EEL 5764: Graduate Computer Architecture

Storage

Ann Gordon-Ross
Electrical and Computer Engineering
University of Florida

http://www.ann.ece.ufl.edu/

These slides are provided by:
David Patterson
Electrical Engineering and Computer Sciences, University of California, Berkeley
Modifications/additions have been made from the originals

Case for Storage

- Shift in focus from computation to communication and storage of information
  - E.g., Cray Research (build the fasted computer possible) vs. Google/Yahoo (massive communication and storage)
  - "The Computing Revolution" (1960s to 1980s) ⇒ "The Information Age" (1990 to today)
    - » Cray is struggling while Google is flourishing
- Storage emphasizes reliability and scalability as well as cost-performance

Case for Storage

- Compiler determines what architecture to use
- OS determines the storage
- Different focus and critical issues
  - If a program crashes, just restart program, user is mildly annoyed
  - If data is lost, users are very angry
- Also has own performance theory—queuing theory—balances throughput vs. response time

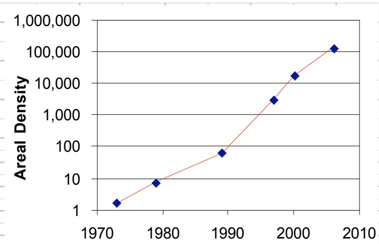
Outline

- Magnetic Disks
- RAID in the past
- RAID in the present
- Advanced Dependability/Reliability/Availability
- I/O Benchmarks, Performance and Dependability
- Intro to Queueing Theory

Disk Figure of Merit: Areal Density

- Designers care about areal density
  - Areal density = Bits Per Inch (BPI) X Tracks Per Inch (TPI)
- Graph shows large gains in density over time
  - Mechanical engineering and error correcting codes have allowed for these increases

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



Historical Perspective

- First disk invented by IBM
  - 1956 IBM Ramac — early 1970s Winchester
  - Developed for mainframe computers
  - proprietary interfaces
- Form factor (item using disk) and capacity drives market more than performance
- 1970s developments
  - 5.25 inch floppy disk formfactor (microcode into mainframe)
  - Emergence of industry standard disk interfaces
- Mid 1980s: Client/server computing
  - 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

- Capacity growth (60%/yr) overshoots bandwidth growth (40%/yr)
- Slow improvement in seek, rotation (8%/yr)
- Time to read whole disk

Year	Sequentially (bandwidth)		Randomly (latency) (1 sector/seek)	
1990	4 minutes		6 hours	
		3x		24x
2000	12 minutes		1 week(!)	
		4.6x		3x
2006	56 minutes		3 weeks (SCSI)	
		3x		2.3x
2006	171 minutes		7 weeks (SATA)	

- Disks are now like tapes, random access is slow!

What have Magnetic Disks been doing?

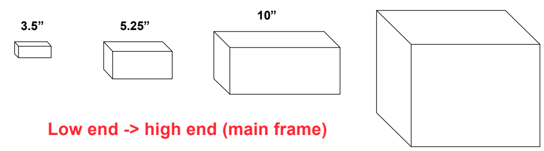
- $/MB: improving 25% per year
- Evolving to smaller physical sizes
  - 14” -> 10” > 8” ->5.25” -> 3.5” -> 2.5” ->1.6”? -> 1”?
- Can we use a lot of smaller disks to close the gap in performance between disks and CPU?
  - Smaller platter equates to shorter seek time

Outline

- Magnetic Disks
- RAID in the past
- RAID in the present
- Advanced Dependability/Reliability/Availability
- I/O Benchmarks, Performance and Dependability
- Intro to Queueing Theory

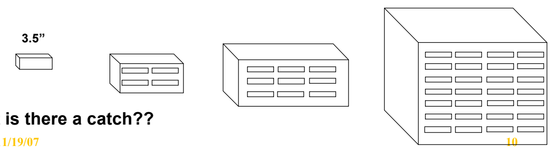
Manufacturing Advantages of Disk Arrays (1987)

- Conventional: 4 disk designs (4 product teams):



Low end -> high end (main frame)

- Disk array: 1 disk design



But is there a catch??

