Hadoop HBase

(A NoSql Data storage on Hadoop)

NoSQL Data Store

- The data store in a Hadoop implementation is usually referred to as a NoSQL store.
- Although this is not technically accurate because some implementations do support a structured query language (such as SQL).
- In fact, some people prefer to use the term "Not Only SQL" for just this reason.
- The particular suitability of a given NoSQL database depends on the problem it must solve.
- There are more than 100+ NoSQL data store implementations available at the time of writing.

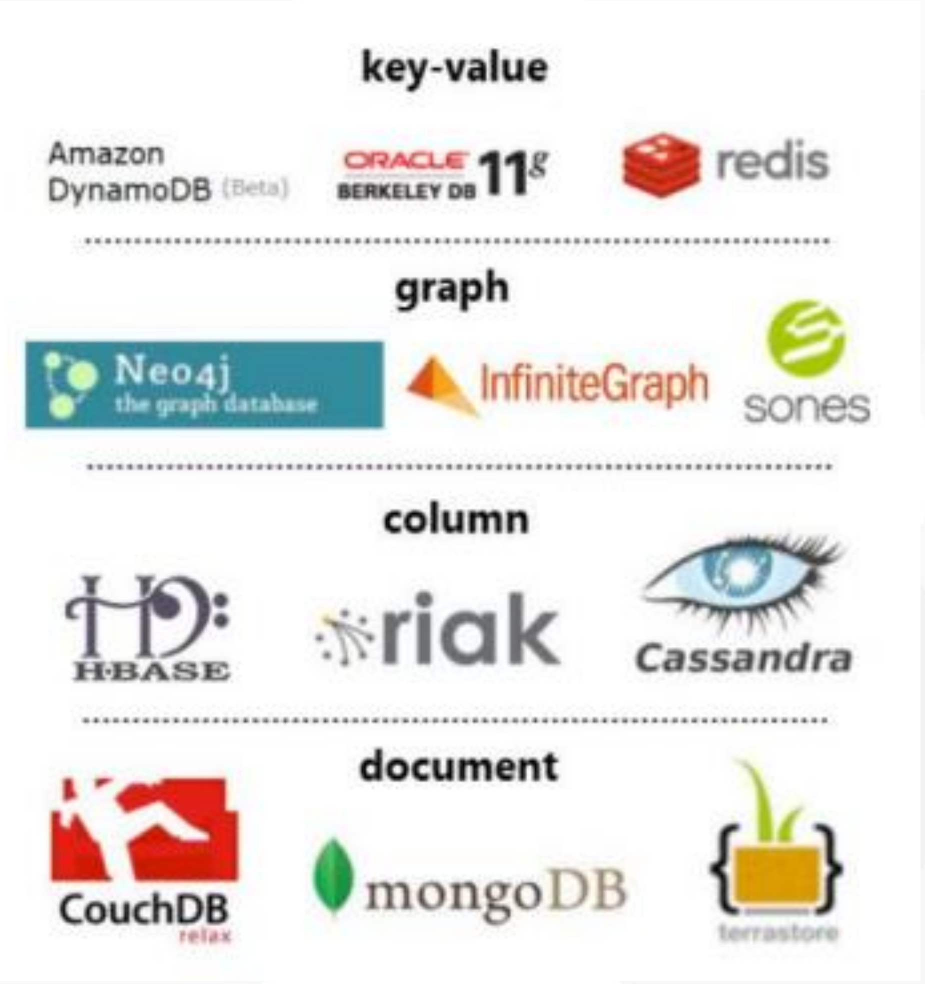
NoSQL Datastore

- They can be divided into the following basic categories:

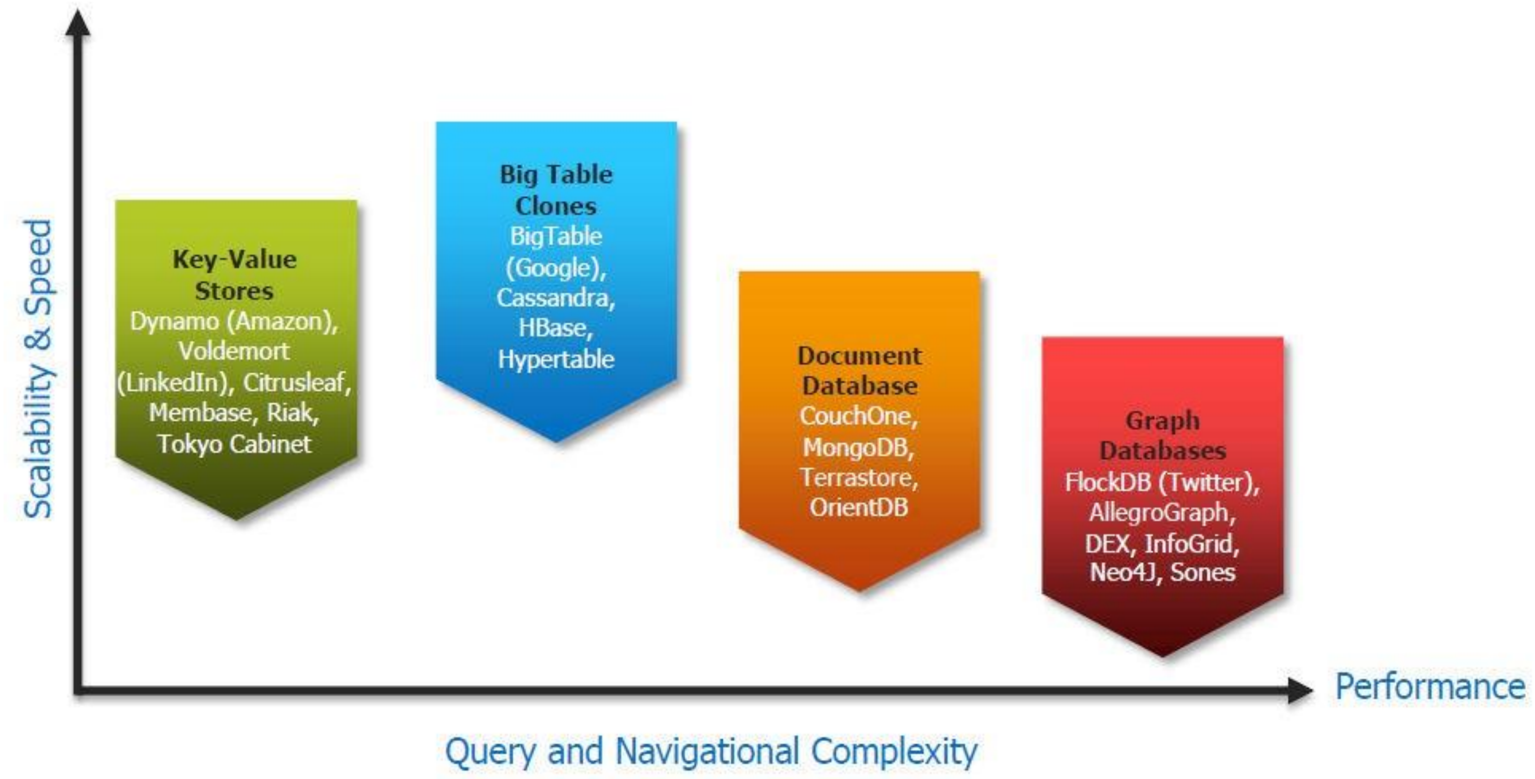
- **Key/Value Stores:** These are data stores that hold data as a series of key/value pairs. There is no fixed schema for the data, and so these types of data store are ideal for unstructured data.
- **Document Stores:** These are data stores optimized to hold structured, semi-structured, and unstructured data items such as JSON objects, XML documents, and binary data. They are usually indexed stores.
- **Column Data Stores:** These are data stores use a schema, but the schema can contain families of columns rather than just single columns. They are ideally suited to storing semi-structured data, where some columns can be predefined but others are capable of storing differing elements of unstructured data.
- **Graph Data Stores:** These are data stores that hold the relationships between objects. They are less common than the other types of data store, many still being experimental, and they tend to have specialist uses.

NoSQL Datastore

- NoSQL storage is typically much cheaper than relational storage, and usually supports a write once capability that allows only for data to be appended.
- To update data in these stores you must drop and recreate the relevant file, or maintain delta files and implement mechanisms to conflate the data.
- This limitation maximizes throughput; storage implementations are usually measured by throughput rather than capacity because this is usually the most significant factor for both storage and query efficiency.



NoSQL Landscape


Scalability & Speed
Key-Value Stores
Dynamo (Amazon), Voldemort (LinkedIn), Citrusleaf, Membase, Riak, Tokyo Cabinet
Big Table Clones
BigTable (Google), Cassandra, HBase, Hypertable
Document Database
CouchOne, MongoDB, Terrastore, OrientDB
Graph Databases
FlockDB (Twitter), AllegroGraph, DEX, InfoGrid, Neo4J, Sones
Performance
Query and Navigational Complexity

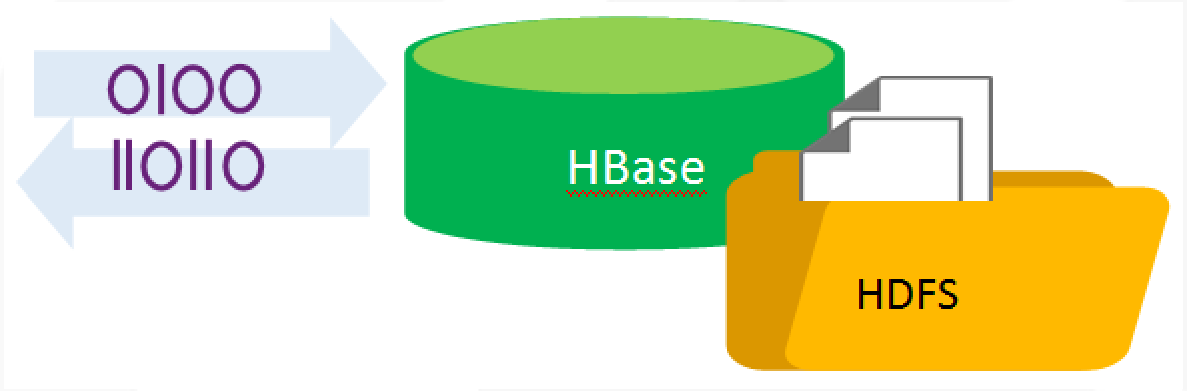
Apache HBASE - Overview

- No SQL: Hbase is a type of “NoSQL” database. NoSql is a general term meaning that the database is not an RDBMS which support SQL as its primary access language
- Hbase is an open source, non relational, distributed, sorted map, data store modeled after Google’s BigTable
- Hbase is really more a Data store than Data Base. Hbase stores data in storefiles on HDFS
- The HBase platform is a column oriented system which runs on top of HDFS (Hadoop Distributed Filesystem), provides real-time read/write access to those large datasets.
- With YARN as the architectural center of Apache Hadoop, multiple data access engines such as Apache HBase interact with data stored in the cluster.

http://research.google.com/archive/bigtable.html

HBase vs. HDFS

- HDFS is a distributed file system that is well suited for the storage of large files. Its documentation states that it is not, however, a general purpose file system, and does not provide fast individual record lookups in files.
- HBase, on the other hand, is built on top of HDFS and provides fast record lookups (and updates) for large tables. This can sometimes be a point of conceptual confusion. HBase internally puts your data in indexed "StoreFiles" that exist on HDFS for high-speed lookups.



Comparison

HDFS	HBase
HDFS is a distributed file system suitable for storing large files.	HBase is a database built on top of the HDFS.
HDFS does not support fast individual record lookups.	HBase provides fast lookups for larger tables.
It provides high latency batch processing	It provides low latency access to single rows from billions of records (Random access).
It provides only sequential access of data.	HBase internally uses Hash tables and provides random access, and it stores the data in indexed HDFS files for faster lookups.

Hbase “NoSQL” and RDBMS

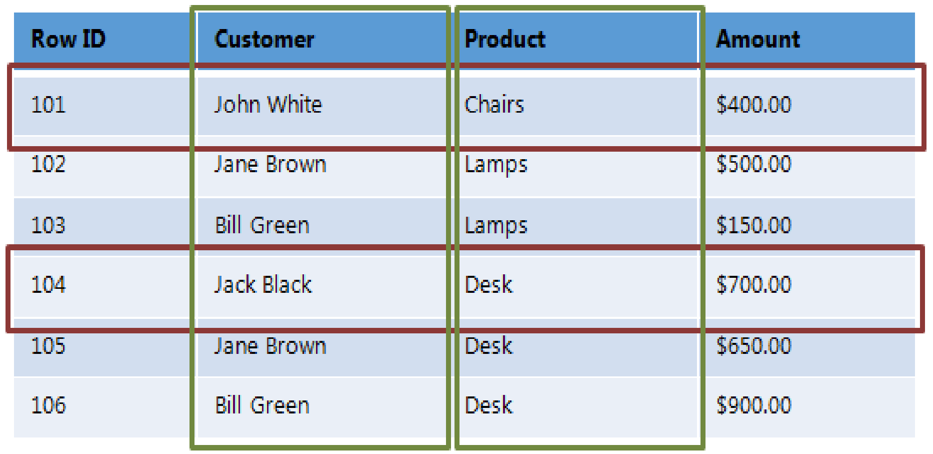
- RDBMS Limitations
  - data sets are growing in PetaBytes
  - RDBMS don’t scale inherently(Scale up and Scale out)
  - Hard to shard / partition
  - Both read/write throughput not possible (Transactional and Analytical)
  - Specialized hardware is very expensive(Oracle grid/cluster)
- NoSQL
  - Strictly consistent reads and writes
  - Automatic failover support between RegionServers
  - Convenient base classes for backing Hadoop MapReduce Jobs with Apache Hbase tables
  - Automatic sharding* and load balancing of tables
  - Near real time lookup

*A **database shard** is a horizontal partition of data in a database or search engine. Each individual partition is referred to as a **shard** or **database shard**. Each shard is held on a separate database server instance, to spread load.

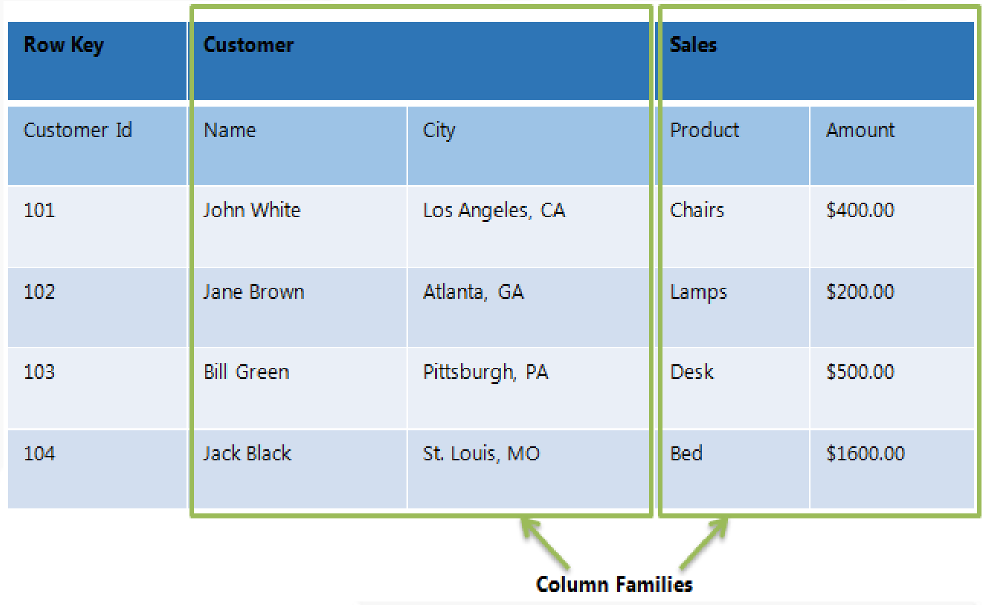
Column Oriented and Row Oriented

Column-oriented databases are those that store data tables as sections of columns of data, rather than as rows of data. Shortly, they will have column families.

