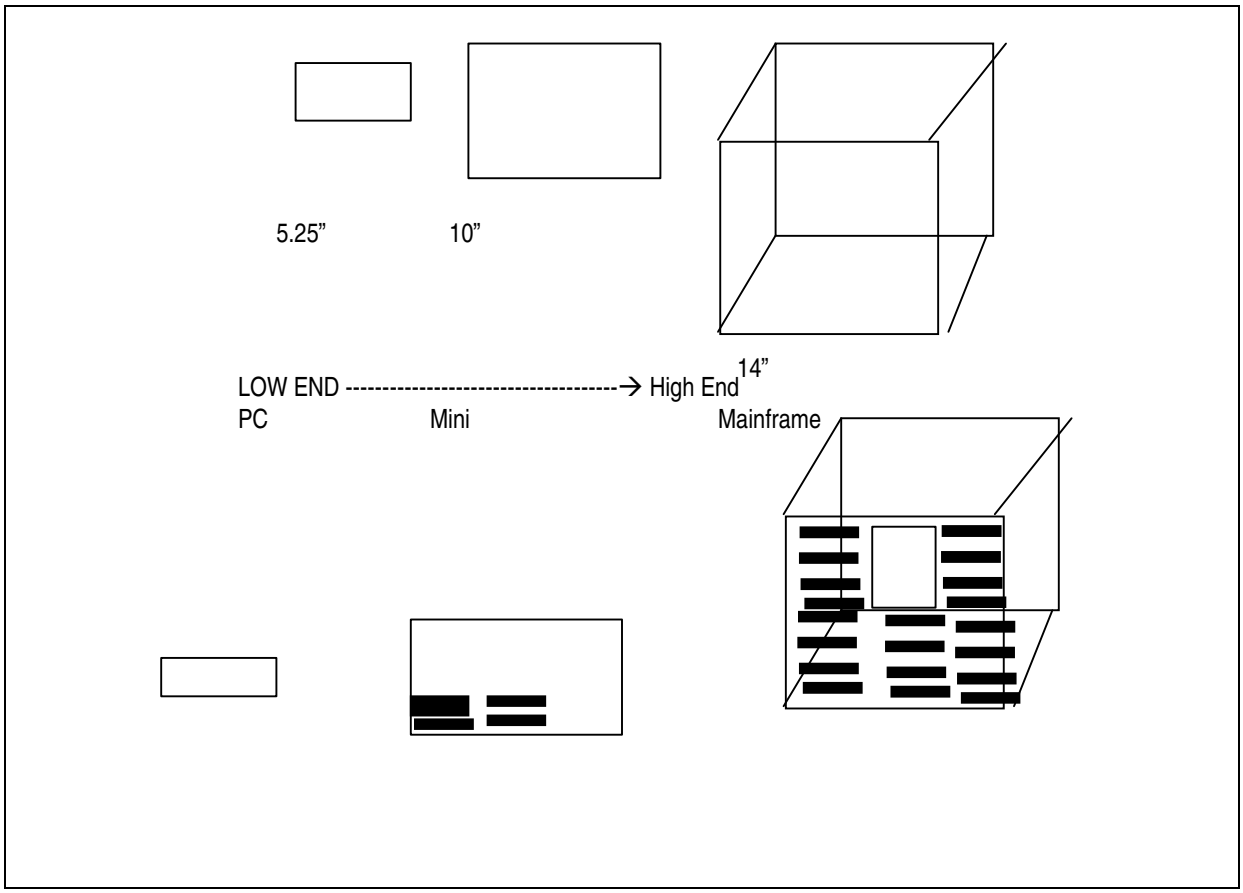
****
Review -- 1 min
****
Relationship of IO to processor architecture

****
Outline - 1 min
****
RAID
Benchmarks

****
Preview - 1 min
****
Networks

****
Lecture - 20 min
****
RAID

RAID: Motivation
- 1970’s-1980’s: Disk Product Families
  - multiple engineering teams



