**ECE 350: Real-Time Operating Systems** Spring 2023

# Lecture 24 — Reliability: RAID

*Jeff Zarnett* *2023-03-04*

## Failure is Always an Option

The previous topic discussed the idea of using transactions in case there is a failure in the system to ensure that data is always in a consistent state, but we closed on somewhat of a darker note that said, well, hard drives die. And they do. So what do we do? Backups are a part of the solution but they're not the only way to ensure that we don't lose data.

Backups are not necessarily always up-to-date; if you have daily backups you could lose the full day's work; hourly backups might lose an hour, etc. And more than that, if the disk dies, you will need to replace it (maybe that's not instant) and then it takes nontrivial time to copy the data from the backup storage location to the new disks and get the system going again. Finally, there's a saying that you really only have backups if you've tested that you can restore from those backups successfully.

So, as it stands, the untimely death of a hard drive means potentially the loss of some data (decreased reliability), and until the situation is addressed, the system is down, or at least at reduced performance (decreased availability). What if I told you there exists a solution to this where you not only get higher reliability and availability, but also maybe better performance? That's not just wishful thinking, it's possible... but it does cost money. Maybe not as much as we might think, though.

To be clear, though, the solution I am describing is not magic: if all your hard drives die at once then there will still be data loss. Similarly, if your data centre catches on fire and burns to the ground, it will decrease availability. But for common scenarios, we can do a lot to reduce the likelihood and impact of a problem.

### RAID

The solution is called RAID: Redundant Array of Independent Disks (previously, the I stood for "Inexpensive"). The idea, in short, is to have multiple different independent disks that work together but appear to the user as if they are one drive/volume. The RAID array may be managed in software or in hardware, but it does not matter for our discussion of the topic today.

The idea behind the former use of the word "inexpensive" is that if we combine more than one relatively-inexpensive disk, we can get higher reliability than we would if we spent more money on higher-quality drives. But drives have become a lot less expensive in general so the independent part has taken over the letter I.

How much redundancy we need is highly dependent on the data that we need to store. As you may know, I lost a hard drive in about 2014, but I wasn't upset about it because it mostly contained my Steam game library and it was easy to just download and reinstall all the games again (even if I still never played most of them!). Not having redundancy there is okay because, to some extent, Valve corporation is the source of my backups. Losing this data is inconvenient, slightly, but not a tragedy. But suppose I have a database backing up my web application where I sell services – downtime here is going to be a problem.

For any device (hard drive, SSD, or even CPU) we can talk about the *mean time to failure* of the device. The mean time to failure is always an estimate: if this is the first time that ABC corporation has produced this particular device, they don't really know how long it will last. They may be able to compare to the previous year's version and use some internal test results to create their estimate, but it's hard to know if the device will last ten years before ten years have elapsed. Also, the estimates assume some usage patterns that may not be how you actually use it. Graphics card manufacturers may assume that you want to use the RTX 3080 to play Cyberpunk 2077 and

base the lifetime estimates on you playing this game $x$ hours per day on average... but if you actually overclock it and use it to mine cryptocurrency (please don't), then its lifetime will be shorter than expected because using it at 110% 24 hours a day will cause a failure sooner than predicted.

How bad is the problem? Let's say you bought a disk that says its mean time to failure is 100 000 hours (11.4 years) – you might think that this is perfectly fine, even if it lives only half its expected life it's still more than five years and you'll replace it by then – but if you are running a data centre and you have 100 of these, the mean time to failure of one of these disks is really only 100 000 / 100 = 1 000 hours or 41.66 days [SGG13]. The textbook example is a little bit silly – hard drive deaths are not evenly distributed but the example makes it seem like one of them dies every six weeks. It's more likely that everything is fine for a while and then drives start dying in quick succession.

**No Free Lunches.** So if we want to have improved reliability, it's obvious that this is going to have a cost. The cost is in the redundancy; having more than one copy of the data and those copies are stored on different physical storage media. The cost can be thought of in two ways: either we have to spend more money for more disks, or, if we have a fixed budget available, we are trading some of the total capacity for redundancy. Both scenarios are easy to visualize: if we want to have redundancy for a 2 TB hard drive then you could buy another 2 TB hard drive for redundancy, or use the same money to have two 1 TB hard drives. Either way, we have less space available than the nominal sum of the drives purchased, but that's inevitable.

