Data 650 – Fall 2019

Assignment 1 – HBase

Kenneth Lulie

Professor Purushothman

University of Maryland University College

September 22, 2019

**Assignment 1: Part One**

1. **Discuss the differences between HBase and relational database tables. When would you use each?**

HBase and Relational Database Management Systems (or “RDBMS”), are both concepts for processing, retrieving and storing data (Zarantech, 2019).

RDBMS store data in a predefined structured format of rows and columns. RDBMS fulfill ACID (Atomicity, Consistency, Isolation, Durability) criteria which makes it suited for translation processing and data warehousing. For example, if you are making bank transactions, you would want to guarantee that either the entire transaction was processed or nothing was (Atomicity), that the database is consistent before and after the transaction (Consistency), that multiple transactions can happen without interference (Isolation) and that once the transaction is processed it will stay processed in non-volatile memory (Durability). RDBMS are typically interacted with using SQL. However, scaling is difficult and requires expensive hardware.

HBase is a column-oriented non-relational database management system that runs on top of Hadoop Distributed File System (IBM, 2019). HBase is a NO SQL system, as opposed to an SQL only system. HBase is designed to scale gracefully using in-expensive hardware and, unlike RDBMS, is designed to handle unlimited types and volumes of data (Zarantech, 2019).

However, HBase does not fulfill all of the criteria of ACID. Instead, the CAP data theorem states a distributed database cannot simultaneously be Consistent, Available, and Partition tolerant. Consistency is when a query submitted after a write event always returns the most recent value. Available is when every request to an available node responds to the request. Partition tolerance is that any arbitrary number of messages between nodes can be dropped. HBase is designed to be tolerant and available, but only strongly-consistent.

The major difference between HBase and RDBMS is that they are designed to solve two different issues. RDBMS makes sacrifices in speed and scalability to be ACID compliant making it suitable for purposes when data absolutely needs to be processed correctly every time, such as transactions. The set schemas of RDBMS also make it better suited for complex queries.

HBase has better response time and scalability than RDBMS, as well as an ability to handle data without setting time consuming schemas and data types. Therefore, HBase is best used when there is a large amount of data, ACID properties are required but not absolutely mandatory for every transaction, applications require scaling, and the data model is sparse.

RDBMS would be chosen when many transactions are done using the data store, such as for Online Transactional Processing. HBase is better suited for large scale analytics and bulk storing of data, such as Online Analytical Processing. For example, RDBMS would be used to handle store level retail transactions, whereas HBase would be better suited to storing and delivering Tweets.

<https://www.ibm.com/analytics/hadoop/hbase>

1. **Discuss the differences between HBase and Hive. When would you use each?**

HBase is a NoSQL database used for real-time data streaming, whereas Hive is not a database at (dezyre, 2016). Hive is a map-reduce SQL based engine that runs with Hadoop. Both are used as data stores for unstructured data.

Hive is best used for data warehousing when programmers/users do not want to write complicated map-reduce code, as they are able to access the data they need by writing SQL-like queries.

HBase is used to provide random access capabilities to Apache Hadoop. It provides scalability, cost effectiveness and automatic sharding, while also working well with sparse data.

Hive would be used if you needed to map HDFS files to Hive tables then query data. For example, Hive would be used to provide a company’s business analysts access to the data for further analysis.

HBase would be used if you have a big data solution need for applications requiring random read or write operations. For example, HBase would be used when storing and accessing massive databases for a large company, such as Google.

1. **What are the differences between the delete and drop HBase Shell commands?**

The Delete command in HBase shell is used when there is a need to delete cell values at specific rows or columns of a defined table. It also suppresses older versions of values (IBM, 2019).

The Drop command in HBase shell drops a table, which constitutes not only deleting the values of all rows and columns within the table, but the associated HDFS directory and data files for the table as well. It is important to note that, in HBase, in order to drop a table you must use the Disable command on the table first.

For example, if you had a table ‘Customers 1990-2000’ in HBase, and were directed to remove all addresses which are dated before 1995, you would use a delete command to target the cells fitting the parameters. If you were directed that the table was no longer needed at all, you would disable the table and then drop it.

1. **What are the differences between get and scan commands?**

The Get HBase shell command is used to retrieve the content of a row or cell as defined by the parameters in the command. The Scan HBase shell command retrieves zero or more rows of a table (IBM, 2019).

For example, if you had a customer database and wanted to retrieve the birthday of a specific person, you could use a Get command to only retrieve that cell data. If you wanted to retrieve the entire row of all customers who had a birthday in August, you could use a Scan command to retrieve that data.

