Consistent Hashing - Background

Background

The act of distributing data across a set of nodes is called **data partitioning**.

There are two challenges when we try to distribute data:

- ✓ How do we know on which node a particular piece of data will be stored?
- ✓ 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 minimize data movement when nodes join or leave?

A naive approach will use a suitable hash function that maps the data key to a number. Then, find the server by applying modulo on this number with total number of servers.

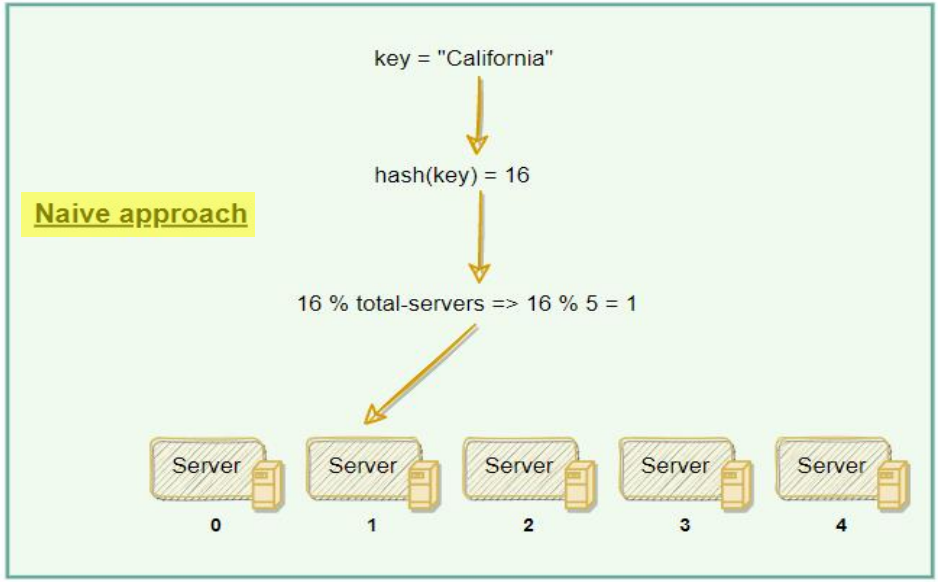
The approach described in the above diagram solves the problem of finding a server for storing/retrieving the data.

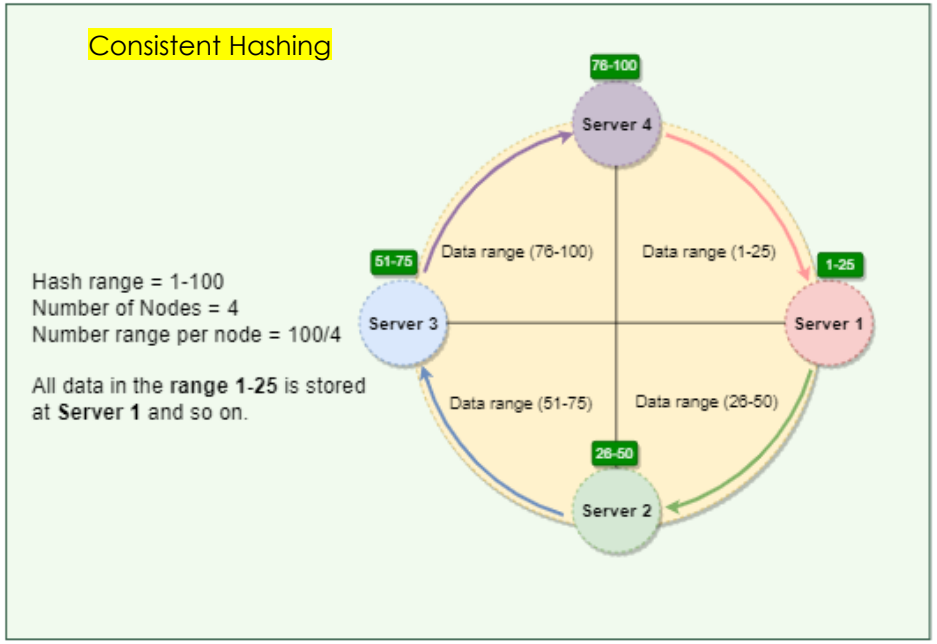
But when we add or remove a server, we must remap all the keys and move our data based on the new server count, which will be a complete mess!

This is one big problem in this approach. Also, by any chance if you are thinking that maintaining number of server same will help here than that is totally wrong. Because addition & subtraction of node happening because the scale we want to achieve.

So, you must find some approach which will help you to minimize the changes or impact of rework in your system. Here comes the approach and improvements suggested by some sharp mind , and that is called consistent hashing.

