Overview of HBase

Author: Peter Bugaj

Table of Contents

Main Components 2

- **Region Server** 3
- **Region** 5
- **HMaster** 6
- **Zookeeper** 7
- **WAL File** 8
- **BlockCache** 8
- **MemStore** 8
- **HFile** 9
- **Table** 11
- **Column Famliy** 12
- **Indexing** 13

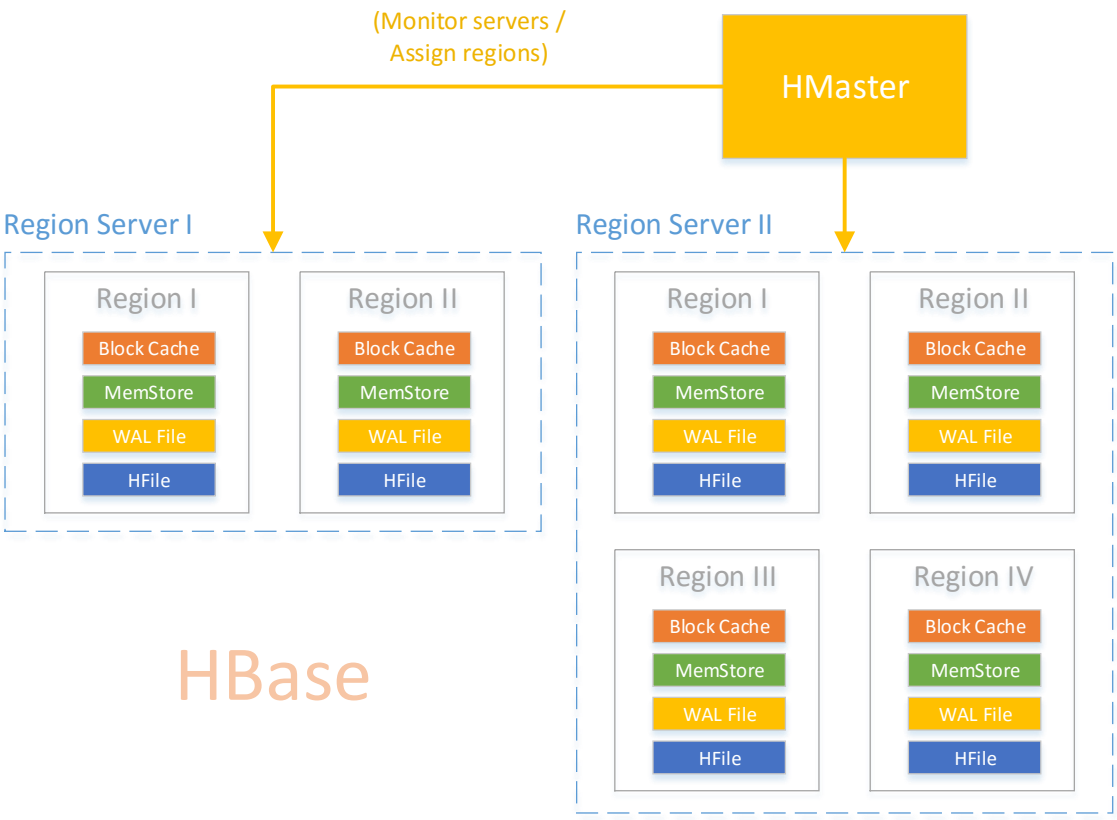
Exercises 14

- **Setting up** 14
- **Basic operations** 14
- **Performance** 15
- **Indexing** 15
- **Crash Recovery** 16

Online References 17

Main Components

- **Region Server**, which manages the regions on HDFS and communicates with the clients.
- **Region**, which stores a a portion of a table on HDFS.
- **WAL file**, the write ahead log.
- **MemStore**, for caching data in memory before being inserted into disk inn chunks.
- **HFile**, for storing the actual data for a region.
- **Block Cache**, for storing frequently read data.
- **HDFS Data node**, the machine that everything runs on.
- **Table**, for storing data across one or more regions.
- **Column Families**, that divides a row into stores.
- **Column**, being a collection of key value pairs.
- **Cell**, which is a row, column, version tuple.
- **HMaster**, for managing Region Servers.
- **Zookeeper**, used by HMaster to stay up to date on the state of each Region Server.



