HBase Essentials

Compiled by Amit S Khedkar

HBase Essentials

Objectives

After completing this lesson, students should be able to get:

- Overview of Hbase
- Limitations of a Relational Model
- Specifics of the HBase data model

Lab: Pre-lab Setup

Lab: INTRODUCTION TO APACHE HBASE CONCEPTS

HBase : Need of HBase

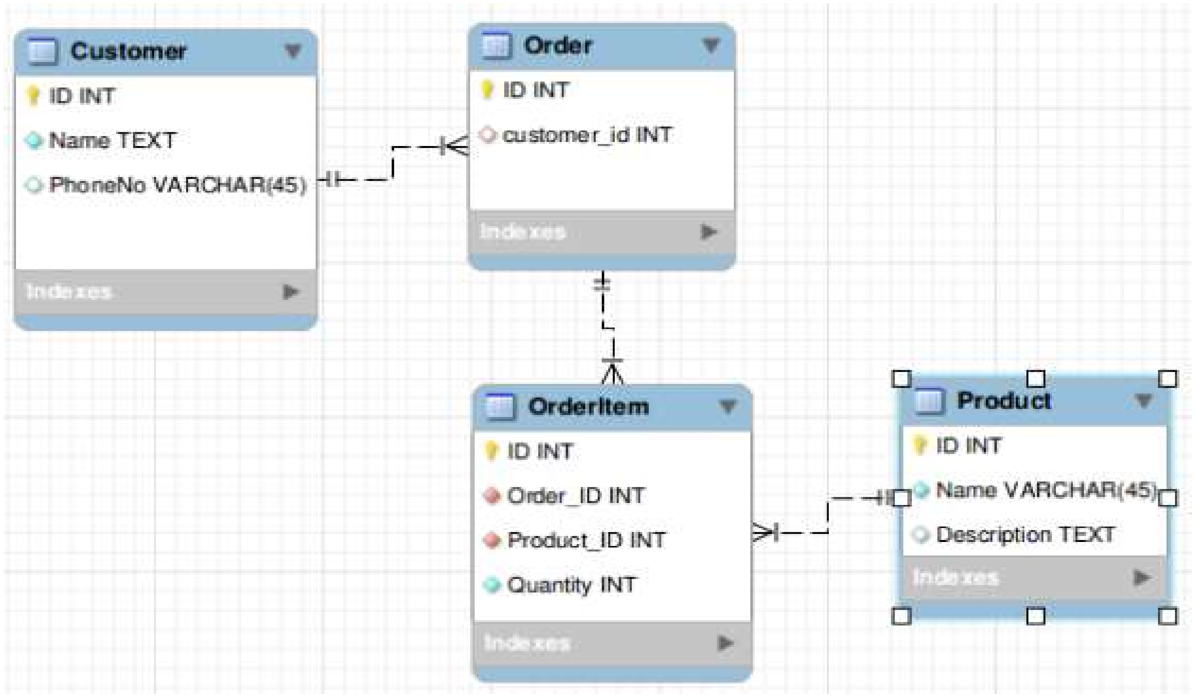
Introduction

- Apache HBase is a database that runs on a Hadoop cluster.
- HBase is not a traditional RDBMS
- Data stored in HBase also does not need to fit into a rigid schema like with an RDBMS
- HBase allows you to build big data applications for scaling, with low latency

Relational Databases vs. HBase - Data Storage Model

- Why do we need NoSQL/HBase?
- First, let’s look at the pros of relational databases before we discuss its limitations:
  - Relational databases have provided a standard persistence model
  - SQL has become a de-facto standard model of data manipulation
  - Relational databases manage concurrency for transactions
  - Relational database have lots of tools

Relational Databases vs. HBase - Data Storage Model

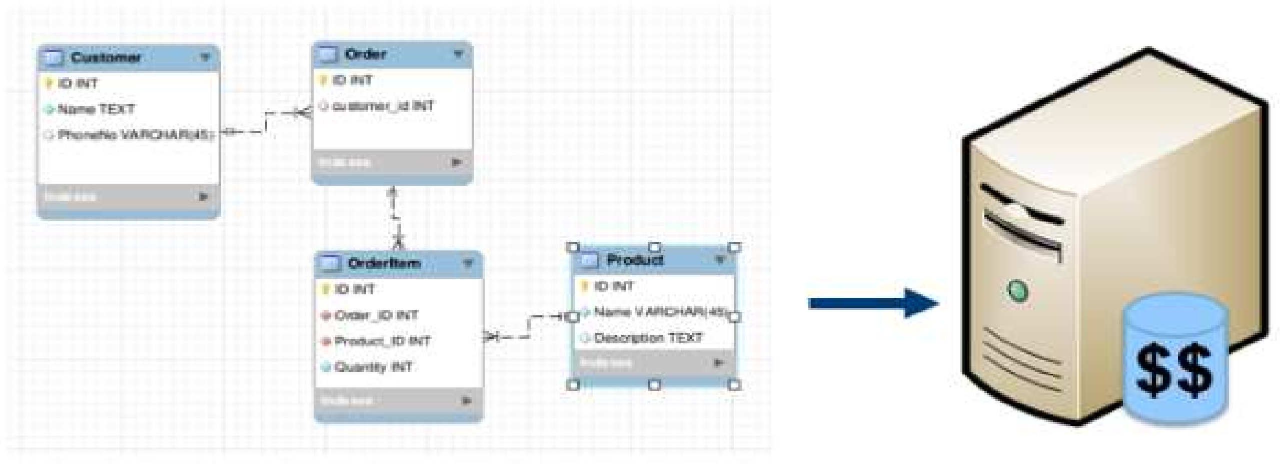


Relational Databases vs. HBase - Data Storage Model

- so what changed?
- With more and more data came the need to scale.
- One way to scale is vertically with a bigger server, but this can get expensive, and there are limits as your data size increases.

Relational Databases vs. HBase - Data Storage Model

RDBMS - Scale UP approach

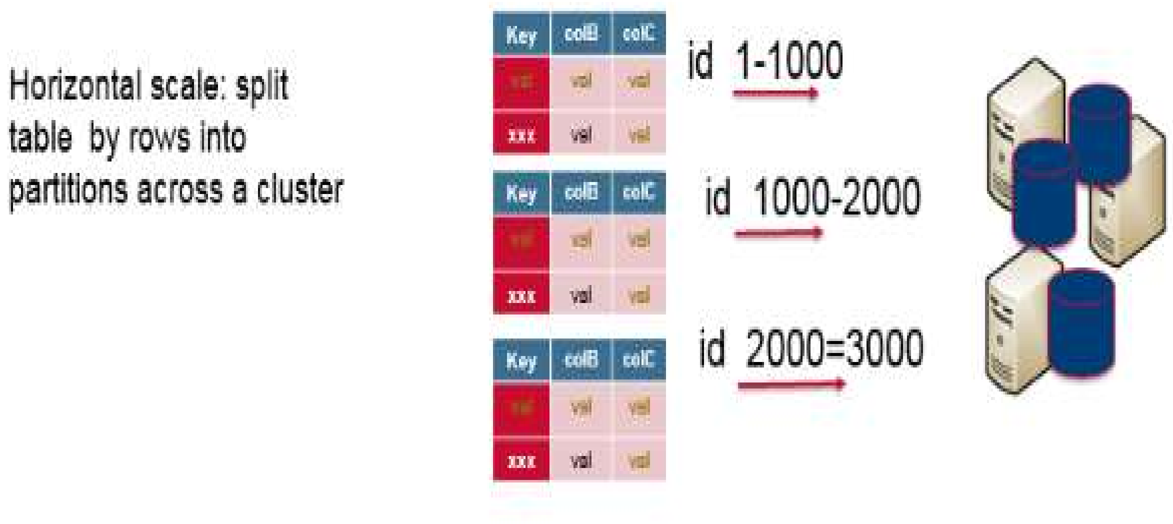


Vertical scale = **big box**

What changed to bring on NoSQL?

- An alternative to vertical scaling is to scale horizontally with a cluster of machines, which can use commodity hardware.
- This can be cheaper and more reliable.
- To horizontally partition or shard a RDBMS, data is distributed on the basis of rows, with some rows residing on a single machine and the other rows residing on other machines

What changed to bring on NoSQL?

