



Replication

Let's learn how Dynamo replicates its data and handles temporary failures through replication.

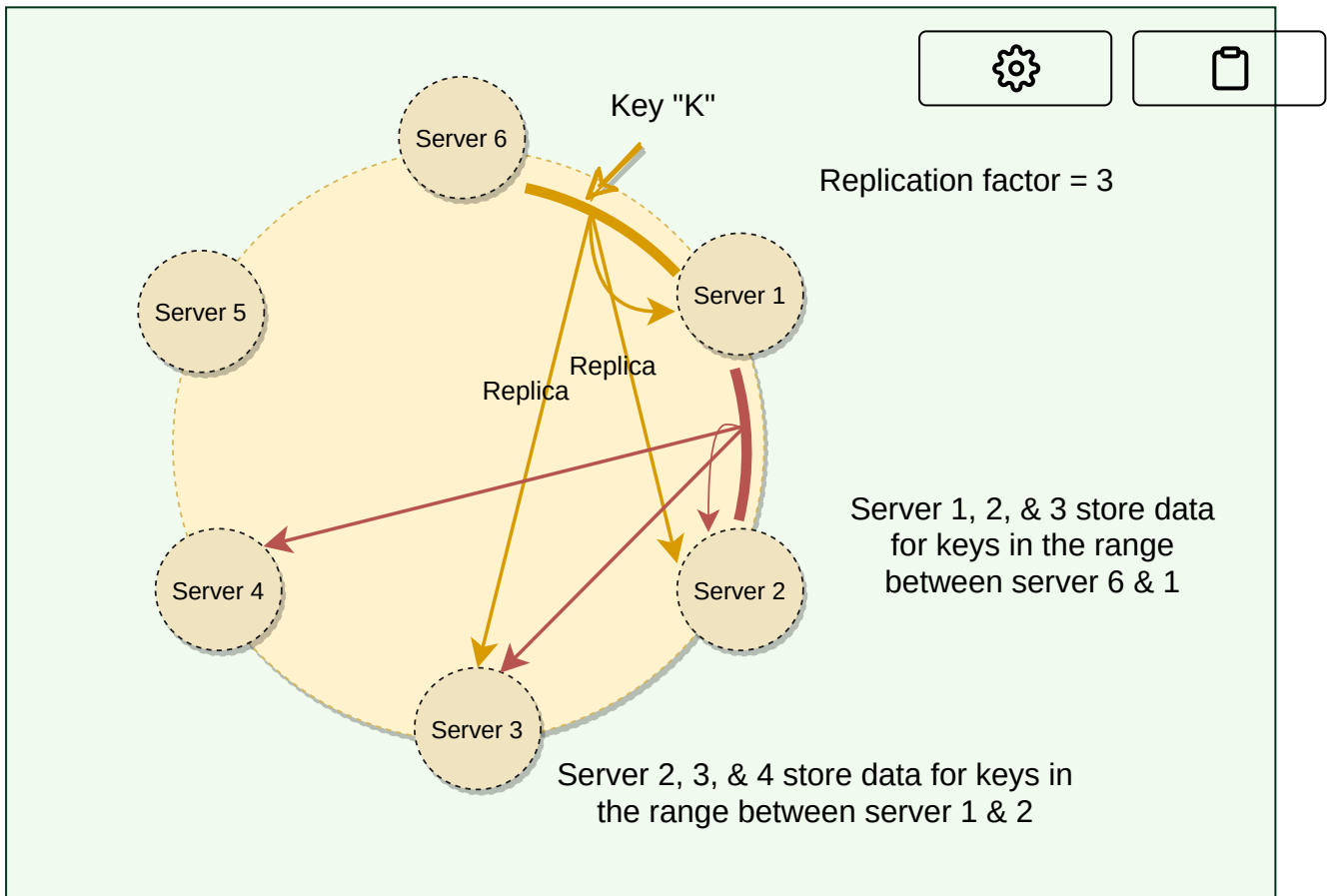
We'll cover the following



- What is optimistic replication?
- Preference List
- Sloppy quorum and handling of temporary failures
- Hinted handoff

What is optimistic replication?#

To ensure high availability and durability, Dynamo replicates each data item on multiple N nodes in the system where the value N is equivalent to the replication factor and is configurable per instance of Dynamo. Each key is assigned to a **coordinator node** (*the node that falls first in the hash range*), which first stores the data locally and then replicates it to $N - 1$ clockwise successor nodes on the ring. This results in each node owning the region on the ring between it and its N th predecessor. This replication is done asynchronously (*in the background*), and Dynamo provides an **eventually consistent** model. This replication technique is called **optimistic replication**, which means that replicas are not guaranteed to be identical at all times.



Replication in consistent hashing

Each node in Dynamo serves as a replica for a different range of data. As Dynamo stores N copies of data spread across different nodes, if one node is down, other replicas can respond to queries for that range of data. If a client cannot contact the coordinator node, it sends the request to a node holding a replica.

