Assignment 1: Hbase

Melissa Hunfalvay

Email: [Melissa.Hunfalvay@gmail.com](mailto:Melissa.Hunfalvay@gmail.com)

Data 650 9040

Summer 2022

Professor Ozan Ozcan

Date: May 31st, 2022

**Part 1**

1. *Discuss the key components of HBase and the function of each component.*

Figure 1 shows the key components of HBase.

Diagram

Description automatically generated

Figure 1: HBase architecture (Apache HBase Tutorial, 2014).

1. HBase is a columnar key-value store that stores data as “wide” tables (unnormalized or de-normalized). These files are called **Hfiles** and allow for large volume and flexible storage capacity.
2. **Write Ahead Logs (WAL)** provide built in recovery through the process of HMaster distributing data which is written in a chronological order with a timestamp in the WAL. Re-executing WAL means changes that were stored in the MemStore file can be recovered.
3. **MemStore** stores all incoming data before moving it to permanent storage. Data is stored in lexicographical order sorted by Key Values.
4. The **HMaster** distributes and allocates data for load balancing and, should a hardware failure occur, the HMaster allocates a new **region server,** Figure 2**).** Enabling both recovery and scalability when needed.

Graphical user interface, diagram

Description automatically generated

Figure 2: Scalability of HBase architecture as HMaster allocates a new region server ([Simple Learn, 2022)](https://www.youtube.com/watch?v=V1fXSCASVDc&t=532s)

1. **API** is the connection tool that allows read and write functions to Hbase.
2. **Zookeeper** not shown in figure 1 but is a recommended add on tool as it notifies HMaster about any such hardware failures. This allows for distributed synchronization across the platform.
3. *Discuss the differences between row-oriented and column-oriented databases. Provide examples of each.*

Row-oriented databases store values associated with the rows together. For instance, the example below would be stored together as one row in a row-oriented database.

Member ID: 2011078; Member Name: Ozcan Ozan; Member since: 2011; Exercise: Exercise: Bicep Curl, Weight: 25; Exercise: Triceps Deeps, Weight: Body Weight; Exercise: Plank, Weight: Body Weight; Trainer 1: Heather Smith; Trainer 2: Barbara Eden; Trainer 3: James Martin

In a column-oriented database, the values in the column are stored together. For instance, bicep curl, 15, 25, 25 (for the dataset in question 9).

Row-oriented databases are designed for smaller tables and Online Transaction Process (OLTP). Column-oriented databases are designed for very large tables and is a suitable for Online Analytical Processing (OLAP).

1. *When would you use Hive instead of HBase?*

The primary purpose of HBase is for real-time querying of data and data storage. Data can be of any type in HBase. HBases is no-SQL key/value database on Hadoop. HBase is used for fast transactional processing (low latency) and online analytical processing (OLAP) operations ([HevoData, 2021](https://hevodata.com/learn/hive-hbase-comparison/" \l ":~:text=Hive%20is%20a%20SQL%2Dlike,key%2Fvalue%20database%20on%20Hadoop.&text=Hive%20is%20mainly%20used%20for,a%20large%20volume%20of%20data).)

Hive is not suitable for real-time querying of data and is used for analytical querying. Hive is a SQL like querying tool uses MapReduce on Hadoop. Hive is used for batch processing and not online analytical processing (OLAP) operations. Due to the batch processing, there is usually longer latency times for Hive processes.

1. *Discuss 7 HBase shell data manipulation commands and what they do*
2. List = see a list of the tables within the database
3. Create = create a new table
4. Get = retrieves data e.g., get ‘table.name’
5. Scan = iterate through a table to see what it contains e.g., scan ‘table.name’
6. Put = inserts data into a table e.g., put ‘table.name’, ‘row1,’ ‘cf:column.name’, ‘value’
7. Version = gives you the version of the table you are currently viewing
8. Whoami: is the command that give the Hbase user information from the HBase cluster
9. Disable: this command starts to disable a table, if a table needs to be deleted or dropped it needs to be disabled first.
10. *How would you implement a query that joins multiple tables in HBase?*

An efficient method for joining tables in Hbase is using the sort-merge join. In this method, the first table is scanned and output with a unique key will be used to match the next table. For example, table 1 unique key is ID, table 2 unique key is ID, and table 3 is also ID. Therefore, this key is used to join multiple tables in Hbase.

