

【OS】 Day41(2)

▼ Class	Operating System: Three Easy Pieces
📅 Date	@February 22, 2022

【Ch38】 Redundant Arrays of Inexpensive Disks(RAIDs)

In this chapter, we introduce [the Redundant Array of Inexpensive Disks](#) better known as [RAID](#), a technique to [use multiple disks in concert to build a faster, bigger, and more reliable disk system](#).

A hardware RAID is very much like a computer system, specialized for the task of managing a group of disks.

Externally, [a RAID looks like a disk](#): a group of blocks one can read or write. Internally, the RAID is a complex beast, consisting of multiple disks, memory(both volatile and non-), and one more more processors to manage the system.

RAIDs offer a number of advantages over a single disk:

1. One advantage is [performance](#). Using multiple disks [in parallel can greatly speed up I/O times](#).
2. Another benefit is [capacity](#). Large data sets demand large disks.
3. Finally, RAID's can improve [reliability](#); spreading data across multiple disks makes the data vulnerable to the loss of a single disk; with some form of [redundancy](#), RAID's can [tolerate the loss of a disk](#) and keep operating as if nothing were wrong.

Amazingly, RAID's provide these advantages [transparently](#) to systems that use them, i.e., a RAID just looks like a big disk to the host system. The beauty of transparency is that [it enables one to simply replace a disk with a RAID and not change a single line of software](#). In this manner, transparency greatly improves the [deployability](#) of RAID.

38.1 Interface and RAID Internals

When a file system issues a logical I/O request to the RAID, the RAID **internally must calculate which disk to access in order to complete the request**, and then issue one or more physical I/Os to do so.

However, consider a RAID that keeps two copies of each block; when writing to such a **mirrored** RAID system, the RAID will have to **perform two physical I/Os for every one logical I/O it is issued**.

At a high level, a RAID is very much **a specialized computer system**: it has a processor, memory, and disks. However, instead of running applications, it **runs specialized software designed to operate the RAID**.

38.2 Fault Model

RAIDs are designed **to detect and recover from certain kinds of disk faults**; thus, knowing exactly which faults to expect is critical in arriving upon a working design.

The first model we will assume is called **the fail-stop fault model**. In this model, a disk can be in exactly one of two states: **working or failed**.

With a working disk, all blocks can be read or written. In contrast, when a disk has failed, we assume **it is permanently lost**.

When a disk has failed, we **assume that this is easily detected**. For example, in a RAID array, we would assume that the RAID controller hardware can immediately observe when a disk has failed.

38.3 How To Evaluate a RAID

We will evaluate each RAID design along three axes.

1. The first axis is **capacity**; given a set of N disks each with B blocks, *how much useful capacity is available to clients of the RAID?*

Without redundancy, the answer is $N \times B$

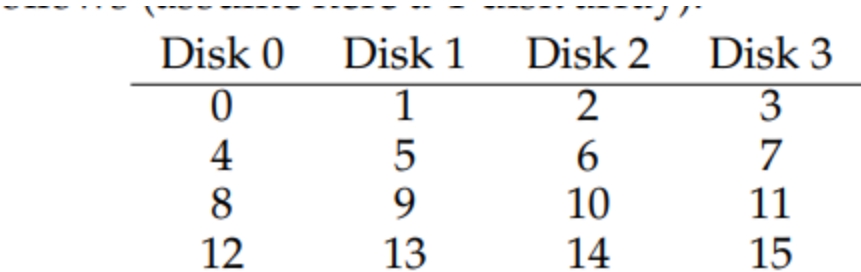
If we have a system that keeps two copies of each block(called **mirroring**), we obtain a useful capacity of $(N \times B)/2$

2. The second axis is **reliability**. **How many disk faults** can the given design tolerate.
3. Finally, **performance**.

38.4 RAID Level 0: Striping

The first RAID level is actually not a RAID level at all, in that **there is no redundancy**.

However, RAID level 0, or **striping** as it is better known, serves as an excellent **upper-bound on performance and capacity**.



The diagram shows a 4-disk array with disks labeled Disk 0, Disk 1, Disk 2, and Disk 3. Data blocks are striped across the disks in a round-robin fashion. The blocks are numbered 0 through 15. Blocks 0, 1, 2, and 3 are on Disk 0; blocks 4, 5, 6, and 7 are on Disk 1; blocks 8, 9, 10, and 11 are on Disk 2; and blocks 12, 13, 14, and 15 are on Disk 3.

Disk 0	Disk 1	Disk 2	Disk 3
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Figure 38.1: **RAID-0: Simple Striping**

The simplest form of striping will stripe blocks across the disks of the system as follows(assume here a 4-disk array):

From the figure above, we get the basic idea: **spread the blocks of the array across the disks in a round-robin fashion**. This approach is designed to **extract the most parallelism from the array** when requests are made for contiguous chunks for the array. We call the blocks in the same row a stripe; thus, blocks 0, 1, 2, and 3 are in the same strip above.

In this example, we have made the simplifying assumption that only 1 block(each of say size 4KB) is placed on each disk before moving on to the next. However, this arrangement need not be the case.

For example, we could arrange the blocks across disks as in the following figure:

Disk 0	Disk 1	Disk 2	Disk 3	
0	2	4	6	chunk size:
1	3	5	7	2 blocks
8	10	12	14	
9	11	13	15	

Figure 38.2: Striping With A Bigger Chunk Size

In this example, we place two 4KB blocks on each disk before moving on to the next disk. Thus, the chunk size of this RAID array is 8KB, and a stripe thus consists of 4 chunks or 32KB of data.

Chunk Sizes

Chunk size mostly affects performance of the array.

For example, a small chunk size implies that many files will get striped across many disks, thus increasing the parallelism of reads and writes to a single file; however, the positioning time to access blocks across multiple disks increases, because the positioning time for the entire request is determined by the maximum of the positioning times of the requests across all drives.

A big chunk size, on the other hand, reduces such intra-file parallelism, and thus relies on multiple concurrent requests to achieve high throughput. However, large chunk sizes reduce positioning time. If, for example, a single file fits within a chunk and thus is placed on a single disk, the positioning time incurred while accessing it will just be the positioning time of a single disk.