Arrays of Disks to Close the Performance Gap (1988 disks)

- Replace small number of large disks with a large number of small disks

	IBM 3380	Smaller disk	Smaller disk x50
Data Capacity	7.5 GBytes	320 MBytes	16 GBytes
Volume	24 cu. ft.	0.2 cu. ft.	20 cu. ft
Power	1.65 KW	10 W	0.5 KW
Data Rate	12 MB/s	2 MB/s	100 MB/s
I/O Rate	200 I/Os/s	40 I/Os/s	2000 I/Os/s
Cost	$100k	$2k	$100k

- Data arrays have potential for
  - Large data and I/O rates
  - High MB per cu. ft
  - High MB per KW

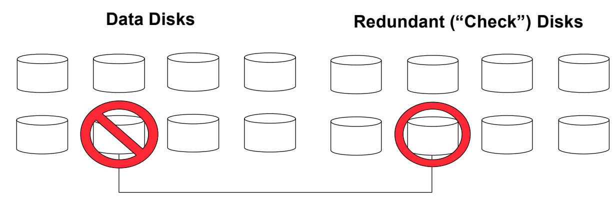
Array Reliability

- 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!
- Originally concerned with performance, but reliability became an issue

Improving Reliability with Redundancy

- Add redundant drives to handle failures
  - Redundant
  - Array of
  - Inexpensive (Independent? - First disks weren't cheap)
  - Disks
- Redundancy offers 2 advantages:
  - Data not lost: Reconstruct data onto new disks
  - Continuous operation in presence of failure
- Several RAID organizations
  - Mirroring/Shadowing (Level 1 RAID)
  - ECC (Level 2 RAID)
  - Parity (Level 3 RAID)
  - Rotated Parity (Level 5 RAID)
  - Levels were used to distinguish between work at different institutions

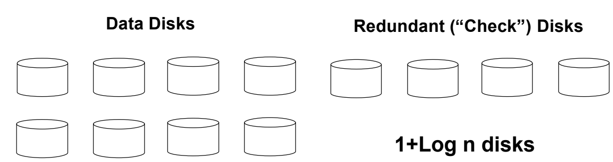
Redundancy via Mirroring/Shadowing (Level 1 RAID)



Redundancy via Mirroring/Shadowing (Level 1 RAID)

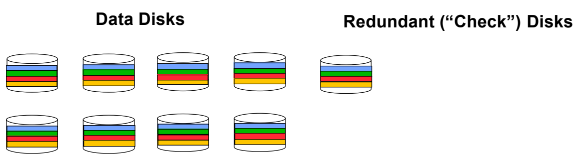
- Each disk is fully duplicated onto its "mirror"
  Very high availability can be achieved
- Bandwidth sacrifice on write:
  Logical write = two physical writes
  - Reads may be optimized
- Most expensive solution: 100% capacity overhead

Redundancy via EEC (Level 2 RAID)



Used idea of error correction codes from memory and applied to disks. Parity is calculated over subsets of disks, and you can figure out which disk failed and correct it. Single error correction

Redundancy via Parity (Level 3 RAID)

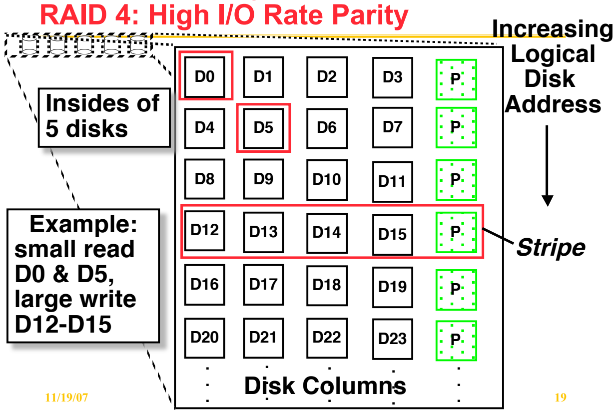


- Single parity disk - parity is striped across disks
- Now only need a single redundant disk
  - Now attractive for low cost solution

