Dynamo: Introduction

Let’s explore Dynamo and its use cases.

We'll cover the following

Goal#

Design a **distributed key-value store** that is highly available (i.e., reliable), highly scalable, and completely decentralized.

What is Dynamo?#

Dynamo is a **highly available key-value store** developed by Amazon for their internal use. Many Amazon services, such as shopping cart, bestseller lists, sales rank, product catalog, etc., need only primary-key access to data. A multi-table relational database system would be an overkill for such services and would also limit scalability and availability. Dynamo provides a flexible design to let applications choose their desired level of availability and consistency.

Background#

Dynamo – not to be confused with DynamoDB, which was inspired by Dynamo's design – is a distributed key-value storage system that provides an **"always-on"** (or highly available) experience at a massive scale. In CAP theorem (https://www.educative.io/collection/page/5668639101419520/5559029852536832/5998984290631680) terms, Dynamo falls within the category of **AP systems** (*i.e., available and partition tolerant*) and is designed for **high availability and partition tolerance at the expense of strong consistency**. The primary motivation for designing Dynamo as a highly available system was the observation that the availability of a system directly correlates to the number of customers served. Therefore, the main goal is that the system, even when it is imperfect, should be available to the customer as it brings more customer satisfaction. On the other hand, inconsistencies can be resolved in the background, and most of the time they will not be noticeable by the customer. Derived from this core principle, Dynamo is aggressively optimized for availability.

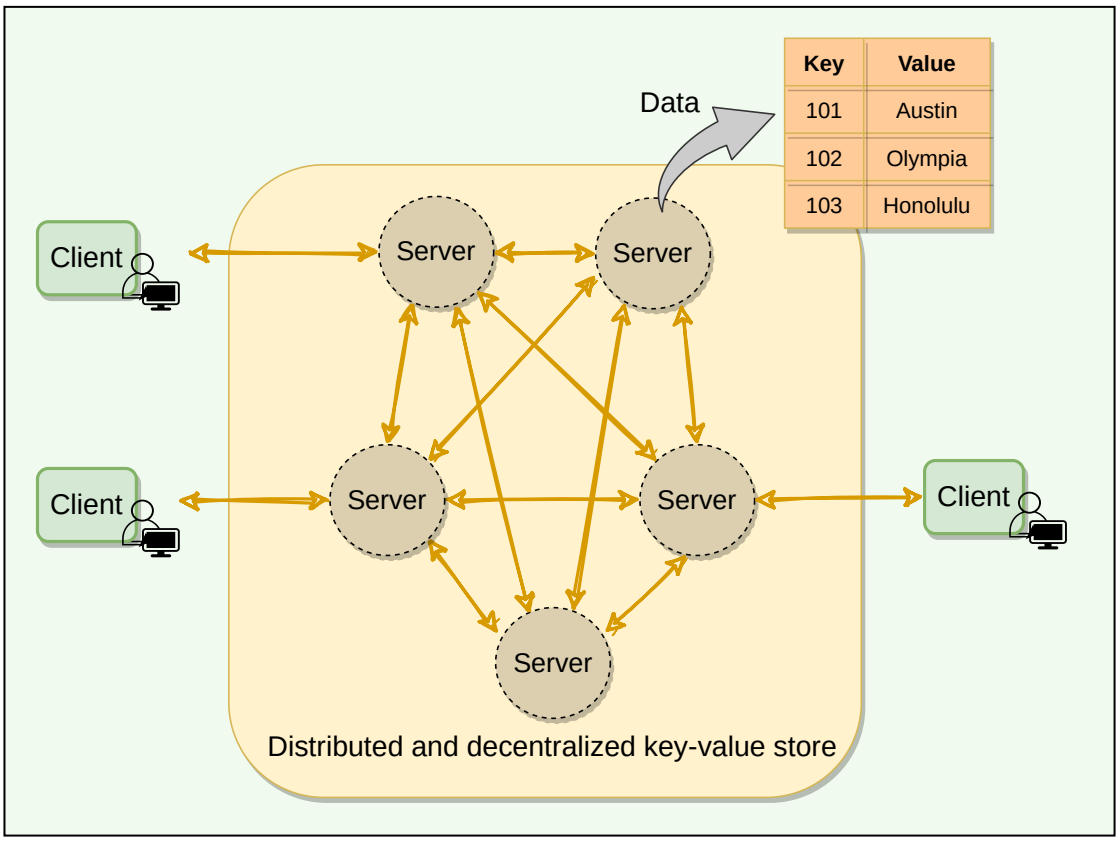
The Dynamo design was highly influential as it inspired many NoSQL databases, like Cassandra (https://cassandra.apache.org/), Riak (https://riak.com/), and Voldemort (http://www.project-voldemort.com/voldemort/) – not to mention Amazon's own DynamoDB (https://aws.amazon.com/dynamodb/).

Design goals#

As stated above, the main goal of Dynamo is to be **highly available**. Here is the summary of its other design goals:

- **Scalable:** The system should be ***highly scalable***. We should be able to throw a machine into the system to see proportional improvement.

- **Decentralized:** To avoid single points of failure and performance bottlenecks, there should not be any central/leader process.
- **Eventually Consistent:** Data can be ***optimistically replicated*** to become eventually consistent. This means that instead of incurring write-time costs to ensure data correctness throughout the system (*i.e., strong consistency*), inconsistencies can be resolved at some other time (*e.g., during reads*). Eventual consistency is used to achieve high availability.



High-level view of a distributed key-value store

Dynamo's use cases#

By default, **Dynamo is an eventually consistent database**. Therefore, any application where strong consistency is not a concern can utilize Dynamo. Though Dynamo can support strong consistency, it comes with a performance impact. Hence, if strong consistency is a requirement for an application, then Dynamo might not be a good option.

Dynamo is used at Amazon to manage services that have very high-reliability requirements and need tight control over the trade-offs between **availability**, **consistency**, **cost-effectiveness**, and **performance**. Amazon's platform has a very diverse set of applications with different storage requirements. Many applications chose Dynamo because of its flexibility for selecting the appropriate trade-offs to achieve high availability and guaranteed performance in the most cost-effective manner.

Many services on Amazon's platform require only primary-key access to a data store. For such services, the common pattern of using a relational database would lead to inefficiencies and limit scalability and availability. Dynamo provides a simple primary-key only interface to meet the requirements of these applications.

System APIs#

The Dynamo clients use `put()` and `get()` operations to write and read data corresponding to a specified key. This key uniquely identifies an object.

- **`get(key)`:** The `get` operation finds the nodes where the object associated with the given `key` is located and returns either a single object or a list of objects with conflicting versions along with a `context`. The `context` contains encoded metadata about the object that is meaningless to the caller and includes information such as the version of the object (more on this below).

- **`put(key, context, object)`:** The `put` operation finds the nodes where the object associated with the given `key` should be stored and writes the given `object` to the disk. The `context` is a value that is returned with a `get` operation and then sent back with the `put` operation. The `context` is always stored along with the object and is used like a cookie to verify the validity of the object supplied in the `put` request.

Dynamo treats both the object and the key as an arbitrary array of bytes (*typically less than 1 MB*). It applies the MD5 hashing algorithm on the key to generate a 128-bit identifier which is used to determine the storage nodes that are responsible for serving the key.

← Back

Designing Ticketmaster

Next →

High-level Architecture

Mark as Completed

Report an Issue