Large-Scale and Multi-Structured Databases

Column Databases Details

Prof. Pietro Ducange

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB
Innovation for industry 4.0

Key-Value and Document Databases: Some issues

NoSQL databases discussed so far, may help us to solve the problem of handling ***huge amount*** of data.

However, some issues occur:

- ***Key-value*** databases ***lack support*** for organizing ***many attributes*** and keeping ***frequently used*** data together.

- Document databases may ***not*** have some of the useful features of relational databases, such as a ***SQL-like query*** language.

BigTable: The Google NoSQL Database

It was introduced in 2006 for Google large services such as Google Earth and Google Finance.

Main Features:

- ***Column based*** with a ***dynamic*** control over columns
- Data values are indexed by ***row*** identifier, ***column*** name, and ***a time stamp***
- Reads and writes of a row are ***atomic***
- ***Rows*** are maintained in a ***sorted order***

BigTable: An Example

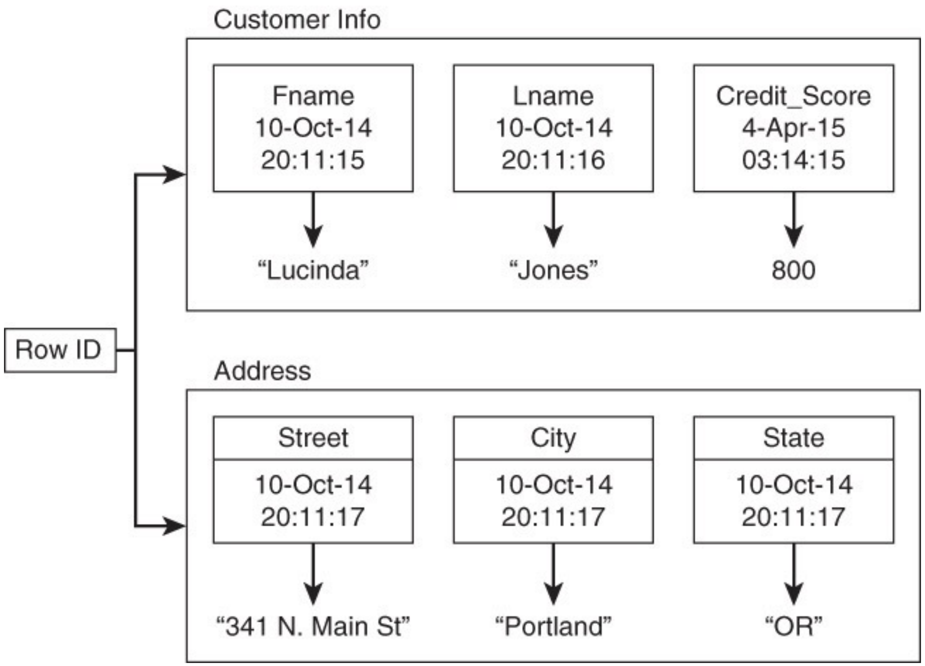


Image extracted from: "Dan Sullivan, NoSQL For Mere Mortals, Addison-Wesley, 2015

- A data record is a ***row*** composed of several ***column families***.
- Each family consists of a set of ***related columns*** (frequently used together.)
- ***Column values*** within a column family are ***kept together*** on the disk, whereas values of the same row belonging to different families may be stored far.
- ***Columns families*** must be defined ***a priori*** (like relational tables).
- Applications may ***add*** and ***remove columns*** inside families (like key-value pairs).

BigTable: Indexing

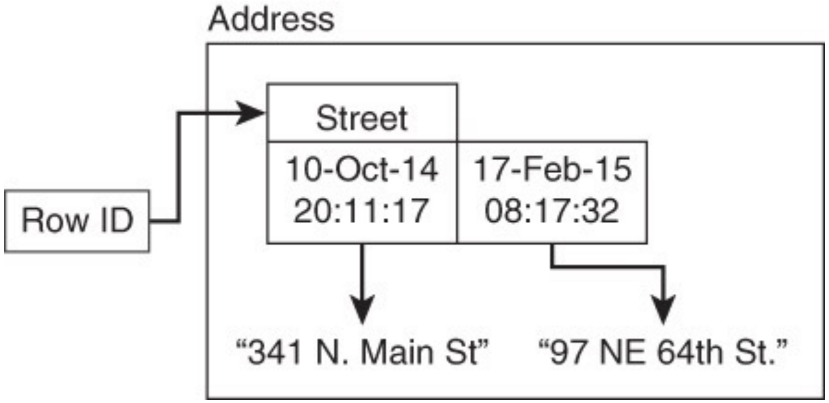


Image extracted from: "Dan Sullivan, NoSQL For Mere Mortals, Addison-Wesley, 2015"

A data values is indexed by its :

- ***row identifier*** (similar role than a primary key in relational databases)
- ***column name***, which identifies a column (similar role than a key in key-value databases)
- ***time stamp***, which allows the existence of multi versions of a column value can exist.

Keyspace

It is the ***top-level logical container***. It holds column families, row keys, and related data structures. Typically, there is one keyspace per application.

