Distributed Data - Partitioning

UA.DETI.CBD
José Luis Oliveira / Carlos Costa

1

Main topics

- Concept and objective
- Partitioning approaches for large datasets
- Indexing of data with partitioning
- Rebalancing (add or remove nodes)
- Routing of query requests

2

2

Partitioning

- **What?**
  - splitting a big database into smaller subsets called partitions, so that different partitions can be assigned to different nodes (also known as sharding)
- **Why?**
  - for very large datasets or very high query throughput
- **How?**
  - Usually, partitions are defined in such a way that each piece of data (each record, row or document) belongs to exactly one partition
  - Each partition is a small database of its own
  - Database may support operations over multiple partitions at the same time
- **Terminology**
  - shard in MongoDB, Elasticsearch and SolrCloud, a region in HBase, a tablet in BigTable, a vnode in Cassandra and Riak, and a vBucket in Couchbase

UNIVERSIDADE DE AVEIRO 3

3

Partitioning

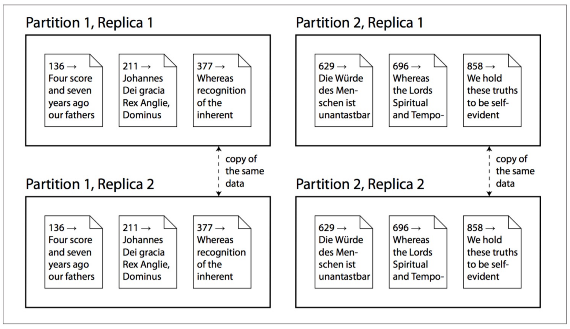
- The main reason for partitioning data is **scalability**
  - A large dataset can be spitted across many machines
  - Distinct partitions can be placed on different nodes in a shared-nothing cluster
- **Query load** can be also distributed
  - Small/regular queries operate on a single partition – each node can independently execute the queries for its own partition
  - Large/complex queries can potentially be parallelized across many nodes
- **Query throughput** can be scaled by adding more nodes

UNIVERSIDADE DE AVEIRO 4

4

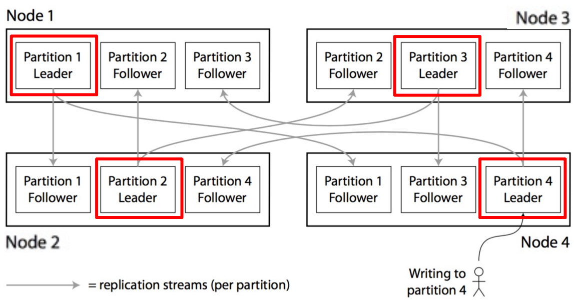
Partitioning and Replication

- Partitioning is usually combined with replication
  - a node may store more than one partition
  - one partition may be stored on several different nodes
- Example: A database split into two partitions, with two replicas per partition



Leader-follower replication model

- Each partition's leader is assigned to one node and its followers are assigned to other nodes
- Each node may be the leader for some partitions, and a follower for other partitions
- Example:
  - 4 partitions
  - 3 replicas



Note: the choice of partitioning scheme is mostly independent of the choice of replication scheme, so we will ignore replication in this module

Data partitioning

Original Table

CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
1	TAEKO	OHNUKI	BLUE
2	O.V.	WRIGHT	GREEN
3	SELDA	BAĞCAN	PURPLE
4	JIM	PEPPER	AUBERGINE

Vertical Partitions

VP1

CUSTOMER ID	FIRST NAME	LAST NAME
1	TAEKO	OHNUKI
2	O.V.	WRIGHT
3	SELDA	BAĞCAN
4	JIM	PEPPER

VP2

CUSTOMER ID	FAVORITE COLOR
1	BLUE
2	GREEN
3	PURPLE
4	AUBERGINE

Horizontal Partitions

HP1

CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
1	TAEKO	OHNUKI	BLUE
2	O.V.	WRIGHT	GREEN

