RAID

RAID stands for Redundant Array of Inexpensive Disks, with “Inexpensive” sometimes replaced with “Independent”. It’s about taking several possibly slow and unreliable disks and putting them together to get something that is faster and more reliable than any individual disk, and has more total capacity. Often the RAID array will have a hardware controller that handles the details of which disks to write to when. This can be done by software or by the OS itself, but using a controller is much faster.

There are several different RAID levels, which are designed for performance, reliability, or a combination of both. We will cover the important ones below. For each of the levels below, we will consider how a file will be spread across the individual disks of the RAID array. The file itself is broken into *blocks*, which are equal sized pieces of the file, usually around 1 to 4 kilobytes each, depending on the system.

For each RAID level, we will be interested in three things: (1) redundancy (how it copes with the failure of a disk), (2) capacity (how much storage space we get), and (3) I/O speed (how fast reading and writing to the array is).

Level 0 (Striping)

Suppose we have a file that consists of 16 blocks, numbered 1 to 16, and suppose we have 4 disks in our array. Below is a table showing how the blocks are spread across the disks. Each of the 4 columns is a different disk.

1	2	3	4
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

The way to read this table is that Blocks 1, 5, 9 and 13 are stored on Disk #1, Blocks 2, 6, 10, and 14 are stored on Disk #2, etc. If the blocks are 4000 bytes, then Block 1 holds bytes 0 to 3999 of the file, Block 2 holds bytes 4000 to 7999, etc. The main idea of striping is that the files are spread evenly across all of the disks. Here is a breakdown of how well it does in terms of redundancy, capacity, and speed.

- Redundancy — None. If any disk fails, then we lose part of the file.
- Capacity — Best possible. In the example above, if the disks are all the same size, then the capacity is 4 times the capacity of a single disk.
- Speed — Best possible. We can write to the disks in parallel. For instance, in the example above, while we are writing Block 5 to Disk #1, we can be writing Block 6 to Disk #2, Block 7 to Disk #3 and Block 8 to Disk #4. This is 4 times faster than writing all the blocks to a single disk.

Level 0 is all about performance instead of reliability. It’s what you use if you want maximum space and speed and don’t care if you lose data. Each disk you add increases the capacity and the speed, but it also increases the chances that one of the drives will fail. For instance, if you have 100 drives, each with a 1% chance of failure in the next month, there’s a pretty good chance that at least one of them will fail (around 63%).

Level 1 (Mirroring)

Suppose we have 3 disks and a 4-block file. Here is the table showing how the blocks are spread across the disks.

1	2	3
1	1	1
2	2	2
3	3	3
4	4	4

The name “mirroring” comes from the fact that each disk is a mirror image of the others. Usually Level 1 is done with just two disks, but in theory there could be more. Here is a breakdown of how well it does.

- Redundancy — Best possible. Each disk is a copy of the others. We won’t lose data unless every disk fails.
- Capacity — No gain. The capacity is the same as the capacity of a single disk.
- Speed — No gain. The writes all happen in parallel, so this is not any slower than a single disk, but it’s also no faster.

Level 1 is what you use if all you care about is reliability and not performance.

Level 10 or 1+0

Level 10 (or Level 1+0) is a combination of Level 0 and Level 1, which is where it gets its name. The file is striped across some of the disks, and some of the disks are mirrors of others. Suppose we have 4 disks and a file of 6 blocks. Below is the table showing how the blocks are spread across the disks.

1	2	3	4
1	1	2	2
3	3	4	4
5	5	6	6

If we have 6 disks, there are a couple of different ways we could combine mirroring and striping. Here are examples with an 8-block file.

1	2	3	4	5	6
1	1	2	2	3	3
3	3	4	4	5	5
1	1	2	2	8	8

1	2	3	4	5	6
1	1	1	2	2	2
3	3	3	4	4	4
5	5	5	6	6	6
7	7	7	8	8	8

Here is a breakdown of how well Level 10 performs.

- Redundancy — In four-disk example, we can recover from losing one disk, and possibly two, depending on which disks fail.
- Capacity — In the four-disk example, we get double the capacity of a single disk.
- Speed — Since the writes can happen in parallel, in four-disk example, we get a two times speedup.

Level 10 is a nice combination of performance and reliability. It is used for databases and web servers.

Level 4

Levels 2 and 3 are rarely used, so we will skip them. Level 4 is also rarely used, but its concepts are important for Levels 5 and 6, so we will cover it. Below is a table for a file with 16 blocks, where we use 5 disks.

