Migration from Relational Databases to HBase: A Feasibility Assessment

Zakaria Bousalem[1(✉)], Ilias Cherti[1], and Gansen Zhao[2]

[1] Faculty of Science and Technologies, Hassan 1st University, Settat, Morocco
zakaria.bousalem@gmail.com, iliaschertil0@gmail.com
[2] School of Computer Science, South China Normal University, Guangzhou, China
gzhao@m.scnu.edu.cn

Abstract. Relational Databases are currently at the heart of information system of the companies. In recent years, the relational model has become de facto standard thanks to its maturity and efficiency. However, the fact that the data of some companies or institutions have become too large, new systems has appeared namely NoSQL which belongs to the Big Data era. Big Data comes due to the emergence of new online services on which customers have become increasingly connected, which creates a large digital data unbearable by the traditional management technical tools, which raise new challenges for companies especially to access, store and analyse data. In this paper we will propose a feasibility study of migration from relational databases to NoSQL databases specifically HBase database, by applying the operations of the relational algebra in HBase data model and explore the implementation of these operations on HBase by using the native functions of this DBMS and also by using the MapReduce Framework.

Keywords: Migration · Relational database · Relational algebra · HBase · Column-oriented · NoSQL · Big data · MapReduce · Feasibility assessment

1 Introduction

Relational database management systems (RDBMS) are the most common solution in many applications for storing and retrieving data due to its maturity and reliability. Relational databases are based on the Codd model (relational) [1] which has privileged a system of relations based solely on the values of the data, and a manipulation of these data using a high level language called SQL [3], implementing a new mathematical theory similar to the set theory proposed by Codd called "relational algebra" [1]. Relational algebra defines operations that can be applied on relations. Relational operations allow to create a new relation (table) from elementary operations on other tables namely union, intersection, selection, projection, and join.

However, these systems cannot support the explosion of digital data that modern Web applications have introduced. This explosion of digital data forces new ways of seeing and analyzing the world. These applications must support a large number of simultaneous users (tens of thousands or even millions), ensure the scalability

M. Ezziyyani et al. (eds.), *Advanced Information Technology, Services and Systems*, Lecture Notes in Networks and Systems 25, https://doi.org/10.1007/978-3-319-69137-4_34

generated by storage of large data capacities, be always available, manage semi-structured data and non-structured data and adapt quickly to changing needs with frequent updates and new features.

To address this problem, many solutions have turned to non-relational databases, commonly known as NoSQL databases, to enable massively parallel and geographically distributed database systems to support the internet applications such as facebook, ebay, twitter, Sears and Amazon [6].

The "NoSQL" databases are not usually a replacement, but rather a complementary addition to RDBMS and SQL. Consequently, developing a mapping tool between relational and NoSQL will be very much requested.

In our paper we will work on the HBase database [4, 8] thanks to its popularity. Indeed HBase is a Hadoop subproject, it is a distributed non-relational database management system, Written in Java, with structured storage for large tables. Zhao et al. [11] has carried out a comparison between MongoDB and relational algebra to investigating the feasibility of migrating relational databases to MongoDB, but there is no one for HBase. Migration requires feasibility assessment of the potential performance for new systems. For this purpose we will study the feasibility of applying the operations of the relational algebra in HBase data model and then explore the implementation of these operations on HBase by using the native functions of this DBMS and also byusing the MapReduce Framework [5]. The rest of the paper is structured as follows: In Sect. 2 we will introduce the basic definitions which we will begin with an introduction to the NoSQL databases, after we will present the HBase database, so we will see what MapReduce is. In Sect. 3 we will investigate the application of the relational algebra operations in HBase data model. In Sect. 4 we will explore the most common operations in relation implementation in HBase. Finally, Sect. 5 concludes our paper.

2 Basic Definitions

2.1 NoSQL Databases

A NoSQL database does not mean that no more queries are made, NoSQL simply means Not Only SQL. It is not a new query language for dialoging with the DBMS; it's a new approach for data storage. The term NoSQL refers to a category of massively parallel and geographically distributed databases management systems (DBMSs), most of them are designed to process large datasets within acceptable response time of user queries. They thus enrich the panel of traditional storage engines.

