Adaptive Prejoin Approach for Performance Optimization in MapReduce-based Warehouses

Weiping Qu
Heterogeneous Information Systems Group
University of Kaiserslautern
qu@informatik.uni-kl.de

Michael Rappold*
Department of Computer Science
University of Kaiserslautern
m_rappol@cs.uni-kl.de

Stefan Dessloch
Heterogeneous Information Systems Group
University of Kaiserslautern
dessloch@informatik.uni-kl.de

ABSTRACT

MapReduce-based warehousing solutions (e.g. Hive) for big data analytics with the capabilities of storing and analyzing high volume of both structured and unstructured data in a scalable file system have emerged recently. Their efficient data loading features enable a so-called near real-time warehousing solution in contrast to those offered by conventional data warehouses with complex, long-running ETL processes.

However, there are still many opportunities for performance improvements in MapReduce systems. The performance of analyzing structured data in them cannot cope with the one in traditional data warehouses. For example, join operations are generally regarded as a bottleneck of performing generic complex analytics over structured data with MapReduce jobs.

In this paper, we present one approach for improving performance in MapReduce-based warehouses by pre-joining frequently used dimension columns with fact table redundantly during data transfer and adapting queries to this join-friendly schema automatically at runtime using a rewrite component. This approach is driven by the statistics information derived from previous executed workloads in terms of join operations.

The results show that the execution performance is improved by getting rid of join operations in a set of future workloads whose join exactly fits the pre-joined fact table schema while the performance still remains the same for other workloads.

1. INTRODUCTION

By packaging complex custom imperative programs (text mining, machine learning, etc.) into simple `map` and `reduce` functions and executing them in parallel on files in a large scalable file system, MapReduce/Hadoop[1] systems enable analytics on large amounts of unstructured data or structured data in acceptable response time.

With the continuous growth of data, scalable data stores based on Hadoop/HDFS[2] have achieved more and more attention for big data analytics. In addition, by means of simply pulling data into the file system of MapReduce-based systems, unstructured data without schema information is directly analyzed with parallelizable custom programs, whereas data can only be queried in traditional data warehouses after it has been loaded by ETL tools (cleansing, normalization, etc.), which normally takes a long period of time.

Consequently, many web or business companies add MapReduce systems to their analytical architecture. For example, Fatma Özcan et al. [12] integrate their DB2 warehouse with the Hadoop-based analysis tool - IBM Infosphere BigInsights with connectors between these two platforms. An analytical synthesis is provided, where unstructured data is initially placed in a Hadoop-based system and analyzed by MapReduce programs. Once its schema can be defined, it is further loaded into a DB2 warehouse with more efficient analysis execution capabilities.

Another example is the data warehousing infrastructure at Facebook which involves a web-based tier, a federated MySQL tier and a Hadoop-based analytical cluster - Hive.

Such orchestration of various analytical platforms forms a heterogeneous environment where each platform has a different interface, data model, computational capability, storage system, etc.

Pursuing a global optimization in such a heterogeneous environment is always challenging, since it is generally hard to estimate the computational capability or operational cost concisely on each autonomous platform. The internal query engine and storage system do not tend to be exposed to outside and are not designed for data integration.

In our case, relational databases and Hadoop will be integrated together to deliver an analytical cluster. Simply transferring data from relational databases to Hadoop without considering the computational capabilities in Hadoop can lead to lower performance.

As an example, performing complex analytical workloads over multiple small/large tables (loaded from relational data-

*finished his work during his master study at university of kaiserslautern

[1]one open-source implementation of MapReduce framework from Apache community, see http://hadoop.apache.org

[2]Hadoop Distributed File System - is used to store the data in Hadoop for analysis

25^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 28.05.2013 - 31.05.2013, Ilmenau, Germany.

bases) in Hadoop leads to a number of join operations which slows down the whole processing. The reason is that the join performance is normally weak in MapReduce systems as compared to relational databases [15]. Performance limitations have been shown due to several reasons such as the inherent unary feature of `map` and `reduce` functions.

To achieve better global performance in such an analytical synthesis with multiple platforms from a global perspective of view, several strategies can be applied.

One would be simply improving the join implementation on single MapReduce platform. There have been several existing works trying to improve join performance in MapReduce systems [3, 1].

