Lecture 11 – Storage

Slides were used during lectures by
David Patterson, Berkeley, spring 2006

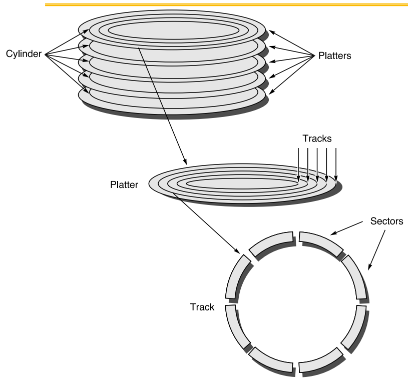
Case for Storage

- Shift in focus from computation to communication and storage 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 actually features used?
  - Operating System for storage
  - Compiler for processor
- Also has own performance theory—queuing theory—balances throughput vs. response time

Outline

- Magnetic Disks
- RAID
- Advanced Dependability/Reliability/Availability
- I/O Benchmarks, Performance and Dependability
- Intro to Queuing Theory
- The End

Disk Organization

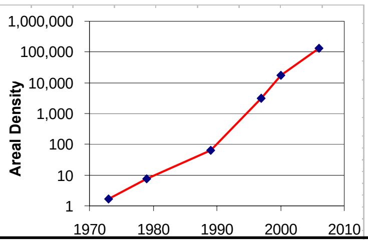


- 5000-30,000 tracks/surface
- 100-500 sectors/track

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

Year	Areal Density
1973	2
1979	8
1989	63
1997	3,090
2000	17,100
2006	130,000



Historical Perspective

- 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
- 1900s: Laptops => 2.5 inch drives
- 2000s: What new devices leading to new drives?

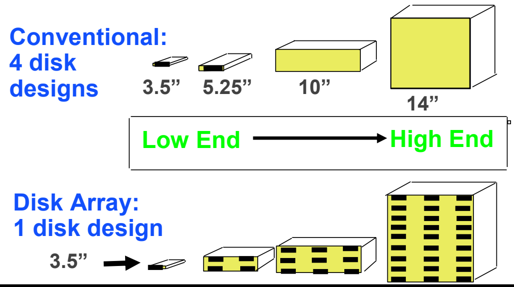
Future Disk Size and Performance

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

Year	Sequentially	Randomly (1 sector/seek)
1990	4 minutes	6 hours
2000	12 minutes	1 week(!)
2006	56 minutes	3 weeks (SCSI)
2006	171 minutes	7 weeks (SATA)

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?



Replace Small Number of Large Disks with Large Number of Small Disks! (1988 Disks)

	IBM 3390K	IBM 3.5" 0061	x70	
Capacity	20 GBytes	320 MBytes	23 GBytes	
Volume	97 cu. ft.	0.1 cu. ft.	11 cu. ft.	9X
Power	3 KW	11 W	1 KW	3X
Data Rate	15 MB/s	1.5 MB/s	120 MB/s	8X
I/O Rate	600 I/Os/s	55 I/Os/s	3900 IOs/s	6X
MTTF	250 KHrs	50 KHrs	??? Hrs	
Cost	$250K	$2K	$150K	

Disk Arrays have potential for large data and I/O rates, high MB per cu. ft., high MB per KW, but what about reliability?

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!

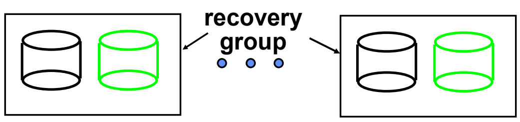
Arrays (without redundancy) too unreliable to be useful!

Hot spares support reconstruction in parallel with access: very high media availability can be achieved

Redundant Arrays of (Inexpensive) Disks

- 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

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)

Redundant Array of Inexpensive Disks RAID 3: Parity Disk

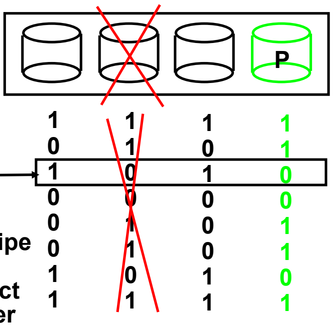
10010011
11001101
10010011
. . .

