Consistent Hashing

While designing a scalable system, the most important aspect is defining how the data will be partitioned and replicated across servers.

Let's first define these terms before moving on:

> ❗ **Data Partitioning** is the process of distributing data across a set of servers — it improves the scalability and performance of the system.

> ❗ **Data Replication** is the process of making multiple copies of data and storing them on different servers — it improves the availability and durability of the data across the system.

Data partition and replication strategies lie at the core of any distributed system.

A carefully designed scheme for partitioning and replicating the data enhances the performance, availability, and reliability of the system, and also defines how efficiently the system will be scaled and managed.

What is Data Partitioning?

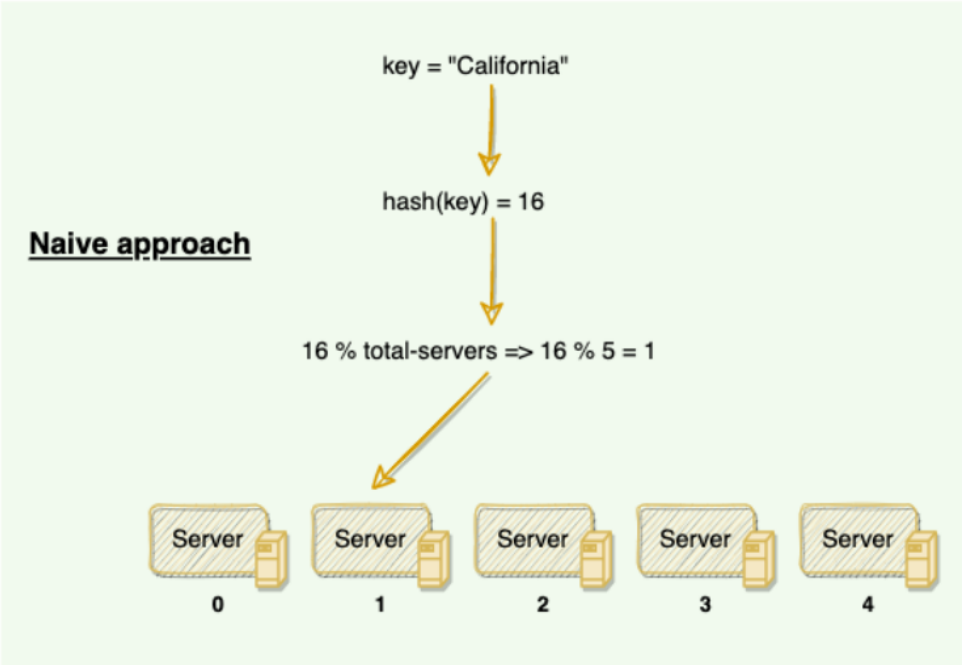
As stated above, the act of distributing data across a set of nodes is called data partitioning.

There are two challenges when we try to distribute data:

1. How do we know on which node a particular piece of data will be stored?
2. When we add or remove nodes, how do we know what data will be moved from existing nodes to the new nodes? Additionally, how can we minimise data movement when nodes join or leave?

A naïve approach will use a suitable hash function to map the data key to a number. Then, find the servery by applying modulo on this number and the total number of servers.

For example:



Data partitioning using simple hashing

The scheme described in the above diagram solves the problem of finding a server for storing/retrieving the data.

But when we add or remove a server, all of our existing mappings will be broken.

This is because the total number of servers will be changed, which was used to find the actual server storing the data.

So, to get things working again, we have to **remap all the keys** and move our data based on the new server count — this will be a very laborious and tedious process.

