CSC 369

Introduction to Distributed Computing

Distributing Data

Two (complementary) approaches to distributing data across nodes:

- Replication
  - Keep an exact copy of data on multiple nodes
- Partitioning
  - Split a large dataset into smaller subsets, store each subset on a different node

Separate mechanisms, but often used together

Partitioning

When data or request volume becomes too large for a single node, we must break up the data into **partitions**

⚠️ In this course, we will use the term "partition"
Other terminology you may encounter:

- shard (MongoDB, Eliasticsearch)
- region (HBase)
- tablet (BigTable)
- vnode (Cassandra, Riak)
- vBucket (Couchbase)

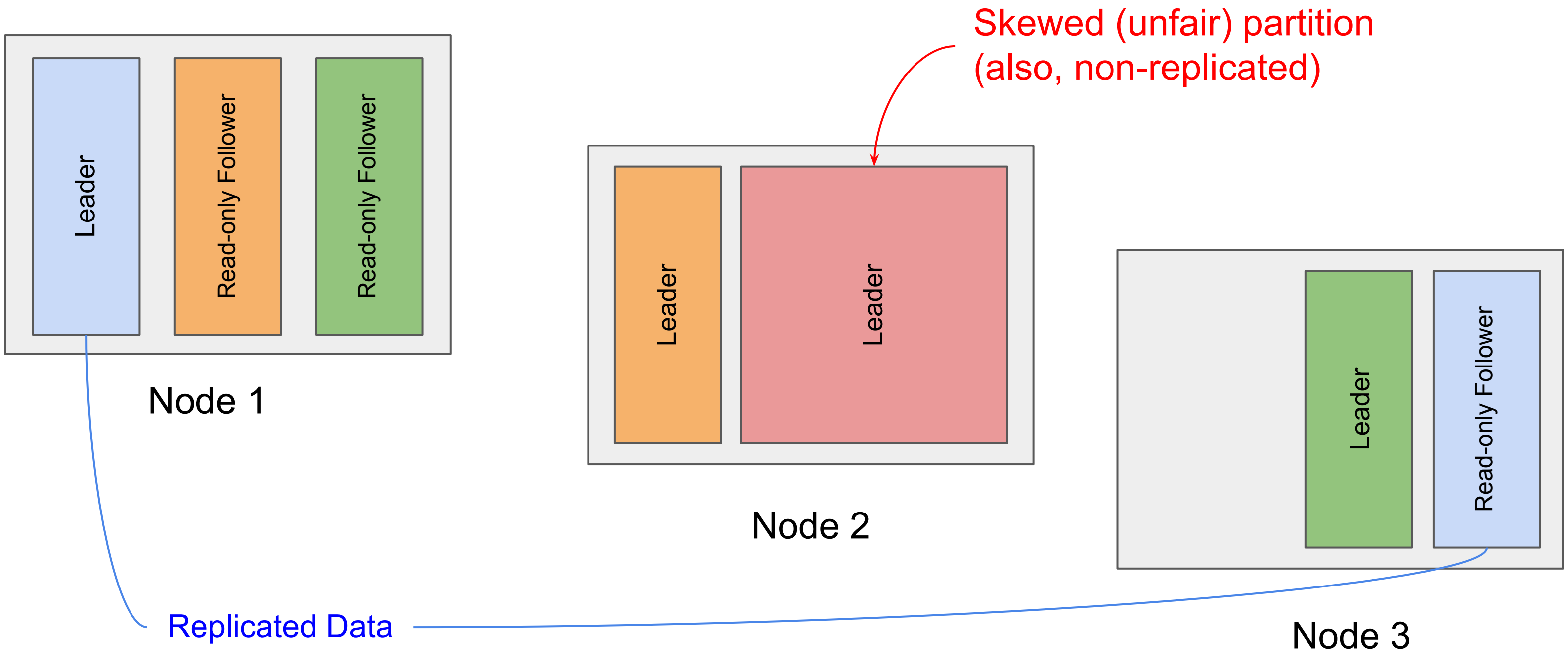
Partitioning

- Each piece of data (record, row, document) belongs to exactly *one* partition
- Different partitions placed on different nodes
- For *operations on single partitions*, each node independently operates on its own