1990’s: RAID

From 1988 paper

Replace Small # of Large Disks w Large # of Small Disks

	IBM 3390	IBM 3.5” 0061	x70
Data Capacity	20 GB	320 MB	23 GB
Volume	97 Cu Feet	0.1 Cu Ft.	11 Cu ft
Power	3 KW	11 W	1 KW
Data Rate	15 MB/s	1.5 MB/s	120 MB/s
I/O Rate	600 I/O/s	55 I/O/s	3900 IO/s
MTTF	250 KHrs	50 KHrs	50Kh/70 → < 1000Hrs
Cost	$250K	$2K	$150K

Conclude: Disk Arrays have potential for

- Large Data and I/O Rates
- High MB per cu ft; High MB per KW
- Awful Reliability

Solution: **Redundancy**

Array Reliablity

- Reliability of N disks = Reliability of 1 disk / N
  - 50000 hours / 70 disks = 700 hours
  - MTTF drops from 6 years to 1 month!
  - Array w/o redundancy too unreliable to be useful

Approach

Store data + parity

DRAW PICTURE – data data data data parity

parity = data xor data xor data xor data

Idea

1 disk fails → recover data from data + parity

Subtleties

- Reliability v. Availability
  - reliability – "is it broken?"
  - availability – "can I access it?"
  - redundancy **hurts** reliability can help availability
- Redundancy alone not enough – also need **repair**
  - hot spares, repair

RAID equation: (Independent failures)

$$\text{MTTFarray} = \frac{\text{MTTFdisk}^2}{\text{N} * (\text{G}-1) * \text{MTTR(disk)}}$$

e.g. 100 disks @ MTTFdisk = 100000 hours; MTTR = 1 hr; G = 16

→ MTTF = 26M hours (3000 years)

- To improve availabilty
  - improve MTTF
  - reduce MTTR (e.g. hot spares)
  - reduce G (increases space overhead)

RAID Taxonomy

---

history – people had done no redundancy, mirroring, and bit-interleaved parity for a while; Patterson, katz and gibson needed to make it clear that they were talking about a different approach. Idea was to put all architectures into a numbering scheme to make it very clear which was which and what was different.

Unintended consequence: advertising "they do RAID 5, but we do RAID 6", "they do RAID 6 and we do RAID 7", ….

Example: "RAID 10" which had some aspects of RAID 2 and some of RAID 5 so they multiplied numbers together. I've seen up to "Raid 52"

→ there is now a RAID board that decides what numbers correspond to what architectures

RAID 0 – no parity <picture>

RAID 1 – mirroring <picture>

good bandwidth and IOPS

expensive – double cost of array per MB

- double cost of array per write performance
- efficient for read performance

RAID 2 – memory style ECC (e.g. hamming code)

bad idea – disks are "fail stop" -- notice when they fail

RAID 3 – bit-interleaved parity

bad idea – small access goes to all disks;

awkward interleaving DMA for large access

RAID 4 – Block-interleaved parity (dedicated parity disk)

<PICTURE>

→ overhead = 1/G space; 1/G bandwidth (large reads, writes); 1/G small reads, 2*G small writes

