Storage

Egill Friðriksson Gard Aasness Maximilian Georg Kurzawski

1 Motivation

One of the most commonly heard words in the tech industry these days is "big data", which refers to the massive amounts of data being stored and processed by different companies and agencies worldwide. Big data involves gathering, storing and analysing data to, for example, identify trends and offer better services. The industry is however faced with an important challenge: how is it possible to effectively store all this data? As the amount of data keeps increasing, it becomes excessive for a single node to handle. This is solved with distributed databases, i.e. data gathered is spread across multiple nodes in a network. This allows for the storage of large amount of data.

There are many ways to implement a distributed database, depending on which features the database should provide. The CAP theorem, which states that a database can at most offer two of the following features: consistency, availability and partition tolerance, is highly relevant as it enables them to be tailored to specific use cases. To be considered a distributed database, it has to offer partition tolerance, meaning that it should be able to operate normally during occurrences of network partitions. That leaves the option of either choosing availability or consistency as the second feature, as one will rule out the other. To achieve these features, there are certain challenges that needs to be addressed, such as fault tolerance, scalability and performance.

We will discuss these challenges further in regards to how some of the most known distributed databases have chosen to deal with them.

2 Contributions

For the first contribution, Amazon's Dynamo is studied. Dynamo is a highly available and scalable data store, used by a number of Amazon.com's core services. Since even slightest outages have significant financial consequences for Amazon, reliability of their storage system is the most important factor. Dynamo differs from popular decentralized storage systems since Dynamo is targeted mainly at applications that need an "always writeable" data store where no updates are rejected due to failures or concurrent writes. One of Dynamo's main goals is that it requires at least 99.9 percent of read and write operations to be performed within a few hundred milliseconds. This is because it is built for latency sensitive applications. Dynamo serves a diverse set of applications and it can be configured to achieve the desired level of performance for each one.

Similar to Dynamo, Cassandra is also a distributed database which offers partition tolerance and availability as the two main features. It was originally developed by Facebook to handle inbox searching, which refers to the possibility of users to search through their messages, and as the largest social network platform, it needed to be scalable and reliable with a high performance to support growth and handle a high write throughput. Seeing how Cassandra was built to be responsible for a massive amount of data spread over servers in data centers all over the world, it was designed to prioritize availability by treating failures as a normal case happening all the time, instead of viewing it as an exception.

Another solution to consider is the Flat Datacenter Storage (FDS), which is a simple, shared and centralized model of storage. It is high-performance, fault tolerant, large-scale, locality oblivious system that stores big binary large objects (blobs). Its strengths lie in its ability to expose the full bandwidth of its disks to all clients and servers uniformly. FDS is used for processing big data where thousands of computers process data in parallel.

Bigtable is another alternative. The main contributions of Bigtable include providing a flexible, high-performance solution for a wide range of Google products such as web indexing, Google Earth, and

Google Finance, despite their varying data demands. It introduces a unique data model that allows dynamic control over data layout and format, emphasizing column-oriented storage and the use of row and column names as arbitrary strings for data indexing. Additionally, Bigtable offers strong consistency for single-row transactions and eventual consistency for multi-row transactions, empowering developers to tailor the level of consistency to their specific use cases.

Lastly, the contributions of The Google File system are inspected. GFS gave Google's enormous data centers a scalable and fault-tolerant storage solution. GFS developed a more straightforward and effective data storage strategy that prioritizes large files over smaller files. This design decision decreased metadata overhead and enhanced performance as a whole. By automatically replicating data across numerous servers, GFS also addressed the issue of data reliability, ensuring data availability even in the event of hardware failures.

3 Solution

Now that some of the main problems of storing big data have been explored, the different solutions are examined.

Amazon's Dynamo achieves scalability and availability with consistent hashing and object versioning. Dynamo is therefore designed to sacrifice consistency (under certain failure scenarios) for availability to provide an "always-on" experience. For example, Amazon's Shopping Cart Service should always allow the customer to add an item to their cart even though the most recent state of the cart is unavailable. Dynamo allows multiple versions of the object (i.e. cart) to exist and detects parallel versions using vector clocks. Eventually, the object is reconciled by the client. The system prioritizes the addition of new items so therefore "add to cart" always works but deleted items may resurface. In order to achieve 99.9 percent of read and write operations to be carried out within 300 milliseconds, the system must sacrifice performance, cost efficiency, availability and durability guarantees. Techniques such as the load balanced selection of write coordinators are use to reach this level of performance. The main advantage of Dynamo is that its client applications can adjust three key values to reach optimal performance. These values are the number of hosts keeping replications of each item (N), the minimum number of nodes that must participate in a successful read and write operations (R and W).

