15-440 Distributed Systems

RAID

Thanks to Greg Ganger and Remzi Arapaci-Dusseau for slides

Replacement Rates

HPC1		COM1		COM2	
Component	**%**	**Component**	**%**	**Component**	**%**
Hard drive	30.6	Power supply	34.8	Hard drive	49.1
Memory	28.5	Memory	20.1	Motherboard	23.4
Misc/Unk	14.4	Hard drive	18.1	Power supply	10.1
CPU	12.4	Case	11.4	RAID card	4.1
motherboard	4.9	Fan	8	Memory	3.4
Controller	2.9	CPU	2	SCSI cable	2.2
QSW	1.7	SCSI Board	0.6	Fan	2.2
Power supply	1.6	NIC Card	1.2	CPU	2.2
MLB	1	LV Pwr Board	0.6	CD-ROM	0.6
SCSI BP	0.3	CPU heatsink	0.6	Raid Controller	0.6

Outline

- Using multiple disks
  - Why have multiple disks?
  - problem and approaches
- RAID levels and performance
- Estimating availability

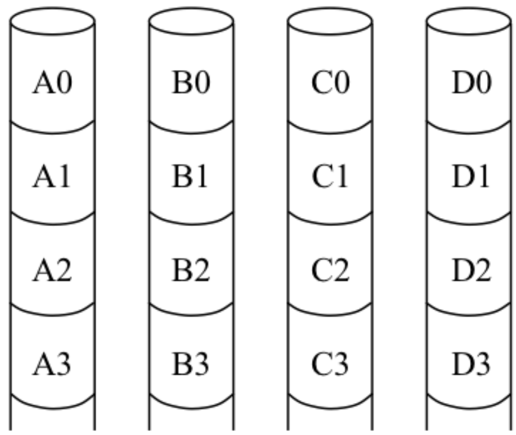
3

Motivation: Why use multiple disks?

- Capacity
  - More disks allows us to store more data
- Performance
  - Access multiple disks in parallel
  - Each disk can be working on independent read or write
  - Overlap seek and rotational positioning time for all
- Reliability
  - Recover from disk (or single sector) failures
  - Will need to store multiple copies of data to recover

- So, what is the simplest arrangement?

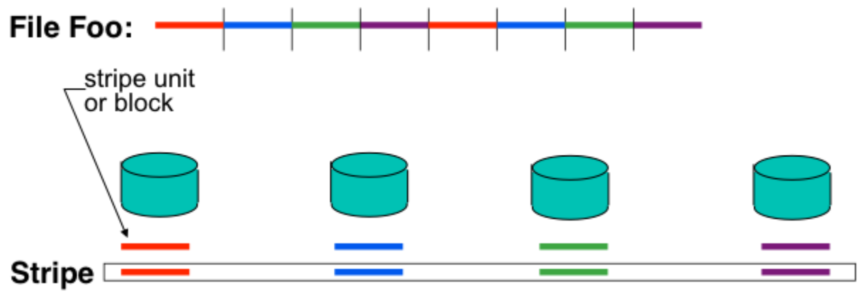
Just a bunch of disks (JBOD)



- Yes, it's a goofy name
  - industry really does sell "JBOD enclosures"

Disk Striping

- Interleave data across multiple disks
  - Large file streaming can enjoy parallel transfers
  - High throughput requests can enjoy thorough load balancing
    - If blocks of hot files equally likely on all disks (really?)



6

Now, What If A Disk Fails?

- In a JBOD (independent disk) system
  - one or more file systems lost
- In a striped system
  - a part of each file system lost

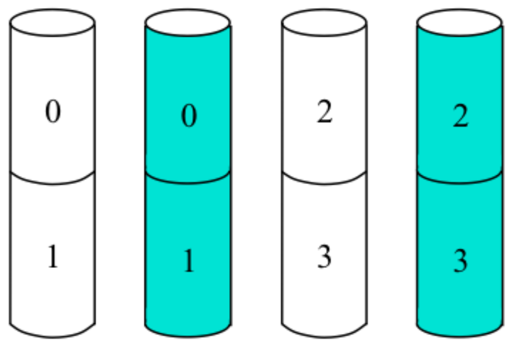
- Backups can help, but
  - backing up takes time and effort
  - backup doesn't help recover data lost during that day
    - Any data loss is a big deal to a bank or stock exchange