HP2

CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
3	SELDA	BAĞCAN	PURPLE
4	JIM	PEPPER	AUBERGINE

Data partitioning

- **Challenge**: Partition of large amount of data
- **Goal**: spread the data and the query load across nodes … in a uniform way
- **Problems**:
  - skewed: some partitions have more data or serve a greater number of queries than others
  - hot spot: when one partition has disproportionately high load (skewed extreme case)
- **Solution**:
  - Algorithms to split records across nodes

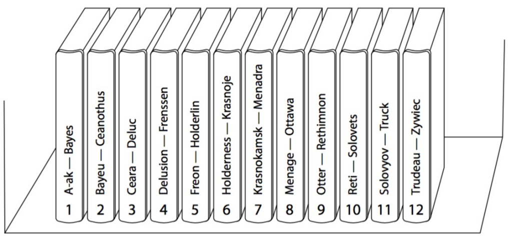
Partitioning of key-value data

- Assumption
  - assume to have a key-value data model
  - always **access a record by its primary key**
- Approaches
  1. partitioning by key range
  2. partitioning by hash of key
- Skewed workloads

9

1. Partitioning by key range

- Assign a continuous range of keys to each partition
  - know the boundaries between the ranges
  - boundaries chosen automatically or manually
  - easily determine which partition contains a given key
  - can make requests directly to the appropriate node
- Example: encyclopedia volumes



10

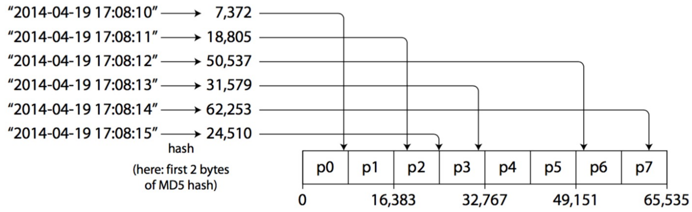
1. Partitioning by key range (cont.)

- Ranges of keys are not necessarily evenly spaced
  - data may not be evenly distributed
    - encyclopedia volume 1 contains words starting with A and B, but volume 12 contains words starting with T, U, V, X, Y and Z
- Within each partition, keys can be sorted
  - Allowing range queries
- Key can be a concatenated index in order to fetch several related records in one query
  - e.g., a timestamp 'year-month-day-hour-minute-second'
- Key range **downside**:
  - certain access patterns can lead to hot spots

11

2. Partitioning by hash of key

- Hash function to determine the partition for a key
  - reduce the risk of skew and hot spots
  - used by many distributed datastores, e.g., Cassandra and MongoDB
- A good hash function takes skewed data and makes it uniformly distributed
- Example: a hash function takes a string and returns a 32-bit integer



12

2. Partitioning by hash of key (cont.)

- Problem – no order of keys
  - lost of efficient range queries
- **MongoDB** with hash-based sharding enabled
  - range query are sent to all partitions
- Riak, Couchbase and Voldemort
  - range queries on the primary key are not supported
- **Cassandra** - compromise between two strategies
  - a table can be declared with a compound primary key consisting of several columns.
  - Only the first part of that key is hashed (partition key) to determine the partition,
    - the other columns (clustering key) are used as a concatenated index for sorting the data in SSTables
  - can search by a fixed value for the first column and perform an efficient range scan based on the other columns of the key
  - allows an elegant data model for one-to-many relationships

UNIVERSIDADE DE AVEIRO 13

13

Skewed workloads

- Hashing a key cannot avoid hot spots entirely
  - in a extreme case, all reads and writes are for the same key. So, all requests will be routed to the same partition
  - this kind of workload is unusual but not impossible
    - for example, a celebrity in a social media site with millions of followers may cause a storm of activity when they do something
- Most data systems are not able to automatically detect and compensate for such a highly skewed workload
- Responsibility of the application to reduce the skew

