Cassandra

Overview of Cassandra

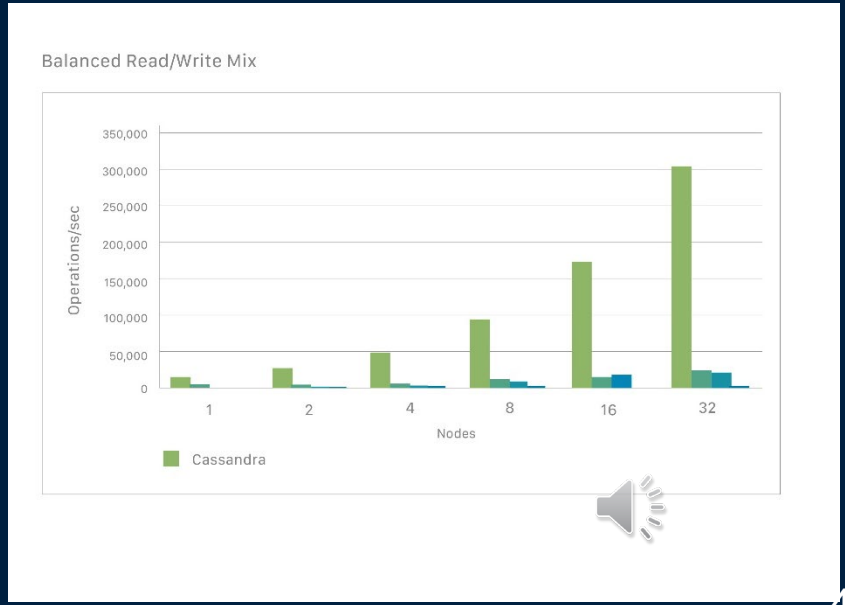
- NoSQL distributed database
- Lightweight, Open-source, Non-relational
- Largely Distributed, column based, scalable
- Its distribution design is based on Amazon's Dynamo and its data model on Google's Bigtable, yields both technical and business advantages

Overview of Cassandra

- Easily scale under high stress
- Prevents data loss from hardware failure
- Could run on multiple machines: maximize benefit out of Cassandra
- Usually has multiple nodes, with each representing a single instance
- Master-less architecture
- Nodes can be organized into cluster

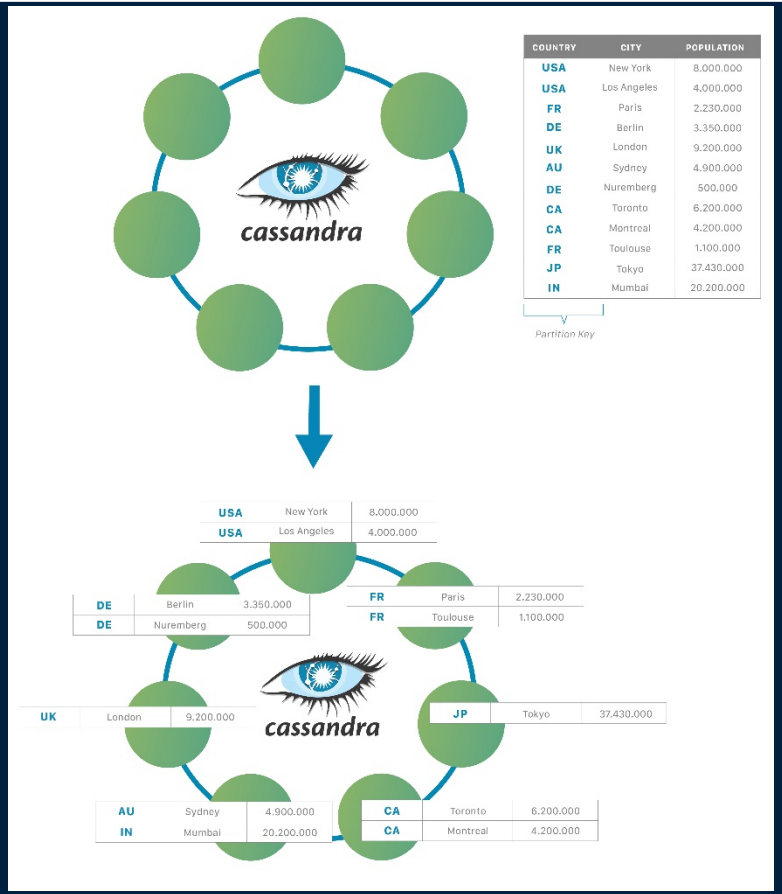
Overview of Cassandra

- Enables developers to scale their databases dynamically
- Since it is node-based, Cassandra scales horizontally using lower commodity hardware
- Double the number of nodes will double your capacity / throughput
- Also have the flexibility to scale back



