Chapter 10: Mass Storage Structure
RAID

Chen Hao
haochen@aimlab.org
Hunan University

Crux: How To Make A Large, Fast, Reliable Disk

How can we make a large, fast, and reliable storage system? What are the key techniques? What are trade-offs between different approaches?

Redundant Arrays of Inexpensive Disks

- 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.

- The term was introduced in the late 1980s by a group of researchers at U.C. Berkeley (led by Professors David Patterson and Randy Katz and then student Garth Gibson).

- 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 or more processors to manage the system.
- A hardware RAID is very much like a computer system, specialized for the task of managing a group of disks.

- RAIDs offer a number of advantages over a single disk.
- One advantage is *performance*. Using multiple disks in parallel can greatly speed up I/O times.
- Another benefit is *capacity*. Large data sets demand large disks.
- Finally, RAIDs can improve *reliability*; spreading data across multiple disks makes the data vulnerable to the loss of a single disk; with some form of redundancy, RAIDs can tolerate the loss of a disk and keep operating as if nothing were wrong.

Interface And RAID Internals

- To a file system above, a RAID looks like a big, (hopefully) fast, and (hopefully) reliable disk.
- Just as with a single disk, it presents itself as a linear array of blocks, each of which can be read or written by the file system.
- When a file system issues a *logical I/O* request to the RAID, the RAID internally must calculate which disk (or disks) to access in order to complete the request, and then issue one or more *physical I/Os* to do so.
- The exact nature of these physical I/Os depends on the RAID level, as we will discuss in detail below.

How To Evaluate A RAID

- We evaluate each RAID design along three axes.
- The first axis is capacity; given a set of N disks, how much useful capacity is available to clients of the RAID?
- The second axis of evaluation is reliability. How many disk faults can the given design tolerate?
- The third axis is performance. Performance is somewhat challenging to evaluate, because it depends heavily on the workload presented to the disk array.

- We now consider three important RAID designs: RAID Level 0 (striping), RAID Level 1 (mirroring), and RAID Levels 4/5 (parity-based redundancy).

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 and thus is worth understanding.

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

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

RAID-0: Simple Striping

- 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 Table:

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	

Striping with a Bigger Chunk Size

- 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.
- Determining the “best” chunk size is hard to do. For our discussion, we assume that the array uses a chunk size of a single block (4KB). Most arrays use larger chunk sizes (e.g., 64 KB).

Evaluating RAID Performance

- In analyzing RAID performance, one can consider two different performance metrics.
- The first is *single-request latency*. Understanding the latency of a single I/O request to a RAID is useful as it reveals how much parallelism can exist during a single logical I/O operation.
- The second is *steady-state throughput* of the RAID, i.e., the total bandwidth of many concurrent requests. The steady-state bandwidth is critical, and thus will be the main focus of our analyses.
- We assume that there are two types of workloads: sequential and random. We assume that a disk can transfer data at S MB/s under a sequential workload, and R MB/s when under a random workload.

- Specifically, let's calculate S and R given the following disk characteristics. Assume a sequential transfer of size 10 MB on average, and a random transfer of 10 KB on average. Also, assume the following disk characteristics:

Average seek time	7 ms
Average rotational delay	3 ms
Transfer rate of disk	50 MB/s

$$S = \frac{Amount\ of\ Data}{Time\ to\ access} = \frac{10\ MB}{210\ ms} = 47.62\ MB/s$$

$$R = \frac{Amount\ of\ Data}{Time\ to\ access} = \frac{10\ KB}{10.195\ ms} = 0.981\ MB/s$$

RAID-0 Analysis

- Let us now evaluate the capacity, reliability, and performance of striping.
- From the perspective of capacity, it is perfect: given N disks, striping delivers N disks worth of useful capacity.
- From the standpoint of reliability, striping is also perfect, but in the bad way: any disk failure will lead to data loss.
- Finally, performance is excellent: all disks are utilized, often in parallel, to service user I/O requests.
- From the perspective of steady-state throughput, we'd expect to get the full bandwidth of the system. Thus, throughput equals N (the number of disks) multiplied by S ($N \cdot S$ MB/s) . For a large number of random I/Os, we can again use all of the disks, and thus obtain $N \cdot R$ MB/s.