Region Server

- Interfaces with the client API. The client communicates with the server directly to fetch or to insert data. The Zookeeper gives a metadata file to tell which Server the client should communicate to.

- Serves and manages data in regions, which have a configurable size setting. Several hundred regions can be managed by one such region server.

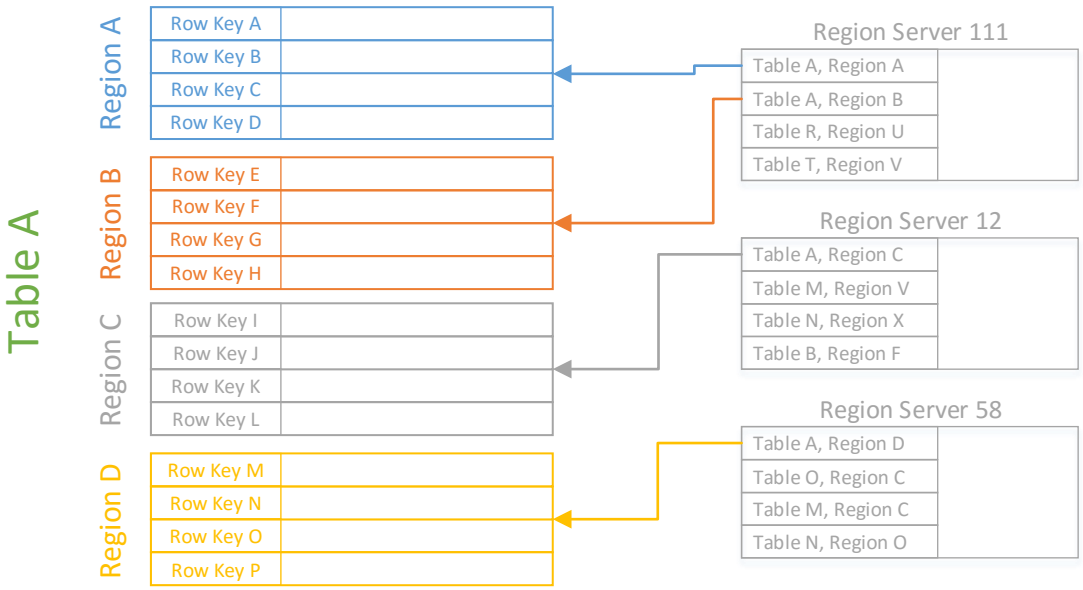
- Data for the region server is stored on the same HDFS node the server is running on. Data remains collocated, with the exception of a region being moved to another region server for load balancing. In such a case, a region then gets managed by a different region server, with the data in its HFile stored on the node of the previous server. This locality is fixed through periodic major compaction which rewrites the HFiles and collocates them on the server their region is under.

- When a server crashes, the regions containing the server's HFiles are reassigned to another active server. This also breaks the locality of the data. The memCache of the crashed server is recovered by reading through the WAL file and by sending the WAL data to the location of the new regions. The HFiles are replicated by HDFS in the case when the original HFile is not accessible.

- Monitored by the HMaster through the Zookepeer. The Zookeeper receives heart beats from the server about its state.

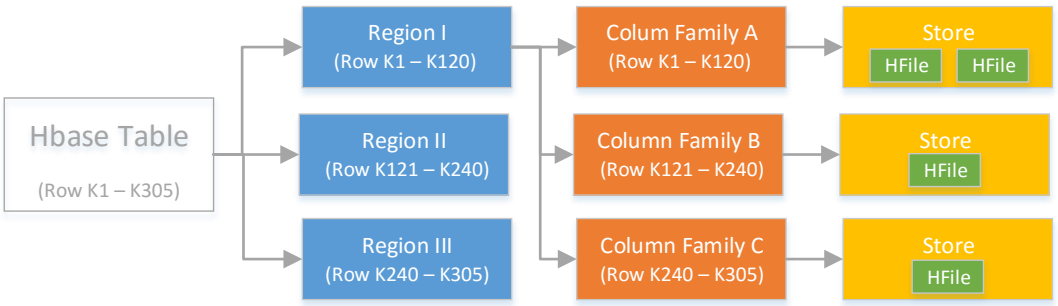
- The HMaster sends metadata files to the Zookeeper about the regions stored on the servers.

