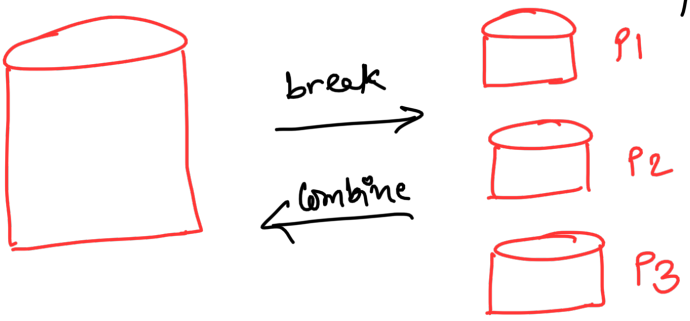
Designing Data Intensive Applications

CHAPTER - 6

Partitioning

↑ break data into parts ~ partitions



why? to scale the system. After certain point, data can't fit into single machine.

↳ the partitioned data should also be replicated on multiple nodes for fault tolerance. → high availability

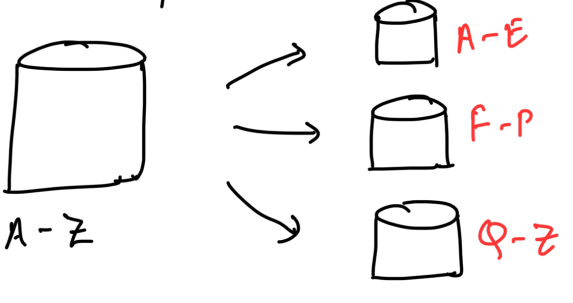
Partitioning of Key-Value Data

↳ Idea is to make sure data is evenly distributed among all nodes.

↳ If data is unevenly stored → called Skewed

① Partitioning By Key Range

Each partition can be assigned at continous range of keys.



ⓐ within each partition, keys are sorted. (SSTable & LSM Tree)

ⓑ range queries are easy.

ⓒ some access patterns can cause hotspots. example Timestamp.

Attach specific prefix to timestamp to solve issue.

② Partitioning By Hash of Key

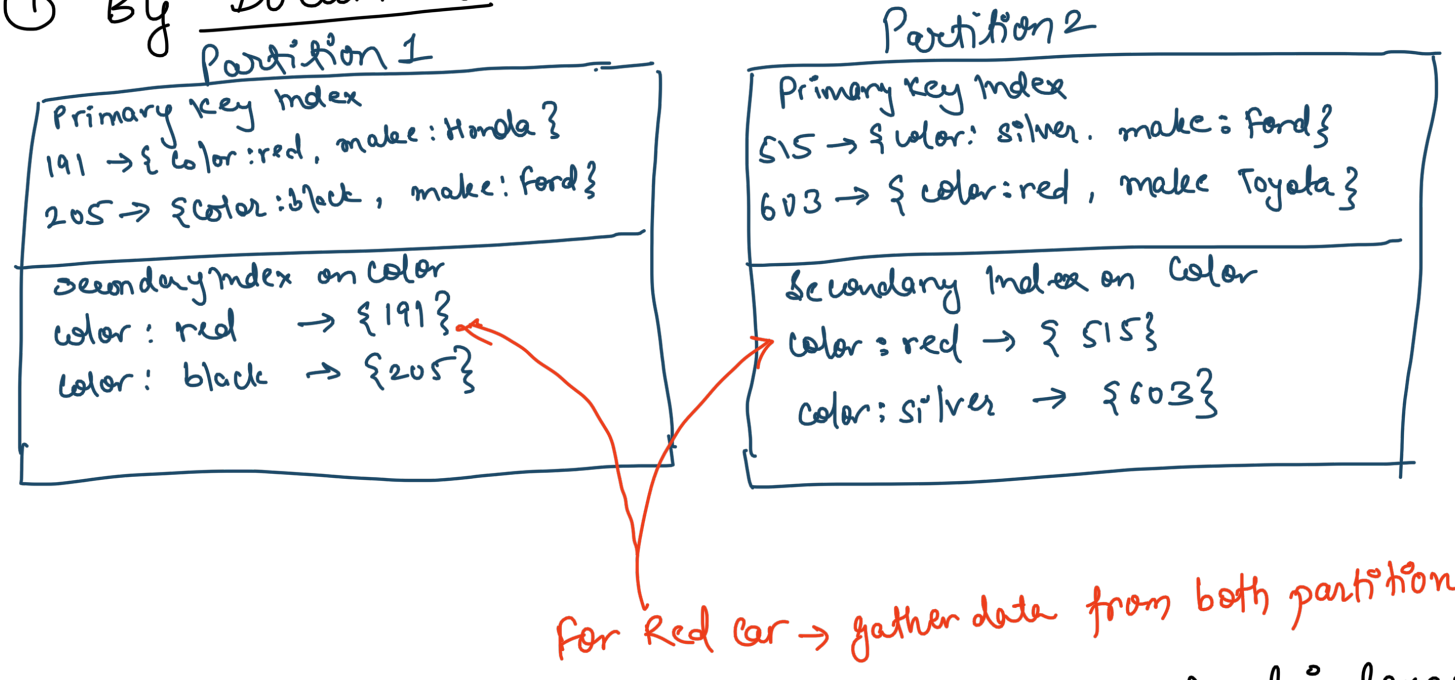
↳ to avoid skew and hotspots.
eg. Cassandra & MongoDB use MD5.

