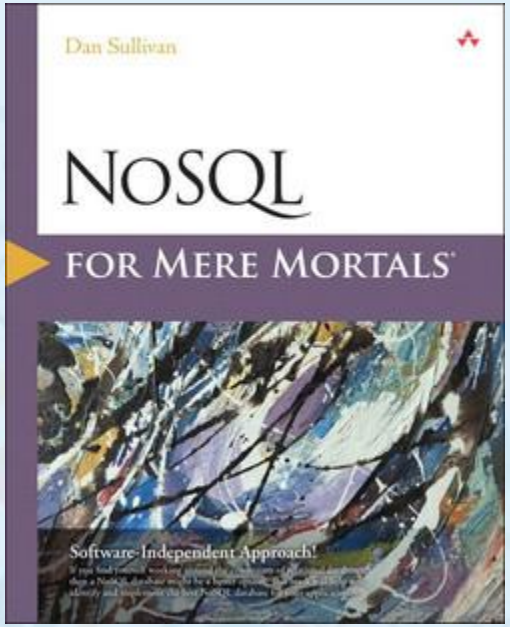
Introduction to Column Family Databases

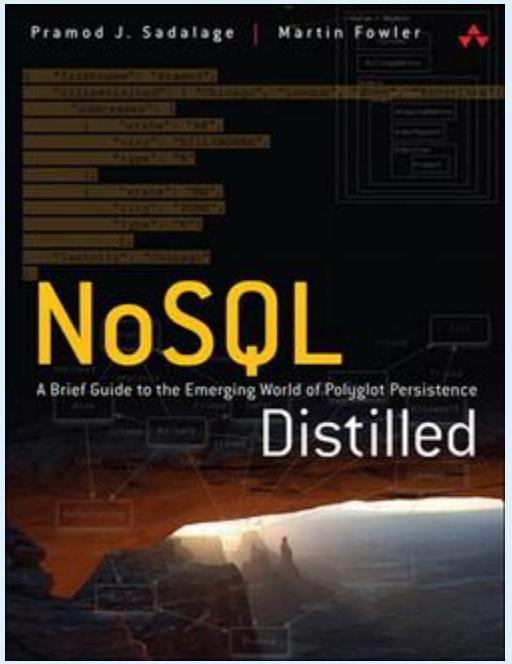
06016414 and 06026207: NoSQL Database Systems

Instructor: Asst. Prof. Dr. Praphan Pavarangkoon

Books

NoSQL for Mere Mortals by Dan Sullivan





NoSQL Distilled A Brief Guide to the Emerging World of Polyglot Persistence by Pramod J. Sadalage, Martin Fowler

Outline

- Introduction to Column Family Databases
  - In the Beginning, There Was Google BigTable
  - Differences and Similarities to Key-Value and Document Databases
  - Architectures Used in Column Family Databases
- Lab Session
  - Get Started with Apache Cassandra

What is Big Data?

- Deciding what is Big Data or a large database is somewhat subjective.
- Are a million rows in a MySQL table a large database?
- Very large database (VLDB):
  - Realm of billions of rows and tens of thousands of columns in a table

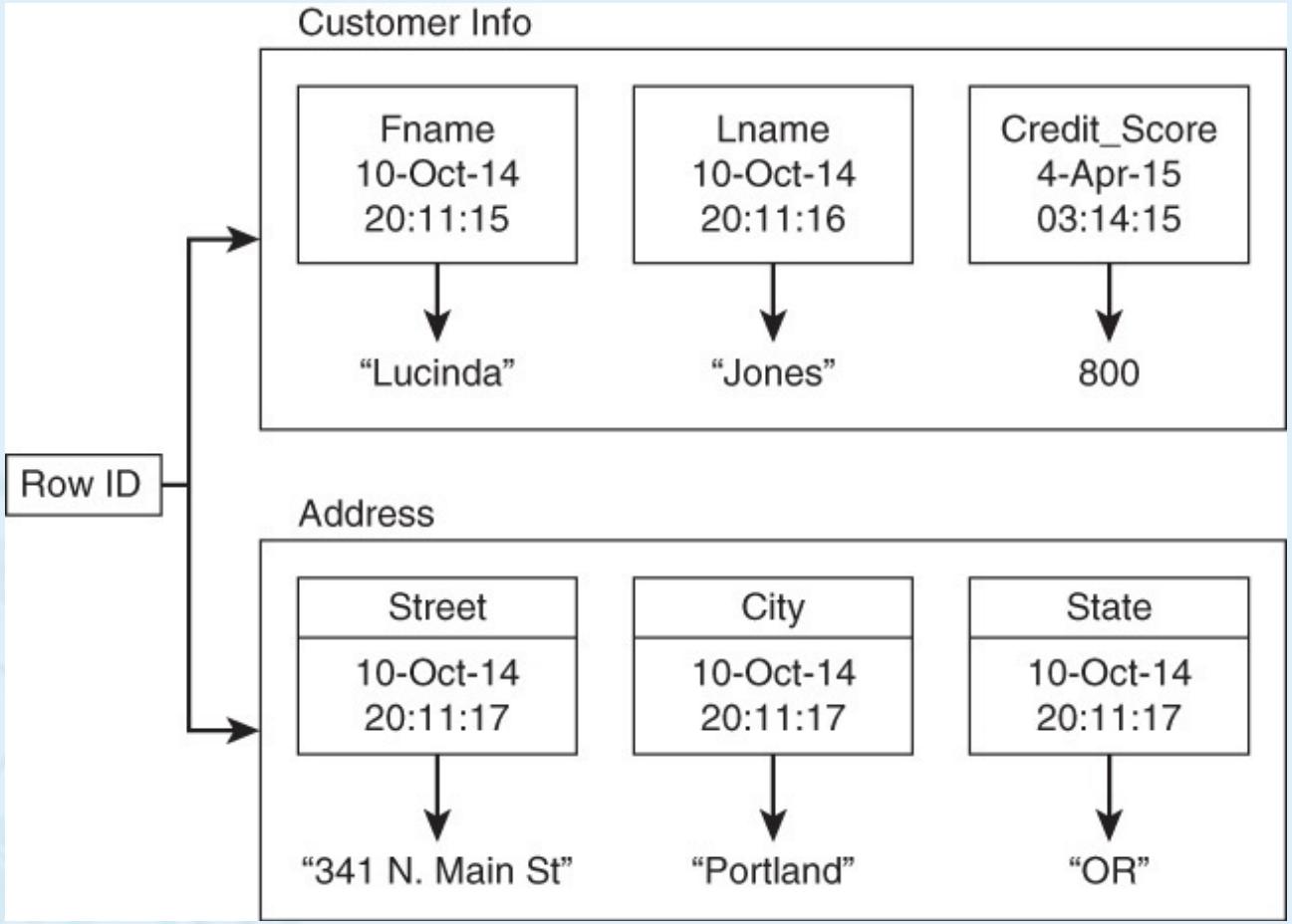
What is Big Data? (cont.)