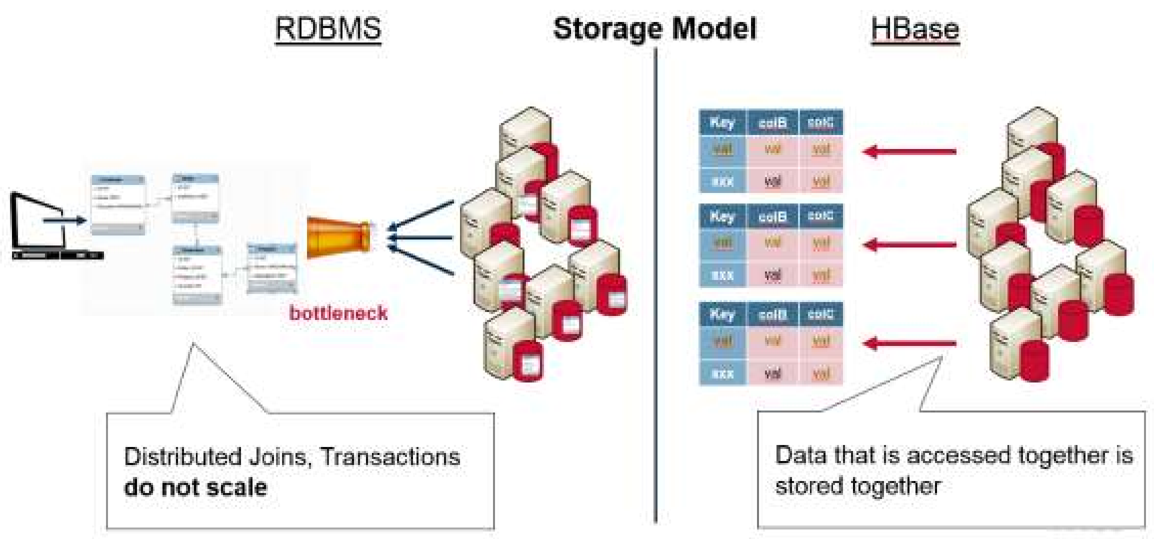


Horizontal scale: split table by rows into partitions across a cluster
Key
colB
colC
val
xxx
id 1-1000
id 1000-2000
id 2000=3000

What changed to bring on NoSQL?

- There are Limitations of a Relational Model
  - Database normalization eliminates redundant data, which makes storage efficient. However, a normalized schema causes joins for queries, in order to bring the data back together again.
- While HBase does not support relationships and joins, ***data that is accessed together is stored together*** so it avoids the limitations associated with a relational model

What changed to bring on NoSQL?


RDBMS
Storage Model
HBase
bottleneck
Key
colB
colC
val
xxx
Distributed Joins, Transactions do not scale
Data that is accessed together is stored together

HBase Designed for Distribution, Scale, and Speed

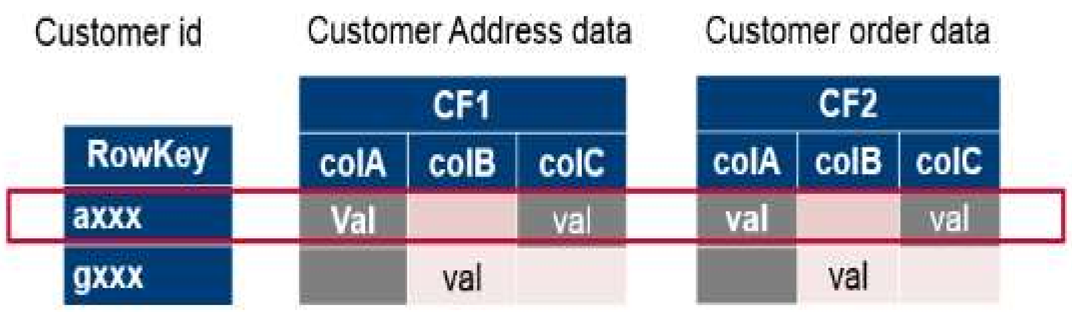
Distribution, Scale, and Speed

- HBase was designed to scale due to the fact that ***data that is accessed together is stored together***.
- Grouping the data by key is central to running on a cluster.
- In horizontal partitioning or sharding, the key range is used for sharding, which distributes different data across multiple servers.
- Each server is the source for a subset of data.

Distribution, Scale, and Speed

- HBase is referred to as a ***column family***-oriented data store.
- It's ***also row-oriented***: each row is indexed by a key that you can use for lookup (for example, lookup a customer with the ID of 1234).
- Each column family groups like data (customer address, order) within rows.
- Think of a row as the join of all values in all column families.

Distribution, Scale, and Speed



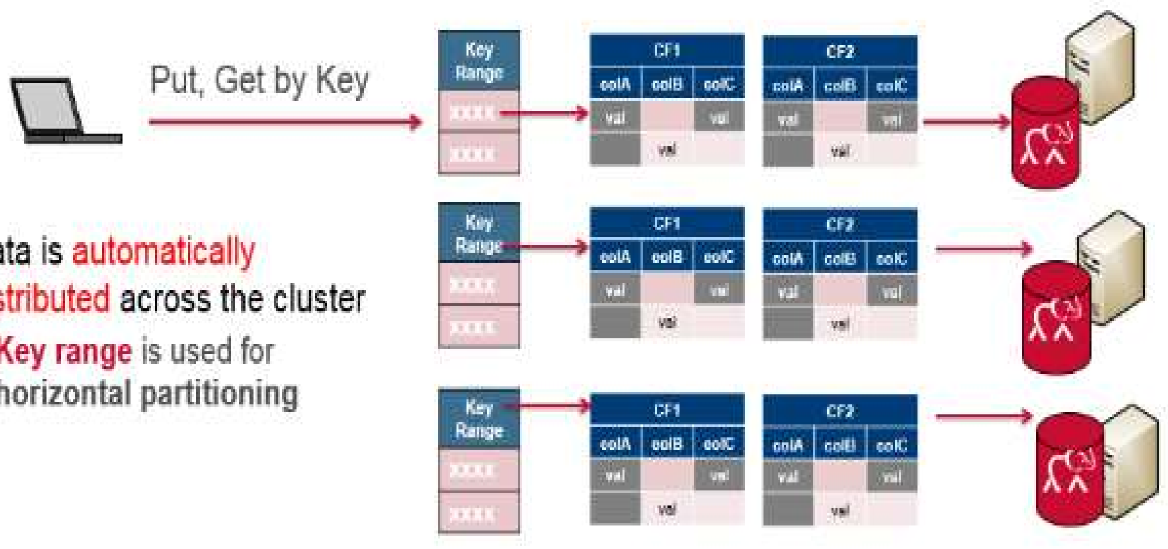
Data is accessed and stored together:

- RowKey is the primary index
- Column Families group similar data by row key

Distribution, Scale, and Speed

- HBase is also considered a ***distributed*** database.
- Grouping the data by key is central to running on a cluster and sharding.
- The key acts as the atomic unit for updates.