logical record

Striped physical records

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



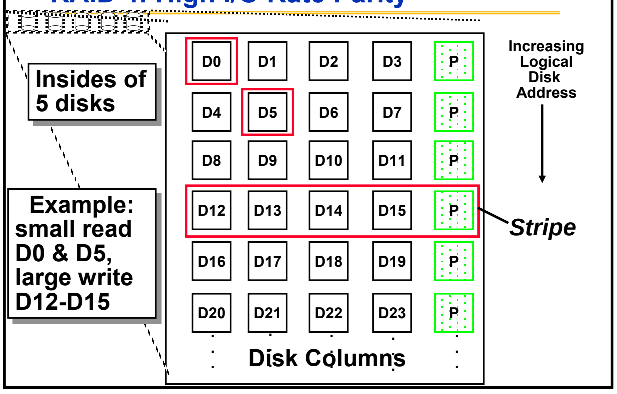
RAID 3

- 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

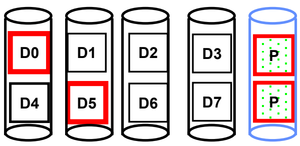
- RAID 3 relies on parity disk to discover errors on Read
- But every sector has an error detection field
- To catch errors on read, rely on error detection field vs. the parity disk
- Allows independent reads to different disks simultaneously

Redundant Arrays of Inexpensive Disks RAID 4: High I/O Rate Parity

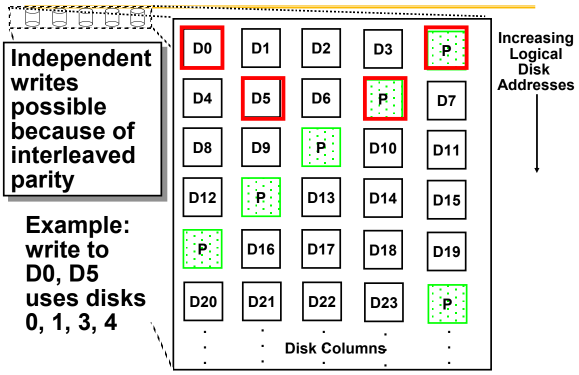


Inspiration for RAID 5

- 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



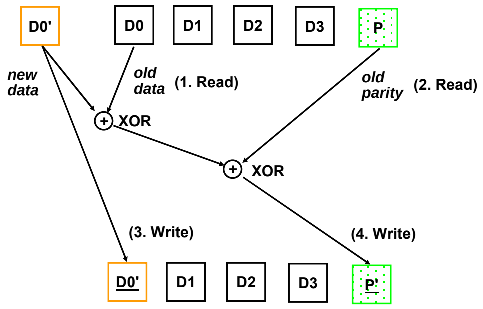
Redundant Arrays of Inexpensive Disks RAID 5: High I/O Rate Interleaved Parity



Problems of Disk Arrays: Small Writes

RAID-5: Small Write Algorithm

1 Logical Write = 2 Physical Reads + 2 Physical Writes



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

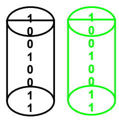
Data Disk 0	Data Disk 1	Data Disk 2	Data Disk 3	Row Parity	Diagonal Parity
0	1	2	3	4	0
1	2	3	4	0	1
2	3	4	0	1	2
3	4	0	1	2	3
4	0	1	2	3	4
0	1	2	3	4	0

Berkeley History: RAID-I

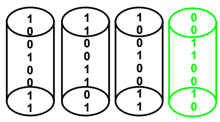
- **RAID-I (1989)**
  Consisted of a Sun 4/280 workstation with 128 MB of DRAM, four dual-string SCSI controllers, 28 5.25-inch SCSI disks and specialized disk striping software
- **Today RAID is $24 billion dollar industry, 80% nonPC disks sold in RAIDs**

Summary: RAID Techniques: Goal was performance, popularity due to reliability of storage

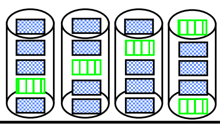
- ***Disk Mirroring, Shadowing (RAID 1)***
  Each disk is fully duplicated onto its "shadow"
  Logical write = two physical writes
  100% capacity overhead



- ***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?