Consistent Hashing to the Rescue

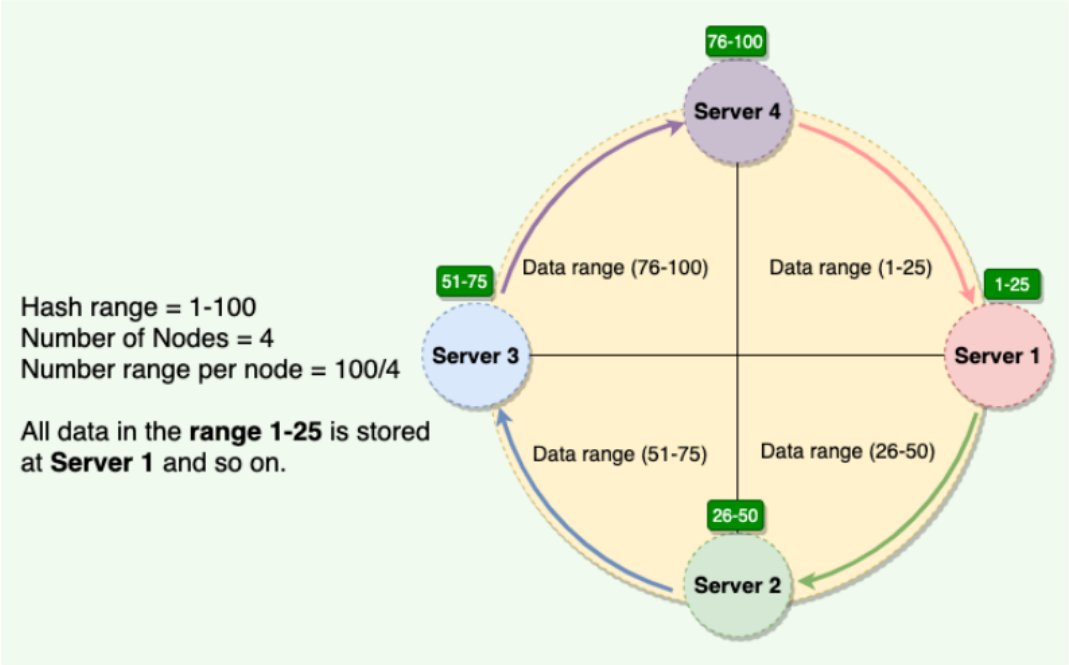
Distributed systems can use **Consistent Hashing** to distribute data across nodes.

Consistent hashing maps data to physical nodes, and ensures that only a small set of keys move when servers are added or removed.

Consistent hashing stores the data managed by a distributed system in a ring.

Each node in the ring is assigned a range of data.

Here is an example of the consistent hash ring:



Consistent Hashing ring

With consistent hashing, the ring is divided into smaller, predefined ranges.

Each node is assigned one of these ranges.

The start of a range is called a **token**. This means that each node will be assigned one token.

The range assigned to each node is computed as follows:

- **Range start**: token value
- **Range end:** next token value - 1

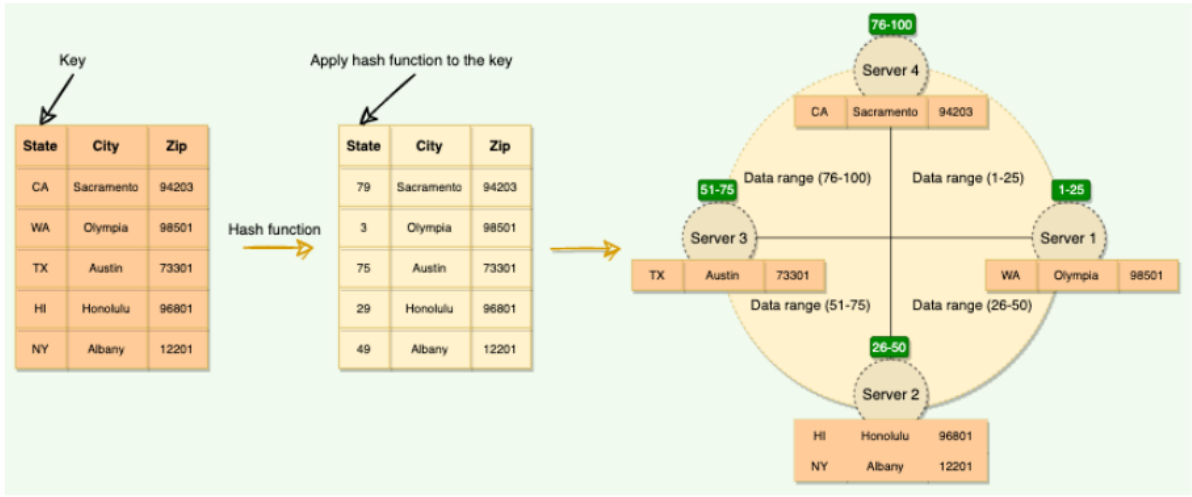
Here are the tokens and data ranges of the four nodes described in the above diagram:

Server	Token	Range Start	Range End
Server 1	1	1	25
Server 2	26	26	50
Server 3	51	51	75
Server 4	76	76	100

Whenever the system needs to read or write data, the first step it performs is to apply the MD5 hashing algorithm to the key.

The output of this hashing algorithm determines within what range the data lies and hence, on which node the data will be stored.

