Big Data Platforms (5 ECTS) DATA14003/AYDATA14003en Lecture 4

Keijo Heljanko

Department of Computer Science
University of Helsinki
keijo.heljanko@helsinki.fi

16.9-2021

Fault Tolerance Strategies for Storage

- Hard disks are a common cause of failures in large scale systems with field reports giving around 3% average yearly replacement rates
- There are many ways to make more reliable storage out of unreliable hard disks. Here we list some examples only, from smaller scale to larger scale:
    - RAID - Redundant array of independent (originally: inexpensive) disks. Commonly used levels for improving reliability: RAID 1, RAID 5, RAID 6, RAID 10. Used inside servers & specialized SAN/NAS hardware
    - Replication and/or erasure coding of data inside datacenter: Example: Three way default replication in HDFS, with data residing on at least two racks
    - Geographical replication across datacenters: Example: Amazon S3 storage service offers geographic replication

RAID - Redundant Array of Independent Disks

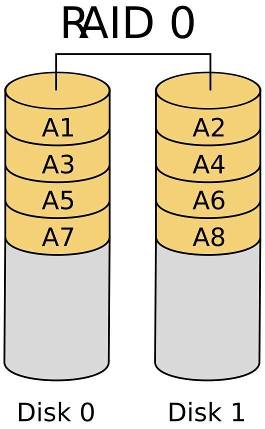
- Most commonly used fault tolerance mechanism in small scale
- RAID 0 : Striping data over many disks - No fault tolerance, improves performance
- Commonly used levels for fault tolerance:
  - RAID 1 : Mirroring - All data is stored on two hard disks
  - RAID 5 : Block-level striping with distributed parity
  - RAID 6 : Block-level striping with double distributed parity
  - RAID 10 : A nested RAID level: A stripe of mirrored disk pairs

RAID - Redundant Array of Independent Disks (cnt.)

- For additional info, see: `http://en.wikipedia.org/wiki/Standard_RAID_levels`, from where the RAID images in the following pages are obtained
- Terminology used: IOPS - I/O operations per second - the number of small random reads/writes per second

RAID 0 - Striping

- Stripes data over a large number of disks to improve sequential+random reads&writes
- Very bad choice for fault tolerance, only for scratch data that can be regenerated easily

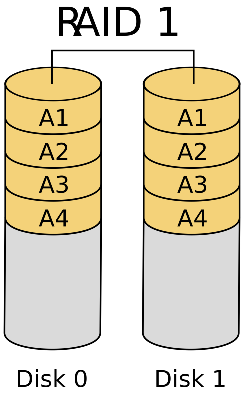


RAID 1 - Mirroring

- Each data block is written to two hard disks: first one is the master copy, and secondly a mirror slave copy is written after the master
- Reads are served by either one of the two hard disks
- Loses half the storage space and halves write bandwidth/IOPS compared to using two drives

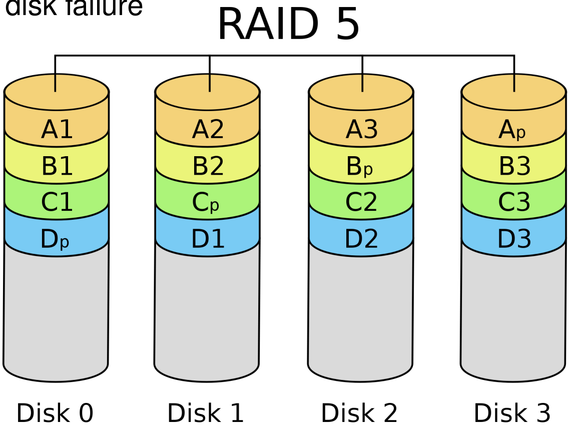
RAID 1 - Mirroring (cnt.)

- Data is available if either hard disk is available
- Easy repair: Replace the failed hard disk and copy all of the data over to the replacement drive



RAID 5 - Block-level striping with distributed parity

- Data is stored on $n + 1$ hard disks, where a parity checksum block is stored for each n blocks of data. The parity blocks are distributed over all disks. Tolerates one hard disk failure



RAID 5 - Properties

- Sequential reads and writes are striped over all disks
- Loses only one hard disk worth of storage to parity
- Sequential read and write speeds are good, random read IOPS are good
- Random small write requires reading one data block + parity block, and writing back modified data block + modified parity block, requiring 4 x IOPS in the worst case (battery backed up caches try to minimize this overhead.)

