

Contents

About these reading notes	2
Chapter 38: Redundant arrays of inexpensive disks (RAID)	2
38.1 Interface and RAID internals	3
38.2 Fault model	3
38.3 How to evaluate a RAID	3
38.4 RAID level 0: striping	4
Chunk sizes	4
Back to RAID-0 Analysis	5
Evaluating RAID performance	5
Back to RAID-0 analysis, again	5
38.5 RAID level 1: mirroring	6
RAID-1 analysis	6
38.6 RAID level 4: saving space with parity	7
RAID-4 analysis	7
38.7 RAID Level 5: rotating parity	7
RAID-5 analysis	8
38.8 RAID comparison: a summary	8
38.9 Other interesting RAID issues	8

About these reading notes

These are my own personal reading notes that I took (me, Franklin) as I read the textbook. I'm providing these to you as an additional resource for you to use while you're reading chapters from the textbook. These notes do not stand alone on their own — you might be able to get the *idea* of a chapter while reading these, but you're definitely not going to **get** the chapter by reading these alone.

These notes are inconsistently all of the following:

- Me summarizing parts of the text.
- Me commenting on parts of the text.
- Me asking questions to myself about the text.
 - ... and *sometimes* answering those questions.

The way that I would expect you to read or use these notes is to effectively permit me to be your inner monologue while you're reading the textbook. As you're reading chapters and sections within chapters, you can take a look at what I've written here to get an idea of how I'm thinking about this content.

Chapter 38: Redundant arrays of inexpensive disks (RAID)

Back down from abstractions and filesystems, let's try to address "disks are too small/slow/fragile" by adding *more* disks to the situation.

RAID is a good approach, even given the facts:

1. We have commodity disks that are 20TB in size (cost is prohibitive, \$500 compared to 5×4TB around \$400, but density in the 20TB disk is higher than 5×4TB disks).
2. SSDs are **fast**, but *tiny* compared to rotational disks.

Simply: RAID makes a bunch of physical hard drives look like one single drive from the perspective of the OS.

RAID can be implemented in hardware or software. When it's implemented in hardware, the OS just sees a big bag of blocks, the same as it did when it was looking at an individual disk. Really importantly, when RAID is implemented in hardware, the OS has **no idea** that the data it's writing is going to multiple physical disks.

RAID gives us three desirable properties, but each with a trade off:

1. Performance
2. Capacity

3. Reliability

Note: This is not in or about the textbook, but I want to be clear: RAID is **not** a back up strategy. Not in any way, shape, or form. Yes, you get desirable properties that do look like backing stuff up (e.g., redundancy means that you're writing the same blocks to multiple physical disks) and making things more resilient, but RAID, in and of itself, is not a backup strategy. If you want an actual backup strategy, look at something like [the 3-2-1 backup strategy](#).

38.1 Interface and RAID internals

An OS sends logical I/O requests to a RAID controller, the RAID controller has to translate those to *physical* requests – it has to figure out what drive the requested data actually belongs to.

Aside: we don't actually see what a RAID physically looks like here, just a hand-wavy description of what it looks like.

Depending on how the RAID is configured (because there are multiple configurations), the RAID hardware may have to perform multiple physical writes for every logical write (again, the OS says “please write this block” and the RAID controller then has to decide how and where to place that block, across multiple physical disks).

The authors describe a bit of what RAID looks like as a controller, and they're basically describing a tiny computer (yo dawg).

38.2 Fault model

Let's just keep things simple for now. We will use a fault model called “fail-stop”, which is a binary “An entire drive works or doesn't work”. We don't care about disks where *some* of the blocks work and others don't, we don't care about data corruption.

38.3 How to evaluate a RAID

We're going to be looking at multiple types of RAID, and we need to be able to compare them to each other. We'll compare the different kinds of RAID on 3 metrics:

1. Capacity: given n blocks across m disks, how many total blocks do we get?
2. Reliability: how many disks can fail without affecting the data on the entire array? How many can fail and we can still get all the written data back?
3. Performance: how quickly can we read or write to the entire array of disks compared to just one?

38.4 RAID level 0: striping

The authors immediately note that striping is not **Redundant** at all, though it is an array (again: RAID is never a backup strategy).

The basic idea here is that we distribute all blocks across all disks in the array in a round robin fashion. We can trivially calculate which block belongs on which disk using modular arithmetic.