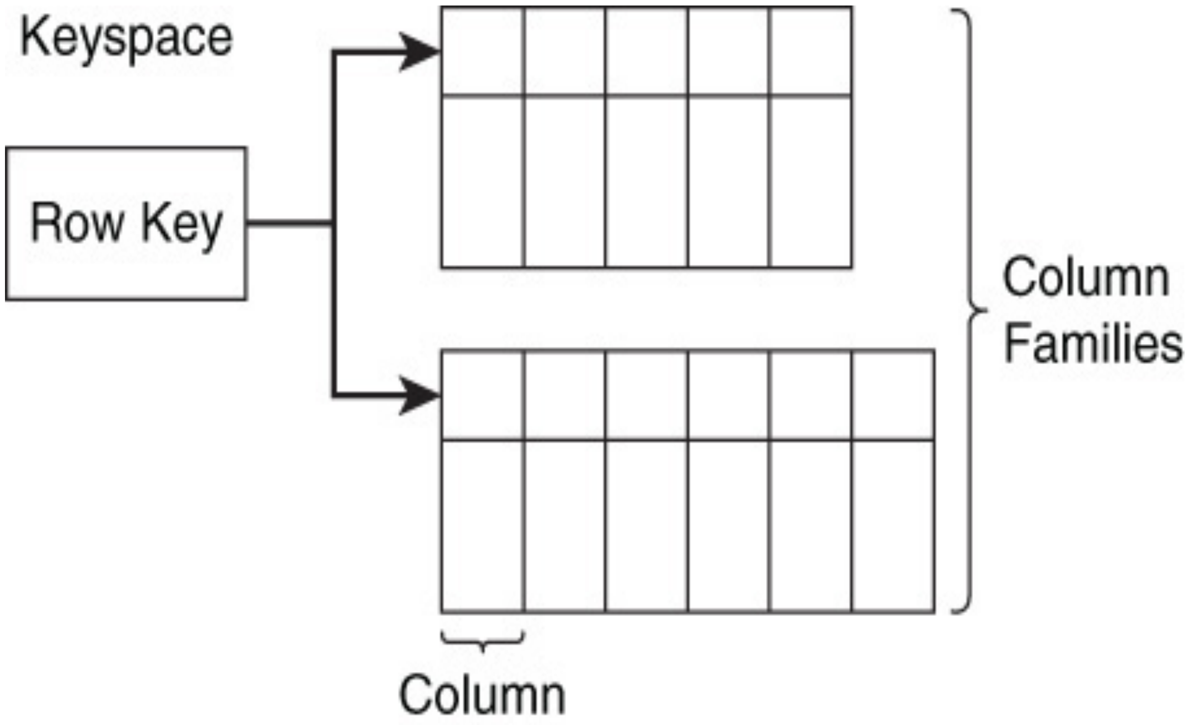


Image extracted from: "Dan Sullivan, NoSQL For Mere Mortals, Addison-Wesley, 2015

Row Keys

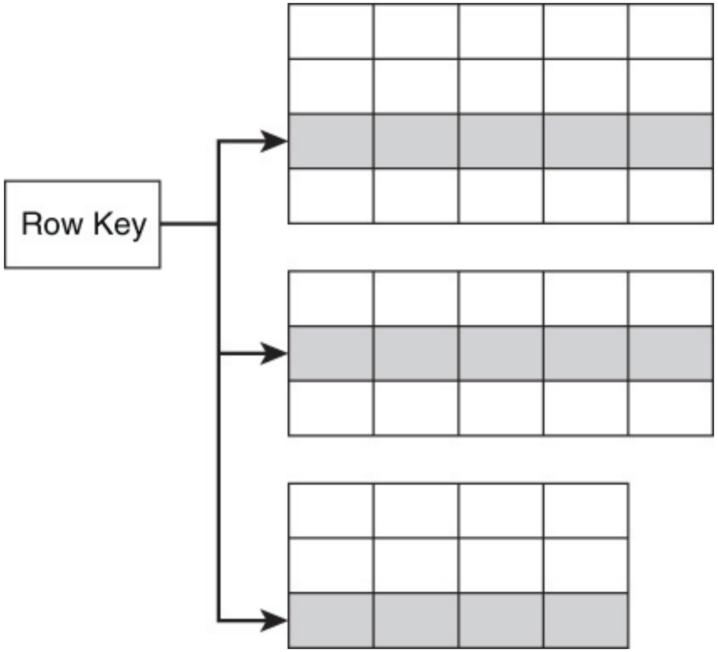


Image extracted from: "Dan Sullivan, NoSQL For Mere Mortals, Addison-Wesley, 2015

- Row keys are one of the components used to ***uniquely identify*** values stored in a database.
- Row keys are also used to ***partition*** and to ***order*** data.
- Algorithms for data partitioning and ordering depends on ***specific databases***.
- As an example, ***Cassandra*** adopts the ***partitioner***, a specific object, which is used for sorting data.
- Moreover, by default, the partitioner ***randomly distributes*** rows across nodes.

Columns

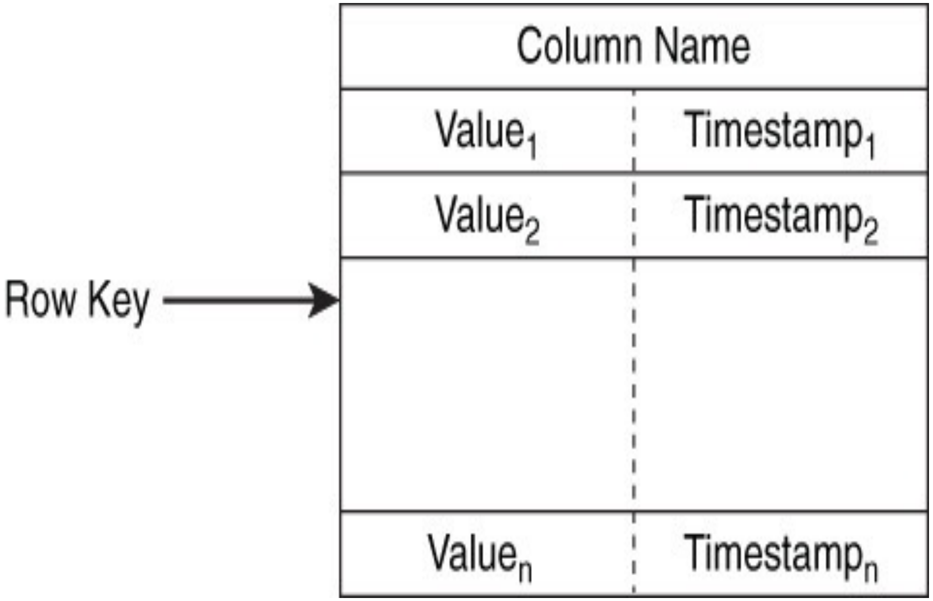


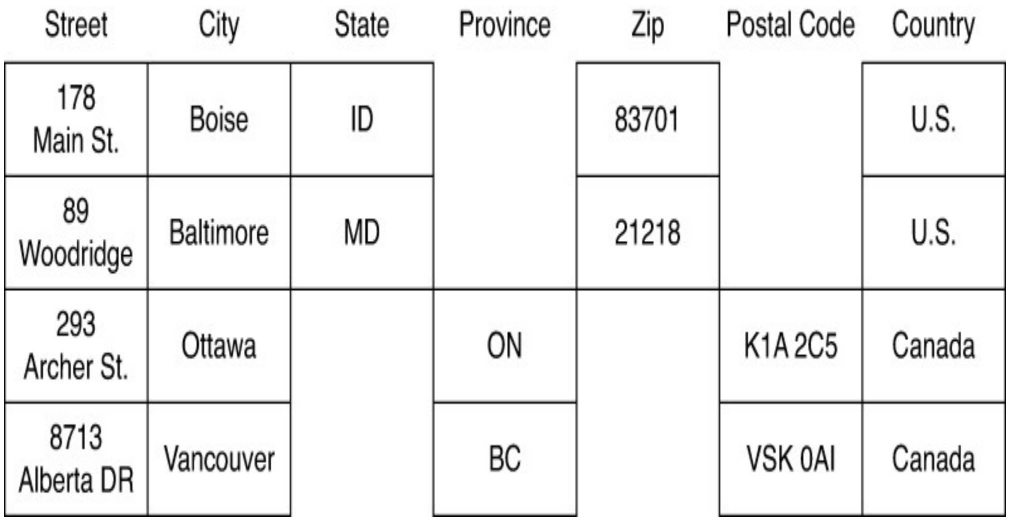
Image extracted from: “Dan Sullivan, NoSQL For Mere Mortals, Addison-Wesley, 2015

- A column, along with a row key and version stamp, uniquely identifies values.
- Values may be typed, such as in Cassandra, or not, such as in BigTable.
- The time stamp or other version stamp is a way to order values of a column.

