1. Summary

Amazon's e-commerce business had grown to a massive level by the mid 2000s, and one of the main problems it faced as it continued to grow was how to scale up their data processing and storage reliably. With several critical e-commerce transactions occurring each second, even a minute's failure of a server could lead to large financial losses. Amazon developed Dynamo, a highly available key-value data store, in order to provide a way to store data reliably, so as to give it an appearance of being always on.

Dynamo was a NoSQL database, and it was formed as a combination of various algorithms. It proved to be the first successful NoSQL database working on scale and acted as a precursor to several other NoSQL databases. Dynamo was used internally by Amazon, and later AWS developed DynamoDB which was available to use by customers as a service provided by AWS.

Some of the considerations made by Amazon while building Dynamo were - 1. They observed that a majority of the operations being run by the engineers was of key-value type, where only a single value was returned against a corresponding unique ID value. As relational operations such as JOIN are computationally expensive, the engineers decided to model Dynamo as a NoSQL database. 2. Amazon chose to prioritise availability and fault tolerance over consistency for Dynamo. They observed that the system not being able to return results would cause a large amount of loss, however a system which is not immediately consistent (but eventually becomes consistent) would not impact the sales that much. 3. The team predicted more growth for Amazon, and hence also focused on making the system infinitely scalable. As they weren't stringent on immediate consistency, it allowed Dynamo to easily make replicas of the data, which allowed it to scale more easily without affecting its performance.

The Dynamo architecture consisted of multiple techniques and algorithms which worked together to meet these requirements as a scalable database solution. Broadly speaking, the techniques used were - 1. Consistent hashing for partitioning the data. 2. Vector clocks with reconciliation during reads to enable high availability during writes 3. Sloppy Quorum and hinted handoff for handling temporary failures. 4. Merkle trees to recover from permanent failures. 5. Gossip-based membership protocol and failure detection.

2. Strengths

The following are the main advantages of using Dynamo - 1. By using consistent hashing for partitioning data, it becomes much more easier to scale out the system. Addition of anew node would not lead to redistribution of data over all partitions. 2. By implementing quorums, the data read or write can be validated as correct as it ensures that it is replicated over the required number of servers. This ensures high availability and durability even if some servers are unavailable. 3. Vector clocks allow version numbers to be decoupled from update rates, this means that multiple updates can be written to the database simultaneously without having to worry about version number conflicts. 4. Dynamo has a very simple interface, making it largely user friendly. Its main two functions are the get() and put() functions.

3. Unclear Aspects

The following aspects remain unclear with respect to Dynamo - 1. The paper does not describe in detail how the various techniques are able to interact with each other. 2. The paper does not go into detail how each individual technique works, how they were integrated into the system, and what were their advantages compared to any other alternatives. 3. Unclear about the amount of overhead taken by all these techniques.

4. Limitations

The following are the limitations and areas of improvement for Dynamo - 1. 2. 3.