Storage

COSC 5351 Advanced Computer Architecture

Slides modified from Hennessy CS252 course slides
Outline

- Why do we need disks
- Magnetic Disks
- RAID
- Advanced Dependability/Reliability/Availability
- I/O Benchmarks, and Performance
- Conclusion
Case for Storage

- Shift in focus from computation to storage and communication of information
  - E.g., Cray Research/Thinking Machines vs. Google/Yahoo
  - “The Computing Revolution” (1960s to 1980s)
    ⇒ “The Information Age” (1990 to today)

- Storage emphasizes reliability and scalability as well as cost–performance

- What is “Software king” that determines which HW features actually used?
  - Operating System for storage
  - Compiler for processor

- Also has own performance theory—queuing theory—balances throughput vs. response time
Figure D.1 Cost versus access time for DRAM and magnetic disk in 1980, 1985, 1990, 1995, 2000, and 2005. The two-order-of-magnitude gap in cost and five-order-of-magnitude gap in access times between semiconductor memory and rotating magnetic disks have inspired a host of competing technologies to try to fill them. So far, such attempts have been made obsolete before production by improvements in magnetic disks, DRAMs, or both. Note that between 1990 and 2005 the cost per gigabyte DRAM chips made less improvement, while disk cost made dramatic improvement.
Why disks?

- Why not store in DRAM
  - They can be >100,000x faster
    - But that performances costs 150X more per GB
  - They have more bandwidth (2011)
    - About 80x, the bandwidth/GB is 12000X
      - But the $/bandwidth is 160x more costly

- Is there a technology cheaper than DRAM but faster than disks?
  - Many have tried, but by the time they make it to market DRAM gets cheaper and disks get faster
  - Right now SSDs are competing.
    - For servers, the 1M write limits is an issue, plus they are much more costly than disks
## Disk Figure of Merit: Areal Density

- **Bits recorded along a track**
  - Metric is **Bits Per Inch (BPI)**

- **Number of tracks per surface**
  - Metric is **Tracks Per Inch (TPI)**

- **Disk Designs Brag about bit density per unit area**
  - Metric is **Bits Per Square Inch: Areal Density** = BPI x TPI

<table>
<thead>
<tr>
<th>Year</th>
<th>Areal Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>1973</td>
<td>2</td>
</tr>
<tr>
<td>1979</td>
<td>8</td>
</tr>
<tr>
<td>1989</td>
<td>63</td>
</tr>
<tr>
<td>1997</td>
<td>3,090</td>
</tr>
<tr>
<td>2000</td>
<td>17,100</td>
</tr>
<tr>
<td>2006</td>
<td>130,000</td>
</tr>
<tr>
<td>2011</td>
<td>6.36E+11</td>
</tr>
</tbody>
</table>

### Chart

![Graph showing the increase in areal density from 1970 to 2010](chart.png)

**11 Sep 2011, Hitachi announced 636Gb/in²**
1956 IBM Ramac — early 1970s Winchester
- Developed for mainframe computers, proprietary interfaces
- Steady shrink in form factor: 27 in. to 14 in.

Form factor and capacity drives market more than performance

1970s developments
- 5.25 inch floppy disk formfactor (microcode into mainframe)
- Emergence of industry standard disk interfaces

Early 1980s: PCs and first generation workstations

Mid 1980s: Client/server computing
- Centralized storage on file server
  - accelerates disk downsizing: 8 inch to 5.25
- Mass market disk drives become a reality
  - industry standards: SCSI, IPI, IDE
  - 5.25 inch to 3.5 inch drives for PCs, End of proprietary interfaces

1990s: Laptops => 2.5 inch drives

2000s: What new devices leading to new drives?
## Future Disk Size and Performance

- Continued advance in capacity (60%/yr) and bandwidth (40%/yr)
- Slow improvement in seek, rotation (8%/yr)
- Time to read whole disk

<table>
<thead>
<tr>
<th>Year</th>
<th>Sequentially</th>
<th>Randomly (1 sector/seek)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1990</td>
<td>4 minutes</td>
<td>6 hours</td>
</tr>
<tr>
<td>2000</td>
<td>12 minutes</td>
<td>1 week(!)</td>
</tr>
<tr>
<td>2006</td>
<td>56 minutes</td>
<td>3 weeks (SCSI)</td>
</tr>
<tr>
<td>2006</td>
<td>171 minutes</td>
<td>7 weeks (SATA)</td>
</tr>
</tbody>
</table>
Use Arrays of Small Disks?