Column Families

Columns that are ***frequently*** used together are ***grouped*** into the same column family.

Column family ***analogous*** to a ***table*** in a relational database. However:



Street	City	State	Province	Zip	Postal Code	Country
178 Main St.	Boise	ID		83701		U.S.
89 Woodridge	Baltimore	MD		21218		U.S.
293 Archer St.	Ottawa		ON		K1A 2C5	Canada
8713 Alberta DR	Vancouver		BC		VSK 0AI	Canada

Image extracted from: "Dan Sullivan, NoSQL For Mere Mortals, Addison-Wesley, 2015

- Rows in columns families are usually ***ordered*** and ***versioned***
- Columns in a family can be modified ***dynamically***.
- Modifications just require making a reference to them from the ***client application***.

Cluster and Partitions

The most recent column databases are designed for ensuring high ***availability*** and ***scalability***, along with different ***consistency levels***.

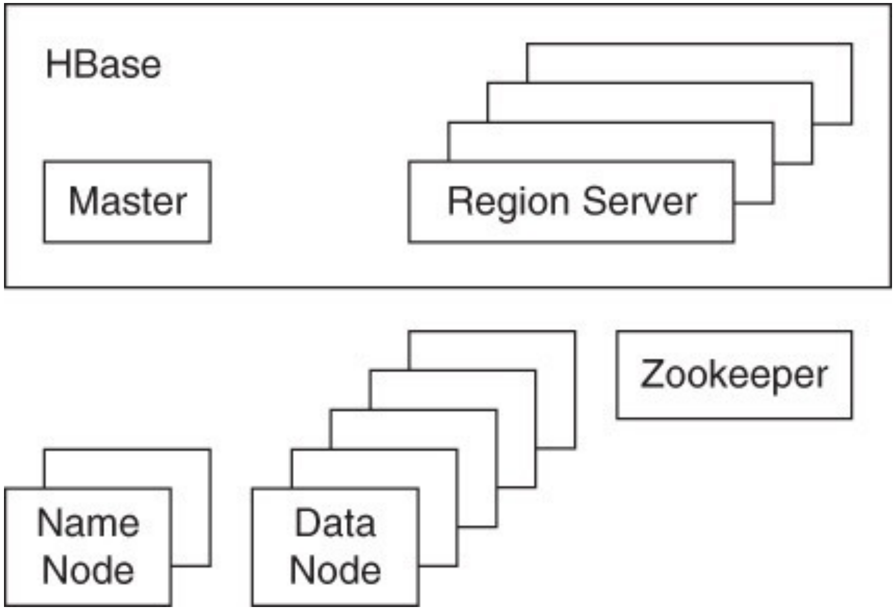
Clusters and partitions are fundamental for the ***distributed implementation*** of the databases.

A partition is a ***logical subset*** of a database.

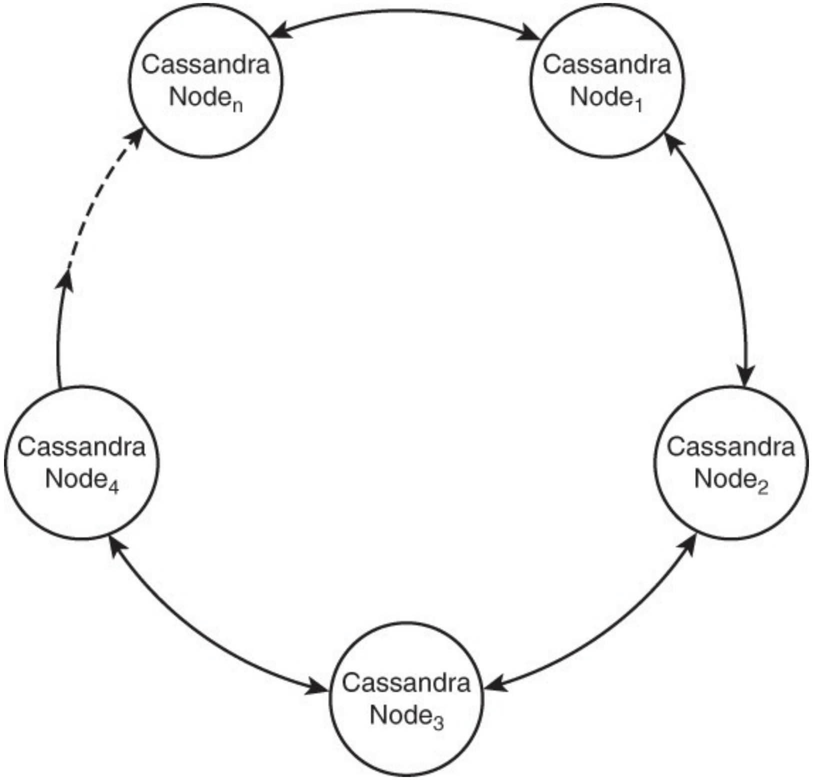
Partitions are usually used to store a set of data based on ***some attribute*** of the data in column databases, they can be generated on the basis of:

- A ***range of values***, such as the value of a row ID/column name
- A ***hash value***, such as the hash value of a column name
- A ***list of values***, such as the names of states or provinces
- A ***composite*** of two or more of the above options

Column DB Architectures

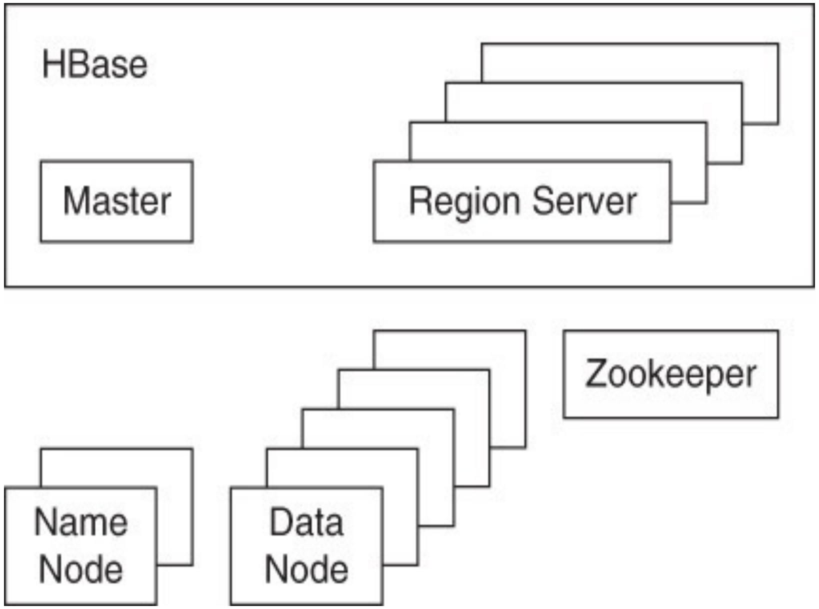


Apache HBase Architecture



Cassandra peer-to-peer architecture

HBase



Depends on multiple types of nodes that make up the ***Hadoop*** environment.