Source:https://cassandra.apache.org/doc/latest/

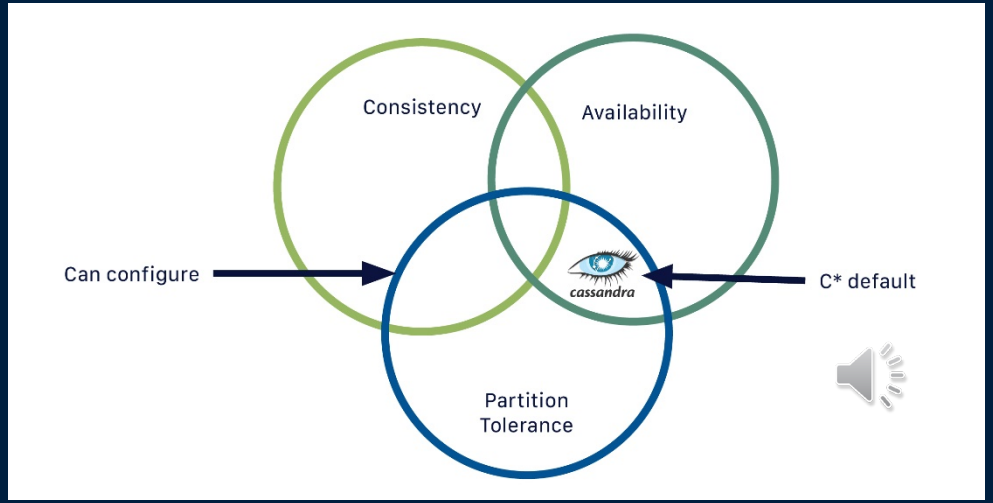
Overview of Cassandra



Source:https://cassandra.apache.org/doc/latest/

“CAP” of Cassandra

- **Consistency**: all backup data in different server nodes share the same value at the same time
- **Availability**: the entire clusters can still respond to the request even if node failures occur
- **Partition tolerance**: the entire clusters can still work even if a network partition causes communication interruption between nodes



Source:https://cassandra.apache.org/doc/latest/

Features

- **Keyspace**: Defines how a dataset is replicated, per datacenter. Replication is the number of copies saved per cluster.
- **Table**: Defines the typed schema for a collection of partitions. Cassandra tables can flexibly add new columns to tables with zero downtime.
- **Partition**: Defines the mandatory part of the primary key, where all rows in Cassandra must have in order to identify the node in a cluster where the row is stored.
- **Row**: Contains a collection of columns identified by a unique primary key made up of the partition key and optionally additional clustering keys.
- **Column**: A single datum with a type which belongs to a row.

Partition

- Data is automatically distributed by partitioning, with positive performance consequences
- Each node contains a specific set of tokens, data are distributed based on the range of these tokens across the cluster
- The partition key is responsible for distributing data among nodes and determining data locality
- A hash function is applied to the partition key, and the output is used to determine which node the data belongs to

Partition

COUNTRY	CITY	POPULATION
AU	Sydney	4.900.000
CA	Toronto	6.200.000
CA	Montreal	4.200.000
DE	Berlin	3.350.000
DE	Nuremberg	500.000