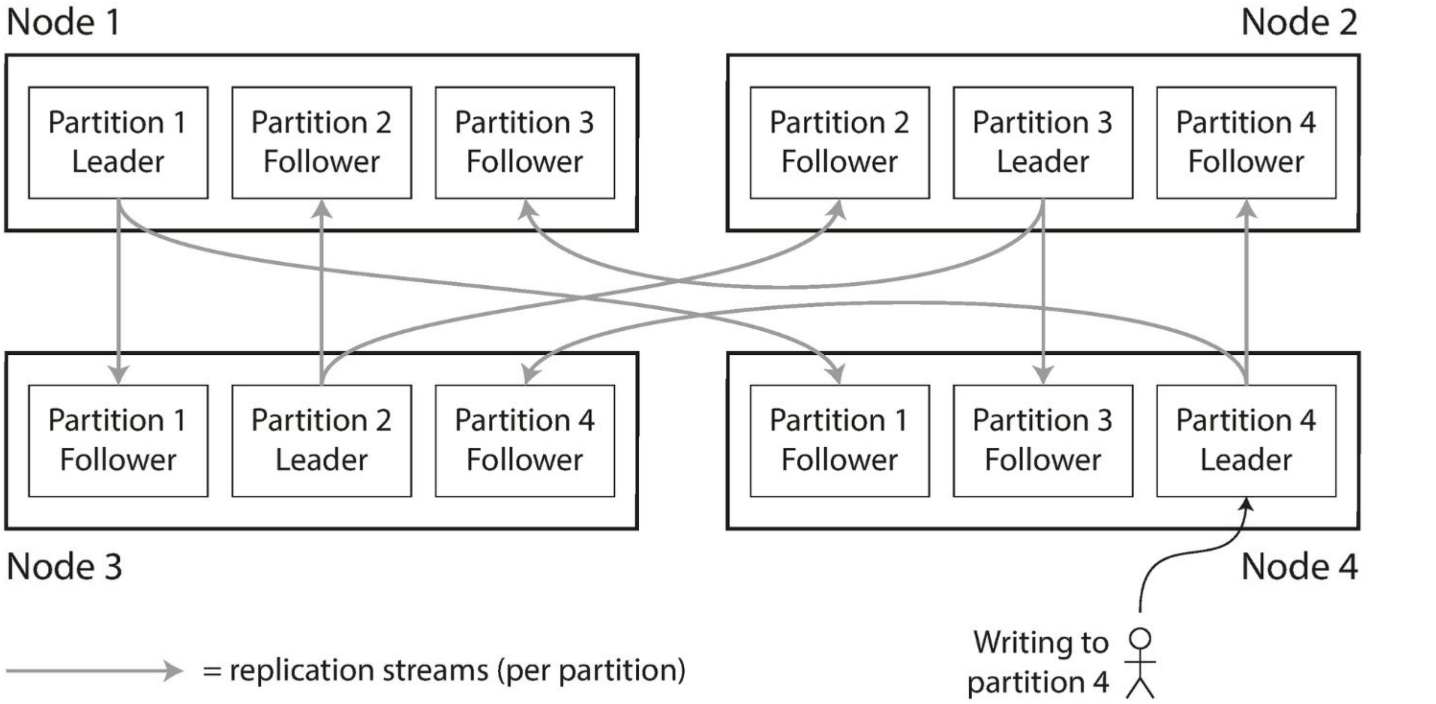
Partitioning Considerations

- Complex operations that must span multiple partitions
- Addition/removal of nodes
- Rebalancing partitions
- Routing requests to the correct partitions
- Even distribution of data and load

Partitioning and Replication



Partitioning and Replication



Partitioning Data

Goal: spread data and query load evenly

Strategies:

- Randomly assign data to partitions
- Assign a range to each partition
- Hash-based
- Parent-child interleaving

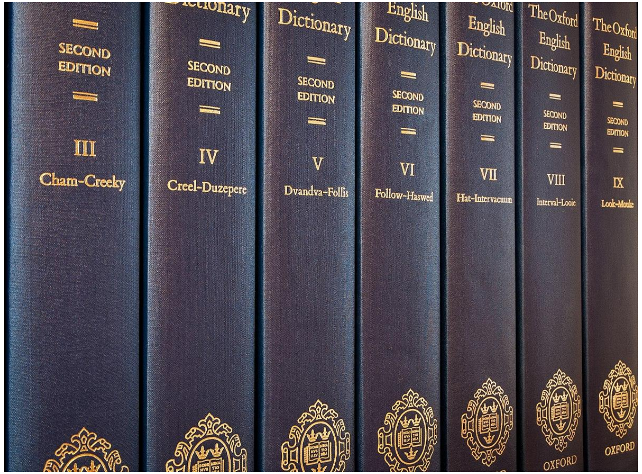
Partitioning Strategy - Random

Assign records to partitions records completely at random

Pros	Cons
simple	look up because it’s random
even if do it correctly	

Partitioning Strategy - Range-Based

Assume each record has a key. Assign a continuous range of keys to each partition. ex. year, name, etc.