• Katz and Patterson asked in 1987:
  • Can smaller disks be used to close gap in performance between disks and CPUs?

Conventional:
4 disk designs

Low End ➔ High End

Disk Array:
1 disk design
Advantages of Small Form Factor Disk Drive Drives

Low cost/MB
High MB/volume
High MB/watt
Low cost/Actuator

Cost and Environmental Efficiencies
Replace Small Number of Large Disks with Large Number of Small Disks! (1988 Disks)

<table>
<thead>
<tr>
<th></th>
<th>IBM 3390K</th>
<th>IBM 3.5&quot; 0061</th>
<th>x70</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>20 GBytes</td>
<td>320 MBytes</td>
<td>23 GBytes</td>
</tr>
<tr>
<td>Volume</td>
<td>97 cu. ft.</td>
<td>0.1 cu. ft.</td>
<td>11 cu. ft.</td>
</tr>
<tr>
<td>Power</td>
<td>3 KW</td>
<td>11 W</td>
<td>1 KW</td>
</tr>
<tr>
<td>Data Rate</td>
<td>15 MB/s</td>
<td>1.5 MB/s</td>
<td>120 MB/s</td>
</tr>
<tr>
<td>I/O Rate</td>
<td>600 I/Os/s</td>
<td>55 I/Os/s</td>
<td>3900 I/Os/s</td>
</tr>
<tr>
<td>MTTF</td>
<td>250 KHrs</td>
<td>50 KHrs</td>
<td>??? Hrs</td>
</tr>
<tr>
<td>Cost</td>
<td>$250K</td>
<td>$2K</td>
<td>$150K</td>
</tr>
</tbody>
</table>

Disk Arrays have potential for large data and I/O rates, high MB per cu. ft., high MB per KW, but what about reliability?
3380 J – 1Gb
Array Reliability

• Reliability of N disks = Reliability of 1 Disk ÷ N

      50,000 Hours ÷ 70 disks = 700 hours

      Disk system MTTF: Drops from 6 years to 1 month!

• Arrays (without redundancy) too unreliable to be useful!

Hot spares support reconstruction in parallel with access: very high media availability can be achieved
Files are "striped" across multiple disks

Redundancy yields high data availability

- **Availability**: service still provided to user, even if some components failed

Disks will still fail

Contents reconstructed from data redundantly stored in the array

- Capacity penalty to store redundant info
- Bandwidth penalty to update redundant info
RAID 0

- JBOD
- Just use as a baseline
- Data may be striped or not striped
Redundant Arrays of Inexpensive Disks
RAID 1: Disk Mirroring/Shadowing

• Each disk is fully duplicated onto its “mirror”
  Very high availability can be achieved
• Bandwidth sacrifice on write:
  Logical write = two physical writes
  • Reads may be optimized
• Most expensive solution: 100% capacity overhead

• (RAID 2 not interesting, so skip)
RAID 10 or 01

- How does mirroring interact with striping
- Say we had 8 disks and we wanted to use RAID 1 on them.
  - Should we create 4 pairs of mirrored disks and stripe the data across them
    - RAID 1+0 or RAID 10 – “striped mirrors”
  - Should we create two sets of four disks each as RAID 0 and mirror them.
    - RAID 0+1 or RAID 01 – “mirrored stripes”
Redundant Array of Inexpensive Disks
RAID 3: Parity Disk