1. *What is a namespace in HBase?*

Namespace is used to logically group tables in HBase. An analogous concept may be the use of a last name versus first name. Last name can be used for an entire family e.g., my last name is Hunfalvay, same as my brothers, my sisters, my parents. First name is unique. We would group the family based on last name, in this case Hunfalvay. A new family could be Smith, which would be grouped in a new namespace. The last name is a logical grouping of a family and therefore the namespace.

1. *What happens when you delete table cell(s) in HBase?*

The delete command will delete a cell value for the specific coordinates (row-column) within the table. Then, when using the scan command, the deleted cell suppresses older versions of values. The data is not actually deleted. Instead, a Tombstone marker is added which effectively makes the data invisible. Sample syntax is delete ‘table.name’, ‘row.value’, ‘column.name’

1. *Discuss the approaches for storing multimedia data, including videos and images, in HBase.*

HBase is a column-oriented storage database. The row key is unique to each row. The columns are divided into column families and then column qualifiers. Multiple column qualifiers maybe within column families. The intersection of rows and columns are cells where the data is located. Row-oriented databases allow for some cells to remain sparce or blank and others to be populated with data.

Data is stored based on the row. Within the Hadoop cluster is a region H log. The H log is a memory store for the different file types. These are separated across different computers in the HDFS storage system (see Figure 2).

Graphical user interface, application, Word

Description automatically generated

Figure 3: HBase architecture regional servers, tables, and storage

As previously mentioned, HBase tables are divided horizontally by row key range. These ranges are divided by region. Regions are assigned to nodes within a specific cluster which is called a region server. Regions contain all rows in a table between the start and end key.

Data is chunked into blocks, usually of 256MB and replicated across the clusters.

The tables are stored as key-value pairs and stored in memory. As the memory buffer fills up it is spilled into the disc. The function of the combiner is to add up the values and combine values together. This then spills the data from the buffer into the disk. There is not enough space in the buffer to hold all the results, so the files are stored on the disk. Specifically, if there are 100 mappers and 50 reducers there would be 5,000 small files that have to be stored in the disk. They then need to be shuffled over the network to reach the corresponding reducer. The disadvantage of this method is that with large volumes of data and many, many files this process can take a long time.

1. *The short pseudocode example concludes with discussion of conceptual view after parsing 2 sample records. Show the conceptual view after parsing the following third sample record. Add the table with 3rd record.*

Member ID: 2011078; Member Name: Ozcan Ozan; Member since: 2011; Exercise: Exercise: Bicep Curl, Weight: 25; Exercise: Triceps Deeps, Weight: Body Weight; Exercise: Plank, Weight: Body Weight; Trainer 1: Heather Smith; Trainer 2: Barbara Eden; Trainer 3: James Martin

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Member | | | Weight | | | | | | | |  | Trainer | |  |
| Key | Member Id | name | Member since | squat | Bicep curl | pushup | deadlift | Triceps extinction | Triceps deeps | Dead row | lunges | plank | Trainer 1 | Trainer 2 | Trainer 3 |
| 1 | 2000055 | Yelena Bytenskaya | 2000 | 40 | 15 | Body weight | 30 | 20 |  |  |  |  |  |  |  |
| 2 | 2010022 | Dave Linesh | 2010 | 70 | 25 |  |  |  | Body weight | 45 | 35 |  | Jim Smith | Barbara Eden |  |
| 3 | 2011078 | Ozcan Ozan | 2011 |  | 25 |  |  |  | Body Weight |  |  | Body Weight | Heather Smith | Barbara Eden | James Martin |

**Part 2**

**Introduction and Problem Statement**

UMGC desires to move the academic records system from disparate sources to a unified, cloud-based location where the data can be easily accessed, queried, and understood across all academic records.