Distribution, Scale, and Speed



Data is automatically distributed across the cluster

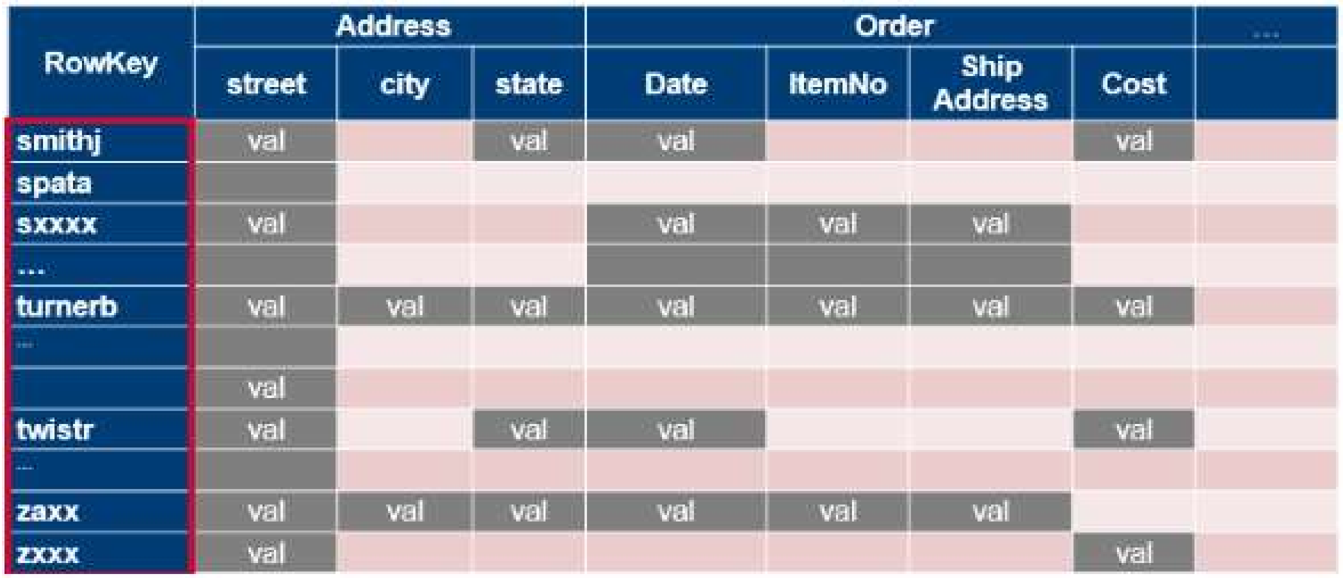
- Key range is used for horizontal partitioning

HBase Data Model

Data Model

- Data stored in HBase is located by its ***rowkey***
- This is like a primary key from a relational database.
- Records in HBase are stored in sorted order, according to rowkey.

Data Model

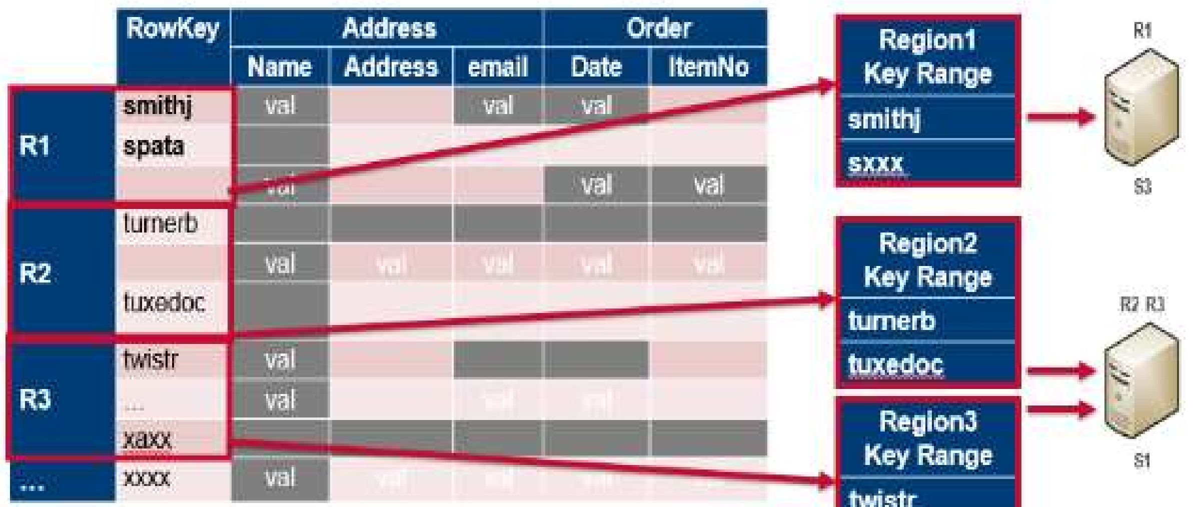


RowKey	Address			Order				...
	street	city	state	Date	ItemNo	Ship Address	Cost	
smithj	val		val	val			val	
spata								
sxxxx	val			val	val	val		
...								
turnerb	val	val	val	val	val	val	val	
...								
	val							
twistr	val		val	val			val	
...								
zaxx	val	val	val	val	val	val		
zxxx	val						val	

Data Model

- Tables are divided into sequences of rows, by key range, called ***regions***.
- These regions are then assigned to the data nodes in the cluster called ***RegionServers***.
- This scales read and write capacity by spreading regions across the cluster.
- This is done automatically and is how HBase was designed for horizontal sharding.

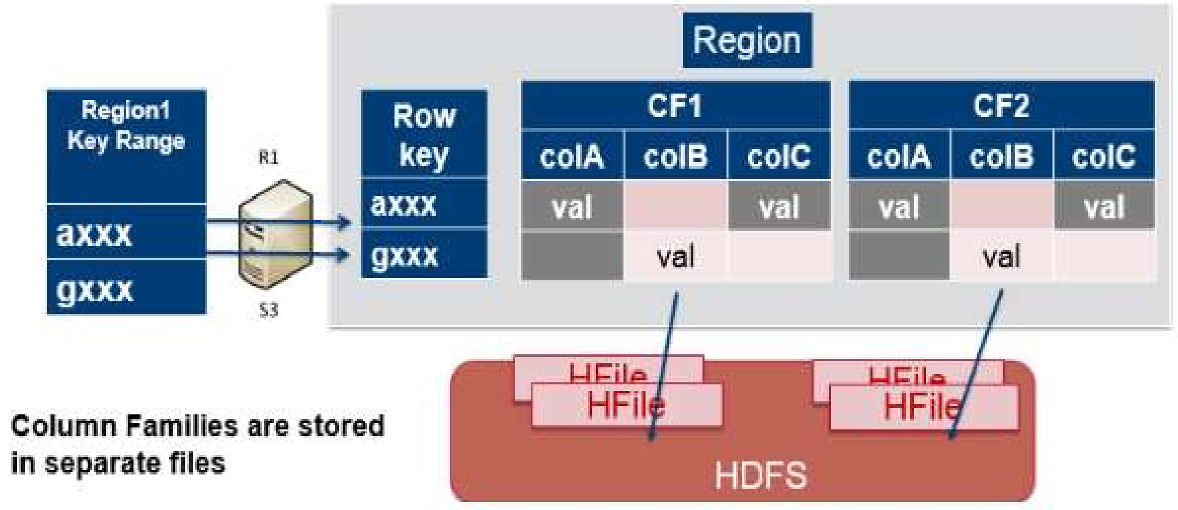
Data Model



Tables are partitioned into **key ranges** (**regions**)
Region= served by nodes (Region Servers)
Regions are spread across cluster

Data Model

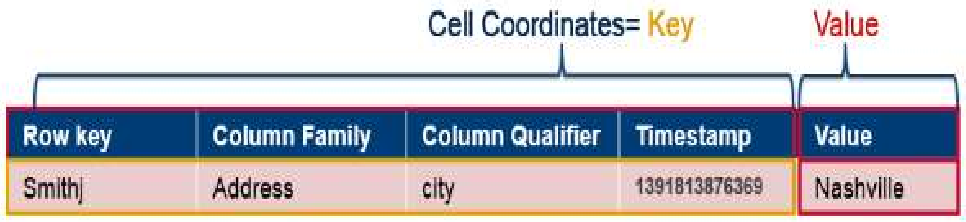
- column families are mapped to storage files.



Data Model

- The data is stored in HBase table cells.
- The entire ***cell***, with the added structural information, is called ***Key Value***.
- The entire cell, the row key, column family name, column name, timestamp, and value are stored for every cell for which you have set a value.

