AN ASYNCHRONOUS METHOD FOR WRITE OPTIMIZATION OF COLUMN-STORE DATABASES IN MAP-REDUCE

Feng Yu¹, Eric S. Jones¹, Weidong Xiong², Mohammed Hamdi², Wen-Chi Hou²
¹Youngstown State University, Youngstown, OH, USA
²Southern Illinois University, Carbondale, IL, USA
fyu@ysu.edu, esjones@student.ysu.edu
{wxiong,mhamdi,hou}@siu.edu

Abstract

Column-store databases feature a faster data reading speed compared with traditional row-based databases. However, optimizing write operations in a column-store database is a well-known challenge. Most existing works on write performance optimization focus on main-memory column-store databases. In this work, we extend the research on column-store databases in the Map-Reduce environment. We propose a data storage format called Timestamped Binary Association Table (or TBAT) without the need of global indexing. Based on TBAT, a new update method, called Asynchronous Map-Only Update (or AMO Update), is designed to replace the traditional update. A significant improvement in speed performance is shown in experiments when comparing the AMO update with the traditional update.

Keywords: column-store; write optimization; map-reduce

1 INTRODUCTION

Column-store databases (also known as columnar databases or column-oriented databases) refer to the category of databases that vertically partition data and separately store each column. The data storage in a column-store database is vertically partitioned and sharded by projecting each column into a separate fragment. A vertical fragment is referred as a BAT (Binary Association Table) (Peter Boncz et al., 2006), which is stored contiguously on a large enough disk page in order to mitigate seeking overheads across multiple ranges of data. The data in each BAT is densely patched in order to improve I/O performance, and also rapidly compressed utilizing light-weight compression scheme to improve storage efficiency.

The column-store database fits well into the write-once-and-read-many environments. Given the fact that the values for each attribute are stored separately in BATs, the column-store database works especially well for OLAP and data mining queries that retrieve a large number of tuples but only considers a small collection of attributes (Plattner & Plattner, 2009). It can simply retrieve only the attributes included in the query prediction without the need to read the entire tuple. The information retrieving speed is much faster in a column-store database. Another featured benefit of the column-store database is data compression, which can reach a higher compression rate and higher speed than a traditional row-based database. One of the major reasons is that the information entropy in the data
of one column is lower compared to that of row-based data.

The history of the column-store database can be traced back to 1970s when transposed files were implemented in the early development of DBMS, followed by applying vertical partitioning as a technique of table attribute clustering. By the mid 1980s, the advantage of a fully decomposed storage model (DSM) over the traditional row-based storage model (NSM or Normalized Storage Model) was studied (Copeland & Khoshafian, 1985; Khoshafian, Copeland, Jagodits, Boral, & Valduri, 1987; Zukowski, Nes, & Boncz, 2008).

TAXIR (TAXonomic Information Retrieval) is the first automatic application of column-store database focusing on biological information retrieval and management (Brill et al., 1971; Estabrook & Brill, 1969). KDB and Sybase IQ were the first two commercially available column-store databases developed in 93 and 95, respectively. It’s not until about 2005 when many open-source and commercial implementations of column-store databases took off (Abadi et al., 2009). The well-known column-store databases include Apache Cassandra (Ladwig & Harth, 2011), Apache HBase (George, 2011), MonetDB (PA Boncz & Kersten, 2002), SAP HANA, and Vertica (Lamb et al., 2012).

Optimizing write operations in a column-store database has always been a challenge (Krueger et al., 2010). The data in a column-store database is vertically decomposed into BATs and randomly distributed over the storage. Furthermore, assuming an external memory storage is employed, there is a non-trivial probability that a BAT is too large to fit into one page on the storage. Therefore, the writing on a column-store database will be significantly delayed by ad hoc access to large BATs across multiple pages. Optimizing the write operations in a column-store database is a demanding request.

Existing works majorly focus on write optimizations in a main-memory column-store database. Krueger et al. (Krueger et al., 2010, 2011) introduced the differential update to improve the write performance in MonetDB. A special columnar data structure, called delta buffer, is introduced to temporarily store decomposed row-based input data. However, to the best of our knowledge, very few works focused on optimizing the write performance on the out-of-core (OOC or external memory) column-store databases. Vertica (Lamb et al., 2012), a column-store database on large volume OOC storage, introduces a specially designed data storage procedure called k-safety to ensure ACID of update transactions on large volumes of data and improve the data importation efficiency. Nevertheless, k-safety focuses more on the transaction control rather than the write performance improvement for high-velocity update query streams.