There are different categories of NoSQL DBMS [2]:

- **Key/Value:** The simplest NoSQL DBMS. It is in fact a huge hashmap with millions of entries. E.g. Redis, Riak, Voldemort (LinkedIn).
- **Document Oriented:** DBMS of key/value type with a document in value. The principle is to associate with a key a document regrouping different values. These documents are often represented by JSON or XML files. E.g. CouchDB, MongoDB
- **Column-oriented:** DBMS most resembling to the relational DBMSs, however Column-oriented DBMS allow missing values (unlike the relational model). These DBMSs are based on a notion of pair {key, value}. The column name can be seen

as the key. The column-oriented model, it's the model used in Hadoop. E.g. Cassandra, HBase, BigTable (Google)

- **Graph**: The goal is to represent the information in the form of nodes connected by edge (oriented or not). A node or edge can have attributes. This kind of DBMS stores effectively the relationships between data points, it's very useful for fraud detection, Real-Time recommendation engines and network and IT operations [19]. E.g. Neo4j, FlockDB

2.2 HBase

HBase is a distributed, column-oriented DBMS based on Hadoop. HBase provides random access and consistency for large amounts of unstructured and semi-structured data in a schema-less database organized by column families [9]. HBase uses HDFS as the file system for data storage and supports both queries and MapReduce. It was designed from the Google DBMS "BigTable" [10]. It's capable to storing a very large data (billions of rows/columns).

As show in Fig. 1, the HBase data model is based on six concepts [12], which are:

- **Table:** In HBase the data is organized in tables. Tables' names are strings.
- **Row:** In each table, the data is organized in rows. A row is identified by a unique key (RowKey).

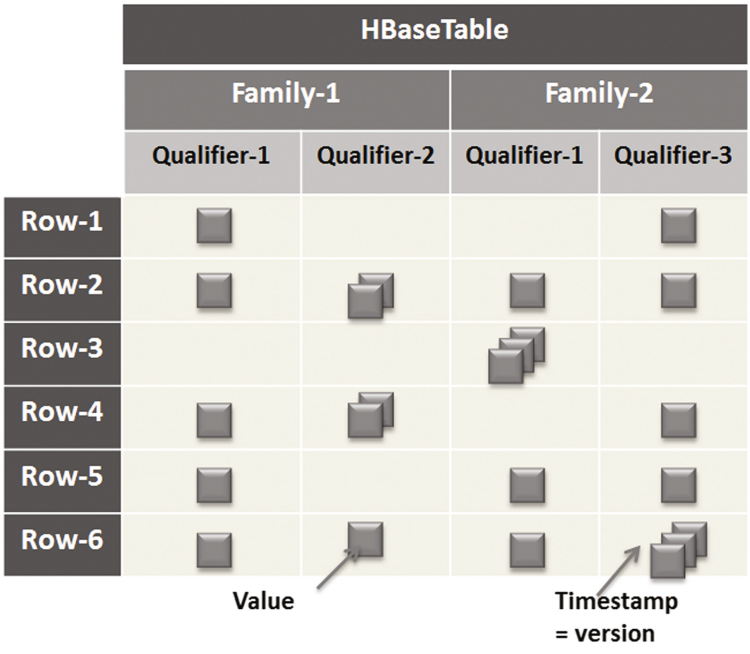

Fig. 1. HBase model [20]

- **Column Family:** Data within a row is grouped by "Column Families ". Each row of the table has the same "Column Families", which can be populated or not. The "Column Family" is set when the table is created in HBase. The names of "Column Family" are strings.
- **Column Qualifier:** Access to data within a "Column Family" is done via the "column qualifier" or column. It is not specified at the creation of the table but earlier at the insertion of the data.
- **Cell:** Stores the values of this cell. The combination of the "RowKey", the "Column Family" and the "Column Qualifier" uniquely identifies a cell.
- **Version:** The values within a cell are versioned. The versions are identified by their timestamp.