Currently academic records are stored locally and in multiple locations such as Ascii-based text files, hard disks, optical discs, CD’s, and DVD’s. A limitation with the current method includes a lack of access to academic records across the multiple storage devices.

The volume and velocity of data is growing exponentially and the lack of access across records is increasingly problematic as the university is unable to glean important global trends and insights. For example, it is impossible to determine basic insights such as registration levels. Pass/fail rates of classes or classes taught by specific teachers is not easily accomplished in the current state of storage and without a cloud-based architecture.

Practical and time-sensitive limitations include the possibility of a loss of storage devices, and a compounding problem of needing more storage devices, as these devices are being phased out of the market.

Given these limitations, UMGC desires the following from the academic records:

1. Access to all records at anytime
2. Records to be saved and not lost. For instance, the need for redundancy should there be hardware failures.
3. An ability to query any parameter in the database in a simple manner
4. A flexible file system that allows for changes, additions, deletions, and any other modifications to the academic records
5. The ability to easily scale as data grows (without having to rebuild the entire structure)

The recommended solution is the Hadoop framework, which is a cloud-based, distributed, scalable big data store with Hbase real time real/write access to persistent (saved) academic records stored in the database (<https://hbase.apache.org/> ).

**Method and Implementation**

**Design**

To accomplish the goals, and overcome the problems currently faced by UMGC, the Big Data Systems used will include HBase architecture built on top of the Hadoop Distributed File System (HDFS; Figure 1).

Graphical user interface, diagram

Description automatically generated

Figure 2: Scalability of HBase architecture as HMaster allocates a new region server ([Simple Learn, 2022)](https://www.youtube.com/watch?v=V1fXSCASVDc&t=532s)

The Hadoop framework is a cloud-based, data storage for any type of data. It can scale via the Hadoop Distributed File System (HDFS) and has large processing power. Hadoop is used by many large data companies today such as Uber, Airbnb, Netflix, and Twitter (<https://stackshare.io/hadoop>).

The HDFS system uses MapReduce to process the data. Map and Reduce are two separate processes. MapReduce was created by Google to solve the bottleneck issue when using a centralized system to process multiple files in a simultaneous manner (<http://www.tutorialspoint.com/map_reduce/map_reduce_introduction.htm> ).

Using MapReduce, the data is first input then split into key-value pairs. Then, the Map phase is used to take the key-value pairs and process them to generate zero or more key-value pairs, known as intermediate keys. These are then shuffled and sorted by the reducer which in turn runs a reducer function on them. MapReduce on Hbase integrates easily with Hadoop (<https://hbase.apache.org/> ).

Hbase is a database built on top of the Hadoop (HDFS) file system and provides the following capabilities:

1. Clients can read and write to this platform. Hbase uses Java API to communicate. This is a fast, well utilized, read, and write capability for clients.
2. HBase is a columnar key-value store that stores data as “wide” tables (unnormalized or de-normalized). These files are called Hfiles and allow for large volume and flexible storage capacity.
3. Write Ahead Logs (WAL) provide built in recovery through the process of HMaster distributing data which is written in a chronological order with a timestamp in the WAL. Re-executing WAL means changes that were stored in the MemStore file can be recovered.
4. MemStore stores all incoming data before moving it to permanent storage. Data is stored in lexicographical order sorted by Key Values.
5. A timestamp is added to each data input allowing for easier searchability and data versioning.
6. The HMaster distributes and allocates data for load balancing and, should a hardware failure occur, the HMaster allocates a new region server. Enabling both recovery and scalability when needed.
7. Zookeeper is a tool that notifies HMaster about any such hardware failures. This allows for distributed synchronization across the platform.