<table>
<thead>
<tr>
<th>Logical record</th>
<th>Striped physical records</th>
</tr>
</thead>
<tbody>
<tr>
<td>10010011</td>
<td>1</td>
</tr>
<tr>
<td>11001101</td>
<td>1</td>
</tr>
<tr>
<td>10010011</td>
<td>1</td>
</tr>
</tbody>
</table>

P contains sum of other disks per stripe mod 2 ("parity")

If disk fails, subtract P from sum of other disks to find missing information
RAID 3

- Sum computed across recovery group to protect against hard disk failures, stored in P disk
- Logically, a single high capacity, high transfer rate disk: good for large transfers
- Wider arrays reduce capacity costs, but decreases availability
- 33% capacity cost for parity if 3 data disks and 1 parity disk
Inspiration for RAID 4

- RAID 3 relies on parity disk to discover errors on Read
- But every sector has an error detection field
- To catch errors on read, rely on error detection field vs. the parity disk
- Allows independent reads to different disks simultaneously
Redundant Arrays of Inexpensive Disks
RAID 4: High I/O Rate Parity

Example: small read D0 & D5, large write D12-D15

Increasing Logical Disk Address

Stripe

Disk Columns

COSC5351 Advanced Computer Architecture
Inspiration for RAID 5

- RAID 4 works well for small reads
- Small writes (write to one disk):
  - Option 1: read other data disks, create new sum and write to Parity Disk
  - Option 2: since P has old sum, compare old data to new data, add the difference to P
- Small writes are limited by Parity Disk: Write to D0, D5 both also write to P disk
Independent writes possible because of interleaved parity

Example: write to D0, D5 uses disks 0, 1, 3, 4
Problems of Disk Arrays: Small Writes

RAID-5: Small Write Algorithm

1 Logical Write = 2 Physical Reads + 2 Physical Writes

(1. Read) XOR old data

(2. Read) XOR old parity

(3. Write)

(4. Write) XOR new data

D0' D1 D2 D3 P

D0' D1 D2 D3 P'

RAID - 5: Small Write Algorithm

1 Logical Write = 2 Physical Reads + 2 Physical Writes

COSC5351 Advanced Computer Architecture
RAID 6: Recovering from 2 failures

Why > 1 failure recovery?
- operator accidentally replaces the wrong disk during a failure
- since disk bandwidth is growing more slowly than disk capacity, the MTT Repair a disk in a RAID system is increasing
  ⇒ increases the chances of a 2nd failure during repair since takes longer
- reading much more data during reconstruction meant increasing the chance of an uncorrectable media failure, which would result in data loss
RAID 6: Recovering from 2 failures

- Network Appliance’s *row-diagonal parity* or *RAID-DP*

- Like the standard RAID schemes, it uses redundant space based on parity calculation per stripe

- Since it is protecting against a double failure, it adds two check blocks per stripe of data.
  - If \( p+1 \) disks total, \( p-1 \) disks have data; assume \( p=5 \)

- Row parity disk is just like in RAID 4
  - Even parity across the other 4 data blocks in its stripe

- Each block of the diagonal parity disk contains the even parity of the blocks in the same diagonal
Example $p = 5$

- Row diagonal parity starts by recovering one of the 4 blocks on the failed disk using diagonal parity
  - Since each diagonal misses one disk, and all diagonals miss a different disk, 2 diagonals are only missing 1 block
- Once the data for those blocks is recovered, then the standard RAID recovery scheme can be used to recover two more blocks in the standard RAID 4 stripes
- Process continues until two failed disks are restored

<table>
<thead>
<tr>
<th></th>
<th>Data Disk 0</th>
<th>Data Disk 1</th>
<th>Data Disk 2</th>
<th>Data Disk 3</th>
<th>Row Parity</th>
<th>Diagonal Parity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