- Companies such as Google, Facebook, Amazon, and Yahoo! must contend with demands for very large database management solutions.
- In 2006, Google published a paper entitled “BigTable: A Distributed Storage System for Structured Data.”.
- BigTable became the model for implementing very large-scale NoSQL databases.
- Other column family databases include Cassandra, HBase, and Accumulo.

In the Beginning, There Was Google BigTable

- The following are core features of Google BigTable:
  - Developers have dynamic control over columns.
  - Data values are indexed by row identifier, column name, and a time stamp.
  - Data modelers and developers have control over location of data.
  - Reads and writes of a row are atomic.
  - Rows are maintained in a sorted order.

A row in a column family database is organized as a set of column families.

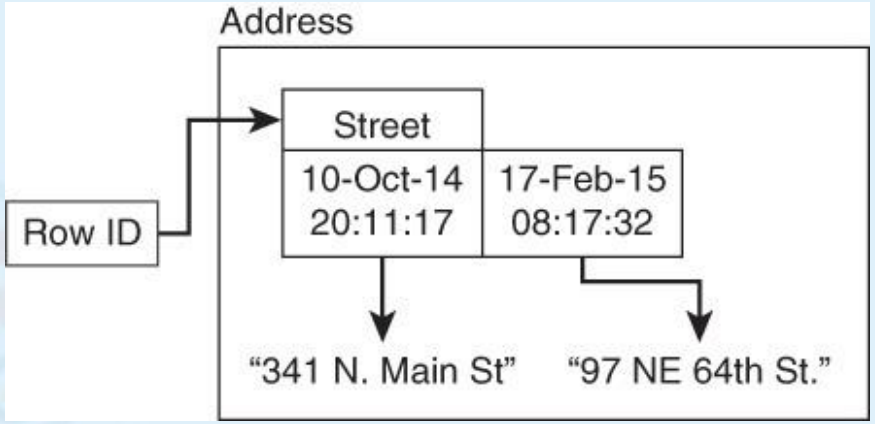


Utilizing Dynamic Control over Columns

- The use of column families and dynamic columns enables database modelers to define broad, course-grained structures (that is, column families) without anticipating all possible fine-grained variations in attributes.

Indexing by Row, Column Name, and Time Stamp

- In BigTable, a data value is indexed by its row identifier, column name, and time stamp.
  - The row identifier is analogous to a primary key in a relational database.
  - The column name uniquely identifies a column.
  - The time stamp orders versions of the column value.

