Why do we need this?

1. When you have multi servers, how to distribute requests good?

**Consistent hashing**

1. A diagram of a computer

   Description automatically generated

Objecst and servers hashed to the same value here

1. The hashed IP addresses of the nodes are used to assign a position for the nodes on the hash ring

2.

**Partitinoing data here**

Data set must bepartitioned among multiple caches servers(nodes) to horizontally sclae

**What’s key ranged partitioning here?**

A diagram of a computer system

Description automatically generated

How exactly does this consistent hashign work here?

The follwisng is what happens when you try to store

A diagram of a diagram

Description automatically generated

The basic gist behind the consistent hashing algorithm is to hash both node identifier and the key hashign value here

0. Note the ip address of the node is also hashed

0.5.T he key of the data object is hashed using the same hash function to locate the position of the key on the hash ring

1. Hash ‘xyz’ into 3 here

2. You traverse the ring in a clockwise fashion until sth is found here

**What if a node fails now?**

The failure (crash) of a node results in the movement of data objects from the failed node to the immediate neighboring node in the clockwise direction. The remaining nodes on the hash ring are unaffected [5].

**What if a new node is added here?**

When a new node is provisioned and added to the hash ring, the keys (data objects) that fall within the range of the new node are moved out from the immediate neighboring node in the clockwise direction.

Design a consistent hashing here

1 .What's the problem of the rehashing problem

When using the modulus operator -> hash the key

When a request comes in, assign it a request id a key, key 1 and then calc its hash and then map data to a value and then apply modulus (on # of nodes) and then determine which server thsi will go here based on the server id

A black background with blue text

Description automatically generated

**And then next:**

What's the problem with regular hashing here?

1. The regular approach works well when the size of the server pool is fixed, and the data distribution is even. However, problems arise when new servers are added, or existing servers are removed

2.

And then what do we do here then?

**Consistent hashing short summary here:**

1. Dynamically add or remove server here, it uses a ring here, and has a total # n here,

2. Server mapped to a hash ring

3. For example here, In Figure 8, after a new server 4 is added, only key0 needs to be redistributed. k1

4. . Before *server 4* is added, *key0* is stored on *server 0*. Now, *key0* will be stored on *server 4* because *server 4* is the first server it encounters by going clockwise from *key0*’s position on the ring.

**How does consistent hashing work?**

A screenshot of a computer

Description automatically generated

Step 1: hash the node first from node 1 -> s1 here

Step 2: hash the request next

A screen shot of a computer

Description automatically generated

And then

How to know which key -> which node? It will have a server id associated with the above

A screen shot of a computer

Description automatically generated

**Key here**

**what happens when u add a node here?**

Situation 1: when a new node is added here, what happens: (say node 4 is added which maps to node 4)

WHy key 3 is reallocated?

Key 3 will be reallc to s4 here, due to clockwise direction when reallocating here, k3 used to belong to s1

A screenshot of a computer

Description automatically generated

WHen node 3 is removed, key 2 will then be reallocated in clockwise direciton and only need to rehash key 2.

**What's the con with consistent hashing here?**

1. Server 1 could handle more data then the regular server here

2. Non-uniform distributino of nodes here,

How do we solve this problem?

1. Use virtual nodes to store better here, apply multiple hash functions to a key up here

2. Each physical node represented by 3 virtual nodes on the ring here

A screenshot of a computer screen

Description automatically generated

3. And then here we have the code as above,

A node with more cpu can have more keys as well as well:

How are the server nodes stored here?

The 3 is arbitrarily chosen; and in real-world systems, the number of virtual nodes is much larger. Instead of using s0, we have s0\_0, s0\_1, and s02 to represent \_server 0 on the ring. Similarly, s1\_0, s1\_1, and s1\_2 represent server 1 on the ring. With virtual nodes, each server is responsible for multiple partitions. Partitions (edges) with label s0 are managed by server 0. On the other hand, partitions with label s1 are managed by server 1.

2. Consistent hashing solves the above problem as said

Design a backend here for the ecommerce application massive # of data.