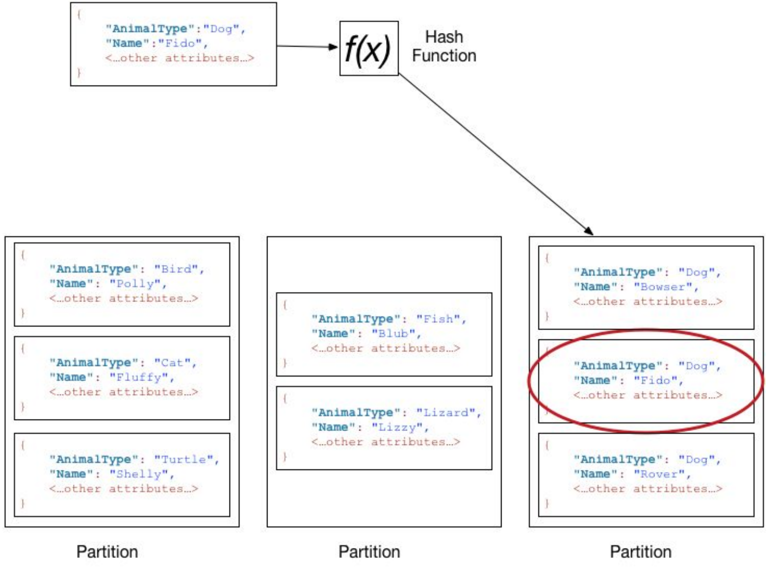
Partitioning Strategy - Range-Based

Assume each record has a key. Assign a continuous range of keys to each partition.

Pros	Cons
common data will be next to each other	might overload node if don’t know the range a head of time. example, some keys are more popular than others.
instantaneous look up by key name	

Partitioning Strategy - Hash-Based

Assume each record has a key. Apply a hash function to each key, assign each partition a range of hashes.

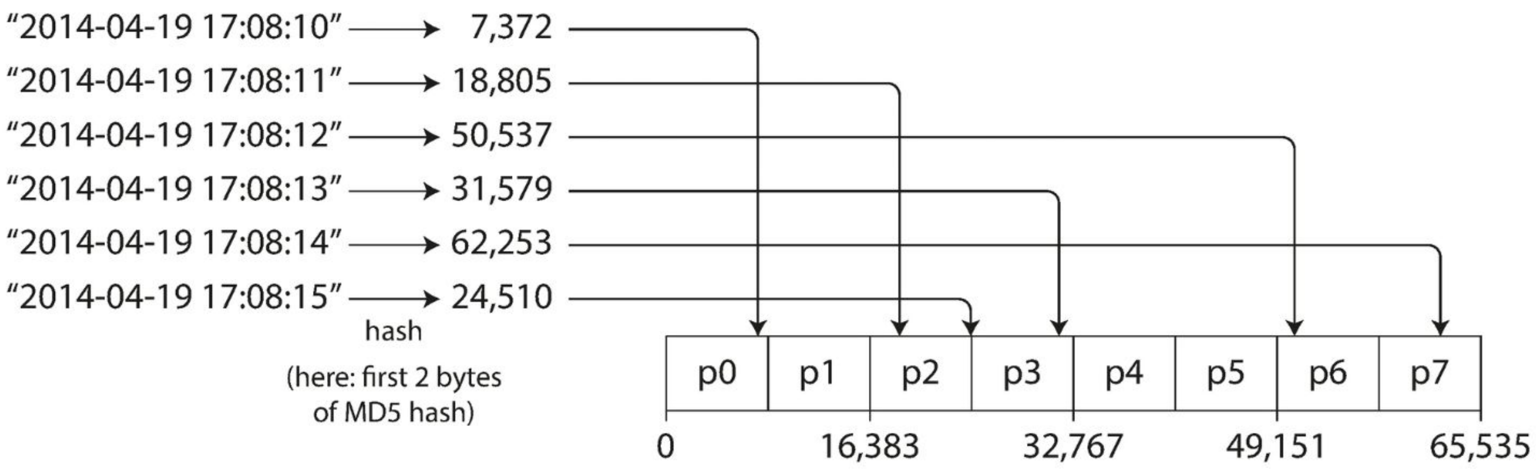


Partitioning Strategy - Hash-Based

Assume each record has a key. Apply a hash function to each key, assign each partition a range of hashes.