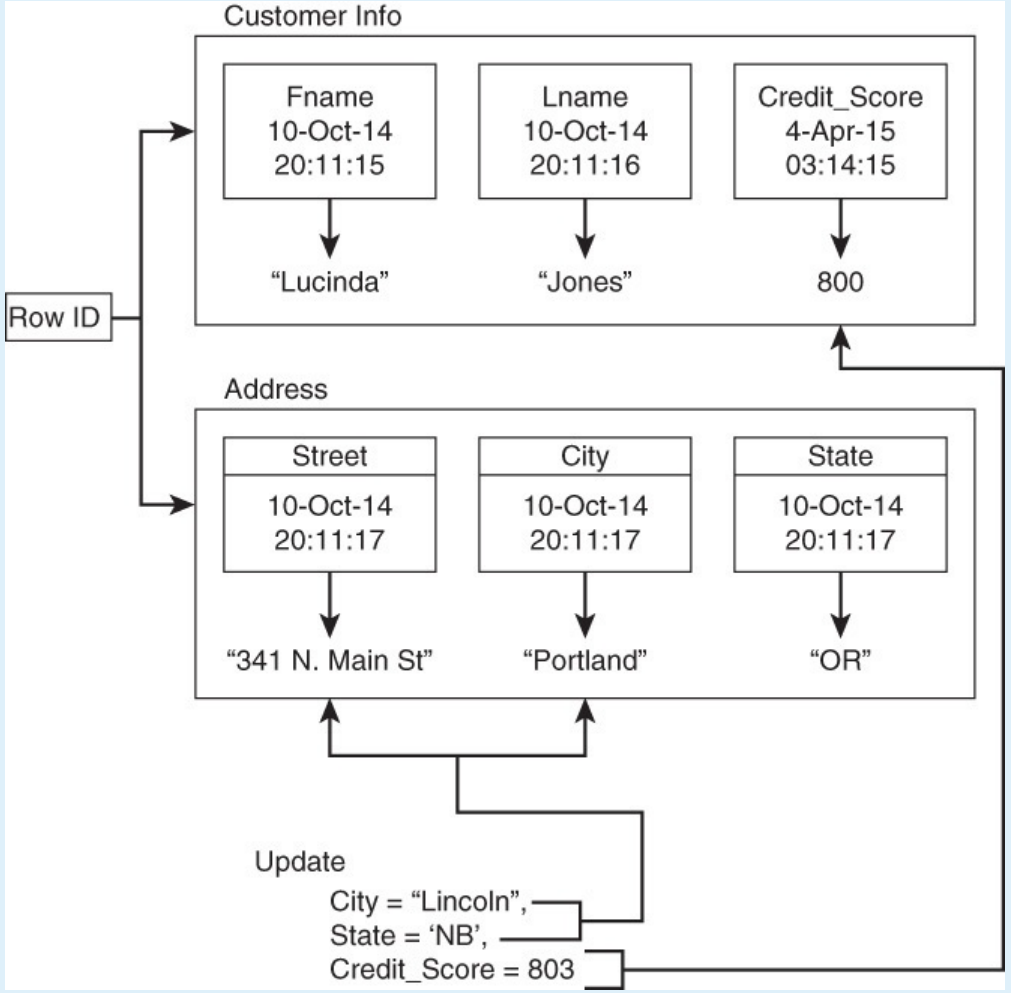


Controlling Location of Data

- One way to avoid the need to read multiple blocks of data located on different parts of the disk is to keep data close together when it is frequently used together.
- Columns families store columns together in persistent storage, making it more likely that reading a single data block can satisfy a query.

Reading and Writing Atomic Rows

- The designers of BigTable decided to make all read and write operations atomic regardless of the number of columns read or written.
- This means that as you read a set of columns, you will be able to read all the columns needed or none of them.



Maintaining Rows in Sorted Order

- BigTable maintains rows in sorted order.
- This makes it straightforward to perform range queries.
- Google BigTable introduced a data management system designed to scale to petabytes of data using commodity hardware.
- The design balanced data modeling features with the need to scale.
- The designers of Google BigTable anticipated the need for hundreds of column families, tens of thousands (or more) columns, and billions of rows.

Differences and Similarities to Key-Value and Document Databases

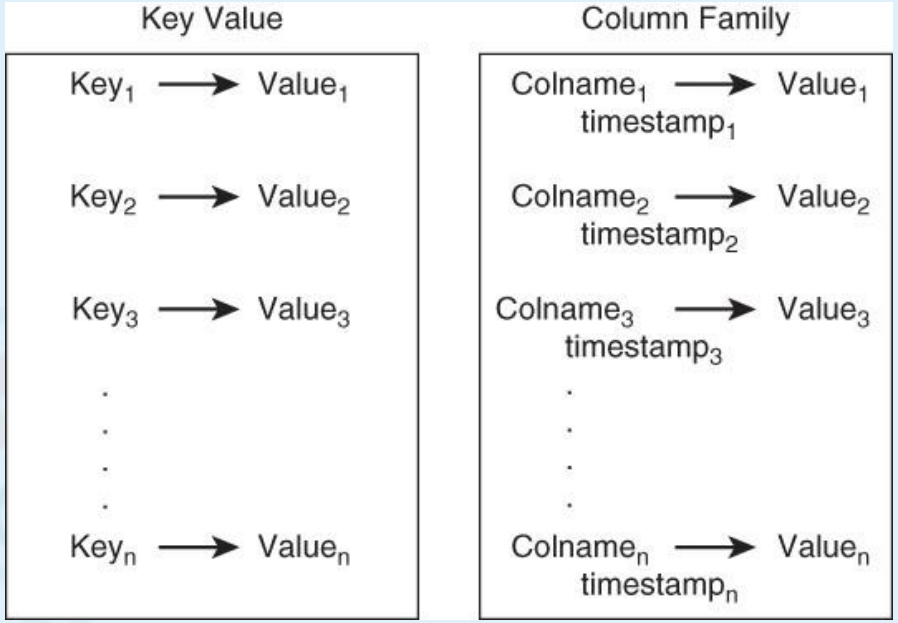
- Column family databases have characteristics similar to other NoSQL databases—key-value and document databases in particular.
- In addition, many NoSQL databases employ distributed database techniques to address scalability and availability concerns.

Column Family Database Features

- Key-value databases are the simplest of all NoSQL architectures.
- They consist of a keyspace, which is essentially a logical structure for isolating keys and values maintained for a particular purpose.
- Column families are analogous to keyspaces in key-value databases.
- Developers are free to add keys and values in key-value databases just as they are free to add columns and values to column families.

Column Family Database Features (cont.)

- In Cassandra terminology, a keyspace is analogous to a database in relational databases.
- Unlike key-value databases, the values in columns are indexed by a row identifier as well as by a column name (and time stamp).



Column Family Database Similarities to and Differences from Document Databases

- Document databases extend the functionality found in key-value databases by allowing for highly structured and accessible data structures.
- Documents are analogous to rows in a relational database and store multiple fields of data, typically in a JSON or XML structure.

Column Family Database Similarities to and Differences from Document Databases (cont.)

- If you stored the following document in a key-value database, you could set or retrieve the entire document, but you could not query and extract a subset of the data, such as the address.

```
{
    "customer_id":187693,
    "name": "Kiera Brown",
    "address" : {
        "street" : "1232 Sandy Blvd.",
        "city" :  "Vancouver",
        "state" :  "Washington",
        "zip" :  "99121"
                }
   "first_order" : "01/15/2013",
   "last_order" : "06/27/2014"
}
```

Column Family Database Similarities to and Differences from Document Databases (cont.)

- Document databases enable you to query and filter based on elements in the document.
- For example, you could retrieve the address of customer Kiera Brown with the following command (using MongoDB syntax):

```
db.customers.find( { "customer_id":187693 }, { "address":
  1 } )
```

- Column family databases support similar types of querying that allow you to select subsets of data available in a row.
- Cassandra uses a SQL-like language called Cassandra Query Language (CQL) that uses the familiar SELECT statement to retrieve data.