Partition Key

Partitioner

Hashing Function

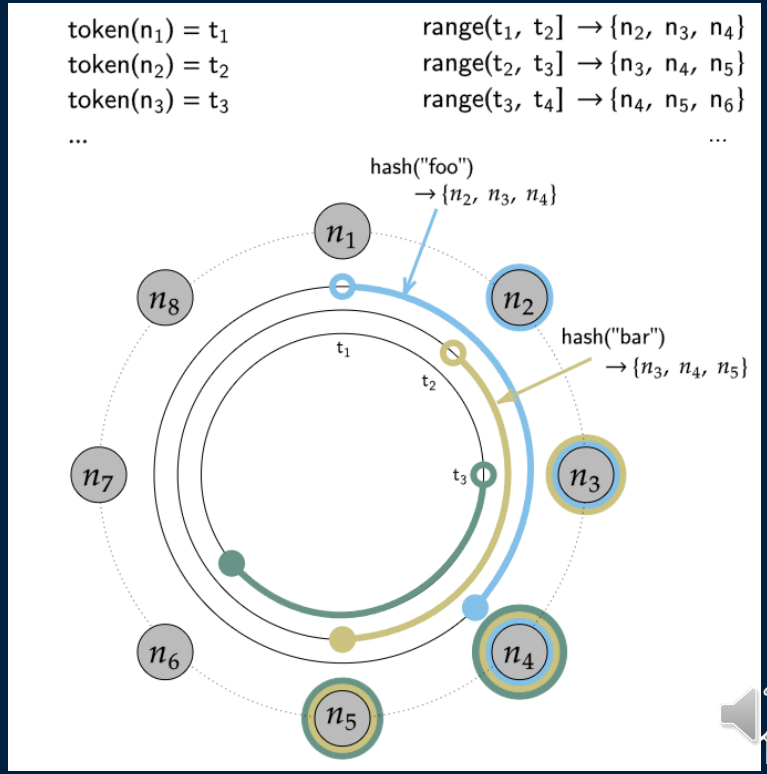
COUNTRY	CITY	POPULATION
59	Sydney	4.900.000
12	Toronto	6.200.000
12	Montreal	4.200.000
45	Berlin	3.350.000
45	Nuremberg	500.000

Tokens

Source:https://cassandra.apache.org/doc/latest/

Partition-Consistent Hashing

- Cassandra partitions data over storage nodes using a special form of hashing called consistent hashing
- For example, if we have an eight-node-cluster with evenly spaced tokens, and a replication factor (RF) of 3:
- Ranges of keys, also known as token ranges, map to the same physical set of nodes



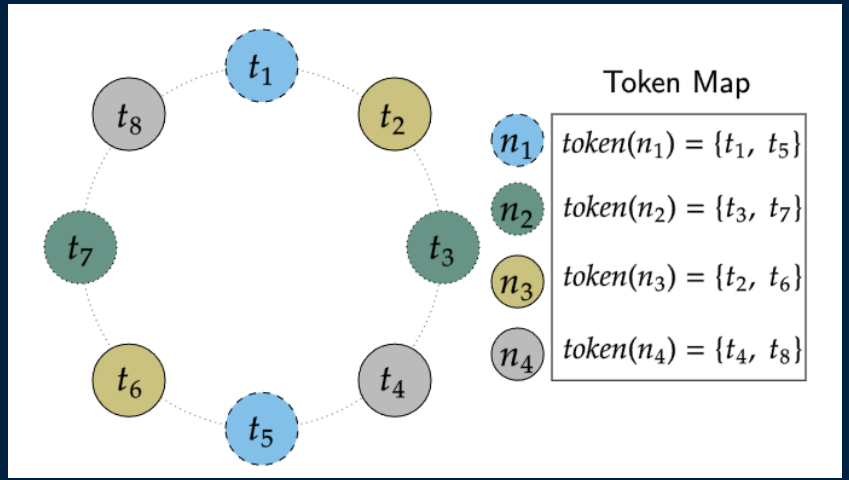
Source:https://cassandra.apache.org/doc/latest/

Partition-Multiple Tokens

- Cassandra seeks to avoid token imbalance because uneven token ranges lead to uneven request load
- It uses “virtual nodes” to solve the imbalance problem, which assigns multiple tokens in the token ring to each physical node
- Small clusters look larger and even with a single physical node addition we can make it look like we added many more nodes.
- **Token**: A single position on the dynamo style hash ring.
- **Endpoint**: A single physical IP and port on the network.
- **Host ID**: A unique identifier for a single "physical" node, usually present at one gEndpoint and containing one or more gTokens.
- **Virtual Node** (or **vnode**): A gToken on the hash ring owned by the same physical node, one with the same gHost ID.