Hmaster: assigns regions to Region Server (slave).

Region Servers: are the end nodes that handle all user requests.

Regions in a specific region server contain all the rows between specified keys.

Zookeeper is a type of node that enables coordination between nodes within a Hadoop cluster.

HBase

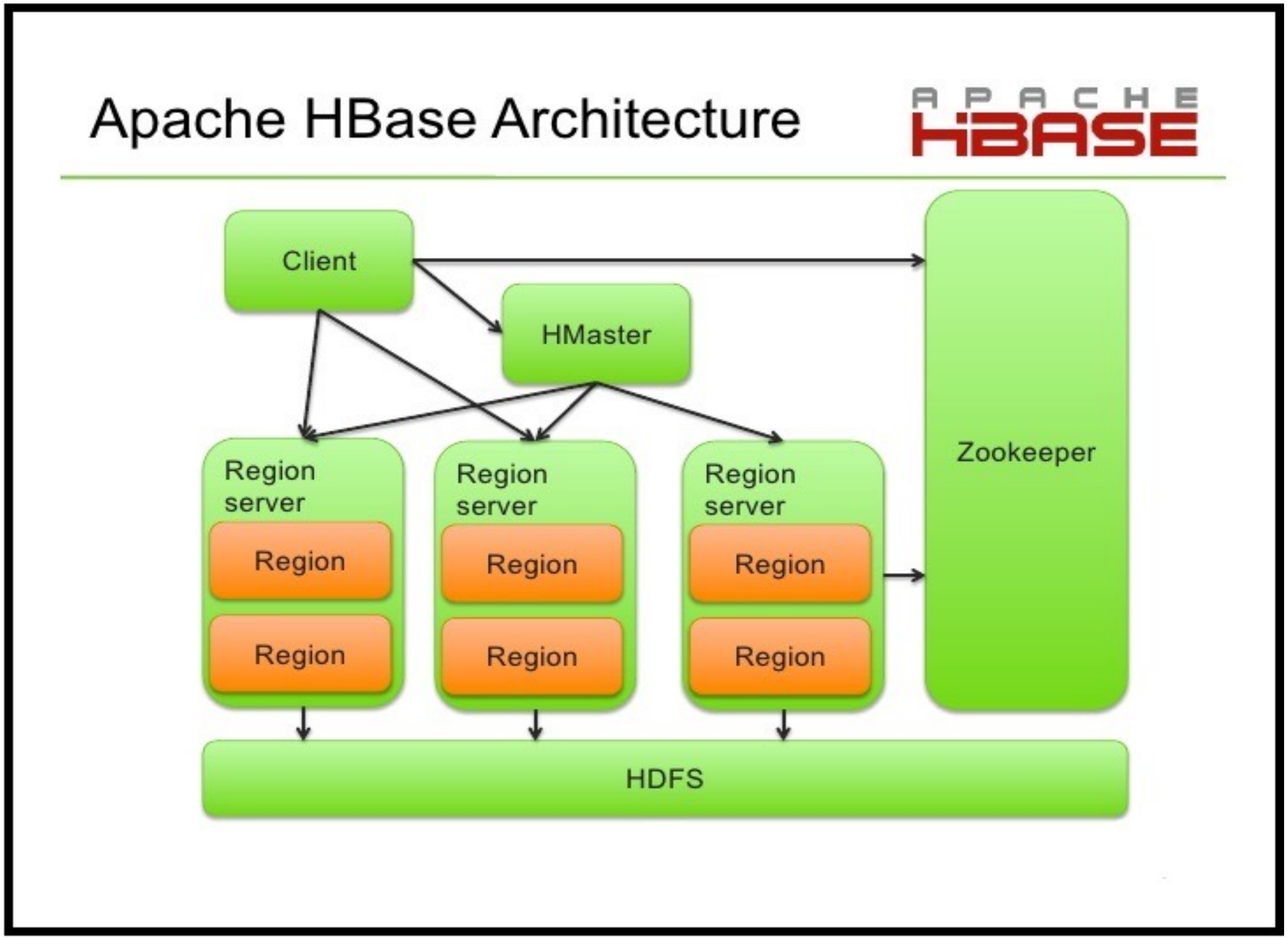


Image extracted from: https://www.tutorialandexample.com/hbase-architecture

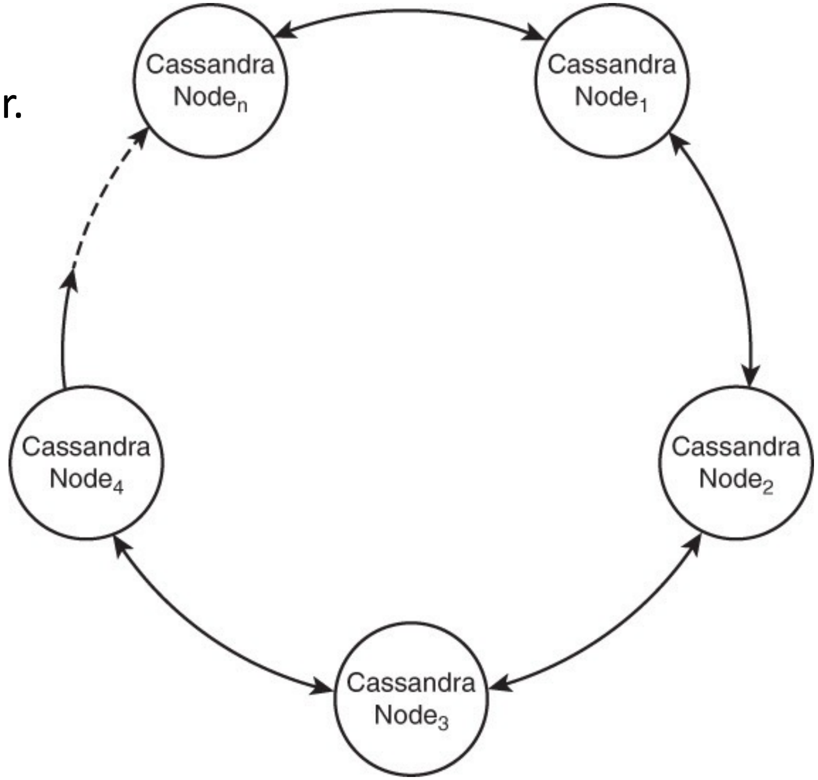
Cassandra

All Cassandra nodes run the ***same software***.