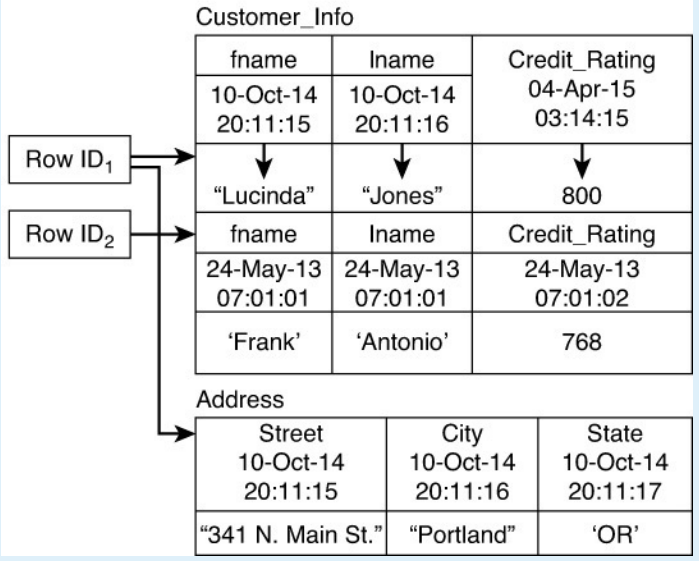
Column Family Database Similarities to and Differences from Document Databases (cont.)

- Column family databases, like document databases, do not require all columns in all rows.
- Some rows in a column family database may have values for all columns, whereas others will have values for only some columns in some column families.

Document Database:

{fname: 'Lucinda',
lname: 'Jones',
Credit_Rating: 800,
Address:
{Street: '341 N. Main St.',
City: 'Portland',
State: 'OR'
}
{fname: 'Frank',
lname: 'Antonio',
Credit_Rating: 768
}

Column Family Database:



Column Family Database Versus Relational Databases

- Both column family databases and relational databases use unique identifiers for rows of data.
- These are known as row keys in column family databases and as primary keys in relational databases.
- Both row keys and primary keys are indexed for rapid retrieval.

Column Family Database Versus Relational Databases (cont.)

- Column family databases use the concept of maps (also known as dictionaries or associative arrays).
- A column key maps from a column name to a column value.

Row Key A	Column 1 Key	Column 2 Key	Column 3 Key	...	Column N Key
	Column 1 Value	Column 2 Value	Column 3 Value	...	Column N Value

- Other important differences between column family databases and relational databases pertain to typed columns, transactions, joins, and subqueries.
- Column family databases do not support the concept of a typed column.

Avoiding Multirow Transactions

- Although you can expect to find atomic reads and writes with respect to a single row, column family databases such as Cassandra do not support multirow transactions.

Avoiding Subqueries

- There should be minimal need for joins and subqueries in a column family database.
- Column families promote denormalization and that eliminates, or at least reduces, the need for joins.
- In relational databases, a subquery is an inner query that runs, typically, as part of the WHERE clause of an outer query.

Avoiding Subqueries (cont.)

- For example, you might need to select all sales transactions performed by a salesperson with a last name of Smith. A SQL query such as the following could be used:

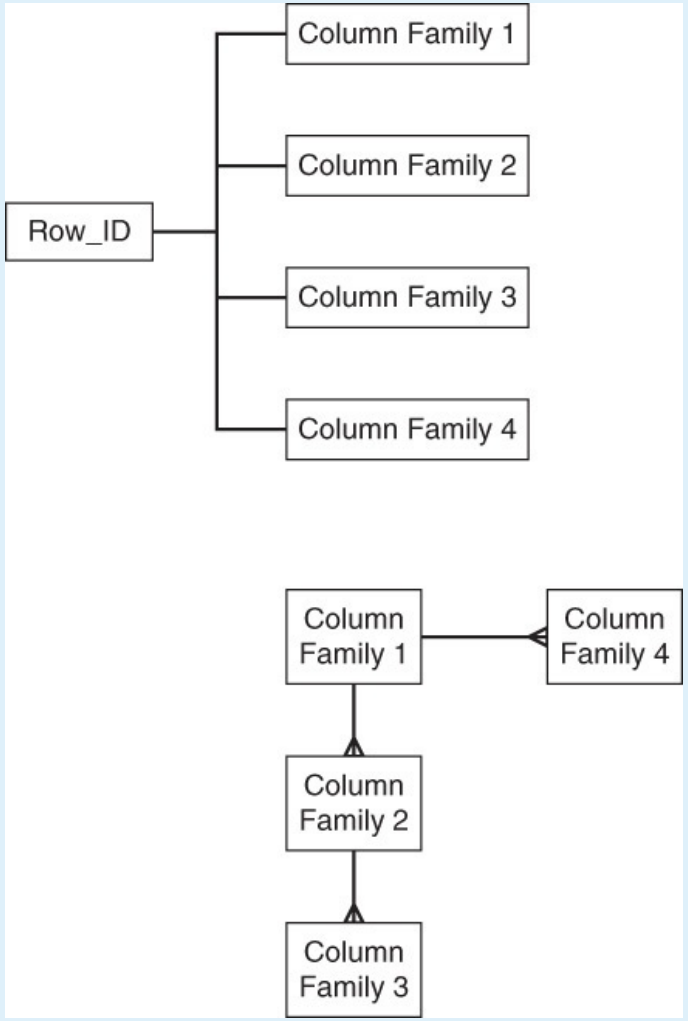
```
SELECT
   *
FROM
   sales_transactions
WHERE
SELECT
    sales_person_id
FROM
   sales_persons
WHERE
   last_name = 'Smith'
```

Avoiding Subqueries (cont.)