Given that we have two drives of the same size, well then, the obvious approach is what is called *mirroring*, which is to say that every write that takes place is duplicated to both disks. If one of the disks dies, then the data is on the other disk and it's not lost. Excellent! Of course, having lost one of the disks, it's important to replace the dead one so that the reliability is restored. How quickly?

Quicker is obviously better, but if you want a quick calculation: if the mean time to failure is 100 000 hours and the mean time to repair is 10 hours, the mean time to data loss is $(100\ 000)^2/(2 \times 10) = 500 \times 10^6$ hours (about 57 000 years) [SGG13]. That's certainly a long time, but is it realistic? What we're saying effectively, is, how long will it take before disk 1 fails but cannot be replaced in time before the failure of disk 2. Assuming we attend to the problem quickly, you can imagine this is rare.

... Or is it? What we're assuming here is that disk failures are independent. We are not expecting that disks will actually survive 57 000 years – let's not kid ourselves – but only that hypothetically, if a disk dies and we replace it fast enough, and maintain this watch forever, then we expect the average time to lose data is 57 000 years. But that's not really how likely it is to lose data, because in reality these things may not be independent at all. Leaving aside things like the data centre catching on fire, if we buy two disks from the same manufacturer from the same manufacturing run, it's entirely possible that both of these disks will live about the same amount of time. So it makes it more likely their demises are closer together and not independently randomly distributed. And if we do throw in a new disk to the system and have to copy all the data from the old disk, that is in fact a heavy load on the surviving disk and if it's already fragile, that might be the last thing that pushes the old disk over the edge. Oops.

The simple version of mirroring we've considered does have additional costs and questions: every write needs to be done twice and what happens if one such write succeeds and one fails? It's a good question, but let's save that for the moment, because there are more possible arrangements than just mirroring.

## Level, Please

RAID is describes as having different "levels"; each level has a different configuration of the drives. Some of them have more redundancy at the cost of space (or money), others have less. There's also potentially some performance benefits to choosing one configuration or another. The different levels should not be viewed as higher equals better; it's important to choose the right level based on your needs and budget.

There are seven basic levels that have broad acceptance in industry (RAID 0 through 6) but there are also some combinations of these that are relevant [Sta18]. Some researchers or companies may suggest different levels or

numbers, but we'll restrict the discussion here to the commonly-accepted ones that we find in textbooks and other discussions. Even amongst the levels that exist, not all of them are actually commonly used.

**No Thanks.** It wouldn't be right to exclude the do-nothing option here, which is to say, no RAID configuration at all. This is sometimes called JBOD – Just a Bunch Of Disks – and it's simply choosing not to configure the disks you have in a RAID array and simply treat them as individual disks. No redundancy and no reliability increase, but it's entirely manual how you want to handle this.

This isn't entirely insane: you could say that using, say, an external hard drive for backups is like this. Or if you have two internal SSDs, copying important files from one to another to guard against data loss. But okay, let's get on with the actual RAID levels, shall we?

**RAID 0: What Redundancy?** The first option to consider is RAID 0: disk striping. This just splits the data and files across multiple disks, so Disk 1 will have block 0, Disk 2 block 2, Disk 3 block 3...

The major advantage of the RAID 0 configuration is that it is faster! Writes can be done in parallel: e.g., writing a file that has many blocks to a RAID 0 array of $n$ disks can be done at a speed of $n$ blocks at a time, which increases the parallelism by a factor of $n$. The same is true for reading, it can also be sped up by a factor of $n$. So throughput for reads and writes goes way up.

There is no redundancy at all in this system. It is faster to do reads and writes, but at the cost of reliability: if one drive dies, all the data is lost! That might seem weird, because topic we're considering here is reliability; this sounds like the opposite of what we want.

There are scenarios where you would be fine with the risks of RAID 0. Suppose you have a transit app that gives you the bus, subway, and train schedules. Instead of retrieving the schedules from the database on every request, you could keep them in a cache. If memory is big enough, great, keep them in memory. But if it's not and you need them to be on disk, you might want to make a RAID 0 array of two smaller disks so that you get better speed for reading and writing things in this cache. In this scenario, we can live with the fact that any one disk's untimely death takes the whole thing down, because the cache is just a more-easily accessible copy of the original data and it's trivial to regenerate that at any time.