International Federation for Information Processing (IFIP) Standard terminology

- 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 because 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 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**

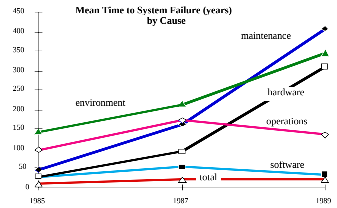
Fault Tolerance vs Disaster Tolerance

- 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

Case Studies - Tandem Trends

Reported MTTF by Component



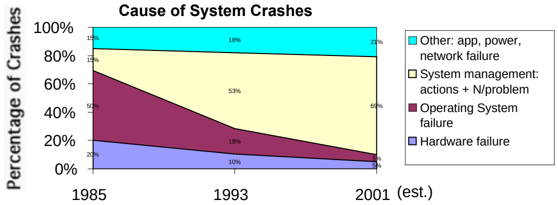
	1985	1987	1990	
SOFTWARE	2	53	33	Years
HARDWARE	29	91	310	Years
MAINTENANCE	45	162	409	Years
OPERATIONS	99	171	136	Years
ENVIRONMENT	142	214	346	Years
SYSTEM	8	20	21	Years

Problem: Systematic Under-reporting

From Jim Gray's "Talk at UC Berkeley on Fault Tolerance " 11/9/00

Is Maintenance the Key?

- **Rule of Thumb: Maintenance 10X HW**
  - **so over 5 year product life, ~ 95% of cost is maintenance**



- **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

Component	Total in System	Total Failed	% Failed
SCSI Controller	44	1	2.3%
SCSI Cable	39	1	2.6%
SCSI Disk	368	7	1.9%
IDE Disk	24	6	25.0%
Disk Enclosure -Backplane	46	13	28.3%
Disk Enclosure - Power Supply	92	3	3.3%
Ethernet Controller	20	1	5.0%
Ethernet Switch	2	1	50.0%
Ethernet Cable	42	1	2.3%
CPU/Motherboard	20	0	0%

Does Hardware Fail Fast?
4 of 384 Disks that failed in Tertiary Disk

Messages in system log for failed disk	No. log msgs	Duration (hours)
Hardware Failure (Peripheral device write fault [for] Field Replaceable Unit)	1763	186
Not Ready (Diagnostic failure: ASCQ = Component ID [of] Field Replaceable Unit)	1460	90
Recovered Error (Failure Prediction Threshold Exceeded [for] Field Replaceable Unit)	1313	5
Recovered Error (Failure Prediction Threshold Exceeded [for] Field Replaceable Unit)	431	17

High Availability System Classes
Goal: Build Class 6 Systems

System Type	Unavailable (min/year)	Availability	Availability Class
Unmanaged	50,000	90.%	1
Managed	5,000	99.%	2
Well Managed	500	99.9%	3
Fault Tolerant	50	99.99%	4
High-Availability	5	99.999%	5
Very-High-Availability	.5	99.9999%	6
Ultra-Availability	.05	99.99999%	7

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

How Realistic is "5 Nines"?

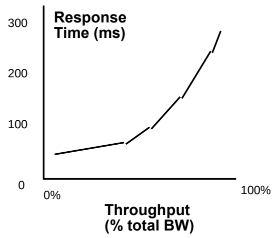
- HP claims HP-9000 server HW and HP-UX OS can deliver 99.999% availability guarantee "in certain pre-defined, pre-tested customer environments"
  - Application faults?
  - Operator faults?
  - Environmental faults?
- Collocation sites (lots of computers in 1 building on Internet) have
  - 1 network 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

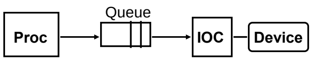
Outline

- Magnetic Disks
- RAID
- Advanced Dependability/Reliability/Availability
- **I/O Benchmarks, Performance and Dependability**
- **Intro to Queuing Theory**
- **The End**

I/O Performance

Metrics:
Response Time
vs. Throughput





Response time = Queue + Device Service time

I/O Benchmarks

- 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)

I/O Benchmarks: Transaction Processing

- Early 1980s great interest in OLTP
  - Expecting demand for high TPS (e.g., ATM machines, credit cards)
  - Tandem's success implied medium range OLTP expands
  - Each vendor picked own conditions for TPS claims, report only CPU times with widely different I/O
  - Conflicting claims led to disbelief of all benchmarks ⇒ chaos
- 1984 Jim Gray (Tandem) distributed paper to Tandem + 19 in other companies propose standard benchmark
- Published "A measure of transaction processing power," Datamation, 1985 by Anonymous et. al
  - To indicate that this was effort of large group
  - To avoid delays of legal department of each author's firm
  - Still get mail at Tandem to author "Anonymous"
- Led to Transaction Processing Council in 1988
  - www.tpc.org