- The part of the statement that begins with SELECT sales_person_id FROM… is a subquery and executes in the context of the outer query.
- These types of subqueries are supported by relational databases but not by column family databases.

Avoiding Subqueries (cont.)

- Instead, a column family with salesperson information could be included with sales transaction data that would likely be maintained in another column family.



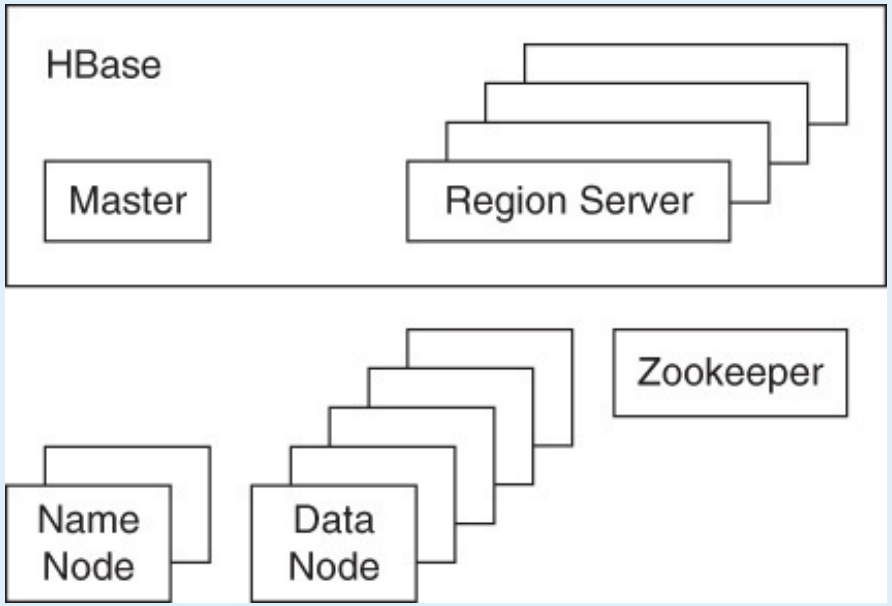
Architectures Used in Column Family Databases

- Broadly speaking, there are two commonly used types of architectures used with distributed databases: multiple node type and peer-to-peer type.
- Multiple node type architectures have at least two types of nodes, although there may be more.
  - HBase is built on Hadoop and makes use of various Hadoop nodes, including name nodes, data nodes, and a centralized server for maintaining configuration data about the cluster.
- Peer-to-peer type architectures have only one type of node.
  - Cassandra, for example, has a single type of node.
  - Any node can assume responsibility for any service or task that must be run in the cluster.

HBase Architecture: Variety of Nodes

- Apache HBase uses the Hadoop infrastructure.
- The Hadoop File System, HDFS, uses a master-slave architecture that consists of name nodes and data nodes.
- Zookeeper is a type of node that enables coordination between nodes within a Hadoop cluster.
- RegionServers are instances that manage Regions, which are storage units for HBase table data.

HBase Architecture: Variety of Nodes (cont.)

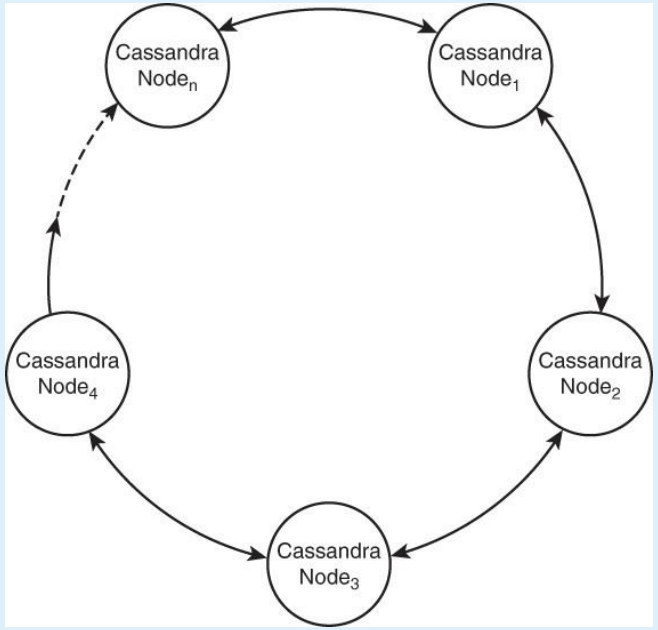


- An advantage of this type of architecture is that servers can be deployed and tuned for specific tasks.

Cassandra Architecture: Peer-to-Peer

- Apache Cassandra, like Apache HBase, is designed for high availability, scalability, and consistency.
- Rather than use a hierarchical structure with fixed functions per server, Cassandra uses a peer-to-peer model.
- All Cassandra nodes run the same software.

Cassandra Architecture: Peer-to-Peer (cont.)



- There are several advantages to the peer-to-peer approach.
  - The first is simplicity: No node can be a single point of failure.
  - Scaling up and down is fairly straightforward: Servers are added or removed from the cluster.

Cassandra Architecture: Peer-to-Peer (cont.)