Another reason we might want to use a RAID 0 configuration is because we want to combine it with another option. We haven't discussed those yet, so we can return to this idea of combining things later.

**RAID 1: Mirror, Mirror.** RAID 1 is mirroring, rather like we discussed in the introduction to the section. The data that appears on each disk is replicated exactly to the other so that they are identical. That is a lot easier to explain than the upcoming RAID levels that involve parity schemes. This works with an even number of disks, as half of the disks are mirrors of the other half.

Read speed is improved, since any piece of data could be found on one of two disks, so we can get the data from the less-busy drive. If it's a larger request, we can read from two disks in parallel and get the data faster. If we have many more reads than writes in our application, this is potentially a big performance improvement alongside more reliability.

In terms of write speed, it's realistically no worse than just writing directly to one disk; there's no write penalty. If one of the disks is busier than the other, the write is not truly complete until it has been written to both. This might require waiting, which would be slower than just writing to one disk. The alternative approach is to write it to one disk and asynchronously carry out the replication to the other.

Replication is fine, but it comes with some additional headache of what happens if that replication fails. Or, alternatively, if two different in-progress transactions failed We have tools to deal with that scenario: transactions, recovery, or even NVRAM (non-volatile RAM) to ensure that the queued operations proceed even in the event of a power failure [SGG13].

Should one the disks be unavailable, we can just get whatever data we need from the disk that survives and the read speed may be reduced but is no worse than when there was only one drive in the first place.

If we have two disks, it makes sense that our usable capacity is half of the purchased capacity, because each disk is an exact copy of the other. But if we have four disks, two disks are a copy of the other two. Eight disks means four and four. That's excellent for reliability, but if the scenario we want to guard against is the failure of one drive, why are we using four backups? Better safe than sorry, but at some point it is overkill. What if we didn't need quite so many disks to guard against the failure of one of them?

**RAID 2: Bit Parity** RAID 2 looks a lot like how memory does error correction in ECC RAM; memory uses parity bits to detect if there is an error. Remember how parity bits work from previous courses? Does Hamming code sound familiar?

If you just want to detect single-bit errors, each byte has a parity bit that is 0 if there's an even number of 1s in the byte and the parity bit is 1 if there's an odd number of bits. In the event of data corruption of some sort, we can detect the error but it's hard to know which bit is flipped. If we have a two or more bit scheme, we can correct single-bit errors and detect two-bit errors. And the trend continues: more parity bits equals more ability to detect and possibly correct them. More overhead, but also more ability to correct data.

ECC can be used in disk arrays. One such example is that if you have the first bit of each byte stored on disk 1, the second bit on disk 2, and so on such that the 8th bit of each byte is stored on disk 8 and you have yet another disk that stores the parity bits; in the event of a loss of one of the disks, for each byte you have seven of the eight bits and you can work out from the parity bit whether the missing bit should be a zero or one [SGG13]. You don't actually need nine drives, though it does help make the organization a little clearer.

Based on this scheme, a read or a write will necessarily engage all of the disks in the array so it can read data quickly and write data quickly, but it's also painful to split every read or write up to the bit level. It might work better for larger files since we could read multiple blocks at once; for small files it requires reading eight blocks where one would normally do. So random access performance is not great.

This does have slightly lower overhead than RAID 1 as you don't have 50% of the disks given over to replication, but it's still pretty high overhead. If you have four disks of original data, you might need three parity disks for it. It's hard to come up with a use case for RAID 2: it makes sense where lots of disk errors occur and data elements are frequently corrupted, but exiting hard drives are reasonably reliable so there's not much benefit to using this scheme [Sta18].

**RAID 3: Byte Parity** RAID 3 looks a lot like RAID 2, but instead of having each bit of a byte spread across different disks, bytes are spread across the disks and parity is slightly different: a parity bit is computer for the set of bits at the same position on all the data disks.

An example from [Sta18] about how to calculate the parity: if we have drives called D0 through D4 and D4 is the parity disk, the parity for bit $i$ is calculated as $D4(i) = D3(i) \oplus D2(i) \oplus D1(i) \oplus D0(i)$ (where $\oplus$ is the XOR operator). If drive D1 has died, then you can recover the data by calculating $D1(i) = D4(i) \oplus D3(i) \oplus D2(i) \oplus D0(i)$.