In summary, Hbase (on top of HDFS) is consistent when write records are returned. This means that people reading the records will all see the same data. As the academic records grow larger the system will scale automatically as the regions split. Furthermore, so data is not lost, the HDFS will replicate and spread. HBase provides fast, random-access of the UMGC academic records.

**Hbase Data Model**

The data model will follow that of the original Google Bigtable (Chang, Dean, Ghemawat, et al., 2006) as a sparse, persistent, multidimensional map. It will be indexed by a row key and a column key and a timestamp for versioning (Khurana, 2012).

Column families group data based on logical grouping concepts such as contact information or degree information. For the UMGC academic records the families are: Individual, Degree and Instructors (Figure 2). Column families must be defined up front and are not easily modified.

Column qualifiers are vertical data within the column families (Figure 2). Column qualifiers do not need to be specified in advance, do not need to be consistent between rows and do not need to have a specific data type. This makes them flexible for UMGC as the data may expand and change over time.

A picture containing timeline

Description automatically generated

Graphical user interface

Description automatically generated with medium confidence

Figure 2: HBase table model showing column families and columns for UMGC academic records

Rows are horizontal and data is stored according to its row. Each row is identified by a unique row key (Figure 3).

Cells are found at the intersection of rows and columns (Figure 3). Cells are versioned based on their timestamp. The default number of cell versions is three. The default will remain in place for UMGC as it is deemed likely that more than three versions for any one column is unlikely. Should this need to be updated in the future it is an easy parameter modification. Chart

Description automatically generated with low confidence

Figure 3: Partial table design for record 1 illustrating cells, keys, families, column families and row (see Hunfalvay\_HBase.xls for complete record). Note: The additional timestamp column is for illustrative purposes only.

Three sample UMGC academic records are as follows:

**Record 1**

Program: Information Technology, Specialization: Database Systems, Course: DBST651 Grade: A, Course: ITEC630 Grade: B, Course: DBST667 Grade: A, instructor 1: James Smith (DBST651), instructor 2: Jennifer Lopez (DBST651), instructor 3: Jennifer Lopez (ITEC630), instructor 4: Catharine Murphy (DBST667), student name: Yelena Bytenskaya, EmplD: 123456 , User Name : ybytensk, instructor id: 234567, graduated: Yes

**Record 2**

Program: Data Analytics, Course: DATA610 Grade: B, Course: DATA620 Grade: A, Course: DATA630 Grade: C, Course: DATA630 Grade: A, Course: DATA640 Grade: B, Course: DATA650 Grade: A, Course: DATA670 Grade: B, instructor 1: Steve Knode (DATA610), instructor2: Caroline Beam (DATA620), instructor 3: Bati Firdu (DATA630), instructor 4: Elena Gortcheva (DATA650) , instructor 5: Ozan Ozcan (DATA650) instructor 7: Jon McKeeby (DATA670), instructor 8: Steve Knode (DATA670), instructor 9: Steve Knode (DATA640), instructor 10: TA Yelena Bytenskaya (DATA650), student name: Linesh Dave, EmlID:567890, user name: ldave, instructor id: 567907, graduated: Yes

**Record 3**

Program: Information Technology, Specialization: Database Systems, Specialization: Project Management, Specialization: Software Engineering, course: DBST651 grade: F, course: DBST651 grade: B, course: ITEC610 grade: B, course: ITEC620 grade: A , course PMAN634 grade: C, instructor 1: Brandon Morris (ITEC 610), instructor 2: Elena Gortcheva (ITEC620), Instructor 3: James Green (DBST651), Instructor4: TA Yelena Bytenskaya (DBST651), student name: Jeff Martin, emplID: 987654, user name: jmartin, graduated: No

Pseudocode for these records is found in the attached txt file. Figure 4 shows a portion of the .txt file and explains each row. This pseudocode reads in each record and parses it into HBase table. Prior to the .txt file a table would have been created using the command “create” to create a table called “academic record.” After completing the put command we would “scan” the contents of the table to check the content.