1	2	3	4	5
1	2	3	4	P_1
5	6	7	8	P_2
9	10	11	12	P_3
13	14	15	16	P_4

The blocks P_1, P_2, etc. are called *parity blocks*. They are used to help recover data. Here is how parity works: Suppose first, for simplicity, that the blocks are only a single bit in size, like in the example below. In each row, the parity bit is will be 0 if there are an even number of ones in the row and 1 if there are an odd number of 1s.

1	2	3	4	5
1	0	1	1	$\underline{1}$
0	0	0	1	$\underline{1}$
1	0	0	1	$\underline{0}$
1	1	1	1	$\underline{0}$

For instance, in the first row above, the blocks are 1 0 1 1, which contains an odd number of ones, so the parity bit is 1. There are also an odd number of 1s in the second row, so the result is 1. In the last two rows there are an even number of 1s, so the parity is 0 for both. For humans, it is easiest to compute the parity by counting ones, but for a computer, it is easier and faster to compute it by XOR-ing all the bits together.

Here is how that parity is used to recover data: Consider the first row from the table above, and suppose that Disk #3 went down and we need to recover its contents, as shown below.

1	2	3	4	5
1	0	?	1	$\underline{1}$

Among Disks #1, #2, and #4, there are two 1s. The parity bit is 1, which means there should be an odd number of 1s in total from the four disks. So the missing bit from Disk #3 must be a 1. A computer would compute this by XOR-ing the three remaining bits along with the parity bit. In a real situation, files are longer than one bit in size. To handle that, you do the parity operation separately on each bit of the block. For example, here an example with 4-bit blocks.

1	2	3	4	5
1011	0111	1010	1000	$\underline{1110}$

The first bits of the four blocks are 1, 0, 1, 1. There are an odd number of 1s there, so the first parity bit is 1. This same process is done for all of the bits.

Level 5 and 6

As mentioned, Level 4 is not much used. The reason is that the parity blocks are all stored on the same disk, which means when writing, you lose parallelism. For instance, referring back to the first table in the Level 4 section, suppose you needed to write just blocks 1, 6, 11, and 16. Those blocks are all on different disks, so you could do all of those in parallel, meaning that the total amount of time is no slower than writing to a single disk. However, each of those writes affects the parity, and we would have to write that parity to Disk #5 four times, one after another, meaning the overall write time is no different than writing to a single disk four times, one after the other. Levels 5 and 6 fix this by spreading the parity around. Below is what Level 5 looks like with a 16-block file and 5 disks.

1	2	3	4	5
1	2	3	4	P_1
5	6	7	P_2	8
9	10	P_3	11	12
13	P_4	14	15	16

If one disk goes down, we will lose some file blocks and some parity blocks, but we can still reconstruct everything using the info on the other disks. Recovery from a disk failure can be slow, as it takes time to rebuild the contents of the lost disk using the parity blocks. Mirroring is faster in this regard.

Level 6 is like Level 5 except that there are two parity blocks. Here is an example with a 18-block file and 5 disks.

1	2	3	4	5
1	2	3	P_1	P_1'
4	5	P_2	P_2'	6
7	P_3	P_3'	8	9
P_4	P_4'	10	11	12
P_5'	13	14	15	P_5
16	17	18	P_6	P_6'

Here is a breakdown, like we've done for the others.

- Redundancy — If a disk goes down, we can use the others to reconstruct the missing data. Level 5 can handle at most one disk failure, while Level 6 can handle two.
- Capacity — Good. For Level 5 with 5 disks, 1/5 of the data is used for parity and 4/5 for holding data, so we get 4 times the capacity of a single disk. For Level 6, it's 2/5 for parity and 3/5 for data, so we get 3 times the capacity of a single disk.
- Speed — Good for reading since the reads can be done in parallel. Writes are somewhat slower because of the parity computations. Level 6 is slower than Level 5 because there is more parity stuff to do.

There are also Levels 50 and 60 that are combinations of Levels 5 and 6 with Level 0, in a similar way to how Level 10 is a combination of Level 1 with Level 0.

Levels 5 and 6 provide is a nice tradeoff between reliability and performance.

Overall summary

The most popular levels seem to be Level 10 and Level 6. If you're choosing between the different levels, read up online to see which one best suit your use case. Level 10 with 6 or more disks and Level 6 both can withstand losing 2 disks, which may be important if you are worried about losing data. Often if one disk fails, another one might not be far behind. The rebuilding process can take many hours, especially with Level 6, and it's possible another disk could fail in that time.