Disk Failures and Disk Recovery

CS157B

Chris Pollett

Feb 2, 2005.

Outline

- Disk Failures
- Failure Model for Disks
- Mirroring as a Redundancy Technique
- Parity Blocks
- Raid Levels
- Coping with Multiple Disk Crashes

Disk Failures

- Intermittent failures - we try to do a read or write but are unsuccessful. After repeated tries though we succeed.
- Media decay - some bits on a sector become permanently corrupted and unreadable.
- Write Failures - we write some data but fail. Worse still, the old data is also gone.
- Disk crash -- entire disk is toasted. That is, suddenly, but permanently unreadable.

Today, we want to understand how to combat such failures.

Intermittent Failures

- Disk sectors are usually stored with some extra bits that enable us to tell if the sector’s content is correct or not on a read or a write.
- We model a disk read as returning a pair (w,s) where w is the data and s is the status as to whether the read was okay.
- In an intermittent failure, we may get several bad s’s before a good one.
- After a write, we can do a read to see if the s is correct, and hence, if the write was successful

Checksums

- What should we use for s?
  - A common thing to do is to take the parity of the bits in the sector for s. That is, s is 1 if the number of 1 bits in the sector is odd and 0 otherwise.
  - For example, if w=0101100 then s= 1. If w=1111000 then s=0.

Stable Storage

- Checksums let us test for errors -- they don't let us correct them.
- We want a policy in effect to be sure that, when we write to disk, we will get a sector which is correctly written, or that any data that was in the sector before the write can be safely recovered.
- This policy is called **stable storage**.

Implementing Stable Storage

- We pair sectors and each pair represents one-sector contents X.
- Denote the left and right copies X_L and X_R.
- To write:
  - Write X into X_L .Check that s is good. If not repeat the write. If fail repeatedly assume X_L corrupted and replace it with a different sector. Repeat this process for X_R.
- To read:
  - Try to read X from X_L. Check if s is good. If yes, then that is X, if not repeat. If fail repeatedly try to get the value from X_R.

- Above scheme works well for media and write failures.

Failure Model for Disks

- We now want to consider RAID (redundant arrays of independent disks) schemes that work well for disk crashes.
- The average time till a disk crash is called the mean time to failure (MTTF). For modern disk this is about 10 years.
- We model this by assuming roughly 10% of the disks fail in a given year.
- We want to use the multiple disks to make the mean time to data loss to be a much bigger number than the MTTF.

Mirroring as a Redundancy Technique

- Called Raid level 1.
- Have two disks with the same data.
- One is called the data disk and the other is the redundant disk.
- If a disk fails we replace it with a new disk and copy the data from the other disk that is still okay.
- What is the chance the second disk could fail during the replacement process?
  - If replacement takes 3hours = 1/8 of a day = 1/2920 of a year, then the odds are 1/10*1/2920 or one in 29200 failures. The odds that one of the two disks fails in a given year is 2/10 =1/5. So MTTF for 2 disk is 5 years. So the mean time to data loss is 5*29200 = 146,000 years.

Parity Blocks

- Mirroring is somewhat wasteful.
- The RAID level 4 scheme assumes we have n disks disks and one parity disk.
- The ith block of the parity disks consists of the parities of the data stored in the ith block of the other disks
  - For example, if disk 1 block i had 1010 and disk2 block i had 1101 then a parity disk would have 0111.
- To read in this setup we don’t bother with the parity disk.
- To write need to read old block and parity block. Then write new block and recalculate the parity.

RAID 4 and Failures

- If the parity disk crashes, we just replace it.
- If one disk fails then we can replace that disk and compute the value for each of its block based on the other disks and the parity blocks. For example, if
  - Disk1blk1: 1001
  - Disk2blk1: 1111
  - Disk3blk1: ????
  - ParityBlk1:0010
  - Then Disk3blk1 must be:0100

RAID 5

- RAID 4 has a bottleneck whenever we need to do writes because we must also update the parity disk.
- Notice the rule of recovery for a parity disk is the same as for any other disk in RAID level 4.
- In RAID level 5 to reduce the bottleneck problem we change which disk is the parity disk depending on the block number. For instance, if have n disks then disk j is used as the parity disk for block i if i%n = j.

RAID 6

- RAID 4 and 5 allow one to recover from single disk crashes.
- RAID 6 allows one to recover from multiple disk crashes.
- The idea is to divide the disks into some being data and some being redundant.
- We use the redundant disks to fill in the rest of a Hamming code word based on the data for that bit position in that block of the data disks.
- Such Hamming codes are error correcting and can correct for more than one disk errors.