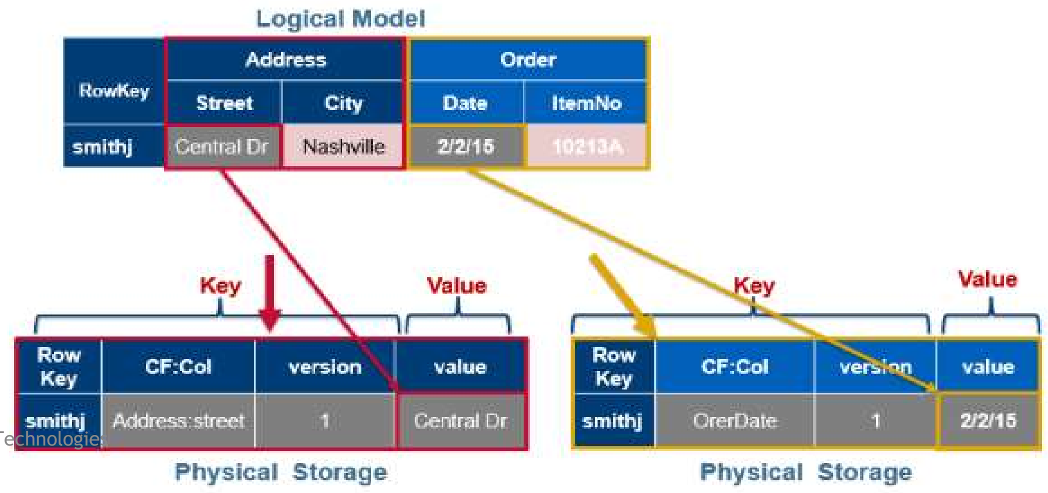
Data Model


Cell Coordinates= Key
Value

Row key	Column Family	Column Qualifier	Timestamp	Value
Smithj	Address	city	1391813876369	Nashville

Data Model

Logically, cells are stored in a table format, but physically, rows are stored as linear sets of cells containing all the key value information inside them.



RowKey	Address		Order	
	Street	City	Date	ItemNo
smithj	Central Dr	Nashville	2/2/15	10213A

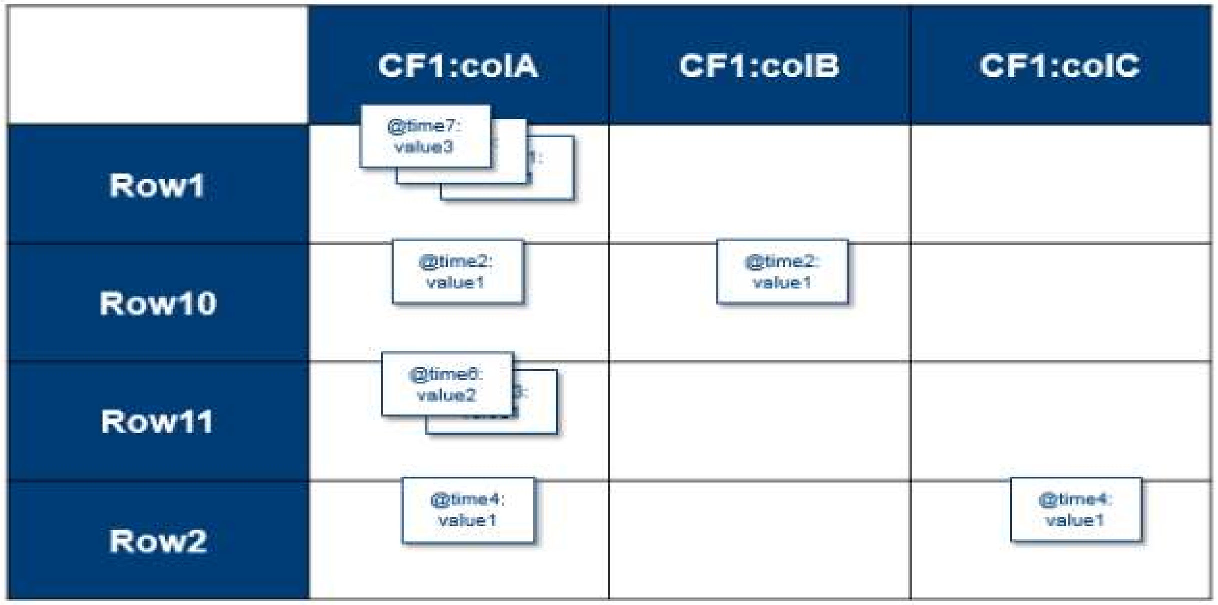
Row Key	CF:Col	version	value
smithj	Address:street	1	Central Dr

Row Key	CF:Col	version	value
smithj	OrerDate	1	2/2/15

Sparse data with cell versions

- The complete coordinates to a cell's value are: Table:Row:Family:Column:Timestamp ➔ Value.
- HBase tables are sparsely populated. If data doesn’t exist at a column, it’s not stored.
- Table cells are versioned uninterpreted arrays of bytes.
- You can use the timestamp or set up your own versioning system.
- For every coordinate row:family:column, there can be multiple versions of the value.

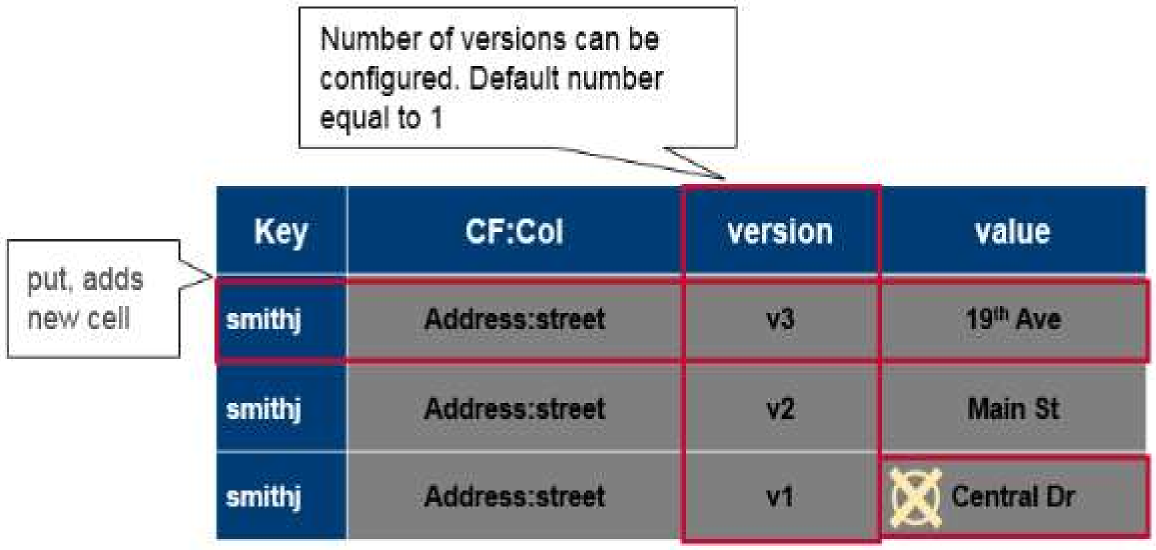
Sparse data with cell versions



Sparse data with cell versions

- A *put* is both an insert (create) and an update, and each one gets its own version.
- *Delete* gets a tombstone marker. The tombstone marker prevents the data being returned in queries.
- *Get* requests return specific version(s) based on parameters. If you do not specify any parameters, the most recent version is returned.
- You can configure how many versions you want to keep and this is done per column family.

Sparse data with cell versions



Number of versions can be configured. Default number equal to 1

put, adds new cell

Key	CF:Col	version	value
smithj	Address:street	v3	19th Ave
smithj	Address:street	v2	Main St
smithj	Address:street	v1	Central Dr

Knowledge Check

Questions(TODO)

1. Name the three V’s of Big Data.
2. Name four of the six types of data commonly found in Hadoop.
3. Why is HDP comprised of so many different frameworks?
4. What two frameworks make up the core of Hadoop?
5. What is the base command-line interface command for manipulating files and directories in HDFS?
6. YARN allocates resources to applications via ________________________.