2.3 MapReduce

MapReduce [5, 7] is a framework for parallel distributed computing over large amounts of data. Distributed computing is done via a cluster of machines. MapReduce fully manages the cluster and load balancing. This allows handling distributed computing without any knowledge of the underlying infrastructure. MapReduce is based on the notion of job. A job is split in a set of tasks. There are two types of tasks: **Map task** and **Reduce Task**.

3 Relational Algebra in Hbase

3.1 Union Set

Union it's a basic operation in relational algebra which requires two union-compatible operands.

Given two HBase tables T1 and T2, the definition of the union set is:
$T1 \cup T2 = \{x | x \in T1 \text{ or } x \in T2\}$ where T1 and T2 are union-compatible.

3.2 Cartesian Product

A Cartesian product is a binary operation that combines two relations R1 and R2 and builds a third relationship exclusively containing all the possible combinations of occurrences of R1 and R2 relations, we note R1 $\times$ R2.

The number of occurrences of the resulting relationship of the Cartesian product is the number of occurrences of R1 multiplied by the number of occurrences of R2. In HBase we can define the Cartesian product as follows:

$$T1 \times T2 = \{(r, s) : r \in T1 \text{ and } s \in T2\}$$

Where T1 and T2 are two tables in HBase, r and s are rows in these tables.

3.3 Intersection

The intersection is an operation that holds in two relations R1 and R2 with the same pattern and building a third relation that contains all rows of R1 also belong to R2, but no other rows. In HBase we can define the intersection as follows:

$$T1 \cap T2 = \{r : r \in T1 \text{ and } r \in T2\}$$

Where T1 and T2 are two tables in HBase and r is a row in these tables.

3.4 Selection

A selection is a unary operation that extracts certain row (or rows) from an HBase table where the selection condition P is satisfied.

- Notation: $\sigma_p(T)$
- Parameter: Table T and P is a propositional formula formed of a combination of comparisons and logical operators.
- Result: $\sigma_p(T) = \{r \in T : r \text{ satisfies the conditions given by } P\}$

3.5 Projection

The projection of an HBase table T1 is the HBase table T2 obtained by selecting the rows with columns in the C set and eliminating duplicate rows.

- Notation: $\pi_{c_1,\ldots,c_n} \ldots\ldots T1$
- Parameter: T1 is an HBase Table and C is a set of columns of selecting rows
- Result: T2 is an HBase Table with only the columns specified in C.

3.6 θ-Join and Equijoin

θ-join (theta join) is a binary operation that consists of all combinations of rows in two HBase tables T1 and T2 that satisfy a condition.

- Notation:

$$T1 \underset{\theta}{\bowtie}_{r1\, v} T2 \text{ OR } T1 \bowtie_{r1\, r2} T2$$

- Parameter: T1 and T2 are two HBase Tables. r1 and r2 are two columns qualifier, θ is binary relational operator that can be : $>$, $\geq$, $=$, $<$ or $\leq$, v is a constant value.
- Result: a subset of Cartesian product where the condition is satisfied.

We call this operation equijoin where the θ operator contains equality.

3.7 Natural Join

The natural join is a binary operation; it's the result of the Cartesian product of two HBase tables with the condition that must be at least one common attribute with the same name and the same value. If this condition is omitted, and the two HBase tables have no common attributes, the natural join becomes simply the Cartesian product. It's defined as follows:

$$T1 \bowtie T2 = \pi_{\mathrm{C} \cup \mathrm{D}}\, \sigma_{\left((T1.a1=T2.a1)^{\wedge}(T1.a2=T2.a2)^{\wedge}\cdots^{\wedge}(T1.an=T2.an)\right)}(T1 \times T2)$$

Where C is the set of the column names of the HBase Table T1 and D is the set of the column names of the HBase Table T2.