RAID Level 1: Mirroring

- With a mirrored system, we simply make more than one copy of each block in the system; each copy should be placed on a separate disk, of course. By doing so, we can tolerate disk failures.

- In a typical mirrored system, we will assume that for each logical block, the RAID keeps two physical copies of it. Here is an example:

Disk 0	Disk 1	Disk 2	Disk 3
0	0	1	1
2	2	3	3
4	4	5	5
6	6	7	7

Simple RAID-1: Mirroring

- When reading a block from a mirrored array, the RAID has a choice: it can read either copy.
- When writing a block, though, no such choice exists: the RAID must update *both* copies of the data, in order to preserve reliability. Do note, though, that these writes can take place in parallel;

RAID-1 Analysis - Capacity

- From a capacity standpoint, RAID-1 is expensive; with the mirroring level = 2, we only obtain half of our peak useful capacity.
- Thus, with N disks, the useful capacity of mirroring is N/2.

RAID-1 Analysis - Reliability

- From a reliability standpoint, RAID-1 does well. It can tolerate the failure of any one disk.

- You may also notice RAID-1 can actually do better than this, with a little luck.

- More generally, a mirrored system (with mirroring level of 2) can tolerate 1 disk failure for certain, and up to N/2 failures depending on which disks fail.

RAID-1 Analysis - Performance

- From the perspective of the latency of a single read request, we can see it is the same as the latency on a single disk.

- A write is a little different: it requires two physical writes to complete before it is done. These two writes happen in parallel, and thus the time will be roughly equivalent to the time of a single write.

RAID-1 Analysis - Performance

- For steady-state throughput, when writing out to disk sequentially, each logical write must result in two physical writes. Thus, the maximum bandwidth is (N/2) · S MB/s.
- The sequential read will only obtain a bandwidth of (N/2) · S MB/s. Why?
- Imagine we need to read blocks 0, 1, 2, 3, 4, 5, 6, and 7. Let's say we issue the read of 0 to disk 0, the read of 1 to disk 2, the read of 2 to disk 1, and the read of 3 to disk 3. We continue by issuing reads to 4, 5, 6, and 7 to disks 0, 2, 1, and 3, respectively.

Disk 0	Disk 1	Disk 2	Disk 3
0	0	1	1
2	2	3	3
4	4	5	5
6	6	7	7

RAID-1 Analysis - Performance

- Random reads are the best case for a mirrored RAID. In this case, we can distribute the reads across all the disks, and thus obtain the full possible bandwidth. Thus, for random reads, RAID-1 delivers $N \cdot R$ MB/s.
- Random writes perform as you might expect: $(N/2) \cdot R$ MB/s.

RAID Level 4: Saving Space With Parity

- We now present a different method of adding redundancy to a disk array known as parity.

- Parity-based approaches attempt to use less capacity and thus overcome the huge space penalty paid by mirrored systems.

- They do so at a cost, however: performance.

- In a five-disk RAID-4 system, we might observe the following layout:

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
4	5	6	7	P1
8	9	10	11	P2
12	13	14	15	P3

- To compute parity, we need to use a mathematical function that enables us to withstand the loss of any one block from our stripe. It turns out the simple function XOR does the trick quite nicely.

C0	C1	C2	C3	P
0	0	1	1	XOR(0,0,1,1) = 0
0	1	0	0	XOR(0,1,0,0) = 1

- Now you might be wondering: we are talking about XORing all of these bits, and yet above we know that the RAID places 4KB (or larger) blocks on each disk; how do we apply XOR to a bunch of blocks to compute the parity?

- It turns out this is easy as well. Simply perform a bitwise XOR across each bit of the data blocks; put the result of each bitwise XOR into the corresponding bit slot in the parity block.

- Given 4 blocks 0010, 1001, 1100, 1001, how to get the parity block?

RAID-4 Analysis - Capacity

- From a capacity standpoint, RAID-4 uses 1 disk for parity information for every group of disks it is protecting. Thus, our useful capacity for a RAID group is N-1.

RAID-4 Analysis - Reliability

- Reliability is also quite easy to understand: RAID-4 tolerates 1 disk failure and no more. If more than one disk is lost, there is simply no way to reconstruct the lost data.