Extending the column-store databases onto the big data file systems such as Google File Systems (GFS) (Ghemawat, Gobioff, & Leung, 2003) and Hadoop Distributed File Systems (HDFS) (White, 2010) has always been a demanding request (Babu, 2012). The state-of-the-art big data column-store databases such as BigTable (Chang et al., 2008) and HBase (Aiyer et al., 2012) store multidimensional data labeled by a special attribute, namely timestamp and require global indexing for fast data retrieval. However, global indexing requires extra resources of both computing and storage and can lower the speed of the write operations when rebuilding the index.

In this research, we focus on optimizing the write operations (mainly updates) on an OOC column-store database in a Map-Reduce environment. An operation called Asynchronous Map-Only Update (or AMO Update) was originally designed utilizing the Map-Reduce environment and a new data structure called Timestamped Binary Association Table (or TBAT). Furthermore, a Map-Reduce selection algorithm is developed to enable fast data retrieval. A major contribution of this work is that TBAT and AMO update allow the users to flexibly define column data type without the need of any extra data structure such as the global index.

The rest of the paper is structured as follows. Section 2 is the background introduction of column-store databases. Section 3 states the proposed method of update optimization on the OOC column-store database in Map-Reduce. Preliminary experimental results are illustrated in section 4. Section 5 is the conclusion and future works.

2 Background
The data structure of a column-store database exclusively uses BATs (Binary Association Tables). A BAT is a fragment of an attribute in the original row-based storage. It usually consists of an OID (Object Identifier) or ROWID, along with a column of attribute values, which in a pair is called a BUN (Binary Units). It is a physical model in a column-store database and the sole bulk data structure it implements. The BAT is categorized in a special group of storage models called DSM (Decomposed Storage Model) (Zukowski et al., 2008).

The row-based storage data is the original user input data, called the front-end data or logical data. To input the data into a column-store database, a mapping rule should be defined from the logical data structure to the physical data structure, namely BAT.

**Example 1 (Convert Row-Based Table to BAT).** Suppose the table name is CUSTOMER, with ID as the primary key.

CUSTOMER (ID, NAME, BALANCE)

**PRIMARY KEY: ID**

The row-based data is shown in Figure 1. In a columnar database, this logical table will be decomposed into three separate BATs, namely CUSTOMER_ID, CUSTOMER_NAME, and CUSTOMER_BALANCE. Each BAT contains two columns: an OID and an attribute value column with the column name as the corresponding column data type. ☐

This format is also referred to as the Fully Vertical Fragmentation (Abadi, 2008). The fully vertical fragmentation has many advantages. First of all, data accessing is efficient for queries accessing many rows but with fewer columns involved in the query. Another advantage is the reduction of the workload on the CPU and memory generated by OLAP and data mining queries, which typically consider only a few columns in a logical table.

Compared to the fully vertical fragmentation, another pattern is the partial vertical fragmentation (Gluch, Grust, Mainberger, & Scholl, 1997). It assumes the prior knowledge of which columns are frequently accessed together. Also, it employs the attribute usage matrix to determine optimal clustering of columns into vertical fragments. However, OLAP and data mining are application areas that indicate ad-hoc queries, as a good OLAP or data mining system must be able to quickly answer queries involving attributes of arbitrary combinations. Nevertheless, the partial vertical fragmentation is useful to detect the data block location in a distributed database system.

### 3 Update on Column-Stores in Map-Reduce

Update on a column-store database can be categorized into two types according to the residence of the target data to be changed, namely in-memory update and out-of-core update (or OOC update). An OOC update is to change the data stored in the external storage device. In this work, we focus on the external storage in Map-Reduce, for instance, HDFS. Each file in the HDFS is sharded into multiple chunks and distributed over the cluster.

There are two major bottlenecks of updates on out-of-core data. First is to seek the OID(s) of the target tuple(s). This could be time-consuming when the tuples are in a large volume and serialized on multiple file blocks on HDFS. Secondly, once the OID of the target tuple is retrieved, there can be multiple values to be changed. The database system needs to access several BATs across different blocks on the distributed storage, which will generate more ad hoc random access costs on disk and over the network.

Optimizing the OOC updates is, therefore, a demanding request. We proposed the Asynchronous Map-Only Update (or AMO Update) and a specialized...
data structure called Timestamped BAT to improve the writing performance caused by OOC update in an OOC column-store database on HDFS.