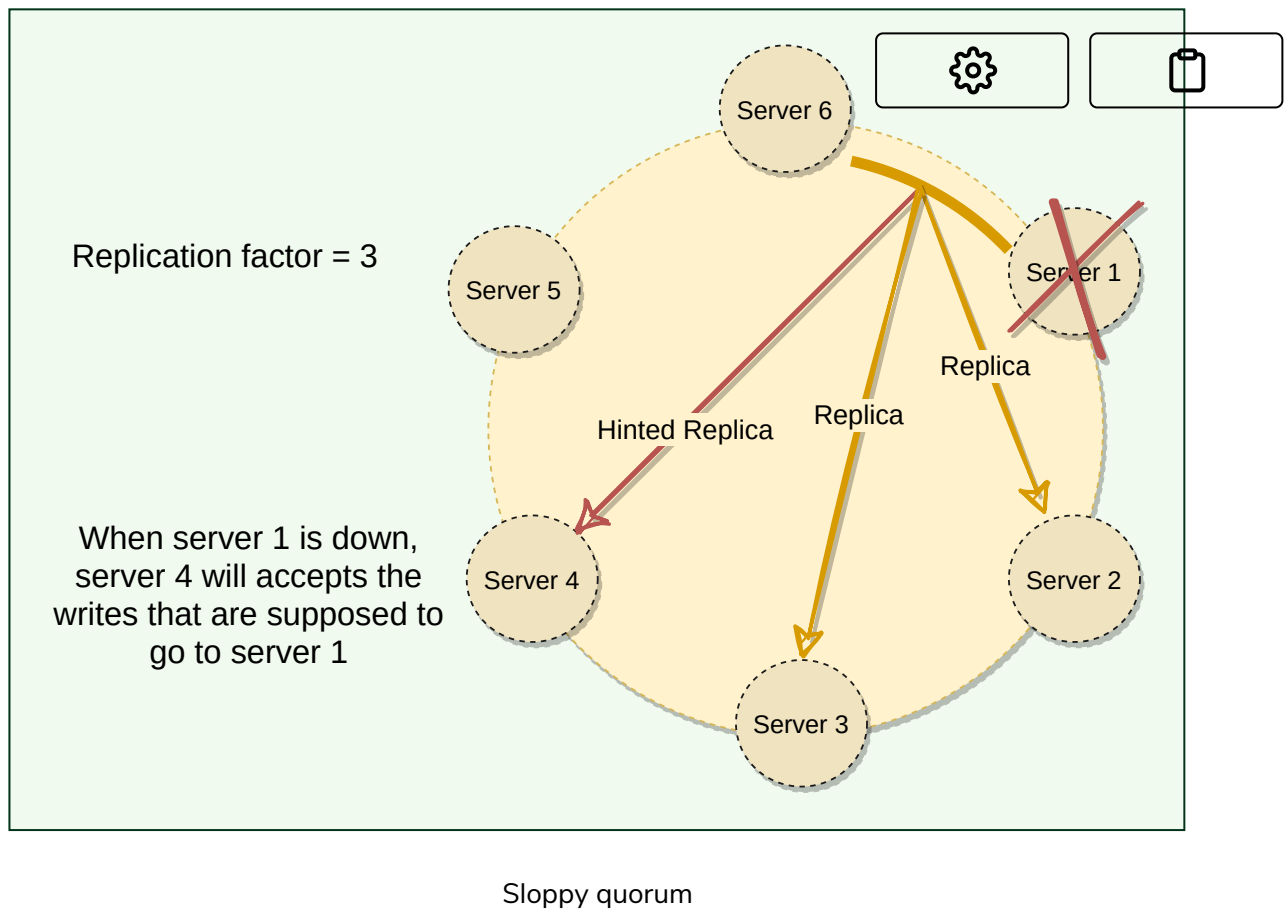
Preference List#

The list of nodes responsible for storing a particular key is called the preference list. Dynamo is designed so that every node in the system can determine which nodes should be in this list for any specific key (discussed later). This list contains more than N nodes to account for failure and skip virtual nodes on the ring so that the list only contains distinct physical nodes.

Sloppy quorum and handling of temporary failures#

Following traditional quorum approaches, any distributed system becomes unavailable during server failures or network partitions and would have reduced availability even under simple failure conditions. To increase the availability, Dynamo does not enforce strict quorum requirements, and instead uses something called **sloppy quorum**. With this approach, all read/write operations are performed on the first N healthy nodes from the preference list, which may not always be the first N nodes encountered while moving clockwise on the consistent hashing ring.

Consider the example of Dynamo configuration given in the figure below with $N = 3$. In this example, if `Server 1` is temporarily down or unreachable during a write operation, its data will now be stored on `Server 4`. Thus, Dynamo transfers the replica stored on the failing node (*i.e.*, `Server 1`) to the next node of the consistent hash ring that does not have the replica (*i.e.*, `Server 4`). This is done to avoid unavailability caused by a short-term machine or network failure and to maintain desired availability and durability guarantees. The replica sent to `Server 4` will have a hint in its metadata that suggests which node was the intended recipient of the replica (*in this case*, `Server 1`). Nodes that receive hinted replicas will keep them in a separate local database that is scanned periodically. Upon detecting that `Server 1` has recovered, `Server 4` will attempt to deliver the replica to `Server 1`. Once the transfer succeeds, `Server 4` may delete the object from its local store without decreasing the total number of replicas in the system.





Hinted handoff#

The interesting trick described above to increase availability is known as hinted handoff, i.e., **when a node is unreachable, another node can accept writes on its behalf**. The write is then kept in a local buffer and sent out once the destination node is reachable again. This makes Dynamo “**always writeable**.” Thus, even in the extreme case where only a single node is alive, write requests will still get accepted and eventually processed.

The main problem is that since a sloppy quorum is not a strict majority, the data can and will diverge, i.e., it is possible for two concurrent writes to the same key to be accepted by non-overlapping sets of nodes. This means that multiple conflicting values against the same key can exist in the system, and we can get stale or conflicting data while reading. Dynamo allows this and resolves these conflicts using Vector Clocks.

 **Back**

Data Partitioning

 **Next** 

Vector Clocks and Conflicting Data

☒ Mark as Completed

 Report an Issue