Text

Description automatically generated

Cell Values

Column Families

Row Key

Figure 4: Portion of the pseudocode .txt file explained

A portion of the Hbase table for the three records are in Figure 5, the entire table is in the attached excel file (Hunfalvay\_HBase\_Tables.xls). Figure 5 shows the ability to scale and the sparce nature of HBase tables. It also shows how records are updated based on the timestamps. This is not in the data but is illustrative to show how records are added and versioned. For Jeff Martin the grade received on the last attempt (Grade = B) for DBST651 overwrites the grade received on the prior attempt (Grade = F). Both records are kept for academic advising purposes.

Graphical user interface, table

Description automatically generated

Sparce & scalable Hbase table

Figure 5: Sparce and scalable HBase tables with an illustration of the timestamp update for Jeff Martin’s grades

**Design Verification**

To effectively examine this table design, the input data must first be reviewed. Currently, there is historic data, stored in multiple locations such as Ascii-based text files, hard disks, optical discs, CD’s, and DVD’s. Changing to this cloud-based format will enable the client to access this data. However, without explicit dates for when the events occurred, for example, when a student took a class, or an instructor taught a class, then the associated timestamp will be the write file input timestamp. For historic data this will not match the actual date the class was taken. For future data, it may or may not match depending on when the write record was added to the database.

This is important as without the time relevant dates such as semester and year, results obtained for certain types of queries will be inaccurate as the data is not available. For example, if the client wanted to query the historic data to find out which semester across all the years that most students failed, they could not do that with the current information.

As previously specified, to resolve this issue, UMGC could either, add data at the beginning of each semester and therefore have an accurate timestamp. Or, add a semester and year specifically in the table as a column qualifier. My recommendation would be the later, as explicitly adding the semester and year will ensure the information is correct in case those entering the data are delayed for whatever reason. Adding the information in the form of a column qualifier will also allow the client to query specifically on that column qualifier.

All individuals, whether students or instructors have their information in the “Individual” column family. The advantage of this is that there is only one place to update the information such as name and ID. If a student was a student and not an instructor, then the table for the instructor information remains blank (sparse) and the table scales effectively.

A possible disadvantage of having all the individual information in the same family is that the client needs to know to logically look for all administrative type information, whether student *or* instructor in the same location. The instructors serve different roles from the students however and this is something to be mindful of when creating the table.

A decision was made to leave the students and instructors in the same family and use a generic family name “Individual” instead of “Student” or “Instructor”. In this limited (3 records) given as examples, two students who graduated went on to become instructors. If that trend continues, then having the student and instructor information in the same family makes logical sense. Furthermore, storing redundant information in multiple places increases storage, requires more administration during the write and read process. However, if many instructors are added who were not students, this may be a reason to split the Individual family into two distinct families that is “Students” and “Instructors.”

To account for programs that offer specializations and where students may do any number of specializations, or no specializations, or add new specializations, the specialization type is the column qualifier (e.g., database systems, project management). This enables the client to query the number of students in each program specialization. However, the disadvantage is that the client will need to query each individual specialization to determine if a specific student has chosen that specialization. In other words, if I want to know which specializations Jeff Martin has chosen then I would need to query each column qualifier to obtain an answer. As the number of possible specializations grows this challenge becomes more pronounced and time consuming. A filter can be used to return the column qualifiers where values exist. This is helpful but still requires each column to be specifically queried and with many columns this could be increasingly more time consuming.

Should a new specialization be added, for example to the Information Technology program and Jeff Martin (record 3) wanted to add this specialization, then a new column qualifier would be added with the specialization name and the cell would be populated with “Yes” indicating Jeff Martin has selected to be enrolled in the new specialization. This is an advantage of the Hbase table style as scalability is a simple process.

To account for a student taking multiple classes and each class having a specific grade the column name is the class and the grade is within the class. The advantage of this table structure is the client can query based on a specific class identifier e.g., DATA620 and obtain a grade if the student took that class. A disadvantage of this is similar to that of the specializations, in that, if the client wants to know all the classes a student took, they will need to query each specific course and if there are many courses this could be time consuming. A filter can be used to return the column qualifiers where values exist. This is helpful but still requires each column to be specifically queried and with many columns this could be increasingly more time consuming.