Row-Oriented Database	Column-Oriented Database
It is suitable for Online Transaction Process (OLTP).	It is suitable for Online Analytical Processing (OLAP).
Such databases are designed for small number of rows and columns.	Column-oriented databases are designed for huge tables.



Row ID	Customer	Product	Amount
101	John White	Chairs	$400.00
102	Jane Brown	Lamps	$500.00
103	Bill Green	Lamps	$150.00
104	Jack Black	Desk	$700.00
105	Jane Brown	Desk	$650.00
106	Bill Green	Desk	$900.00



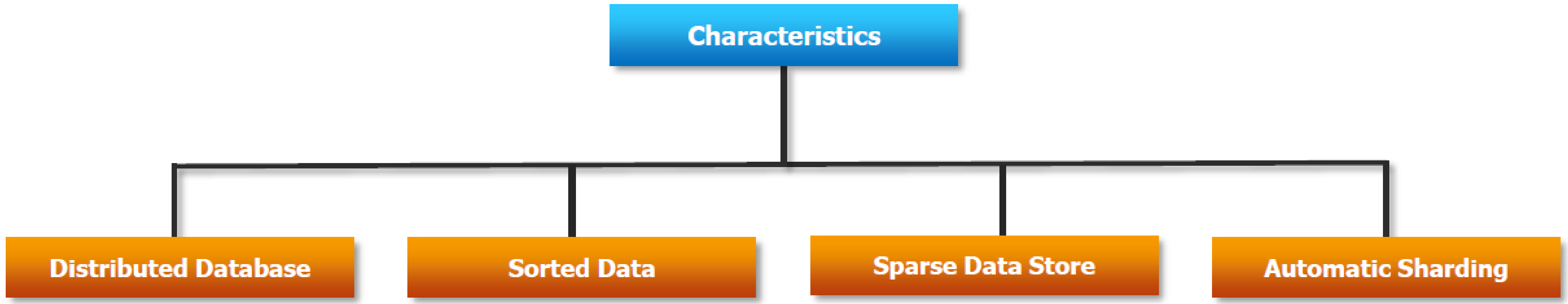
Row Key	Customer		Sales	
Customer Id	Name	City	Product	Amount
101	John White	Los Angeles, CA	Chairs	$400.00
102	Jane Brown	Atlanta, GA	Lamps	$200.00
103	Bill Green	Pittsburgh, PA	Desk	$500.00
104	Jack Black	St. Louis, MO	Bed	$1600.00

Column Families

Comparison

HBase	RDBMS
Column-oriented	Row oriented (mostly)
Flexible schema, add columns on the fly	Fixed schema.
Good with sparse tables,	Not optimized for sparse tables.
Joins using MR –not optimized	Optimized for joins.
Tight integration with MR	Not really...
Horizontal scalability –just add hardware	Hard to shard and scale
Good for semi-structured data as well as structured data	Good for structured data

Hbase characteristics



Characteristics

Distributed

- One key feature of HBase is that the data can be spread over 100s or 1000s of machines and reach billions of cells. HBase manages the load balancing automatically.

Sorted

- These cells are sorted by the key. This is a very important property as it allows for searching .

Sparse

- Not all columns are populated for every row.

Characteristics

Consistent

HBase guarantees the following:

- All changes the with the same rowkey are atomic.
- A reader will always read the last written (and committed) values.

Map

- HBase maintains maps of Keys to Values (key -> value). Each of these mappings is called a "KeyValue" or a "Cell". You can find a value by its key.

Characteristics

Multidimensional

- The key itself has structure. Each key consists of the following parts:
  - row-key, column family, column, and time-stamp.
- So the mapping is multidimensional:
  - (rowkey, column family, column, timestamp) -> value

Sharding

- A database shard is a horizontal partition of data in a database or search engine.
- Each individual partition is referred to as a shard. Each shard is held on a separate database server instance, to spread load.
- Some data within a database remains present in all shards, but some only appears in a single shard.
- Each shard (or server) acts as the single source for this subset of data.

HBASE – Usage & Applications

- ✓Apache HBase is used to have random, real-time read/write access to Big Data.
- ✓It hosts very large tables on top of clusters of commodity hardware.
- ✓Apache HBase is a non-relational database modeled after Google's Bigtable. Bigtable acts up on Google File System, likewise Apache HBase works on top of Hadoop and HDFS.
- ✓It is used whenever there is a need to write heavy applications.
- ✓HBase is used whenever we need to provide fast random access to available data.
- ✓Companies such as Facebook, Twitter, Yahoo, and Adobe use HBase internally.

Hbase data model

- The Data Model in HBase is designed to accommodate semi-structured data that could vary in field size, data type and columns.
- Additionally, the layout of the data model makes it easier to partition the data and distribute it across the cluster.
- The Data Model in HBase is made of different logical components such as Tables, Rows, Column Families, Columns, Cells and Versions.

Hbase data model

- **Tables** – The HBase Tables are more like logical collection of rows stored in separate partitions called Regions.

- **Rows** – A row is one instance of data in a table and is identified by a rowkey. Rowkeys are unique in a Table/

- **Column Families** – Data in a row are grouped together as Column Families.
- Each Column Family has one more Columns and these Columns in a family are stored together in a low level storage file known as HFile.

Hbase data model

- **Columns** – A Column is identified by a Column Qualifier that consists of the Column Family name concatenated with the Column name using a colon – example: columnfamily:columnname.

- **Cell –** A Cell stores data and is essentially a unique combination of rowkey, Column Family and the Column (Column Qualifier). The data stored in a Cell is called its value

- **Version** – The data stored in a cell is versioned and versions of data are identified by the timestamp. The number of versions of data retained in a column family is configurable and this value by default is 3.

- **Timestamp -** A timestamp is written alongside each value and is the identifier for a given version of a value. By default, the timestamp represents the time on the RegionServer when the data was written, but you can specify a different timestamp value when you put data into cell

Hbase data model

Namespace - A Namespace is logical grouping of tables analogous to a database in relation database system. This is ground work for upcoming multi-tenany related features

- Quota Management
- Namespace security Administration
- Region server groups

Predefined Namespaces

- hbase
- default

Catalog Tables - The catalog table hbase:meta exists as an HBase table

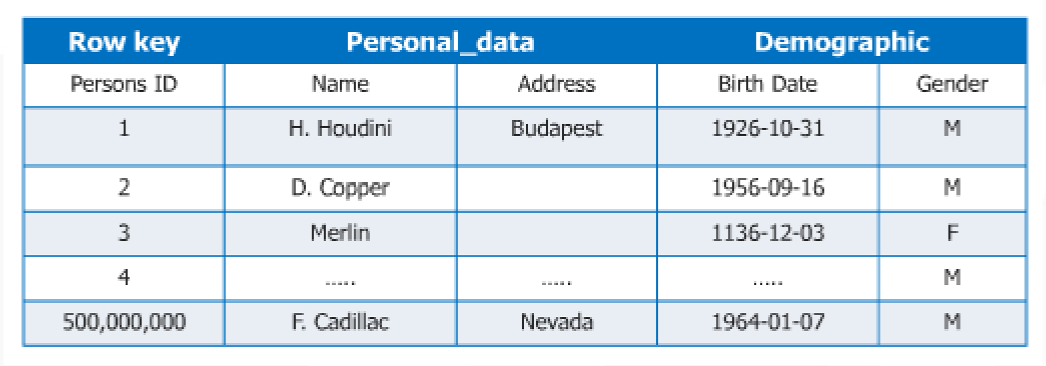
Hbase Data Model

Hbase:meta: The hbase:meta tables(.META.) keeps a list of regions in the system
The location of hbase:meta was previously tracked within the –ROOT- table, and now stored in ZooKeeper

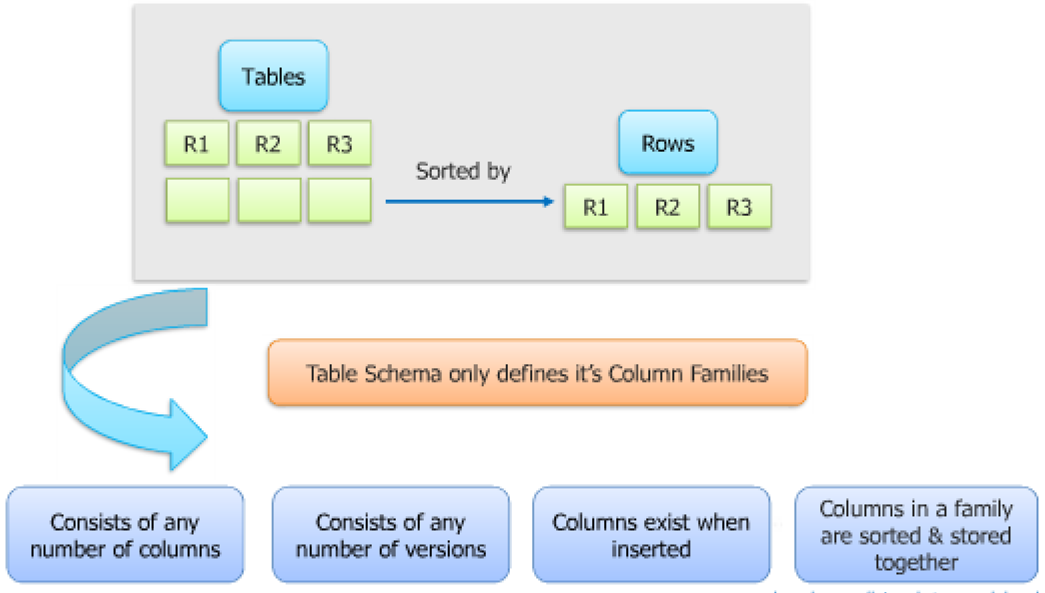
The hbase:meta table structure is as follows:
Key:
Region key of the format ([table],[region start key],[region id])
Values:
- info:regioninfo (serialized HRegionInfo instance for this region)
- info:server (server:port of the RegionServer containing this region)
- info:serverstartcode (start-time of the RegionServer process containing this region)

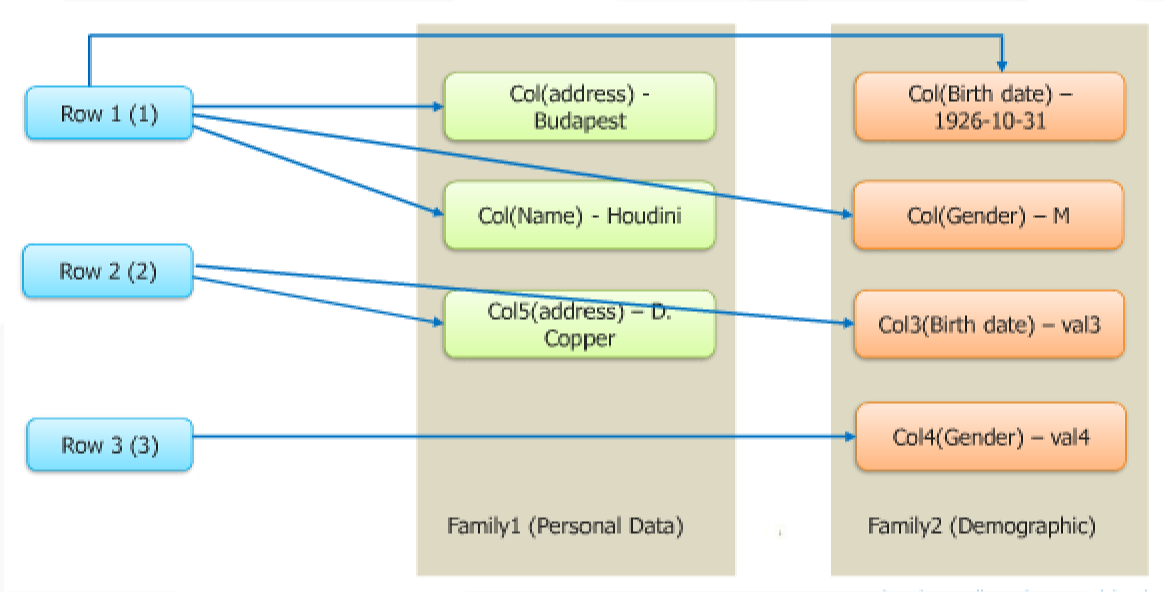
When a table is in the process of splitting, two other columns will be created, called info:splitA and info:splitB. These columns represent the two daughter regions. The values for these columns are also serialized HRegionInfo instances. After the region has been split, eventually this row will be deleted.

Data Model

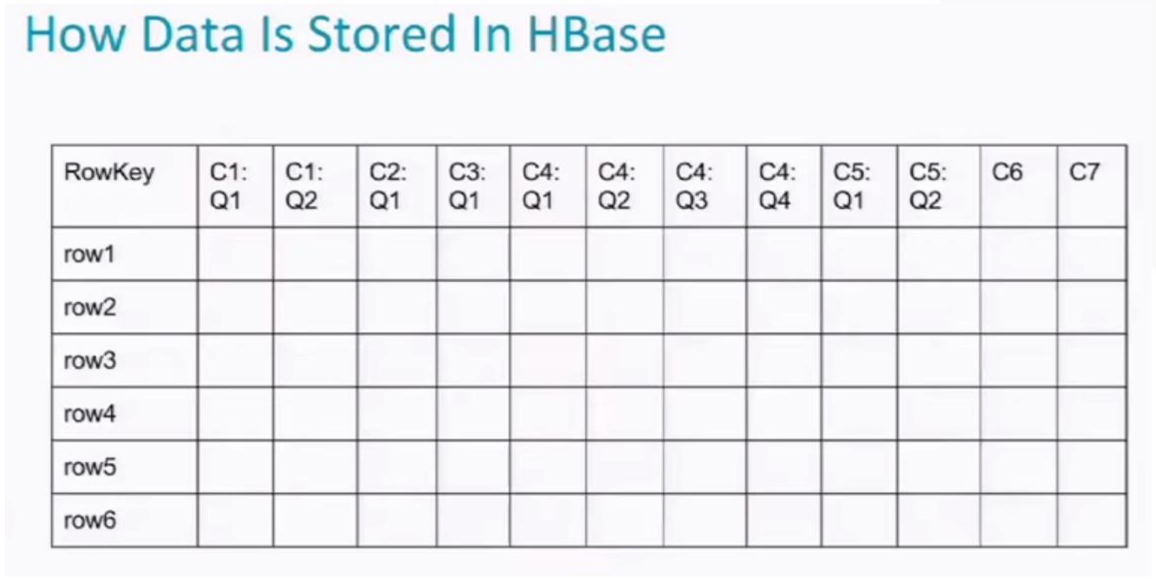


Row key	Personal_data		Demographic	
Persons ID	Name	Address	Birth Date	Gender
1	H. Houdini	Budapest	1926-10-31	M
2	D. Copper		1956-09-16	M
3	Merlin		1136-12-03	F
4	.....	.....	.....	M
500,000,000	F. Cadillac	Nevada	1964-01-07	M


Tables
R1
R2
R3
Sorted by
Rows
R1
R2
R3
Table Schema only defines it's Column Families
Consists of any number of columns
Consists of any number of versions
Columns exist when inserted
Columns in a family are sorted & stored together