On the other hand, Cassandra offers availability and partition tolerance through features such as replication, fault tolerance, scalability and partitioning. Cassandra's ability to scale is achieved by applying consistent hashing to dynamically partition the data across nodes in the cluster. During the consistent hashing, coordinators that are responsible for sets of nodes are chosen. Replication across multiple data centers allows Cassandra to handle data center failures without any loss of data, achieving fault tolerance. This is because it replicates each row across multiple data centers. The coordinators chosen in the consistent hashing are responsible for the replication, and the application policy decides which of these replication policies will be used: *Rack Unaware*, *Rack Aware* or *Datacenter Aware*.

One of FDS's main advantages is the fact that it utilizes full bisection bandwidth and uses the available throughput and latency budget of every disk in a cluster. Since disks communicate at their full bandwidth, recovery from disk failures considerably faster than many other systems. For recovery, FDS uses metadata servers that detect tractserver (server that stores read and write operations) timeouts using heartbeat messages. It then declares the tractserver dead and invalidates the tract list table (TLT) and picks random tractservers to fill in the empty spaces. Then it sends updated TLT assignments to every server affected and waits for each tractserver to acknowledge the new TLT assignment. For availability and fault tolerance, FDS uses replication. In FDS there are no local disks yet it achieves cluster-wide I/O performance on par with systems that exploit locality.

When designing Bigtable, the authors (Google) tackle the addressed problems by designing Bigtable as a distributed, persistent, and sparse multidimensional sorted map indexed by row key, column key, and timestamp. This design choice enables atomic operations for reads and writes under a single row key, regardless of the number of columns involved. Furthermore, Bigtable supports multiple versions of cell data indexed by timestamps, allowing users to manage collisions and store data efficiently. The paper emphasizes Bigtable's scalability, high performance, and high availability, showcasing its adaptability and effectiveness in managing Google's extensive data resources. The ongoing development of features like secondary indices and cross-data-center replication further enhances its capabilities, making Bigtable a crucial component of Google's data infrastructure.

Another example is how GFS solved the aforementioned challenges. GFS distributes files across numerous storage servers by dividing them into fixed-size chunks. It keeps track of metadata and manages a master server to coordinate file activities like reads and writes. To enable fault tolerance, data is replicated across various servers. Because of the system's streamlined user interface, it is simpler for programmers to create scalable and dependable applications on top of GFS. As a result of offering a scalable, fault-tolerant, and effective file system specifically designed to meet Google's enormous data processing needs, GFS revolutionized distributed storage.

4 Discussion

It is clear that the solutions tackle the same set of problems but depend on the specific requirements and use cases of the applications.

Although not perfect, it is clear that Dynamo has yielded satisfactory results for Amazon services, providing the desired levels of availability and performance and has been successful in handling server failures, data center failures and network partitions. As mentioned before, the trade-offs for maintaining such a stringent latency requirements and high availability are performance, cost efficiency and availability and guarantee guarantees.

Cassandra has specialized in dealing with failures and thus always being available. As a result to these priorities, according to the CAP theorem, Cassandra sacrifices consistency. Therefore, Cassandra might not be the best option if an application does not have massive amounts of data and requires reliable and accurate data. However, for big data situations where data has to be distributed (resulting in regular node failures) and also requires the possibility of scaling with time, Cassandra is a good fit.

FDS performs considerably better than other storage systems when it comes to recovering from failed disks, a task that takes hours reduced to a couple of seconds. It also allows for large matrix operations, sorts, joins and comparisons that were previously off-limits to programmers. FDS therefore paves the way for new types of applications and science. FDS's design is based on a CLOS network and due to this assumption the authors consider accessing remote disks as fast as accessing local disks. If this is not the case, the FDS will not perform as optimally as described in the literature.

Bigtable aligns itself more with availability and partition tolerance while offering developers the flexibility to choose between strong consistency and eventual consistency based on their specific use cases.

The Google File System (GFS) primarily focuses on ensuring high availability and partition tolerance while affording developers the flexibility to select their desired level of consistency, whether it's strong or eventual, based on their specific application requirements.

5 Conclusion

In conclusion, the five distributed database solutions we've examined: Dynamo, Cassandra, Flat Datacenter Storage (FDS), Bigtable, and the Google File System (GFS) each present innovative approaches to addressing the challenges of scalability, availability, fault tolerance, and performance in distributed computing. Dynamo and Cassandra focus on availability and high write throughput, while Bigtable offers flexibility and strong consistency options. FDS and GFS are tailored for processing large-scale data with a focus on performance and fault tolerance. Each solution has its strengths and trade-offs, making them suitable for different scenarios within the realm of distributed databases.