I/O Benchmarks: TP1 by Anon et. al

- DebitCredit Scalability: size of account, branch, teller, history function of throughput

TPS	Number of ATMs	Account-file size
10	1,000	0.1 GB
100	10,000	1.0 GB
1,000	100,000	10.0 GB
10,000	1,000,000	100.0 GB

  - Each input TPS =>100,000 account records, 10 branches, 100 ATMs
  - Accounts must grow since a person is not likely to use the bank more frequently just because the bank has a faster computer!
- Response time: 95% transactions take ≤ 1 second
- Report price (initial purchase price + 5 year maintenance = cost of ownership)
- Hire auditor to certify results

Unusual Characteristics of TPC

- Price is included in the benchmarks
  - cost of HW, SW, and 5-year maintenance agreements included ⇒ price-performance as well as performance
- The data set generally must scale in size as the throughput increases
  - trying to model real systems, demand on system and size of the data stored in it increase together
- The benchmark results are audited
  - Must be approved by certified TPC auditor, who enforces TPC rules ⇒ only fair results are submitted
- Throughput is the performance metric but response times are limited
  - eg, TPC-C: 90% transaction response times < 5 seconds
- An independent organization maintains the benchmarks
  - COO ballots on changes, meetings, to settle disputes...

TPC Benchmark History/Status

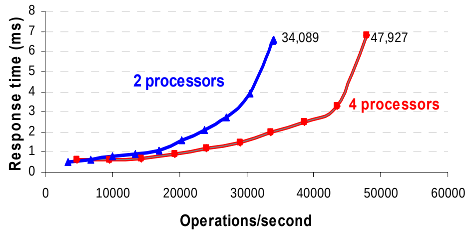
Benchmark	Data Size (GB)	Performance Metric	1st Results
A: Debit Credit (retired)	0.1 to 10	transactions/s	Jul-90
B: Batch Debit Credit (retired)	0.1 to 10	transactions/s	Jul-91
C: Complex Query OLTP	**100 to 3000 (min. 07 * tpm)**	**new order trans/min (tpm)**	**Sep-92**
D: Decision Support (retired)	100, 300, 1000	queries/hour	Dec-95
H: Ad hoc decision support	**100, 300, 1000**	**queries/hour**	**Oct-99**
R: Business reporting decision support (retired)	1000	queries/hour	Aug-99
W: Transactional web	**~ 50, 500**	**web inter-actions/sec.**	**Jul-00**
App: app. server & web services		**Web Service Interactions/sec (SIPS)**	**Jun-05**

I/O Benchmarks via SPEC

- **SFS 3.0 Attempt by NFS companies to agree on standard benchmark**
  - Run on multiple clients & networks (to prevent bottlenecks)
  - Same caching policy in all clients
  - Reads: 85% full block & 15% partial blocks
  - Writes: 50% full block & 50% partial blocks
  - Average response time: 40 ms
  - Scaling: for every 100 NFS ops/sec, increase capacity 1GB
- **Results: plot of server load (throughput) vs. response time & number of users**
  - Assumes: 1 user => 10 NFS ops/sec
  - 3.0 for NFS 3.0
- **Added SPECMail (mailserver), SPECWeb (webserver) benchmarks**

2005 Example SPEC SFS Result: NetApp FAS3050c NFS servers

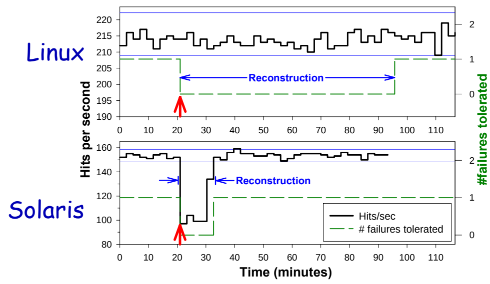
- 2.8 GHz Pentium Xeon microprocessors, 2 GB of DRAM per processor, 1GB of Non-volatile memory per system
- 4 FDDI networks; 32 NFS Daemons, 24 GB file size
- 168 fibre channel disks: 72 GB, 15000 RPM, 2 or 4 FC controllers



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 (2)

- **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)

Introduction to Queuing Theory

- **More interested in long term, steady state than in startup => Arrivals = Departures**
- **Little's Law:**
  Mean number tasks in system = arrival rate *x* mean response time
  - Observed by many, Little was first to prove
- **Applies to any system in equilibrium, as long as black box not creating or destroying tasks**