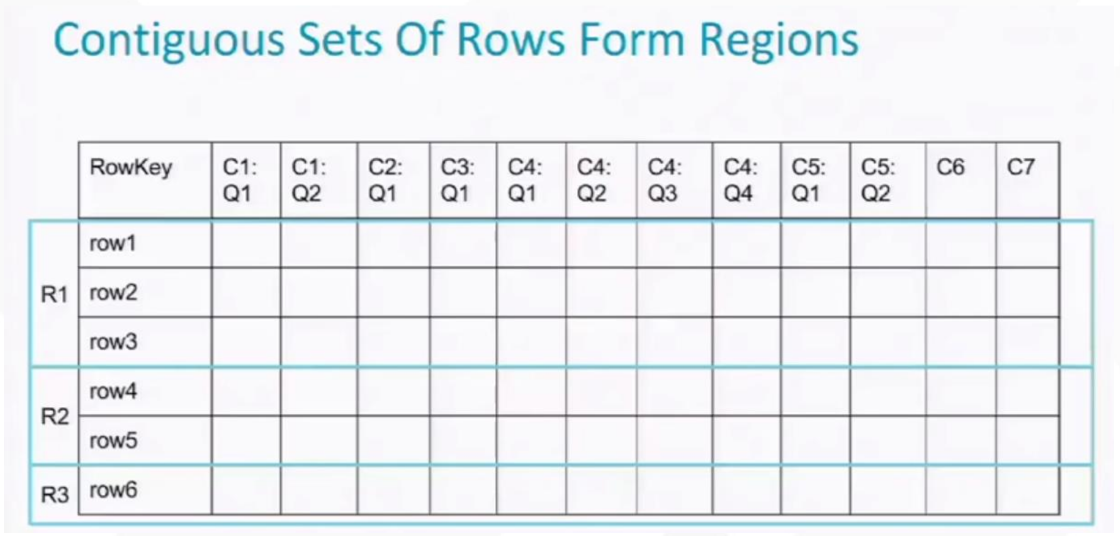


How does data get stored in HBase



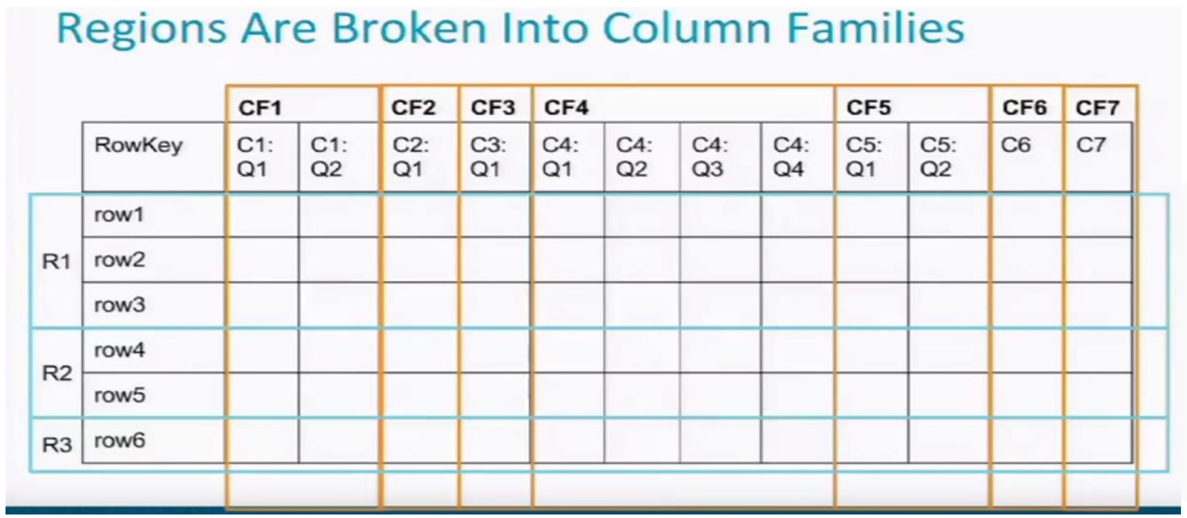
How Data Is Stored In HBase

RowKey	C1: Q1	C1: Q2	C2: Q1	C3: Q1	C4: Q1	C4: Q2	C4: Q3	C4: Q4	C5: Q1	C5: Q2	C6	C7
row1												
row2												
row3												
row4												
row5												
row6												



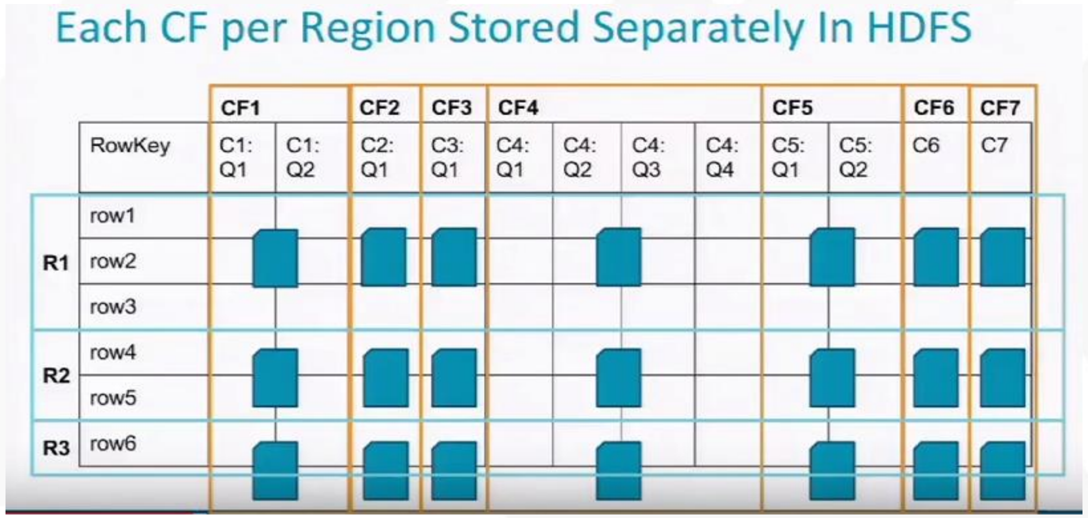
Contiguous Sets Of Rows Form Regions

	RowKey	C1: Q1	C1: Q2	C2: Q1	C3: Q1	C4: Q1	C4: Q2	C4: Q3	C4: Q4	C5: Q1	C5: Q2	C6	C7
R1	row1												
	row2												
	row3												
R2	row4												
	row5												
R3	row6												



Regions Are Broken Into Column Families

		CF1		CF2	CF3	CF4				CF5		CF6	CF7
	RowKey	C1: Q1	C1: Q2	C2: Q1	C3: Q1	C4: Q1	C4: Q2	C4: Q3	C4: Q4	C5: Q1	C5: Q2	C6	C7
R1	row1												
	row2												
	row3												
R2	row4												
	row5												
R3	row6												



Each CF per Region Stored Separately In HDFS

		CF1		CF2	CF3	CF4				CF5		CF6	CF7
	RowKey	C1: Q1	C1: Q2	C2: Q1	C3: Q1	C4: Q1	C4: Q2	C4: Q3	C4: Q4	C5: Q1	C5: Q2	C6	C7
R1	row1												
	row2	■		■	■		■			■		■	■
	row3												
R2	row4	■		■	■		■			■		■	■
	row5												
R3	row6	■		■	■		■			■		■	■

Sharding

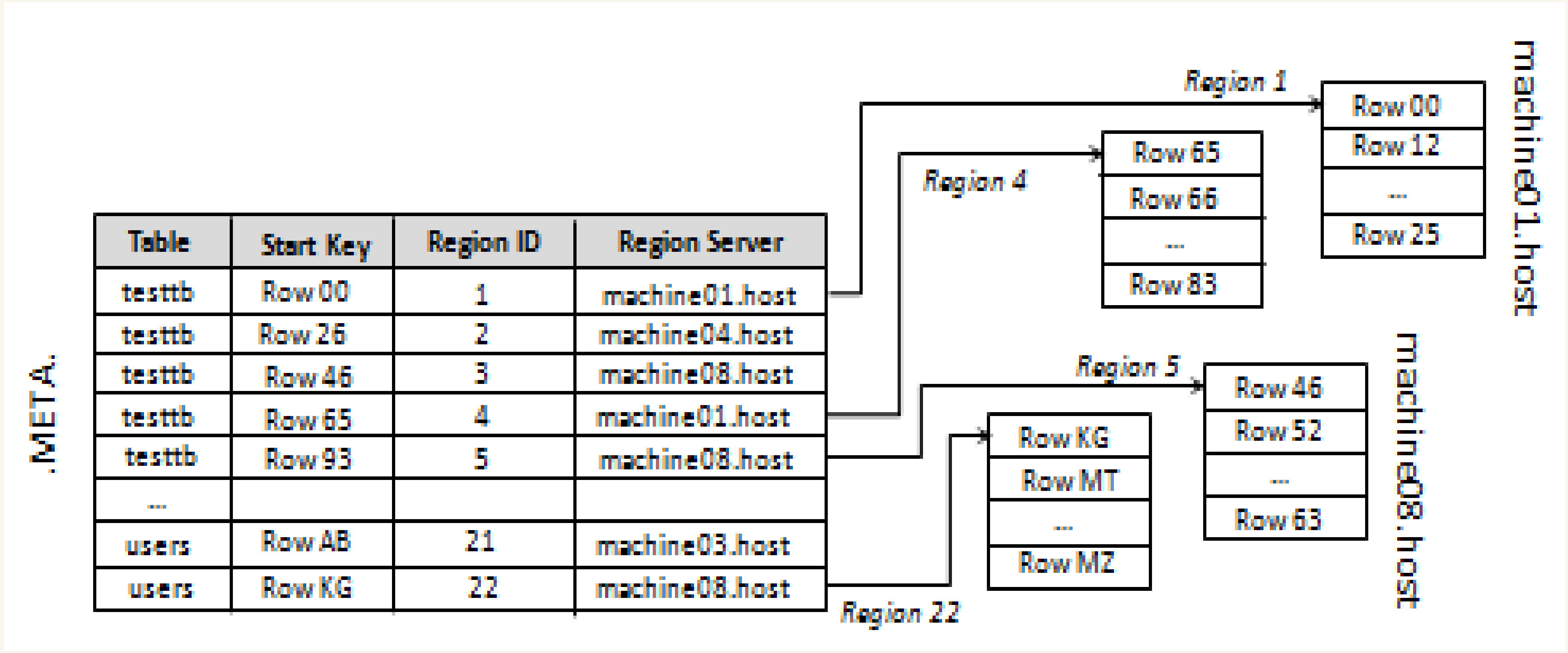
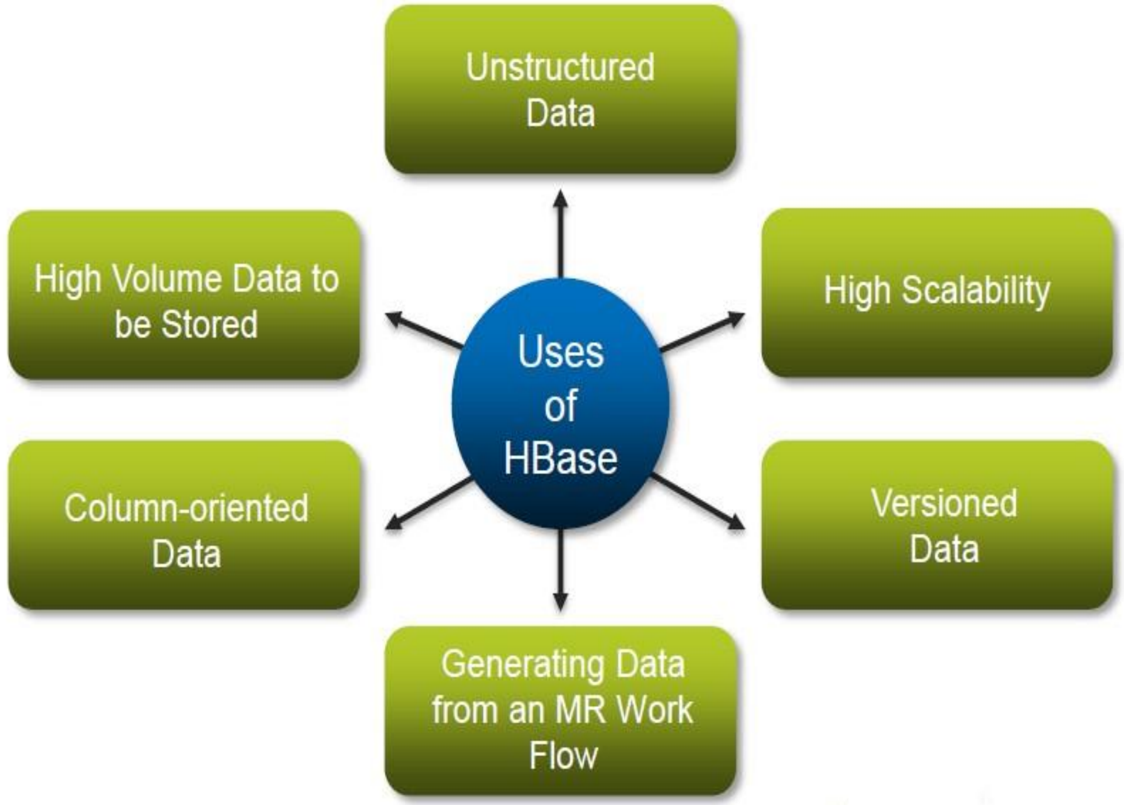


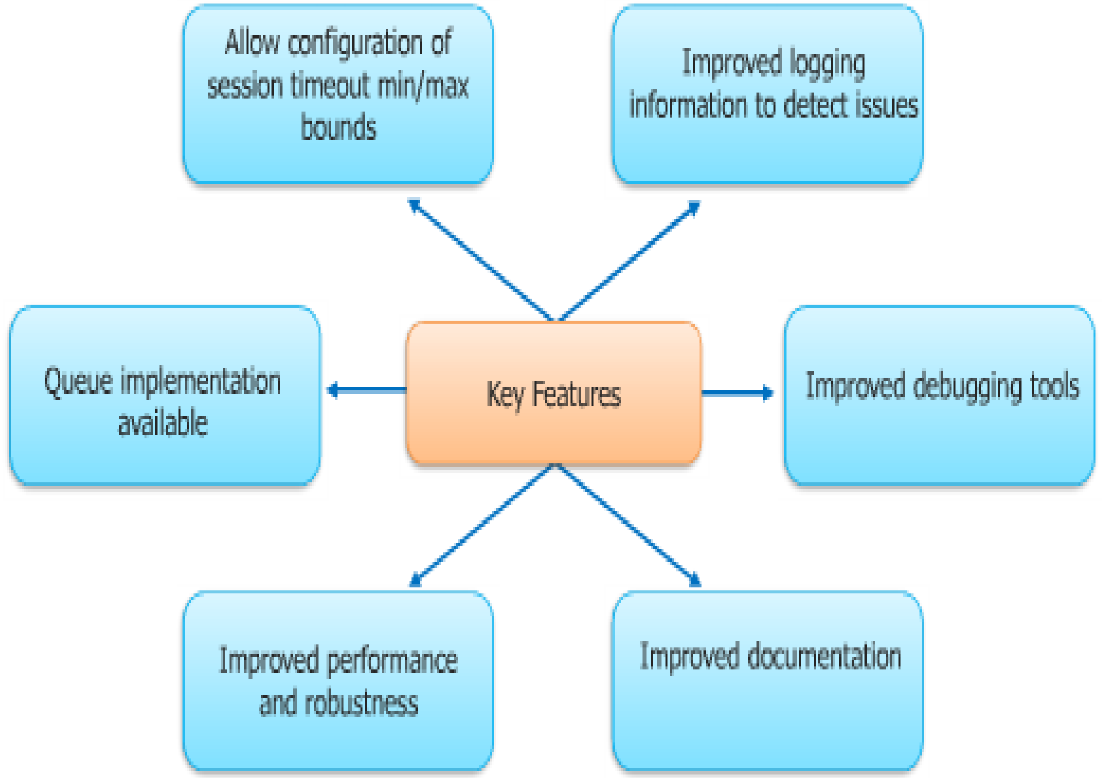
Table	Start Key	Region ID	Region Server
testtb	Row 00	1	machine01.host
testtb	Row 26	2	machine04.host
testtb	Row 46	3	machine08.host
testtb	Row 65	4	machine01.host
testtb	Row 93	5	machine08.host
...			
users	Row AB	21	machine03.host
users	Row KG	22	machine08.host

When not to use Hbase

- When you have only a few thousand/million rows.
- Lacks RDBMS Commands.
- When you have hardware less than 5 Data Nodes when replication factor is 3.