Tolerating and masking disk failures

- If a disk fails, it's data is gone
  - may be recoverable, but may not be
- To keep operating in face of failure
  - must have some kind of data redundancy
- Common forms of data redundancy
  - replication
  - erasure-correcting codes
  - error-correcting codes

Redundancy via replicas

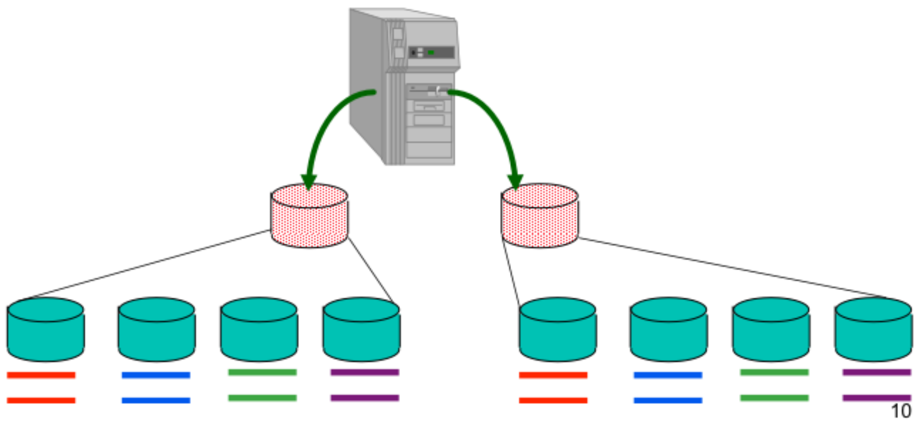
- Two (or more) copies
  - mirroring, shadowing, duplexing, etc.
- Write both, read either



9

Mirroring & Striping

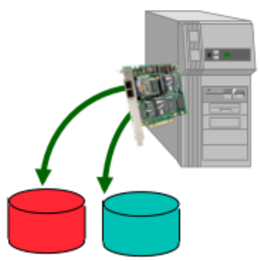
- Mirror to 2 virtual drives, where each virtual drive is really a set of striped drives
  - Provides reliability of mirroring
  - Provides striping for performance (with write update costs)

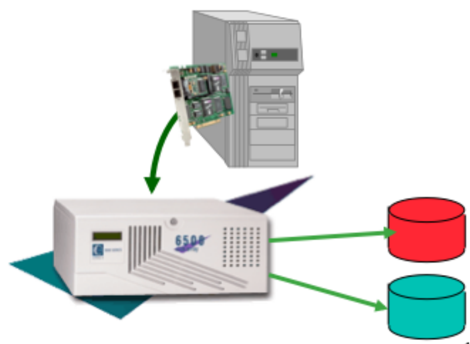


10

Implementing Disk Mirroring

- Mirroring can be done in either software or hardware
- Software solutions are available in most OS's
  - Windows2000, Linux, Solaris
- Hardware solutions
  - Could be done in Host Bus Adaptor(s)
  - Could be done in Disk Array Controller





Lower Cost Data Redundancy

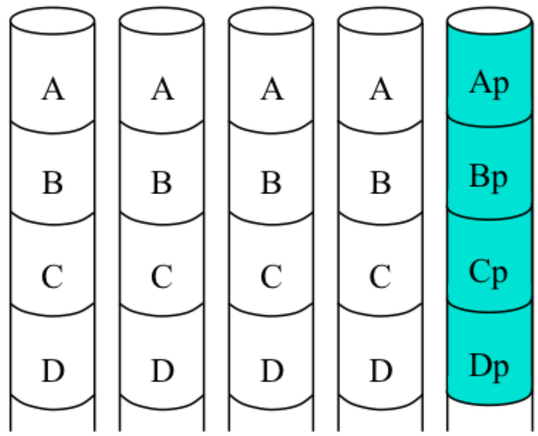
- Disk failures are self-identifying (a.k.a. erasures)
  - Don’t have to find the error

- Fact: N-error-detecting code is also N-erasure-correcting
  - Error-detecting codes can’t find an error, just know its there
  - But if you independently know where error is, allows repair