Inspiration for RAID 4

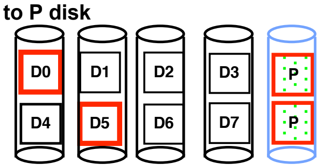
- RAID 3 relies on parity disk to discover errors on Read
- But every sector has an error detection field
- To catch errors on read, rely on error detection field on the disk vs. the parity disk
- Allows independent reads to different disks simultaneously
- Define:
  - Small read/write - read/write to one disk
  - Large read/write - read/write to more than one disk

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

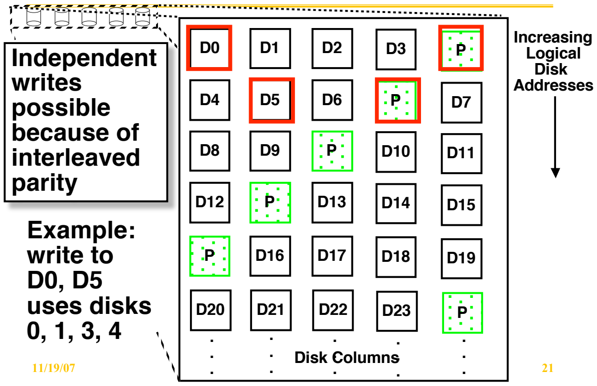


Inspiration for RAID 5

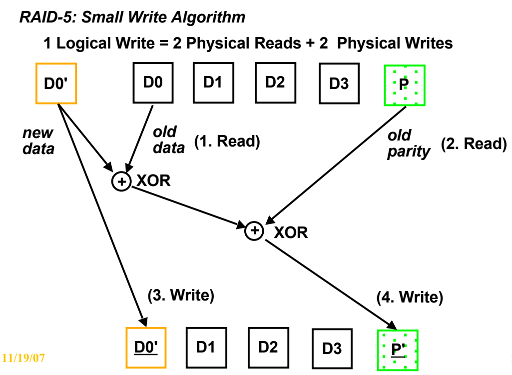
- RAID 4 works well for small reads
- Small writes:
  - Option 1: read other data disks, create new sum and write to Parity Disk (P)
  - Option 2: since P has old sum, compare old data to new data, add the difference to P
- Parity disk becomes bottleneck: Write to D0, D5 both also write to P disk



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



Problems of Disk Arrays: Small Writes



Outline

- Magnetic Disks
- RAID in the past
- **RAID in the present**
- **Advanced Dependability/Reliability/Availability**
- **I/O Benchmarks, Performance and Dependability**
- **Intro to Queueing Theory**

RAID 6: Recovering from 2 failures

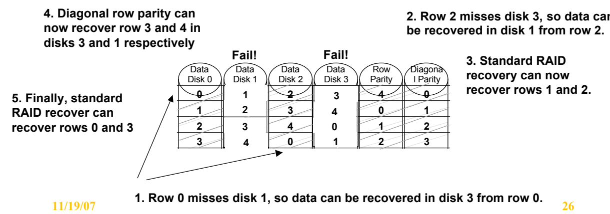
- **RAID 6 was always there but not so popular**
  - **Has recently become more popular. Why?**
- **Recover from more than 1 failure - Why?**
  - operator accidentally replaces the wrong disk during a failure
  - since disk bandwidth is growing more slowly than disk capacity, the MTT Repair a disk in a RAID system is increasing
    - » Long time to copy data back to disk after replacement
    - » 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
    - » **Uncorrectable error - ECC doesn't catch. Insert another error**

RAID 6: Recovering from 2 failures

- Recovering from 2 failures
  - Network Appliance's (make NSF file servers primarily) *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.
  - 2 check disks - row and diagonal parity
  - 2 ways to calculate parity
- Row parity disk is just like in RAID 4
  - Even parity across the other n-2 data blocks in its stripe
  - So n-2 disks contain data and 2 do not for each parity stripe
- Each block of the diagonal parity disk contains the even parity of the blocks in the same diagonal
  - Each diagonal does not cover 1 disk, hence you only need n-1 diagonals to protect n disks

Example n=5

- Assume disks 1 and 3 fail
- Can't recover using row parity because 2 data blocks are missing
- However, we can use row parity 0 since it covers every disk except disk 1, thus we can recover some information on disk 3
- Recover in an iterative fashion, alternating between row and diagonal parity recovery