HBase Architectural Components

Objectives

After completing this lesson, students should be able to get:

- Regions
- HBase Hmaster
- ZooKeeper: The Coordinator
- HBase First Read or Write
- HBase Meta Table
- Region Server Components
- HBase MemStore
- HBase Region Flush
- HBase Hfile
- HBase Minor and Major Compaction
- Region Split

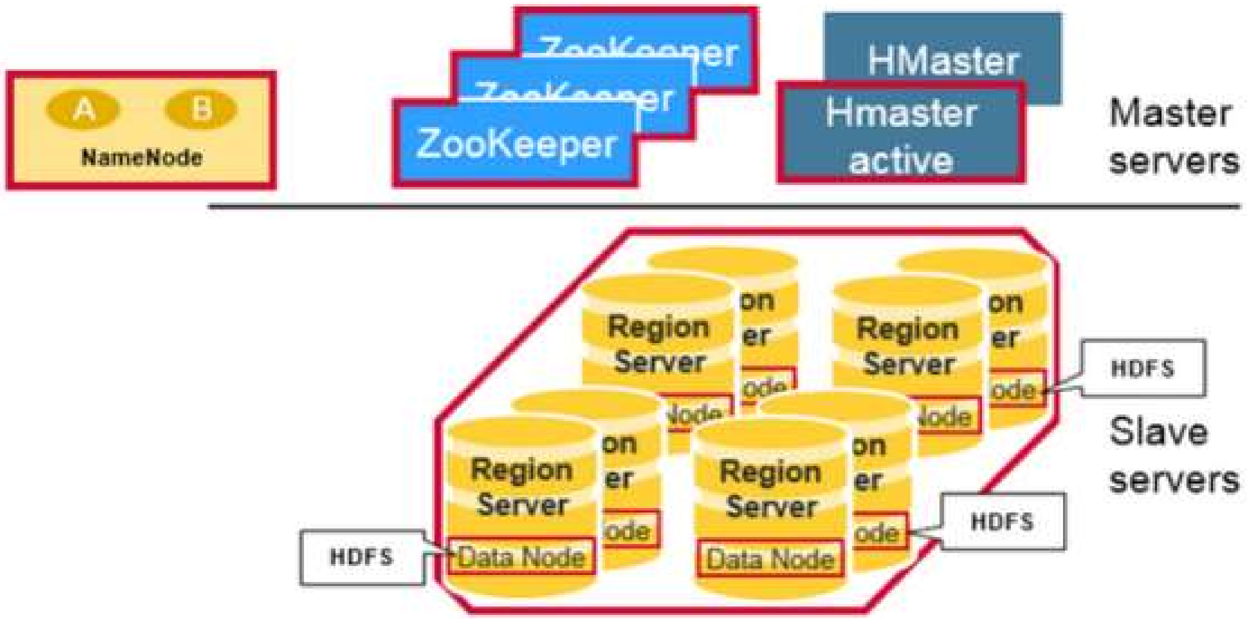
HBase Architectural Components

- Physically, HBase is composed of three types of servers in a master slave type of architecture.
  - ***Region servers*** serve data for reads and writes. When accessing data, clients communicate with HBase Region Servers directly.
  - Region assignment, DDL (create, delete tables) operations are handled by the HBase ***Master*** process.
  - ***Zookeeper***, which is part of HDFS, maintains a live cluster state.

HBase Architectural Components

- The Hadoop ***DataNode*** stores the data that the ***Region Server*** is managing.
- All HBase data is stored in HDFS files. Region Servers are collocated with the HDFS DataNodes, which enable data locality (putting the data close to where it is needed) for the data served by the RegionServers.
- HBase data is local when it is written, but when a region is moved, it is not local until compaction.
- The ***NameNode*** maintains metadata information for all the physical data blocks that comprise the files.

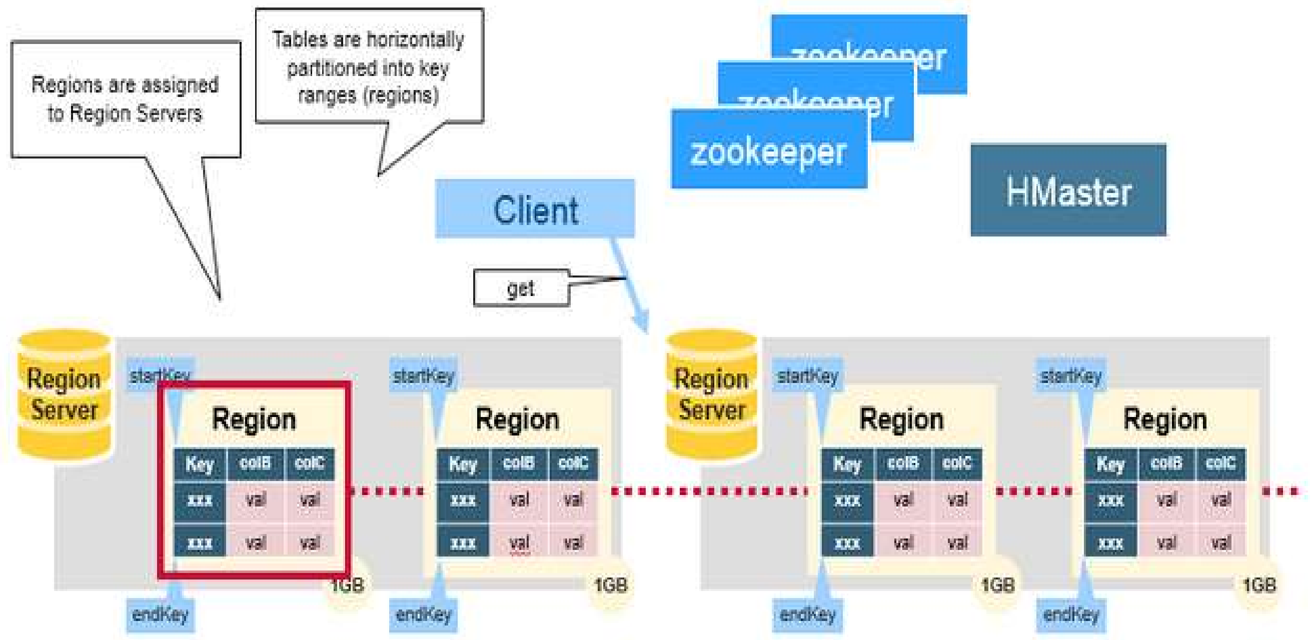
HBase Architectural Components



Regions

- HBase Tables are divided horizontally by row key range into ***Regions***.
- A region contains all rows in the table between the region’s start key and end key.
- Regions are assigned to the nodes in the cluster, called ***Region Servers***, and these serve data for reads and writes.
- A region server can serve about 1,000 regions.

Regions


Regions are assigned to Region Servers
Tables are horizontally partitioned into key ranges (regions)
zookeeper
zookeeper
zookeeper
HMaster
Client
get
Region Server
startKey
Region
Key
colB
colC
xxx
val
endKey
1GB
Region Server

HBase HMaster

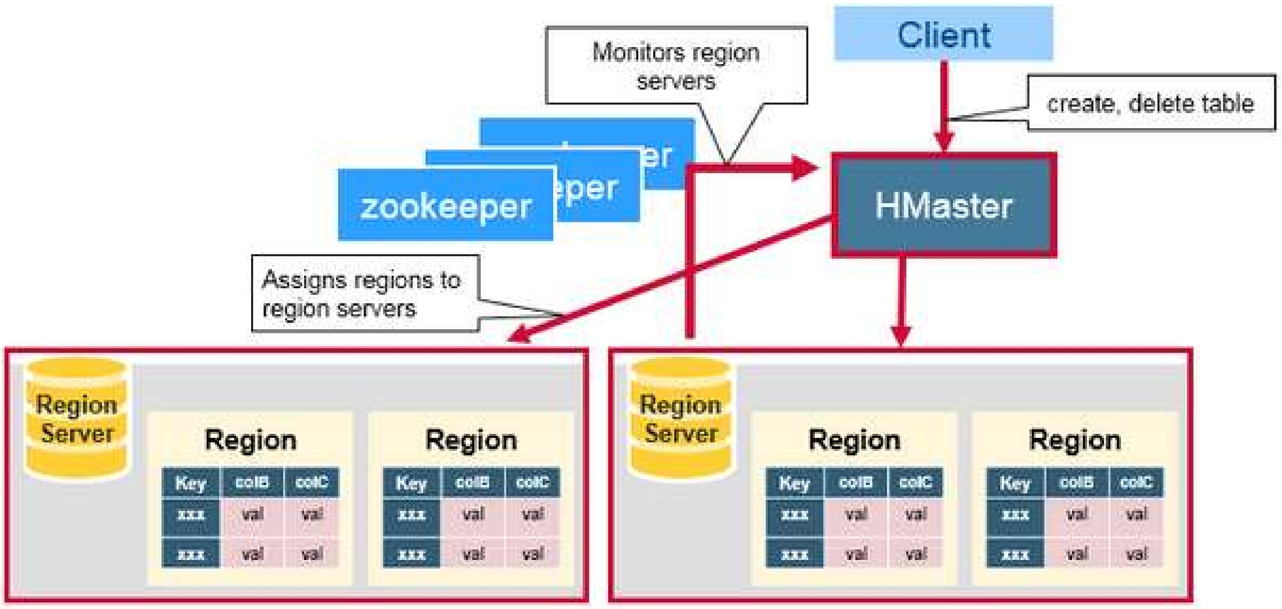
- A master is responsible for:
  - Coordinating the region servers