When to use HBase



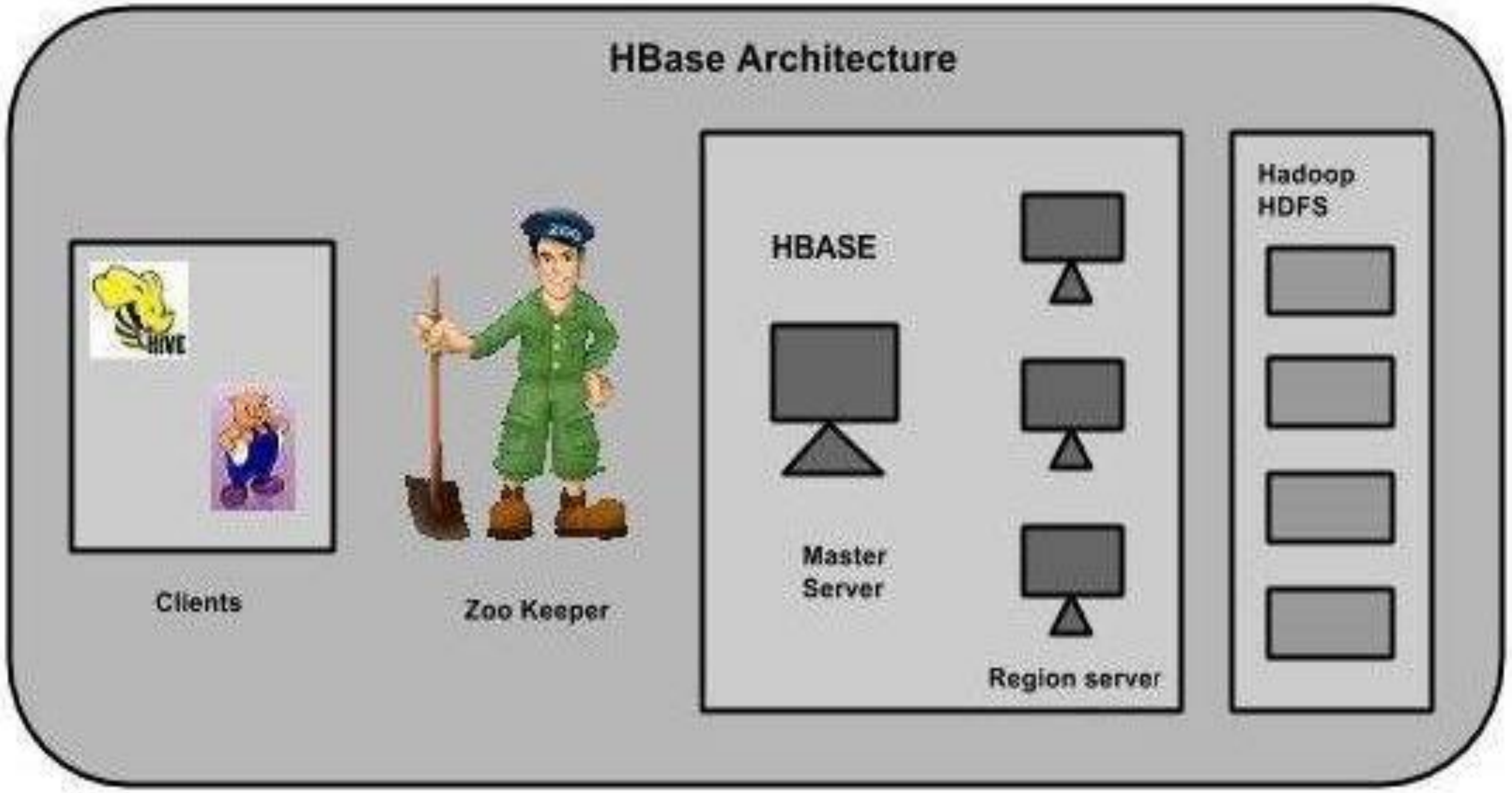

Allow configuration of session timeout min/max bounds
Improved logging information to detect issues
Queue implementation available
Key Features
Improved debugging tools
Improved performance and robustness
Improved documentation

Architecture - Components of HBASE

The three major components of HBase, which takes part in an operation are as follows:

- ✓Hmaster
- ✓Zookeeper
- ✓RegionServer

These three components work together to make HBase a fully functional and efficient database.

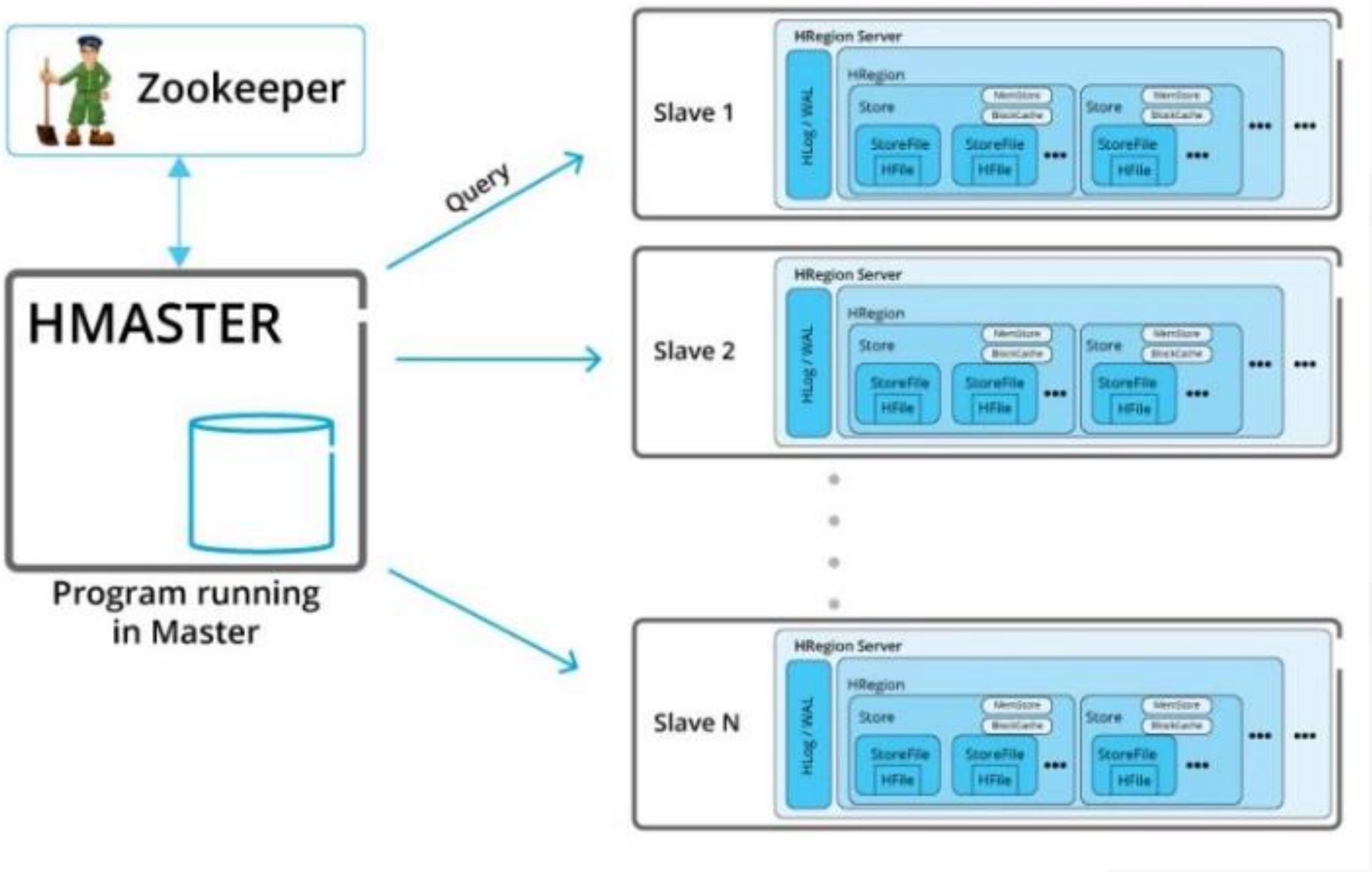


HMaster

The journey of an operation starts with the Client sending a request to the HBase. The following are the steps in the order of its execution.

- ✓ Hmaster gives instruction to Hregion.
- ✓ Hmaster takes information from Zookeeper if any services fail to respond.
- ✓ Hmaster is the one, which responds and takes the request in.
- ✓ Hmaster on startup coordinates & monitors Region Server also assign Region
- ✓ Hmaster creates Region and let know the Hregion Server to store data to the following region. Which ones?
- ✓ Hmaster is responsible for creating a table.
- ✓ Hmaster is responsible for load balancing.
- ✓ HMaster monitors nodes to discover all available region servers, and also monitors these nodes for server failures.
- ✓ HMaster reassigns the regions from the crashed server to active Region servers.
- ✓ There can be multiple Hmaster, just in case of backup as an inactive state.
- ✓ The active HMaster sends heartbeats to Zookeeper, and the inactive HMaster listens for notifications of the active HMaster failure.
- ✓ HMaster splits the WAL belonging to the crashed region server into separate files and stores these file in the new region servers' data nodes.
- ✓Similar to this, several requests might be coming to Hmaster, making it too busy to perform all these work by itself.

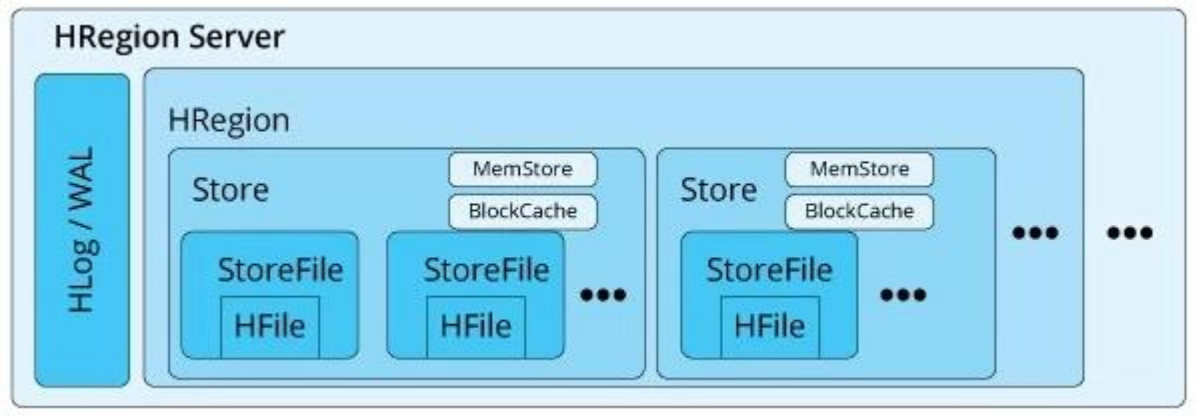
HMaster


Zookeeper
HMASTER
Program running in Master
Query
Slave 1
Slave 2
Slave N
HRegion Server
HLog / WAL
HRegion
Store
MemStore
BlockCache
StoreFile
HFile

Region Server

✓Region Server actually stores data.
✓ Region server does both the work of reading and writing data into the table.
✓ Write Ahead Log (WAL) gives fault tolerant feature, which is also known as Hlog.
✓ A region of a table is served to the client by a Region Server.
✓ A region server can serve about 1,000 regions (which may belong to the same table or different tables).
✓ When accessing data, the clients communicate with HBase Region Servers directly.
✓ Region Server has BlockCach, which is a read cache that frequently stores the read data in memory. Least Recently Used data is removed when the cache is full.
✓ Region Server have MemStore, which is the write cache. It keeps the new data which has not yet been written to disk. There is one MemStore available per column family per region.
✓ Hfiles store the rows as sorted KeyValues on disk.
✓ Multiple Hfile make 1 region.

The servers lay in different nodes in distributed system and are slave nodes. The master node is known as Hmaster.



Zookeeper

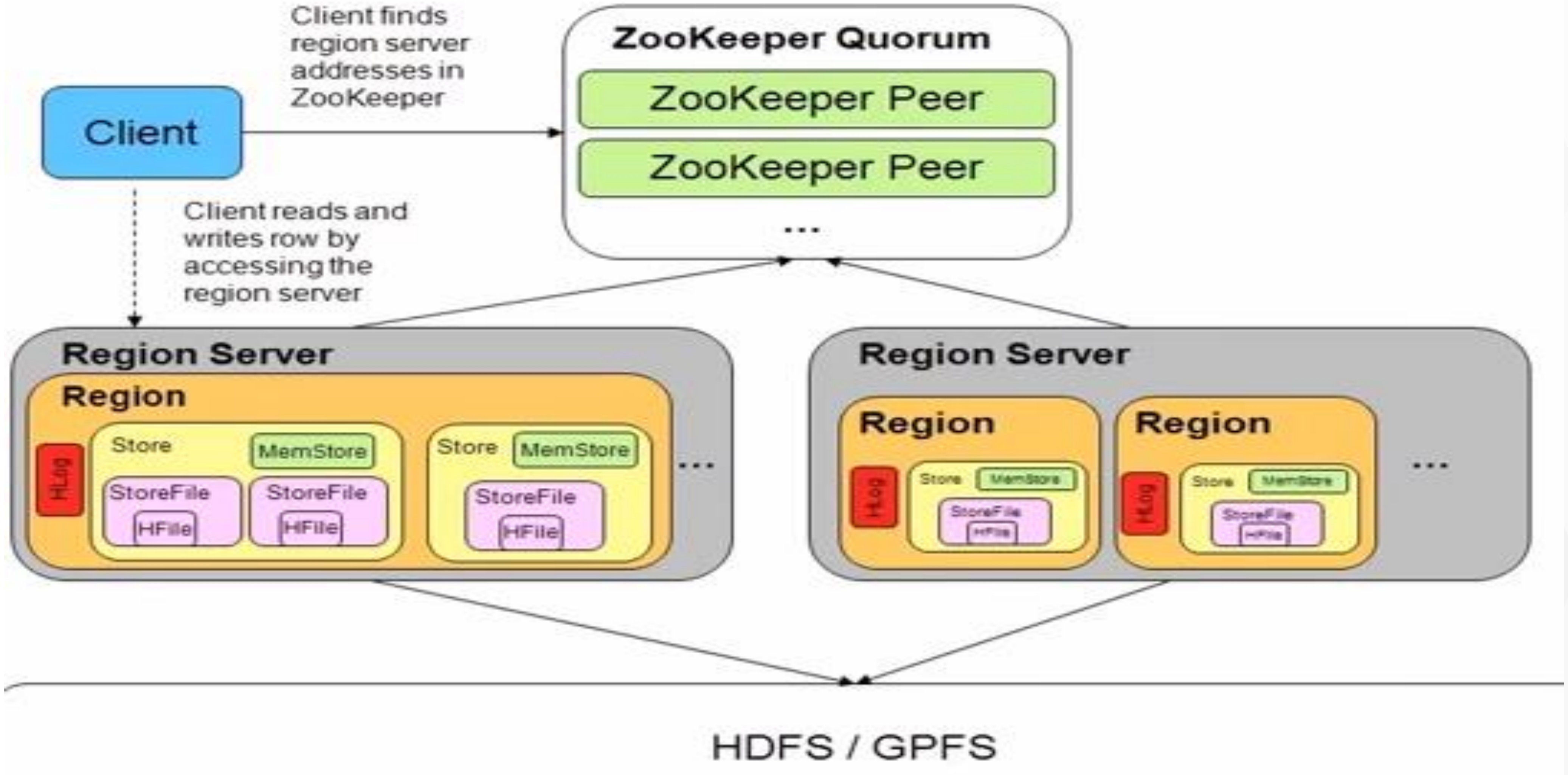
✓Manage configuration across nodes – If you have dozens or hundreds of nodes, it becomes hard to keep configuration in sync across nodes and quickly make changes. ZooKeeper helps you to quickly push the configuration changes.