limitation – small write problem

small write → 4 I/O's

2 of those I/O's to parity disk

I/O's per second limited by parity disk

MAX Iops = disk IOPS / 2

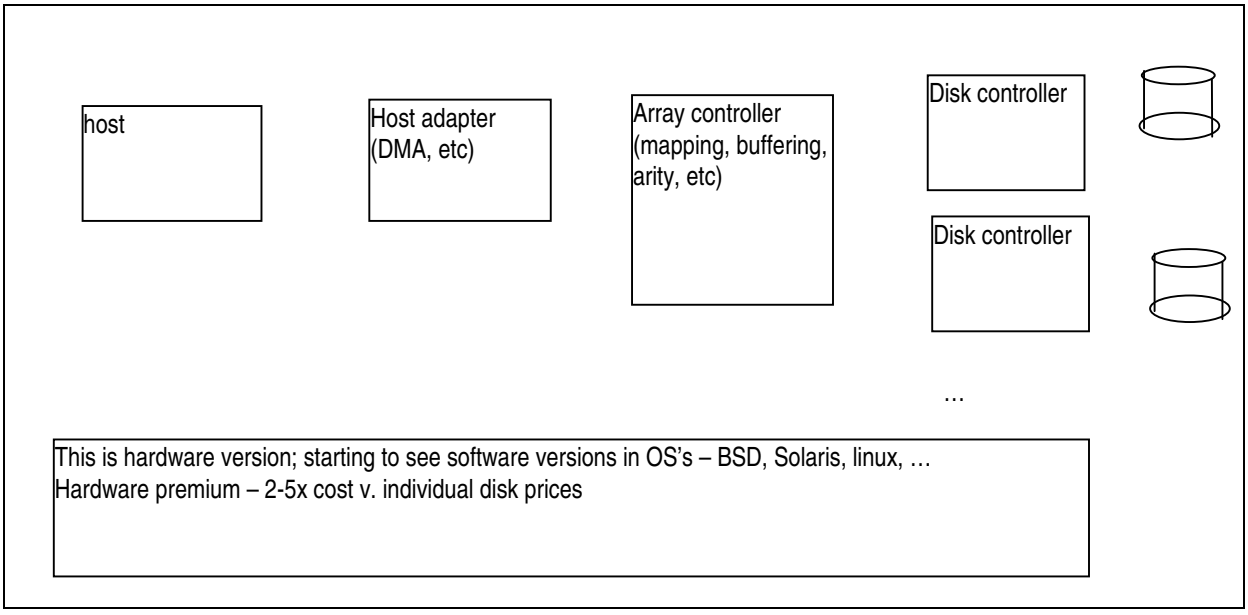
RAID 5 – Block Interleaved distributed parity
still do 4 I/O's per small write
distributed across all disks
MAX Iops = N * diskIOPS / 4

RAID 6 – P + Q redundancy

$$\text{MTTF_pq array} = \frac{\text{MTTF_disk}^3}{N * (G-1) * (G-2) * \text{MTTR}^2\text{_disk}}$$

e.g. 100 disks @ 100Khrs per disk; mttr = 1 hour; G = 16
→ MTTF_array = 333 billion hours (38 M years)

Subsystem Organization – How do you build it??



System-level availability

Any of those pieces can fail
worry about cable failures?

Yes – at rates of 1 per billion years, cable failures are significant

3 levels – bottom up
1) disks – more complicated than RAID model
2) other hardware in the box
3) system – software, environment, etc.

1) Disk failure models

- correlated disk failures
  - same shipment, same environment
- system crash during parity update + disk failure

need atomic update of data + parity → logging, NVRAM, …

- predict failures by watching "soft errors"

"negative" MTTR – pull disk before it fails

- reconstruction can "cause" failures
  - uncorrectable bit errors of 1 per 10^14 bits read
  - → 1 512 byte sector in 24 billion cannot be read
  - → P(sucessful reading 100M sectors
    = (1-1/(2.4*10^10)) ^ 2*10^8 = 99.2%