As we saw above, each node is supposed to store data for a fixed range. Thus, the hash generated from the key tells us the node where the data will be stored.



Distributing data on the Consistent Hashing ring

The consistent hashing scheme described above works great when a node is added or removed from the ring, as in these cases, only the next node is affected.

However, this scheme can result in non-uniform data and load distribution.

But this problem can be resolved with the help of virtual nodes.

Virtual Nodes

Adding and removing nodes in any distributed system is quite common. Existing nodes can die and may need to be decommissioned. Similarly, new nodes may be added to an existing cluster to meet growing demands.

To efficiently handle these scenarios, consistent hashing makes use of **virtual nodes** (or **Vnodes**)

As we saw above, the basic consistent hashing algorithm assigns a single token (or a consecutive hash range) to each physical node.

This was a static division of ranges that requires calculating tokens based on a given number of nodes.

This scheme made adding or replacing a node an expensive operation, as, in this case, we would like to rebalance and distribute the data to all other nodes, resulting in moving a lot of data.

Here are a few potential issues associated with a manual and fixed division of the ranges:

- Adding or removing nodes.

  Adding or removing nodes will result in recomputing the tokens causing a significant administrative overhead for a large cluster.

- **Hotspots** — in a distributed system, any server responsible for a huge partition of data can become a bottleneck for the system. A large share of data storage and retrieval requests will go to that node, which can effectively bring the performance of the whole system down. Such loaded servers are called hotspots.

  Since each node is assigned one large range, if the data is not evenly distributed, some nodes can become hotspots.

- Node rebuilding.

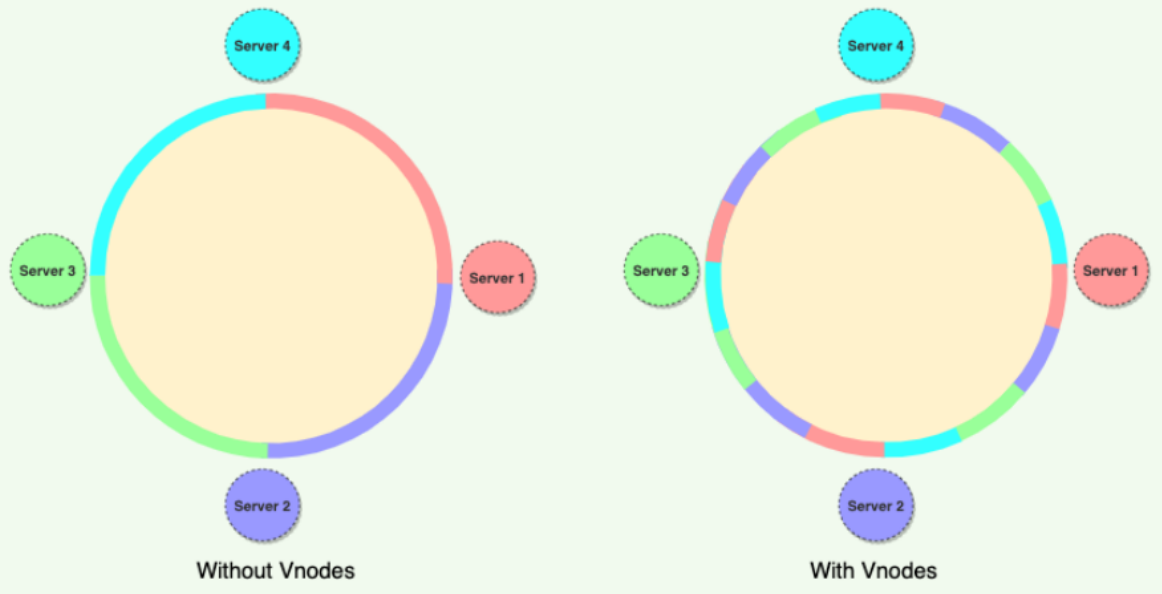
  Since each node's data might be replicated (for fault-tolerance) on a fixed number of other nodes, when we need to rebuild a node, only its replica nodes

can provide the data. This puts a lot of pressure on the replica nodes and can lead to service degradation.

To handle these issues, consistent hashing introduces a new scheme of distributing the tokens to physical nodes.

Instead of assigning a single token to a node, the hash range is divided into multiple smaller ranges, and each physical nodes is assigned several of these smaller ranges.

Each of these subranges is considered a **virtual node**. With Vnodes, instead of a node being responsible for just one token, it is responsible for many tokens (or subranges).