    - Assigning regions on startup , re-assigning regions for recovery or load balancing

    - Monitoring all RegionServer instances in the cluster (listens for notifications from zookeeper)
  - Admin functions

    - Interface for creating, deleting, updating tables

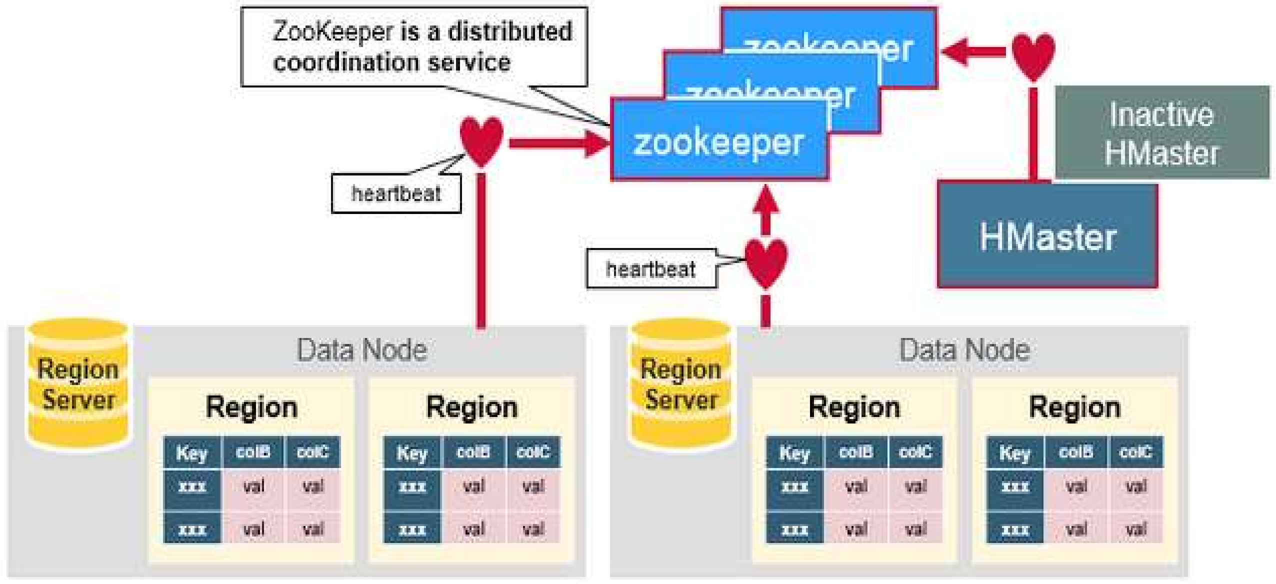
HBase HMaster



ZooKeeper: The Coordinator

- HBase uses ZooKeeper as a distributed coordination service to maintain server state in the cluster.
- Zookeeper maintains which servers are alive and available, and provides server failure notification.

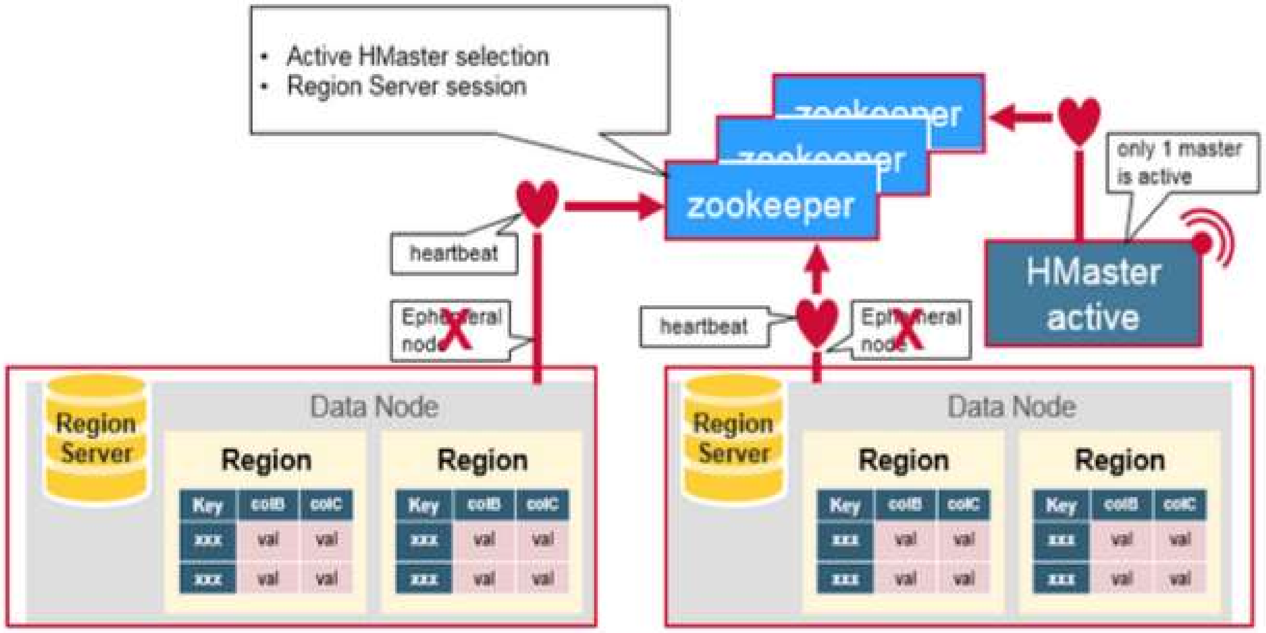
ZooKeeper: The Coordinator



How the Components Work Together

- Zookeeper is used to coordinate shared state information for members of distributed systems.
- Region servers and the active HMaster connect with a session to ZooKeeper.
- The ZooKeeper maintains ephemeral nodes for active sessions via heartbeats.