Nodes may serve ***different functions*** for the cluster.

Advantages of Cassandra:

1. Simplicity
2. No node can be a single point of failure.
3. Scaling up and down is fairly straightforward
4. Servers communicate with each other
5. When a node is removed servers hosting replicas of data from the removed node respond to read and write requests.



Commit Logs

- Commit Logs are ***append only*** files that always ***write*** data to the end of the file.

- These files allow column databases to ***avoid waiting too much*** for the definitive writes on persistent supports.

- Whenever a write operation is sent to the database, is stored into a commit log. Then, the data is updated into the specific server and disk block.

- Even in case of failure, the ***recovery*** process reads the commit logs and writes entries to partitions.

Bloom Filters

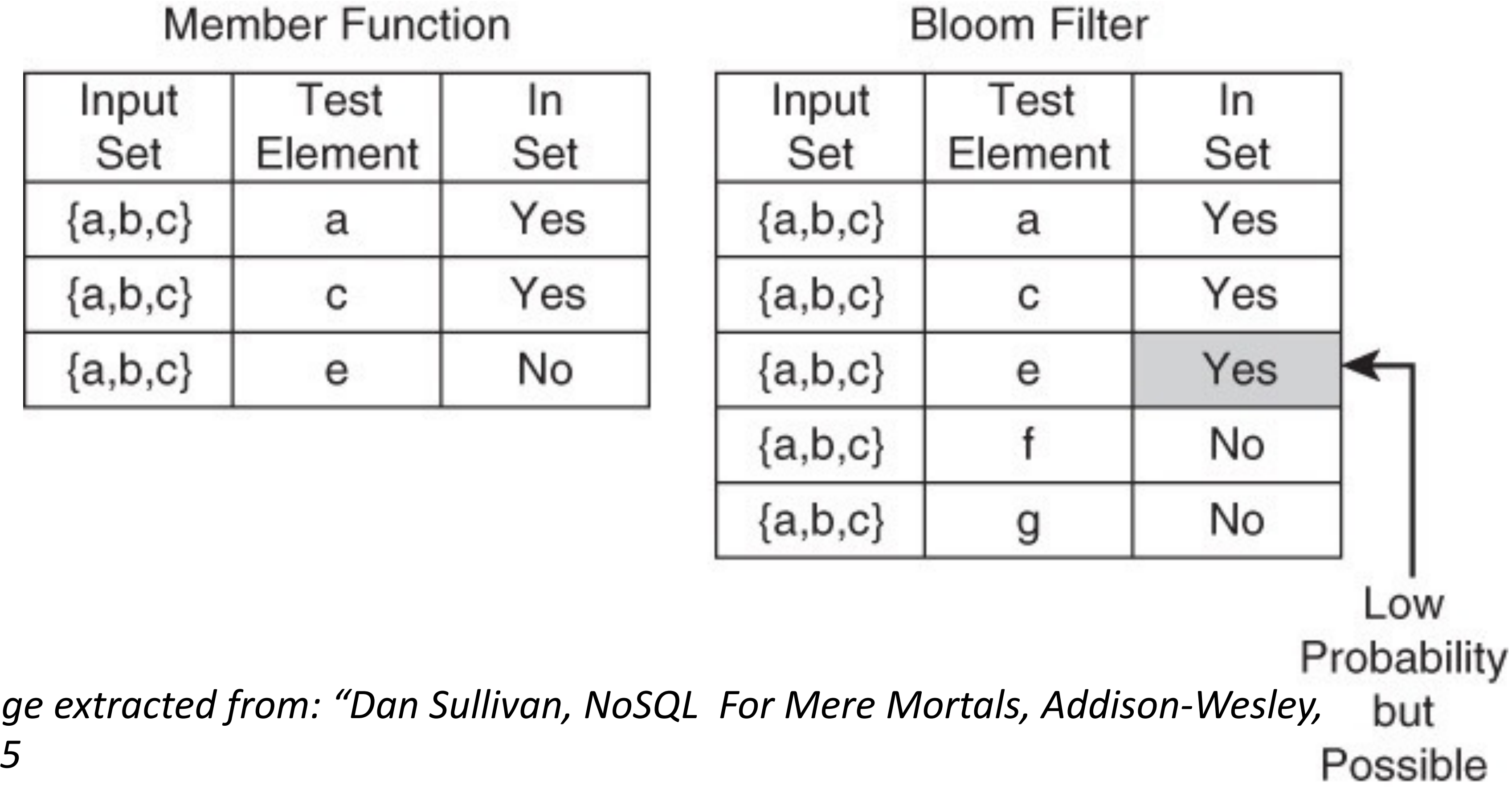
Bloom filters are ***probabilistic*** data structures.

Bloom filters help to ***reduce*** the number of ***reading operations***, avoiding reading partitions that certainly ***do not contain*** a specific searched data.

A Bloom filter tests whether or not an element is a member of a set: the answer of the test is "***no***" or "***maybe***". Thus, ***false positive*** may be obtained.

However, if the answer is "no" for sure the element, namely the searched data, is not present into the specific partition.

Bloom Filters



Member Function

Input Set	Test Element	In Set
{a,b,c}	a	Yes
{a,b,c}	c	Yes
{a,b,c}	e	No

Bloom Filter

Input Set	Test Element	In Set
{a,b,c}	a	Yes
{a,b,c}	c	Yes
{a,b,c}	e	Yes
{a,b,c}	f	No
{a,b,c}	g	No

Low Probability but Possible

Image extracted from: "Dan Sullivan, NoSQL For Mere Mortals, Addison-Wesley, 2015

Bloom Filters: An Example

- Let consider a ***specific partition***. A bloom filter for this partition can be an ***array of n binary values***. At the beginning, all the elements are set to 0.

- Let consider also a set of ***k << n hash functions*** which ***map*** each ***data*** to be stored into the partition ***in an element*** of the array.

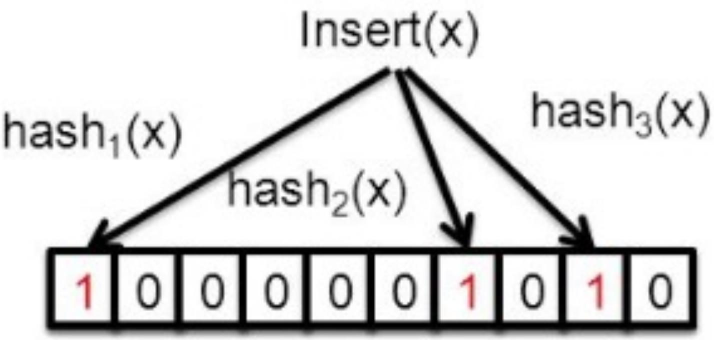
- When a new data is ***added (write)*** to the partition, all the hash functions are calculated and the corresponding bit are ***set to one***.