- Parity is single-disk-failure-correcting code
  - recall that parity is computed via XOR
  - it’s like the low bit of the sum

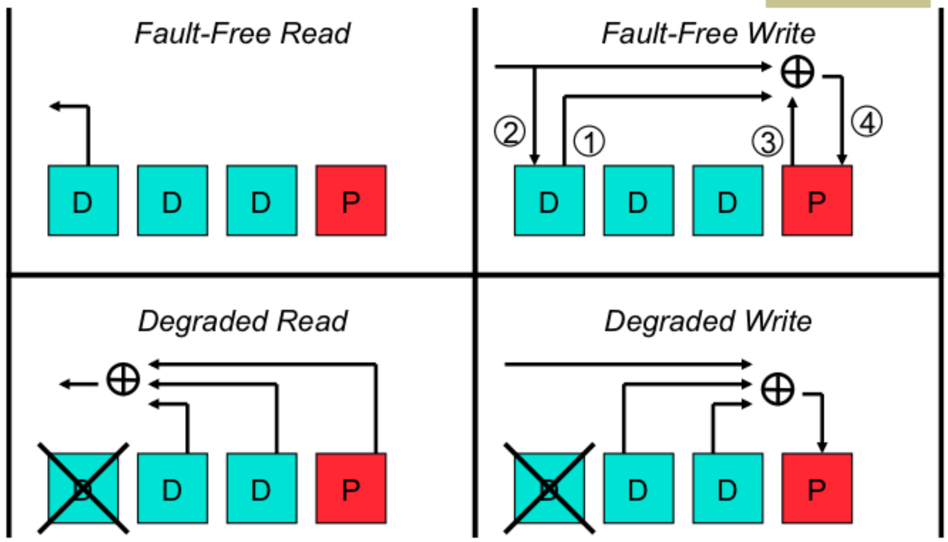
Simplest approach: Parity Disk

- Capacity: one extra disk needed per stripe



13

Updating and using the parity



14

Performance

- Suppose 1 drive gives bandwidth B
- Fault-Free Read = 3B
- Degraded Read = 1B
- Fault-Free Write = 0.5 B
  - But can do 2B Fault-Free Read at the same time
- Degraded Write = 1 B

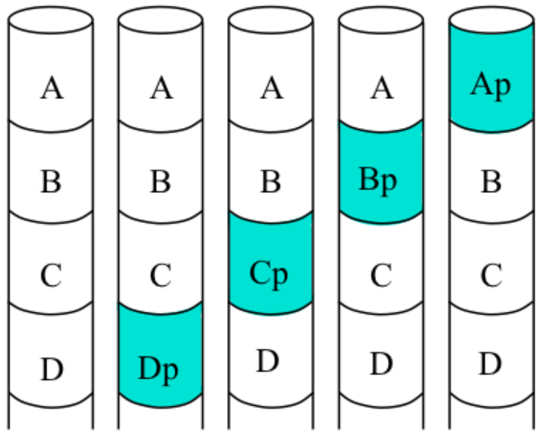
The parity disk bottleneck

- Reads go only to the data disks
  - But, hopefully load balanced across the disks
- All writes go to the parity disk
  - And, worse, usually result in Read-Modify-Write sequence
  - So, parity disk can easily be a bottleneck

16

Solution: Striping the Parity

- Removes parity disk bottleneck



17

Outline

- Using multiple disks
  - Why have multiple disks?
  - problem and approaches
- RAID levels and performance
- Estimating availability

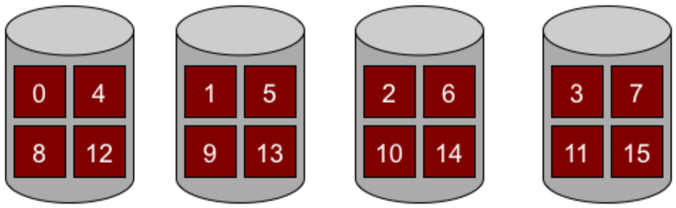
18

RAID Taxonomy

- Redundant Array of Inexpensive Independent Disks
  - Constructed by UC-Berkeley researchers in late 80s (Garth Gibson)
- RAID 0 – Coarse-grained Striping with no redundancy
- RAID 1 – Mirroring of independent disks
- RAID 2 – Fine-grained data striping plus Hamming code disks
  - Uses Hamming codes to detect and correct multiple errors
  - Originally implemented when drives didn't always detect errors
  - Not used in real systems
- RAID 3 – Fine-grained data striping plus parity disk
- RAID 4 – Coarse-grained data striping plus parity disk
- RAID 5 – Coarse-grained data striping plus striped parity
- RAID 6 – Extends RAID 5 by adding another parity block
- RAID 10 – RAID 1 (mirroring) + RAID 0 (striping)

