# Raid

Hello, and welcome to this lesson on RAID, where we will be exploring how to combine multiple hard drives together to achieve high performance, high-capacity disk systems that are easy to use.

We will start by a few observations that people started to make back in the eighties about the performance limitations of hard drives.

Once the performance of CPUs, I/O buses, and memory became sufficiently large, it became realistic to think about how to combine multiple independent disks together to form high-performance disk clusters that would be seen as a single disk by the operating system, or at least by the user. One successful such approach was RAID, or “Redundant Arrays of Independent Disks”.

RAID occurs in many variations. The building blocks for most variations are what is called RAID-level-0 to RAID-level-6. We will look at all these levels in detail.

Very common are RAID-0, RAID-1, RAID-5, and more recently RAID-6. More complicated versions can be built by hierarchically combining multiple of these basic levels together. We will look at an example.

At the end of this lesson you will understand how RAID works, and what the relative pros and cons are of different RAID levels. You will also understand some of the performance implications, and how performance sometimes has to be traded off against reliability in the presence of disk failures.

Let’s begin.

[Slide 2]

In the late 80’s it became painfully clear that disks are very slow, and there was no technology on the horizon to make them faster.

So people started looking around for approaches to bundle multiple disks together and gain performance benefits by using those disks in parallel.

As part of their work on the SPRITE operating systems, a group at the University of California at Berkeley coined the term RAID, which stands for “Redundant Arrays of Inexpensive Disks”. Of course marketing people like the term “Arrays of independent disks” much better.

[Slide 3]

Their goal was to cluster relatively affordable smaller disks together into what would be visible by the file system as a large, high-performance disk.

This clustering could be done by connecting the disks to what would become a RAID-capable hard driver controller, or to have each disk be connected through its own controller, and then have a

RAID device driver that would either manage the individual disk drive device drivers or would access the individual device controllers.

In both cases, the array of disks would be visible to the file system as one large, high-performance disk.

[Slide 4]

The storage capacity of the cluster of disks would be leveraged by distributing the data across the small disks.

This would also improve performance because the I/O requests would be distributed across the disks as well.

Some of the disk capacity now could be used to error detection and error recovery data.

## RAID (continuation)

As we mentioned earlier, the approach behind RAID is to bundle multiple smaller capacity disks into a single, high-performance, large-capacity disk.

The benefits of such an approach are an improved performance, both in terms of improved disk bandwidth as well as, in some configurations, improved disk access latency.

Also, at least in theory, RAID would allow for incremental upgrading of the storage infrastructure. One would add disks to the system, and by magic the file system would have access to a larger disk.

The problem with this approach is reliability. As we keep adding devices to the system, the overall reliability of the system decreases. As more devices can fail, it will take less time until the first device fails.

If, for example, we double the number of disks, say from 1 to 2, the failure rate of the two disks must be added together; the failure rate therefore doubles. In most failure models, when the failure rate doubles, the mean-time-to-failure, that is the expected time until the system fails, is reduced to half. This is particularly painful for hard drives, which are prone to failure to begin with.

In order to keep high reliabilities despite the large number of disks one has to resort to redundancy.

## RAID: Basic Principles

RAID relies on two basic principles. Namely “striping”, which defines how data is distributed across the multiple disks, and “redundancy”, which tells how redundancy is added to increase reliability.

We distinguish between block-level striping, where entire blocks are written to a disk at a time, and bit-level striping, where blocks are split up and written to multiple disks in parallel. (5)

In block-level striping the disks operate largely independently and don’t need to be synchronized.

Each disk has its own queue of requests.

As blocks are written to the RAID array, they are dispatched each to the appropriate disk in the array.

[Slide 7]

With bit-level striping, the block is striped, and each stripe is sent to a disk.

RAID defines different levels, which differ in how the blocks are striped and in how redundancy is provided.

## RAID Level 0

The simplest level is RAID-0, which is basically block-level striping without redundancy.