Another one would be using heuristics for global performance optimization. In this paper, we will take a look at the second one. In order to validate our general idea of improving global performance on multiple platforms, we deliver our adaptive approach in terms of join performance. We take the data flow architecture at Facebook as a starting point and the contributions are summarized as follows:

1. Adaptively pre-joining tables during data transfer for better performance in Hadoop/Hive.

2. Rewriting incoming queries according to changing table schema.

The remainder of this paper is structured as follows: Section 2 describes the background of this paper. Section 3 gives a naïve approach of fully pre-joining related tables. Based on the performance observation of this naïve approach, more considerations have been taken into account and an adaptive pre-join approach is proposed in Section 4, followed by the implementation and experimental evaluation shown in Section 5. Section 6 shows some related works. Section 7 concludes with a summary and future work.

2. BACKGROUND

In this section, we will introduce our starting point, i.e. the analytical data flow architecture at Facebook and its MapReduce-based analytical platform - Hive. In addition, the performance issue in terms of join is also stated subsequently.

2.1 Facebook Data Flow Architecture

Instead of using a traditional data warehouse, Facebook uses Hive - a MapReduce-based analytical platform - to perform analytics on information describing advertisement. The MapReduce/Hadoop system offers high scalability which enables Facebook to perform data analytics over 15PB of data and load 60TB of new data every day [17]. The architecture of data flow at Facebook is described as follows.

As depicted in Figure 1, data is extracted from two types of data sources: a federated MySQL tier and a web-based tier. The former offers the category, the name and corresponding information of the advertisements as dimension data while the actions such as viewing an advertisement, clicking on it, fanning a Facebook page are extracted as fact data from the latter.

There are two types of analytical cluster: production Hive cluster and ad hoc Hive cluster. Periodic queries are performed on the production Hive cluster while the ad hoc queries are executed on the ad hoc Hive cluster.

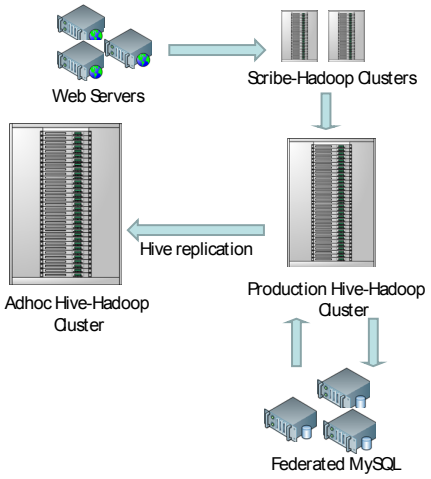


Figure 1: Facebook Data Flow Architecture[17]

2.2 Hive

Hive [16] is an open source data warehousing solution built on top of MapReduce/Hadoop. Analytics is essentially done by MapReduce jobs and data is still stored and managed in Hadoop/HDFS.

Hive supports a higher-level SQL-like language called HiveQL for users who are familiar with SQL for accessing files in Hadoop/HDFS, which highly increases the productivity of using MapReduce systems. When a HiveQL query comes in, it will be automatically translated into corresponding MapReduce jobs with the same analytical semantics. For this purpose, Hive has its own meta-data store which maps the HDFS files to the relational data model. Files are logically interpreted as relational tables during HiveQL query execution.

Furthermore, in contrast to high data loading cost (using ETL jobs) in traditional data warehouses, Hive benefits from its efficient loading process which pulls raw files directly into Hadoop/HDFS and further publishes them as tables. This feature makes Hive much more suitable for dealing with large volumes of data (i.e. big data).

2.3 Join in Hadoop/Hive

There has been an ongoing debate comparing parallel database systems and MapReduce/Hadoop. In [13], experiments showed that performance of selection, aggregation and join tasks in Hadoop could not reach parallel databases (Vertica & DBMS-X). Several reasons of the performance difference have been also explained by Stonebraker et al. in [15] such as repetitive record parsing, and high I/O cost due to non-compression & non-indexing.

Moreover, as MapReduce was not originally designed to combine information from two or more data sources, join implementations are always cumbersome [3]. The join performance relies heavily on the implementation of MapReduce jobs which have been considered as not straightforward.

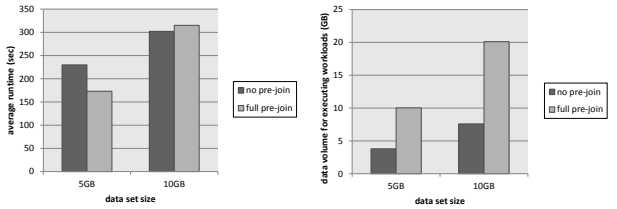
As Hive is built on top of MapReduce/Hadoop, the join operation is essentially done by corresponding MapReduce jobs. Thus, Hive suffers from these issues even though there have been efforts [5] to improve join performance in MapReduce systems or in Hive.