Deriving Little's Law

- $Time_{observe}$ = elapsed time that observe a system
- $Number_{task}$ = number of (overlapping) tasks during $Time_{observe}$
- $Time_{accumulated}$ = sum of elapsed times for each task

Then

- Mean number tasks in system = $Time_{accumulated}$ / $Time_{observe}$
- Mean response time = $Time_{accumulated}$ / $Number_{task}$
- Arrival Rate = $Number_{task}$ / $Time_{observe}$

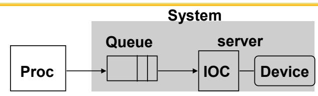
Factoring RHS of 1st equation

- $Time_{accumulated} / Time_{observe} = Time_{accumulated} / Number_{task} \times Number_{task} / Time_{observe}$

Then get Little's Law:

- Mean number tasks in system = Arrival Rate x Mean response time

A Little Queuing Theory: Notation



- **Notation:**
  - ***$Time_{server}$*** **average time to service a task**
  - **Average service rate = 1 / *$Time_{server}$* (traditionally μ)**
  - ***$Time_{queue}$*** **average time/task in queue**
  - ***$Time_{system}$*** **average time/task in system**
    - ***= $Time_{queue}$ + $Time_{server}$***
  - ***Arrival rate*** **avg no. of arriving tasks/sec (traditionally λ)**
  - ***$Length_{server}$*** **average number of tasks in service**
  - ***$Length_{queue}$*** **average length of queue**
  - ***$Length_{system}$*** **average number of tasks in service**
    - ***= $Length_{queue}$ + $Length_{server}$***
- **Little's Law: *$Length_{server}$ = Arrival rate x $Time_{server}$***
  (Mean number tasks = arrival rate x mean service time)

Server Utilization

- **For a single server, service rate = 1 / *$Time_{server}$***
- **Server utilization must be between 0 and 1, since system is in equilibrium (arrivals = departures); often called traffic intensity, traditionally ρ)**
- **Server utilization**
  = mean number tasks in service
  = Arrival rate x $Time_{server}$
- **What is disk utilization if get 50 I/O requests per second for disk and average disk service time is 10 ms (0.01 sec)?**
- **Server utilization = 50/sec x 0.01 sec = 0.5**
- **Or server is busy on average 50% of time**

Time in Queue vs. Length of Queue

- **We assume First In First Out (FIFO) queue**
- **Relationship of time in queue (*$Time_{queue}$*) to mean number of tasks in queue (*$Length_{queue}$*) ?**
- ***$Time_{queue}$ = $Length_{queue}$ x $Time_{server}$***
  + "Mean time to complete service of task when new task arrives if server is busy"
- **New task can arrive at any instant; how predict last part?**
- **To predict performance, need to know sometime about distribution of events**

Distribution of Random Variables

- **A variable is random if it takes one of a specified set of values with a specified probability**
  - Cannot know exactly next value, but may know probability of all possible values
- **I/O Requests can be modeled by a random variable because OS normally switching between several processes generating independent I/O requests**
  - Also given probabilistic nature of disks in seek and rotational delays
- **Can characterize distribution of values of a random variable with discrete values using a *histogram***
  - Divides range between the min & max values into *buckets*
  - Histograms then plot the number in each bucket as columns
  - Works for discrete values e.g., number of I/O requests?
- **What about if not discrete? Very fine buckets**

Characterizing distribution of a random variable

Need mean time and a measure of variance

For mean, use weighted arithmetic mean (WAM):

- **fi = frequency of task i**
- **Ti = time for tasks I**

Weighted arithmetic mean = $f1 \times T1 + f2 \times T2 + \ldots + fn \times Tn$

For variance, instead of standard deviation, use Variance (square of standard deviation) for WAM:

Variance = $(f1 \times T1^2 + f2 \times T2^2 + \ldots + fn \times Tn^2) - WAM^2$

- **If time is miliseconds, Variance units are square milliseconds!**

Got a unitless measure of variance?

Squared Coefficient of Variance (C^2)

- C^2 = Variance / WAM^2
  ⇒ C = sqrt(Variance)/WAM = StDev/WAM
  - Unitless measure
- Trying to characterize random events, but need distribution of random events with tractable math
- Most popular such distribution is *exponential distribution*, where C = 1
- Note using constant to characterize variability about the mean
  - Invariance of C over time ⇒ history of events has no impact on probability of an event occurring now
  - Called *memoryless*, an important assumption to predict behavior
  - (Suppose not; then have to worry about the exact arrival times of requests relative to each other ⇒ make math not tractable!)