RAID-0: Striping

- Stripe blocks across disks in a “chunk” size
  - How to pick a reasonable chunk size?



How to calculate where chunk # lives?

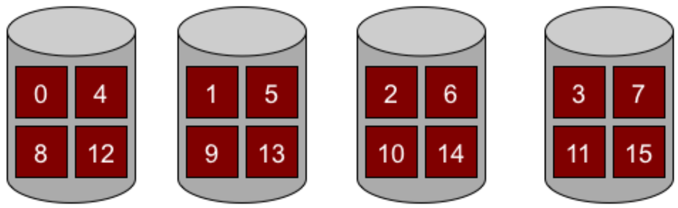
Disk:

Offset within disk:

Disk = chunk % 4

Offset = chunk DIV 4

RAID-0: Striping



- Evaluate for D disks
- Capacity: How much space is wasted?
- Performance: How much faster than 1 disk?
- Reliability: More or less reliable than 1 disk?

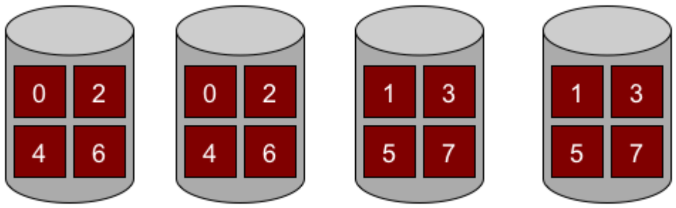
Capacity = N → 0% waste

Reliability = 0 → every failures causes data loss

Performance = large reads can use all disks → N times BW

RAID-1: Mirroring

- Motivation: Handle disk failures
- Put copy (mirror or replica) of each chunk on another disk



Capacity:
Reliability:
Performance:

Capacity = ½ total (worse with more replicas)

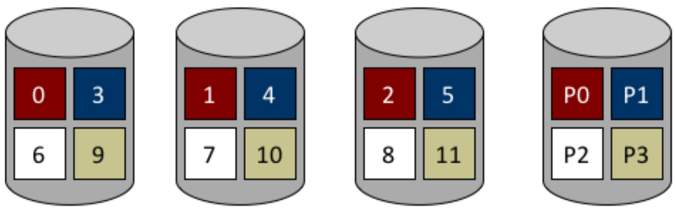
Read performance = 2x for hot items, aggregate can be N times single drive

Write performance = ½ drive for single item, ½ N for total array

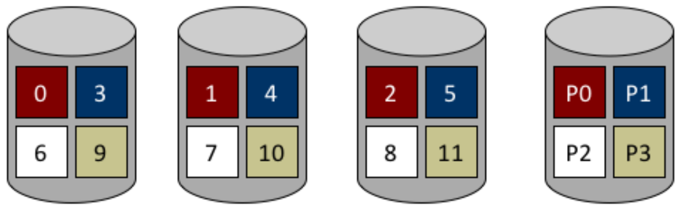
Reliability

RAID-4: Parity

- Motivation: Improve capacity
- Idea: Allocate parity block to encode info about blocks
  - Parity checks all other blocks in stripe across other disks
- Parity block = XOR over others (gives “even” parity)
  - Example: 0 1 0 → Parity value?
- How do you recover from a failed disk?
  - Example: x 0 0 and parity of 1
  - What is the failed value?