✓ Implement reliable messaging – With ZooKeeper, you can easily implement a producer/consumer queue that guarantees delivery, even if some consumers or even one of the ZooKeeper servers fails.

✓ Implement redundant services – With ZooKeeper, a group of identical nodes (e.g. database servers) can elect a leader/master and let ZooKeeper refer all clients to that master server. If the master fails, ZooKeeper will assign a new leader and notify all clients.

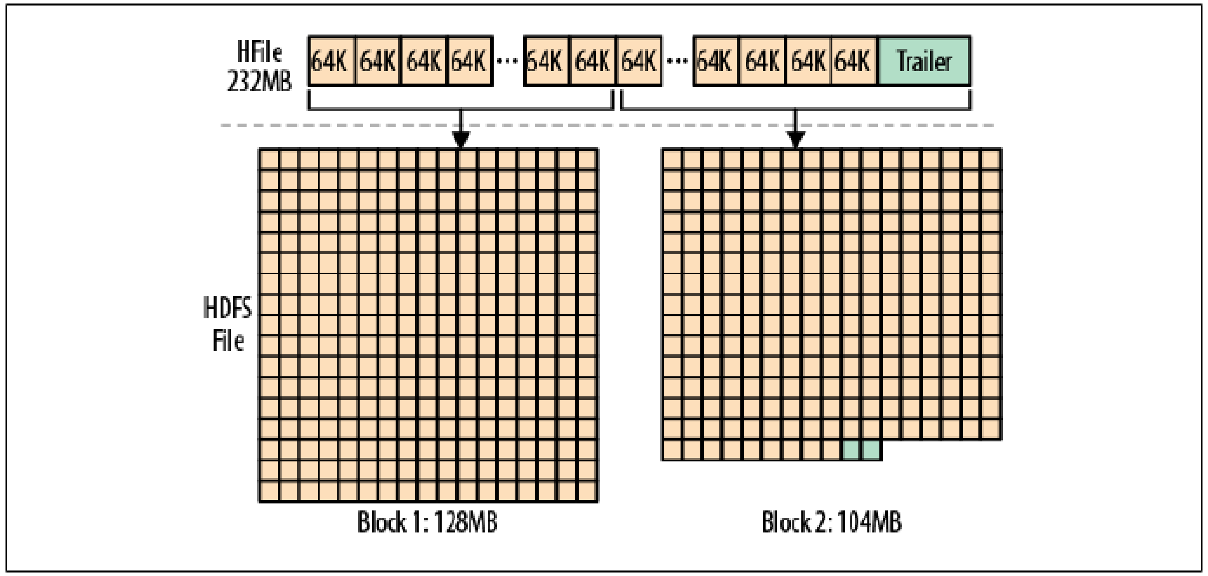
✓ Synchronize process execution – With ZooKeeper, multiple nodes can coordinate the start and end of a process or calculation. This ensures that anyfollow-up processing is done only after all nodes have finished their calculations.

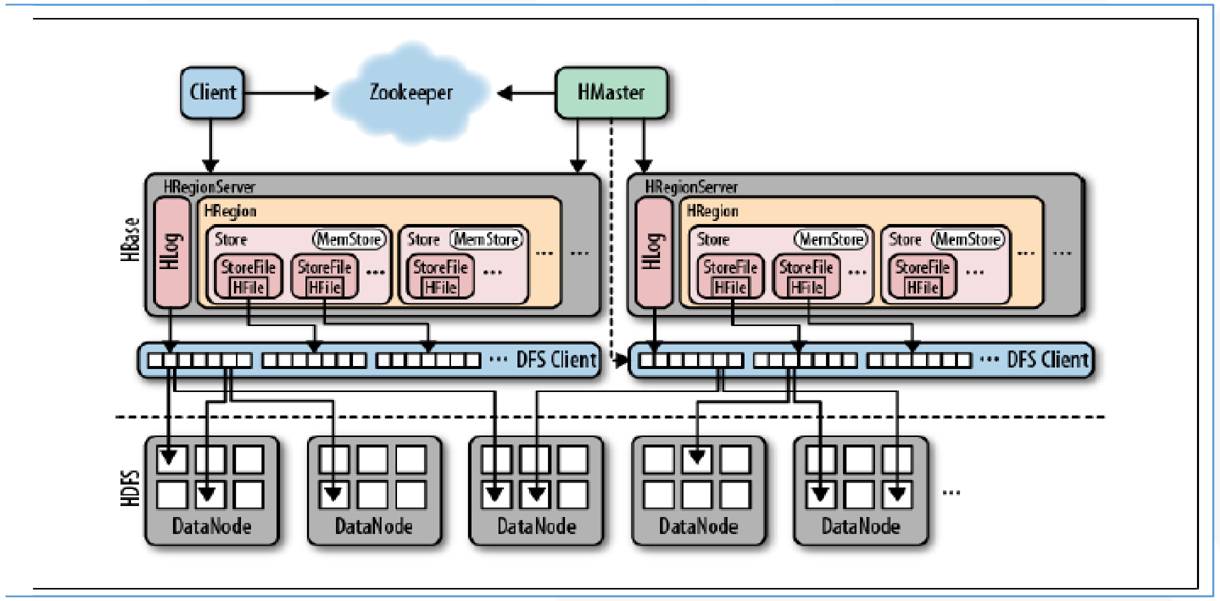
HBase Cluster Architecture



Request workflow

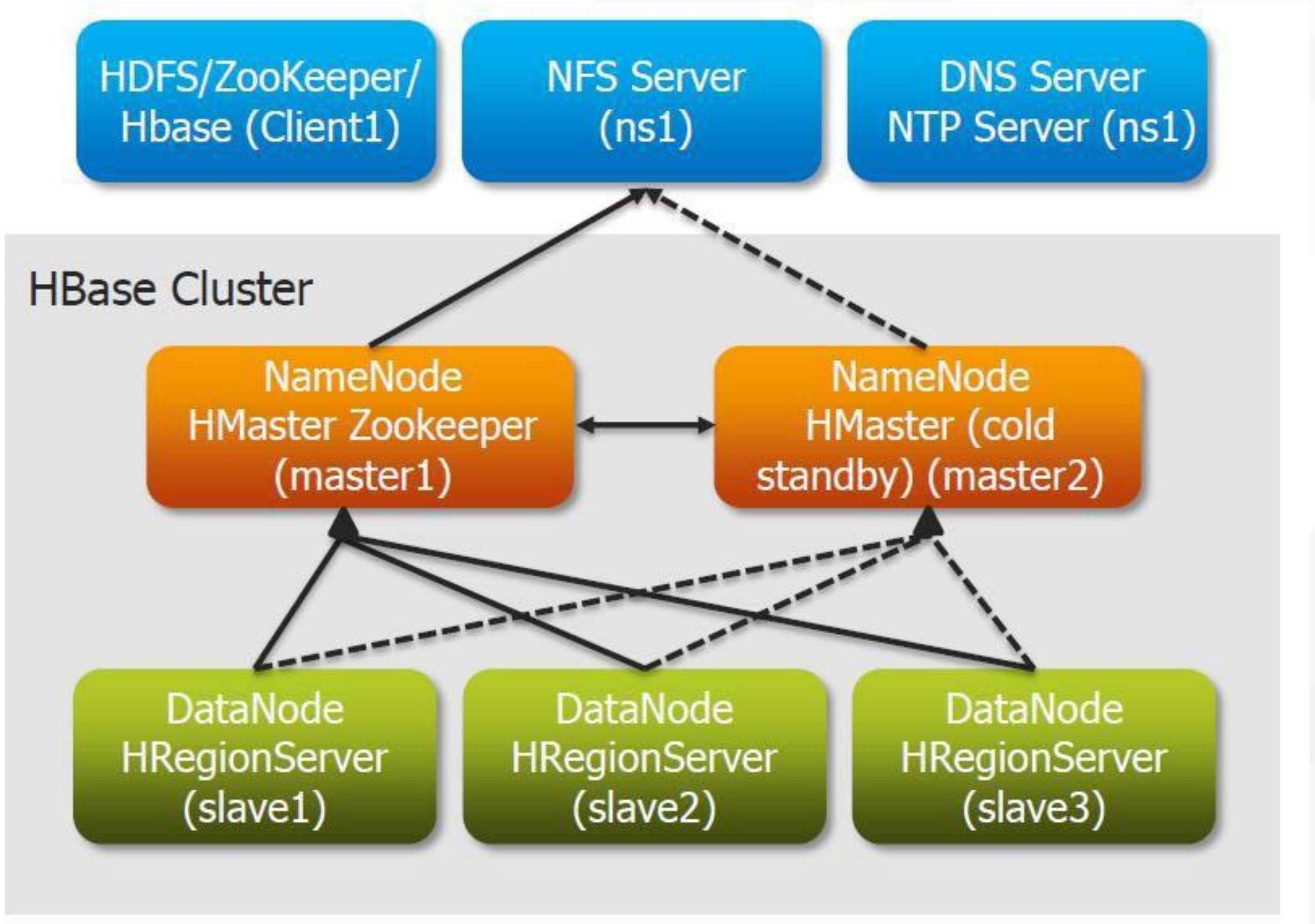
- First client will contact Zookeeper ensemble
- Retrieve the region server name that host .META table
- Query the region server to get the server that hosts the .META table that host particular row
- client cache this information and HRegionServer hosting that region directly
- The client issues an Htable.put request to the Hregion server
- First step is to write data to the write ahead log(Hlog)
- Once the data is written to WAL, it is placed in the MemStore.
- Once MemStore is full, it flushed data and written to Hfile which is stored in HDFS





ANALYTIXLABS

High availability


HDFS/ZooKeeper/ Hbase (Client1)
NFS Server (ns1)
DNS Server NTP Server (ns1)
HBase Cluster
NameNode HMaster Zookeeper (master1)
NameNode HMaster (cold standby) (master2)
DataNode HRegionServer (slave1)
DataNode HRegionServer (slave2)
DataNode HRegionServer (slave3)

Managing Hbase

- HBase Master Web UI

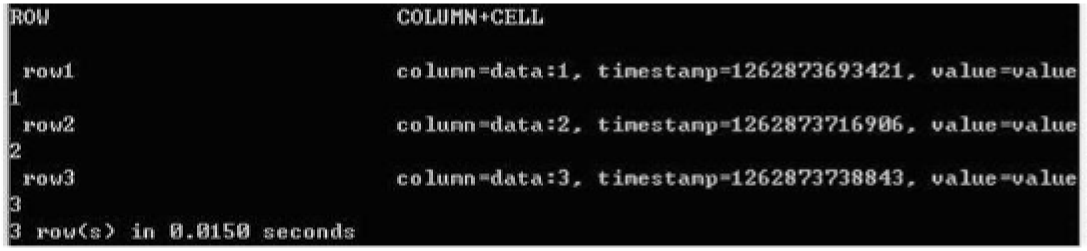
  http://hbase_master_server:60010/master.jsp

- HBase Shell
  - manage tables
  - access data
  - manage the cluster
  - execute Java methods

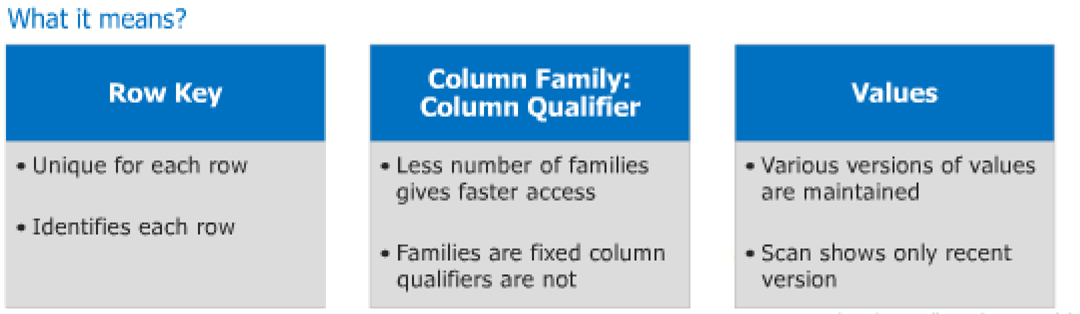
Managing Hbase

- Row Counter
- WAL tool
- Hfiletool
- hbck-checking
  - health of an HBase cluster,
  - Its fsck for HBase (HBaseFsck),
- Query Hbase using a SQL-like Language
  - Apache Hive on Hbase
  - Apache Phoenix on Hbase
  - Apache Drill on Hbase
  - Apache Spark on Hbase

How does it look like?



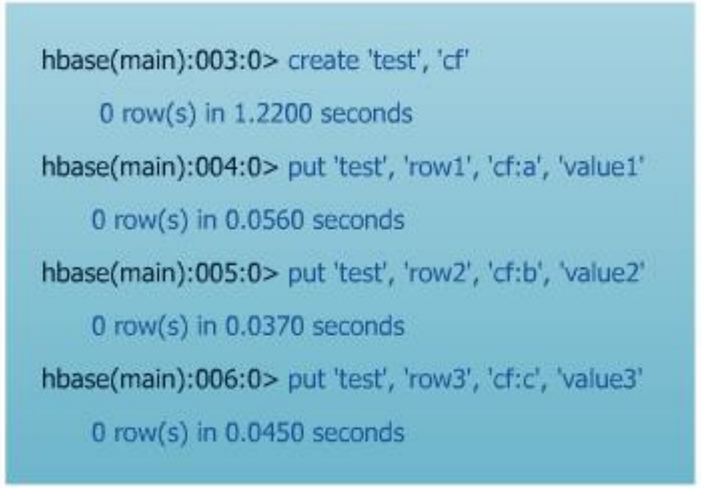
```
ROW                           COLUMN+CELL

 row1                         column=data:1, timestamp=1262873693421, value=value
1
 row2                         column=data:2, timestamp=1262873716906, value=value
2
 row3                         column=data:3, timestamp=1262873738843, value=value
3
3 row(s) in 0.0150 seconds
```



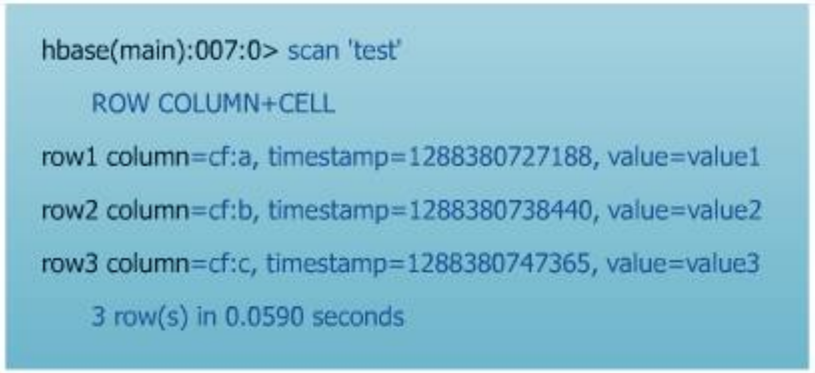
What it means?