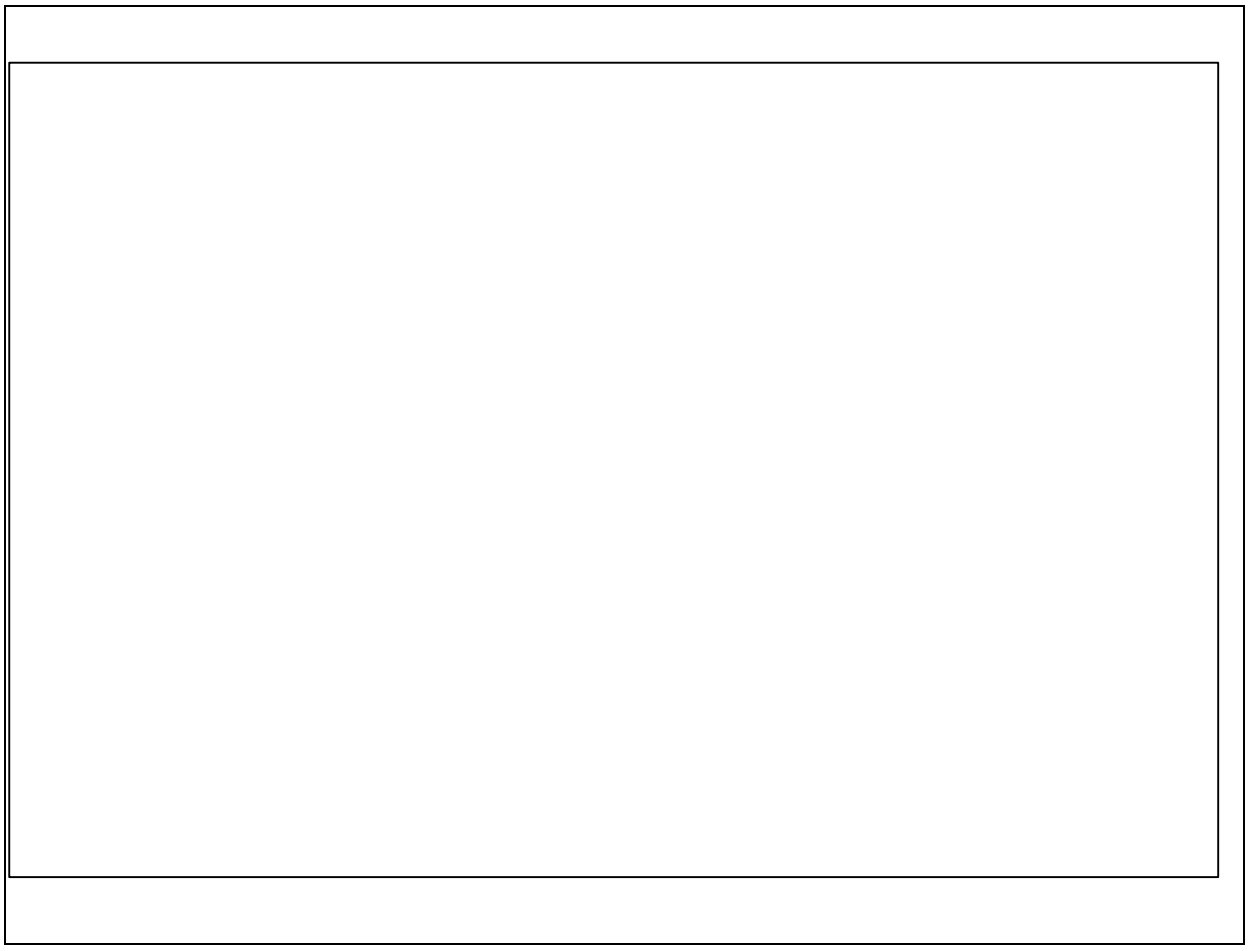
  → 0.8% of disk failures result in data loss