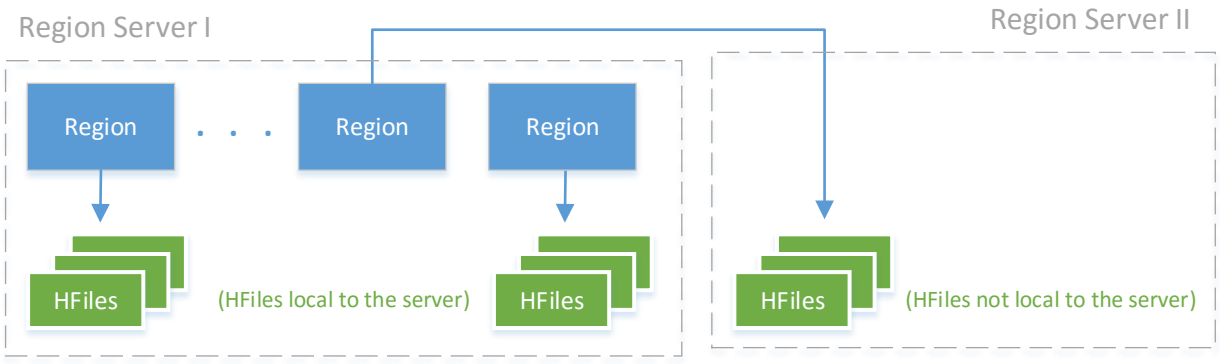
- Has a Block Cache for frequently read data, and a MemCache, for caching writes not yet written to an HFile. Hence it includes a WAL file to log this unsaved data written to the MemCache.



Region

- Stores table rows between a specific start and end key assigned to that region.
- Created by the HMaster, and then assigned by the HMaster to be managed by a Region Server.
- When a table is created, it is stored in only one region. As it grows, it gets split across more regions.
- The region rows are divided by Column Family which are managed by entities called Stores. Each store stores data in an HFile, which may or may not be collocated with the region server. Locality breaks when a region is moved, either due due to load balancing, or region server failure.
- The region's HFile is collocated with the region server when first created, or after a major compaction happens.
- Managed by at most one region server.
- All columns in a column family are stored in the same region.
- The region's data is served by the region server to the client. The client uses the meta data file in the Zookeeper to know which region server to call for that region. The client will refer to the meta data file only once and use it until that region moves and a call to access that region is missed.
- During a region split, each child region contains exactly half of the data in the parent region. This split is then reported to the HMaster.
- The default sized selected determines how many servers will manage the data in parallel. A large size will force a table to stay on one server, making other servers idle. A small size will force more splits to happen.





HMaster

- Deletes, creates and assigns regions to specific regions. Communicates frequently with the Zookeeper by making these schema updates in its metadata file. This allows the HMaster to be treated as an admin for the region servers.

- Receives notifications from the Zookeeper about failed Region Servers not sending heart beats.

