CS 333 :: Topic Notes :: Redundant Arrays of Inexpensive Disks (RAID)

Why Explore RAID?

When first designing an undergraduate storage course for a liberal arts college, I told myself that RAID is something that should *not* be covered. RAID was too low-level and too narrow. Instead, we should focus on the "big ideas" that are applicable to multiple storage applications, or better yet, ideas that represent general systems design principles.

Then I thought more about what RAID is and what it is not. I realized that the problem with many RAID treatments I've seen is that they focus on memorizing the different RAID "levels". Is it important that you can map the term "RAID-0" to a specific configuration? I don't think so; memorizing RAID levels is simply the wrong way to approach it.

RAID applies a collection of ideas that are now ubiquitous in system design. Whether we realize it or not, these ideas are in real systems that we use, and they will show up in several topics throughout the rest of this course. The things I want you to take away from the chapter on RAID are the following big ideas:

- Striping
- Replication
- Parity
- Faults (models and tolerance)

When thinking about each of these ideas, I want you to think about how each of them affect:

- Performance
- Capacity
- Fault tolerance

Fault Models

One of RAID's key motivations is fault tolerance. But before we can describe how RAID overcomes faults, we need to define what a fault is, and what types of faults RAID can overcome.

RAID assumes disk errors are fail-stop. A fail-stop error is one where the disk does not work *at all*. So RAID's assumptions are that the disk either works entirely, or it fails entirely, and RAID assumes that the system can always detect which state the disk is in.

Questions

- In addition to fail-stop, what other ways can you imagine a device failing?
- Why might these be challenging failure models for a system to overcome?

Evaluation Criteria

As stated above, we should be thinking about RAID using three criteria:

- Performance
  - Is performance better or worse than naively using a single disk? Performance should be evaluated for a range of common workloads (sequential reads and writes, random reads and writes, overwrites, etc.) and in terms of both latency and throughput.
- Capacity
  - The textbook defines *N* as the number of disks, and *B* as the number of blocks that a disk can hold. We will evaluate capacity in terms of the total amount of data a RAID system can store; an ideal system would be able to hold *N* x *B* units of data.
- Fault tolerance
  - How many disks can fail before the system loses data? Usually we can define a minimum number of disks failures that a RAID system can recover from. In some cases, the system can recover from more than that minimum number of failures if the stars align and specific combinations of disks fail.

Technique 1: Striping

When striping, you (evenly) spread your object space over a collection of devices.

- A system that performs striping may be able to satisfy a large request by issuing several smaller requests in parallel to independent devices. For sequential I/O, this can often result in substantial performance gains.

With RAID, the objects correspond to LBAs or chunks. In general, systems can stripe in a variety of ways. The key idea is that a single request is spread across multiple independent units, allowing the system to benefit from parallelization. In general, performance increases as the size of the request grows.

Performance

When striping, a large I/O comprising *L* blocks can be broken into roughly *N* requests each of size *L/N*.

- The performance for a large sequential I/O of size *L* is then the performance of the slowest device to perform an I/O of size *L/N* (both the latency and throughput)
- This is about as good as we can do. We get almost full parallelization!

When striping, there is little benefit for small or random I/Os.

- For the smallest I/O, it corresponds to reading a single block from a single device.
- For requests that aren't much larger than a "stripe size", most of the time is spent performing the I/Os' setup costs on the individual disks. The transfer costs are dominated by the setup costs, so the performance gain is minimal (if any). In fact, a single sluggish disk can bring down the performance...

Capacity

No space is wasted. We utilize the full capacity of all *N* disks.

Fault Tolerance

We cannot survive any disk failures without data loss. Striping by itself only yields performance benefits.

Technique 2: Replication (mirroring)

When replicating, we create *R* copies of each object, and place the copies on independent disks. In the case of RAID, objects are blocks or chunks. The general idea of replication can be applied to arbitrary objects. For example, data may be "geo-replicated" so that copies of data are kept in multiple data-centers around the world. This helps in a variety of ways:

- If one data center goes down or is disconnected, data can still be retrieved from other data centers.
- Data can be accessed from the closest or least busy data center.
- Data can be written to the closest data center, and in many cases, the application can often be satisfied that its data will "eventually" make its way to all the replication sites.

So replication is generally useful. In the context of RAID, it provides a simple model for surviving failures.

Performance

Among all the types of workloads, mirroring helps the performance of random reads the most: if the system receives two independent random read requests, it can send them two independent disks. The requests therefore don't have to compete for a single disk arm, and they can parallelize the seek costs.

For write requests, however, we must write all of the replicates. This obviously harms throughput compared to writing each piece of data exactly once. If we have *N* disks and create *R* replicates, we can achieve at most *N/R* of the peak bandwidth of all the disks.

Capacity

Replication reduces the storage capacity; since we are replicating the data, we "waste" space with the extra copies. As the number of replicates goes up, the system's usable capacity goes down.

Fault Tolerance

So why replicate? Replication harms the performance of most operations, and it reduces the system's usable capacity?

The primary reason to replicate is that replication lets us survive disk failures; if we lose one disk, we still have copies of each object that was stored on that disk. So if we create *R* replicates, we can survive *R-1* failures.

Technique 3: Parity

Parity gives some of the fault tolerance benefits of replication, but uses less space. The system groups data together into units called *parity groups*. In the simplest application of parity, a single parity block is stored for each parity group, often by XORing all of the bits of the blocks in the parity group. If any one block from the parity group is lost, that block can be reconstructed using the remaining blocks in the parity group.

Performance

The performance of parity depends on the operation.

For sequential writes, the data can be striped across *N-1* disks (the last disk must be used for the parity block). This is quite good. These benefits also apply to sequential reads: you can read from *N-1* disks in parallel. The analysis is similar to striping.

For updates and random writes, it is more complicated.

- If using *additive parity*, you read the entire parity group in order to recalculate the parity block. Then, you write at least two blocks: the changed block(s) and the new parity block.
- If using *subtractive parity*, you can instead read just the changed block(s) and the parity block. Then, for each bit in each changed block, you decide whether you need to update the parity block by comparing the old bit value to the new bit value.

Thus, for non-sequential writes, parity performance is poor.

Capacity

The only capacity overheads in a parity-based system come from the parity blocks themselves. Thus, the usable capacity is reduced by one disk.

Fault Tolerance

The system can survive the loss of any one disk. For each parity group, the missing block can be reconstructed using the other members of the parity group.

Reflecting on RAID

RAID ideas are useful on several dimensions. They can be applied to improve performance (striping for sequential I/O, replication for random reads), and they can be applied to survive hardware failures (replication and parity).

The ideas are also not limited to writing LBAs to hard drives.

- RAID is largely device agnostic (although we will see that there are challenges when applying RAID to devices are not random-write capable). You can apply RAID ideas to arrays of SSDs, arrays of storage servers, or even sets of data centers.
- Striping, replication, and parity can be abstracted to other domains. For example, within SSDs, striping can be applied by the FTL for parallelism. Parity can be used to provide integrity and detect data tampering. There are many creative ways to combine and apply these ideas.

I implore you not to memorize RAID level numbers. Your time is much better spent thinking about how you might combine the core concepts. How does striping interact with replication? Do you replicate first or stripe first? How many parity disks should you use? Should you distribute the parity blocks?