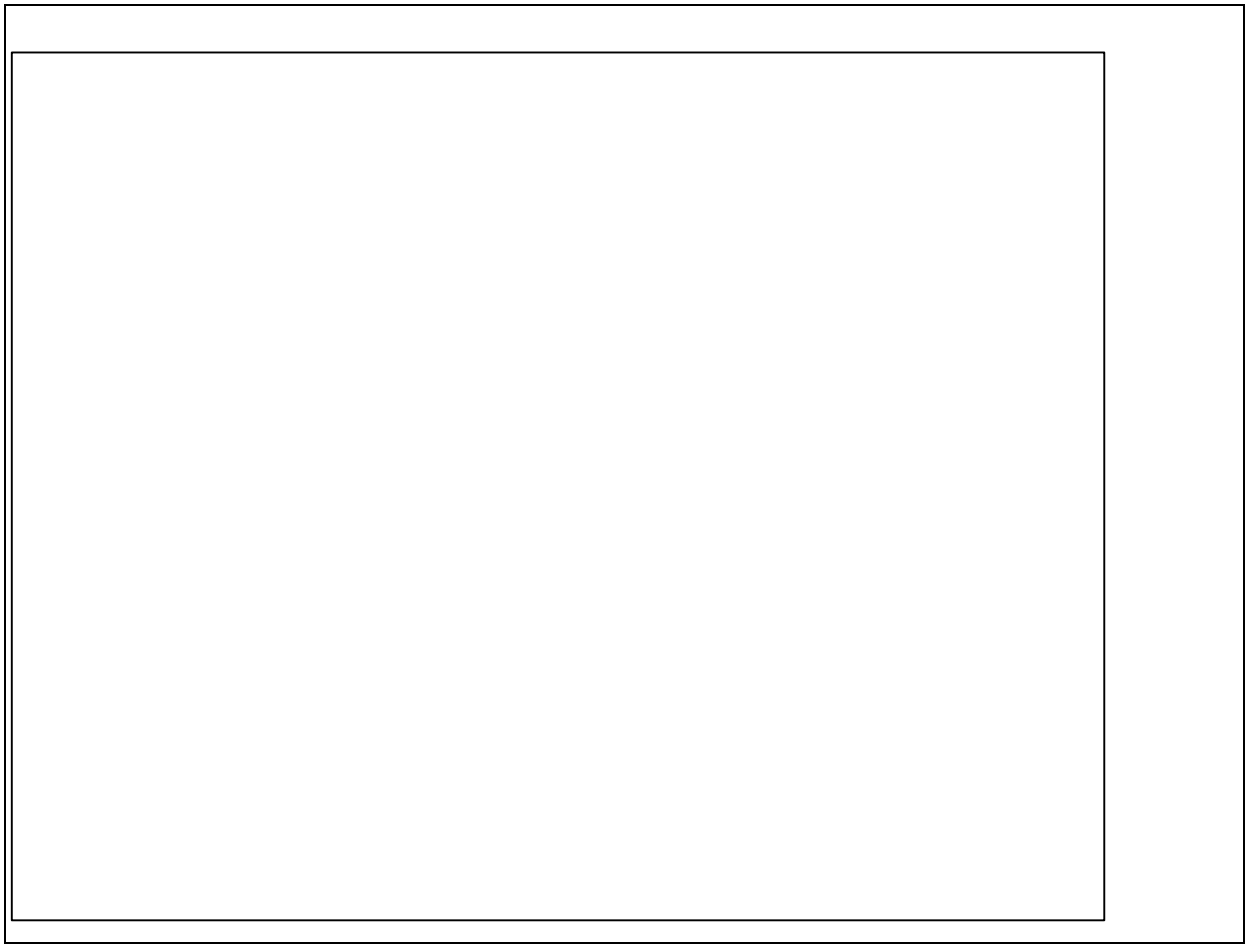
(Not so much "causing" failure – just that there is another failure mode other than head crash that the earlier model didn't account for)

2) Non-disk Hardware failures

Most common component failures (in order of failures)
fans, power supplies, controllers, cables

Solution 1: "Orthogonal RAID"





Solution 2: “Fully dual-redundant”

3) System-level availability
sources:

- environment – e.g. power outage
    - need UPS
- operations – e.g. “rm -r *”
- maintenance – e.g. kick power cord out of wall while cleaning
- software

e.g. bug in software

- hardware

    e.g. disk failure, CPU failure, memory failure

GRAPH: figure 1 Jim Gray “A Census of Tandem System Availability between 1985 and 1990” IEEE Transactions on Reliability v39 n4 Oct 1990

Moral: Hardware is pretty good (HW and maintenence terms)
environment is significant (need uninterruptable power supply)
software, operations are big problem

Future Directions

Network Attached Storage
avoid server memory bus, I/O bus bottleneck
Intelligent disks – put bigger processors, memory on disk drive
Serverless Network File Systems

****
Admin - 3 min
****

****
Lecture - 24 min
****

Benchmarks

“For better or worse, benchmarks shape a field”

I/O Benchmarks
typically measure throughput
possibly with upper limit on response time

Key issue – benchmark scaling
what if fix problem size, given 60%/year increase in DRAM capacity?

Benchmark	Size of Data	%Time in IO (1992)
IO Stones	1MB	26%

Andrew 4.5MB 4%
(1MB? Fits in L2 cache on modern machines!!!)

Observation – most benchmarks synthetic

- ♦ scaling
- ♦ hard to deal with large data sets

Self-scaling, Synthetic Benchmarks

Idea: automatically increase workload to stress system being measured

3 examples

- TPC – transaction processsing (TPC-A, TPC-B, TPC-C, TPC-D)
- NFS: SPEC SFS (aka Laddis)
- Unix I/O: Willy

Transaction Processing

TP aka OLTP (On-lin transaction processing)

- Changes to a large body of shared information from many terminals, with the TP system guaranteeing proper behavior on failures
- e.g. if bank's computer fails when a customer withdraws money, the TP system would guarantee that the account is debited if the customer received the mondy and that the account is unchanged if the money was not received
- Airline reservation systems and banks use TP

Key idea: Atomic ***Transactions***

Each transaction ➔ 2-10 disk I/Os + 5K-20K CPU instructions per disk I/O

SW efficiency crucial to avoiding disk accesses

Classic metric: TPS (Transactions per second)

- but under what workload? How were machines configured?