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**

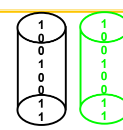
Summary: RAID Techniques: Goal was performance, popularity due to reliability of storage

- ***Disk Mirroring, Shadowing (RAID 1)***

  Each disk is fully duplicated onto its "shadow"

  Logical write = two physical writes

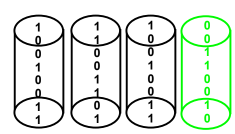
  100% capacity overhead



- ***Parity Data Bandwidth Array (RAID 3)***

  Parity computed horizontally

  Logically a single high data bw disk

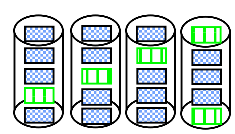


- ***High I/O Rate Parity Array (RAID 5)***

  Interleaved parity blocks

  Independent reads and writes

  Logical write = 2 reads + 2 writes



Outline

- Magnetic Disks
- RAID in the past
- RAID in the present
- **Advanced Dependability/Reliability/Availability**
- **I/O Benchmarks, Performance and Dependability**
- **Intro to Queueing Theory**

Definitions

- **Examples on why precise definitions so important for reliability**
  - **Confusion between different communities**
- 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?

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

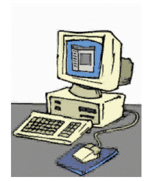
Fault Categories

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

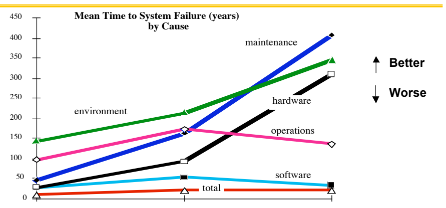


- Fault-Tolerance (or more properly, Error-Tolerance): **mask local faults (prevent errors from becoming failures)**
  - RAID disks
  - Uninterruptible Power Supplies
  - Cluster Failover
- Disaster Tolerance: **masks site errors (prevent site errors from causing service failures) - Could wipe everything out**
  - 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

Why do computers fail reported MTTF by Component



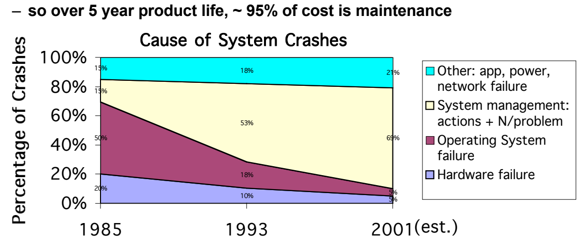
	1985	1987	1990	
SOFTWARE	2	53	33	Years
HARDWARE	29	91	310	Years
MAINTENANCE	45	162	409	Years
OPERATIONS	99	171	136	Years
ENVIRONMENT	142	214	346	Years
SYSTEM	8	20	21	Years

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 costs 10X more than HW**
  - so over 5 year product life, ~ 95% of cost is maintenance



- **Hard to quantify human operator failures**
  - People may not be truthful if their job may depend on it

HW Failures in Real Systems: Tertiary Disks

- 20 PC cluster in seven 7-foot high, 19-inch wide racks
  - 368 8.4 GB, 7200 RPM, 3.5-inch IBM disks
  - P6-200MHz with 96 MB of DRAM each
  - FreeBSD 3.0
  - 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

There were early warnings in the logs! Could just monitor logs.

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

Quantifying Availability

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 for a day due to problem in Domain Name Server: if only outage per year, 99.7% or 2 Nines
  - Needed 250 years of interruption free service to meet their target “nines”

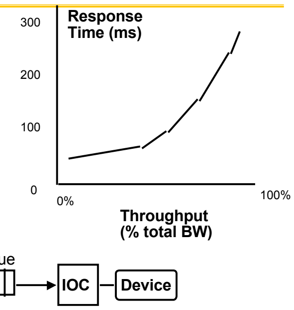
Outline

- Magnetic Disks
- RAID in the past
- RAID in the present
- Advanced Dependability/Reliability/Availability
- I/O Benchmarks, Performance and Dependability
- Intro to Queueing Theory

11/19/07 41

I/O Performance

Metrics:
Response Time
vs. Throughput



Response time = Queue + Device Service time

11/19/07 42

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)
  - Systems must promise some QOS
    - » 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)

