Data Management in Large-Scale Distributed Systems

Apache Cassandra

Thomas Ropars

thomas.ropars@univ-grenoble-alpes.fr

http://tropars.github.io/

2022

References

- The lecture notes of V. Leroy
- The lecture notes of F. Zanon Boito
- Designing Data-Intensive Applications by Martin Kleppmann
  - Chapters 2 and 7

In this lecture

- The design of Apache Cassandra
- Scaling to large number of nodes
  - Trade-off between consistency and performance
  - Distributed hash tables
  - Quorum systems

Agenda

Introduction

Paritioning and Data Distribution

Consistency and Replication

Apache Cassandra

- Column family data store
- Paper published by Facebook in 2010 (A. Lakshman and P. Malik)
  - Used for implementing search functionalities
  - Released as open source
- Warning: Some changes in the design have been made since the first version of Cassandra

Disclaimer

About the information provided in this lecture:

- Not necessarily up-to-date with to the most recent version of Cassandra
- The goal is to understand some generally applicable ideas
- We are not going to describe all parts of Cassandra:
  - Focus on partitioning and consistency

The design principles of Cassandra are mostly inspired from other systems:

- Google BigTable
  - Storage engine
- Amazon Dynamo
  - Partitioning and consistency

Agenda

Introduction

Paritioning and Data Distribution

Consistency and Replication

Partitioning in Cassandra

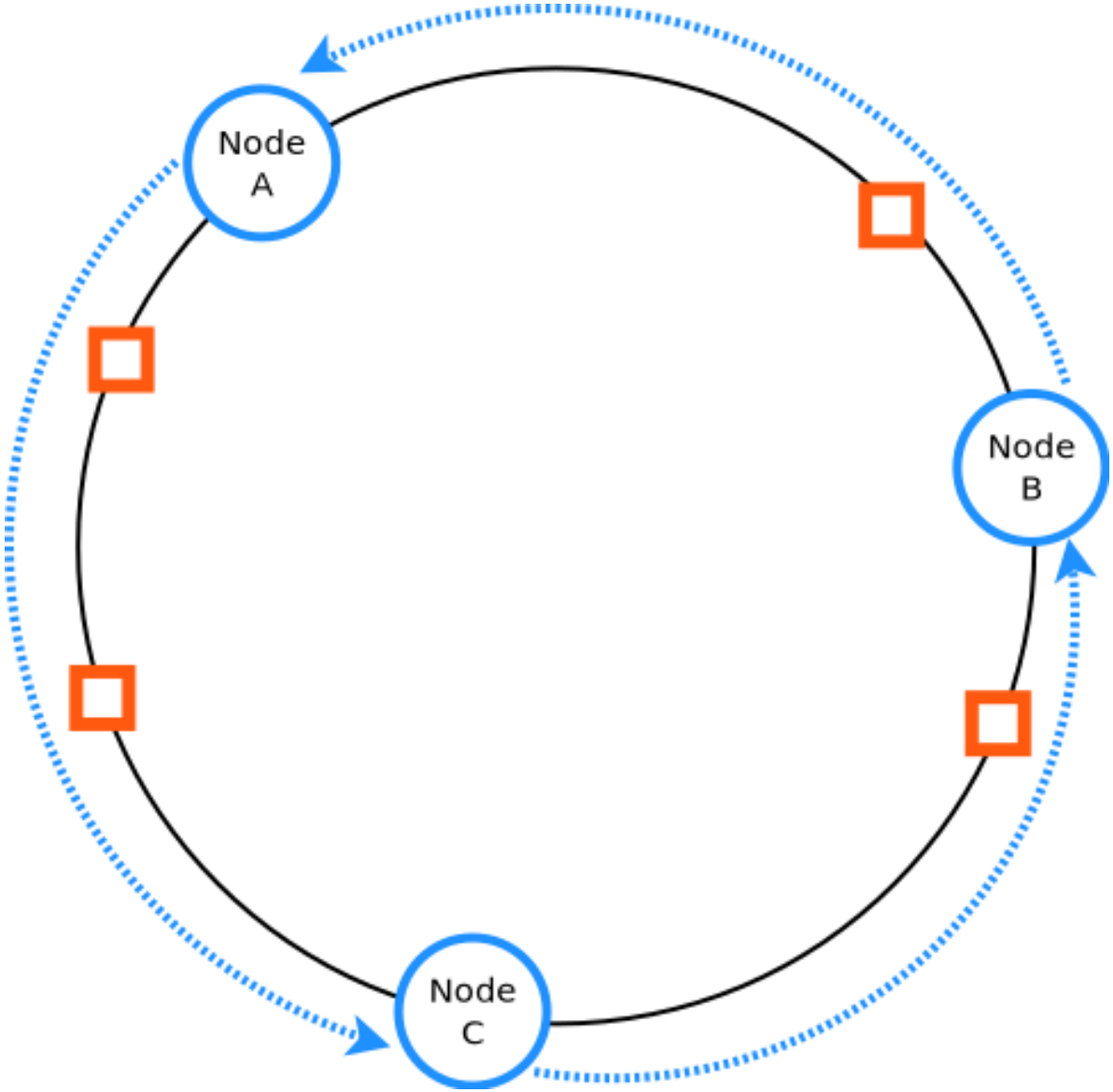
Partitioning based on a hashed name space