Pros	Cons
look up is constant	if the hash function is bad, it might lead to over loaded nodes
	search is not efficient
	in a range of data

Partitioning by Hash of Key



Partitioning Strategy - Interleaved

????????

Co-locate parent/child records (eg. user & user's events), to minimize cost of commonly-performed joins.

```
{ id: 123, email: "user@host" }
{ user: 123, event: "comment", tsamp: "..." }
{ user: 123, event: "delete", tsamp: "..." }
{ user: 123, event: "create", tsamp: "..." }
{ id: 234, email: "user234@host" }
{ user: 234, event: "comment", tsamp: "..." }
...
```

Partition 1

```
{ id: 456, email: "user456@host" }
{ user: 456, event: "create", tsamp: "..." }
{ user: 456, event: "comment", tsamp: "..." }
{ user: 456, event: "delete", tsamp: "..." }
{ id: 567, email: "user567@host" }
{ user: 567, event: "create", tsamp: "..." }
{ user: 567, event: "create", tsamp: "..." }
...
```

Partition 2

Partitioning and Indexes

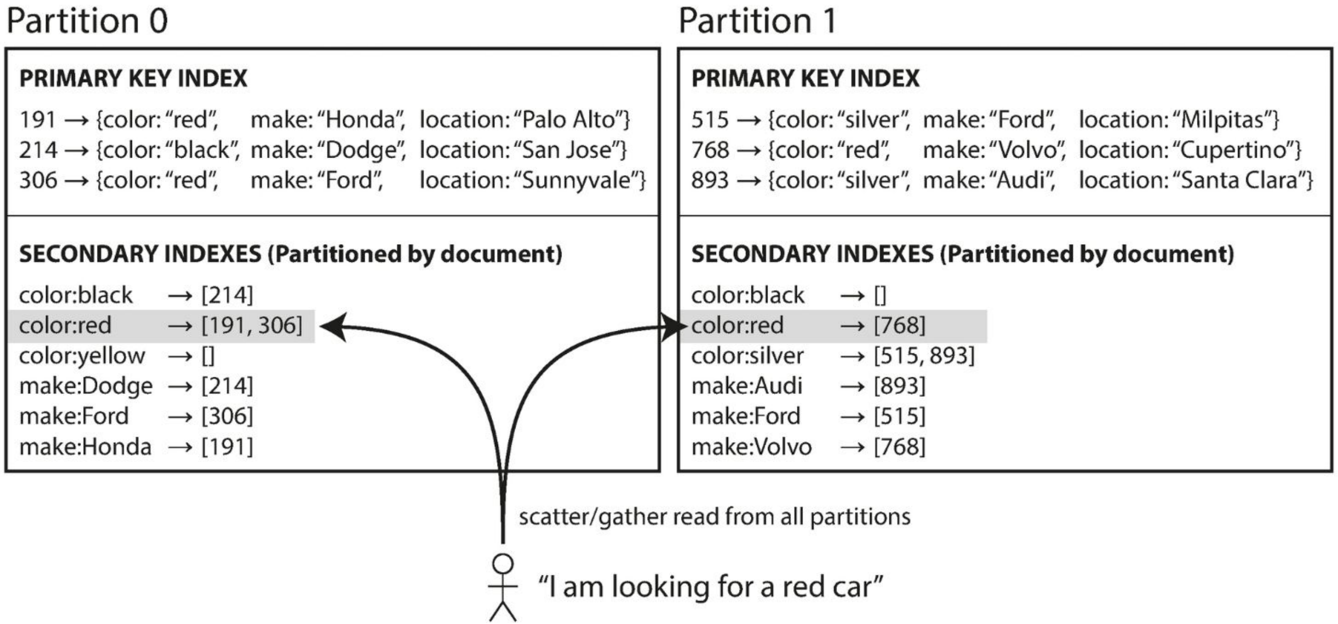
- (Simplistic) assumption thus far: all lookups are by key

- Support for secondary indexes often useful
  - For example: find cars by color/make/model
  - Key feature of relational databases, search tools (Elasticsearch, Solr)

- Problem: how to map secondary indexes to partitions?