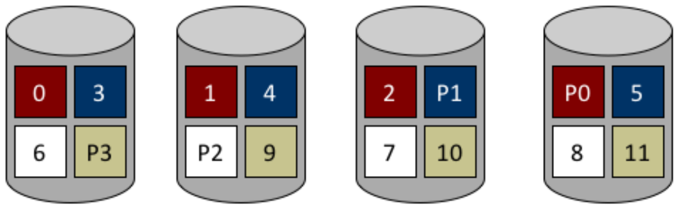
RAID-4: Parity



- Capacity:
- Reliability:
- Performance:
  - Reads
  - Writes: How to update parity block?
    - Two different approaches
      - Small number of disks (or large write):
      - Large number of disks (or small write):
    - Parity disk is the bottleneck

RAID-5: Rotated Parity

Rotate location of parity across all disks



- Capacity:
- Reliability:
- Performance:
  - Reads:
  - Writes:
  - Still requires 4 I/Os per write, but not always to same parity disk

Comparison

	RAID-0	RAID-1	RAID-4	RAID-5
Capacity	N	$N/2$	$N-1$	$N-1$
Reliability	0	1 (for sure) $\frac{N}{2}$ (if lucky)	1	1
Throughput				
Sequential Read	$N \cdot S$	N x R	$(N-1) \cdot S$	$(N-1) \cdot S$
Sequential Write	$N \cdot S$	$(N/2) \cdot S$	$(N-1) \cdot S$	$(N-1) \cdot S$
Random Read	$N \cdot R$	$N \cdot R$	$(N-1) \cdot R$	$N \cdot R$
Random Write	$N \cdot R$	$(N/2) \cdot R$	$\frac{1}{2} \cdot R$	$\frac{N}{4} R$
Latency				
Read	D	D	D	D
Write	D	D	$2D$	$2D$

Key takeaways: writes are expensive, small writes are really expensive! File systems may help (see LFS)

Outline

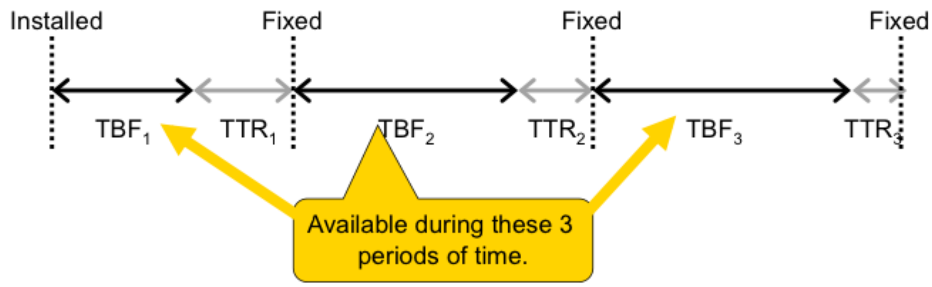
- Using multiple disks
  - Why have multiple disks?
  - problem and approaches
- RAID levels and performance
- Estimating availability

27

Sidebar: Availability metric

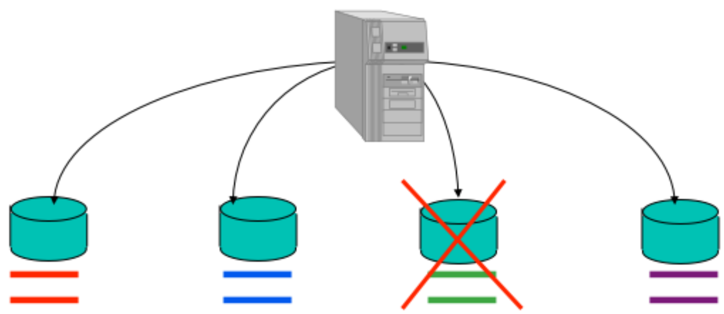
- Fraction of time that server is able to handle requests
  - Computed from MTBF and MTTR (Mean Time To Repair)

