Partitioning

What is partitioning?

In large systems, we are dealing with tons of data, and as a result tables may become too big/perform too many queries for one single machine.

Partitioning is splitting this table up into many chunks to go on various database nodes.

Partitioning is often used in conjunction with replication.

Objectives of partitioning

On each node we want:

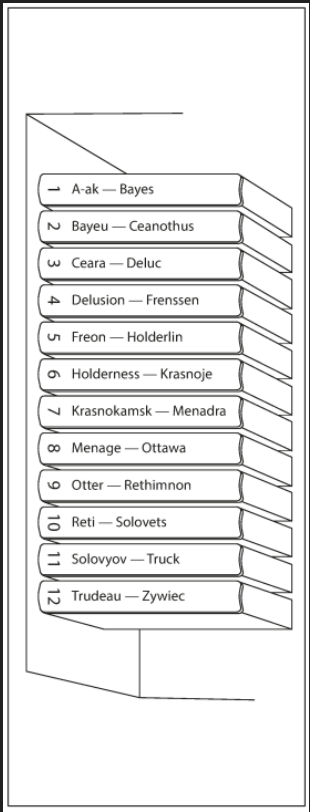
- A relatively similar amount of data
- A relatively similar amount of reads/and writes to the data

If we are unable to achieve this, certain nodes will be overloaded relative to others, known as hot spots

Methodologies for partitioning

- By ranges of keys
- By ranges of the hash of keys
  - Note: do not take a hash of the key and then do a modulo with the number of nodes, as this will cause the location of every key to change if a partition node is added or removed

Key Range Partitioning



- Not necessarily even ranges, of keys, some ranges may be hotter
- Keep keys sorted within a partition to best support range queries

Pros:

- Simple and allows for effective range queries

Cons:

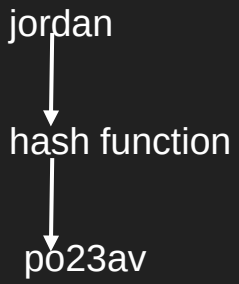
- Have to actually determine the ranges to make sure they are relatively even in data and load (can be done manually or by database)
- Can easily lead to hotspots (if for example partitioning by range of timestamps)

Hash Range Partitioning

Take a hash of the key, and put it into the proper partition

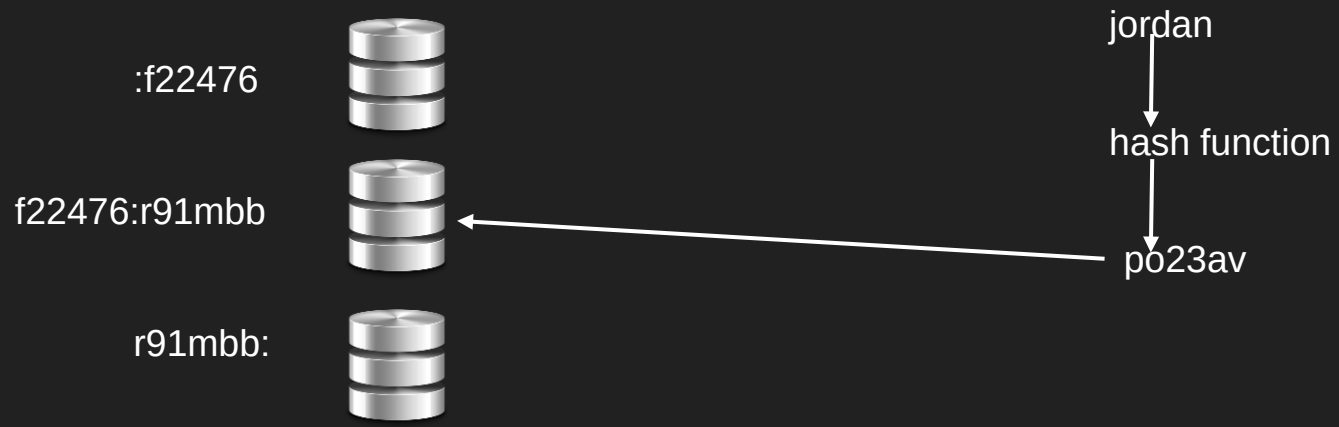
Hash Range Partitioning

Take a hash of the key, and put it into the proper partition



Hash Range Partitioning

Take a hash of the key, and put it into the proper partition



Hash Range Partitioning Tradeoffs

Pros:

- Keys are evenly distributed between nodes (assuming good hash function)

Cons:

- No more range queries on the partition key, have to check every partition
- If a key has a lot of activity will still lead to hot spots

Indexes in a partitioned database configuration

Recall: An index is additional metadata that shows memory addresses of rows corresponding to certain field values in the row

Secondary Index

Position: point guard - [10, 14, 21, 37]
Position: center - [1, 8, 12, 19]
Position: power forward - [5, 11]
Position: small forward - [6, 7, 13, 22]
Position: shooting guard - [3, 15, 16]