Algorithm 1: BAT_UPDATE_MR

Input: bat: the BAT file to update; update list: the list of BUN’s to update; threshold: max size of update list to fit into memory

1: if sizeof(update list) ≤ threshold then
  2:   temp = bat ∧<map-side update_list ON OID
        /* map-side left outer join*/
3:   else
  4:   temp = bat ∧<reduce-side update_list ON OID
        /* reduce-side left outer join*/
5:   for all line ∈ temp do
6:     if line.update_list.OID!=NULL /* if this line is matched in the outer join, i.e. it has been updated*/
  7:       output(BUN(line.update_list.OID, line.update_list.VALUE))
8:     Else
  9:       Output(BUN(line.bat.OID, line.bat.VALUE))
return SUCCESS

3.1 UPDATE ON BAT IN MAP-REDUCE

To perform an update on a target BUN, the update process on the BAT involves two phases. The first is to search for the location of the target record (or BUN) in BAT by OID. Second is to update the record value at the corresponding location. In this work, we further extend the BAT into the Map-Reduce environment.

Algorithm 1 describes a typical procedure of updates on BAT in Map-Reduce. Searching for target records in a BAT is not efficient if nested loops are involved. This phase can be converted into an outer join operation between the BAT and the update list (of BUNs). In addition, Algorithm 1 adjusts to the size of arbitrary update list. When the update list size is small enough to fit into the memory, a map-side outer join is performed. Otherwise, a common reduce-side outer join can be performed when the update size is too large. In both cases, the join result is retained on HDFS as the intermediate result. A filtering phase is performed and the VALUE and OID of the target BUN are always retained if there is an update on the updating location.

3.2 TIMESTAMPED BAT

Traditional BATs are composed of two columns in each BUN data pair, namely OID and ATTRIBUTE VALUE. In this work, we propose a specially designed decomposed storage model with a simplicity of implementation, called Timestamped BAT (or TBAT). Figure 2 depicts the structure of TBUN and TBAT_SLIP in HDFS.

Figure 2: TBAT Data Structure in Map-Reduce

<table>
<thead>
<tr>
<th>optime</th>
<th>oid</th>
<th>float</th>
</tr>
</thead>
<tbody>
<tr>
<td>time1</td>
<td>101</td>
<td>100.00</td>
</tr>
<tr>
<td>time1</td>
<td>102</td>
<td>200.00</td>
</tr>
<tr>
<td>time1</td>
<td>103</td>
<td>300.00</td>
</tr>
</tbody>
</table>

(a) TBAT CUSTOMER_BALANCE

<table>
<thead>
<tr>
<th>optime</th>
<th>oid</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>time1</td>
<td>101</td>
<td>1</td>
</tr>
<tr>
<td>time1</td>
<td>102</td>
<td>2</td>
</tr>
<tr>
<td>time1</td>
<td>103</td>
<td>3</td>
</tr>
</tbody>
</table>

(b) TBAT CUSTOMER_ID

Figure 3: A TBAT Example

As the name suggests, each tuple, TBUN, in a TBAT is different from a BUN in a traditional BAT in that its structure follows a timestamp, value, OPTIME format, which is used to record the time when an insert, update, or deletion operation is performed on this BAT tuple. The data type of OPTIME is a four-byte TIMESTAMP, for example as in MySQL, that occupies relatively small space. OID is the object identifier of ROWID type, and ATTRV is the attribute value corresponding to the OID.

Figure 3 depicts the example of TBATs of CUSTOMER_BALANCE and CUSTOMER_ID, respectively.
Specifically, **CUSTOMER_BALANCE** and **CUSTOMER_ID** are the two TBATs decomposed from the original base table **CUSTOMER**. The reason for all **OPTIME** to be the same is because the initial data are assumed to be inserted in one batch of insertion. In addition, the **OIDs** are assumed to start from 101 in this example.

The major improvement of TBAT is that, first of all, TBAT does not require any global pre-sorting or indexing. Based on TBAT, we propose an update algorithm and a selection algorithm that can achieve efficient data update and retrieval without any help from extra global data structures. Compared with the previous lessons learned using the distributed index, this proposed data structure is designed to prevent those distributed failure problems. Secondly, a user-defined attribute type **ATTRV** to be included in a TBUN. Therefore, the user can flexibly define arbitrary kinds of schema translated from relational database models.