Partition-Multiple Tokens

- When a new node is added, decommissioned, or becomes unavailable, distribution of data across cluster is kept equal
- However, every token introduces up to 2*(RF-1) additional neighbors on the token ring, which means that there are more combinations of node failures where we lose availability for a portion of the token ring.
- Cluster-wide maintenance operations are often slowed due to increasing number of tokens per node



Source:https://cassandra.apache.org/doc/latest/

Partition-Example

- The first field/component of a primary key is hashed to generate the partition key
- The remaining fields/components are the clustering keys to sort data within a partition

```
CREATE TABLE server_logs(
    log_hour timestamp PRIMARYKEY,
    log_level text,
    message text,
    server text
    )
```

partition key: **log_hour**

clustering columns: none

```
CREATE TABLE server_logs(
    log_hour timestamp,
    log_level text,
    message text,
    server text,
    PRIMARY KEY (log_hour, log_level)
    )
```

partition key: **log_hour**

clustering columns: **log_level**

```
CREATE TABLE server_logs(
    log_hour timestamp,
    log_level text,
    message text,
    server text,
    PRIMARY KEY ((log_hour, server))
    )
```

partition key: **log_hour, server**

clustering columns: none

```
CREATE TABLE server_logs(
    log_hour timestamp,
    log_level text,
    message text,
    server text,
    PRIMARY KEY ((log_hour, server),log_level)
    )WITH CLUSTERING ORDER BY (column3 DESC);
```

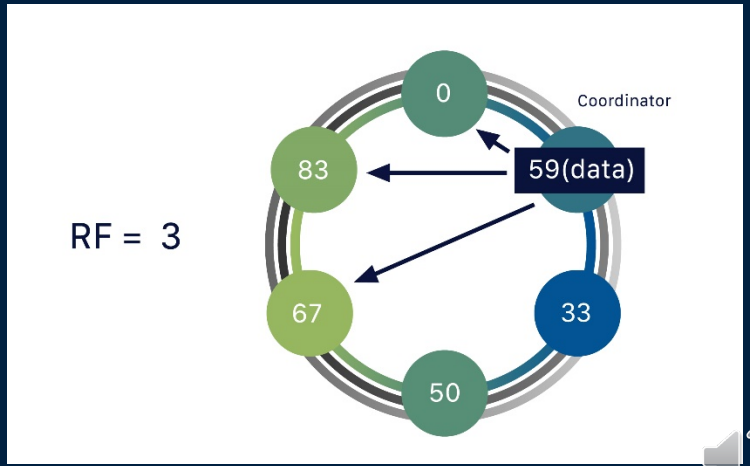
partition key: **log_hour, server**

clustering columns: **log_level**

Source:https://www.instaclustr.com/blog/cassandra-data-partitioning/

Replication

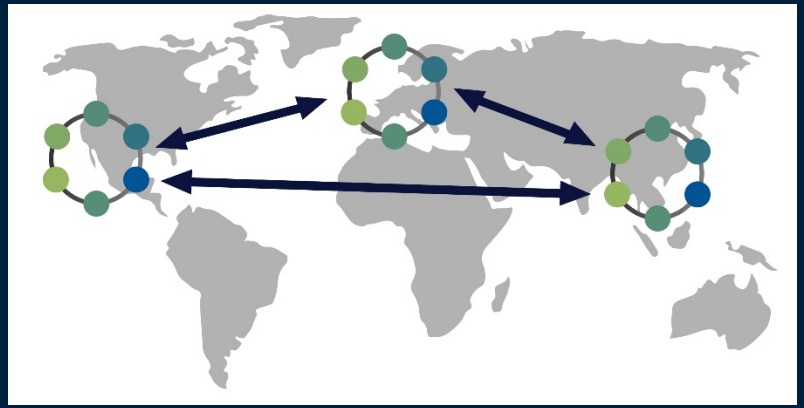
- One piece of data can be replicated to multiple nodes
    - Ensures reliability and fault tolerance
- The coordinator hashes the partition key to determine the token range the data belongs to and then replicates a mutation to the replicas of that data according to the replication strategy
- RF-replication factor: describes how many copies of data should exist in the database
- The coordinator stores a ”hint” for the data for recovery



Source:https://cassandra.apache.org/doc/latest/

Replication

- Replicas are always chosen such that they are distinct physical nodes which is achieved by skipping virtual nodes if needed.
- Every keyspace of data has its own replication strategy
- Multiple replicas can be accessed to provide data, so we can load balance amongst to achieve the best performance
- Data is replicated around different data centers