Row Key	Column Family: Column Qualifier	Values
• Unique for each row • Identifies each row	• Less number of families gives faster access • Families are fixed column qualifiers are not	• Various versions of values are maintained • Scan shows only recent version

Hbase shell

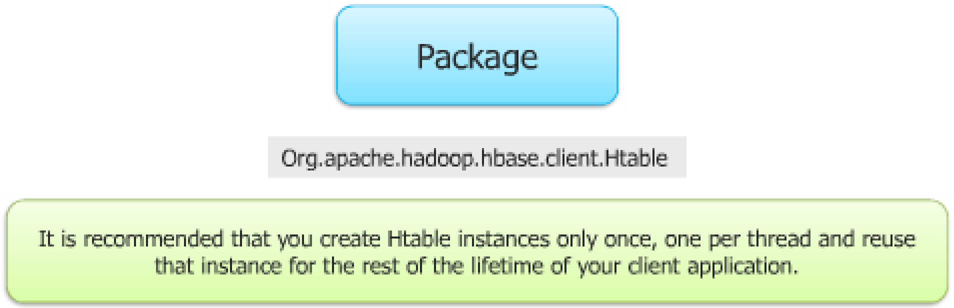


```
hbase(main):003:0> create 'test', 'cf'
    0 row(s) in 1.2200 seconds
hbase(main):004:0> put 'test', 'row1', 'cf:a', 'value1'
    0 row(s) in 0.0560 seconds
hbase(main):005:0> put 'test', 'row2', 'cf:b', 'value2'
    0 row(s) in 0.0370 seconds
hbase(main):006:0> put 'test', 'row3', 'cf:c', 'value3'
    0 row(s) in 0.0450 seconds
```



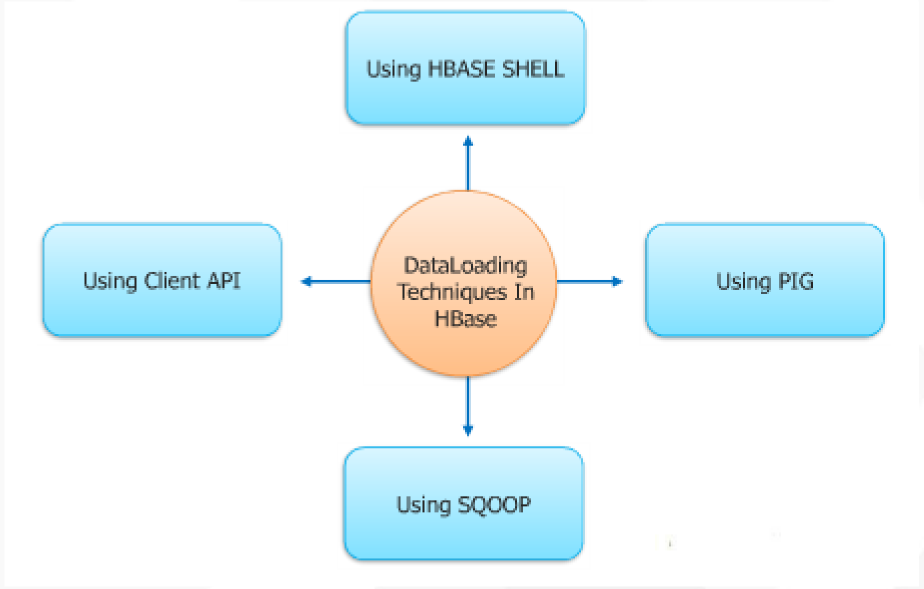
```
hbase(main):007:0> scan 'test'
    ROW COLUMN+CELL
row1 column=cf:a, timestamp=1288380727188, value=value1
row2 column=cf:b, timestamp=1288380738440, value=value2
row3 column=cf:c, timestamp=1288380747365, value=value3
    3 row(s) in 0.0590 seconds
```

Hbase Client API

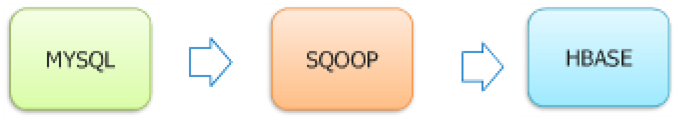


It is recommended that you create Htable instances only once, one per thread and reuse that instance for the rest of the lifetime of your client application.

Data Loading Techniques



Loading data into Hbase using sqoop



Sqoop can be used to directly import data from RDBMS to Hbase:

Example:

```
sqoop import
--connect jdbc:mysql://<ip address>\<database name>
--username <username_for_mysql_user> --password <Password>
--table <mysql_table name>
--hbase-table<hbase_target_table_name>
--column-family <column_family_name>
--hbase-row-key <row_key_column>
--hbase-create-table
```

Hbase Java client interfaces

Configuration	Where to find the cluster and tuneable settings. Similar to JDBC connection String
HBaseAdmin	Helps to manage administrative tasks
HBaseDescriptor	Has the details of the table
HTable	Is a handle on a single table. Is used to issue Put, Get, Scan commands to the table

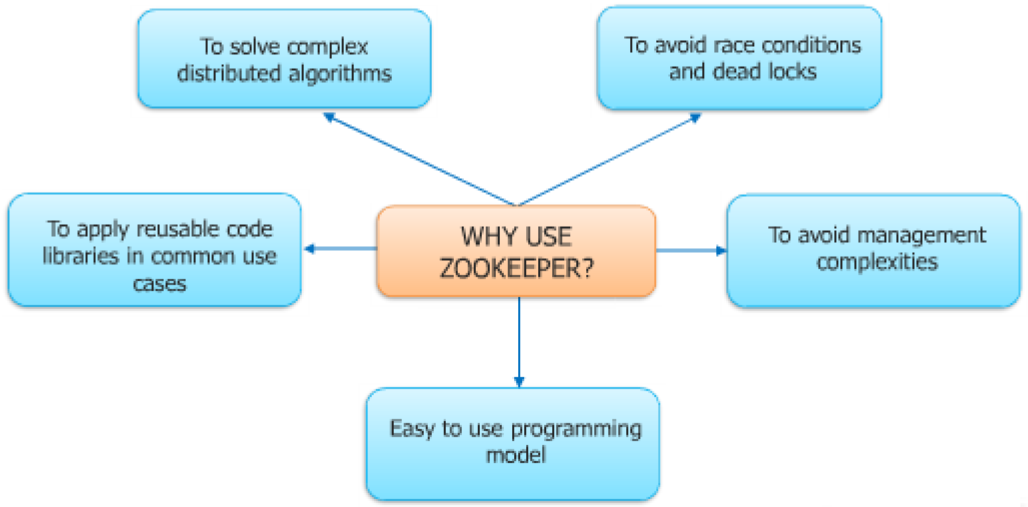
Zookeeper

Hbase is using ZooKeeper as its distributed coordination service. This includes tracking of region servers, where root regions is hosted Hbase creates a list of znodes under its root node. The default is /hbase and is configured with the zookeeper.znode.parent property. Here is list of contained znodes and their purpose

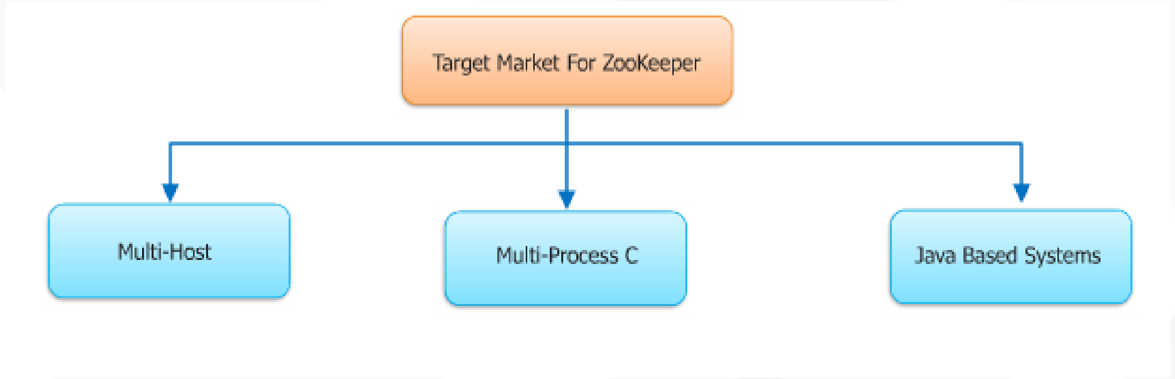
http://localhost:60010/master-status

- [cloudera@localhost bin]$ ./zkCli.sh -server localhost.localdomain
- [zk: localhost.localdomain(CONNECTED) 0] get /hbase/hbaseid
- [zk: localhost.localdomain(CONNECTED) 0] get /hbase/master
- [zk: localhost.localdomain(CONNECTED) 0] get /hbase/root-region-server
- [zk: localhost.localdomain(CONNECTED) 0] get /hbase/rs
- [cloudera@localhost ~]$ hadoop dfs -ls -R /hbase

Why use Zookeeper


To solve complex distributed algorithms
To avoid race conditions and dead locks
To apply reusable code libraries in common use cases
WHY USE ZOOKEEPER?
To avoid management complexities
Easy to use programming model

Target Market for zookeeper

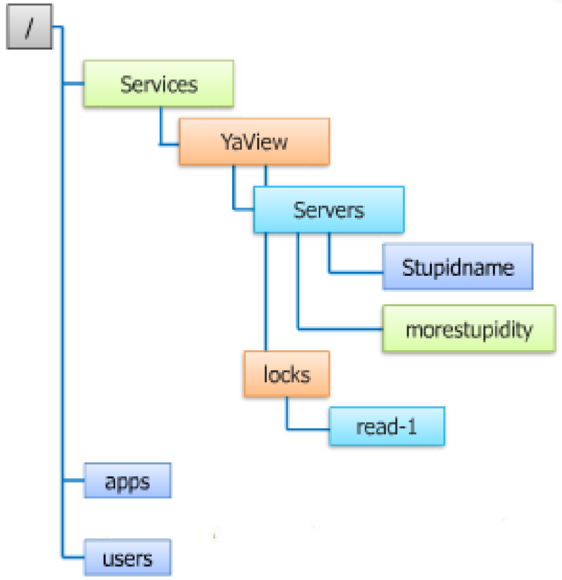


Zookeeper Data Model

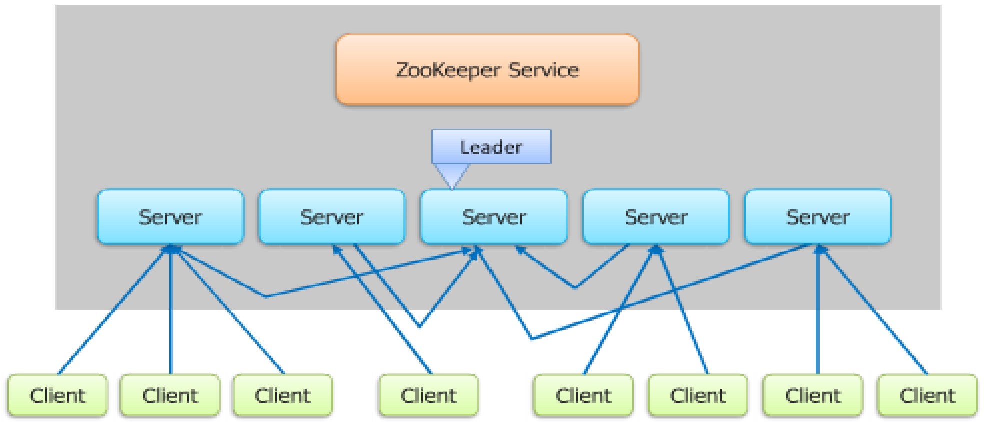
→ Hierarchal namespace (like a File System)

→ Each znode has data and children

→ Data is read and written in its entity



Zookeeper Service

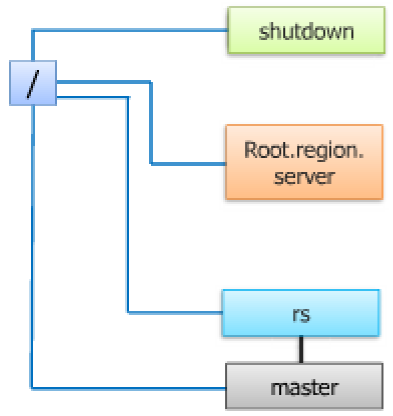


1. Wait Free
2. Simple, Robust, Good Performance
3. Tuned for Read Dominant Workloads
4. Familiar Models and Interfaces
5. Need to be able to Wait Efficiently

Zookeeper and Hbase

- → Master Failover
- → Region Servers and Master discovery via ZooKeeper
  - » HBase clients connect to ZooKeeper to find configuration data
  - » Region Servers and Master failure detection

Zookeeper and Hbase



- → Master
  - → If more than one master, they fight.
- → Root Region Server
  - → This znode holds the location of the server hosting the root of all tables in Hbase.
  - → A directory in which there is a znode per Hbase region server
  - → Region Servers register themselves with ZooKeeper when they come online

On Region Server failure (detected via ephemeral znodes and notification via ZooKeeper), the master splits the edits out per region.

CRUD Operations in HBase

CRUD Operations

There are four main operations performed in HBase and they are:

- ✓Create
- ✓Read (Get and Scan)
- ✓Add (update)
- ✓Delete.

Read & Write operations in Hbase

Two major components which play vital role in Read and Write are: HFile and META Table.