3

And then using this

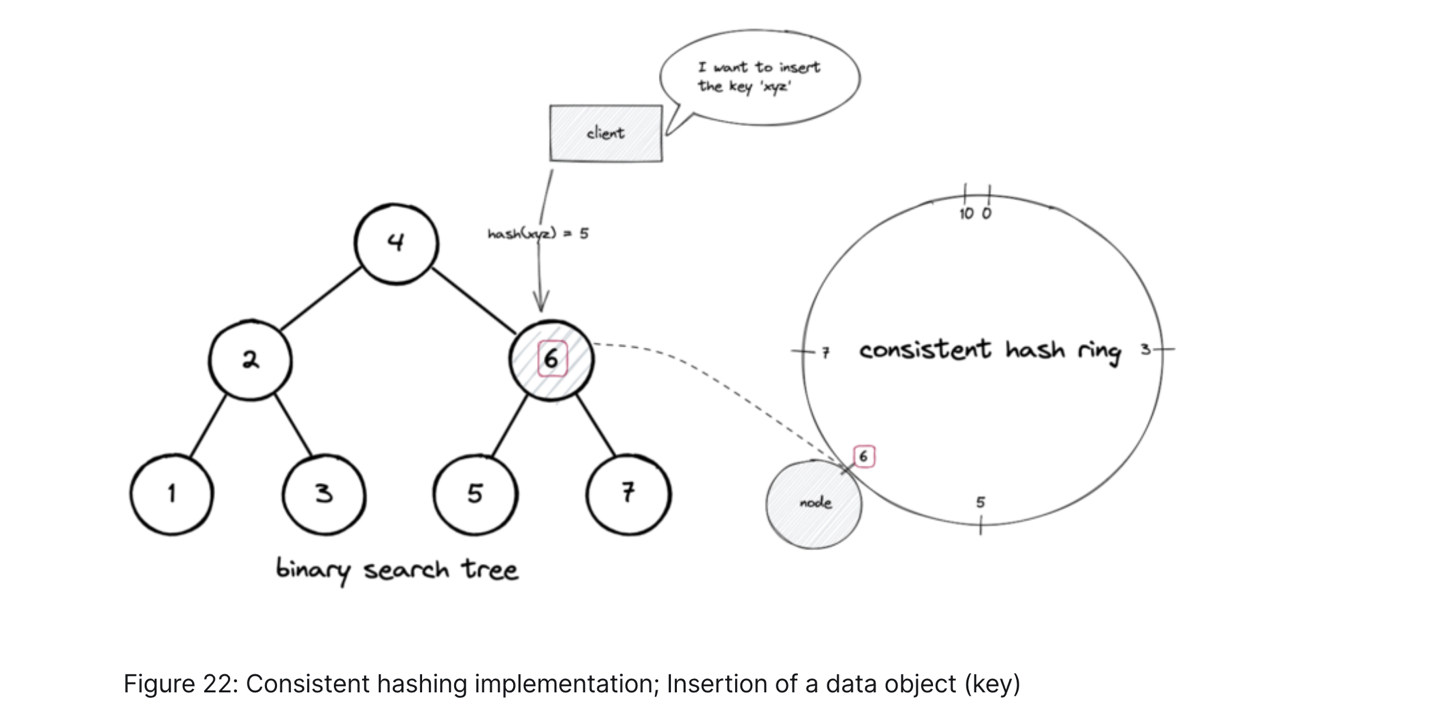
**Consistent hashing the implementaiont here**

How is this done:

1. The self-balancing binary search tree (**BST**) data structure is used to store the positions of the nodes on the hash ring

2. Key of the bst:  positions of the nodes on the hash ring.

3.



What are the benefits of consistent hashing?

The following are the advantages of consistent hashing [3]:

* horizontally scalable
* minimized data movement when the number of nodes changes
* quick replication and partitioning of data

**What are the drawbacks of consistent hashing?**

The following are the disadvantages of consistent hashing [5]:

* cascading failure due to hotspots
* non-uniform distribution of nodes and data
* oblivious to the heterogeneity in the performance of nodes

Imagine you send a lot of requests to just 1 node here.

when a specific data object becomes extremely popular, consistent hashing will still send all the requests for the popular data object to the same subset of nodes resulting in a degradation of the service