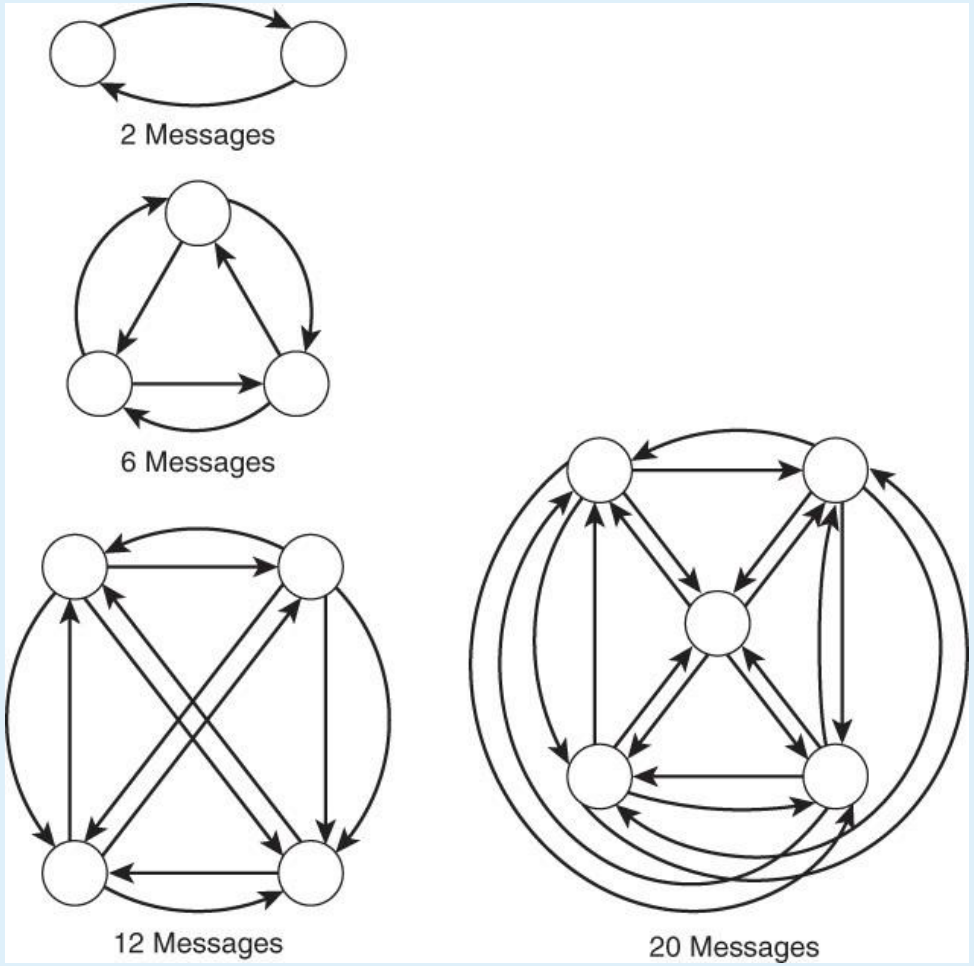
- The servers in the cluster are responsible for managing a number of operations, including the following:
  - Sharing information about the state of servers in the cluster
  - Ensuring nodes have the latest version of data
  - Ensuring write data is stored when the server that should receive the write is unavailable
- Cassandra has protocols to implement all of these functions.

Getting the Word Around: Gossip Protocol

- The problem is that this type of all-servers-to-all-other-servers protocol can quickly increase the volume of traffic on the network and the amount of time each server has to dedicate to communicating with other servers.

Getting the Word Around: Gossip Protocol (cont.)

- Consider a variety of scenarios.
- The number of messages sent is a function of the number of servers in the cluster.
- If *N* is the number of servers, then $N \times (N-1)$ is the number of messages needed to update all servers with information about all other servers.



Getting the Word Around: Gossip Protocol (cont.)

- A more-efficient method of sharing information is to have each server update another server about itself as well as all the servers it knows about.
- To get an idea of how efficient an information-sharing scheme can be, consider a seven-node cluster.
  1. Servers 1 and 2 send status information to Server 3.
  2. Servers 4 and 5 send status information to Server 6.
  3. Servers 3 and 6 send their own status information plus status information about two other servers to Server 7.
  4. Server 7 now has information about every server in the cluster.
  5. Server 7 sends the complete set of information to Servers 3 and 6.
  6. Server 3 then passes the information on to Servers 1 and 2 while Server 6 passes the information on to Servers 4 and 5.
  7. All nodes in the cluster now have complete status information about the cluster.

Getting the Word Around: Gossip Protocol (cont.)

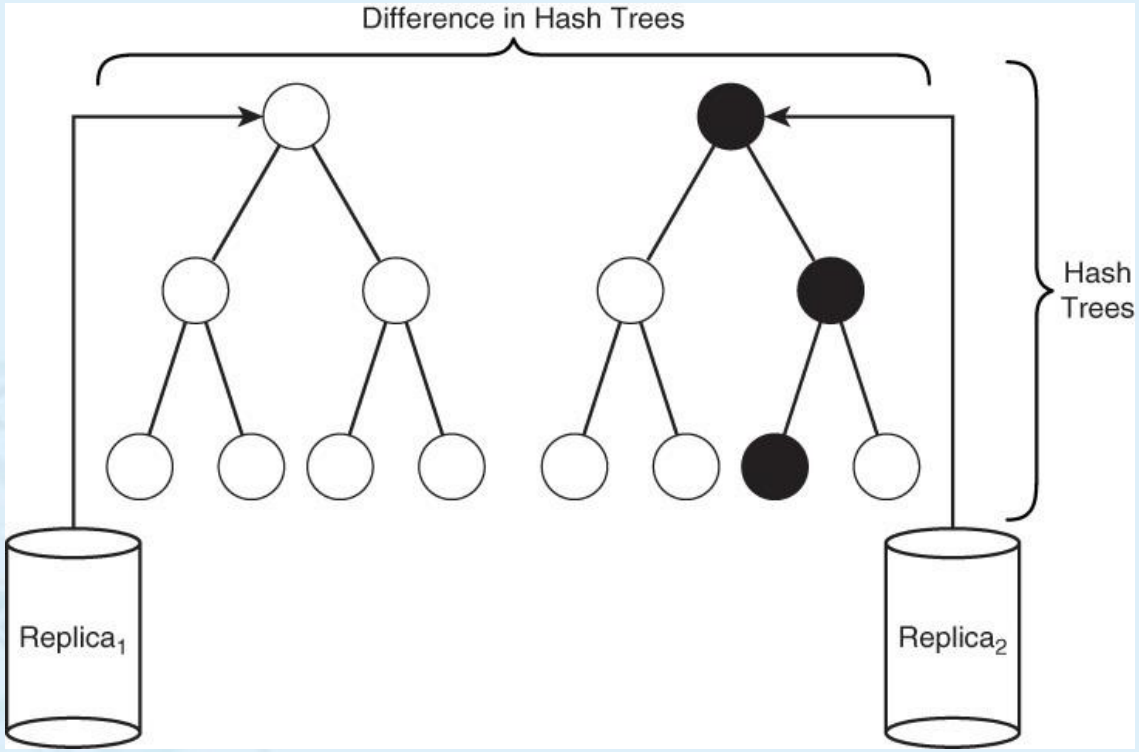
- Cassandra's gossip protocol works as follows.
  - A node in the cluster initiates a gossip session with a randomly selected node.
  - The initiating node sends a starter message (known as a Gossip-DigestSyn message) to a target node.
  - The target node replies with an acknowledgment (known as a GossipDigestAck message).
  - After receiving the acknowledgment from the target node, the initiating node sends a final acknowledgment (a GossipDigestAck2 message) to the target node.