Poisson Distribution

- Most widely used exponential distribution is Poisson
- Described by probability mass function:
  Probability (k) = $e^{-a} \times a^k / k!$
  - where a = Rate of events x Elapsed time
- If interarrival times exponentially distributed & use arrival rate from above for rate of events, number of arrivals in time interval *t* is a *Poisson process*

Time in Queue

- Time new task must wait for server to complete a task assuming server busy
  - Assuming it's a Poisson process
- Average residual service time = ½ x Arithmetic mean x (1 + C^2)
  - When distribution is not random & all values = average ⇒ standard deviation is 0 ⇒ C is 0
    ⇒ average residual service time
    = half average service time
  - When distribution is random & Poisson ⇒ C is 1
    ⇒ average residual service time
    = weighted arithmetic mean

Time in Queue

- All tasks in queue ($Length_{queue}$) ahead of new task must be completed before task can be serviced
  - Each task takes on average $Time_{server}$
  - Task at server takes average residual service time to complete
- Chance server is busy is *server utilization* ⇒ expected time for service is Server utilization × Average residual service time
- $Time_{queue}$ = $Length_{queue}$ x $Time_{server}$ + Server utilization x Average residual service time
- Substituting definitions for $Length_{queue}$, Average residual service time, & rearranging:
  $Time_{queue}$ = $Time_{server}$ x Server utilization/(1-Server utilization)

M/M/1 Queuing Model

- System is in equilibrium
- Times between 2 successive requests arriving, *"interarrival times"*, are exponentially distributed
- Number of sources of requests is unlimited "infinite population model"
- Server can start next job immediately
- Single queue, no limit to length of queue, and FIFO discipline, so all tasks in line must be completed
- There is one server
- Called M/M/1 (book also derives M/M/m)
  1. Exponentially random request arrival (C^2 = 1)
  2. Exponentially random service time (C^2 = 1)
  3. 1 server
  - *M* standing for Markov, mathematician who defined and analyzed the memoryless processes

Example

40 disk I/Os / sec, requests are exponentially distributed, and average service time is 20 ms

⇒ Arrival rate/sec = 40, $Time_{server}$ = 0.02 sec

1. On average, how utilized is the disk?
   Server utilization = Arrival rate × $Time_{server}$
   = 40 x 0.02 = 0.8 = 80%
2. What is the average time spent in the queue?
   $Time_{queue}$ = $Time_{server}$ x Server utilization/(1-Server utilization)
   = 20 ms x 0.8/(1-0.8) = 20 x 4 = 80 ms
3. What is the average response time for a disk request, including the queuing time and disk service time?
   $Time_{system}$=$Time_{queue}$ + $Time_{server}$ = 80+20 ms = 100 ms

How much better with 2X faster disk?

Average service time is 10 ms

⇒ Arrival rate/sec = 40, $Time_{server}$ = 0.01 sec

1. On average, how utilized is the disk?

   Server utilization = Arrival rate × $Time_{server}$ = 40 x 0.01 = 0.4 = 40%

2. What is the average time spent in the queue?

   $Time_{queue}$ = $Time_{server}$ x Server utilization/(1-Server utilization)

   = 10 ms x 0.4/(1-0.4) = 10 x 2/3 = 6.7 ms

3. What is the average response time for a disk request, including the queuing time and disk service time?

   $Time_{system}$=$Time_{queue}$ + $Time_{server}$=6.7+10 ms = 16.7 ms

   6X faster response time with 2X faster disk!

Value of Queuing Theory in practice

- Learn quickly do not try to utilize resource 100% but how far should back off?
- Allows designers to decide impact of faster hardware on utilization and hence on response time
- Works surprisingly well

Cross cutting Issues: Buses ⇒ point-to-point links and switches

Standard	width	length	Clock rate	MB/s	Max
(Parallel) ATA	8b	0.5 m	133 MHz	133	2
Serial ATA	2b	2 m	3 GHz	300	?
(Parallel) SCSI	16b	12 m	80 MHz (DDR)	320	15
Serial Attach SCSI	1b	10 m	--	375	16,256
PCI	32/64	0.5 m	33 / 66 MHz	533	?
PCI Express	2b	0.5 m	3 GHz	250	?