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 – Approach (how?) 1/3

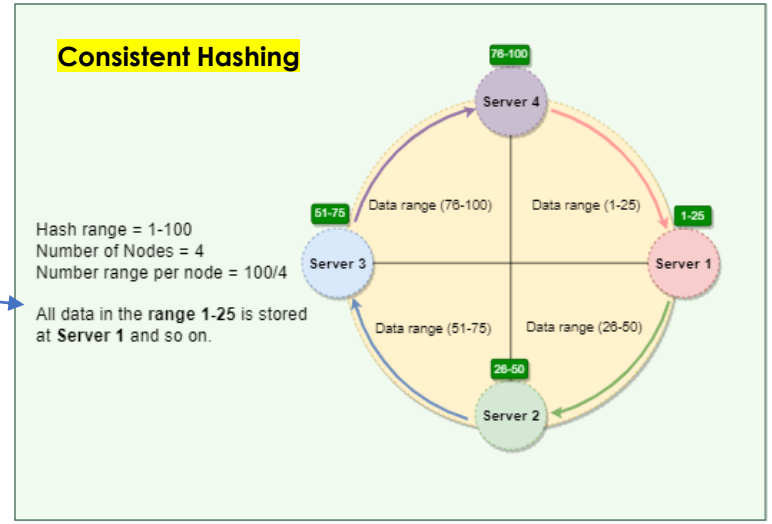
Approach Description

Consistent Hashing technique stores the data managed by a distributed system in a ring. Each node in the ring is assigned a range of data. In the right-side diagram, you can see the consistent hash ring.

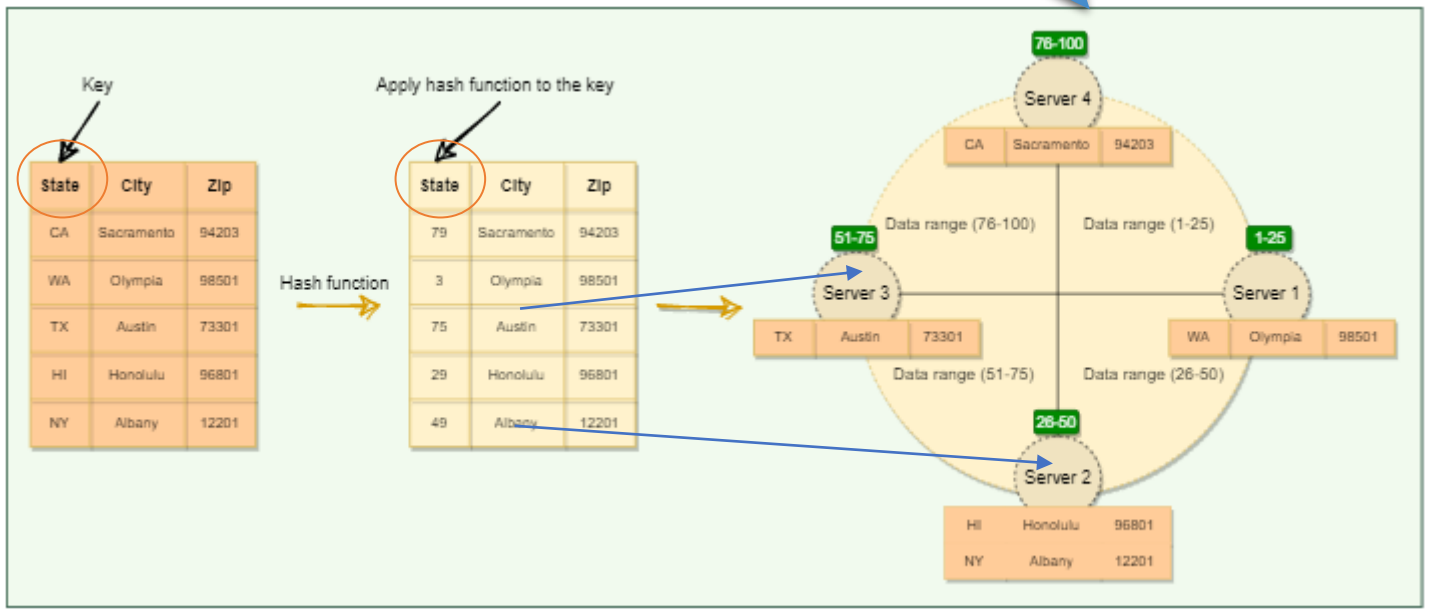
Here are the tokens and data ranges of the four nodes described in the Node vs Range diagram.

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 which 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.



Overall process explained in this diagram



Node vs Range

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

Consistent Hashing – Approach 2/3

Approach Description Contd…

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

For example, when a node is removed, the next node becomes responsible for all the keys stored on the outgoing node.

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

So, next problem for us to minimize the above problem. This problem can be solved with the **help of Virtual nodes.**

How? We will see the next section.