$$\text{Availability} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}$$

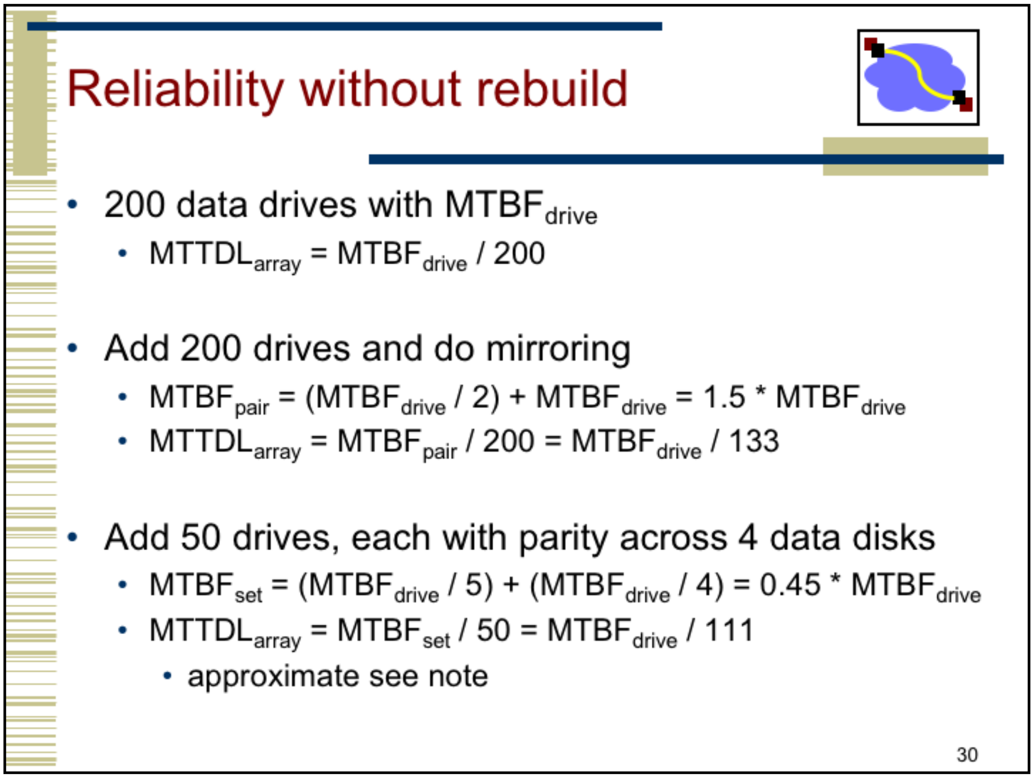


How often are failures?

- MTBF (Mean Time Between Failures)
  - $MTBF_{disk}$ ~ 1,200,00 hours (~136 years, <1% per year)
- $MTBF_{mutli\text{-}disk\ system}$ = mean time to first disk failure
  - which is $MTBF_{disk}$ / (number of disks)
  - For a striped array of 200 drives
  - $MTBF_{array}$ = 136 years / 200 drives = 0.65 years



29



The last step here is an approximation that an array is a single reliable "virtual drive". Note that this doesn't quite work in some cases.

The key issue (from the original RAID paper http://www.cs.cmu.edu/~garth/RAIDpaper/Patterson88.pdf):

"The second step is the reliability of the whole system, which is approximately (since the MTTF_group is not distributed exponentially) MTTF_group/num_groups"

The problem is that this estimate breaks down dramatically at large numbers. For example:

What is the mean time to data loss for a system with 100,000 disk, which are organized into 10,000 10-disk arrays, using data striping and striped parity (i.e., RAID 5)? Assume each disk has an MTBF of 100 years.

$MTBF_{array}$ =(100years/10)+(100years/9)=21years
MTTDL = $MTBF_{array}$/10,000 = 19hrs

But…. MTBF for a single drive in the entire system is 100yr/100000 drives approx = 8.9 hours.
MTBF for three drives = 3 x 8.9hrs = 26.7hrs which is more than $MTBF_{array}$

The system shouldn't lose data before three drives fail.

Why? Because MTBF_array / 10000 arrays actually calculates E[time first drive fails in any array] fails in any array | first drive fails in every array]. However what you really want is E[time first drive E[time second drive fails in any array | first drive failed in a single array], and the second terms are

Rebuild: restoring redundancy after failure

- After a drive failure
  - data is still available for access
  - but, a second failure is BAD
- So, should reconstruct the data onto a new drive
  - on-line spares are common features of high-end disk arrays
    - reduce time to start rebuild
  - must balance rebuild rate with foreground performance impact
    - a performance vs. reliability trade-offs
- How data is reconstructed
  - Mirroring: just read good copy
  - Parity: read all remaining drives (including parity) and compute