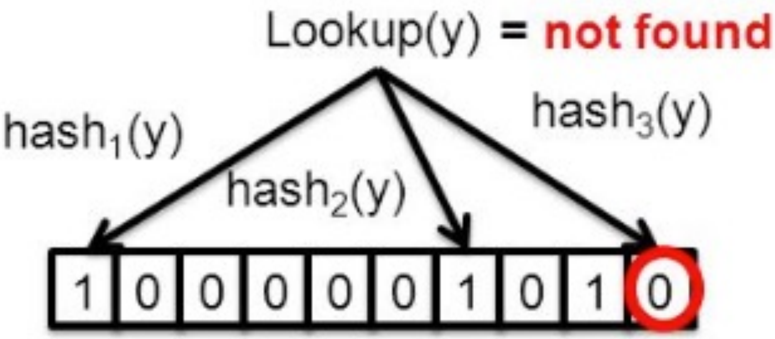
- When a ***read*** operation is required, all the hash functions are calculated and the access to the partition is actually performed ***if and only if*** all the corresponding element in the array are ***set to 1***.

- The maximum false positive rate depends on the values of n and k.

Bloom Filters: An Example

- Let consider a Bloom Filter with n = 10 and k = 3.
- The ***Insert*** function describes a ***write*** operation
- The ***Lookup*** function describes a ***read*** operation





Bloom Filters: some considerations

Important property of Bloom filters is that the ***time*** it takes both to add an element and to check if an element is in the set is ***constant***.

The ***probability*** of a false positive ***decreases*** as n increases, while it ***increases*** as the number of elements stored in the Bloom filter decreases.

Removing an element from the simple Bloom filter is ***impossible*** because ***false negatives are not allowed***.

Suggestion: use $n = O(h)$ and $k=\ln 2(n/h)$where h is the number of value to check against the filter.

Consistency Level

Consistency level refers to the consistency ***between copies*** of data on ***different replicas***.

We can consider different level of consistency:

- ***Strict consistency***: all replicas must have the same data
- ***Low consistency***: the data must be written in at least one replica
- ***Moderate consistency***: all the intermediate solutions

The consistency level depends on the specific ***application requirements.***

Replication

- Consistency level is strictly related to the ***replication process***.
- Replication process regards where ***place*** replicas and how to ***keep*** them ***updated***.
- Usually, there is a pre-defined server for the ***first replica*** which is in charge of identifying the position of the other replicas among the available servers.
- The primary server is often determined by a ***hash function***.
- Each database has its own ***replication strategies***.
- For example, Cassandra uses clusters organized in ***logical rings*** and additional replicas are stored in successive nodes in the rings in ***clockwise directions***.

Anti-Entropy

Anti-entropy is the process of ***detecting differences*** in replicas.

It is important to detect and resolve inconsistencies among replicas ***exchanging a low amount*** of data.

Hash trees may be exchanged among servers in order to easily check differences. Each leaf of the tree contains ***the hash value of a data set*** in the partition. Each parent node contains the ***hash value of the hashes*** its child nodes.

