CS-E4640 - Big Data Platforms
Lecture 5

Keijo Heljanko

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

10.10-2018

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: In HDFS-RAID 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 `http://wiki.apache.org/hadoop/HDFS-RAID` for an erasure code implementation for HDFS employed at Facebook

Erasure Codes in Storage

- See also an approach with better performance under failures:
  Maheswaran Sathiamoorthy, Megasthenis Asteris, Dimitris S. Papailiopoulos, Alexandros G. Dimakis, Ramkumar Vadali, Scott Chen, Dhruba Borthakur: XORing Elephants: Novel Erasure Codes for Big Data. PVLDB 6(5): 325-336 (2013)
  `http://www.vldb.org/pvldb/vol6/p325-sathiamoorthy.pdf`
- Xorbas Hadoop project Web page:
  `http://smahesh.com/HadoopUSC/`

HDFS-RAID

- Basically implements an erasure encoding of data: RAID 6 style but with larger stripes and more parity blocks allowing for more simultaneous hard disk failures
- Also file block allocation policy needs to be changed to never allocate blocks of the same stripe on the same physical hardware
- HDFS-RAID requires very large files to get a full stripe of data with the large HDFS block size
- Exactly the same efficiency during rebuild (RAID 10) vs. storage efficiency (RAID 6) considerations apply when deciding whether to use erasure codes at scale
- Another Erasure coding approach is represented in HDFS-7285 under the name “Native HDFS erasure coding”

Avoiding RAID 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

Hard Disk Read Errors Discussion Pointers

- For a thought provoking discussion start on the topic, see the provocative blog post:
  `http://www.zdnet.com/blog/storage/why-raid-5-stops-working-in-2009/162`
- A good overview article on the RAID 6 and beyond is: "Adam Leventhal: *Triple-Parity RAID and Beyond*. ACM Queue 7(11): 30 (2009)"

Database ACID guarantees

A traditional (centralized) database can guarantee its client ACID (atomicity, consistency, isolation, durability) properties:

- Atomicity: Database modifications must follow an ”all or nothing” rule. Each transaction is said to be atomic. If one part of the transaction fails, the entire transaction fails and the database state is left unchanged.
- Consistency (as defined in databases): Any transaction the database performs will take it from one consistent state to another.
- Isolation: No transaction should be able to interfere with another transaction at all.
- Durability: Once a transaction has been committed, it will remain so.

Distributed Databases

In cloud computing we have to implement distributed databases in order to scale to a large number of users sharing common system state.
We define the three general properties of distributed systems popularized by Eric Brewer:

- Consistency (as defined by Brewer): All nodes have a consistent view of the contents of the (distributed) database
- Availability: A guarantee that every database request eventually receives a response about whether it was successful or whether it failed
- Partition Tolerance: The system continues to operate despite arbitrary message loss

Brewer's CAP Theorem

- In a PODC 2000 conference invited talk Eric Brewer made a conjecture that it is impossible to create a distributed system that is at the same time satisfies all three CAP properties:
  - Consistency
  - Availability
  - Partition tolerance
- This conjecture was proved to be a Theorem in the paper: "Seth Gilbert and Nancy A. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News, 33(2):51-59, 2002."

Brewer's CAP Theorem

Because it is impossible to have all three, you have to choose between two of CAP:

- CA: Consistent & Available but not Partition tolerant
  - A non-distributed (centralized) database system
- CP: Consistent & Partition Tolerant but not Available
  - A distributed database that can not be modified when network splits to partitions
- AP: Available & Partition Tolerant but not Consistent
  - A distributed database that can become inconsistent when network splits into partitions

Example CA Systems

- Single-site (non-distributed) databases
- LDAP - Lightweight Directory Access Protocol (for user authentication)
- NFS - Network File System
- Centralized version control - Svn
- HDFS Namenode

These systems are often based on two-phase commit algorithms, or cache invalidation algorithms.

Example CP Systems

- Distributed databases - Example: Google Bigtable, Apache HBase
- Distributed coordination systems - Example: Google Chubby, Apache Zookeeper

The systems often use a master copy location for data modifications combined with pessimistic locking or majority (aka quorum) algorithms such as Paxos by Leslie Lamport.

Example AP Systems

- Filesystems allowing updates while disconnected from the main server such as AFS and Coda.
- Web caching
- DNS - Domain Name System
- Distributed version control system - Git
- “Eventually Consistent” Datastores - Amazon Dynamo, Apache Cassandra

The employed mechanisms include cache expiration times and leases. Sometimes intelligent merge mechanisms exists for automatically merging independent updates.

System Tradeoffs - CA Systems

- Limited scalability - use when expected system load is low
- Easiest to implement
- Easy to program against
- High latency if clients are not geographically close

System Tradeoffs - CP Systems

- Good scalability
- Relatively easy to program against
- Data consistency achieved in a scalable way
- High latency of updates
- Quorum algorithms (e.g., Paxos) can make the system available if a majority of the system components are connected to each other
- Will eventually result in user visible unavailability for updates after network partition

System Tradeoffs - AP Systems

- Extremely good scalability
- Very difficult to program against - There is no single algorithm to merge inconsistent updates. Analogy: In distributed version control systems there will always be some conflicting updates that can not both be done, and there is no single general method to intelligently choose which one of the conflicting updates should be applied, if any.

System Tradeoffs - AP Systems (cnt.)

- Data consistency sacrificed - Can sometimes be tolerated, see e.g., Web page caching
- Low latency updates possible, always update the “closest copy”, and start to propagate changes in the background to other data copies
- Will eventually result in user visible inconsistency of data after network partition

Why AP Systems?

- Much of the Internet infrastructure is AP - Web caching, DNS, etc.
- Works best with data that does not change, or changes infrequently
- Often much simpler to implement in a scalable fashion as CP systems, e.g., just cache data with a timeout
- Needs application specific code to handle inconsistent updates, no generic way to handle this!
- Needed for low latency applications

Example: Amazon Web Store

Most systems are combinations of available and consistent subsystems:

- Shopping Basket: AP - Always available for updates, even during network partitions. In case of two conflicting shopping basket contents - merge takes the union of the two shopping baskets
- In addition CP system for billing and for maintaining master copy of inventory
- Inconsistencies between the two systems are manifested at order confirmation time - items might need to be backordered / cancelled
- Quite often CP system added latency can be hidden in another long latency operation such as emailing the user

Example: Google Gmail

- AP when marking messages read - needs to be available, easy algorithm to merge inconsistent copies (take the union of read messages)
- CP when sending emails - Email will either be sent or not, and this needs to be acknowledged to the user

Example: Hadoop based Web Application

- HDFS Namenode - CA system, a centralized database of filesystem namespace, easy implementation effort
- HBase - CP distributed database, will disallow modifications under network partition, uses Apache Zookeeper for coordination
- Additional (optional) Web Caching layer - AP system, Used to minimize the request load on the background database system

Combinations of AP and CP Systems

- AP system usually used for user interaction due to its better scalability (cheaper to run) and lower latency. Requires application specific code to merge inconsistent updates
- CP system used to store “the master copy” of data, has large latency that needs to be hidden from the user
- Also CA systems used - Often the simplest implementation, needs careful architecture design not to become the bottleneck