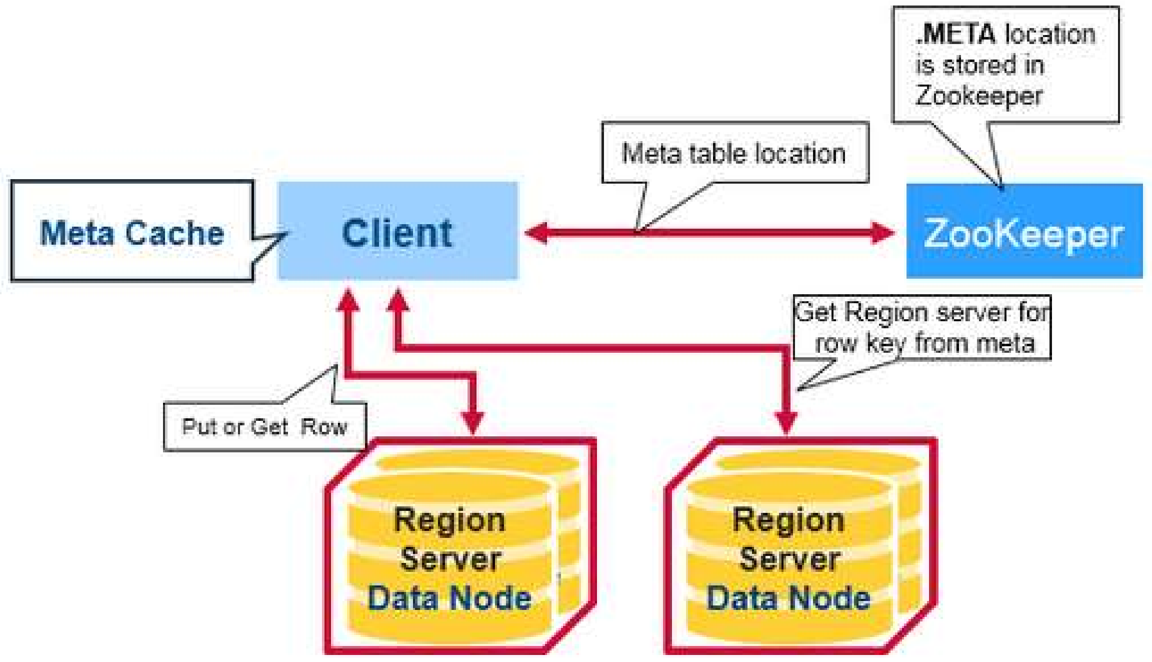
How the Components Work Together



HBase First Read or Write

- There is a special HBase ***Catalog*** table called the ***META*** table, which holds the location of the regions in the cluster.
- ZooKeeper stores the location of the META table.

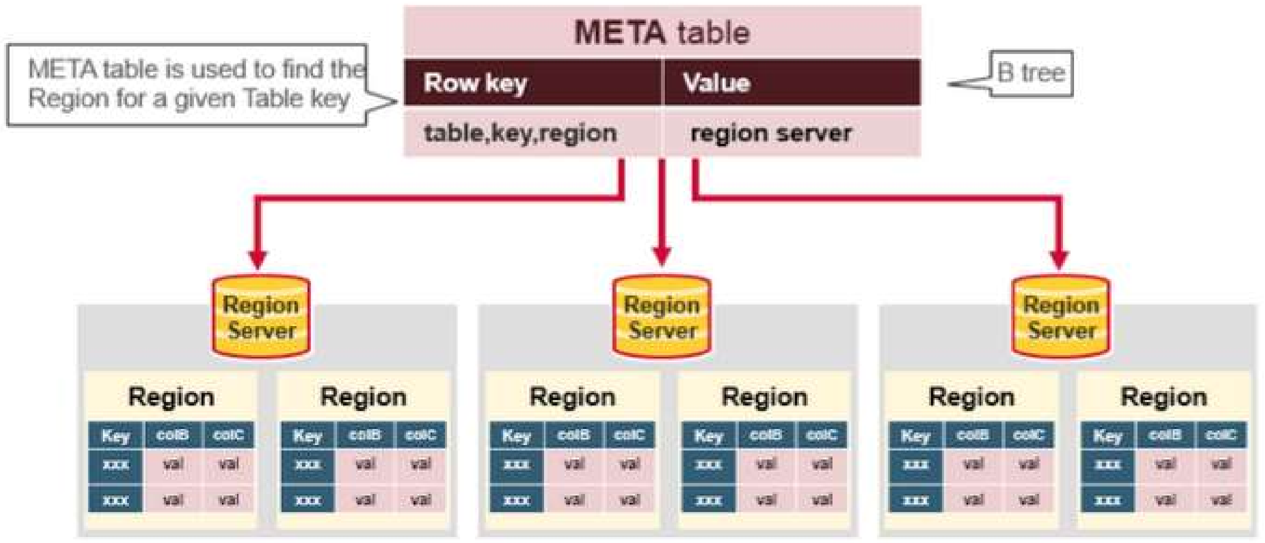
HBase First Read or Write



HBase Meta Table

- The META table is an HBase table that keeps a list of all regions in the system.
- The .META. table is like a b tree.
- The .META. table structure is as follows:
  - Key: region start key,region id
  - Values: RegionServer

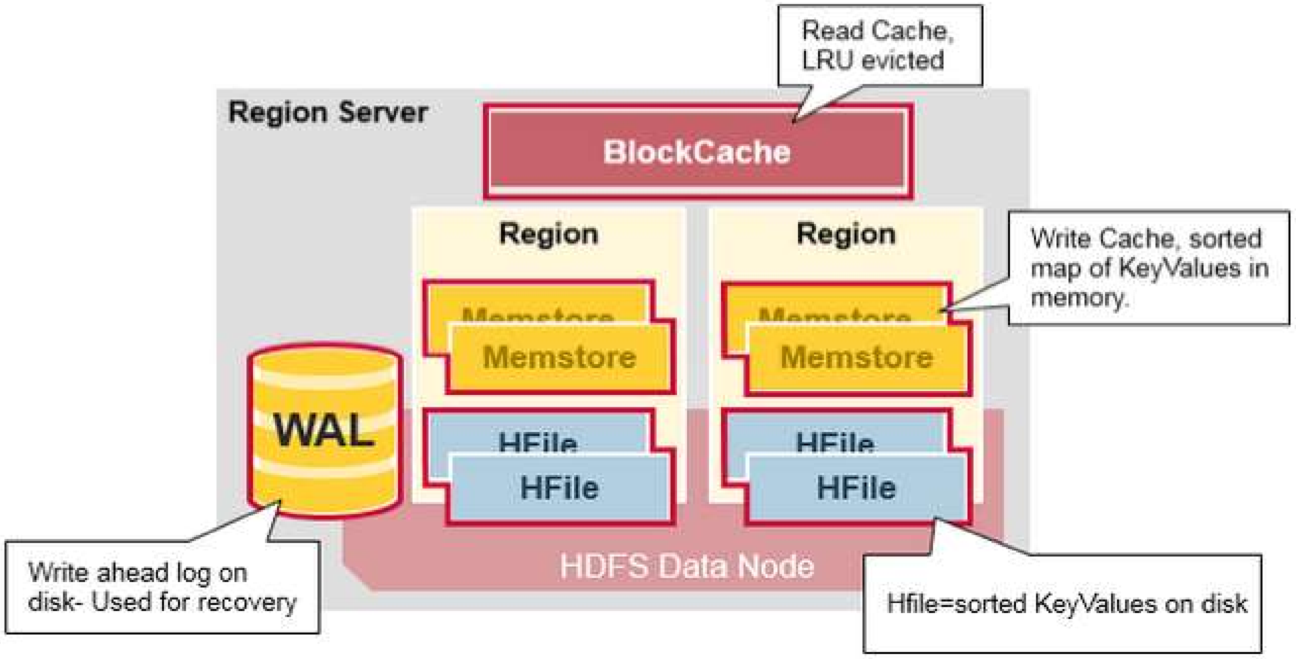
HBase Meta Table



Region Server Components

- A Region Server runs on an HDFS data node and has the following components:
  - ***WAL:*** Write Ahead Log is a file on the distributed file system. The WAL is used to store new data that hasn't yet been persisted to permanent storage; it is used for recovery in the case of failure.
  - ***BlockCache:*** is the read cache. It stores frequently read data in memory. Least Recently Used data is evicted when full.
  - ***MemStore:*** is the write cache. It stores new data which has not yet been written to disk. It is sorted before writing to disk. ***There is one MemStore per column family per region***.
  - ***Hfiles*** store the rows as sorted KeyValues on disk.