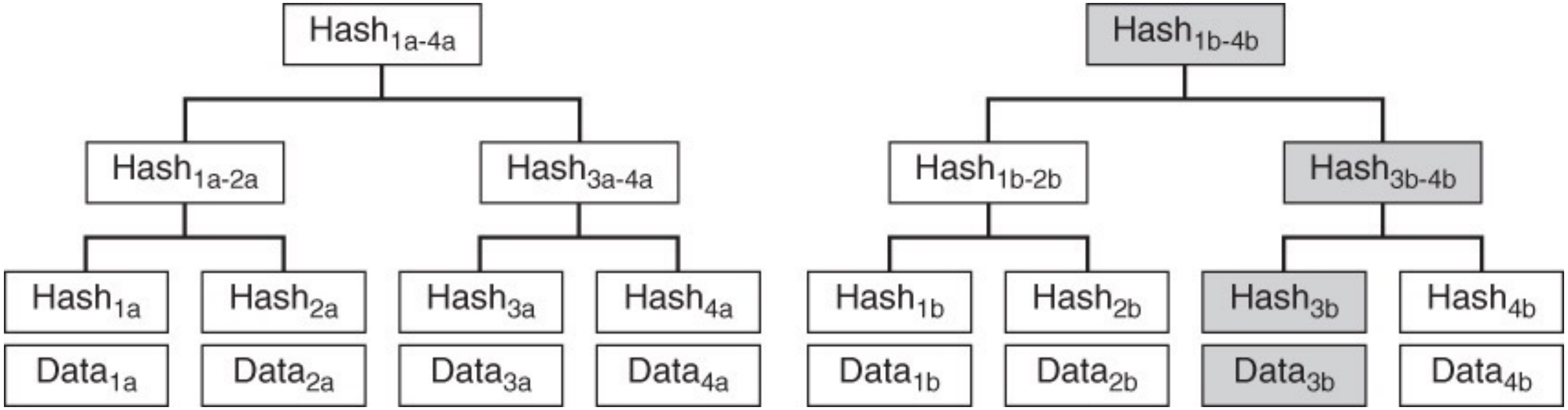


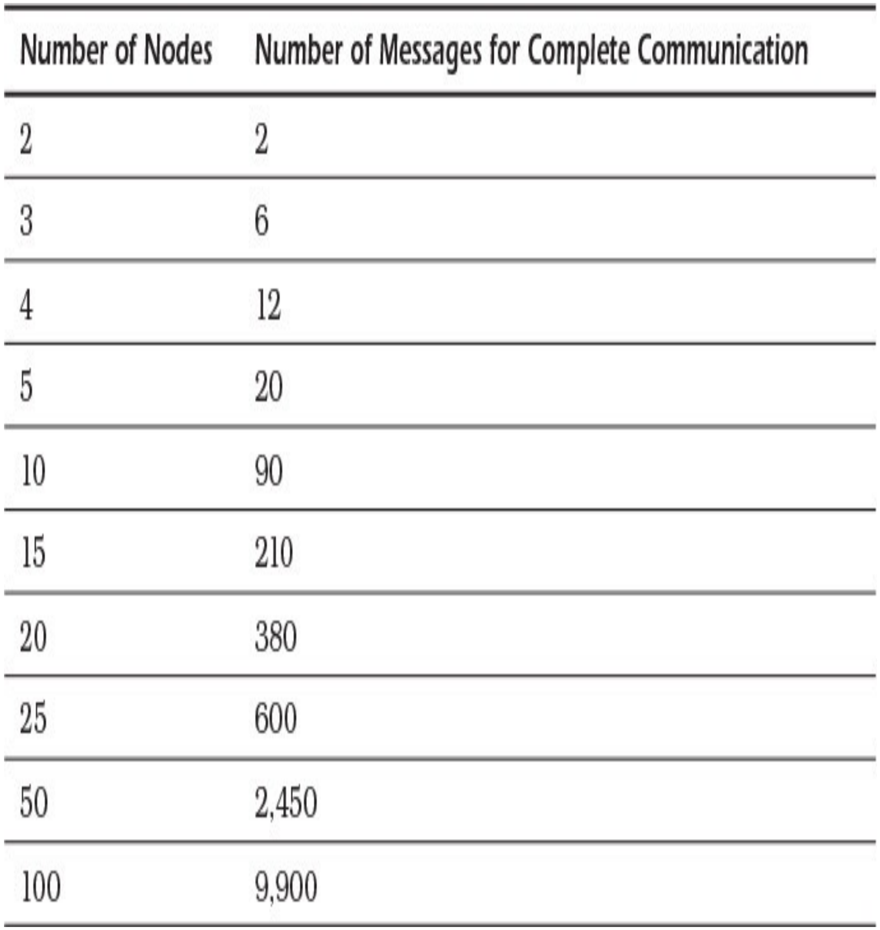
Image extracted from: "Dan Sullivan, NoSQL For Mere Mortals, Addison-Wesley, 2015

Communication Protocols

In distributed systems, each node needs to be ***aware*** about the ***status*** of the other nodes.

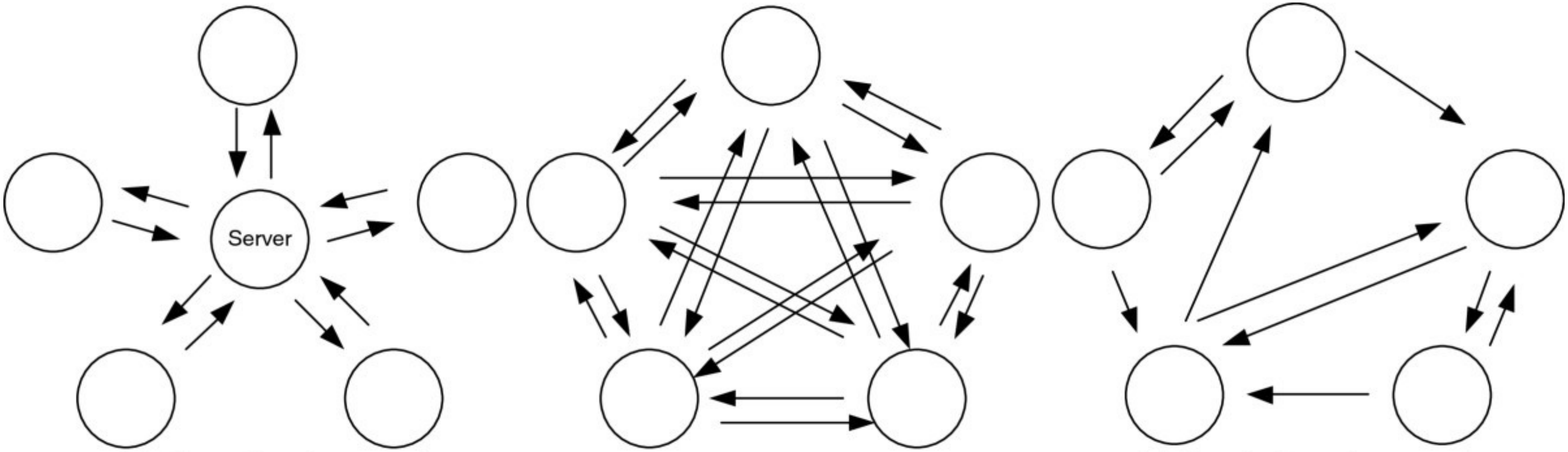
In small cluster, each node can ***exchange messages*** with each other node.

The number of total ***messages to exchange*** for a complete communication is equal to ***n×(n–1)/2***, where *n* is the number of nodes of a cluster.



Number of Nodes	Number of Messages for Complete Communication
2	2
3	6
4	12
5	20
10	90
15	210
20	380
25	600
50	2,450
100	9,900

Communication Protocols



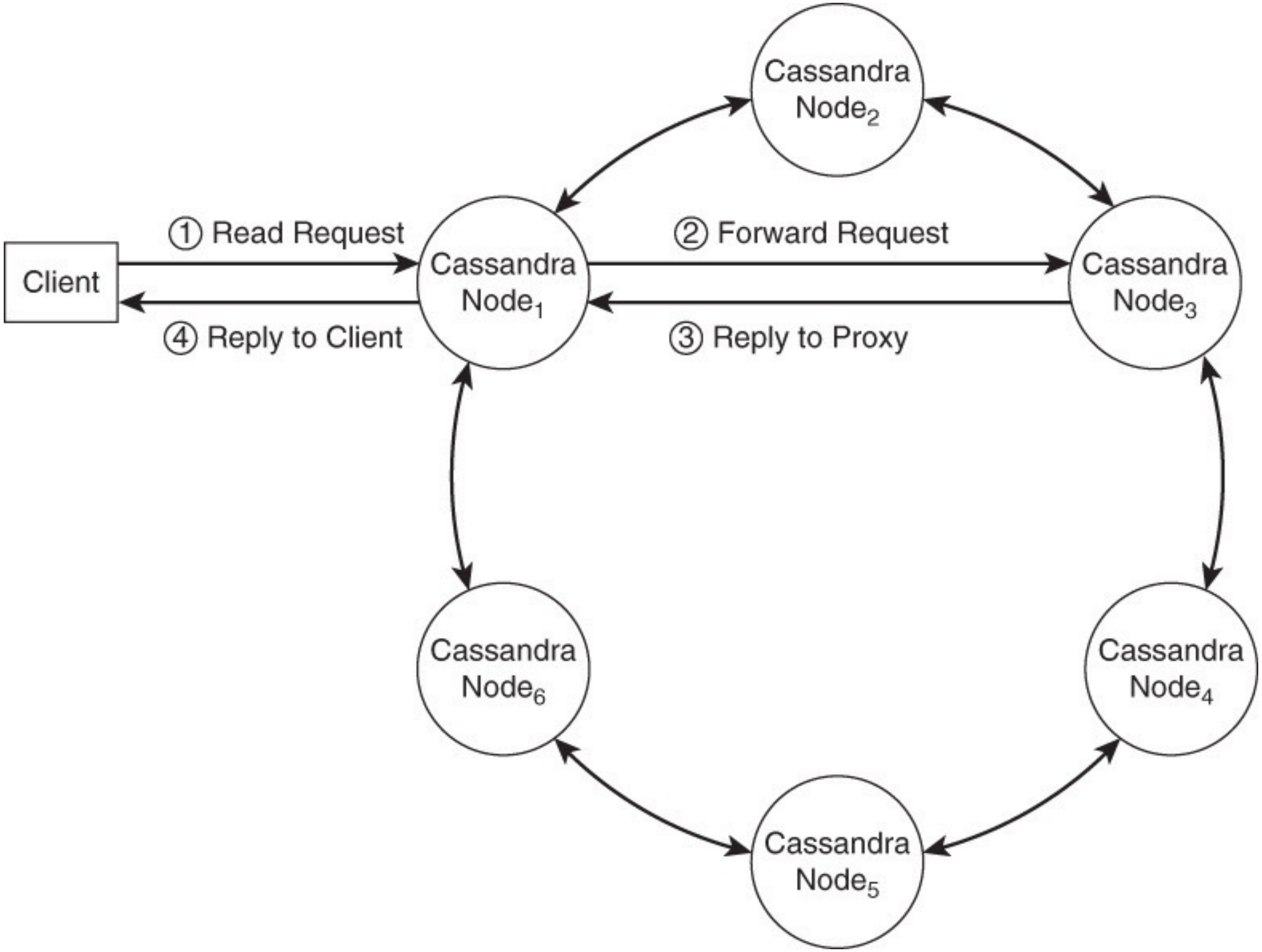
(a) Centralized approach, which uses a centralized server for forwarding and mixing of multimedia streams