3. FULL PRE-JOIN APPROACH

Due to the fact that the join performance is a performance bottleneck in Hive with its inherent MapReduce feature, one naïve thinking for improving total workload performance would be to simply eliminate the join task from the workload by performing a rewritten workload with the same analytical semantics over pre-joined tables created in the data load phase. A performance gain would be expected by performing large table scan with high parallelism of increasing working nodes in Hadoop instead of join. In addition, the scalable storage system allows us to create redundant pre-joined tables for some workloads with specific join patterns.

In an experiment, we tried to validate this strategy. An analytical workload (TPC-H Query 3) was executed over two data sets of TPC-H benchmark (with scale factor 5 & 10) of the original table schema (with join at runtime) and a fully pre-joined table schema (without join) which fully joins all the related dimension tables with the fact table during the load phase, respectively. In this case, we trade storage overhead for better total performance.

As shown on the left side of the Figure 2(a), the performance gain of the total workload (including the join) over the data set with SF 5 can be seen with 6GB storage overhead introduced by fully pre-joining the related tables into one redundant table (shown in Figure 2(b)). The overall



(a) Average Runtimes (b) Accessed Data Volume

Figure 2: Running TPC-H Query-3 on Original and Full Pre-joined Table Schema

performance can be significantly increased if workloads with the same join pattern later frequently occur, especially for periodic queries over production Hive-Hadoop cluster in the Facebook example.

However, the result of performing the same query on the data set with SF 10 size is disappointing as there is no performance gain while paying 12.5GB storage for redundancy (shown in Figure 2(b)), which is not what we expected. The reason could be that the overhead of scanning such redundant fully pre-joined tables and the high I/O cost as well offset the performance gain as the accessed data volume grows.

4. ADAPTIVE PRE-JOIN APPROACH

Taking the lessons learned from the full pre-join approach above, we propose an adaptive pre-join approach in this paper.

Instead of pre-joining full dimension tables with the fact table, we try to identify the dimension columns which occurred frequently in the `select`, `where`, etc. clauses of previous executed queries for filtering, aggregation and so on. We refer to these columns as additional columns as compared to the join columns in the join predicates. By collecting a list of additional column sets from previous queries, for example,

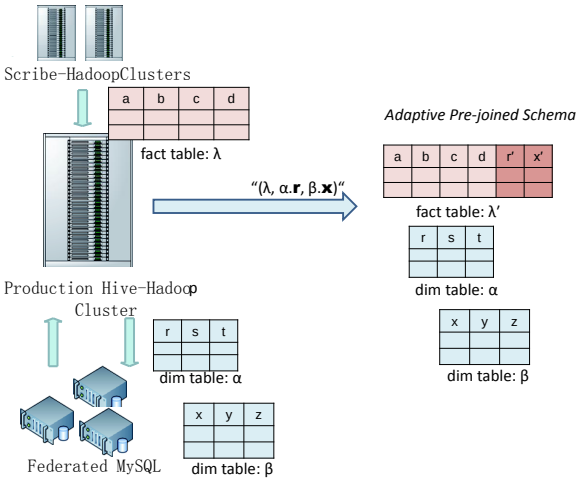


Figure 3: Adaptive Pre-joined Schema in Facebook Example

the periodic queries on production Hive-Hadoop cluster, a frequent column set could be extracted.

One example is illustrated in Figure 3. The frequent set of additional columns has been extracted. The column `r` in dimension table α is frequently joined with fact table in company in the previous workloads as a filter or aggregate column, as the same for the column `x` in dimension table β. During next load phase, the fact table is expanded by redundantly pre-joining these two additional columns `r` and `x` with it.

Depending on the statistics information of previous queries, different frequent sets of additional columns could be found in diverse time intervals. Thus, the fact table is pre-joined in an adaptive manner.

Assume that the additional columns identified in previous queries will also frequently occur in the future ones (as in the Facebook example), the benefits of adaptive pre-join approach are two-fold:

First, when all the columns (including dimension columns) in a certain incoming query which requires a join operation have been contained in the pre-joined fact table, this query could be directly performed on the pre-joined fact table without join.

