Data-Intensive
Applications
THE BIG IDEAS BEHIND RELIABLE, SCALABLE, AND MAINTAINABLE SYSTEMS

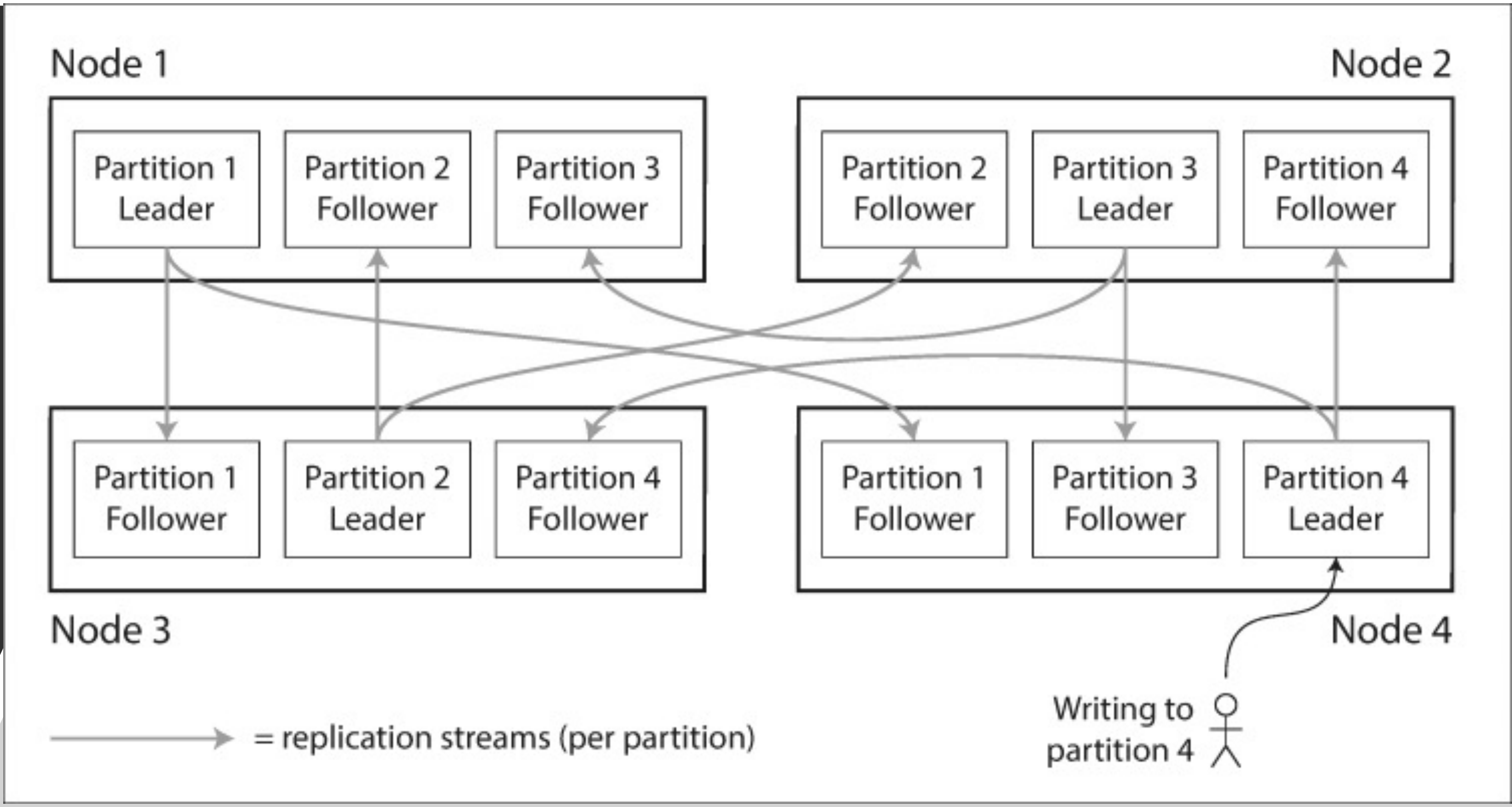
Chapter 6 : Partitioning

Partitioning

- Break data into partitions.
- Other names:
  - Shard
  - Partition
  - Region
  - Table
  - vnode
  - vBucket
- Each piece of data(record, row or document) belongs to exactly one partition.
- Each partition is a small database of its own.
- Why we need partitioning?
  - Scalability

Partitioning and Replication

- Copy of each partition is stored on multiple nodes.
- Each record belongs to one partition, but may be stored on multiple nodes for fault tolerance.
- A node may store more than one partition.