HFile

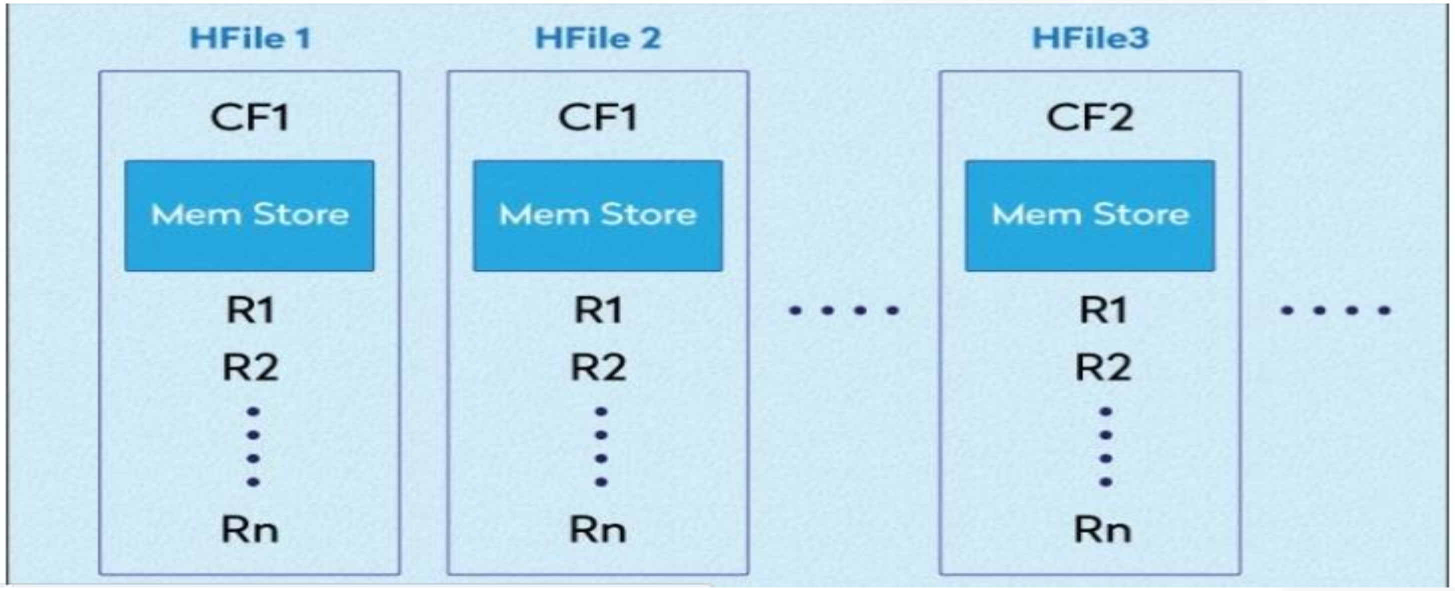
It is the basic level HBase architecture where the tables exist in physical form. It is important to understand this component since *Read* and *Write* can take place here.

The key features of HFile are:

- ✓Row key is primary identifier.
- ✓Keys are stored in lexicographical order
- ✓According to this order, data is stored and split across the nodes.
- ✓HFile is allocated to 1 region
- ✓HFiles store the rows as sorted KeyValues on disk.
- ✓When the MemStore accumulates data more than its limit, the entire sorted set is written to a new HFile in HDFS.
- ✓HBase uses multiple HFiles per column family, containing the actual cells, or KeyValue instances.
- ✓The highest sequence number is stored as a meta field in each HFile, to better state where it has ended previously and where to continue next.

Read & Write operations in Hbase

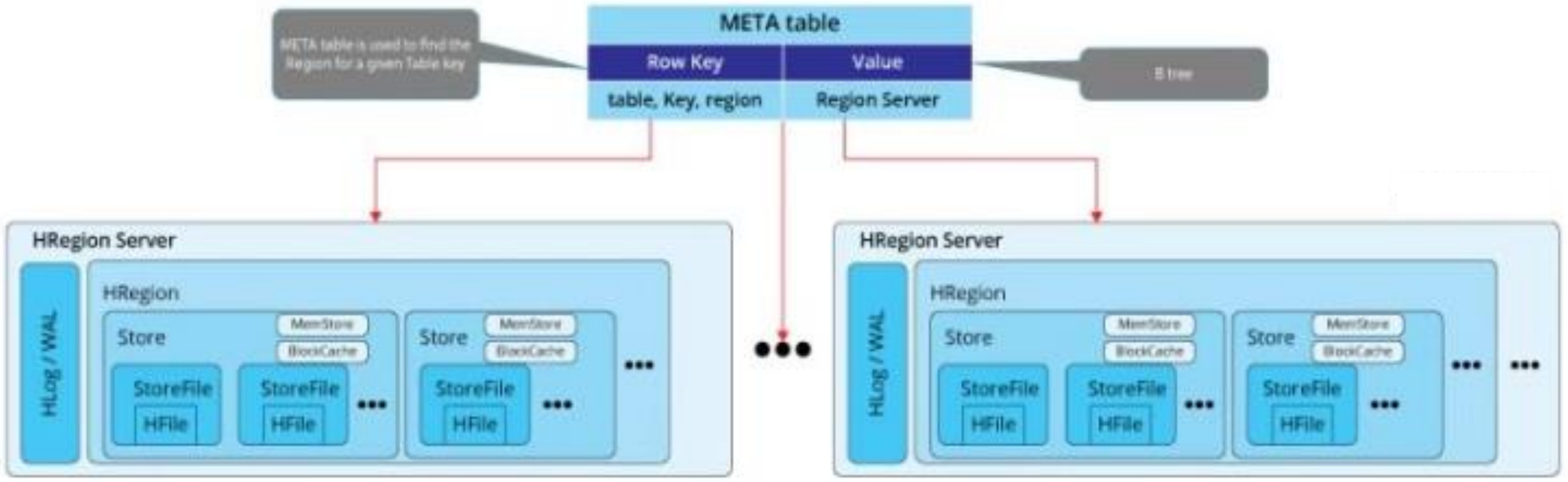
✓The highest sequence number is stored as a meta field in each HFile, to better state where it has ended previously and where to continue next.
✓HFile contains a multi-layered index which allows HBase to search the data without having to read the whole file.
✓HDFS replicates the WAL and HFile blocks
✓HFile block replication happens automatically
✓IO in HBase happens at HFile block level which is 64KB by default
✓One HFile can contain only one column family. What if data grows in a particular HFile? Different HFile is created with same column family and writing data is continued.
✓No two column families can exist in single HFile.



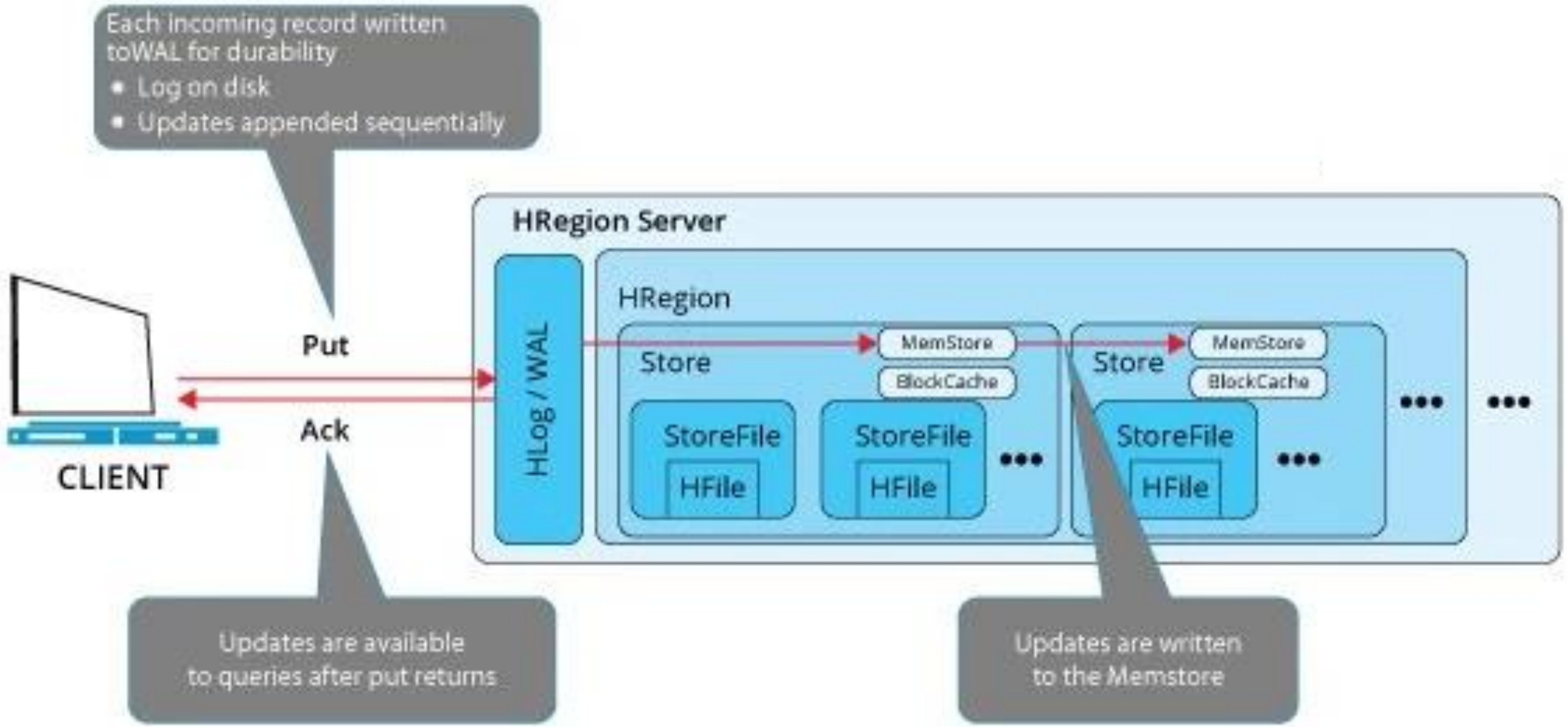
Read & Write operations in Hbase

Integrating HFile component to have HRegion is controlled by HRegion Server.
Another component is META Table. This is majorly used in Read operation, as Read operation needs to know which HRegion server has to be accessed for reading actual data.
Also, after every Write process, this table is updated so that for the next Read, the table will have the updated data.

META Table
This META table is a 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



Write operations in Hbase

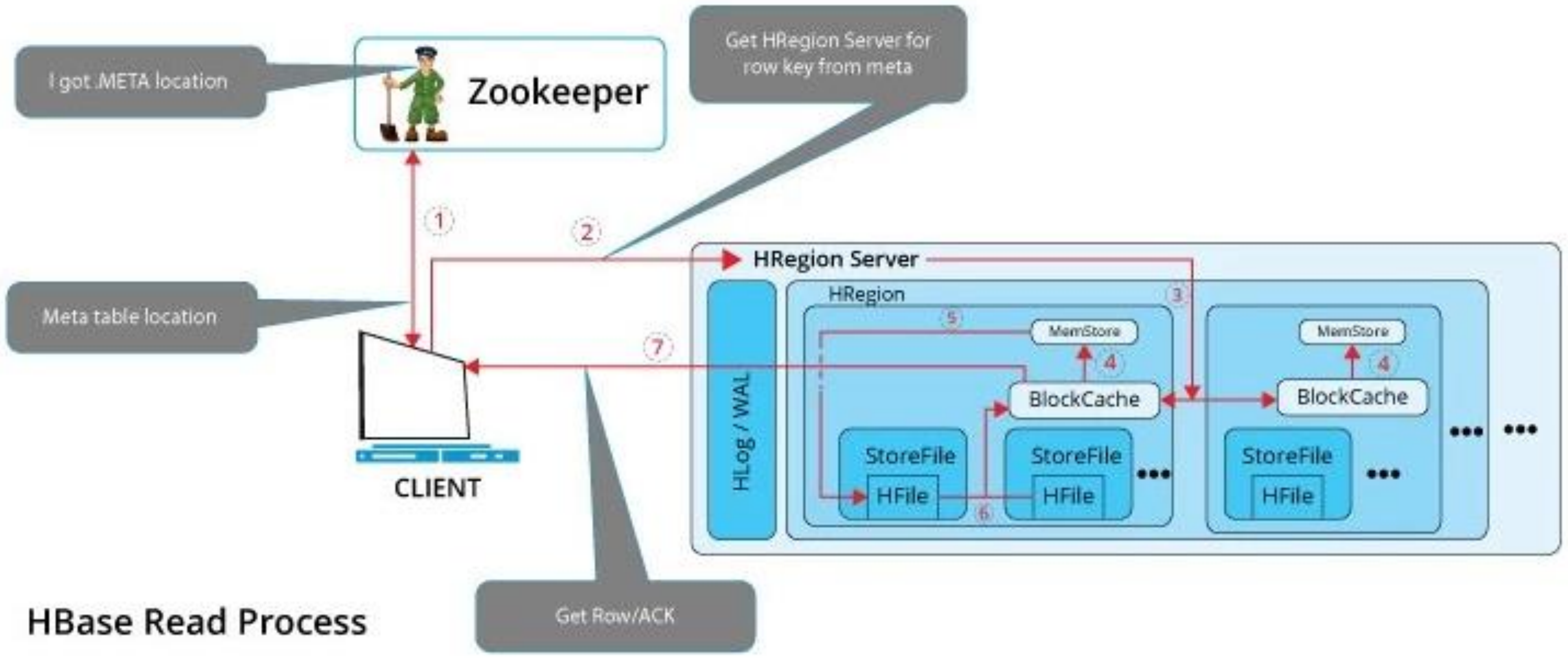

Each incoming record written toWAL for durability
• Log on disk
• Updates appended sequentially
HRegion Server
HRegion
Put
Ack
CLIENT
HLog / WAL
Store
MemStore
BlockCache
StoreFile
HFile
StoreFile
HFile
Store
MemStore
BlockCache
StoreFile
HFile
Updates are available to queries after put returns
Updates are written to the Memstore

Write operations in Hbase

When the client gives a command to Write, the following steps occur:

1. Instruction is directed to Write Ahead Log and first writes important logs to it. Although it is not the area where the data is stored, it is done for the fault tolerant purpose. So, later if any error occurs while writing data, HBase always has WAL to look into.
2. Once the log entry is done, the data to be written is forwarded to MemStore which is actually the RAM of the data node. All the data is written in MemStore which is faster than RDBMS (Relational databases).
3. Later, the data is dumped in HFile, where the actual data is stored in HDFS. If the MemCache is full, the data is stored in HFile directly.
4. Once writing data is completed, ACK (Acknowledgement) is sent to client as a confirmation of task completed.

Read operations in Hbase



Read operations in Hbase

Read process starts when a client sends request to Hbase. A request is sent to zookeeper which keeps all the status for the distributed system, where HBase is also present. Refer the figure above.

1. Zookeeper has location for META table which is present in HRegion Server. When a client requests zookeeper, it gives the address for the table (1).
2. The process continues to Hregion Server and gets to META table, where it gets the region address of table where the data is present to be read (2).
3. Moving forward to a specific HRegion, the process enters the BlockCache where data is present from previous read. If a user queries the same records, the client will get the same data in no time. If the table is found, the process returns to client with the data as result (3).
4. If the table is not found, the process starts to search MemStore since data would have been written to HFile sometime back. If it is found, the process returns to client with the data as result (4).
5. If the table is not found, the process moves forward in search of data within the HFile. The data will be located here and once the search is completed, the process takes required data and moves forward (5).
6. The data taken from HFile is the latest read data and can be read by user again. Hence the data is written in BlockCache, so that the next time, it can be instantly accessed by the client (6). This is the benefit of step 6; the read process can be completed just after step 3 the next time for the same data because of this read procedure of Hbase.
7. When the data is written in BlockCache and all the search is completed, the read process with required data will be returned to client along with ACK(Acknowledgment) (7).

