The paper gives a detailed introduction to the Amazon’s Dynamo storage system. The Dynamo storage system is a distributed key-value store for highly available, low-latency access to relatively small data objects which size is usually smaller than 1MB.

Similar to the Facebook’s Cassandra, Dynamo treats the failures as norm rather than exception as well. However, Dynamo sacrifices the consistency for some failure cases to maintain the “always-on” experience. It uses replication for fault tolerance and allows customization of the conflict resolution strategy employed by client applications. Dynamo has a Distributed Hash Table network design, uses vector clocks for reconciliation, and gossip-based membership.

The end goal for the Amazon developers was to not only provide a highly-available key-value store service, but also allow configuration of tradeoffs between consistency, availability, and performance. Amazon engineers observed that different applications have different requirements, so they should be able to configure the system to best suit their needs. By taking a divide and conquer approach to the problem, the engineers were able to exploit pre-existing solutions to particular sub-problems to simplify their task.

Since Dynamo is focused on write availability rather than read availability. So even in presence of network partitioning and server faults, Dynamo can still receive updates by the client. To keep the high availability, the responsibility of resolving version conflict is delegated to application itself by Dynamo is impressive. The database system excludes the corner cases on the philosophy that worst-case complexity usually comes from a few limited cases and hence it guarantees services to 99.9% of assignments which is really an interesting and strong point of DynamoDB. This forces the designers to push the complexity of conflict resolution to the reader such that no write is rejected.  Dynamo provides eventual consistency and thus does not guarantee strong consistency of the data returned by the client operations. It uses Vector clock-based timestamping of the data objects. If it finds multiple replicas of the same object, then it tries to resolve the conflict by causality information based on vector timestamp. If that does not succeed, the client needs to resolve the conflict semantically.

Then, in section four, the paper introduces the Dynamo’s architecture. To scale incrementally, Dynamo has to dynamically partition the data over the node sets. For partitioning, if a node becomes unavailable, the load handled by this node is evenly dispersed across the remaining available nodes. When a node becomes available again, or a new node is added to the system, the newly available node accepts a roughly equivalent amount of load from each of the other available nodes. The number of virtual nodes that a node is responsible can decide based on its capacity, accounting for heterogeneity in the physical infrastructure.

In a word, Dynamo is a successful storage system that can provide “always on” experience to some of Amazon’s core services. However, there are still some weak parts of the Dynamo storage system. A big limitation of DynamoDB is the lack of multiple indices, which is a common problem in non-relational databases. Scanning has a really high time complexity in DynamoDB, which is great for lookups by key, not so good for queries, and abysmal for queries with multiple predicates. Besides, although DynamoDB supports transactions, it is not in the traditional SQL sense. Each write operation is atomic to an item. A write operation either successfully updates all of the item's attributes or none of its attributes. There are no multi-operation transactions.