### 3.3 Asynchronous Map-Only Update on Column-Stores

Based on TBAT, we propose the Asynchronous Map-Only Update (or AMO Update) in the Map-Reduce environment. The principle of AMO update is to avoid seeking and writing in every effort and to use the timestamp field of TBAT to label the newly updated data that is directly appended to the end of a TBAT. In such a manner, we don’t have to frequently perform ad hoc data seeking and writing by simply accepting multiple versions of TBUN data with the same **OID** but different attribute values and timestamps.

The AMO update is simple and straightforward in Map-Reduce. We simply describe the procedure of the AMO update as follows. The update list of target BUNs is also assumed to be collected in a distributed environment. Once the update list is submitted for execution, the mapper of the AMO update simply appends the list of updating TBUNs at the end of the TBAT file. In HDFS, there is only one mapper operation involved that simply shards the update list into slips and flushes them to the distributed storage. The file append operation has been supported in Apache Hadoop since the 0.20.0 release.

Compared with the update on BAT, the cost of AMO update is significantly lower than the BAT update, since the AMO update is a map-only procedure to append an update list while the BAT update involves a more expensive outer join operation in Map-Reduce.

**Example 2 (AMO Update).** Without loss of generality, we use an example of AMO update targeting on a single tuple. It can be easily generalized to any update targeting on a collection of tuples. A SQL query of an example of update operation is shown as follows.

```
UPDATE customer SET balance = 201.00 WHERE id = 2
```

The record with **OID** equals 102 in the TBAT customer balance is the target tuple. The target value is to change the attribute value from the original value to 201.00. Instead of seeking the position to the record with **OID**=102, AMO update directly appends at the end of the TBAT a new tuple as (time2, 102, 201.00). The timestamp, when AMO update is performed, is assumed to be time2, and 201.00 is the newly updated value. The TBAT customer balance after the AMO update is illustrated in **Figure 4**.

---

### 3.4 TBAT Map-Reduce Selection

AMO update will not hurt the data consistency on TBAT. There could be multiple versions of the same **OID** data with different **ATTRV** and **OPTIME** stored throughout the HDFS. The target TBUN is the one with the latest timestamp and attribute value. To fully utilize the power of Map-Reduce in HDFS, we propose a selection algorithm that can filter out previous version data.

**Algorithm 2** describes the selection on TBAT in Map-Reduce, given a selection range and a target **OID**. It can be easily extended to the selection with an input list of **OIDs**. There are two filtering phases.

---

<table>
<thead>
<tr>
<th>optime</th>
<th>oid</th>
<th>float</th>
</tr>
</thead>
<tbody>
<tr>
<td>time1</td>
<td>101</td>
<td>100.00</td>
</tr>
<tr>
<td>time1</td>
<td>102</td>
<td>200.00</td>
</tr>
<tr>
<td>time1</td>
<td>103</td>
<td>300.00</td>
</tr>
<tr>
<td>time2</td>
<td>102</td>
<td>201.00</td>
</tr>
</tbody>
</table>

**Figure 4. TBAT customer balance after AMO Update**
involved in the algorithm. The MAPPER performs the first filtering on the map side, where, for each TBAT slip, the MAPPER selects only the TBUNs with their OIDs that fall into the given selection range. Then key-value pairs are emitted by the MAPPER that are comprised of the OID of the selected TBUN and a nested pair, which consists of the VALUE of the TBUN and the OPTIME or timestamp. The emitted key-value pairs will be shuffled and sorted according to the key, i.e. the OID of the mapper-selected TBUN, and then sent to the reducer. For each OID, the REDUCER algorithm on the reduce side will select the pair, as the target pair, with the most recent timestamp from the input list of pairs. Finally, the value of the target pair is returned as one result satisfying the selection range.

Algorithm 2: TBAT_SELECTION_MR

1: procedure MAPPER (tbat_block: a block of TBAT file in HDFS; oid_range: range of selection for OIDs)
2:   for all line ∈ tbat_block do
3:     if line.OID ∈ oid_range then
4:       emit(line.OID, new pair(line.VALUE, line.TIMESTAMP)) /*The emitted key is the OID and the emitted value is a nested pair.*/
5:   procedure REDUCER (oid: the OID of a TBUN in a reduced TBAT chunk; pairs: the list of pairs associated with the same OID)
6:     target_pair = max_TIMESTAMP(pairs) /*only select the most recent pair*/
7:     output(oid, target_pair.VALUE)

Example 3 (Selection). We continue to use the previous example and select the balance of customer with id=2 after the previous update query is executed. The selection query is as follows.