(b) Fully connected overlay approach, where all peers are in direct contact with one another

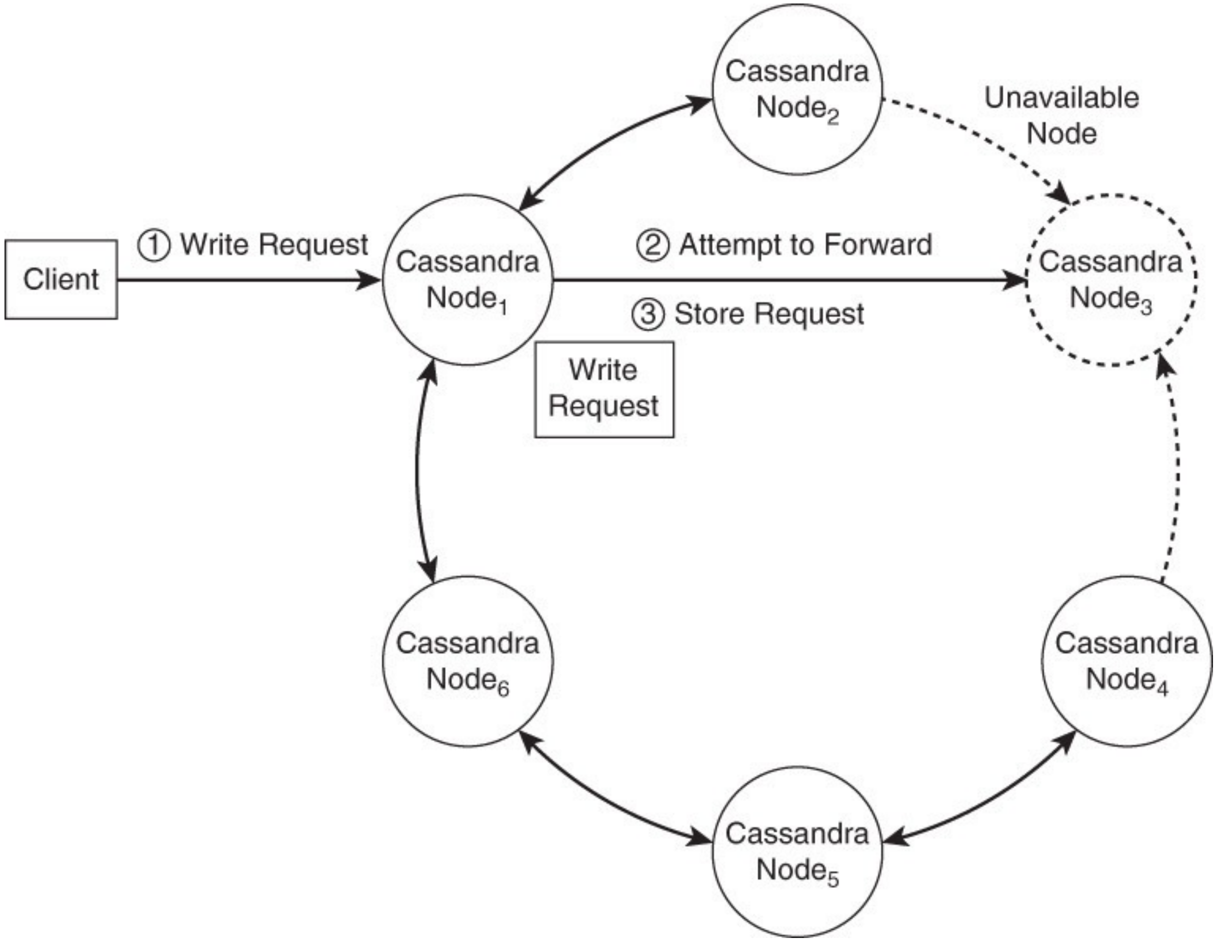
(c) Gossip-based approach, where peers operate in parallel, and each peer communicates with one or more randomly selected partner

Image extracted from: Luk, V.WH., Wong, A.KS., Lea, CT. et al. J Internet Serv Appl (2013) 4: 14. https://doi.org/10.1186/1869-0238-4-14

Cassandra Read Protocol



Cassandra Write Protocol



Hinted Handoff

What happens if the node interested in a write operation is down?

Two scenarios may happen:

1) The operation will be lost, thus the target node will be in inconsistent state after its recover

2) A specific node can be designated to receive write operations and forward them to the target node when it becomes available.

Hinted Handoff

The hinted handoff ***process*** is in charge of:

1. ***Creating*** a data structure to store information about the write operation and where it should ultimately be sent

2. Periodically ***checking*** the status of the target server

3. ***Sending*** the write operation when the target is available

Once the write data is ***successfully written*** to the target node, it can be considered a successful write for the purposes of ***consistency*** and ***replication***.

Suggested Readings

Chapter 9 of the book “*Dan Sullivan, NoSQL For Mere Mortals, Addison-Wesley, 2015”.*

Chapter 10 of the book “*Dan Sullivan, NoSQL For Mere Mortals, Addison-Wesley, 2015.*