Partitioning of key-value data

- Spread the data and query load evenly across nodes - fair partitioning
- Unfair partitioning – Skewed
- Partition with disproportionately high load is a hot spot.
- How can we avoid it?
- Distribute data randomly
- But to read – query all nodes in parallel.

Partitioning by key-range

- Assign a continuous range of keys(from some minimum to maximum) to each partition.
- If we know boundaries between each range - we can determine partition.
- If we know which partition is assigned to which node - request directly to node.
- Partition boundaries might be manually chosen or automatically by database.
- Within each partition, keys are in sorted order(SSTables or LSM trees)
- Range scans are easy.
- Hot spots can occur, example one partition per day, so all writes end up in one partition in a day and others sit idle.

Partitioning by Hash of Key

- Risk of skew and hot spots – distributed datastores use a hash function.
- Hash function uniformly distributes skewed data.
- Assign a partition – range of hashes(rather than range of keys)
- Boundaries can be evenly spaced or chosen pseudo randomly(called consistent hashing).
- This is hash partitioning rather than consistent hashing.
- Difficult to do range queries.
- Range queries need to be sent to all partitions.

Skewed workloads and relieving hot spots

- Hashing a key reduces hot spots.
- When all reads and writes are for same key – all requests being routed to same partition.
- A celebrity user with millions of followers - hash of key doesn't help as hash of two identical ids are same.
- If a key is hot key – add a random number to beginning or end of key.
- But reads – have to do additional work.
- Thus append random number to only a small number of hot keys.

Partitioning and Secondary indexes

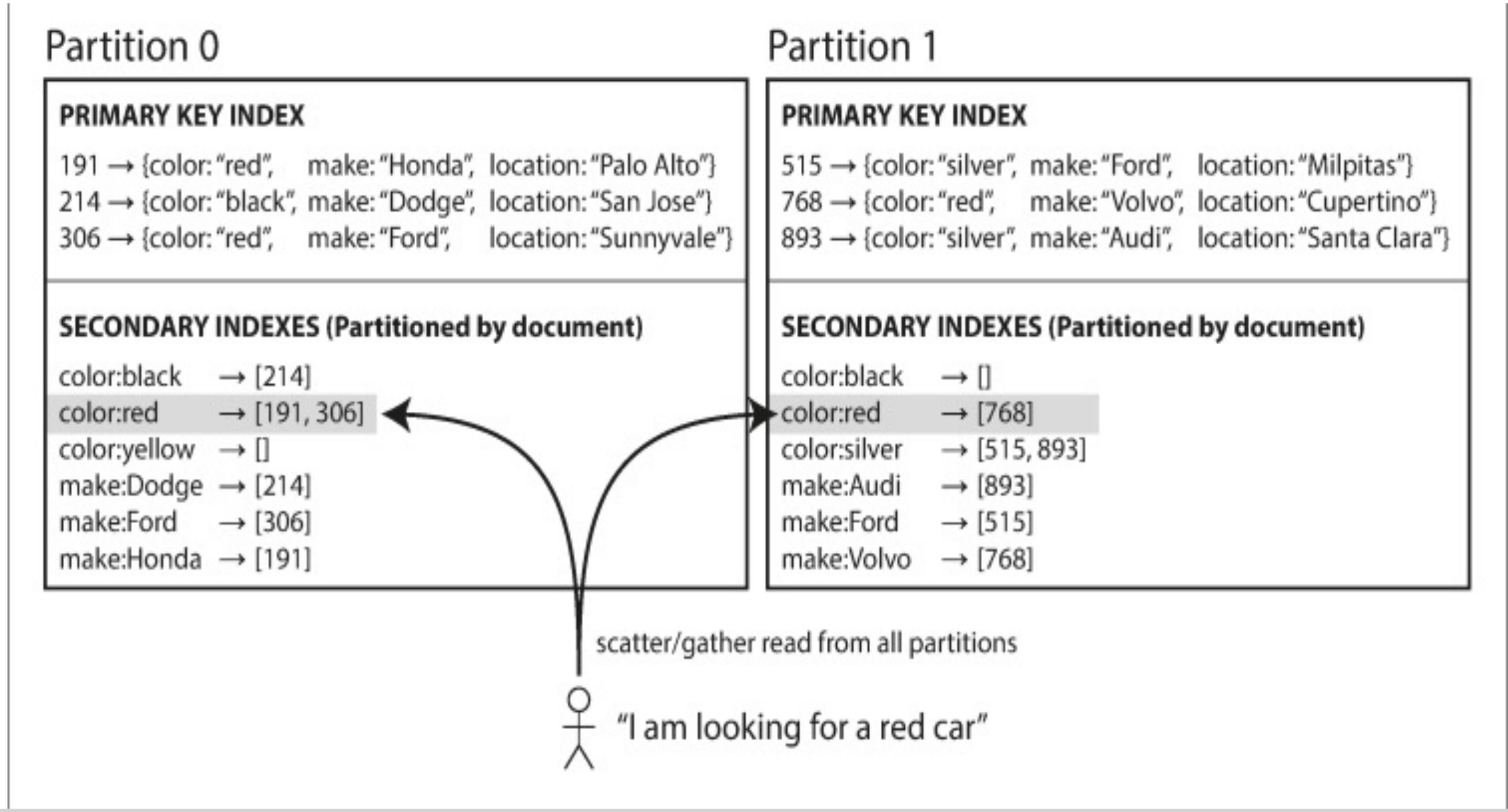
Secondary index – Doesn’t identify a record uniquely but searches on a particular value.