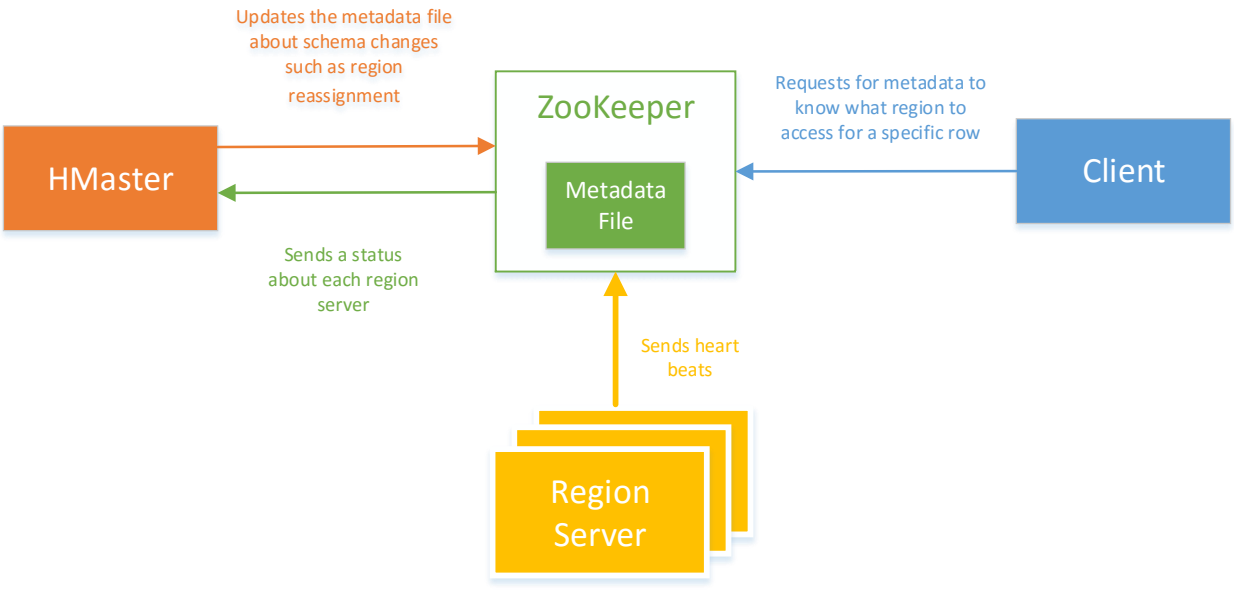
- Reassigns regions between region servers for load balancing or for recovery.

- Receives notifications about regions being split by the region servers.

- Reads the WAL file and rewrites data to other regions during a recovery process.

Zookeeper

- Receives heart beats from the region servers and notifies the HMaster when a server is down.
- Monitors the HMaster status and brings up the inactive HMaster if the current one goes down.
- Maintains the metadata file about which region is assigned to which region server and about the rows stored per region.
- Receives requests from the HMaster to update the metadata file and requests from the client to read the metafile.
- For performance purposes, the client reads the metadata file once and reads it again only if the regions where reassigned, causing it to miss a specific row.
- Keeps track of what region servers are available and provides this information to the HMaster.



WAL File

- Stores data that is not yet written to permanent storage.
- During a write request, data is first written to the WAL file, before it is written to the MemStore.
- During crash recovery, regions are first assigned to new region servers. To recover the data for each region server lost in the MemStore, the data in the WAL file is replayed for each region.
- Replicated by HDFS along with the HFiles onto mutiples servers.

BlockCache

- Stores frequently read data in memory. Typically this is any data that is read at least once.

MemStore

- Stores new data not yet written to the disk. Once enough data is stored, it is flushed to an HFile.
- Data is sorted by KeyValye and flushed to disk in that sorted order.
- One MemStore exists per column family per region.
- HMaster and MemStore both store the last written sequence number to know what is committed.