1. **When would you use the deleteall HBase Shell command?**

The Deleteall Hbase Shell command deletes all cells in a defined row, removing the row from the database (IBM, 2019). For example, this command would be used if you wanted to remove all data pertaining to specific rows in a database, such as removing all customers who were last active before a give ndate.

1. **When would you use the drop\_all HBase Shell command?**

The drop\_all Hbase Shell command drops all tables meeting the given regular expression passed to the command. It is important to note that these tables must be disabled first (IBM, 2019). The disable\_all Hbase Shell command can be used for this purpose if given the same regular expression.

You would need to use the drop\_all command if you had a regex to identify multiple tables that needed to be dropped. For example, a school had multiple tables pertaining to a class of students, such as ‘class2019\_Attendence’,’class2019\_rost’,’class2019\_schedule’, and needed to remove all data pertaining to that class. You could first run a ‘disable\_all “class2019%” ‘command followed by a ‘drop\_all “class2019%”, which first disable all tables starting with ‘class2019’, followed by dropping all tables starting with ‘class2019’.

1. **What is data versioning in HBase and how is it implemented? What are the pros and cons of versioning?**

Data versioning in HBase is done by storing multiple values of rows based on the timestamp of each row (Apache, 2019). The maximum number of row versions to store is configured per column family using the HColumnDescriptor. If the number of versions is higher than the parameter the oldest versions beyond the number to be kept will be removed.

The pros to versioning is that it lets you have access to older versions of data, allowing you to investigate the history of the data as well as having access to revert to an older value if newer data was added erroneously. The con to versioning is that you can drastically increase the amount of data you are storing and managing.

1. **What is the row key in HBase?**

A Row key is the identifier for a row of data in HBase (Apache, 2019). It is important to carefully consider the row key naming schema so that searches and sorting can be optimized. Row keys are sorted alphabetically, so naming related rows that will be searched together is encouraged. However, to avoid hotspotting, or when client traffic is directed at a specific node, you may consider salting. In this case, salting would be adding random data to the start of each row key to ensure storage of rows across multiple clusters.

1. **Define the column family. Is it possible to create an HBase table without any column families? Why or why not?**

Columns in Hbase are grouped into Column families, and all column members of a column family will have the same prefix (IBM, 2019). For example, in the column family ‘student’ you could have ‘student:age’ and ‘student:class’ as two columns within the same column family. Column families must be declared at the schema creation but you add columns within column families after creation. Additionally, column family members are stored physically together on the file system, so column families should contain columns which are generally similar to each other in characteristics or expected to be queried together.

Since each column must belong in a column family, it is not possible to have an HBase table without a column family. Without columns, the data would just be consisting of row keys with no data to point to

**Using HBase for Retail Bookstore Data Management**

**Introduction**

Since the first introduction of computer storage, the amount of data captured and retained for further use has increased exponentially. When computers were first introduced, it took hardware the size of rooms to handle megabytes, but Moore’s law has reduced the cost of storage and computing dramatically enabling the creation and storage of over 2.7 Zettabytes in the world today (Node, 2019). With the ever decreasing expense of capturing, retaining, and analyzing data, uses that were once uneconomical are now essential to the business model of the biggest companies.

Relational Database Management Systems were originally the main economical way of storing data, and only the most important data points were captured. Today, Hadoop and distributed file systems take advantage of cheap hardware to revolutionize the way we store and use data. Today, companies capture data that wouldn’t have been captured even a decade ago, and use data science to extract value from that data.

In this exercise, appropriateness and implementation of Hbase, a column-oriented non-relational database management, will be reviewed for use in managing the data needs of an example company. Specifically, a Hbase database will be designed as well in addition to creating pseudocode suitable for translating existing records to be ingested into the database.

**Problem Statement**

**Initial Creation of Data Management for Bookstore Inventory**

When designing a database solution, one of the first questions to be answered is if the solution should use a RDBMS or a distributed file solution such as Hadoop (Apache, 2019). Generally, this question can be answered by asking if there is an absolute need for the database to fulfill the properties of the ACID data theorem, a data theorem that states systems fulfilling its criteria must have Atomicity, Consistency, Isolation, and Durability at all times (IBM, 2019), and how scalable the data system must be. For this purpose, the database system will consist of records of books and so the consequences of rare ACID compliance failures will not have a large undue effect. Further, hundreds of millions of records may need to be stored implying the need for large scalability. Therefore, Hbase on top of the Hadoop File System or HDFS was selected as the solution for this business need.

