Lecture #28

****
Review -- 1 min
****
RAID: use redundancy to make multiple disks feasible

→ good performance

→ excellent availability

1 disk: 1M hours mttf

100 disks: 90 data, 10 parity, (G=11) 1hr mttr: 1B hr mttf

100K years!

100 disks: 80 data, 20 parity (G=12), 1 hr mttr: 1x10^14 hr mttf

1B years!

RAID Architectures

- mirroring
- parity on blocks
- interleaved parity on blocks

****
Outline - 1 min
****
System-level availability

Tape Storage

Benchmarking

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

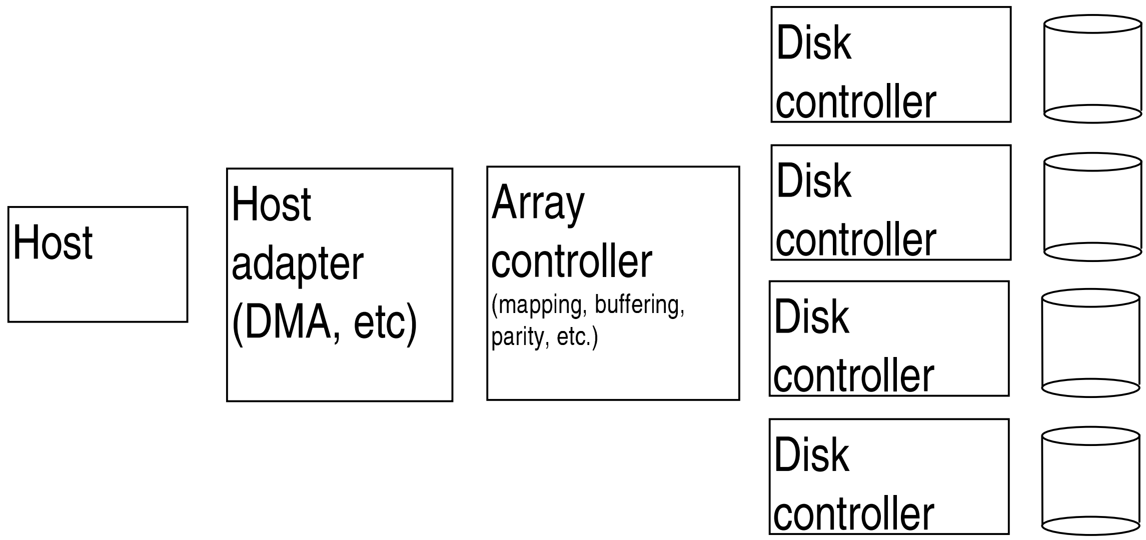
Intro – networking basics

Performance models

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

System-level availability

motivation: disk-only piece now seems pretty reliable
other pieces will limit availability



This is hardware version

- Starting to see Array controller in OS's
- Hardware premium – 2-5x cost v. individual disk prices