PARTITIONING SECONDARY INDEXES BY DOCUMENT

PARTITIONING SECONDARY INDEXES BY TERM

Partitioning Secondary Indexes by Document

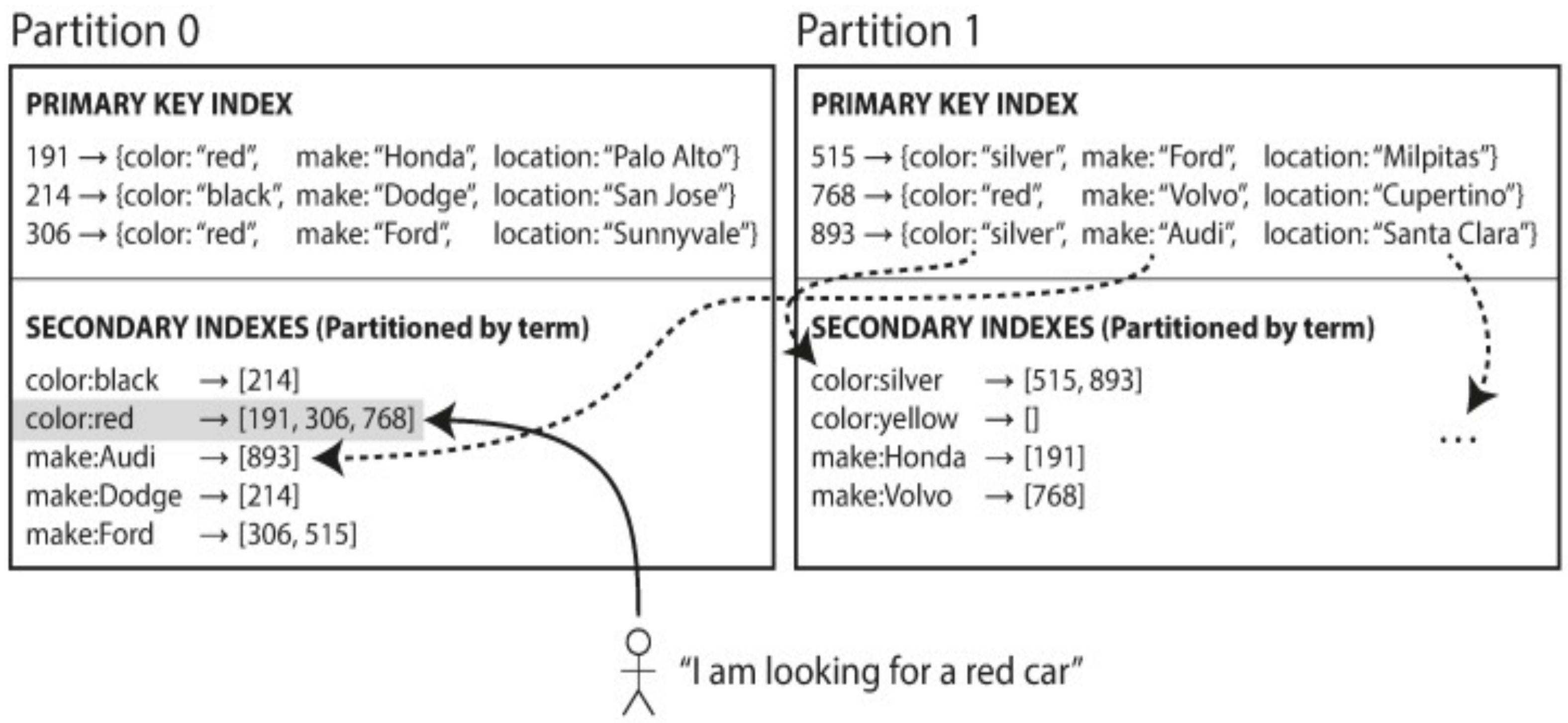
- Document-partitioned index is known as local index
- Query all the partitions and combine all the results you get back –Scatter/gather.
- Prone to tail latency amplification
- If possible – secondary index queries can be served from single partition.