↳ assign each partition range of hashes.
↳ key will be stored in partition whose hash falls in specific range.

Disadvantage
↳ capability of efficient range queries.

Partioning Secondary Indexes

① By Document



→ Each partition is completely segregated, create local indexes.
→ Read queries on secondary indexes could be expensive → gathering data from all partitions.
→ used in MongoDB, Cassandra, Elastic Search

② By Term

↳ construct global index which covers data in all partitions.

↳ global index is also partitioned.
↳ you can use similar to range partitioning.

Benefits

↳ reads are more efficient. → hit only specific partition to gather data.

Downsides

↳ writes can be heavy. → write to single document can affect multiple partitions ↳ terms in doc located on different partitions

↳ example DynamoDB global secondary indexes.

Rebalancing Partitions

↳ add more CPU to handle load.
↳ add more disks.
↳ replace machine.

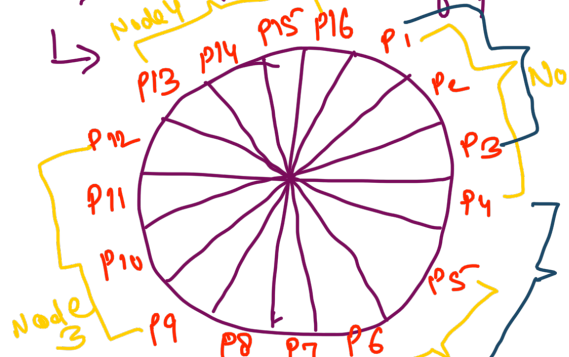
} → requires movement of data from one node to another.

Strategies

① Hash Mod N → not effective approach as number of nodes (N) will change, hence changing node number where data will be stored.

② Fixed Number of partitions → used in Riak, Elasticsearch couchbase, voldemort.

↳ Create lot of partitions and assign these partitions to nodes.



↳ as new nodes are added, few partitions are assigned to node to make distribution even.

↳ no of partitions are fixed and only

assignment changes.

→ partition size should be not too small OR not too large.

It's important that you come up with number after proper analysis.

management overhead

rebalancing on node failures is expensive.

③ Dynamic Partitioning

↳ Key range partitioned databases (HBase & RethinkDB) create partitions dynamically.

→ as partition grows and exceeds certain limit, it is split into two parts.

→ on data deletion, if required, partitions can be merged

④ Partitioning proportionally to Nodes

In #3 → no of partitions proportional to size of dataset.

In #2 → size of partition is proportional to size of dataset.

in cassandra → no of partitions proportional to no of nodes.

↳ fix no of partitions per node.

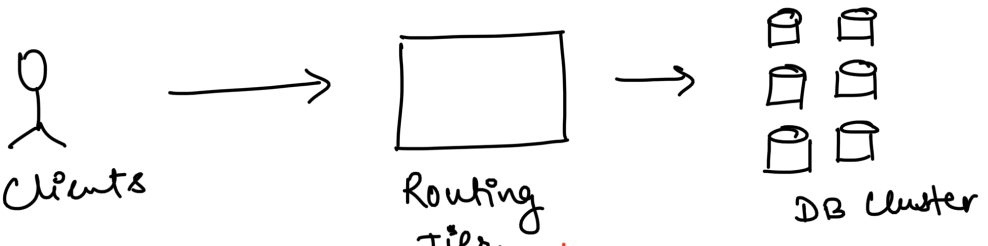
↳ Add new node → partitions becomes smaller as data is moved.

ⓐ randomly choose fix no of partitions to split

ⓑ After split, take ownership of half of them and leave other half in place.

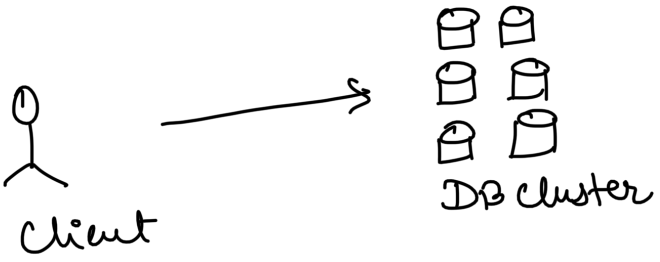
Request Routing → How will you serve read query as rebalancing happens and data moves from one node to another?

① route request to any node via round robin. If data not found, move to another node.

② 

determine which node should handle request & partition aware load balancer.

③ Clients are aware of partitioning and assignment of partitions to nodes.



How will you know assignment of partitions to nodes?

↳ Use coordination Service example Zookeeper.
↳ ex. HBase, Kafka

Keeps mapping of partitions to nodes.

↳ Use gossip protocol between nodes. eg. cassandra.

↳ Own routing tier. eg. couchBase use routing tier called moxi.

Subscribe for more such content.

YouTube Channel - MSDeep Singh

Instagram | LinkedIn | github - msdeep14

Happy Learning 🙂