- Data items are identified by keys
- Data are assigned to nodes based on a hash of the key
  - Tries to avoid hot spots

Namespace represented as a ring

- Allows increasing incrementally the size of the system
- Each node is assigned a random identifier
  - Defines the position of a node in the ring
- Each node is responsible for all the keys in the range between its identifier and the one of the previous node.

Partitioning in Cassandra



Better version of the partitioning

Limits of the current approach:

Better version of the partitioning

Limits of the current approach: High risk of imbalance

- Some nodes may store more keys than others
  - Nodes are not necessarily well distributed on the ring
  - Especially true with a low number of nodes
- Issues when nodes join or leave the system
  - When a node joins, it gets part of the load of its successor
  - When a node leaves, all the corresponding keys are assigned to the successor

Better version of the partitioning

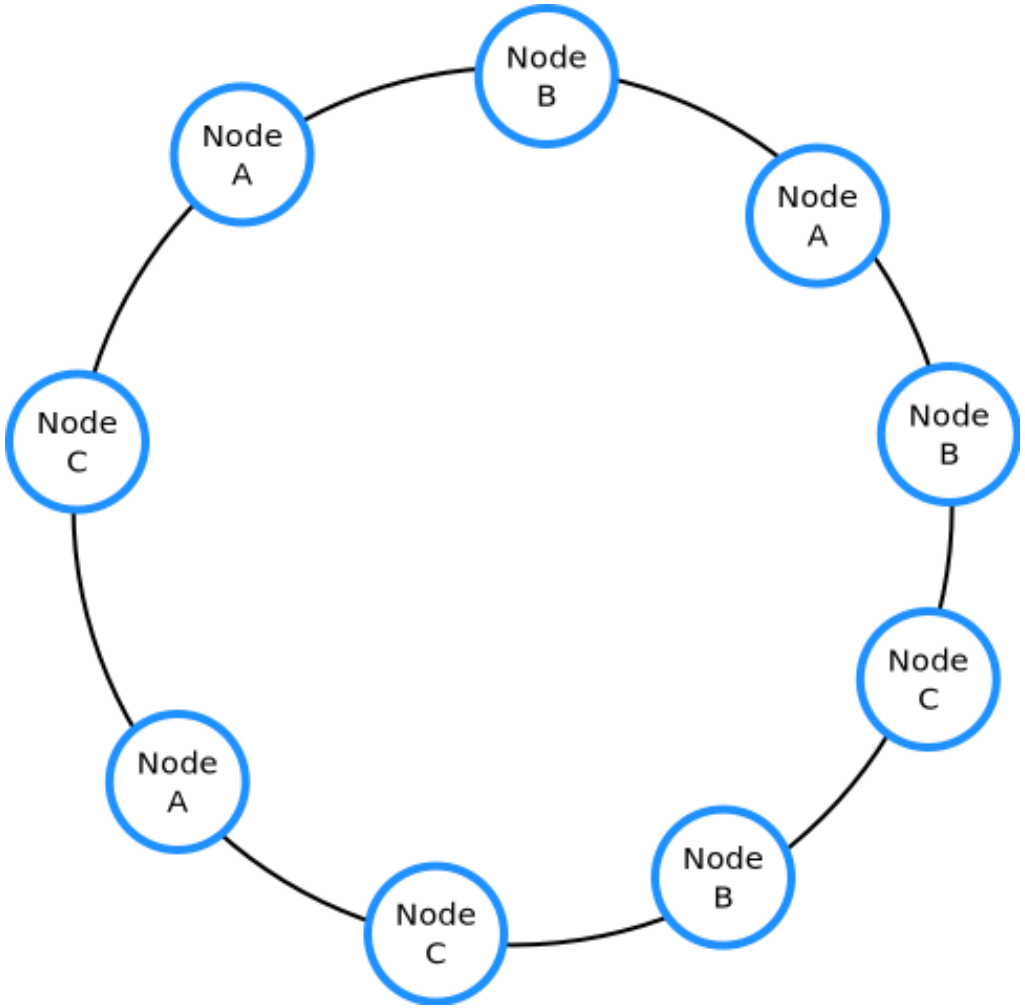
Limits of the current approach: High risk of imbalance

- Some nodes may store more keys than others
  - Nodes are not necessarily well distributed on the ring
  - Especially true with a low number of nodes
- Issues when nodes join or leave the system
  - When a node joins, it gets part of the load of its successor
  - When a node leaves, all the corresponding keys are assigned to the successor

Concept of virtual nodes

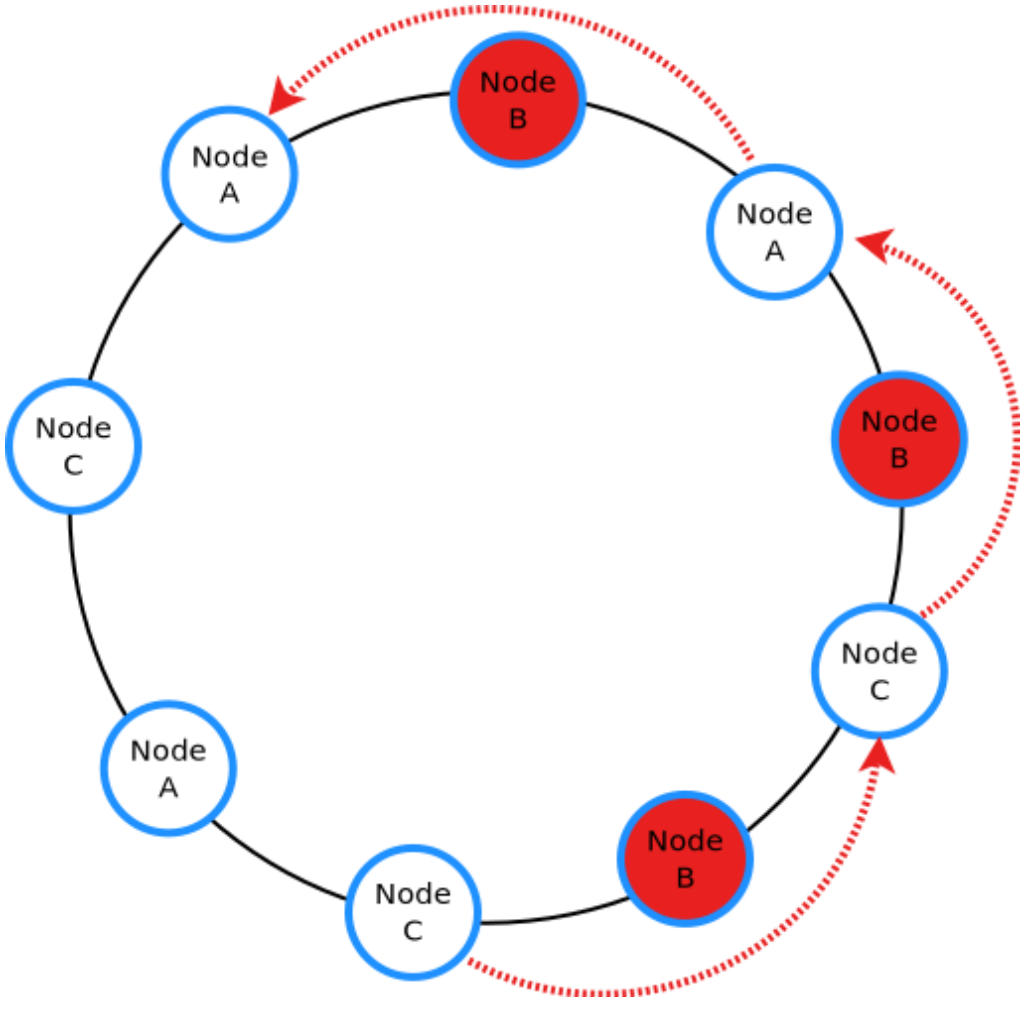
- Assign multiple random positions to each node