3.8 Division

The division is a binary operation; it's a very powerful and useful operation, it's written as follows:

$$\mathrm{T1}(k) \div \mathrm{T2}(c)$$

Where T1and T2 are two HBase tables, k and c are the sets of column names of these HBase tables:

K = $\{k_1, \ldots\ldots, k_m, c_1, \ldots\ldots, c_n\}$, $c = \{c_1, \ldots\ldots\ldots, c_n\}$ Where $c \subset k$

The result of this operation consists of all rows r(x) in T1 that appear in T1 in combination with every tuple from T2, where x = k – c

The division it can be defined as follows:

$$T1(k) \div T2(c) = \{t | t \in \pi_{k-c}(T1) \text{ and } \forall u \in T2(t \times u \in T1)\}$$

4 Operations of Hbase

In this section we will model the HBase query capability by using the relational algebra.

4.1 Get

The "get" command is used to read data from an HBase table. This command returns a single line according to the row ID parameter. It's the equivalent of the SELECT command in SQL. Its syntax is as follows:

```
get '<table name>', '<row Id>'
```

by using this command we can read the data from an HBase table, but only one record, where the row Id of the row equals the second parameter of the command '<row Id>'. We can also specify the column showing in the result by using the following syntax:

get'<table name>', '<row Id>', {COLUMN ⇒ '<column family>:<column name>'}

This command can be modeled with relational algebra as follows:

get 'T', 'r1', {COLUMN ⇒['cf1:c1', 'cf1:c2',..,' cf1:cn', cfm:c1', 'cf2:c2',..,'cf2:cn']}

$$= \pi_{cfi:cj,i=1,2\ldots m,j=1,2\ldots n} \sigma_{rowkey='r1'}(T)$$

T is an HBase table,

4.2 Group by

Group by is often used in aggregate functions, it allows grouping tuples by the value of an attribute and applies an aggregate function for each group.

By default HBase does not support group by and aggregate functions, but it's possible to perform these tasks on data by using the MapReduce framework.

In the "Shuffle and Sort" phase [7], MapReduce performs sorting and grouping by key to ensure that the input parameter of a reducer is a set of tuples t = (k, [v]) where [v] is the collection of all the values associated with the key k. A reducer can call the aggregate functions on this list of grouped values.

Group by is an additional relational algebra operation [13]. We can present the MapReduce function that performs the Group by operation by this algebra calculation:

Algorithm 1.

selectedColumn ← $\pi_{c1,\ldots\ldots cn}$ T

for each unique k in πc_G selectedColumn

for all Row r ∈ selectedColumn

v[] ← $\pi c_A \sigma c_{G=k}$ r

return reducer(k,v[])

Where T is an HBase table, c is a set of selected column, c_A is the column on which will be applied an aggregate function, c_G is the column on which the grouping will be performed and v[] is a set of values of c_A column grouped by c_G.

4.3 Aggregate Function

Aggregate functions: are functions that will group the values of multiples rows. They have applied on a numeric column and return a single result for all selected rows or for each group of rows. Also for aggregate functions, HBase does not support these functions, but by using the MapReduce framework or HBase Coprocessors EndPoints we can implement them. Common aggregate functions include: count, sum, avg, max, and min. We can present these functions in relational algebra as follows:

$$G_1, \ldots\ldots, G_k g_{F1(C1),\ldots\ldots,Fn(Cn)}(T)$$

Where G is a set of grouping column, each C is one of the columns qualifiers of the HBase table T. each aggregate function F will be applied for each group according to $G_1,\ldots..,G_k$.

In HBase these functions can be handled by using the MapReduce framework or HBase Coprocessors EndPoints. We will treat three functions: count, sum, and avg. In this paper we will use MapReduce.

Count

The goal is to enumerate all the distinct value of a column in an HBase table, with the number of times that they are present within the table for each of them

Algorithm 2 . Driver class for Count operation

```
class driver
  method main(...)
    scan ← HBaseTable.scan()
    scan.addColumn(columnFamilyName, columnName)
    jobStart
```

Algorithm 3 . Mapper class for Count operation

```
class Mapper
  method Map(key, cellValue)
    Emit(cellValue, 1)
```

Algorithm 4 . Reducer class for Count operation

```
class Reducer
  method Reduce(cellValue, integers [1,1,1,1,.....])
    count ← 0
    for each integer i ∈ integers [i1,i2,.....] do
     count ← count + i
    Emit(cellValue, integer count)
```

The driver class, which runs on a client machine, is responsible for scanning the HBase table, selecting the grouping column, configuring the job and submitting it for execution.