RAID 5 - Properties (cnt.)

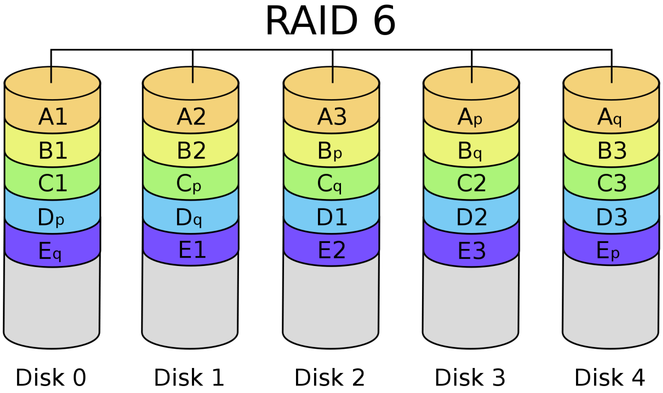
- Rebuilding a disk requires reading all of the contents of the other *n* disks to regenerate the missing disk using parity - this is a slow process
- Slow during rebuild: When one disk has failed, each missing data block read requires reading *n* blocks of data while rebuild is in progress
- Vulnerability to a second hard disk failure during array rebuild and long rebuild times make most vendors instead recommend RAID 6 (see two slides ahead) with large capacity SATA drives

RAID 5 - Properties (cnt.)

- “RAID 5 Write Hole”: A power failure during writes to a stripe (remember, $n+1$ disks need to be updated together in the worst case for large writes) can leave the array in a corrupted state - battery backed up memory caches (and Uninterruptible Power Supply “UPS”) are used to try to get rid of this problem
- In the write hole scenario the parity no longer matches the contents of the data disks. This can cause garbage to be written to a data disk if one of the data disks fails and the incorrect parity is used to regenerate its contents
- Also RAID 5 does not protect from a single drive with bad blocks from corrupting the array if a disk gives out bad data during parity calculations. Use of “hot spares” is recommended to speed up recovery start

RAID 6 - Block-level striping with double distributed parity

- Data is stored on $n + 2$ hard disks, where two different parity checksum blocks are stored for each n blocks of data. Tolerates two hard disk failures



RAID 6 - Properties

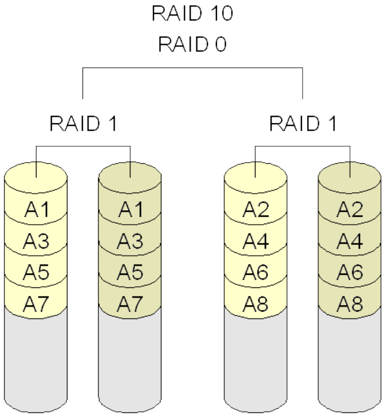
- Sequential reads and writes are striped over all disks
- Loses two hard disk worth of storage to parity
- Sequential read and write speeds are good, random read IOPS are good
- Random small write requires reading one data block + two parity blocks, and writing back modified data block + two modified parity blocks, requiring 6 x IOPS in the worst case (battery backed up caches try to minimize this overhead)
- Slow during rebuild: When 1-2 disks have failed, each missing data block read requires reading *n* blocks of data while rebuild is in progress

RAID 6 - Properties (cnt.)

- Can tolerate one additional disk failure during rebuild, contents of any two missing disks can be rebuilt using the remaining n disks
- “RAID 6 Write Hole”: A power failure during writes to a stripe (remember, $n + 2$ disks need to be updated together in the worst case for large writes) can leave the array in a corrupted state - battery backed up memory caches (and UPS) are used to try to get rid of this problem
- In two disk failures: RAID 6 does not protect from a single drive with bad blocks from corrupting the array if a disk gives out bad data during parity calculations

RAID 10 - Stripe of Mirrors

- Data is stored on 2*n* hard disks, each mirror pair consists of two hard disks and the pairs are striped together to a single logical device (see: `http://en.wikipedia.org/wiki/Nested_RAID_levels`).