In a RAID-0 system, each block is placed in one of the disks in the array, and as requests for blocks are handled, they are forwarded to the appropriate disk.

A possible formula for the placement of blocks to disks could be the following, where the disk for a block would be the block number divided by the number of disks, and the offset, or the sector number on the disk, would be the remainder after this division.

As a result, subsequent blocks would be striped across the disks, thus allowing for very high sequential access to disk blocks.

The benefit of such a RAID-0 system is two-fold. On one hand the capacity is multiplied by the number of disks in the array. In addition, the read-write performance increases as well. Since I/O operations now are striped across multiple disks, the IO bandwidth increases with the number of disks in the array. The disk access latency benefits as well from the parallelism.

The problem with this approach is that it multiplies the failure rate of the disks in the system without providing any redundancy. The system will fail and lose all the data when the first disk fails, and the first disk will fail much earlier than in a single-disk system.

## RAID Level 1

Redundancy is addressed, in a simplistic fashion, in RAID level 1.

Here the set of disks, in this example just one disk, is duplicated, we say “mirrored”, by adding a second disk, the so-called “mirror”, and then issuing any write request to both disks.

When one of the two disks fails, one has a perfectly good copy to continue with.

[Slide 10]

The performance of RAID-1 for READ operations is a bit better than no RAID at all, as one can issue the read request to both disks and wait until the first of the two disks comes back with the requested block. This is called “split read”, and gives some performance improvement.

For WRITE, there is a penalty because the write operation has to be issued to both disks.

The problem with this approach is that the increased reliability comes at the cost of 100% redundancy. In order to survive a disk failure we are duplicating the cost and the failure rate.

## RAID Level 2

RAID-level 2, sometimes called “memory-style error-correcting parity”, has a more sophisticated approach to redundancy.

It uses bit-level striping, also called “small strips”. Disks have to be tightly synchronized for this, otherwise there are significant performance penalties.

When a block comes in, it gets split into stripes.

At this point, additional error correction bits are attached in a fashion similar that that in memory. Given that we need to protect an entire block, the number of bits will be significant. They are stored on separate disks that only store error correction code.

The stripes and the error correction stripes are all forwarded to the disk.

[Slide 12]

If a disk fails, the missing data can be recovered by applying the error correction code.

In practice this is a total overkill, as error correction code that are appropriate for memory are not at all for disks. One characteristic is that when a disk fails, the system knows which disk it is. This allows for much more compact error codes.

RAID 2 may be appropriate for systems that operate in highly adverse environment, with lots of transient error, but it is rarely deployed.

## RAID Level 3

The next level is RAID 3.

Similarly to RAID 2 it uses bit-level striping.

Instead of using expensive Error Correction Codes it works with a simple parity scheme.

Again, if a disk fails, the system knows which one it is.

This allows for the use of a simple scheme, to recover from a single-disk failure.

The parity of four strips S0 to S3 is computed by “xoring” the stripes. The result is stored in Stripe 4, which is stored in a separate disk, the parity disk.

When a disk fails, say Disk 1, which contains Stripe 1, the content can be reconstructed by xoring the remaining stripes and the parity stripe.

Note that we can use parity for error detection and correction.

In this fashion, RAID 3 can recover from a single-disk failure.

The problem with RAID 3 is, however, that it uses bit-level striping, which requires expensive synchronization of the disks.

## RAID Level 4

RAID-4 does not have this problem, as it employs block-level striping.

RAID-4, in fact is very similar to RAID 3, except that it does block-level striping. This simplifies the system significantly, as the disks don't need to be synchronized and can run independently.

RAID-4 maintains a separate parity disk, which stores the parity blocks for all the corresponding blocks on the other disks. For example,

Block number 4 contains the parity information of all blocks 4 on the other disks.

Whenever a block is written to a disk, the parity information is updated in the corresponding parity block on the parity disk.