The Mapper class will produce a list of pairs (key, value) [(k_1; v_1)]. Before being sent to the reducer class, the file is automatically sorted by key by Hadoop in the "shuffle & sort" phase.

The Reducer class; it will receive a group of pairs (key, values) (k_1; [v_1,v_1,….]) as input. Its role will be kept the unique key, calculates the sum of the values of all the pairs (key, values) received as input, and to generate a single pair (key, value) [(k_2; v_2)] as output, composed of the unique key and the obtained total.

The Count function can be handled also by using the HBase commands "count" or "get_counter" but without the "group by" feature.

Sum

It calculates the sum of a column in an HBase table containing numeric values.

Algorithm 5 . Driver class for Sum operation

```
class driver
  method main(...)
    scan ← HBaseTable.scan()
    scan.addColumn(columnFamilyName1,columnName1)
    scan.addColumn(columnFamilyNamen,columnNamem)
    jobStart
```

Algorithm 6 . Mapper class for Sum operation

```
class Mapper
  method Map(key , row)
  Emit(row.groupingColumn, row.sumingColumn)
```

Algorithm 7 . Reducer class for Sum operation

```
class Reducer
  method Reduce(string groupingColumn, values
  [v1,v2,…])
   sum ← 0
   for each value v ∈ values [v1,v2,.....] do
     sum ← sum + v
   Emit(groupingColumn , sum)
```

The function "sum" has the same principle of the function "count" the only difference is in the Map phase; that performs for each row associates for the grouping column the value of the summing column instead of the value of 1.

Avg

It allows calculating an average value of a column in an HBase table containing numeric values. Avg function has the same driver class of the sum function. In the mapper class, the map function sends a series of pairs (key, value) composed by the grouping column as a key and the value of the column on which the average will be calculated as value. Then, in "shuffle & sort" phase, Hadoop performs the sorting by key. Therefore, the Reducer can sum the values then calculate the average by dividing the sum by the number of items in the set of the values.

Algorithm 8 . Mapper class for Avg operation

```
class Mapper
  method Map(key , row)
    Emit(row.groupingColumn, row.avgColumn)
```

Algorithm 9 . Reducer class for Avg operation

```
class Reducer
   method Reduce(string groupingColumn, values
   [v1,v2,.....])
    sum←0
    count← 0
    for each value v ∈ values [v1,v2,.....] do
      sum ← sum + v
      count ← count+1
    avg ← sum / count
    Emit(groupingColumn , avg)
```

4.4 Join

Join in relational database allows associating several tables in the same query. It allows exploiting the power of relational databases to get results that combine efficiently data from multiple tables. HBase doesn't support the join operation, but it can be handled by using the declarative query languages that built on top of Hadoop like Pig [17] or Apache Hive [16] that launches implicitly MapReduce jobs for joining two HBase tables.

There are various join processing algorithms for MapReduce [21] environment like Repartition Join [15], Broadcast Join [15], and Trojan Join [18]. Following is the pseudo code of the Repartition Join, the most commonly used join algorithm.

Algorithm 10 . Map and Reduce functions for Repartition Join algorithm

```
Map (K: null, V : a record from a split of either R or
L)
   join_key ← extract the join column from V
   tagged_record ← add a tag of either R or L to V
   emit (join_key, tagged_record)
```

Algorithm 11 . Reducer class for Repartition Join

```
Reduce (K′: a join key,
LIST_V′: records from R and L with join key K′)
   create buffers BR and BL for R and L, respectively
   for each record t in LIST_V′ do
      append t to one of the buffers according to its
      tag
   for each pair of records (r, l) in BR × BL do
emit (null, new record(r, l))
```

Repartition Join [15]

In our case, to joining two HBase tables T1 and T2; the input parameter is a row of either T1 or T2 as a value. The first step in Map phase is to extract the join column from the row then adding a tag for each row to identify its originating table in Reduce phase using the secondary sorting [14], and finally emitting the pairs (k,v) where k is the join key and v is the tagged row.