RAID 10 - Properties

- Tolerates only one hard disk failure in the worst case (*n* - one disk from each mirror pair - in the best case)
- Loses half of the storage space
- Sequential reads and writes are striped over all disks
- Sequential read and write speeds are good, random read and write IOPS are good

RAID 10 - Properties (cnt.)

- Each data block is written to two hard disks. Random small write require 2 x IOPS in the worst case
- Quicker during rebuild: Missing data is just copied over from the other disk of the mirror. Use of “hot spares” recommended to speed up recovery start
- Can not tolerate the failure of both disks in the same mirror

RAID 10 - Properties (cnt.)

- On unclean shutdown missing changes should be copied from the first master drive to the second slave drive - No “Write hole problem” if such ordering can be guaranteed
- If there is no ordering of writes, it is not easy to figure out which hard drive has the most recent version of the data. Using an UPS on RAID 10 server is still a very good idea
- RAID 10 does not protect from a single drive with bad blocks from corrupting the array during rebuild

RAID Usage Scenarios

- RAID 0: Temporary space for data that can be discarded
- RAID 1: Small scale installations, server boot disks, etc.
- RAID 5: Some fault tolerance with minimum storage overhead, small random writes can be problematic, RAID 5 Write Hole problem without special hardware (battery backed up cache / UPS)

RAID Usage Scenarios (cnt.)

- RAID 6: Fairly good fault tolerance with reasonable storage overhead, small random writes can be even more problematic than in RAID 5, RAID 6 Write Hole problem without special hardware (battery backed up cache / UPS)
- RAID 10: Less fault tolerant than RAID 6 but more fault tolerant than RAID 5. Loses half of the storage capacity. Good small random write performance makes this often the choice for database workloads. Can avoid the “Write hole problem” under write ordering guarantees.

Remember: RAID is no substitute for backups!

RAID Hardware Issues

- RAID 1 and RAID 10 can be made safe with proper software configuration to use without special hardware support such as battery backed up caches and UPSs. We still recommend using UPS
- RAID 5 and RAID 6 are more storage efficient than RAID 10 but require specialized hardware to protect against corruption due to unclean shutdown (battery backed up cache / UPS)

More on RAID Write Hole Problem

- RAID 5 and RAID 6 need to atomically update all the drives in a stripe to remain consistent
- We know a way to do this from the world of Databases: Atomic Transactions
- How are transactions implemented in databases? By using a persistent log of modifications to be performed and replaying any missing modifications in case of a shutdown in the middle of an atomic sequence of modifications
- Thus RAID 5 and RAID 6 need a persistent transaction log in order to offer atomic transactions for updating a stripe

More on RAID Write Hole Problem (cnt.)

- Battery backed up cache is one way to implement a persistent transaction log
- Another way is to implement a log structured filesystem that is aware of the underlying storage subsystem: Example: Sun (Oracle) ZFS: `http://en.wikipedia.org/wiki/ZFS`
- ZFS implements RAID 5 and RAID 6 equivalent fault tolerance without specialized hardware
- It is a filesystem which is given full control over the raw disk devices and uses a copy-on-write transactional object model to achieve atomicity of updates
- ZFS also has data integrity checksums
- `Btrfs` is a filesystem for Linux with similar features

Scaling Up Storage

There are quite a few problems when scaling up storage to thousands of hard disks:

- One can buy large NAS/SAN systems handling thousands of hard disks.
    - Scaling up vs. scaling out. High end NAS/SAN are not a commodity
    - Their pricing is very high compared to storage capacity
    - The performance is often not as good as with a scale out solution
    - Often still the best solution for modest storage needs

Scaling Out Storage

There are also quite a few problems when scaling out storage to thousands of hard disks over hundreds of small servers:

- Using standard hardware RAID will handle hard disk failures but does not protect against storage server/RAID controller failures
- When having hundreds of servers and RAID controllers, failures are going to be inevitable
- To protect against data unavailability during server / RAID controller repair, data must be replicated on several servers

Scaling Out Storage

- If we need to replicate data over several servers anyhow, why spend money on specialized HW RAID controllers?
- We just need the software to create a “distributed fault tolerant storage” subsystem
- HDFS is a prime example of such a system

HDFS Design Decisions

- HDFS replication on a block level $\approx$ RAID 10 (+ data integrity sub-block CRC-32 checksums)
- For small installations replicating each block twice in HDFS is enough, and we have exactly RAID 10 storage layout in HDFS
- For large installations each HDFS block is replicated three times, also Linux RAID 10 can be configured to use a three hard disks per mirror set, matching HDFS default replication, and allowing two hard disk failures per mirror set

What about RAID 6 for HDFS?