Thermodynamics and Distributed Database: Why We Need Anti-Entropy

- The second law of thermodynamics describes a feature of entropy, which is the state of randomness and lack of order in a system or object.
- Distributed database designers have to address information entropy.
  - Information entropy increases when data is inconsistent in the database.

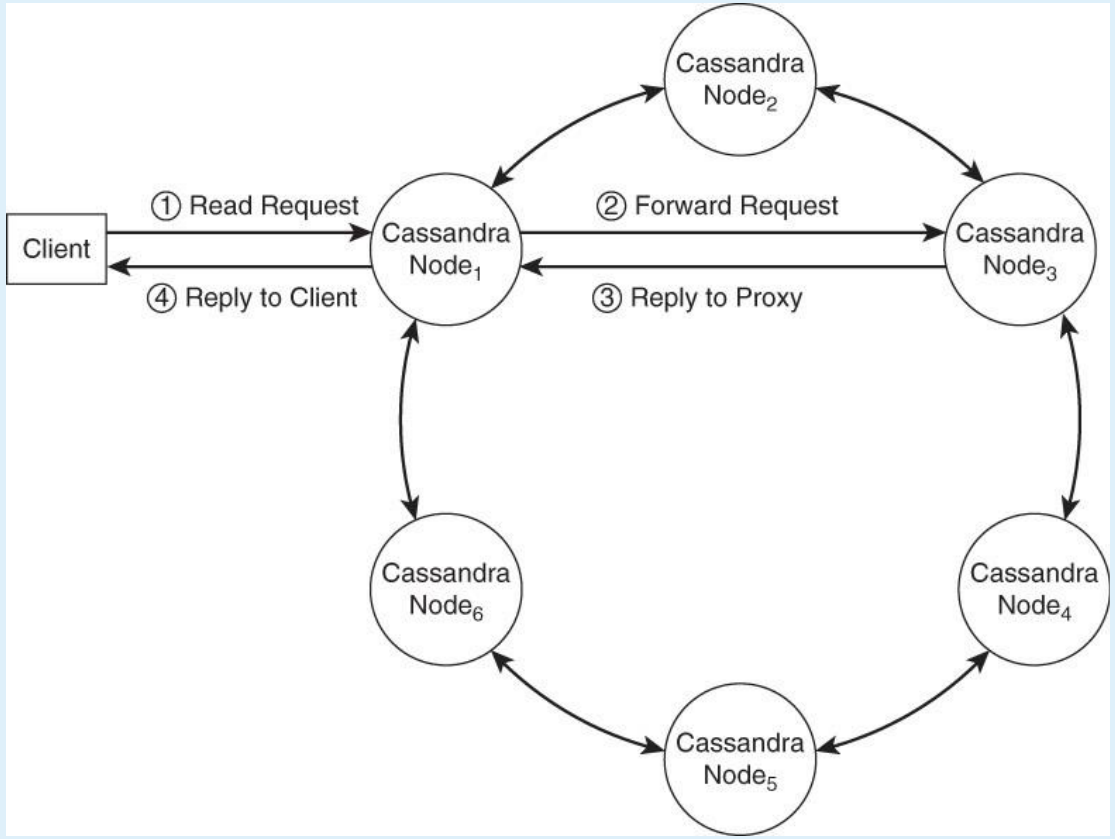
Thermodynamics and Distributed Database: Why We Need Anti-Entropy (cont.)

- Cassandra uses an anti-entropy algorithm, that is, one that increases order, to correct inconsistencies between replicas.