Source:https://cassandra.apache.org/doc/latest/

Replication-Tunable Consistency

- Consistency level (CL) represents the minimum nodes that must acknowledge a read or write operation to be considered successful
- Could still configure the consistency on a per-query basis
- In Dynamo, R(# of nodes that must participate in reads)+W((# of nodes that must participate in writes) > RF
- Cassandra instead choose from a menu of common consistency levels which allow the operator to pick R and W without knowing the RF
- For read operations, the coordinator generally only issues read commands to enough replicas to satisfy the consistency level. Write operations are always sent to all replicas

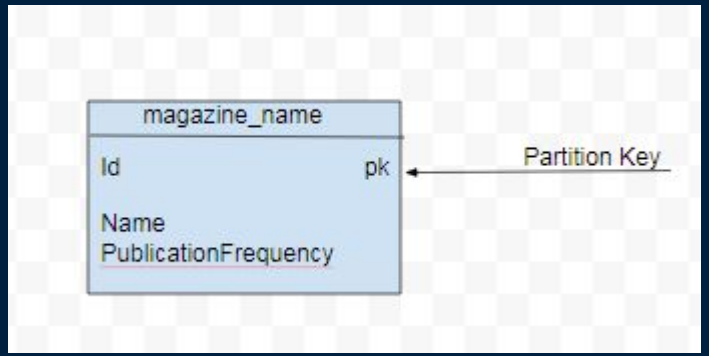
Replication-Gossip

- In Cassandra's gossip system, nodes exchange state information not only about themselves but also about other nodes they know about
- Information is in (generation, version) tuples. Generation is a monotonic timestamp and version is a logical clock the increments roughly every second
- Every second, every node in the cluster:
    1. Updates the local node's heartbeat state (the version) and constructs the node's local view of the cluster gossip endpoint state.
    2. Picks a random other node in the cluster to exchange gossip endpoint state with.
    3. Probabilistically attempts to gossip with any unreachable nodes (if one exists)
    4. Gossips with a seed node if that didn't happen in step 2.
- When an operator first bootstraps a Cassandra cluster they designate certain nodes as seed nodes
- Every node in Cassandra runs a variant of the Phi Accrual Failure Detector, in which every node is constantly making an independent decision of if their peer nodes are available or not.

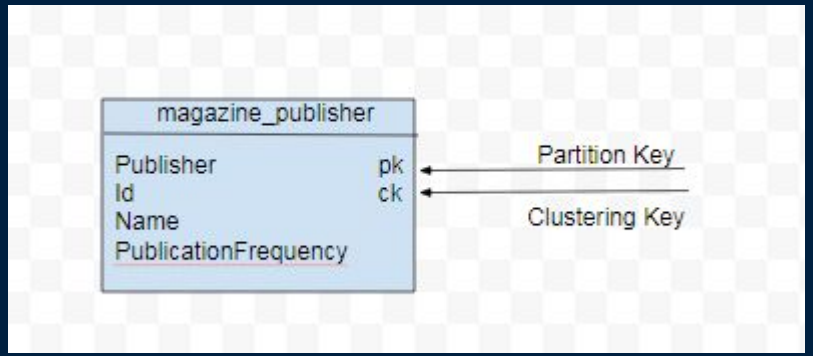
Data Modeling-Example

- Cassandra does not have the concept of foreign keys or relational integrity. Instead it models data by designing efficient queries
- It de-normalize data by duplicating data in multiple tables for a query-centric data model
- For example a magazine data set consists of magazine id, magazine name, publication frequency, publication date, and publisher.

Q1: list all the magazine names including their publication frequency.



Q2: list all the magazine names by publisher



Source:https://cassandra.apache.org/doc/latest/

Data Modeling-Denormalization

- No JOINS: If you have designed a data model and find that you need JOIN operation, you'll either do the work on the client side, or create a denormalized second table that represents the join results.
- No referential integrity: Although Cassandra stores ID's related to other entities in the table as well as primary keys, operations such as cascading deletes are not available
- It is often the case that companies end up denormalizing data in relational databases as well.
- In Cassandra you don't start with the data model; you start with the query model


THE UNIVERSITY OF BRITISH COLUMBIA