[Slide 15]

In order to look at this in a bit more detail, let's represent the disks schematically as stacks of blocks.

If a block has to be written to sector 1 in disk 1,

Sector 1 on the parity disk, which contains the parity of the sector 1's in all disks, must be updated.

The update is actually easy. The new parity information is the current parity xored with the current value in block and the old value in the block.

In this formula, we use the notation X4 of I to denote the I'th bit in Strip 4, which is on the parity disk.

Every time we write to a block, we read the parity strip, and xor it with the (5) new value for the block and the existing value of the block the new value of the parity strip is written to the parity disk.

[Slide 16]

We note that in order to update the parity information we don't need to read the strips on the other disks.

Instead we XOR the existing parity strip with the old value of the block and the new value of the block.

Let's look why this works. X4 is the old parity.

The new parity should be the XOR of the updated value and the existing values on the other disks.

If we add an XOR of X1 XORED with itself, nothing should change. (5)

Let's reorder things, which also does not change anything.

Now we can replace X0 XORED with X1 to X3 by the parity X4

Which gives us the expected result.

When re-computing the parity bit, we don't need to read all the other blocks from the disks. Parity depends on difference from old to new, and old parity.

Performance: Data transfer can suffer because of parity (in particular writes).

[Slide 17]

The problem with RAID 4 is that the parity disk gets lots of load.

In fact, every time we write to one of the blocks we write to the parity disk as well. For example, if the block is in one of the strips 2 of any of the disks, we write to strip to on the parity disk.

For strip 1 we write to strip one on the parity disk.

We do the same for any other strip, such as strip 3 in this case.

The solution, implemented in RAID 5, is to spread the parity strips across disks.

## RAID Level 5

Let's see how this works.

[Slide 19]

In this example, the parity for strip 0 is stored on disk 0.

[Slide 20]

for strip 1 on disk 1

[Slide 21]

and so on.

[Slide 22]

(nothing)

[Slide 23]

(nothing)

[Slide 24]

We then wrap around and store the parity for strip 5 on disk 0 again.

In this way we can recover from a single-disk failure by reconstructing the missing data from the existing data and the parity.

The problem with this approach has become apparent recently in systems with very large disks, for which a failure takes a long time to recover from. If a second failure happens during the recovery, the system loses all its data.

## RAID Level 6

This gave rise to a more recent RAID level, called RAID 6.

Raid 6 is basically the same as RAID 5, except that it maintains to parity strips, on two different disks.

Whenever it writes a block to disk, it computes two parity strips, using two independent parity functions. The two strips are then stored on two separate disks.

In this way it can tolerate two failures.

This comes in handy when a second failure happens during a recovery from a first failure.

## Nested Levels: e.g. RAID Level 1+0 = RAID 10

Given the 6 basic RAID levels one can build larger RAID systems.

For example, one can build a RAID 10 system by combining RAID1 with RAID 0.

We start with two mirrored disks, in RAID-1 fashion, and stripe over them using RAID 0.

The result is a RAID 10 system. Note that we parse the RAID number from the disk up. Here 1 followed by 0.

Other examples:

--- RAID 0+1 (requires min 3 disks)

--- RAID 0+3 (dedicated parity array across striped disks)

--- RAID 3+0 (striping of dedicated parity arrays)

--- RAID 100: implemented using software raid 0 over hardware raid 10

## RAID - Conclusion

We have come to the conclusion of this lesson on RAID.

We motivated the need for disk arrays and then focused on RAID as a way to combine independent disks.

We discussed RAID 0 to RAID 6 and learned how to combine these basic RAID levels into larger systems.

We hope that you enjoyed this lesson on RAID.

RAID has been for many years a way to scale the performance of storage systems using collections of disks. Given the fact that RAID is largely agnostic about the underlying storage technology, it will stay relevant despite the increasing popularity of new storage technologies, such as NAND Flash and Phase Change Memory.

Thank you for watching.