- RAID 6 can improve the storage efficiency over RAID 10 significantly, as only two disks are lost to parity, and stripes can be made quite wide
- Even worse, HDFS has default three way replication that has a 3x storage overhead

Erasure Codes

- Many RAID 6 implementations use an erasure code, for example a Reed-Solomon code to implement the parity calculations and recovery of any two missing disks
- An Erasure code can be extended to recover any number of missing hard disks. Basically this is a so called forward error correction code
- Reed-Solomon codes are also used in CD error correction

See also:
`http://en.wikipedia.org/wiki/Erasure_code` and `http://en.wikipedia.org/wiki/Reed%E2%80%93Solomon_error_correction`

Erasure Codes in Storage

- The Windows Azure Storage uses erasure coding: Cheng Huang et al.: Erasure Coding in Windows Azure Storage, Proceedings of the 2012 USENIX conference on Annual Technical Conference, USENIX ATC'12.
  `https://www.usenix.org/system/files/conference/atc12/atc12-final181_0.pdf`
- Example: An erasure code can be devised that stores 10 data blocks and 4 parity blocks (1.4x storage overhead), and this tolerates four disk failures
- See `https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HDFSErasureCoding.html` for HDFS-EC erasure code implementation released in HDFS 3.1

HDFS-EC

- Basically implements an erasure encoding of data: RAID 6 style but more parity blocks allowing for more simultaneous hard disk failures
- Exactly the same efficiency during rebuild (RAID 10) vs. storage efficiency (RAID 6) considerations apply when deciding whether to use erasure codes at scale
- In HDFS-EC the erasure coding / replication can be configured independently for each directory in the HDFS

Avoiding the RAID Write Hole Problem

- One should avoid “update in place”, for example by using the write-once-read-many (WORM) pattern to never modify data blocks once written
- In general the only hope to avoid the RAID 6 hole problem is to have database style transactions built into RAID 6 implementation: Either in the hardware RAID controller or in a filesystem that does database style transaction logging
- One nice thing about a centralized HDFS NameNode is that it can be used as the central place to do atomic transactions on the HDFS filesystem state that are logged using database transaction logs

Avoiding RAID Hole Problem (cnt.)

- More examples: ZFS and btrfs filesystems: As HDFS they also never updates data in place in disk arrays but always do a fresh copy of it before modifications
- Jim Gray: “Update in place is a poison apple”, from: Jim Gray: *The Transaction Concept: Virtues and Limitations* (Invited Paper) VLDB 1981: 144-154.
- Bottom line: Basically to do reliable distributed updates, one must adopt database techniques (transaction logging) to implement atomic updates of the RAID stripe reliably
- HDFS tries to minimize the amount of transactions by only doing them at file close time. No transactions for stripe updates are needed because stripe updates are disallowed!

Hard Disk Read Errors

- Unrecoverable read errors (UREs) are quite common in hard disks.
- Basically if an URE happens, the hard disk gives out an error saying that it can not read a particular data block stored on the hard disk
- Consumer hard disks are only specified to offer URE rates of at most one error in 10^{15} bits read
- Thus the vendors are only promising that on average hard disks are not able to recover the written data on at most once per 10^{15} bits of data read
- Some enterprise hard disks can be specified upto URE rates of at most one error in 10^{16} bits read

Hard Disk Read Errors

- Large storage systems read way more than 10^{15} bits of data during their lifetime
- Thus additional checksums and error correction are needed at scale
- Examples: HDFS stores a CRC32 checksum per each 512 bytes of data (the smallest random read from HDFS).
- ZFS employs full filesystem checksums, as does Linux btrfs
- A good overview article on the RAID 6 and beyond is: “Adam Leventhal: *Triple-Parity RAID and Beyond*. ACM Queue 7(11): 30 (2009)”