Partitioning and virtual nodes



The key space is better distributed between the nodes

Partitioning and virtual nodes



If a node crashes, the load is redistributed between multiple nodes

Partitioning and replication

Items are replicated for fault tolerance.

Strategies for replica placement

- Simple strategy

- Topology-aware placement

Partitioning and replication

Items are replicated for fault tolerance.

Strategies for replica placement

- Simple strategy
  - Place replicas on the next R nodes in the ring
- Topology-aware placement
  - Iterate through the nodes clockwise until finding a node meeting the required condition
  - For example a node in a different rack

Agenda

Introduction

Paritioning and Data Distribution

Consistency and Replication

Replication in Cassandra

Replication is based on quorums

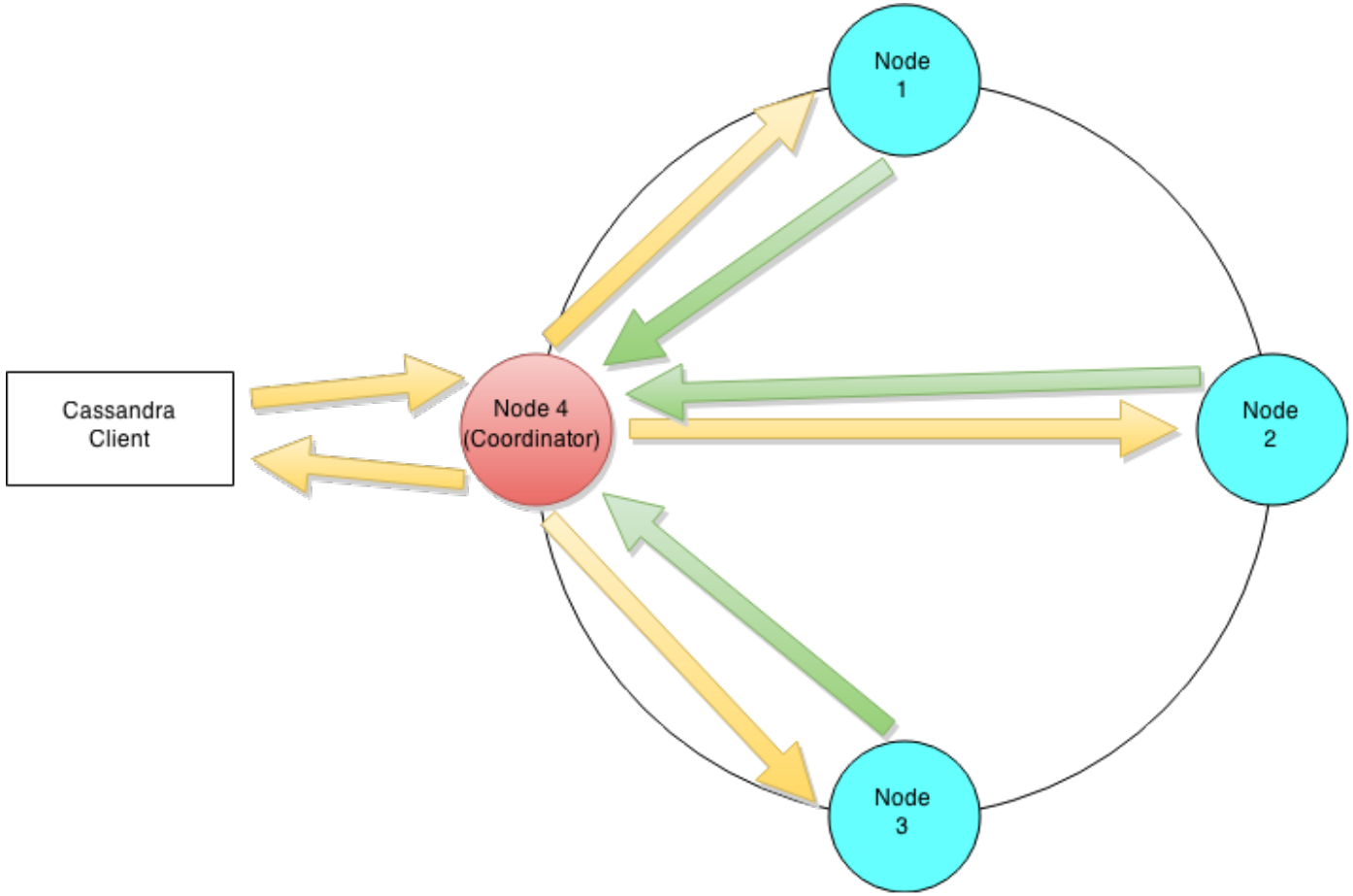
- A read/write request might be processed by a subset of the replicas
  - In practice, in Cassandra, write requests are sent to all replicas.
  - To tolerate f faults, it has to be processed by at least $f + 1$ replicas

Consistency

- The user can choose the level of consistency
  - Trade-off between consistency and performance (and availability)
- Eventual consistency
  - If an item is modified, readers will *eventually* see the new value
  - All reads of an item will *eventually* return the same value

A Read/Write request

Figure from https://dzone.com/articles/introduction-apache-cassandras



- A client can contact any node in the system
- The coordinator contacts all replicas
- The coordinator waits for a specified number of responses before sending an answer to the client

Consistency levels

ONE (default level)

- The coordinator waits for one ack on write before answering the client
- The coordinator waits for one answer on read before answering the client
- Lowest level of consistency
  - Reads might return stale values
  - We will still read the most recent values in most cases

QUORUM

- The coordinator waits for a majority of acks on write before answering the client
- The coordinator waits for a majority of answers on read before answering the client
- High level of consistency
  - At least one replica will return the most recent value

Additional references

Mandatory reading

- Chapter 4 of *Readings in Database Systems*: New DBMS Architectures: `http://www.redbook.io/ch4-newdbms.html`.

Suggested reading

- `https://docs.datastax.com/en/articles/cassandra/cassandrathenandnow.html`, Facebook's Cassandra paper, annotated and compared to Apache Cassandra 2.0.
- Chapter 6 of *Readings in Database Systems*: Weak Isolation and Distribution: `http://www.redbook.io/ch6-isolation.html`.