Like RAID 2, every read or write needs to be performed synchronously across all disks. That can help with speed since the data is being access across multiple devices simultaneously, but we might end up doing a lot of extra reads to read a small file.

One advantage is that only one extra disk is required no matter the number of drives in the system and it stores all the parity information. Less overhead, and a lot less – one disk for redundancy out of eight rather than four. Bit there is the computational overhead of doing the parity math, although this is relatively small in comparison to the times for disk reads and writes. And maybe can be offloaded to the hardware.

Another thing that's nice is that if a disk fails, all data is still available if somewhat slower. Any missing data can be compensated for by calculation of what it should be using the parity information. Writes need some additional

bookkeeping so that when the system is restored to full functionality, everything is where it should be. However, this is nice to have since it means the system can continue executing in a reduced capacity until the problem is solved, rather than going down [Sta18].

**RAID 4: Block Parity** RAID 4 looks a lot like what we've seen already, but instead of having bit- or byte-level parity we have block level parity. There is block-level striping as in RAID 0 and the parity blocks are stored on a dedicated parity disk. As with other schemes using parity, data that is lost due to the failure of a disk can be restored by doing the calculations using the data on the other disks.

Unlike RAID 2 or 3, each disk can independently be accessing a different block at the same time.

There is some penalty to doing an update, if the update is small. If a whole new block is being written, writing the parity block for it is straightforward. If, however, the write is small, the existing information needs to be read and modified, so every write may result in reading the original block and parity information, making the modifications, writing it and the parity data out for a total of two reads and two writes [Sta18].

One potential downside to this scheme is that the disk containing the parity information is modified a lot. That means this drive may be worn out sooner than the others in the system (which is, admittedly, more of a problem for SSDs that can get their writes exhausted). But this could become a bottleneck, also [Sta18].

**RAID 5: Distributed Block parity** RAID 5 looks like RAID 4, except instead of one disk dedicated solely to the parity information, the parity information is distributed across all disks. That is, each disk has some data, and some parity information of other disks.

Parity information could be stored in a round-robin fashion. Obviously, a parity block cannot be stored on the same disk where the original information is, because a loss of this disk would cause an unrecoverable loss of data.

RAID 5 is, in my experience, and according to the textbook [SGG13], the most common parity RAID system. It provides a lot of what we've been looking for – reliability to avoid data loss if any one drive should die, without doubling the number of drives needed.

**RAID 6** RAID 6 just looks like RAID 5 but has extra information that allows the system to survive more disk failures. Rather than parity, some other error correcting codes are used such that we have, say, two bits of redundancy information for every four bits of data; in this case the system could survive two disk failures. But there is a substantial write penalty here since every write requires modification of two parity blocks; tests show that RAID 6 has read performance about the same as RAID 5, but writes are about 30% slower [Sta18].

## Combining Levels

There are a number of ways that RAID may be combined, such as RAID 01, 10, 50... These are best thought of as being something like 1 + 0 rather than ten, because what we're doing is combining both these things at two different levels.

Consider RAID 0 + 1: we have two sets of disks combined in RAID 0, and then on top of that we build in mirroring so we have a RAID 1 of the combined disks. So if we have four disks in total, we have a RAID 0 formed of disks 1 and 2, another RAID 0 formed of disks 3 and 4, and then build a RAID 1 array of the 1+2 and 3+4 disks.

In principle you can combine any strategies any number of times: every time a group of disks is combined using a particular RAID level, you can treat that as a single disk input to the next level.

### Choosing the Right Level

How do we choose the right RAID level for our system? It comes down to some important factors that we've already discussed:

- How critical is it that data is not lost?
- What is the budget?
- How important is performance?
- Do we need to be able to carry on in the event a disk dies?
- Are rebuild times important?

There are really no one-size-fits-all answers, though certainly we can see that some levels of RAID (2, 3, 4, and to a lesser extent, 0) are not very popular for obvious reasons. Every use case is different, so choose what makes sense and what you can afford.

## References


[SGG13] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts (9th Edition)*. John Wiley & Sons, 2013.

[Sta18] William Stallings. *Operating Systems Internals and Design Principles (9th Edition)*. Pearson, 2018.