Unlike RDBMS, HBase is not ACID compliant, and instead looks to the CAP data theorem that states for distributed data systems not be Consistent, Available, and Partition tolerant but can only have two of these properties. HBase is considered Available and Partition Tolerant, but only meets the standards for Strong Consistency rather than full consistency (Apache, 2019). While a complete ACID solution would be generally desirable, the necessity of scaling and storing large amounts of data dictates that distributed storage even with the compromises inherent to that solution are required.

The database designed will need to intake millions of files stored on ASCII based text files with various storage devices including CDs, DVDs, Optical Disks, and high volume hard disks. Additionally, two sample records and metadata for the records to be ingested has been given (Appendix 1). Of specific note in importance is that several data points will have an unknown number of entries per book. Specifically, a book record may have multiple sub-topics, authors, author affiliations, and ISBN numbers. This would be difficult for a RDBMS to handle as the number of columns per record must be defined at the start. However in HBase it is possible to add columns to a column family after creation.

**Design**

**HBase Solution and Implementation**

The data solution for this problem will be implemented using HDFS and HBase. It is assumed that all records will be provided in the same format as the original two records provided, and that a Hadoop ecosystem will be provided to support the database. It is assumed that the goal is to ingest all records, and that duplicative records will be dealt with at a later date.

The database schema is provided below in **Figures 1, 2, 3 & 4**. The row key for each row will be a combination of the title, the month and year of the reprint edition, and a version number calculated by scanning the database for the number of records with the same title and edition number combination and then incrementing by one. If no reprint edition information is found the letter ‘U’ for unknown will be used in the row-key instead. Four Column families will be created. It is assumed there is a maximum of one reprint edition per month.

The first column family will be named ‘Title’, and will consist of the Title, the Topic, and the subtopics. Each subtopic will receive its own column with the schema of incrementing the id in the column name for each new sub-topic to accommodate an unknown number of sub-topics. It is assumed that each book will only have one title and topic.

Figure 1 - Column Family: Title



The second column family will be named ‘Series’ and contain all information pertaining to the series of the title. This family will consist of Series, which will be populated as Yes or the entire column family be null, the series title, and the series number. It is assumed that each record will only ever have one record for each of these columns. Additionally, if the Series column is Yes, the series title and series number must be populated.

Figure 2 - Column Family: Series



The third column family will be named ‘Author’ and consist of all information pertaining to the authors of the book. It will contain an Author column that will increment by one in the column name to accommodate an unknown number of authors. The value of each author column cell will consist of the author number in the original record, followed by a delimiter, followed by the author information. The same approach will be used for the Author Affiliation columns with the exception that the original number of the author affiliation will not be retained.I Instead, it will only capture the value of the author in the original record followed by a delimiter followed by the value. This schema will assist queries to connect author affiliations to authors.

Figure 3 - Column Family: Author



The fourth and final column family will be ‘Publisher&Misc’ and consist of the Publisher column, the Reprint Edition Column, the Language Column, the Binding Column, the ISBN column which will increment to allow multiple values, and the product dimensions column. It is important to note that each row should have a unique Title and Reprint Edition combination.

Figure 4 - Column Family: Publisher&Misc.



To optimize the speed of the querying tool, hashing the row key could assist in the speed. Additionally, by grouping the data in logical column families the speed of queries will be increased as the data will be in the same physical place. For example, information about the author, the series, and title/genres are some common queries to be expected. By having each of this data in separate column families we can assist in speeding up queries. Further, as HBase sorts data alphabetically, having the title as the first part of the row key will ensure some randomness to presumably avoid hotspotting. Hashing the title will have the additional benefit of ensuring a random distribution of the records to further help avoid hotspotting.

**Implementation Methods**

**Pseudocode for Ingesting Records**

Each record will need to be read, parsed, and transformed into input data for database ingestion. Based on the examples, it is assumed that each record will be delivered on its own, with consistency in naming. Further, the data will be always be delivered in a comma delimited format to separate data with the format of key, semicolon and space, value format. Please see **Appendix 2** for the pseudocode to convert the data.

**Conclusion**

This analysis showed a method for converting records from disparate and separate sources into one database that can be scaled to provide a single solution for a database need. However, there are more improvements to increase the query speed that could be done that are out of scope of this assignment. One way to improve the query speed would be to simply improve the number of clusters and the replication factor. If the data is available on more clusters the query would have more resources to use resulting in faster speeds. Another method would be to run a check to delete duplicate records or records that are functionally duplicate. While deleting duplicate records would be a fairly simple task to accomplish, identifying functionally duplicate records may require more analysis. A third factor would be to increase the quality of the clusters such as increasing the memory available. A final factor would be to optimize the Hadoop ecosystem Hbase is operating in.

