【OS】Day43

Class	Operating System: Three Easy Pieces
Date	@February 27, 2022

【Ch38】Redundant Arrays of Inexpensive Disks(3)

38.6 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.

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

Figure 38.4: **RAID-4 With Parity**

For each stripe of data, we have added a single parity block that stores the redundant information for that stripe of blocks. For example, P1 has redundant information that it calculated from blocks 4, 5, 6, and 7.

To compute parity, we need to use a mathematical function that enables us to withstand the loss of any one block from our stripe. 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

The number of 1s in any row, including the parity bit, must be an even number. That is the invariant that the RAID must maintain in order for parity to be correct.

From the information above, we are able to guess how parity information can be used to recover from a failure. Imagine the column C2 is lost.

To figure out what values must have been in the column, we simply have to read in all the other values in that row(including the XOR'd parity bit) and reconstruct the right answer.

By reading the other values in that row(0 from C0, 0 from C1, 1 from C3, and 0 from the parity column P), we get the values 0, 0, 1, and 0. Because we know that XOR keeps an even number of 1's in each row, we know that the missing data must be a 1.

Block0	Block1	Block2	Block3	Parity
00	10	11	10	11
10	01	00	01	10

To apply this onto our disk, we simply perform bitwise XOR on every bit in each block.

RAID-4 Analysis

Since 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 $(N-1) \times B$.

Reliability is also quite easy to understand: RAID-4 tolerates 1 disk failure and no mroe.

Performance:

- Sequential read performance can utilize all of the disks except for the parity disk, and thus deliver a peak effective bandwidth of $(N-1) \times S$ MB/s.
- When we do sequential writes of a big chunk of data, RAID-4 can perform a simple optimization known as a full-stripe write.

  In this case, the RAID can simply calculate the new value of P and then write all of the blocks. The bandwidth is also $(N-1) \times S$ MB/s.
- The performance of random reads is $(N-1) \times R$ MB/s.

  Additive parity requires us to XOR all of the blocks once again.

  Subtractive parity method compares the old value with the new value. If they are the same, we know the parity bit will also remain the same. If they are different, the parity bit needs to be switched.

$$P_{new} = (C_{old} \oplus C_{new}) \oplus P_{old}$$

38.7 RAID Level 5: Rotating Parity

RAID-5 workds almot identically to RAID-4, except that it rotates the parity block across drive.

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

Figure 38.7: **RAID-5 With Rotated Parity**

The random write improves significantly over RAID-4. Imagine if we want to write to block 1 and block 10, now we can perform the writes in parallel(write disk 1 and disk 4 and write disk 0 and disk 2).

38.8 RAID Comparison: A Summary

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

Figure 38.8: **RAID Capacity, Reliability, and Performance**