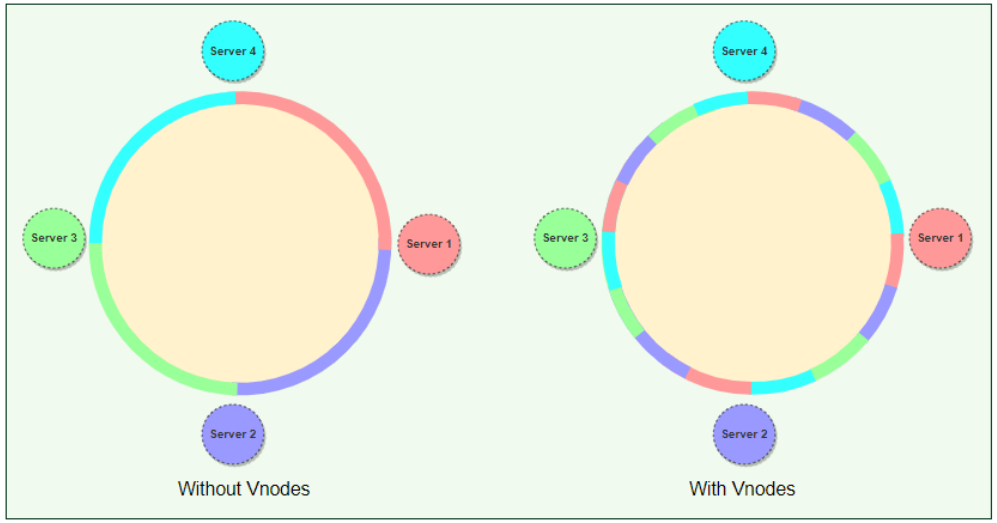
Virtual Nodes

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**: 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.**



Consistent Hashing – Approach 3/3

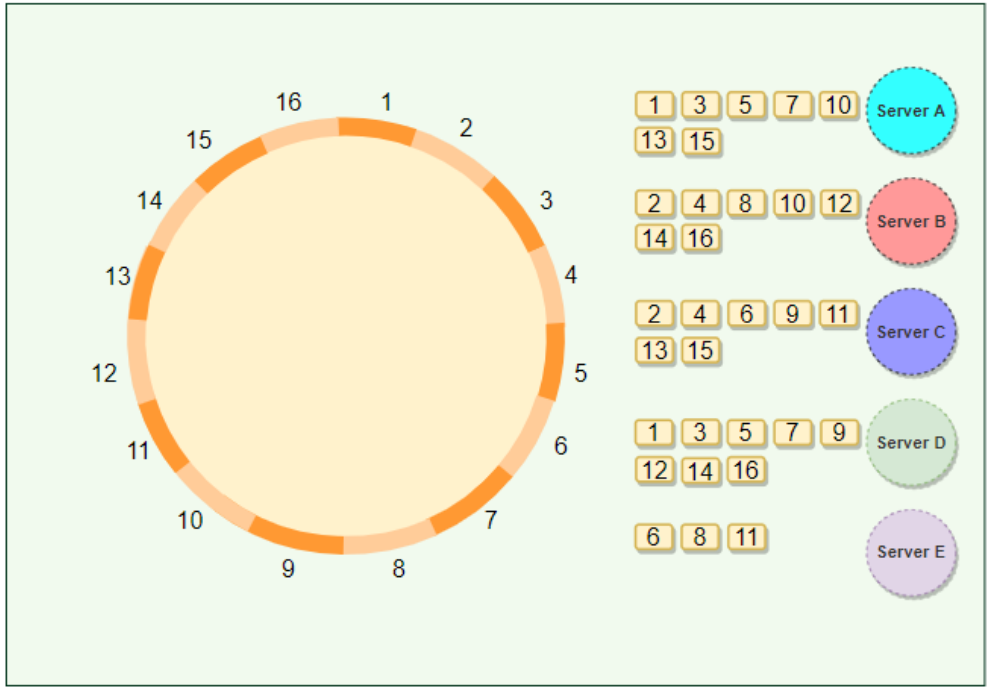
Approach Description Contd…

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

Vnodes are **randomly distributed** across the cluster and are generally **non-contiguous** so that no two neighboring 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 heterogeneous machines in the clusters, some servers might hold more Vnodes than others.

The figure right shows how physical nodes A, B, C, D, & E use Vnodes of the Consistent Hash ring. Each physical node is assigned a set of Vnodes and each Vnode is replicated once.



Advantages of Vnodes

- ✓ 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 rebuild process.
- ✓ Vnodes make it easier to maintain a cluster containing heterogeneous machines. This means, with Vnodes, we can assign a high number of sub-ranges to a powerful server and a lower number of sub-ranges to a less powerful server.
- ✓ In contrast to one big range, since Vnodes help assign smaller ranges to each physical node, this decreases the probability of hotspots.

Thank You
Abinash Mishra
Code Link : Will complete the code part by weekend(30th Jan 2022)… Keep loving the post
https://github.com/abmishra1234/4AM_Club_Coding