To account for multiple instructors teaching the same class the course identifier (e.g., DATA620) and instructor number in sequence is a column qualifier. The advantage of this is that you can query the course and the instructor for a specific student. The disadvantage of this is we cannot assume, if the student took the class more than once and had three instructors for example, which instructors taught which class, unless some form of timeline data (e.g., semester and year) is added to the historical data. If semester and year is added to future records this would solve the issue. However, given the current data, it is not possible to query based on year or semester for instructors and the course they taught. Another disadvantage of coupling the course identifier with the instructor number, is the client cannot query instructors by themselves. The client will need to know which instructors taught which courses as they are required to associate the instructor with a course code to return the instructor’s name. Furthermore, if there is more than one instructor for a course in the same row (i.e., student) such as DATA650 for Linesh Dave then the first instructor in the record is given the column qualifier DATA650.Instructor.1 and the second is DATA650.Instructor.2. The disadvantage is the client would need to specifically query each iteration (e.g., Instructor1,2,3) to determine how many instructors taught the course.

If the client wanted to know if a student took multiple classes with the same instructor, they would need to read all the Class.Instructor column qualifiers. The disadvantage of this is as the classes grow this could be a lot of information to scan through.

As courses can be taken in any order and new courses may be added to the table at any time, the Hbase table is set up to add the courses as column qualifiers and expand (horizontally) as needed. Grades can be added to courses that already exist within the table as students complete those courses. This scalability and flexibility are advantages of HBase tables.

**Implementation Methods & Queries**

In summary, the following queries are suitable for the Hbase tables as designed:

* Which classes has each student taken?
* What grade was achieved for a specific class for a specific student?
* What grade did Jeff Martin get the second time he took DBST651?
* Did a student repeat a course?
* Which specializations are students taking?
* Which instructor has taught a specific course e.g. DATA620?
* Which courses have a TA?
* Which students go on to become instructors?

**Discussion and Conclusion**

There are two systems for evaluating database management. Atomic, Consistent, Isolated, Durable transactions (ACID, Databricks, 2022) and Consistent, Available and Partition Tolerant (CAP) methods.

ACID transactions are foundations of relational database (RDMS) interactions. HBase is not a RDMS. In summary, ACID refers to *Atomic* transactions that are all treated as a single action. All components or none of the components are returned and the state of the database remains unchanged. *Consistent* refers to the structure of the request. Rules and restrictions of the database are present, and violation of these parameters return an invalid data state. *Isolated* refers to the concurrent execution of transactions if the request is executed in a serial manner. *Durable* refers to the persistence of a transaction. A transaction will be saved and not undone to accommodate conflicts in operations.

The advantage of ACID transactions is the rules it forces on database management. If a schema declares that a value must be unique, then the consistency of the system will enforce the uniqueness of that specific value across all operations. In a similar manner, if a foreign key deletes one row in the database, it will delete related rows. A consistent system will ensure the state cannot contain related rows once the base row is deleted. Such rules enforce effective data management.

ACID compliant systems guarantee the data will be accurate, valid, and in line with the constraints imposed.

Another advantage of the ACID compliance system is that complex updated operations do not need to be examined in advance. Furthermore, update operations do not need to be planned based on their mutual interaction mechanisms. The database management system will isolate and provide consistency after the commit occurs.

A final advantage of ACID compliance systems is the storage is reliable. In-memory storage often fails, however, having the ability to rely on durable storage removes this operational concern.

The disadvantage of ACID transactions is that they are often slower at read and write operations. Therefore, for higher throughput systems distributed platforms often perform better as they ingest large quantities of data in parallel.