UNIVERSIDADE DE AVEIRO 14

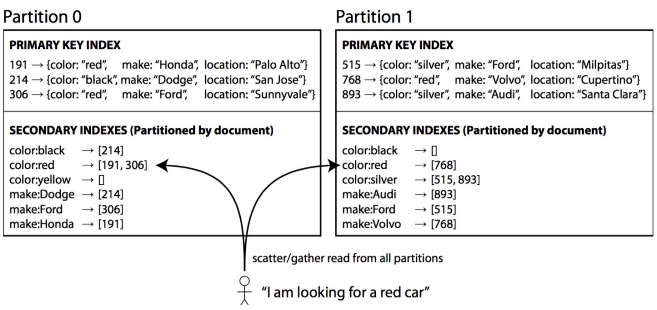
14

Partitioning and secondary indexes

- Records are accessed via primary key that allow to determine the right partition to read and write
- Problem… secondary indexes
- DB Scenario
  - well supported by relational databases and common in document databases
  - Many key-value stores avoid secondary indexes because of their added implementation complexity
- Two main approaches to partitioning a database with secondary indexes:
  - **document-based partitioning**
  - **term-based partitioning**

1. Document partitioned index

- Characteristics:
  - index per partition
  - known as a local index
- Example: database for selling used cars
  - cars unique ID (document ID) used for partitioning the DB
  - secondary index: search by color and by make



1. Document partitioned index (cont.)

- Each partition operates in a separate way
- **Writing** (add, remove or update) – only need to deal with the partition that contains a document ID
  - each partition maintains its own secondary indexes, covering only the documents in that partition
- **Reading** – requires to send the query to all partitions, and combine all the results obtained
  - this querying approach is known as scatter/gather
  - expensive process even if you query the partitions in parallel, is prone to latency increasing
- Used by MongoDB, Riak, Cassandra, Elasticsearch, SolrCloud,..

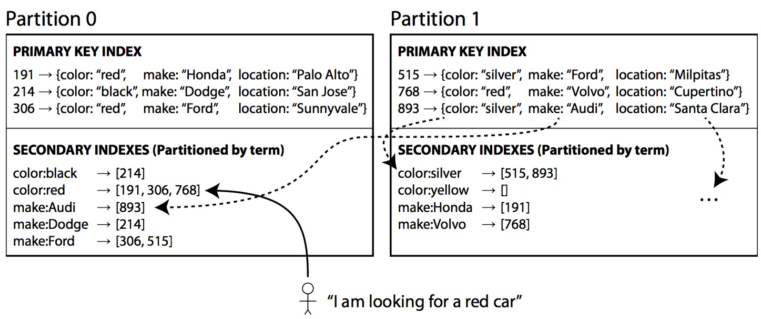
UNIVERSIDADE DE AVEIRO

17

17

2. Term-based index

- Characteristics:
  - a global index that covers data in all partitions
  - partitioned differently from the primary key index
    - store the index in only one node become a bottleneck and against the purpose of partitioning



UNIVERSIDADE DE AVEIRO

18

18

2. Term-based index (cont.)

- Partitioning sec. index by term or by its hash is also possible with (dis)advantages discussed before
- Advantage of term-partitioned index over document-partitioned index is that it can make reads more efficient
  - client only needs to make a request to the partition containing the term
- Downside of a global index is that writes are slower and more complicated
  - write a document may now affect multiple partitions
- Updates to global secondary indexes are often asynchronous
  - if we read the index shortly after a write, the change may not yet be reflected in the index

UNIVERSIDADE DE AVEIRO 19

19

Rebalancing partitions

- **Things that change** in a database over time:
  - query throughput increases => add more CPUs to handle the load
  - dataset size increases => add more disks and RAM to store it
  - a machine fails => other machines assumes its responsibilities
- **Rebalancing**: the process of moving data (partitions) between nodes in the cluster
- Independently of partitioning scheme used, rebalancing has usually some **minimum requirements**:
  - while rebalancing, the database should continue operating (accepting reads and writes)
  - after rebalancing, the load (data storage, read and write requests) should be shared fairly between all nodes
  - don't move more data than necessary between nodes, to avoid overloading the network

UNIVERSIDADE DE AVEIRO 20

20

Rebalancing

- **Scenario**: partitioning by the hash of a key
  - assign *key* to partition 0 if $0 \leq hash(key) < b0$
  - assign *key* to partition 1 if $b_0 \leq hash(key) < b_1$
  - etc.
- **Strategy**: Round-robin 'hash(key) mod N'
  - N is the number of nodes
    - for example, N=10 returns a number between 0 and 9
- But this strategy has major **drawbacks**
  - when the number of nodes N changes, most of the keys would need to be moved from one node to another
  - makes rebalancing excessively expensive