Region Server Components

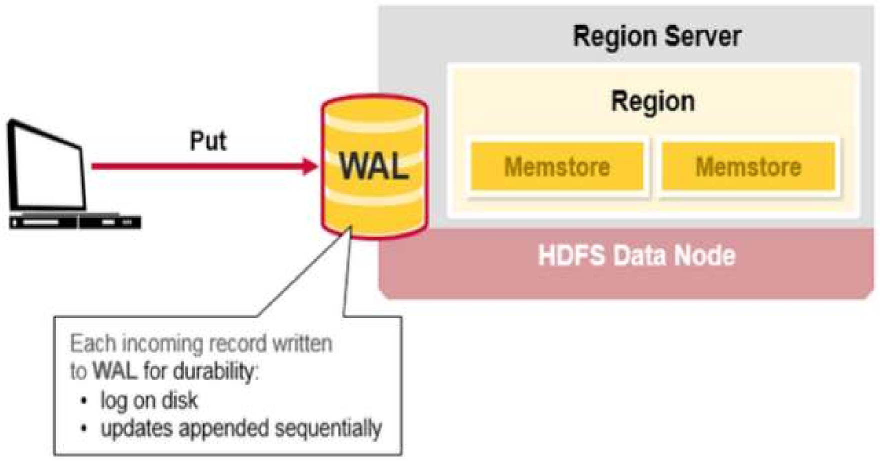


HBase Write Steps (1)

- When the client issues a ***Put*** request, the first step is to write the data to the write-ahead log, the WAL:

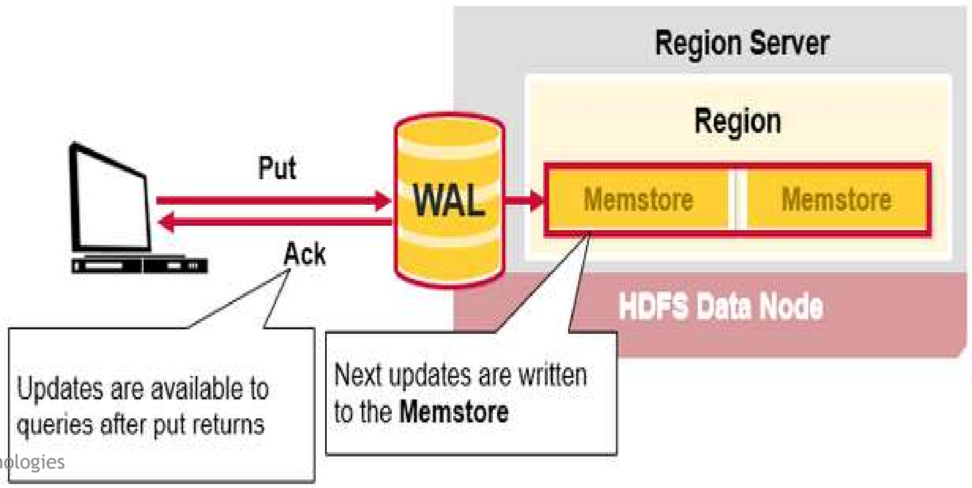
  - Edits are appended to the end of the WAL file that is stored on disk.

  - The WAL is used to recover not-yet-persisted data in case a server crashes.


Region Server
Region
Memstore
Memstore
HDFS Data Node
Put
WAL
Each incoming record written to WAL for durability:
• log on disk
• updates appended sequentially

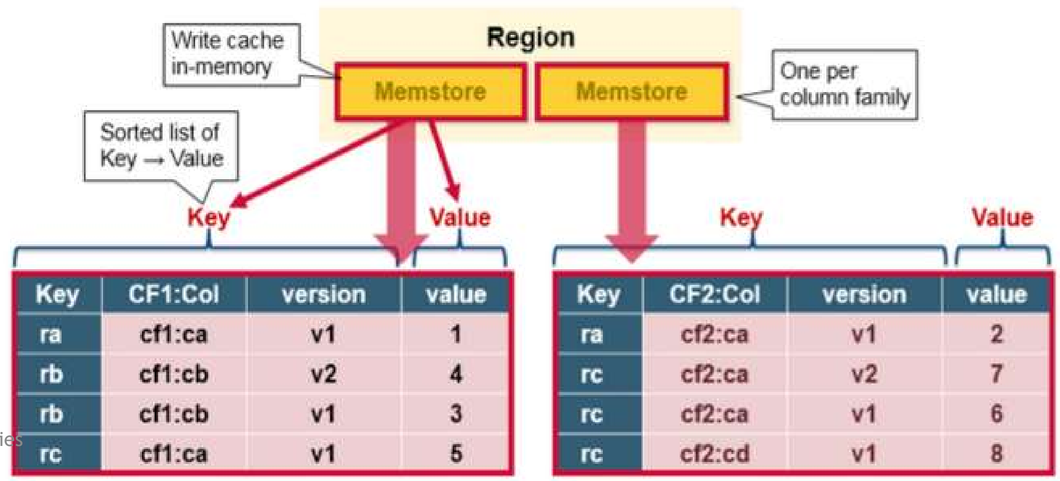
HBase Write Steps (2)

- Once the data is written to the WAL, it is placed in the MemStore.
- Then, the put request acknowledgement returns to the client.


Region Server
Region
Put
WAL
Memstore
Memstore
Ack
HDFS Data Node
Updates are available to queries after put returns
Next updates are written to the **Memstore**

HBase MemStore

- The MemStore stores updates in memory as sorted KeyValues, the same as it would be stored in an HFile.
- There is ***one MemStore per column family***. The updates are sorted per column family.



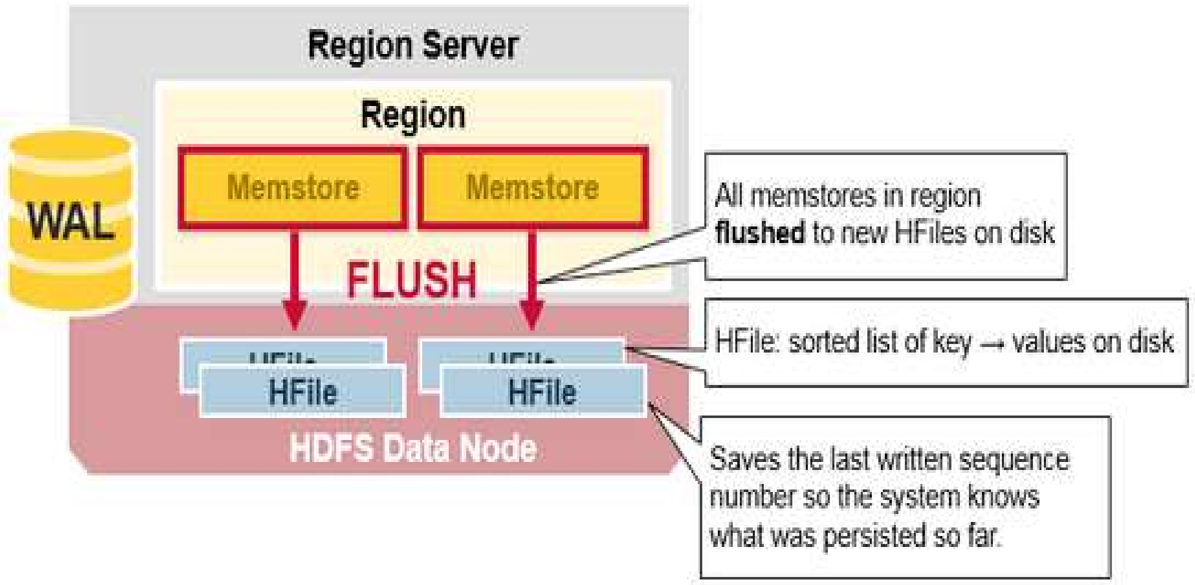
Key	CF1:Col	version	value
ra	cf1:ca	v1	1
rb	cf1:cb	v2	4
rb	cf1:cb	v1	3
rc	cf1:ca	v1	5

Key	CF2:Col	version	value
ra	cf2:ca	v1	2
rc	cf2:ca	v2	7
rc	cf2:ca	v1	6
rc	cf2:cd	v1	8

HBase Region Flush

- When the MemStore accumulates enough data, the entire sorted set is written to a new HFile in HDFS.
- HBase uses multiple HFiles per column family, which contain the actual cells, or KeyValue instances.
- These files are created over time as KeyValue edits sorted in the MemStores are flushed as files to disk.
- Note that this is one reason why there is a limit to the number of column families in HBase.

HBase Region Flush


Region Server
Region
WAL
Memstore
Memstore
FLUSH
HFile
HFile
HFile
HFile
HDFS Data Node
All memstores in region **flushed** to new HFiles on disk
HFile: sorted list of key → values on disk
Saves the last written sequence number so the system knows what was persisted so far.