Comparing Consistent Hashing ring with and without Vnodes

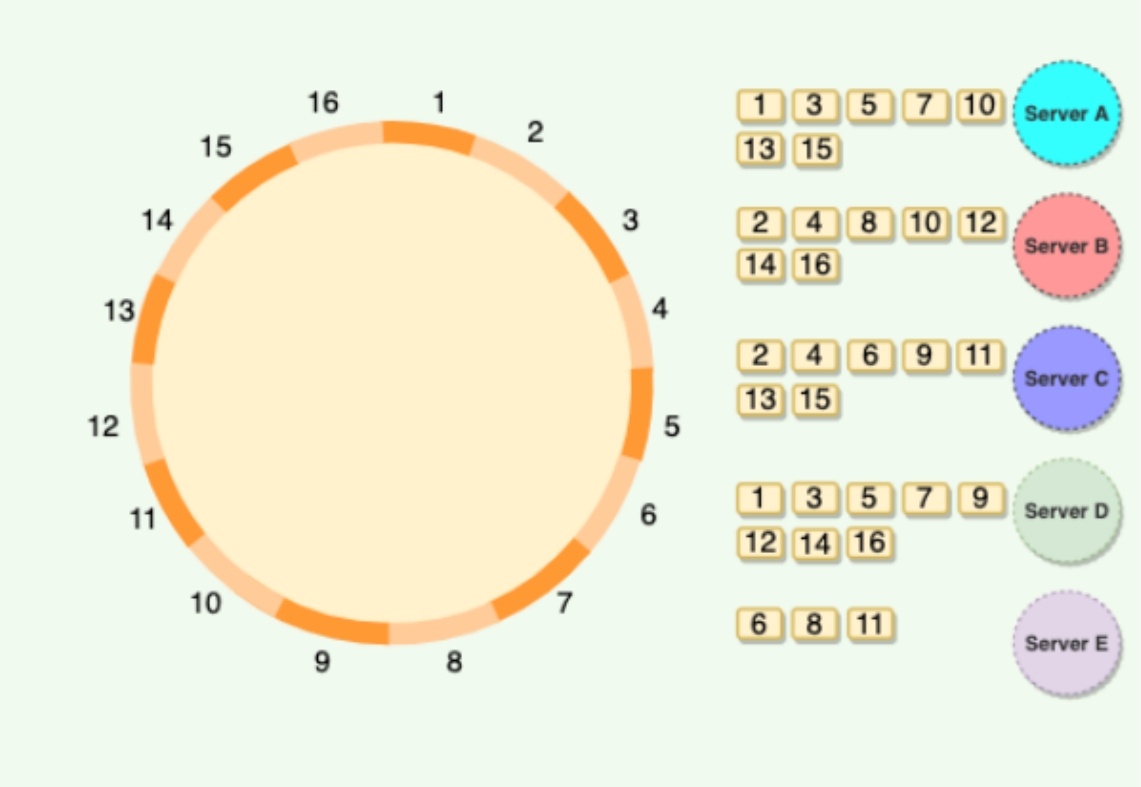
Practically, Vnodes are randomly distributed across the cluster, and are generally non-contiguous so that no two neighbouring Vnodes are assigned to the same physical node or rack.

Additionally, nodes do carry replicas of other nodes for fault tolerance.

Also, since there can be heterogenous machines in the clusters, some servers might hold more Vnodes that others.

The figure below shows how physical nodes A, B, C, D, and E use Vnodes of the consistent hash ring. Each physical node is assigned a set of Vnodes, and each

Vnode is replicated once:



Mapping Vnodes to physical nodes on a Consistent Hashing ring

Advantages of Vnodes

Vnodes give us the following advantages:

1. As Vnodes help spread the load more evenly across the physical nodes on the cluster by dividing the hash ranges into smaller subranges, this speeds up the rebalancing process after adding or removing nodes.

   When a new node is added, it receives many Vnodes from the existing nodes to maintain a balanced cluster.

   Similarly, when a node needs to be rebuilt, instead of getting data from a fixed number of replicas, many nodes participate in the rebuilding process.

2. Vnodes make it easier to maintain a cluster containing heterogenous machines.

   This means that with Vnodes, we can assign a high number of subranges to a powerful server, and a low number of subranges to a less powerful server.

3. In contrast to one big range, since Vnodes help assign smaller ranges to each physical node, this decreases the probability of hotspots.

Examples

Dynamo and **Cassandra** use consistent hashing to distribute their data across nodes.