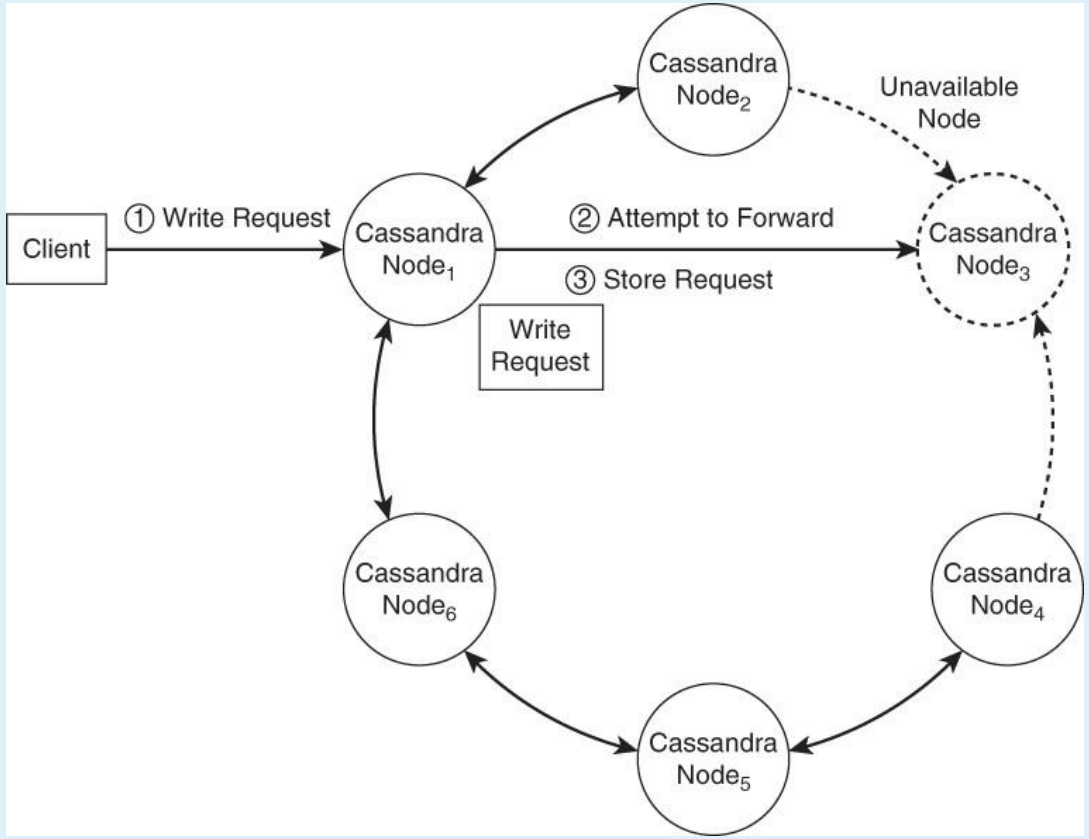
Hold This for Me: Hinted Handoff

- Cassandra is known for being well suited for write-intensive applications.
- This is probably due in part to its ability to keep accepting write requests even when the server that is responsible for handling the write request is unavailable.



Hold This for Me: Hinted Handoff (cont.)

- A hinted handoff entails storing information about the write operation on a proxy node and periodically checking the status of the unavailable node.
- When that node becomes available again, the node with the write information sends, or "hands off," the write request to the recently recovered node



When to Use Column Family Databases

- Column family databases are appropriate choices for large-scale database deployments that require high levels of write performance, a large number of servers or multi-data center availability.
- Cassandra's peer-to-peer architecture with support for hinted handoff means the database will always be able to accept write operations as long as at least one node is functioning and reachable.
  - Write-intensive operations, such as those found in social networking applications, are good candidates for using column family databases.

When to Use Column Family Databases (cont.)

- Column family databases are also appropriate when a large number of servers are required to meet expected workloads.
- Column family databases typically run with more than several servers.
- Cassandra supports multi-data center deployment, including multi-data center replication.

Lab Session

- Get Started with Apache Cassandra

Get Started with Apache Cassandra

- Step 1: Install with Docker
  Pull the docker image. For the latest image, use:

```
docker pull cassandra:latest
```

  This `docker pull` command will get the latest version of the official Docker Apache Cassandra image available from the Dockerhub.

Get Started with Apache Cassandra (cont.)

- Step 2: Start Cassandra with a docker run command

```
docker run --name cass_cluster cassandra:latest
```

The `--name` option will be the name of the Cassandra cluster created. This example uses the name `cass_cluster`.

Note: Replace `cass_cluster` with `cass_<Student ID>`.

Alternative Solution: Starting Cassandra Without Internet

- Step 1: Load the Docker Image
  Use the .tar file you prepared:

```
docker load < C:\Aj.Praphan\cassandra.tar
```

  This command imports the Cassandra image into Docker.

Alternative Solution: Starting Cassandra Without Internet (cont.)

- Step 2: Start Cassandra
  Run the container from the loaded image:

```
docker run --name cass_cluster cassandra
```

Note: Replace `cass_cluster` with `cass_<Student ID>`.

Get Started with Apache Cassandra (cont.)

- Step 3: Start the CQL shell, cqlsh to interact with the Cassandra node created

```
docker exec -it cass_cluster cqlsh
```

This should get you a prompt like so:

```
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 6.0.1 | Cassandra 4.1.3 | CQL spec 3.4.6 | Native protocol v5]
Use HELP for help.
cqlsh>
```

Note: Replace `cass_cluster` with `cass_<Student ID>`.

Get Started with Apache Cassandra (cont.)

- Step 4: Create a keyspace

```
CREATE KEYSPACE IF NOT EXISTS store WITH
REPLICATION = { 'class' : 'SimpleStrategy',
'replication_factor' : '1' };
```

- Step 5: Create a table

```
CREATE TABLE IF NOT EXISTS store.shopping_cart (
userid text PRIMARY KEY,
item_count int,
last_update_timestamp timestamp
);
```

Get Started with Apache Cassandra (cont.)

- Step 6: Insert some data

```
INSERT INTO store.shopping_cart
(userid, item_count, last_update_timestamp)
VALUES ('9876', 2, toTimeStamp(now()));
INSERT INTO store.shopping_cart
(userid, item_count, last_update_timestamp)
VALUES ('1234', 5, toTimeStamp(now()));
```

Get Started with Apache Cassandra (cont.)

- Step 7: Read some data

```
SELECT * FROM store.shopping_cart;
```

- Step 8: Write some more data

```
INSERT INTO store.shopping_cart
(userid, item_count)
VALUES ('4567', 20);
```

Congratulations!

Q & A