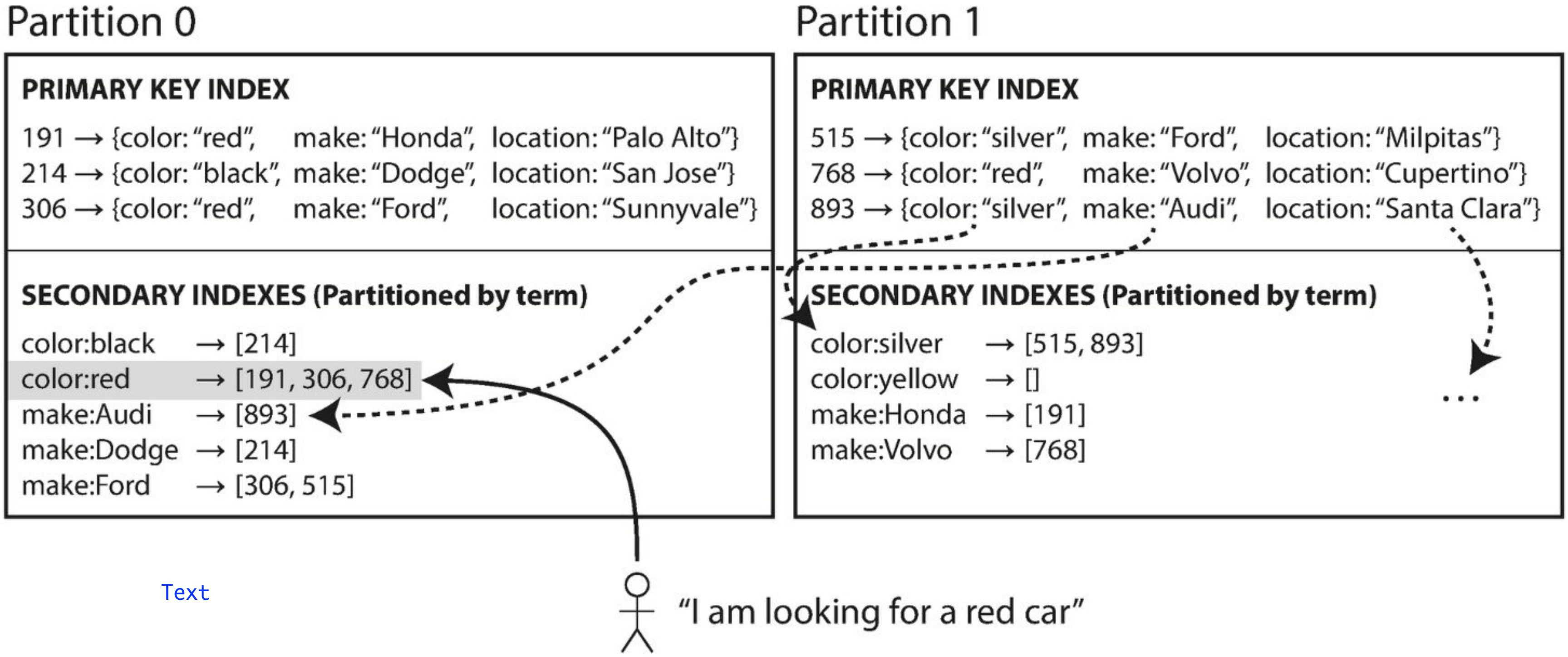
Secondary Index - Partitioned by *Document*

independence. write is very efficient



Secondary Index, Partitioned by *Term*

a write is written to multiple partition. it is not independence anymore



Text

Rebalancing Partitions

- Query load grows, need more CPUs
- Data volume grows, need more disk/RAM
- Node fails
- Need to move data from one node to another, a process called **rebalancing**

Rebalancing Considerations

- After rebalancing, load & data should be evenly distributed among nodes

- Uninterrupted availability during rebalancing process

- Minimize amount of data moved between nodes

Rebalancing Strategies

- hash(key) mod N (where N is the number of nodes)