Secondary Index Options

- Local Indexes
- Global Indexes

Local Indexes

Idea: Hold a secondary index that only holds rows from the partition the index is located on

ID	Name	Position
1	Michael Jordan	Shooting Guard
2	Lebron James	Small Forward
3	Kobe Bryant	Shooting Guard

Secondary Index

Position: point guard - []
Position: center - []
Position: power forward - []
Position: small forward - [1, 3]
Position: shooting guard - [2]

ID	Name	Position
65	Khris Middleton	Small Forward
66	Chris Paul	Point Guard
67	Dwight Howard	Center

Secondary Index

Position: point guard - [66]
Position: center - [67]
Position: power forward - []
Position: small forward - [65]
Position: shooting guard - []

Local Index Tradeoffs

Pros:

- Fast on write because all data that is being kept track of is being stored locally on the partition

Cons:

- Slow on read because if using a secondary index have to query every partition to accumulate the index results

Global Indexes

Idea: Partition the secondary index, the index can contain references to data on any partition

ID	Name	Position
1	Michael Jordan	Shooting Guard
2	Lebron James	Small Forward
3	Kobe Bryant	Shooting Guard

Secondary Index

Position: point guard - [66]
Position: center - [67]

ID	Name	Position
65	Khris Middleton	Small Forward
66	Chris Paul	Point Guard
67	Dwight Howard	Center

Secondary Index

Position: power forward - []
Position: small forward - [2, 65]
Position: shooting guard - [1, 3]

Global Index Tradeoffs

Pros:

- Fast on read because all data for that index is being kept on one partition node

Cons:

- Slow on write because need to write to multiple partitions to update all of the various secondary indexes
- May require a distributed transaction (imagine the case one write succeeds and the other fails)

Rebalancing Partitions

If a node is added or removed, the goal is to keep the majority of the keys in the same place, and only move a few from each node so that we do not use a ton of bandwidth remapping every key (recall to use hash ranges instead of modulo)

Fixed Number of Partitions

In this system, we have 20 partitions no matter how many nodes there are

Fixed Number of Partitions

In this system, we have 20 partitions no matter how many nodes there are
Take all of the white chunks from each server and pass them to the new server!

Fixed Number of Partitions - Considerations

Choose a number of partitions that is reasonable:

- If too low, each partition will get too big and we will not be able to scale the application further (additionally transferring the partition to another node will take a super long time)
- If too high, there will be a lot of overhead on disk devoted to each partition

If your dataset is going to vary significantly in the future, maybe this isn’t for you

Dynamic Partitioning

Certain databases will adjust partition ranges dynamically so that they can reduce hot spots as the data access patterns change over time:

- Once a partition becomes too big, it is split into two pieces and one is assigned to another node
- Sometimes dynamic partitioning is not good because if the database incorrectly assumes a node is down, when there is actually just a slow network, it will repartition leading to more strain on the network

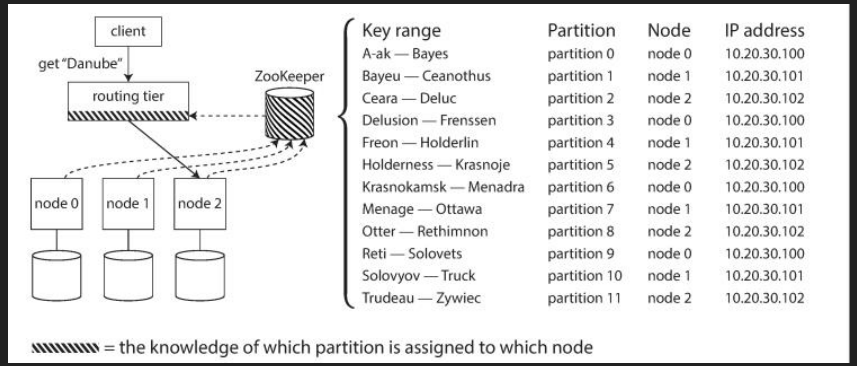
Fixed number of partitions per node

- Each node has a certain number of partitions on it which grow in size proportionally to the dataset
- If a new node joins the cluster it will split some of the partitions on existing nodes into two pieces, and take those for itself
- Very similar to consistent hashing algorithm to avoid unfair data splits

Sharding Summary

Unlike replication, which is always important to have (to increase availability), partitioning adds a lot of complexity to a system and should mainly only be used when the dataset has gotten big enough that putting the whole table on a single node is infeasible.

Generally requires some sort of coordination service or gossip protocol in order to keep track of which range corresponds to which partition



Sharding Summary Continued

Partitioning Methodology:

- Key ranges are better when we need to perform range queries
- Key hash ranges are better when we want to more evenly distribute data

Index Choices:

- Local indexes optimize for write speed
- Global indexes optimize for read speed

Rebalancing Choices:

- Fixed number of partitions is simpler to reason about, but requires choosing a good number
- A changing number of partitions may scale better, but doing so automatically may lead to unnecessarily rebalancing and putting extra stress on our databases