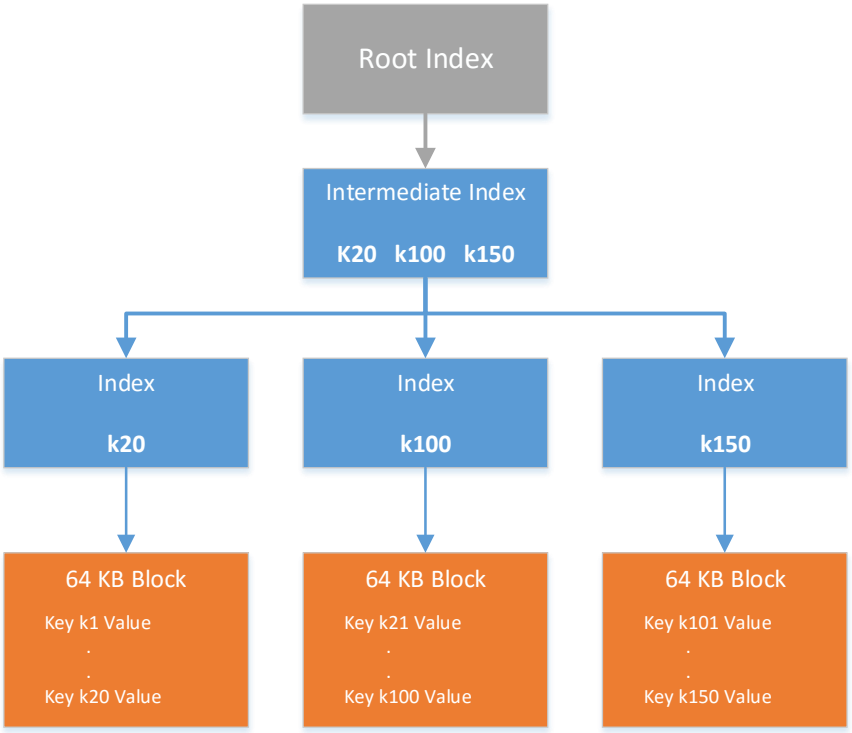
HFile

- Stores data as part of a store for a column family belonging to a region. Multiple HFiles can exist for a specific store for a column family.

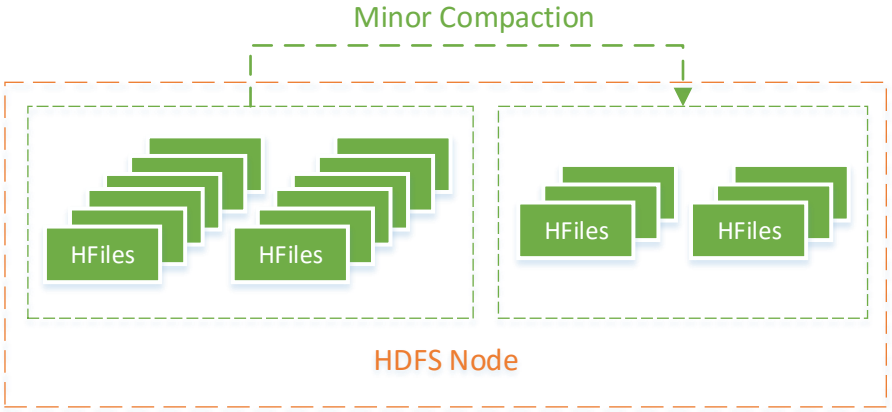
- Periodically, many small HFiles are rewritten into larger HFiles, while keeping the sorted order. This is called minor compaction.

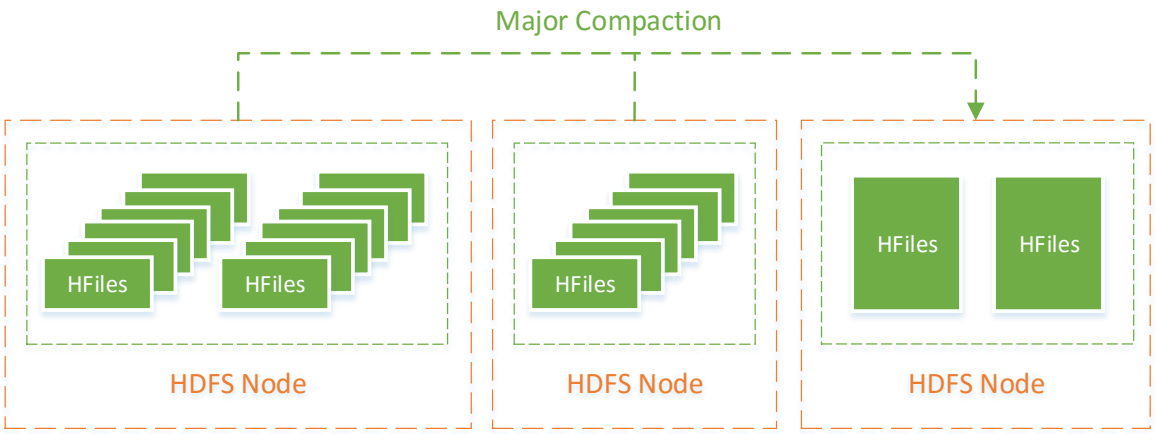
- Major compaction occurs when many HFiles get rewritten into one large HFile and when the locality of the HFile belonging to same node as the region server managing it is fixed. This is done through HBase which ensures that every region running under a region server has HFiles under the exact same node.