Partition 0
PRIMARY KEY INDEX
191 → {color: "red", make: "Honda", location: "Palo Alto"}
214 → {color: "black", make: "Dodge", location: "San Jose"}
306 → {color: "red", make: "Ford", location: "Sunnyvale"}
SECONDARY INDEXES (Partitioned by document)
color:black → [214]
color:red → [191, 306]
color:yellow → []
make:Dodge → [214]
make:Ford → [306]
make:Honda → [191]
Partition 1
PRIMARY KEY INDEX
515 → {color: "silver", make: "Ford", location: "Milpitas"}
768 → {color: "red", make: "Volvo", location: "Cupertino"}
893 → {color: "silver", make: "Audi", location: "Santa Clara"}
SECONDARY INDEXES (Partitioned by document)
color:black → []
color:red → [768]
color:silver → [515, 893]
make:Audi → [893]
make:Ford → [515]
make:Volvo → [768]
scatter/gather read from all partitions
"I am looking for a red car"

Partitioning Secondary Indexes by Term

- Rather than a local index, we can construct a global index – that covers data in all partitions.
- But a global index must also be partitioned – but partitioned differently from primary key index.
- Useful than document-partitioned index – clients needs to make request to partition containing the term, rather than scatter/gather.
- Writes are slower – write to single document may affect multiple partitions of the index.
- Updates to global secondary indexes are often asynchronous.



Partition 0

PRIMARY KEY INDEX

```
191 → {color: "red",    make: "Honda",  location: "Palo Alto"}
214 → {color: "black",  make: "Dodge",  location: "San Jose"}
306 → {color: "red",    make: "Ford",   location: "Sunnyvale"}
```

SECONDARY INDEXES (Partitioned by term)

```
color:black  → [214]
color:red    → [191, 306, 768]
make:Audi    → [893]
make:Dodge   → [214]
make:Ford    → [306, 515]
```

Partition 1

PRIMARY KEY INDEX

```
515 → {color: "silver", make: "Ford",   location: "Milpitas"}
768 → {color: "red",    make: "Volvo",  location: "Cupertino"}
893 → {color: "silver", make: "Audi",   location: "Santa Clara"}
```

SECONDARY INDEXES (Partitioned by term)

```
color:silver  → [515, 893]
color:yellow  → []
make:Honda    → [191]
make:Volvo    → [768]
...
```

Rebalancing Partitions

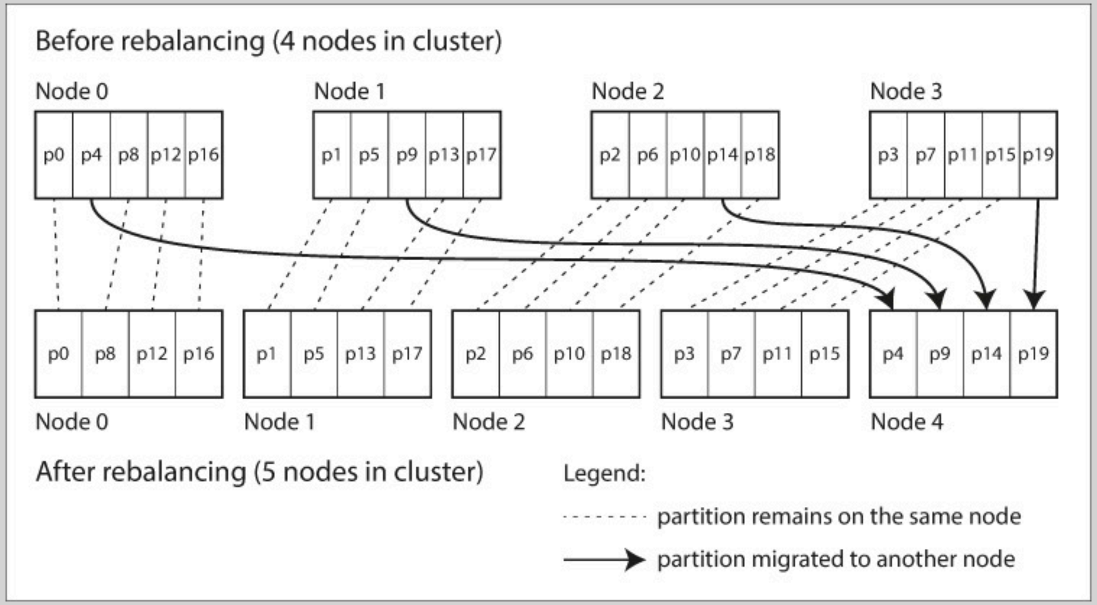
- The query throughput increases, so add more CPUs to handle load.
- Dataset size increases, add more disks and RAM to store it.
- A machine fails, and other machines takes over the failed machine's responsibilities.
- Request and data need to be moved from one node to another in the cluster – rebalancing.
- Minimum requirements of rebalancing:
- Data storage, read and write requests should be shared fairly between nodes in cluster.
- During rebalancing, database should continue accepting reads and writes.
- Not more than necessary data should be moved between nodes to make the rebalancing fast.