11/19/07 43

I/O Benchmarks: Transaction Processing

- Early 1980s great interest in OLTP
  - Demand increasing
  - Hard to compare systems
    - » 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
- Need standard benchmarks
  - 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
  - Berkley still gets mail at Tandem to author "Anonymous"
- Led to Transaction Processing Council in 1988
  - www.tpc.org

11/19/07 44

I/O Benchmarks: TP1 by Anon et. al

- **Scalability requirement**
  - –Who cares if you can get 1M/sec (TPS) on a single record
  - –Need to scale number of records with total transactions

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
- **Response time**
  - –Not all transaction have to happen under the threshold
  - – 95% transactions take ≤ 1 second
- **Price factored in**
  - –(initial purchase price + 5 year maintenance = cost of ownership)
- **Hire auditor to certify results**

11/19/07 45

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
  - size of the data stored
- **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...

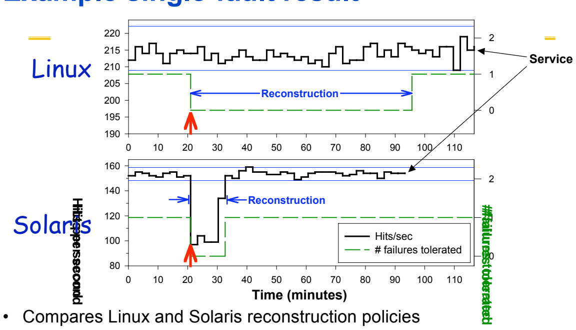
11/19/07 46

Availability benchmark methodology

- **Goal: quantify variation in QoS metrics as events occur that affect system availability**
- **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)
- **Example: Inject error and see how RAID handled it**

11/19/07 47

Example single-fault result



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

11/19/07 48

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)

Outline

- Magnetic Disks
- RAID in the past
- RAID in the present
- Advanced Dependability/Reliability/Availability
- I/O Benchmarks, Performance and Dependability
- **Intro to Queueing Theory**

Introduction to Queueing Theory

- **Interested in evaluating the system while in equilibrium**
  - Move past system startup
  - Arrivals = Departures
  - Queue won't overflow
- **Once in equilibrium, what is the utilization and response time**
- **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

- $\text{Time}_{\text{observe}}$ = elapsed time that observe a system
- $\text{Number}_{\text{task}}$ = number of (overlapping) tasks during $\text{Time}_{\text{observe}}$
- $\text{Time}_{\text{accumulated}}$ = sum of elapsed times for each task

Then

- Mean number tasks in system = $\text{Time}_{\text{accumulated}} / \text{Time}_{\text{observe}}$
- Mean response time = $\text{Time}_{\text{accumulated}} / \text{Number}_{\text{task}}$
- Arrival Rate = $\text{Number}_{\text{task}} / \text{Time}_{\text{observe}}$

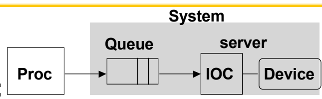
Factoring RHS of 1st equation

- $\text{Time}_{\text{accumulated}} / \text{Time}_{\text{observe}} = \text{Time}_{\text{accumulated}} / \text{Number}_{\text{task}} \times \text{Number}_{\text{task}} / \text{Time}_{\text{observe}}$

Then get Little's Law:

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

A Little Queuing Theory (Inside the Black Box): 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
  $Length_{system}$ = *$Length_{queue}$* + *$Length_{server}$*
- 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):
- f_i = 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 = $(f1 \times T1^2 + f2 \times T2^2 + \ldots + fn \times Tn^2) - WAM^2$
  - Problem - If time is miliseconds, Variance units are square milliseconds!?!?
- Got a unitless measure of variance?

Squared Coefficient of Variance (C^2)

- Get rid of squared time
  - 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!)
  - Assumptions are made to make math tractable, but works better than it might appear

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 are exponentially distributed & use arrival rate from above for rate of events, then the number of arrivals in time interval *t* is a *Poisson process*

Time in Queue - Residual Waiting Time

- 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 are exactly the 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)
- So, given a set of I/O requests, you can determine how many disks you need

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 Queueing 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