UNIVERSIDADE DE AVEIRO 21

21

Rebalancing – fixed nº of partitions

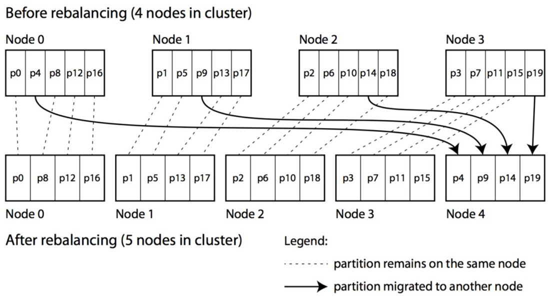
- **Scenario**: partitioning by the hash of a key
- **Approach**: number of partitions is fixed
  - what changes is the assignment of partitions to nodes
  - only entire partitions are moved between nodes
- More partitions than nodes
  - assign several partitions to each node
- Example
  - database running on a cluster of 10 nodes may be split into 1,000 partitions (100 partitions assigned to each node)
- Used by Riak, Cassandra, Elasticsearch, Couchbase, ..

UNIVERSIDADE DE AVEIRO 22

22

Rebalancing – fixed nº of partitions

- **Adding a new node**
  - steals partitions from every existing node, until partitions are fairly distributed once again



- **Removing a node**
  - add reverse process

Rebalancing – Dynamic partitioning

- **Scenario**: key range partitioning
  - problem: skew and hot spots
- **Approach**: create partitions dynamically
- Partition split
  - when partition grows and exceed a configured size
  - split into two partitions ~ half of the size
  - after split, a partition can be transferred to another node
- Partition merge
  - when lots of data is deleted, it can be merged with an adjacent partition
- Used in databases such as HBase and RethinkDB

Rebalancing – Dynamic partitioning

- **Advantage**: number of partitions adapts to data volume
- **Limitation**: when the dataset is small, it works with a single partition. So, all writes are processed by a single node while the other nodes sit idle
  - to mitigate this, HBase and MongoDB allow the setup of an initial set of partitions on an empty database (pre-splitting)
- Dynamic partitioning can also be used with hash-partitioned data
  - MongoDB supports both key-range and hash partitioning, but it splits partitions dynamically in either case

Rebalancing – automatic or manual?

- Rebalancing is an expensive operation
  - requires re-routing requests and moving a large amount of data between nodes
  - it can overload the network or the nodes, and cause performance problems for other systems
- Dangerous in combination with automatic failure detection - cascading failure!
- Fully automated rebalancing:
  - convenient: less operational/maintenance work
  - downside: can have unpredictable results
- Recommendation: human should control the process.
  - slower process but can help to prevent operational surprises
  - Couchbase, Riak generate a suggested partition assignment automatically, but require an administrator commit

Routing process

- We have now partitioned our dataset across multiple nodes running on multiple machines.
- But there remains an open question:
  - when a client wants to make a request, how does it know which node to connect to?
- The problem increase as partitions are rebalanced
  - the assignment of partitions to nodes changes.
- **Service discovery**
  - Allow clients to contact any/all node
    - e.g. via a round-robin load balancer
  - Clients are aware of the partitioning and the assignment of partitions to nodes.
    - can connect directly to the appropriate node, without any intermediary.
  - Send all requests from clients to a **routing tier first**
    - which determines the node that should handle the request and forwards it accordingly.

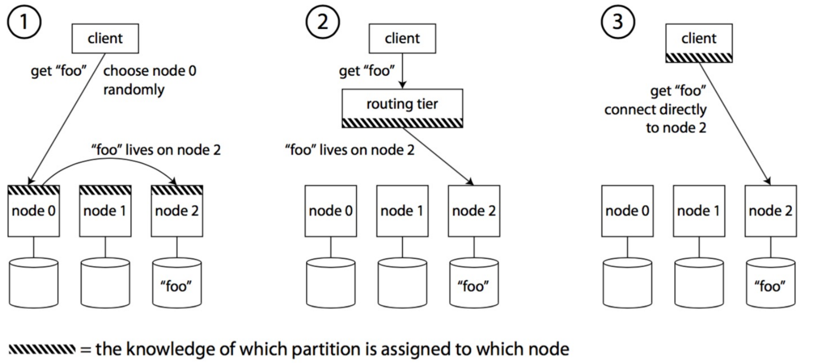
UNIVERSIDADE DE AVEIRO

27

27

Routing process - approaches

- Examples of different ways of routing a request to the right node (1. node; 2. routing tier; 3. client)