Aside: The authors have switched entirely to using the term “blocks” to represent the atomic concept on a drive. This is slightly different from the idea of a “sector” that we saw before, but they are effectively synonymous.

They describe the writing of blocks as “round-robin” – Hey! It’s that idea again!

The name striping comes from the idea that the blocks in a row on a table are referred to as a “stripe” (e.g., blocks 0, 1, 2, 3 in figure 38.1 are a stripe, blocks 4, 5, 6, 7 are a stripe, blocks 8, 9, 10, 11 are a stripe, etc).

We’ll see this in a second, but performance here is high: both reads and writes are distributed across multiple disks.

Aside: A couple of weeks ago you looked at I/O scheduling algorithms. Process scheduling and I/O scheduling algorithms are similar (they’re both scheduling things!), but I/O scheduling algorithms have a fundamental difference from process scheduling: I/O is explicitly **not** concurrent (a hard drive can’t write two things “at the same time”). However, now that we introduce this idea of RAID, I/O *is* concurrent. Do scheduling algorithms have to change? Does the OS *care*? Does the RAID controller have the responsibility of having a more complex I/O scheduler?

Chunk sizes

The chunk size represents how many sequential blocks in a stripe would be placed on the same disk.

The chunk size will directly influence across how many drives a single file will have its blocks scattered. A bigger chunk size might mean that a file in its entirety will be put onto the same drive. A smaller chunk size might mean that the file will be placed across many drives.

We’re trying to balance the sequential read/write performance of a single drive across coalescing bits from many drives.

The actual answer to this is that you’d have to run experiments on chunk size for the workload that you’re trying to evaluate. If you have lots of really big files that you want to improve random access for, you’re going to want to use a different chunk size than if you wanted to improve sequential access for.

Aside: Think about this later: is chunk size strictly a RAID-0 thing? Can RAID-1's performance be affected by chunk size?

Back to RAID-0 Analysis

Capacity of RAID-0 – it's as perfect as you can get: n disks with b blocks gives us $n \times b$ capacity as a single volume.

Reliability is perfect—ly terrible! If a disk fails, all the data that was on that disk is lost without recovery options. As far as the filesystem is concerned, that means that many files are corrupted, and (possibly) the whole filesystem is lost if the superblock and other information regions are on the disk that's been lost.

Performance is great! We can distribute reads and writes across all disks however we please.

Evaluating RAID performance

Now we're getting into a more granular method of assessing what "good" means on a RAID array. Basically, we've got two metrics:

1. single-request latency: how long it takes to service a single I/O request (**Aside:** is this for **one block** or **one file**? Almost certainly one block.)
2. steady-state throughput: how many bytes can maximally be moving through this RAID controller at any given time?

We measure these two metrics using different *workloads*:

1. Sequential: We're going to request n blocks where the address that the OS sees is in sequence (not necessarily the same as the physical location)
2. Random: We're going to request n blocks where the address is totally random.

We're now going back to our disk chapter and reminding ourselves that disks have a transfer rate. The transfer rate for sequential workloads is higher than random workloads.

Aside: *Why* is that true? Can you convince yourself of that using your knowledge of the physical layout of a disk and how blocks are allocated that sequential reads are faster on a rotational disk?

Back to RAID-0 analysis, again

Yeesh, with the asides and back pointers.

I guess it makes sense, we want a variable to refer to random throughput vs sequential throughput.

Performance here is again, as fast as the disks in the array are. For a single request, we just forward the request to the disk, so this is going to be as fast as any disk is. For steady-state throughput with random access or sequential workloads, performance is optimal because with sequential (assuming the data is striped evenly across the array), we can have the sequential transfer rate of one disk times all disks. The same is true for random access.

38.5 RAID level 1: mirroring

Now instead of distributing data evenly across disks, we're *duplicating* data across disks.

We're trying to address the reliability problem with RAID-0. Now we can say that 1 disk failing is OK, we're not going to lose any data.

Aside: The authors make the distinction between RAID-10 and RAID-01. RAID-10 is we pair up disks and do mirrors, then stripe across the *pairs*. Basically, treat each pair of disks as a single disk and then stripe across the single disks. RAID-01 is a stripe, then a mirror? Are we duplicating data on a *single* drive? No, we're making two stripes then mirroring them. That is, disk 0 and 1 are a stripe, and 2 and 3 are a mirror of that stripe.