COSC5351 Advanced Computer Architecture
Example $p = 5$

- Row diagonal parity starts by recovering one of the 4 blocks on the failed disk using diagonal parity
  - Since each diagonal misses one disk, and all diagonals miss a different disk, 2 diagonals are only missing 1 block
- Once the data for those blocks is recovered, then the standard RAID recovery scheme can be used to recover two more blocks in the standard RAID 4 stripes
- Process continues until two failed disks are restored
Berkeley History: RAID–I

- RAID–I (1989)
  - Consisted of a Sun 4/280 workstation with 128 MB of DRAM, four dual-string SCSI controllers, 28 5.25-inch SCSI disks and specialized disk striping software

- Today RAID is $24 billion dollar industry, 80% nonPC disks sold in RAIDs
Summary: RAID Techniques: Goal was performance, popularity due to reliability of storage

- **Disk Mirroring, Shadowing (RAID 1)**
  Each disk is fully duplicated onto its "shadow"
  Logical write = two physical writes
  100% capacity overhead

- **Parity Data Bandwidth Array (RAID 3)**
  Parity computed horizontally
  Logically a single high data bw disk

- **High I/O Rate Parity Array (RAID 5)**
  Interleaved parity blocks
  Independent reads and writes
  Logical write = 2 reads + 2 writes
Definitions

- Examples on why precise definitions so important for reliability
- Is a programming mistake a fault, error, or failure?
  - Are we talking about the time it was designed or the time the program is run?
  - If the running program doesn’t exercise the mistake, is it still a fault/error/failure?
- If an alpha particle hits a DRAM memory cell, is it a fault/error/failure if it doesn’t change the value?
  - Is it a fault/error/failure if the memory doesn’t access the changed bit?
  - Did a fault/error/failure still occur if the memory had error correction and delivered the corrected value to the CPU?
Computer system **dependability**: quality of delivered service such that reliance can be placed on service

*Service* is observed **actual behavior** as perceived by other system(s) interacting with this system’s users

Each module has ideal **specified behavior**, where **service specification** is agreed description of expected behavior

A system **failure** occurs when the actual behavior deviates from the specified behavior

Failure occurred due to an **error**, a defect in module

The cause of an error is a **fault**

When a fault occurs it creates a **latent error**, which becomes **effective** when it is activated

When error actually affects the delivered service, a failure occurs (time from error to failure is **error latency**)
Fault v. (Latent) Error v. Failure

- An error is manifestation *in the system* of a fault, a failure is manifestation *on the service* of an error.
- If an alpha particle hits a DRAM memory cell, is it a fault/error/failure if it doesn’t change the value?
  - Is it a fault/error/failure if the memory doesn’t access the changed bit?
  - Did a fault/error/failure still occur if the memory had error correction and delivered the corrected value to the CPU?
- An alpha particle hitting a DRAM can be a fault
  - if it changes the memory, it creates an error
  - error remains *latent* until effected memory word is read
  - if the effected word error affects the delivered service, a failure occurs
Fault Categories

1. Hardware faults: Devices that fail, such as alpha particle hitting a memory cell
2. Design faults: Faults in software (usually) and hardware design (occasionally)
3. Operation faults: Mistakes by operations and maintenance personnel
4. Environmental faults: Fire, flood, earthquake, power failure, and sabotage

Also by duration:
1. Transient faults exist for limited time and not recurring
2. Intermittent faults cause a system to oscillate between faulty and fault-free operation
3. Permanent faults do not correct themselves over time
Fault Tolerance vs Disaster Tolerance

- **Fault–Tolerance (or more properly, Error–Tolerance):** mask local faults (prevent errors from becoming failures)
  - RAID disks
  - Uninterruptible Power Supplies
  - Cluster Failover

- **Disaster Tolerance:** masks site errors (prevent site errors from causing service failures)
  - Protects against fire, flood, sabotage,..
  - Redundant system and service at remote site.
  - Use design diversity

From Jim Gray's "Talk at UC Berkeley on Fault Tolerance" 11/9/00
Mean Time to System Failure (years) by Cause