Then the MapReduce framework is the responsible for the partitioning, sorting and merging tasks. In these tasks the framework sorts by key and sends all the rows with same join key to the same reducer.

For the Reduce phase, the input parameter is a pair of (k_1; [r_1, r_2,….]) where k_1 is the join key and [r_1, r_2,….] is a list of tagged rows associated to the k_1 key. The joining

operation is performed by splitting and buffering the tagged rows in two sets according to the table tag and handles the cross-product of the two sets.

5 Conclusion and Future Work

In this paper we proposed a feasibility study of migrating relational databases to HBase databases by applying the operations of the relational algebra in HBase data model and explore the implementation of these operations on HBase by using the native functions of this DBMS and also byusing the MapReduce Framework. Based on the above sections we can deduce that is theoretically the migration between relational databases and HBase databases can be handled efficiently. In perspective, we envisage to compare the performance of execution of the commons relational operations (bulk load, select, update, delete, join, group by, and aggregate functions) over a large database in relational and in HBase.

References

1. Codd, E.F.: A relational model of data for large shared data banks. Commun. ACM **13**(6), 377–387 (1970)
2. Moniruzzaman, A.B.M., Hossain, S.A.: Nosql database: new era of databases for big data analytics-classification, characteristics and comparison. arXiv preprint arXiv:1307.0191 (2013)
3. Codd, E.F.: The significance of the SQL/data system announcement. Computerworld **15**(7), 27–30 (1981)
4. George, L.: HBase: the Definitive Guide. O'Reilly Media Inc., Sebastopol (2011)
5. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
6. Abadi, D.J.: Data management in the cloud: limitations and opportunities. IEEE Data Eng. Bull. **32**(1), 3–12 (2009)
7. Yang, H.C., Dasdan, A., Hsiao, R.L., Parker, D.S.: Map-reduce-merge: simplified relational data processing on large clusters. In: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, pp. 1029–1040. ACM (2007)
8. Apache HBase Databases. http://hbase.apache.org/
9. Dimiduk, N., Khurana, A.: HBase in Action. Manning, Shelter Island (2013)
10. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: a distributed storage system for structured data. ACM Trans. Comput. Syst. (TOCS) **26**(2), 4 (2008)
11. Zhao, G., Huang, W., Liang, S., Tang, Y.: Modeling MongoDB with relational model. In: 2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies (EIDWT), pp. 115–121. IEEE (2013)
12. Khurana, A.: Introduction to HBase schema design. White Paper, Cloudera (2012)
13. Ceri, S., Gottlob, G.: Translating SQL into relational algebra: optimization, semantics, and equivalence of SQL queries. IEEE Trans. Softw. Eng. **4**, 324–345 (1985)
14. Lin, J., Dyer, C.: Data-intensive text processing with MapReduce. Synth. Lect. Hum. Lang. Technol. **3**(1), 1–177 (2010)

15. Blanas, S., Patel, J.M., Ercegovac, V., Rao, J., Shekita, E.J., Tian, Y.: A comparison of join algorithms for log processing in mapreduce. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, pp. 975–986. ACM (2010)
16. Apache Hive TM. http://hive.apache.org/
17. Olston, C., Reed, B., Srivastava, U., Kumar, R., Tomkins, A.: Pig latin: a not-so-foreign language for data processing. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp. 1099–1110. ACM (2008)
18. Dittrich, J., Quiané-Ruiz, J.A., Jindal, A., Kargin, Y., Setty, V., Schad, J.: Hadoop++: making a yellow elephant run like a cheetah (without it even noticing). Proc. VLDB Endow. **3**(1–2), 515–529 (2010)
19. Webber, J., Robinson, I.: The Top 5 Use Cases of Graph Databases, Neo Technology (2015)
20. LarbretB.: Hadoop HBase – Introduction (2015). https://www.slideshare.net/larbret/hadoop-hbase
21. Shaikh, A., Jindal, R.: Join query processing in mapreduce environment. In: Advances in Communication, Network, and Computing: Third International Conference, CNC 2012, Chennai, India, February 24–25, 2012, Revised Selected Papers, vol. 108, p. 275. Springer (2012)