Read and write performance are *different* on RAID-1. Reading can be done from *any* of the disks with the mirrored data, but writing must be done to **all** of the disks in the mirror. Writes can be done in parallel (provided the disk isn't busy?).

RAID-1 analysis

1. Capacity: RAID-1 basically halves data (or, more accurately n ths data, where n is the number of mirrors that we have).
2. Reliability: In theory, if mirroring level is 2, then half of the array could fail, provided that the disks failing are only one of each pair.
3. Performance: Reads are good, but not as good as RAID-0. Single requests are still as fast as a single disk is for read, but a *tiny* bit slower for writes because we have to account for all of the disks seek times for writing to the mirror. Throughput is different, though. Sequential read throughput is lower because the worst case scenario is that the RAID controller does a bad job of distributing reads to the disks. Random access throughput has the highest performance for RAID-1 because we can (again) distribute reads across all disks, even for mirrored blocks.

38.6 RAID level 4: saving space with parity

Giving up an entire disk for mirroring data is not an ideal solution. Let's use some maths to help us out with data integrity.

In fact, let's use really simple maths and a simple idea: xor all the bits in a stripe. The xor tells us "Is there an even or an odd number of bits in this stripe?". If *one* disk fails, we can *really* easily determine what that disk's bit was. [NEAT](#).

Even more neater: we can use xor on the remaining bits and the parity bit to tell us what the lost bit is! ☒

RAID-4 analysis

1. Capacity: We lose **1** disk, not half of the disks. So that's better.
2. Reliability: We can lose *any* one disk. But, uh, only one. Lose two and we've lost data. This is definitely worse than mirroring where we could (in theory) lose multiple disks, provided they were in different pairs.
3. Performance: We can do pretty good with read, since we can distribute the sequential steady state reads across $(N - 1)$ disks. Same with random reads. The issues that come up here are with *writing*, since for each write we need to calculate or recalculate the parity bit. "Full-stripe writes" offer similar performance to a RAID-0 array, where each disk in the array gets one write, and is done in parallel. Random writes are the tricky case: if we're just writing *one* block, then we need to *read* the other blocks in the stripe to calculate the parity bit. Once we've calculated the parity bit we do two writes in parallel, the target block and the new parity block. This is called "additive parity".

When doing a single block write, we can also read the old value of the target block and the current parity bit, then xor all three of them together to get the new parity bit.

Calculating the parity bits causes a bottleneck on I/O to the parity bit drive.

Aside: Is it OK to lose the parity disk? **Aside:** Are parallel writes atomic?

38.7 RAID Level 5: rotating parity

Putting all the parity bits on one disk leads to that disk being a bottleneck for the whole system. The idea here is that we put parity bits on *all* the disks, where each one takes a turn for each stripe.

RAID-5 analysis

1. Capacity: is the same as RAID-4, but the bits that would have been put on 1 drive are just across all.
2. Reliability: Again, we can lose any disk and not lose any data, but we can only lose one.
3. Performance: This is where things get different. Sequential read and write for throughput are the same as RAID-4. Random reads are a bit better (+1) because we can randomly read from all drives instead of having to avoid the parity disk. Random write also improves because the problem we had with RAID-4 and the parity drive being a bottleneck is gone, all disks share the load for writing the parity bit.

The summary here is: RAID-5 is RAID-4 + 1, with no losses, so RAID-5 usually just wins.

38.8 RAID comparison: a summary

The summary the authors produce omit the constant values (like seek time on drives) and otherwise provides a handy summary with the number of drives. Figure 38.8 does a real nice job of summarizing the properties of each RAID level.

Basically: Here's what you should pick when you want to favour different things.

38.9 Other interesting RAID issues

The authors write that there are RAID levels 2 and 3 in "the original taxonomy", plus RAID-6 that permits multiple disk failures (how?). They don't describe what the "original taxonomy" is, but thankfully [Wikipedia's page on RAID](#) bails us out and lists all the RAID levels, including 2 and 3.

They go on to say that there are other types of failure modes, and what might happen to the disk array when it's *rebuilding* an array.

Also: software RAID. Yeah, that's also a thing. Is software RAID an OS responsibility? (yep)