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

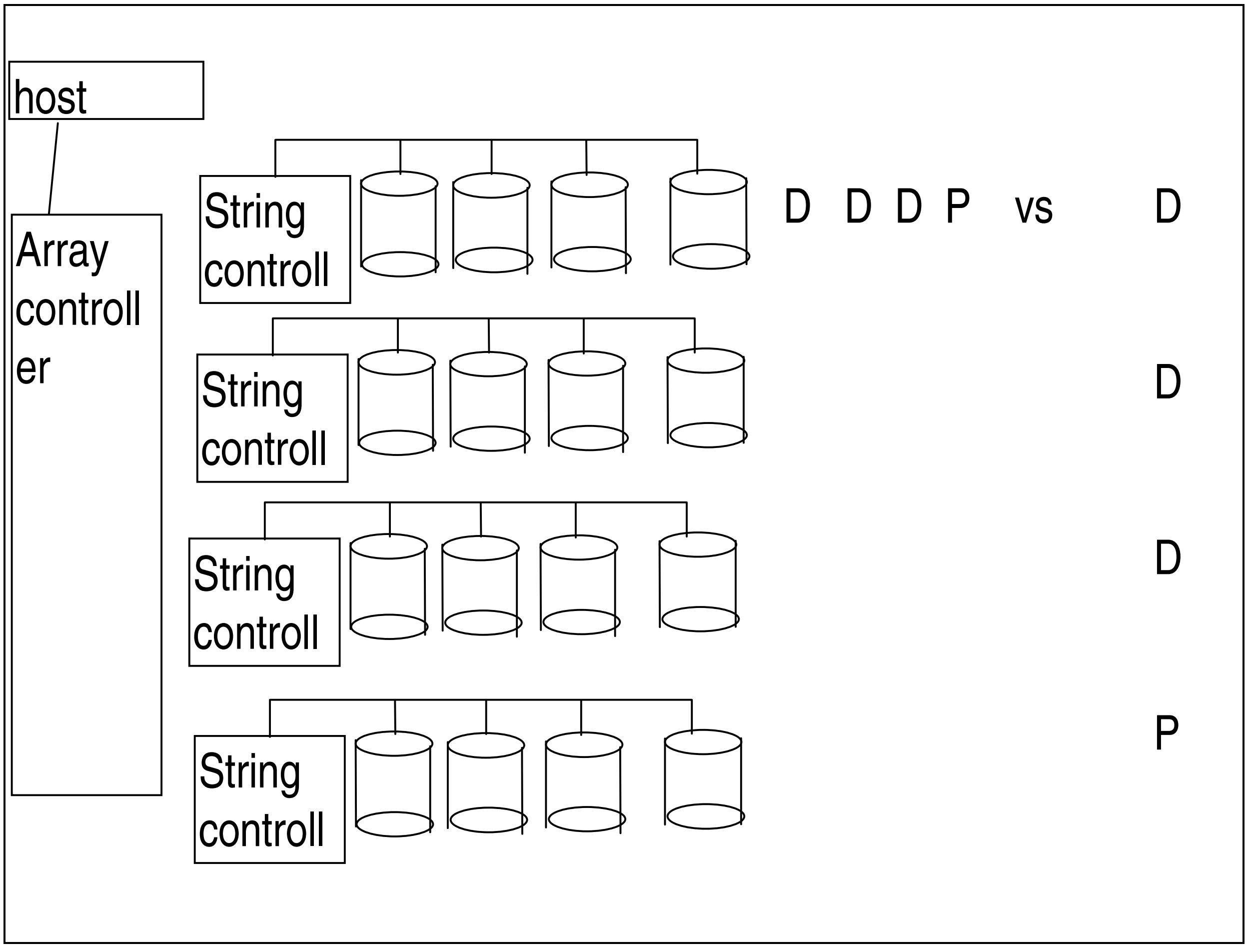
(Not so much "causing" failure – just that there is another failure mode other than head crash that the earlier model didn't account for)