<table>
<thead>
<tr>
<th>Year</th>
<th>Software</th>
<th>Hardware</th>
<th>Maintenance</th>
<th>Operations</th>
<th>Environment</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1985</td>
<td>2</td>
<td>29</td>
<td>45</td>
<td>99</td>
<td>142</td>
<td>360</td>
</tr>
<tr>
<td>1987</td>
<td>53</td>
<td>91</td>
<td>162</td>
<td>171</td>
<td>214</td>
<td>438</td>
</tr>
<tr>
<td>1990</td>
<td>33</td>
<td>310</td>
<td>409</td>
<td>136</td>
<td>346</td>
<td>500</td>
</tr>
</tbody>
</table>

Systematic Under-reporting

Problem: Systematic Under-reporting

From Jim Gray's "Talk at UC Berkeley on Fault Tolerance" 11/9/00
COSC5351 Advanced Computer Architecture 11/2/2011
Is Maintenance the Key?

- Rule of Thumb: Maintenance 10X HW

- VAX crashes ‘85, ‘93 [Murp95]; extrap. to ‘01
- Sys. Man.: N crashes/problem, SysAdmin action
  - Actions: set params bad, bad config, bad app install
- HW/OS 70% in ‘85 to 28% in ‘93. In ‘01, 10%?
HW Failures in Real Systems: Tertiary Disks

• A cluster of 20 PCs in seven 7-foot high, 19-inch wide racks with 368 8.4 GB, 7200 RPM, 3.5-inch IBM disks. The PCs are P6-200MHz with 96 MB of DRAM each. They run FreeBSD 3.0 and the hosts are connected via switched 100 Mbit/second Ethernet.

<table>
<thead>
<tr>
<th>Component</th>
<th>Total in System</th>
<th>Total Failed</th>
<th>% Failed</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCSI Controller</td>
<td>44</td>
<td>1</td>
<td>2.3%</td>
</tr>
<tr>
<td>SCSI Cable</td>
<td>39</td>
<td>1</td>
<td>2.6%</td>
</tr>
<tr>
<td>SCSI Disk</td>
<td>368</td>
<td>7</td>
<td>1.9%</td>
</tr>
<tr>
<td>IDE Disk</td>
<td>24</td>
<td>6</td>
<td>25.0%</td>
</tr>
<tr>
<td>Disk Enclosure - Backplane</td>
<td>46</td>
<td>13</td>
<td>28.3%</td>
</tr>
<tr>
<td>Disk Enclosure - Power Supply</td>
<td>92</td>
<td>3</td>
<td>3.3%</td>
</tr>
<tr>
<td>Ethernet Controller</td>
<td>20</td>
<td>1</td>
<td>5.0%</td>
</tr>
<tr>
<td>Ethernet Switch</td>
<td>2</td>
<td>1</td>
<td>50.0%</td>
</tr>
<tr>
<td>Ethernet Cable</td>
<td>42</td>
<td>1</td>
<td>2.3%</td>
</tr>
<tr>
<td>CPU/Motherboard</td>
<td>20</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>Messages in system log for failed disk</td>
<td>No. log msgs</td>
<td>Duration (hours)</td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------------------------------------</td>
<td>--------------</td>
<td>-----------------</td>
<td></td>
</tr>
<tr>
<td><strong>Hardware Failure</strong> (Peripheral device write fault [for] Field Replaceable Unit)</td>
<td>1763</td>
<td>186</td>
<td></td>
</tr>
<tr>
<td><strong>Not Ready</strong> (Diagnostic failure: ASCQ = Component ID [of] Field Replaceable Unit)</td>
<td>1460</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td><strong>Recovered Error</strong> (Failure Prediction Threshold Exceeded [for] Field Replaceable Unit)</td>
<td>1313</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td><strong>Recovered Error</strong> (Failure Prediction Threshold Exceeded [for] Field Replaceable Unit)</td>
<td>431</td>
<td>17</td>
<td></td>
</tr>
</tbody>
</table>
### High Availability System Classes