SELECT balance FROM customer WHERE id=2

Since customer id is intact, seeking the OID of id=2 is fast. After OID=102 is retrieved, in TBAT customer balance, two tuples will be returned

t1=(time1, 102, 200.00)
t2=(time2, 102, 201.00)

As we compare the timestamps, time2 is later than time1. Then 201.00 is returned which is consistent with the last update value. □

4 PRELIMINARY EXPERIMENT RESULTS

Preliminary experiment results are designed in order to compare the speed performance between AMO updates on TBATs and traditional updates on BATs in Map-Reduce. The experiment is performed on a Cloudera Distributed Hadoop (CDH) version 5.3 cluster with one master node and three slave nodes. The master node is equipped with an Intel Core i5-2400 3.10GHz CPU and 8GB RAM. Each slave node includes an Intel Core2 Duo E8200 2.66GHz CPU and 4GB RAM. The embedded Hadoop version is 2.5.0, and the total HDFS storage capacity is 310GB with the block size of 64MB. We use the Gigabit Ethernet as the cluster interconnection. The experiment test code is implemented in Java SE 1.7 and Apache Pig Latin version 0.12.0.

On HDFS, synthetic BAT datasets of size 1GB and 10GB are randomly generated, consisting of an OID column and an ATTRIV (attribute value) column to simulate a large BAT. Then a TBAT is derived from the BAT with an additional OPTIME (timestamp) attribute. For each dataset, five update input tables are uniformly generated consisting of from 10% to 30% of the original table. We use these tables to simulate the list of update targets.

The absolute running time of AMO updates on TBAT and traditional updates on BAT are shown in Figure 5 and Figure 6. The detailed average running times of the AMO updates on 1GB and 10GB data are 194 and 1698 seconds, while the average running times of traditional updates are 425 and 4413 seconds, respectively. The relative overhead of running time is defined as:

overhead = \frac{Time(BAT) - Time(TBAT)}{Time(TBAT)} \times 100\%

where time(TBAT) is the AMO update running time on TBAT. As shown in Figure 7 and Figure 8, the AMO update is on average 120 and 160 times faster.
than the traditional update in the 1GB and 10GB tests, respectively.

The difference between AMO updates and traditional updates grows greater with the increase of the data size due to the growth of the OID searching overhead of traditional updates.

5 CONCLUSION AND FUTURE WORKS

In this research, we introduce a new method called AMO update for write optimization on OOC column-store databases in the Map-Reduce environment. AMO update employs a simple and effective data structure called TBAT to improve the update performance without the need of global indexing. A Map-Reduce selection algorithm is developed for fast data retrieval. Significant improvements in the running speed of AMO update have been shown in preliminary experiment results.

For future works, we will investigate, in depth, the performance variation of the Map-Reduce selection algorithm on TBAT after a large amount of updates. Another topic is to introduce distributed indexes on TBAT slips in HDFS to improve the global data retrieval speed.

6 REFERENCES

Abadi, D. J. (2008). Query Execution in Column-Oriented Database Systems. MIT.


Authors

Feng Yu is currently an Assistant Professor at the Department of Computer Science and Information Systems at Youngstown State University, OH, USA. He is awarded Ph.D. in computer science in 2013 from Southern Illinois University at Carbondale, IL, USA. His research interests include query optimization in traditional and large-scale databases, NoSQL databases, big data management, and cloud computing.

Eric S. Jones was awarded a dual Bachelor's in History and Political Science in May 2013 and M.S. in Computer Information Systems in May 2015 all from Youngstown State University, Youngstown, OH, USA. Much of his thesis re-
search was focused on column-store databases and their optimization. Mr. Jones now works at NLMK Pennsylvania as an Oracle applications developer.

**Weidong Xiong** is currently a Ph.D. Candidate at the Computer Science Department of Southern Illinois University Carbondale. Before joining the Ph.D. program at SIU, Xiong was a senior software engineer in IT industry with many years of experience of Windows application programming.

**Mohammed Hamdi** is currently a Ph.D. candidate in the Department of Computer Science at Southern Illinois University-Carbondale. He received his master's degree in Computer Science from Southern Illinois University-Carbondale in 2013. His main research interests are databases, query optimization, data mining, and big data.

**Wen-Chi Hou** received the MS and Ph.D. degrees in computer science and engineering from Case Western Reserve University, Cleveland Ohio, in 1985 and 1989, respectively. He is presently a professor of computer science at Southern Illinois University at Carbondale. His interests include statistical databases, query optimization, concurrency control, XML databases.