RAID-4 Analysis - Performance

- Sequential read performance can utilize all of the disks except for the parity disk, and thus deliver a peak effective bandwidth of $(N - 1) \cdot S$ MB/s (an easy case).

- To understand the performance of sequential writes, we must first understand how they are done. When writing a big chunk of data to disk, RAID-4 can perform a simple optimization known as a full-stripe write.

- For example, imagine the case where the blocks 0, 1, 2, and 3 have been sent to the RAID as part of a write request:

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
4	5	6	7	P1
8	9	10	11	P2
12	13	14	15	P3

- In this case, the RAID can simply calculate the new value of P0 (by performing an XOR across the blocks 0, 1, 2, and 3) and then write all of the blocks (including the parity block) to the five disks above in parallel (highlighted in gray in the figure).

- Thus, full-stripe writes are the most efficient way for RAID-4 to write to disk.

- Once we understand the full-stripe write, calculating the performance of sequential writes on RAID-4 is easy; the effective bandwidth is also $(N - 1) \cdot S$ MB/s.

- A set of 1-block random reads will be spread across the data disks of the system but not the parity disk. Thus, the effective performance is: $(N - 1) \cdot R$ MB/s.

- Random writes present the most interesting case for RAID-4. Imagine we wish to overwrite one block, and how can we update it both correctly and efficiently?

- There are two methods. The first, known as additive parity, requires us to do the following.

- To compute the value of the new parity block, read in all of the other data blocks in the stripe in parallel (in the example, blocks 0, 2, and 3) and XOR those with the new block. The result is your new parity block.

- To complete the write, you can then write the new data and new parity to their respective disks, also in parallel.

- The second, known as subtractive parity. For example, imagine this string of bits (4 data bits, one parity):

C0	C1	C2	C3	P
0	0	1	1	XOR(0,0,1,1) = 0

- Let's imagine that we wish to overwrite bit C2 with a new value which we will call C2(new). The subtractive method works in three steps.

- First, we read in the old data at C2 (C2(old) = 1) and the old parity (P(old) = 0).

- Then, we compare the old data and the new data; if they are the same (e.g., C2(new) = C2(old)), then we know the parity bit will also remain the same (i.e., P(new) = P(old)).

- If, however, they are different, then we must flip the old parity bit to the opposite of its current state, that is, if (P(old) == 1), P(new) will be set to 0; if (P(old) == 0), P(new) will be set to 1.

- We can express this whole mess neatly with XOR as it turns out (if you understand XOR, this will now make sense to you):

```
P(new) = (C(old) XOR C(new)) XOR P(old)
```

- For this performance analysis, let us assume we are using the subtractive method.

- Thus, for each write, the RAID has to perform 4 physical I/Os (two reads and two writes). Now imagine there are lots of writes submitted to the RAID; how many can RAID-4 perform in parallel?

- To understand, let us again look at the RAID-4 layout:

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
*4	5	6	7	+P1
8	9	10	11	P2
12	*13	14	15	+P3

- Now imagine there were 2 small writes submitted to the RAID-4 at about the same time, to blocks 4 and 13. The data for those disks is on disks 0 and 1, and thus the read and write to data could happen in parallel, which is good.

- The problem that arises is with the parity disk; both the requests have to read the related parity blocks for 4 and 13, parity blocks 1 and 3.

- The parity disk is a bottleneck under this type of workload; we call this the small-write problem for parity-based RAIDs.

- Because the parity disk has to perform two I/Os (one read, one write) per logical I/O, we can compute the performance of small random writes in RAID-4 by computing the parity disk's performance on those two I/Os, and thus we achieve (R/2) MB/s.
- RAID-4 throughput under random small writes is terrible; it does not improve as you add disks to the system.

RAID Level 5: Rotating Parity

- To address the small-write problem (at least, partially), Patterson, Gibson, and Katz introduced RAID-5. RAID-5 works almost identically to RAID-4, except that it rotates the parity block across drives.

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19

RAID-5 With Rotated Parity

RAID-5 Analysis?