**References**

Team, A. H. B. (n.d.). Retrieved from <https://hbase.apache.org/book.html#schema.versions>

Hive vs.HBase–Different Technologies that work Better Together. (Dec 7, 2016). Retrieved from <https://www.dezyre.com/article/hive-vs-hbase-different-technologies-that-work-better-together/322>

What is HBase? (n.d.). Retrieved from https://www.ibm.com/analytics/hadoop/hbase

Big Data facts - How much data is out there? (2019, March 1). Retrieved from https://www.nodegraph.se/big-data-facts/

Difference Between RDBMS and HBase (n.d.). Retrieved from <https://www.zarantech.com/blog/difference-between-rdbms-and-hbase/>

**Appendix 1**

**Sample Records**

Record 1:

Topic: Fiction, Sub-topic: Young-adult, Sub-topic: Mystery, Sub-topic: Adventure, Title: A study in Charlotte, Series: Yes, Series Title: Charlotte Holmes Series, Series Number: 1, Author 1: Brittany Cavallaro, Author 2:, Author 4:, Publisher: Katherine Tegen Books; Reprint edition: January 3, 2017, Language: English, Paperback: 352 pages, ISBN-10: 0062398911, ISBN-13: 978-0062398918, Product Dimensions: 5.3 x 0.8 x 8 inches

Topic, Sub Topic, Title, (Does it have a series)?, Series

Record 2:

Topic: Non-Fiction, Sub-topic: Adult, , Sub-topic: Young-Adult, Sub-topic: Self Help, Sub-topic: Professional training, Sub-topic: Comedy, Title: How to be a comedian, Series: no, Author 1: George Carlin, Author 2:David Letterman, Author 3: Jay Leno, Author 4: Conan O’Brian, Author 5: Jimmy Kimmel, Author 5: George Lopez, Author 6: Eddie Murphy, Author 7: Ellen Degeneres, Author 8: Richard Pryor, Author Affiliation 1: (Author 2, 12 Second Street, NY 10005), Author 12: Margaret Cho, Author Affiliation 3: (Author 5, 91, Hollywood Street, Los Angeles, CA 90100), Publisher: McGraw Hill Academic Press; Reprint edition: 8, Language: English, Paperback: 666 pages, ISBN-10: 0087738911, ISBN-13: 942-006711918

**Appendix 2**

**Pseudocode for Converting Records to HBase input.**

#Import record, assume function returns as one long string.

Import Record

#Import dictionary of column names that map to schema such as “Author – AuthorID”

Import ColumnDictionary

#Scan for dates in the format and convert to MM-DD-YYYY format before using ‘,’ as delimeter.

For each Date in Record, convert (MM-DD-YYYY)

#Set delimiter to ‘,’

Csv\_reader = csv.reader(record, delimeter = ‘,’)

#Read record with comma delimiter, assume each chunk is returned as a separate row.

Read Record with CSV\_reader

#Create dictionary called Processed to store output

Processed = [,]

#For each row, find the semicolon and use the characters before it as the key, and the characters after it as the Value, accounting for the space.

For each row in csv\_reader,

Separator = Find(“:”)

Key = row[0:Separator]

Value = row[(Seperator+1):]

#if the key is in our list of columns, convert it to the intermediate stage of the column name.

If Key is in ColumnDictionary

Key = ColumnDictionary(Value)

Add [Key,Value] to Processed

#Import a list of columns allowed to have multiple values such as number

Import ColumnNest

X = 1

#Find the number of times a key shows up in the dictionary. This will let us find potential errors and also let us create the right number of columns.

Also, exit and log if we have more records than expected, such as two titles.

#Also do a for loop to properly increment the number of column ids such as for author.

For each Unique Key in Dictionary

X = Count of Key in Dictionary

If X > 1 and Key not in ColumnNest

Log(Error, multiple values when unexpected)

exit()

Else For Each Key in Dictionary

If Key in Dictionary

KeyOutput = Key+I

I = I +1

#Set rowkey to the value of the title and the reprint edition

Rowkey = Dictionary(Title) + Dictionary(ReprintEdition)

#Import function to convert the rowkey and dictionary into HBase format

Import HbaseConverter

HBaseOutput = HbaseConvert(Rowkey,Dictionary)

###convert HbaseOutput to a hbase ingestible format

Import ReadHbase

## Write new records to book table

New Records = ReadHbase(HBaseOutput)

Add(NewRecords,Mytable)