- Contains a multi-layered index to avoid searching through the entire HFile. The index works by the rows being sorted in increasing order by row key. Indexes then point by the row key to the key value data, stored as 64 KB blocks. A block has its own sub index. The last key of each block is stored in an intermediate index. An intermediate index is pointed to by the root index.



- Since multiple HFiles can exist per column family, multiple reads might have to happen on each HFile, and data might need to be merge. This is called read amplification. Read amplification is fixed by minor and major compactions.

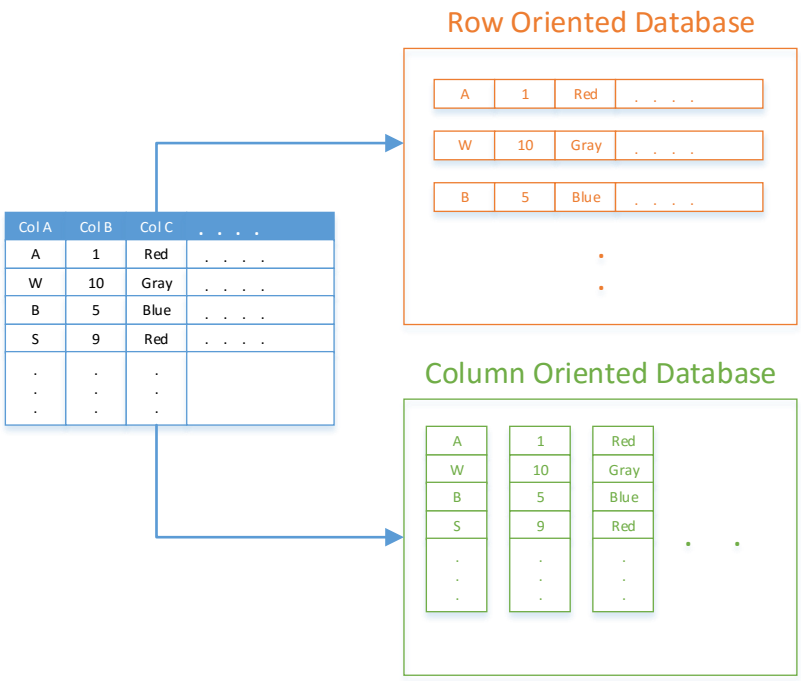




Table

- Vertically divided into column families, which are then divided into columns. Each column family is stored in its own HFile on the disk, making the database store the data in a column oriented order.

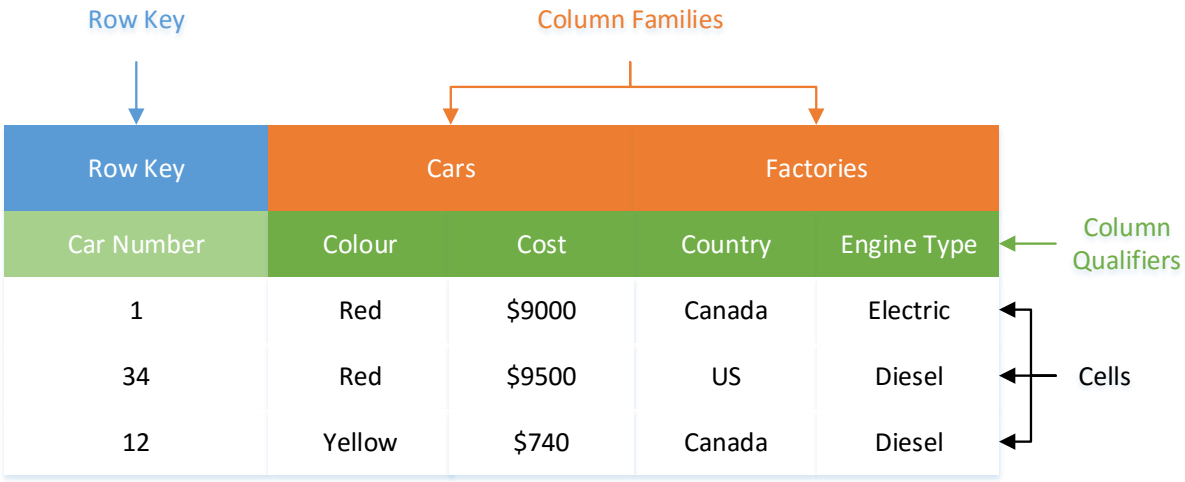
- Divided horizontally by row into regions. If a table grows to big, it is split into more regions.