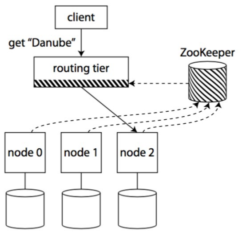
UNIVERSIDADE DE AVEIRO

28

28

Routing service

- Protocols for achieving consensus in a distributed system are hard to implement correctly
- Many distributed data systems rely on a **separate coordination service** to keep track of cluster metadata



Key ranges	Partitions	Nodes	IP addresses
A-ak — Bayes	partition 0	node 0	10.20.30.100
Bayeu — Ceanothus	partition 1	node 1	10.20.30.101
Ceara — Deluc	partition 2	node 2	10.20.30.102
Delusion — Frenssen	partition 3	node 0	10.20.30.100
Freon — Holderlin	partition 4	node 1	10.20.30.101
Holderness — Krasnoje	partition 5	node 2	10.20.30.102
Krasnokamsk — Menadra	partition 6	node 0	10.20.30.100
Menage — Ottawa	partition 7	node 1	10.20.30.101
Otter — Rethimnon	partition 8	node 2	10.20.30.102
Reti — Solovets	partition 9	node 0	10.20.30.100
Solovyov — Truck	partition 10	node 1	10.20.30.101
Trudeau — Zywiec	partition 11	node 2	10.20.30.102

= the knowledge of which partition is assigned to which node

UNIVERSIDADE DE AVEIRO

Routing solutions

APACHE ZooKeeper

- **Apache ZooKeeper**
  - HBase, SolrCloud and Kafka use ZooKeeper to track partition assignment
- **MongoDB** has a similar architecture
  - but relies on its own config server implementation and mongos daemons as routing tier
- **Cassandra** and Riak take a different approach
  - Similar to the node approach, (1) of previous figure
  - gossip protocol among the nodes to disseminate and agree on any changes in cluster state
  - requests can be sent to any node, and that node forwards them to the appropriate node for the requested partition
  - disadvantages: more complexity in the database nodes
  - advantages: avoids the dependency on an external service
- Some DBs (i.e. Couchbase) do not rebalance automatically
  - which simplifies the agreement protocol between nodes

UNIVERSIDADE DE AVEIRO

Summary

- Explored different ways of partitioning a large dataset into smaller subsets
- The main goal of partitioning is to spread the data and the query load evenly across multiple machines, avoiding hot spots
- Requires choosing a partitioning scheme and rebalancing the partitions from time to time as nodes are added or removed from the cluster
- Two main approaches to partitioning:
  - Key range partitioning
  - Hash partitioning

UNIVERSIDADE DE AVEIRO

31

31

Summary

- In partitioning by hash, it is common to:
  - create a fixed number of partitions in advance
  - assign several partitions to each node
  - move entire partitions from one node to another when nodes are added or removed
- Hybrid approaches are also possible, for example:
  - using one part of the key to identify the partition, and another part for the sort order
- Partitioning and secondary indexes. A secondary index also needs to be partitioned:
  - Document-partitioned index
  - Term-partitioned index (global index
- Techniques for routing queries to the appropriate partition

UNIVERSIDADE DE AVEIRO

32

32

Resources

- Martin Kleppmann, ***Designing Data-Intensive Applications***, O'Reilly Media, Inc., 2017.
  - Chapter 6

  - https://docs.mongodb.com/manual/sharding/
  - https://opensource.com/article/20/5/apache-cassandra
  - https://cassandra.apache.org/doc/latest/data_modeling/data_modeling_refining.html
  - https://cassandra.apache.org/doc/latest/data_modeling/intro.html