#### Goal: Build Class 6 Systems

<table>
<thead>
<tr>
<th>System Type</th>
<th>Unavailable (min/year)</th>
<th>Availability</th>
<th>Availability Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unmanaged</td>
<td>50,000</td>
<td>90.0%</td>
<td>1</td>
</tr>
<tr>
<td>Managed</td>
<td>5,000</td>
<td>99.0%</td>
<td>2</td>
</tr>
<tr>
<td>Well Managed</td>
<td>500</td>
<td>99.9%</td>
<td>3</td>
</tr>
<tr>
<td>Fault Tolerant</td>
<td>50</td>
<td>99.99%</td>
<td>4</td>
</tr>
<tr>
<td>High-Availability</td>
<td>5</td>
<td>99.999%</td>
<td>5</td>
</tr>
<tr>
<td>Very-High-Availability</td>
<td>0.5</td>
<td>99.9999%</td>
<td>6</td>
</tr>
<tr>
<td>Ultra-Availability</td>
<td>0.05</td>
<td>99.99999%</td>
<td>7</td>
</tr>
</tbody>
</table>

**UnAvailability = MTTR/MTBF**

can cut it in ½ by cutting MTTR or MTBF

*From Jim Gray's "Talk at UC Berkeley on Fault Tolerance " 11/9/00*
HP claims HP–9000 server HW and HP–UX OS can deliver 99.999% availability guarantee “in certain pre-defined, pre-tested customer environments”
- Application faults?
- Operator faults?
- Environmental faults?

Collocation sites (lots of computers in 1 building on Internet) have
- 1 network outage per year (~1 day)
- 1 power failure per year (~1 day)

Microsoft Network unavailable recently for a day due to problem in Domain Name Server: if only outage per year, 99.7% or 2 Nines
I/O Performance

Metrics:
Response Time vs. Throughput

Response time = Queue + Device Service time
For better or worse, benchmarks shape a field
- Processor benchmarks classically aimed at response time for fixed sized problem
- I/O benchmarks typically measure throughput, possibly with upper limit on response times (or 90% of response times)

Transaction Processing (TP) (or On-line TP=OLTP)
- If bank computer fails when customer withdraw money, TP system guarantees account debited if customer gets $ & account unchanged if no $
- Airline reservation systems & banks use TP

Atomic transactions makes this work

Classic metric is Transactions Per Second (TPS)
Availability benchmark methodology

- Goal: quantify variation in QoS metrics as events occur that affect system availability
- Leverage existing performance benchmarks
  - to generate fair workloads
  - to measure & trace quality of service metrics
- Use fault injection to compromise system
  - hardware faults (disk, memory, network, power)
  - software faults (corrupt input, driver error returns)
  - maintenance events (repairs, SW/HW upgrades)
- Examine single–fault and multi–fault workloads
  - the availability analogues of performance micro– and macro–benchmarks
Example single-fault result

- Compares Linux and Solaris reconstruction
  - Linux: minimal performance impact but longer window of vulnerability to second fault
  - Solaris: large perf. impact but restores redundancy fast
Reconstruction policy

- **Linux**: favors performance over data availability
  - automatically-initiated reconstruction, idle bandwidth
  - virtually no performance impact on application
  - very long window of vulnerability (>1hr for 3GB RAID)

- **Solaris**: favors data availability over app. perf.
  - automatically-initiated reconstruction at high BW
  - as much as 34% drop in application performance
  - short window of vulnerability (10 minutes for 3GB)

- **Windows**: favors neither!
  - manually-initiated reconstruction at moderate BW
  - as much as 18% app. performance drop
  - somewhat short window of vulnerability (23 min/3GB)
Summary

- Disks: Arial Density now 30%/yr vs. 100%/yr in 2000s
- Fault $\Rightarrow$ Latent errors in system $\Rightarrow$ Failure in service
- Components often fail slowly
- Real systems: problems in maintenance, operation as well as hardware, software