**Chapter 5**

Design Consistent Hashing

Consistent Hashing – common method used to distribute data evenly across servers and support horizontal scaling

**Simple hashing**

ServerIndex = hash {key} % N where N is the size of the server pool

*Concept*

1. Pass each object key through a hashing function, mapping to a known range of numeric values
   * A good hashing function will distribute values evenly across the entire range
2. Perform the modular operation on the hash against the number of servers
   * This determines which server the object belongs to
3. As long as the number of servers stays the same, an object key will always map to the same server

*Pros*

* Works well when size of the server pool is fixed, and data distribution is even

*Cons*

* Mishandles when servers are added or removed
  + When the number of servers changes, the keys are redistributed, and many keys will connect to the wrong servers to fetch data
  + Causes nearly all objects/data to relocate

**Consistent hashing**

*Concept*

* Same hashing function is used, outputting a hash space – the range of numeric values, outputted by a hashing function
* Hash ring – connecting the endpoints of a hash space to form a ring structure
* Both keys and servers are hashed and have a position on the ring
* Keys are assigned to the next available server on the ring going clockwise

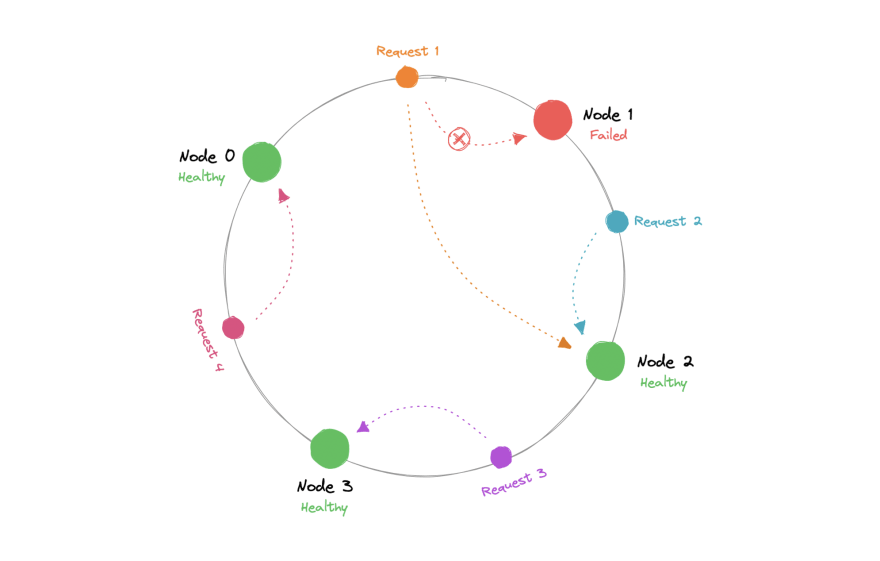
*Adding server*

*In consistent hashing, adding in a new server only requires redistribution of a fraction of the keys rather than all keys*

1. Suppose there are 4 servers placed in ascending order on the ring
2. A new server (server\_4) is added between server\_0 and server\_3, dividing the space between them
3. Only keys between server\_3 and server\_4 needs to be redistributed
   * Instead of server\_0, keys in this space now reroutes to server\_4 because it is the new closest proximity server in the ring

*Removing a server*

* When server\_1 is removed, all keys between server\_0 and server\_1 will be rerouted to server\_2 instead



*Pros*

* When the number of servers changes, only a fraction of the keys affected will be redistributed
* Very good when data is evenly distributed

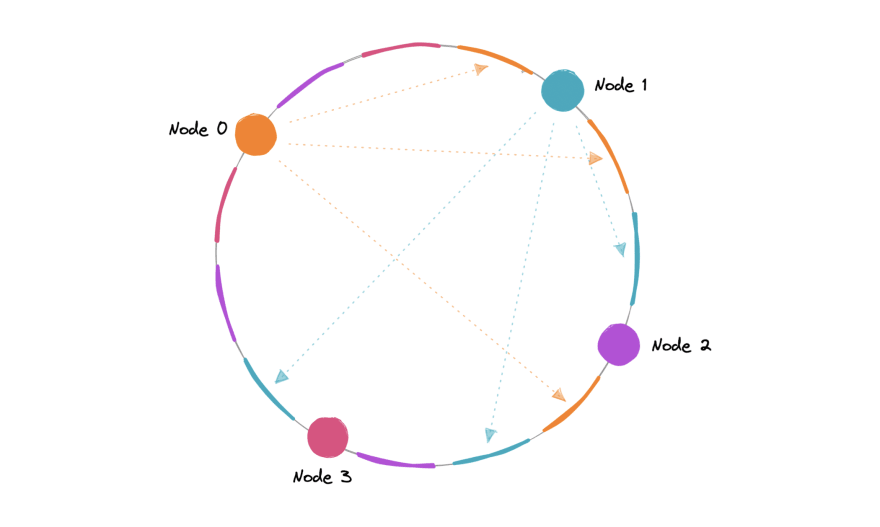
*Cons*

* The distribution of servers on the ring are likely to uneven, especially when they are based on IP or geographic addresses
* Server overloading – even if servers are perfectly partitioned, if servers are down frequently, keys are no longer even distributed and certain servers will be forced to support twice the amount of keys
* Non-uniform key distribution also causes overloading

**Virtual Nodes**

*Concept*

* Servers are no longer represented by their single location on the ring but are split to virtual nodes – multiple smaller partitioned areas distributed in the ring representing the same server
* As the number of virtual nodes increases, the distribution of keys becomes more balanced



*Pros*

* Solves the uneven distribution problems including when servers are down

*Cons*

* More virtual nodes require more storage space for metadata and should be carefully tuned for the business requirements

**Real World Examples**

|  |  |
| --- | --- |
| Amazon DynamoDB, Apache Cassandra | Data partitioning |
| Akamai Content delivery networks | Distribute web contents evenly |
| Maglev Load balancers | Distribute persistent connections evenly |