Strategies for Rebalancing

- Hash mod N
  - Hashes into ranges
  - 0 <= hash(key) < b0 – partition 1
  - b0 <= hash(key) < b1 – partition 2
  - Hash mod N – problem – if number of nodes N changes, most of the keys would need to be moved from one node to another.

Strategies for Rebalancing: Fixed number of partitions

- Create more partitions than nodes, assign several partitions to each node.
- If new node is added, it can steal a few partitions from existing node until partitions are fairly distributed once again. If node is removed – same thing happens.
- Only partitions are moved between nodes. Number of partitions does not change nor assignment of keys to partitions. Only thing change is the assignment of partitions.



Strategies for Rebalancing: Dynamic Partition

- Key range partitioning, fixed number of partitions – inconvenient
- We could end up with all of data in one partition and other partitions empty.
- Dynamically partitioning – When a partition grows to exceed a configured size, it is split into 2 partitions so that approximately half of data ends up on each side of the split.
- Or partition can also be merged – if lots of data gets deleted.
- Advantage: Number of partitions adapts to total data volume.
- Empty database – starts with a single partition, we can allow an initial set of partitions to be configured on an empty database – Pre-splitting

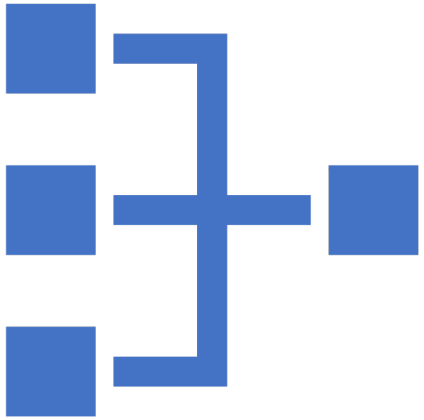
Strategies for Rebalancing: Partitioning proportionally to nodes

- Dynamic partition – number of partitions proportional to size of dataset.
- Fixed partition – size of each partition proportional to size of dataset.
- Fixed number of partitions per node – size of each partition proportional to size of dataset while number of nodes remains unchanged.
- When number of nodes increase, partitions become smaller, it randomly choses a fixed number of existing partitions to split and takes ownership of one half of those split.

Operations: Automatic or Manual rebalancing

- System can decide automatically when to move partitions from one node to another without any admin interaction
- Explicitly configured by an admin and only changes when the admin explicitly reconfigures it.
- Fully automated rebalancing is convenient because less operational work for normal maintenance but its unpredictable.
- Rebalancing is expensive - rerouting requests and move data from one node to another.
- Can result in cascading failures if rebalancing is automatic with automatic failure detection

Request Routing



- How does a client know which node to connect to?
- As partitions get rebalanced, assignment of partitions to nodes changes. This problem is called service discovery.
- Allow clients to contact any node, if it has the partition that contains data, it returns otherwise it forwards request to appropriate node, receives the reply and passes the reply to client.
- Sends all requests from client to routing tier first. It acts as partition-aware load balancer.
- Clients be aware of partitioning and assignment of partitions to nodes. Client can directly connect to node.
- But problem is how do they learn about changes in assignment of partitions to nodes?
- Many distributed data systems rely on separate coordination service called ZooKeeper to keep track of cluster metadata.
- Routing tier, partitioning aware-client cans subscribe to information in ZooKeeper.
- Gossip protocol – used among the nodes to transfer any changes in cluster state.

Summary

- ✓ Partitioning
- ✓ Types pf partitioning
- ✓ Partitioning and secondary indexes
- ✓ Rebalancing partitions
- ✓ Strategies for rebalancing
- ✓ Automatic or Manual rebalancing
- ✓ Request Routing

Thank You!