Interacting with Hbase db via Hbase shell

- hbase(main):010:0>t = create 'testtable2', 'colfam1‘
- hbase(main):011:0> t.put 'row1','colfam1:qual1',’val1’
- hbase(main):010:0> create 'testtable2', 'colfam1‘
- hbase(main):010:0> t = get_table 'testtable2‘
- hbase(main):024:0> t.scan
- hbase(main):026:0> t.describe
- hbase(main):027:0> t.disable
- hbase(main):029:0> t.scan
  ERROR: org.apache.hadoop.hbase.DoNotRetryIOException: testtable2 is disabled
- hbase(main):006:0> t.enable
- hbase(main):007:0> t.scan
- hbase(main):008:0> t.disable
- hbase(main):008:0> t.disable
- hbase(main):010:0> list

<< Please note, do not copy command from here just type othewise it will copy junk char along which will not allow you to perform any above operation>>

Use cases

- **HBase at Groupon**
  Groupon has two distinct use cases. Deliver deals to users via email (a batch process) and provide a relevant user experience on the website. They have increasingly tuned their deals to be more accurate and relevant to individual users (personalization).
- Facebook uses HBase to power their Messages infrastructure
- Webtable

http://wiki.apache.org/hadoop/Hbase/PoweredBy

Apache HBASE – Use Cases

- These following examples make HBase a great choice for storing semi-structured data like log data and then providing that data very quickly to users or applications integrated with HBase.
- One company that provides web security services maintains a system accepting billions of event traces and activity logs from its customer' desktops every day.
- The company's programmers can tightly integrate their security solutions with HBase (to assure that the protection they provide keeps pace with real-time changes in the threat landscape.)

Apache HBASE – Use Cases

- Another company provides stock market ticker plant data that its users query more than thirty thousand times per second, with an SLA of only a few milliseconds.
- Apache HBase provides that super low-latency access over an enormous, rapidly changing data store.
- Enterprises use Apache HBase's low latency storage for scenarios that require real-time analysis and tabular data for end user applications.

Why do we need to integrate Hive with HBase?

✓Hive can store information of hundreds of millions of users effortlessly, but faces some difficulties when it comes to keeping the warehouse up to date with the latest information.
✓Hive uses HDFS as an underlying storage which come with limitations like append-only, block-oriented storage. This makes it impossible to directly apply individual updates to warehouse tables.
✓Up till now the only practical option to overcome this limitation is to pull the snapshots from MySQL databases and dump them to new Hive partitions.
✓This expensive operation of pulling the data from one location to another location is not frequently practiced.(leading to stale data in the warehouse), and it also does not scale well as the data volume continues to shoot through the roof.

To overcome this problem, Apache HBase is used in place of MySQL, with Hive.

How is HBase integrated with Hive?

✓For integrating HBase with Hive, **Storage Handlers** in Hive is used.

✓Storage Handlers are a combination of InputFormat, OutputFormat, SerDe, and specific code that Hive uses to identify an external entity as a Hive table.

✓This allows the user to issue SQL queries seamlessly, whether the table represents a text file stored in Hadoop or a column family stored in a NoSQL database such as *Apache HBase*, *Apache Cassandra*, and *Amazon DynamoDB*.

✓Storage Handlers are not only limited to NoSQL databases, a storage handler could be designed for several different kinds of data stores.

Appendix

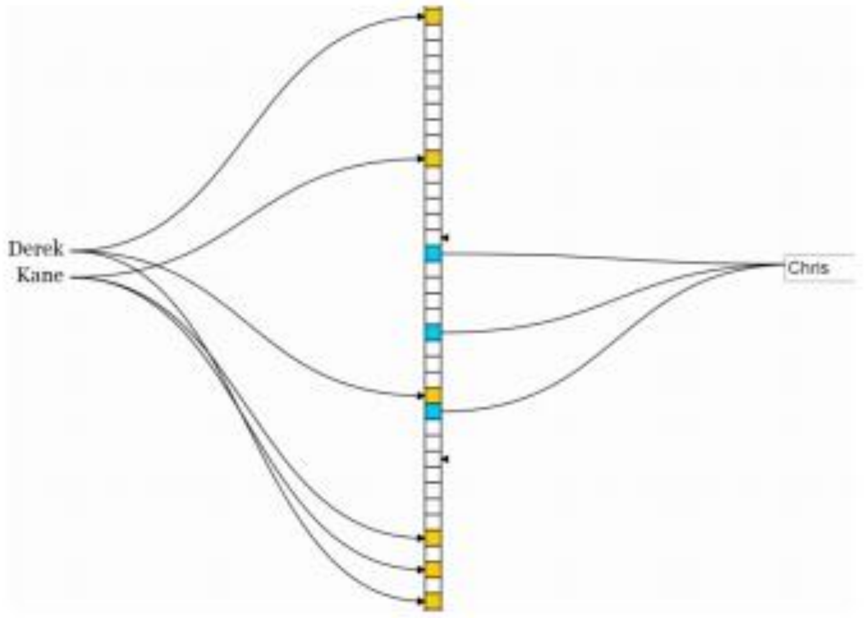
Hbase features

- Linear and modular scalability.
- Strictly consistent reads and writes.
- Automatic and configurable sharding of tables
- Automatic failover support between RegionServers.
- Convenient base classes for backing Hadoop MapReduce jobs with Apache HBase tables.

Hbase features

- Block cache and Bloom Filters for real-time queries.
- Query predicate push down via server side Filters
- Support - Thrift gateway and a REST-ful Web service
- Extensible jruby-based (JIRB) shell
- Support for exporting metrics

Apache HBASE Overview



A Bloom Filter Example

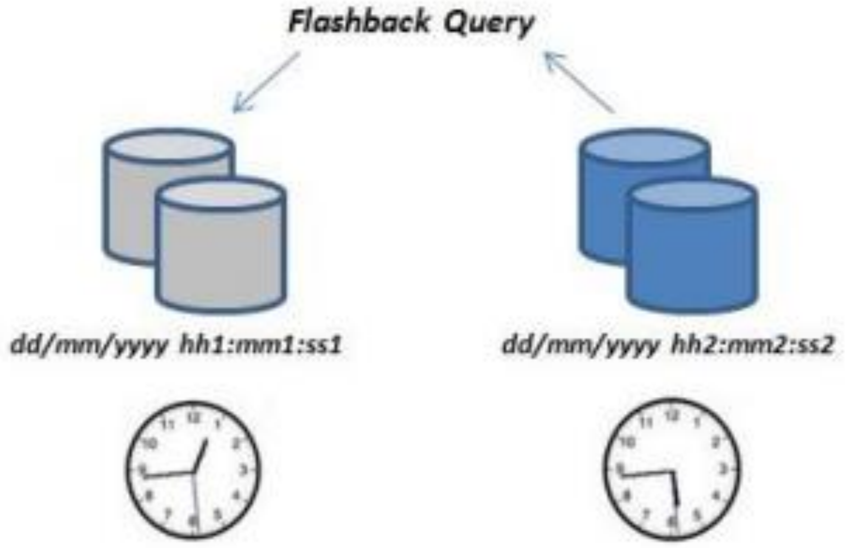
- Tables in HBase can serve as the input and output for MapReduce jobs run in Hadoop, and may be accessed through the Java API but also through REST, Avro or Thrift gateway APIs.
- HBase features compression, in-memory operation, and Bloom filters on a per-column basis as outlined in the original BigTable paper.
- A Bloom filter is a data structure designed to tell you, rapidly and memory-efficiently, whether an element is present in a set.
- If the element is not in the bloom filter, then we know for sure we don't need to perform the expensive lookup. On the other hand, if it is in the bloom filter, we perform the lookup, and we can expect it to fail some proportion of the time (the false positive rate).

Apache HBASE Overview

- Apache HBase provides a fault-tolerant way of storing large quantities of sparse data (small amounts of information caught within a large collection of empty or unimportant data.
- An example of this includes finding the 50 largest items in a group of 2 billion records, or finding the non-zero items representing less than 0.1% of a huge collection).
- HBase is not a direct replacement for a classic SQL database, although recently its performance has improved, and it is now serving several data-driven websites, including Facebook's Messaging Platform.

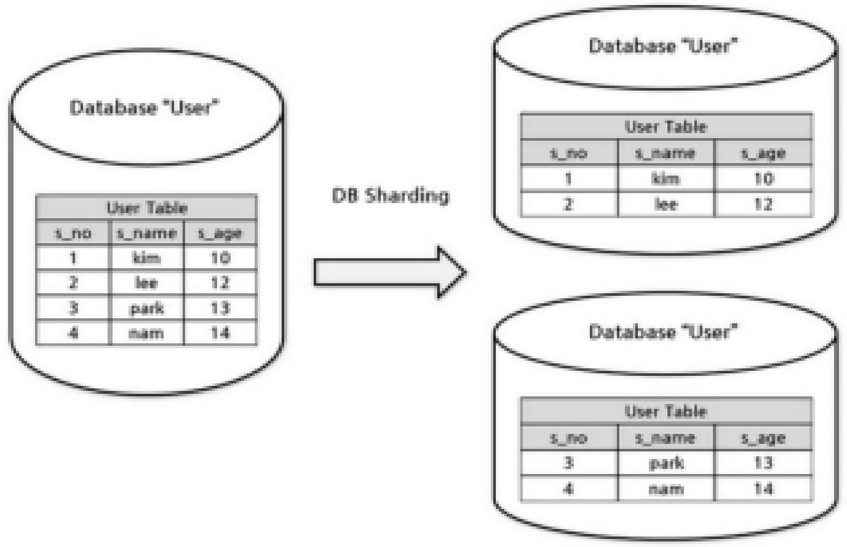
facebook

Apache HBASE Overview



- Apache HBase scales linearly to handle huge data sets with billions of rows and millions of columns, and it easily combines data sources that use a wide variety of different structures and schemas.
- Apache HBase provides random, real time access to your data in Hadoop.
- It was created for hosting very large tables, making it a great choice to store multi-structured or sparse data.
- Users can query HBase for a particular point in time, making "flashback" queries possible.

Apache HBASE



User Table

s_no	s_name	s_age
1	kim	10
2	lee	12
3	park	13
4	nam	14

User Table

s_no	s_name	s_age
1	kim	10
2	lee	12

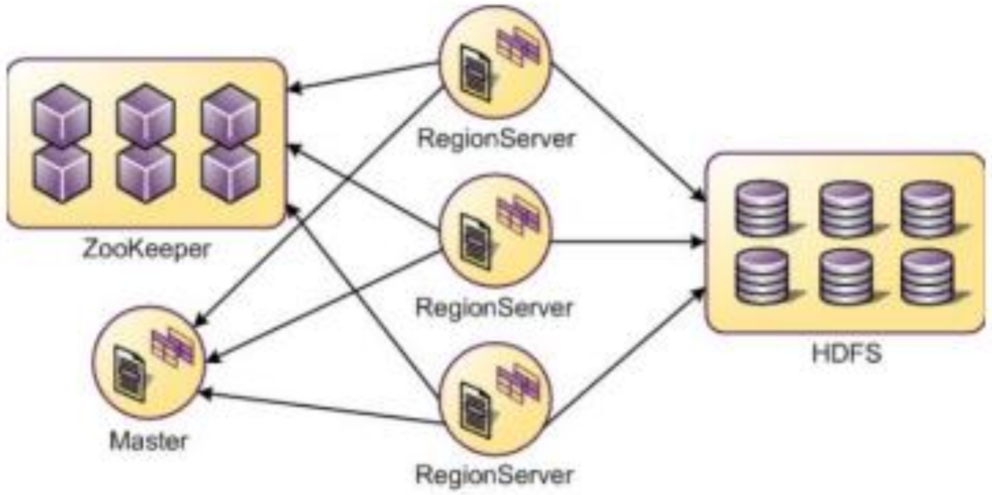
User Table

s_no	s_name	s_age
3	park	13
4	nam	14

- HBase scales linearly by requiring all tables to have a primary key.
- The key space is divided into sequential blocks that are then allotted to a region.
- RegionServers own one or more regions, so the load is spread uniformly across the cluster.
- If the keys within a region are frequently accessed, HBase can further subdivide the region by splitting it automatically, so that manual data sharding is not necessary.

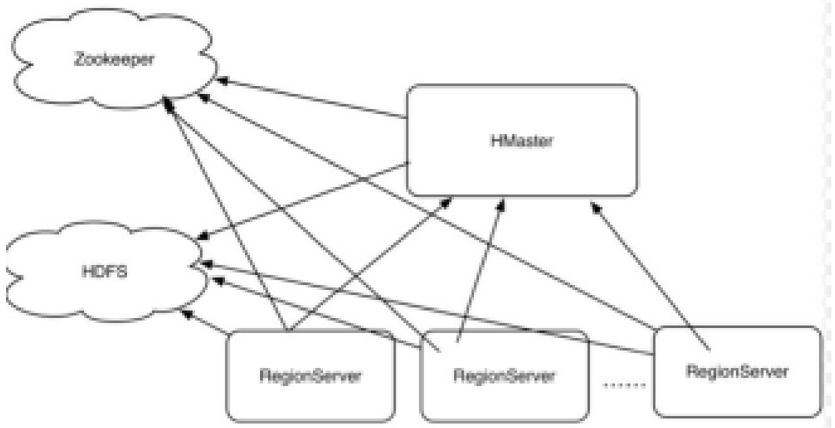
Apache HBASE

- ZooKeeper and HMaster servers make information about the cluster topology available to clients. (More on ZooKeeper later)
- Clients connect to these and download a list of RegionServers, the regions contained within those RegionServers and the key ranges hosted by the regions.
- Clients know exactly where any piece of data is in HBase and can contact the RegionServer directly without any need for a central coordinator.
- RegionServers include a memstore to cache frequently accessed rows in memory.



ANALYTIXLABS

Apache HBASE



Apache HBase provides high data availability in several ways:

- Highly available cluster topology information through production deployments with multiple HMaster and ZooKeeper instances.
- Data distribution across many nodes means that loss of a single node only affects data stored on that node.
- HBase allows data storage, ensuring that loss of a single node does not result in loss of data availability.
- HFile file format stores data directly in HDFS. HFile can be read or written to by Apache Hive, Apache Pig, and MapReduce permitting deep analytics on HBase without data movement.

- For further research on Apache HBase concepts and techniques, please refer back to Apache Software Foundation's User Forums.

Hbase architecture

- Table made of regions
- Region –a range of rows stored together
- Region servers-serves one or more regions
- A region is served by only one region server
- Master server –daemon responsible for managing HBase cluster
- HBase stores its data into HDFS
- Relies on HDFS High Availability and fault tolerable

Hbase architecture

MasterServer

- Assigns regions to the region servers and takes the help of Apache ZooKeeper for this task.
- Handles load balancing of the regions across region servers. It unloads the busy servers and shifts the regions to less occupied servers.
- Maintains the state of the cluster by negotiating the load balancing.
- Is responsible for schema changes and other metadata operations such as creation of tables and column families.

Hbase architecture

Regions

- Regions are nothing but tables that are split up and spread across the region servers.

Region server

- The region servers have regions that -
- Communicate with the client and handle data-related operations.
- Handle read and write requests for all the regions under it.
- Decide the size of the region by following the region size thresholds.

Regions are the basic element of availability and distribution for tables, and are comprised of a Store per Column Family.
The hierarchy of objects is as follows:

```
Table                    (HBase table)
    Region               (Regions for the table)
        Store            (Store per ColumnFamily for each Region for the table)
            MemStore     (MemStore for each Store for each Region for the table)
            StoreFile    (StoreFiles for each Store for each Region for the table)
                Block    (Blocks within a StoreFile within a Store for each Region for the table)
```

Contact Us

Visit us on: http://www.analytixlabs.in/

For more information, please contact us: http://www.analytixlabs.co.in/contact-us/

Or email: info@analytixlabs.co.in

Call us we would love to speak with you: (+91) 9910509849

Join us on:

Twitter - http://twitter.com/#!/AnalytixLabs

Facebook - http://www.facebook.com/analytixlabs

LinkedIn - http://www.linkedin.com/in/analytixlabs

Blog - http://www.analytixlabs.co.in/category/blog/