- For example: 10 nodes, mod 10 returns 0-9
    - Assign to appropriate node

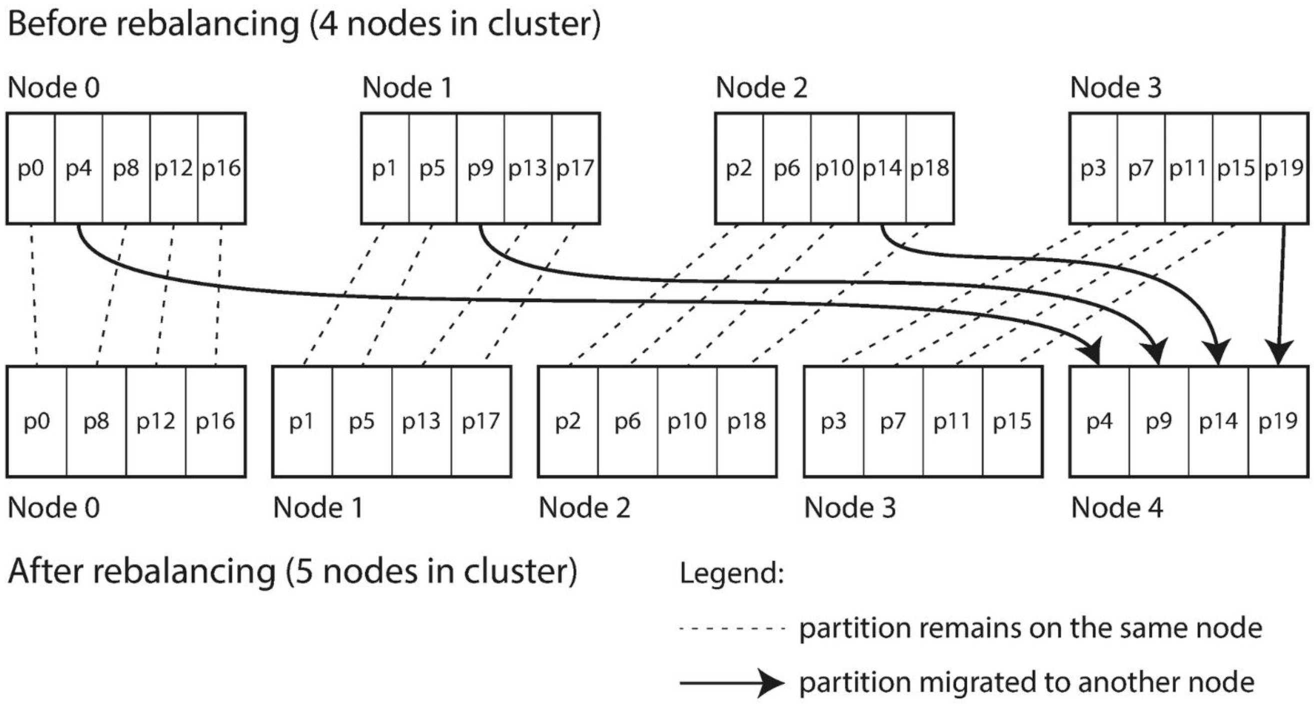
- Good strategy? assume even size and a good hash function

data transfer is not efficient because we have to move the whole thing

Rebalancing: Fixed Number of Partitions

- Create large number of partitions (many more than node count)
- Assign several partitions to each node
- New nodes can "steal" partitions from existing nodes
- If nodes are removed, partitions are divided between remaining nodes

Rebalancing Example



How Many Partitions?

- If each partitions holds a large amount of data, rebalancing and recovery are expensive
- If partitions hold very little data, overhead is high maintenance become a nightmare
- Ideal: "just right"

based on the situation to decide where the balance is

Dynamic Partitioning

- Fixed number of partitions / fixed boundaries not always convenient
- Some implementations begin with a single partition
- As a partition grows, it is split
  - One half transferred to another node
- Partition count adapts to data volume

good and common stratery

Rebalancing: Automatic or Manual?

- Decision to split or move partitions
  - Fully automatic? fully manual? somewhere in between?

cascading effect if something gone wrong even
when the system is taking care fo the rebalancing

- Rebalancing can be expensive
  - Need to take care to avoid cascading failures

- Human-assisted process can avoid issues

up the user to commit

Request Routing

How do we route client requests to the appropriate node?

1. Clients connect to any node, request forwarded as needed
2. Routing tier (partition-aware load balancer)
3. Clients aware of partitioning configuration

?????????

Key issue: propagating change information (new partitions, splits)
All participants must agree

Replication / Partitioning Summary

- Replication and Partitioning
  - Distribute data across nodes for scaling, latency, or availability reasons
  - Two separate approaches, often used together

- Automatic or Manual?
- Managing changing data (writes)
- Increase/decrease number of nodes