Column Famliy

- Stores a group of columns together in HFiles. Each column is known as a column qualifier, which allows data to be search for more specifically within the family.

- Has its own memStore per region.



Indexing

HBase supports a single index, where data is sorted based on the primary row key. Accessing data in any other method not involving the row key does involve scanning all rows.

Exercises

Setting up

1. Create a table with a specific set of column families, column qualifiers, and a primary key specification. Populate this table with one billion rows of data.

Basic operations

2. Show examples of adding, deleting and updating data inside an HBase table.

3. Search for data for specific values as well as searching for data using filters, to return a range of values. Try different filters to increase the range of data returned.

4. Show how data can be streamed as output using concepts such as enumerators.

5. Show how data can be streamed into the table, either from a file, or from an operation such as a for loop.

Performance

6. Insert one billion rows and show the difference between fetching a range of data for a specific set of columns, versus fetching the range of data while returning all columns.

7. Populate a table with a large amount of data and set the maximum HFile size to a very small number. Show how very small HFiles increase the read amplification, resulting in the query being slow.

8. Populate the table with a large amount of data, set the maximum HFile size to a small number, and trigger the minor compaction manually. Show how this increases the query performance.

9. Populate a table with a large amount of data and set the maximum HFile size to a very large number. Show how very large HFiles force HBase to use one server node only, resulting in the query being slower than normal.

10. Populate a table with a large amount of data and set the maximum HFile size to a reasonably size number. Show ho this allows all server nodes to query data, allowing the query to run faster.

Indexing

11. Show the difference of returning a data using a query on a primary index, versus not using it at all. Show how the query performance decreases significantly when running a query without the use of a primary index on a very large range of data.

12. Use secondary indexes, if possible, and show how this improves performance.

Crash Recovery

13. Populate data so that it is stored across all nodes and force on server to shutdown. Show how data from those affected regions can still be accessed.

14. Disable the use of the WAL file and force one server to shut down. Show how data is recoverable if the memStore is flushed beforehand, versus when flushing is disabled and data was not flushed before the crash.

15. After a crash, insert more data, put one server back online and query the data again. Show how such queries are slower than usually, and how their performance increases if collocality of the regions and servers is fixed through a major compaction.

Online References

Overview

- https://www.slideshare.net/xefyr/h-base-for-architectspptx
- https://hortonworks.com/apache/hbase/

Architecture

- https://mapr.com/blog/in-depth-look-hbase-architecture/
- https://www.edureka.co/blog/hbase-architecture/
- http://www.hdfstutorial.com/blog/hbase-architecture/
- http://www.guru99.com/hbase-architecture-data-flow-usecases.html
- https://www.dezyre.com/article/overview-of-hbase-architecture-and-its-components/295
- https://www.tutorialspoint.com/hbase/hbase_architecture.htm
- https://sreejithrpillai.wordpress.com/2015/01/04/hbase-architecture/

Indexing

- https://phoenix.apache.org/secondary_indexing.html

Scaling

- http://blog.cloudera.com/blog/2013/04/how-scaling-really-works-in-apache-hbase/
- http://hbase.apache.org/0.94/book/regions.arch.html

Questions

- http://stackoverflow.com/questions/39496222/are-all-the-data-with-the-same-row-key-stored-in-the-same-node