Another disadvantage of ACID transactions is that they are time-consuming and not suited for large volume OnLine Transaction Processing (OLTP) and OnLine Analytic Processing (OLAP). RDBMS database systems which use ACID transactions may be very complex with many joins and table interactions.

The CAP theorem (Consistent, Available and Partition Tolerant; CAP) is a computer science belief about distributed data systems (Brewer, 2012). *Consistency* refers to replication of data and that all data has the same value. *Available* refers to live nodes processing operations and respond to queries within the system. *Partition tolerant* refers to the design of the distributed system to operate when there are connectivity failures.

The advantage of CAP transactions is that there is high consistency and less downtown across the distributed network. Data is highly available with fast and random access of data. CAP consistency ensures that every replica of the same logical value, spread across nodes in a distributed system, always has the same value.

The disadvantage of CAP transactions the option of partition tolerance is not enforced. There is a risk of some data not being available. There is a risk of reading inconsistent data. Only two out of three of the CAP principles can be achieved at one time. Essentially, if there is a system failure in the distributed data system the theorem states that both consistency and availability are diametrically opposed and therefore cannot be achieved at the same time.

HBase chooses Consistency and Partition Tolerance and compromises on Availability in the CAP theorem (<https://www.quora.com/HBase-follows-which-features-of-CAP-theorem>). Furthermore, should the system crash, recovery can be slow and complex. These problems usually get pushed to the developer teams to fix.

WAL replay may be slow in Hbase. Therefore, recommendations for improving speed include logical column clustering. This will be especially important as the column qualifiers grow.

In conclusion, UMGC desires to have all past and future academic records in a unified, cloud-based location where the data can be easily accessed, queried, and understood across all academic records. The recommended solution is the Hadoop framework, which is a cloud-based, distributed, scalable big data store with Hbase real time real/write access to persistent (saved) academic records stored in the database (<https://hbase.apache.org/> ). This system in not ACID compliant but does utilize CAP principles. The tradeoff of rapid availability and scalability of the data is considered acceptable for this use case. This big data platform is a significant, sustainable, and scalable upgrade to the current disparate methods of academic record data used by UMGC.

**References**

Apache (2014). Apache HBase Tutorial. Retrieved on May 24th, 2022, from [www.tutorialspoint.com](http://www.tutorialspoint.com)

Apache Hbase (2021). Apache Hbase Retrieved on May 27th, 2022, from <https://hbase.apache.org/>

Brewer, E. (2012).  CAP twelve years later: How the "rules" have changed. *Computer*, 45(2), 23–29.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R.E. (2006). Bigtable: A Distributed Storage System for Structured Data. *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation* (OSDI ’06), USENIX, 2006, pp. 205–218.

Databricks. (2022). ACID Transactions. Retrieved on May 28th, 2022, from: <https://databricks.com/glossary/acid-transactions>

Khurana, A. (2012). Introduction to HBase Schema Design. *Login*, 37 (5): 29-36.

Meher, E. (2021). Hive and HBase comparison: 7 critical differences. Retrieved on May 27th, 2022 from: [https://hevodata.com](https://hevodata.com/learn/hive-hbase-comparison/#:~:text=Hive%20is%20a%20SQL%2Dlike,key%2Fvalue%20database%20on%20Hadoop.&text=Hive%20is%20mainly%20used%20for,a%20large%20volume%20of%20data)

Quora (2022). HBase fllows which features of the CAP theorem? Retrieved on May 27th, 2022, from: https://www.quora.com/HBase-follows-which-features-of-CAP-theorem

Simple Learn (2022). HBase Tutorial for Beginners. Retrieved on May 28th, 2022, from <https://www.youtube.com/watch?v=V1fXSCASVDc&t=532s>

Stackshare (2020). Hadoop Open-source software for reliable, scalable, distributed computing. Retrieve on May 25th, 2022, from: <https://stackshare.io/hadoop>

Tutorials Point (2022). MapReduce Introduction. Retrieved on May 25th, 2022 from <http://www.tutorialspoint.com/map_reduce/map_reduce_introduction.htm>