Reliability consequences of adding rebuild

- No data loss, if fast enough
  - That is, if first failure fixed before second one happens
- New math is...
  - $MTTDL_{array} = MTBF_{firstdrive}$ * (1 / prob of 2nd failure before repair)
  - ... which is $MTTR_{drive} / MTBF_{seconddrive}$
- For mirroring
  - $MTBF_{pair} = (MTBF_{drive} / 2) * (MTBF_{drive} / MTTR_{drive})$
- For 5-disk parity-protected arrays
  - $MTBF_{set} = (MTBF_{drive} / 5) * ((MTBF_{drive} / 4)/ MTTR_{drive})$

Three modes of operation

- Normal mode
  - everything working; maximum efficiency
- Degraded mode
  - some disk unavailable
  - must use degraded mode operations
- Rebuild mode
  - reconstructing lost disk's contents onto spare
  - degraded mode operations plus competition with rebuild

Mechanics of rebuild

- Background process
  - use degraded mode read to reconstruct data
  - then, write it to replacement disk
- Implementation issues
  - Interference with foreground activity and controlling rate
    - Rebuild is important for reliability
    - Foreground activity is important for performance
  - Using the rebuilt disk
    - For rebuilt part, reads can use replacement disk
    - Must balance performance benefit with rebuild interference

34

Conclusions

- RAID turns multiple disks into a larger, faster, more reliable disk
- RAID-0: Striping
  Good when performance and capacity really matter, but reliability doesn't
- RAID-1: Mirroring
  Good when reliability and write performance matter, but capacity (cost) doesn't
- RAID-5: Rotating Parity
  Good when capacity and cost matter or workload is read-mostly
  - Good compromise choice

Disk Subsystem Load Balancing

- I/O requests are almost never evenly distributed
  - Some data is requested more than other data
  - Depends on the apps, usage, time, ...
- What is the right data-to-disk assignment policy?
  - Common approach: Fixed data placement
    - Your data is on disk X, period!
    - For good reasons too: you bought it or you're paying more...
  - Fancy: Dynamic data placement
    - If some of your files are accessed a lot, the admin(or even system) may separate the "hot" files across multiple disks
      - In this scenario, entire files systems (or even files) are manually moved by the system admin to specific disks
  - Alternative: Disk striping
    - Stripe all of the data across all of the disks

36

Disk striping details

- How disk striping works
  - Break up total space into fixed-size stripe units
  - Distribute the stripe units among disks in round-robin
  - Compute location of block #B as follows
    - disk# = B%N (%=modulo,N = #ofdisks)
    - LBN# = B / N (computes the LBN on given disk)

37

Hardware vs. Software RAID

- Hardware RAID
  - Storage box you attach to computer
  - Same interface as single disk, but internally much more
    - Multiple disks
    - More complex controller
    - NVRAM (holding parity blocks)
- Software RAID
  - OS (device driver layer) treats multiple disks like a single disk
  - Software does all extra work
- Interface for both
  - Linear array of bytes, just like a single disk (but larger)

RAID 6

- P+Q Redundancy
  - Protects against multiple failures using Reed-Solomon codes
  - Uses 2 “parity” disks
    - P is parity
    - Q is a second code
    - It’s two equations with two unknowns, just make “biggerbits”
      - Group bits into “nibbles” and add different coefficients to each equation (two independent equations in two unknowns)
- Similar to parity striping
  - De-clusters both sets of parity across all drives
  - For small writes, requires 6 I/Os
    - Read old data, old parity1, old parity2
    - Write new data, new parity1, new parity2

The Disk Array Matrix

	Independent	Fine Striping	Course Striping
None	JBOD		RAID0
Replication	Mirroring RAID1		RAID0+1
Parity Disk		RAID3	RAID4
Striped Parity	Gray90		RAID5

Advanced Issues

- What happens if more than one fault?
  - Example: One disk fails plus “latent sector error” on another
  - RAID-5 cannot handle two faults
  - Solution: RAID-6 (e.g., RDP) Add multiple parity blocks
- Why is NVRAM useful?
  - Example: What if update 2, don’t update P0 before power failure (or crash), and then disk 1 fails?
  - NVRAM solution: Use to store blocks updated in same stripe
    - If power failure, can replay all writes in NVRAM
  - Software RAID solution: Perform parity scrub over entire disk