TPC Benchmark history

- Early 1980's: great interest in OLTP
  - Expecting demand for high TPS(e.g. ATM machines, credit cards)
  - Each vendor picked own conditions for TPS claims, report only CPU times with widely different I/O
  - Conflicting claims → disbelief of all benchmarks → chaos in market
- 1984 – Jim Gray of Tandem distributed paper to Tandem employees and 19 other companies to propose standard benchmark
- Published "A measure of transaction processing power" Datamation, 1985 by Anonymous et. Al
  - to indicate this was an effort of a large group
  - to avoid delays of legal department of each author's firm

TP by Anon et. al

- Proposed 3 standard tests to characterize commercial OLTP
  - TP1: OLTP test "DebitCredit" – simulates ATMs
  - Batch sort
  - Batch scan
- DebitCredit
  - One type of transaction – 100 bytes each
  - recorded 3 places: account file, branch file, teller file + all events recorded in history file
- Scaling: size of account, branch, teller, history are all function of throughput

TPS	#ATMs	account-file size
10	1000	0.1GB
100	10K	1.0 GB
1000	100K	10.0 GB
10,000	1000K	100.0 GB

→ each TPS → 100K account records, 10 branches, 100 ATMs

- response time: 95% transactions take < 1 second
- Configuration control: report price (initial purchase price + 5 year maintenence = cost of ownership)

Problems with TP1

Often ignored the user network to terminals
used transaction generator with no think time (made sense for vendor but not what customer would see)

Solution: hire auditor to certify results
auditors soon saw many ways to trick system
→ propsed minimum compliance list (13 pages)
still can't reproduce results

→ 1988: TPPC (Transaction processing performance council)
they create standard TPC benchmarks in 1990

New TP benchmarks

- TPC-A: Revised TP1/DebitCredit
  - Arrivals: Random (TPC) v. uniform (TP1)
  - Terminals: smart v. dumb (affects instr. Path lenght)
  - ATM scaling: 10 terminals per TPS v 100
  - branch scaling: 1 record per TPS v. 10
  - response time constraint: 90% < 2 seconds v. 95% < 1
  - full disclosure: approved by TPC
  - complete TPS/response time plot v. single point
- TPC-B: Same as TPC-A but without terminals (batch processing)
- Other efforts:
  - TPC-C – complex query processing
  - TPC-D – decision support

Lessons from TPC

- importance of scaling
- importance of standard
- peformance/cost metric

NFS Benchmark – SPEC SFS/LADDIS

1993 – attempt by NSF companies to agree on standard benchmark

- multiple "client" load generator machines
- no caching at clients
- read and write (and stat and …) random files
- reads: 85% full block + 15% partial block
- writes: 50% full block + 15% partial block
- each client access private subdirectory (subdirectories can even be in different file systems!)
- max avg response time 50 ms
- scaling: for every 100 NFS ops/sec, increase capacity by 1GB
- Result: plot of server load (throughput) v. response time

Limitations

- low-level benchmark – NFS server specific
  e.g. no client cache (so, an odd workload)
  → not correspond to performance seen by client
- odd workload
  odd access patterns (no client caches)
  no client sharing of data
  each client gets own subdirectory or file system

Willy – self-scaling UNIX file system benchmark

(Chen and Patterson 1993)

Self-scaling to stress different aspects of system
Examine 5 paramers

- unique bytes touched -> gives file cache size
- percentage of reads
- avg I/O request size (Bernoulli C = 1)
- percentage of sequential requests: typically 50%
- Number of processes: concurrency of workload

Idea : fix four parameters while vary 1 paramter
Search space to find high throughput

Example: Fig 6.26

Benchmark conclusions

- scaling to track technology
- TPC: price performance as normalizing config
- auditing to ensure no foul play
- throughput with restricted response time is normal measure
- practical challenges
    - ♦ large data sets → artificial workload (so can artificially generate)
    - ♦ large configurations → expensive to do tests (e.g. for high TPC need thousands of clients)

****
Summary - 1 min
****