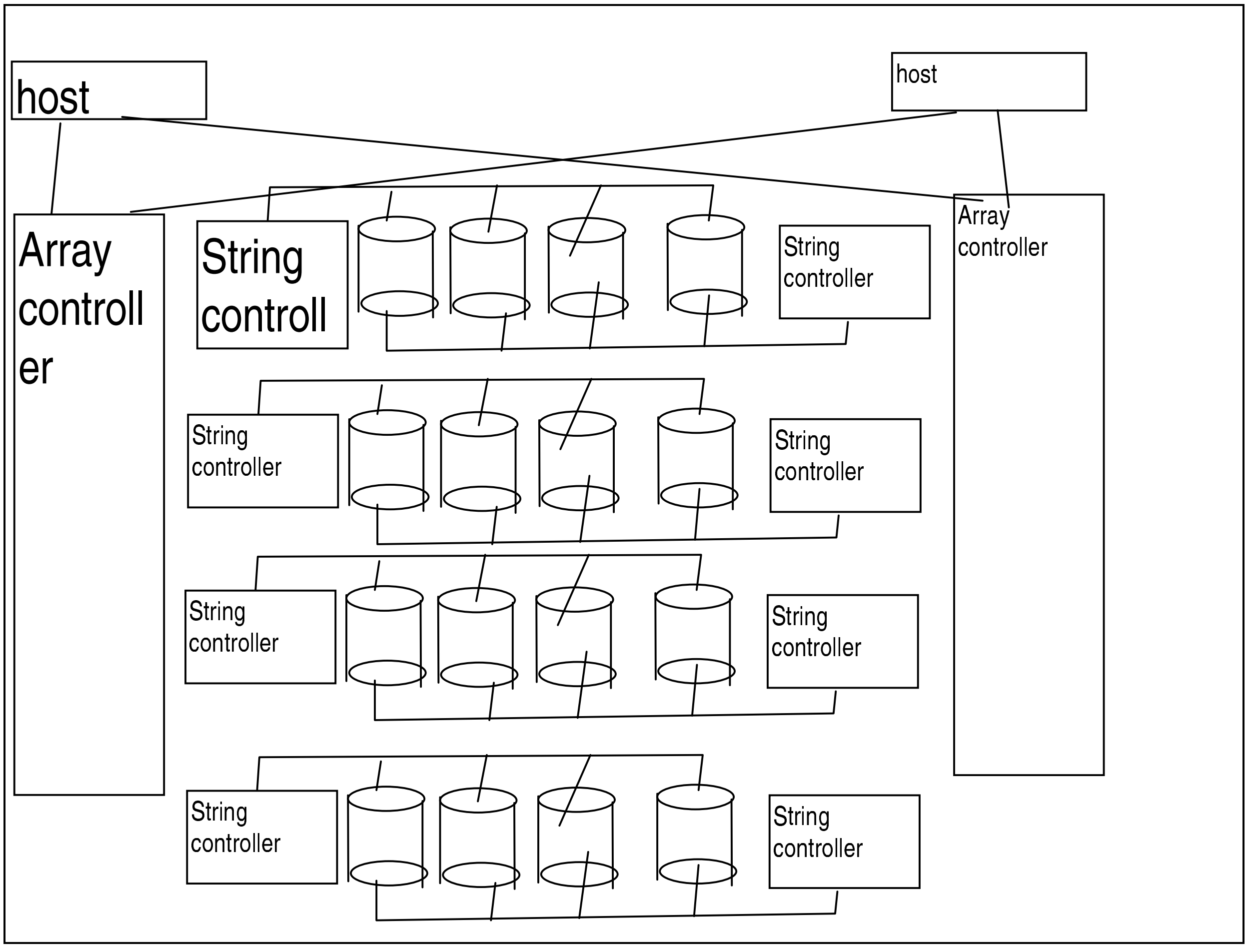
2) Non-disk Hardware failures

Most common component failures (in order of failures)

fans, power supplies, controllers, cables

Solution 1: "Orthogonal RAID"


host
Array controller
String controll
String controll
String controll
String controll
D D D P vs
D
D
D
P



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

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

hw5

checkpoint
- should be starting to see progress; even some preliminary results
- want roadmap of what remains to be done

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

Alternative Storage Technologies
----

idea – tapes are cheaper than disks; for some things that we don't access too often, we can store it more cheaply on tape

Technology motivation
- Helical Scan Tapes
  - Helical v. linear tape – better density
    e.g. VCR
  - Inherently better storage density than disk
    "3-d" tape v. "2-d" disk
  - Cheap media
    plastic ($5) v. metal and silicon ($500)
  - OK to good bandwidth (0.1-10 MB/s)

- Tape robots
    - automatically load tape in 10’s of seconds
      (Helical adds to load time)

Problems With Tape

- Slow seek time (10’s of seconds)
- Slow tape load (10’s of seconds)
- Media cheap, robots expensive (fallacy of media v. system)
- Standards slow technology advance
  → disks catching up
  But tape readers become obsolete
- Tape and head wear
    - replace after 100’s of reads
- Unpredictable access time
    - retry in firmware to improve recording density

Bottom line: previous applications are write seldom, read seldom

e.g. Backup (write once, read seldom), software distribution (write once, read once), data interchange (write once read once)

Case study: Parallel tape (a la RAID)

problem 1: for given size access, parallel increases number of seeks (slow) v. bandwidth (which tapes are pretty good at) → need huge amounts of data to make it worth while

problem 2: unpredictable time due to retry → parallel system slowed by slowest component

Case study (optimism): Digital library

1M book library * 400 pages/book * 4000 bytes per page (text) = 2 TB

images: ~100 TB

Storage Tec 120 TB robot -- $500K in 1994

$500K robot v. $100M library

Old computer hands get to talk about how they learned to program

- type program into cards, carry cards to operator, smile at them so they don’t lose your job or drop it on the floor; come back the next day and see “Oh, I forgot a semicolon on line 378” Type in semicolon (there was a machine to read old card set up to a point, insert a change, and read rest of old card set), try again

- We'll be able to tell same stories about library
  - high tech – had a n electric card catalog
  - then walk to library (10 minutes)
  - find book (10 minutes)
  - check it out and walk back (10 minutes)
  - and get this – while I've got the book no one else can read it for a month!!

Other arguments for

1) cost of handling paper

    storage cost – "Library = 100M mausoleum for dead trees"
    catalogue cost -- $20/book
    reshelve a book -- $1/book
    % new books purchased that are NEVER checked out – 20%

2) technology trends

Technical challenges

1) tape wear out, obsolecence

    ➔ need to periodically recopy data as part of cost model
    (shouldn't be too bad given tech trends.)

Other challenges – economic, legal

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
****