- No. bits and BW is per direction ⇒ 2X for both directions (not shown).
- Since use fewer wires, commonly increase BW via versions with 2X-12X the number of wires and BW

Storage Example: Internet Archive

- Goal of making a historical record of the Internet
  - Internet Archive began in 1996
  - Wayback Machine interface perform time travel to see what the website at a URL looked like in the past
- It contains over a petabyte (10^{15} bytes), and is growing by 20 terabytes (10^{12} bytes) of new data per month
- In addition to storing the historical record, the same hardware is used to crawl the Web every few months to get snapshots of the Internet.

Internet Archive Cluster

- 1U storage node PetaBox GB2000 from Capricorn Technologies
- Contains 4 500 GB Parallel ATA (PATA) disk drives, 512 MB of DDR266 DRAM, one 10/100/1000 Ethernet interface, and a 1 GHz C3 Processor from VIA (80x86).
- Node dissipates ≈ 80 watts
- 40 GB2000s in a standard VME rack, ⇒ 80 TB of raw storage capacity
- 40 nodes are connected with a 48-port 10/100 or 10/100/1000 Ethernet switch
- Rack dissipates about 3 KW
- 1 PetaByte = 12 racks

Estimated Cost

- Via processor, 512 MB of DDR266 DRAM, ATA disk controller, power supply, fans, and enclosure = $500
- 7200 RPM Parallel ATA drives holds 500 GB = $375.
- 48-port 10/100/1000 Ethernet switch and all cables for a rack = $3000.
- Cost $84,500 for a 80-TB rack.
- 160 Disks are ≈ 60% of the cost
- Other costs: power, space, ……

Estimated Performance

- 7200 RPM Parallel ATA drives holds 500 GB, has an average time seek of 8.5 ms, transfers at 50 MB/second from the disk. The PATA link speed is 133 MB/second.
  - performance of the VIA processor is 1000 MIPS.
  - operating system uses 50,000 CPU instructions for a disk I/O.
  - network protocol stacks uses 100,000 CPU instructions to transmit a data block between the cluster and the external world
- ATA controller overhead is 0.1 ms to perform a disk I/O.
- Average I/O size is 16 KB for accesses to the historical record via the Wayback interface, and 50 KB when collecting a new snapshot
- Disks are limit: ≈ 75 I/Os/s per disk, 300/s per node, 12000/s per rack, or about 200 to 600 Mbytes/sec Bandwidth per rack
- Switch needs to support 1.6 to 3.8 Gbits/second over 40 Gbit/sec links

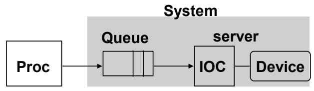
Estimated Reliability

- CPU/memory/enclosure MTTF is 1,000,000 hours (x 40)
- PATA Disk MTTF is 125,000 hours (x 160)
- PATA controller MTTF is 500,000 hours (x 40)
- Ethernet Switch MTTF is 500,000 hours (x 1)
- Power supply MTTF is 200,000 hours (x 40)
- Fan MTTF is 200,000 hours (x 40)
- PATA cable MTTF is 1,000,000 hours (x 40)
- MTTF for the system is 531 hours (≈ 3 weeks)
- 70% of time failures are disks
- 20% of time failures are fans or power supplies

Summary (1/2)

- Disks: Arial Density now 30%/yr vs. 100%/yr in 2000s
- RAID Techniques: Goal was performance, popularity due to reliability of storage
- TPC: price performance as normalizing configuration feature
  - Auditing to ensure no foul play
  - Throughput with restricted response time is normal measure
- Fault ⇒ Latent errors in system ⇒ Failure in service
- Components often fail slowly
- Real systems: problems in maintenance, operation as well as hardware, software

Summary (2/2)



- Little's Law: $Length_{system} = rate \times Time_{system}$
  (Mean number customers = arrival rate x mean service time)
- Appreciation for relationship of latency and utilization:
  - $Time_{system} = Time_{server} + Time_{queue}$
  - $Time_{queue} = Time_{server}$ x Server utilization/(1-Server utilization)

The End

- The last lecture
  - chapter 6: *Storage Systems*
- Exam
  - Mon Jan 14th 2008, 14-17h
  - chap 1-6, app A, C & F
  - remark: sample exams on website based on previous edition of book
- Assignment
  - deadline 2b: Dec 3rd
  - deadline 3: Dec 24th (intro by Eyal on Wed Dec 5th, 13.45h)