	RAID-0	RAID-1	RAID-4
Capacity	N	$N/2$	$N-1$
Reliability	0	1 (for sure) $\frac{N}{2}$ (if lucky)	1
Throughput			
Sequential Read	$N \cdot S$	$(N/2) \cdot S$	$(N-1) \cdot S$
Sequential Write	$N \cdot S$	$(N/2) \cdot S$	$(N-1) \cdot S$
Random Read	$N \cdot R$	$N \cdot R$	$(N-1) \cdot R$
Random Write	$N \cdot R$	$(N/2) \cdot R$	$\frac{1}{2} \cdot R$

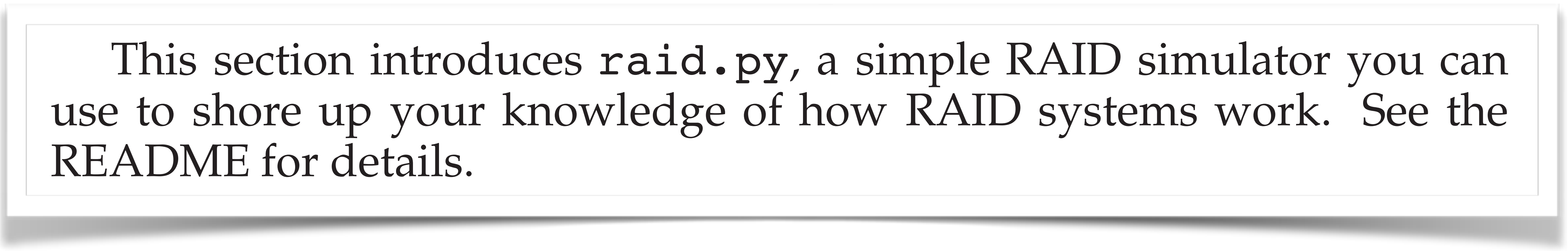
Summary

	RAID-0	RAID-1	RAID-4	RAID-5
Capacity	N	$N/2$	$N-1$	$N-1$
Reliability	0	1 (for sure) $\frac{N}{2}$ (if lucky)	1	1
Throughput				
Sequential Read	$N \cdot S$	$(N/2) \cdot S$	$(N-1) \cdot S$	$(N-1) \cdot S$
Sequential Write	$N \cdot S$	$(N/2) \cdot S$	$(N-1) \cdot S$	$(N-1) \cdot S$
Random Read	$N \cdot R$	$N \cdot R$	$(N-1) \cdot R$	$N \cdot R$
Random Write	$N \cdot R$	$(N/2) \cdot R$	$\frac{1}{2} \cdot R$	$\frac{N}{4} R$

RAID Capacity, Reliability, and Performance

Homework

http://pages.cs.wisc.edu/~remzi/OSTEP/Homework/HW-Raid.tgz



This section introduces `raid.py`, a simple RAID simulator you can use to shore up your knowledge of how RAID systems work. See the README for details.

Questions

1. Use the simulator to perform some basic RAID mapping tests. Run with different levels (0, 1, 4, 5) and see if you can figure out the mappings of a set of requests. For RAID-5, see if you can figure out the difference between left-symmetric and left-asymmetric layouts. Use some different random seeds to generate different problems than above.
2. Do the same as the first problem, but this time vary the chunk size with `-C`. How does chunk size change the mappings?
3. Do the same as above, but use the `-r` flag to reverse the nature of each problem.
4. Now use the reverse flag but increase the size of each request with the `-S` flag. Try specifying sizes of 8k, 12k, and 16k, while varying the RAID level. What happens to the underlying I/O pattern when the size of the request increases? Make sure to try this with the sequential workload too (`-W sequential`); for what request sizes are RAID-4 and RAID-5 much more I/O efficient?
5. Use the timing mode of the simulator (`-t`) to estimate the performance of 100 random reads to the RAID, while varying the RAID levels, using 4 disks.
6. Do the same as above, but increase the number of disks. How does the performance of each RAID level scale as the number of disks increases?
7. Do the same as above, but use all writes (`-w 100`) instead of reads. How does the performance of each RAID level scale now? Can you do a rough estimate of the time it will take to complete the workload of 100 random writes?
8. Run the timing mode one last time, but this time with a sequential workload (`-W sequential`). How does the performance vary with RAID level, and when doing reads versus writes? How about when varying the size of each request? What size should you write to a RAID when using RAID-4 or RAID-5?