Second, the adaptive pre-join approach leads to a smaller table size in contrast to the full pre-join approach, as only subsets of the dimension tables are pre-joined. Thus, the resulting storage overhead is reduced, which plays a significant role especially in big data scenarios (i.e. terabytes, petabytes of data).

To automatically accomplish the adaptive pre-join approach, three sub-steps are developed: frequent column set extraction, pre-join and query rewrite.

4.1 Frequent Column Set Extraction

In the first phase, the statistics collected for extracting frequent set of additional columns is formated as a list of entries each which has the following form:

$$Set: \{Fact,\ Dim\_X.Col\_i,\ Dim\_X.Col\_j \ldots\ Dim\_Y.Col\_k\}$$

The join set always starts with the involved fact table while the joint dimension columns are identified and cap-

tured from the `select`, `where`, etc. clauses or from the sub-queries.

The frequent set of additional columns could be extracted using a set of frequent itemset mining approaches [2, 7, 11]

4.2 Query Rewrite

As the table schema is changed in our case (i.e. newly generated fact table schema), initial queries need to be rewritten for successful execution. Since the fact table is pre-joined with a set of dedicated redundant dimension columns, the tables which are involved in the `from` clause of the original query can be replaced with this new fact table once all the columns have been covered in it.

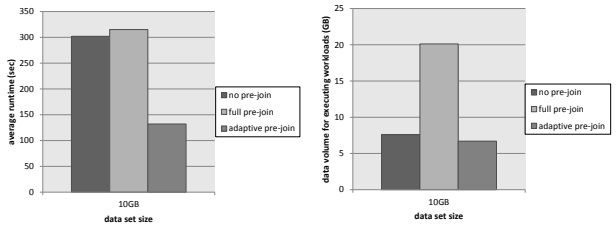
By storing the mapping from newly generated fact table schema to the old schema in the catalog, the query rewrite process can be easily applied. Note that the common issue of handling complex sub-queries for Hive can thereby be facilitated if the columns in the sub-query have been pre-joined with the fact table.

5. IMPLEMENTATION AND EVALUATION

We use Sqoop[3] as the basis to implement our approach. The TPC-H benchmark data set with SF 10 is adaptively pre-joined according to the workload statistics and transferred from MySQL to Hive. First, the extracted join pattern information is sent to Sqoop as additional transformation logic embedded in the data transfer jobs for generating the adaptive pre-joined table schema on the original data sources. Furthermore, the generated schema is stored in Hive to enable automatic query rewrite at runtime.

We tested the adaptive pre-join approach on a six-node cluster (Xeon Quadcore CPU at 2.53GHz, 4GB RAM, 1TB SATA-II disk, Gigabit Ethernet) running Hadoop and Hive.

After running the same TPC-H Query 3 over the adaptive pre-joined table schema, the result in the Figure 4(a) shows that the average runtime is significantly reduced. The join



(a) Average Runtimes (b) Accessed Data Volume

Figure 4: Running TPC-H Query-3 on Original, Full Pre-joined and Adaptive Pre-joined Table Schema

task has been eliminated for this query and the additional overheads (record parsing, I/O cost) have been relieved due to the smaller size of redundancy as shown in Figure 4(b).

6. RELATED WORK

An adaptively pre-joined fact table is essentially a materialized view in Hive. Creating materialized views in data warehouses is nothing new but a technique used for query optimization. Since 1990s, a substantial effort [6, 8] has been to answer queries using views in data warehouses. Furthermore, several subsequent works [14, 10] have focuses on dynamic view management based on runtime statistics (e.g. reference frequency, result data size, execution cost) and measured profits for better query performance. In our work, we reviewed these sophisticated techniques in a MapReduce-based environment.

Cheetah [4] is a high performance, custom data warehouse on top of MapReduce. It is very similar to the MapReduce-based warehouse Hive introduced in this paper. The performance issue of join implementation has also been addressed in Cheetah. To reduce the network overhead for joining big dimension table with fact table at query runtime, big dimension tables are denormalized and all the dimension attributes are directly stored into the fact table. In contrast, we choose to only denormalize the frequently used dimension attributes with the fact table since we believe that less I/O cost can be achieved in this way.

7. CONCLUSION AND FUTURE WORK

We propose a schema adaption approach for global optimization in an analytical synthesis of relational databases and a MapReduce-based warehouse - Hive. As MapReduce systems have weak join performance, frequently used columns of dimension tables are pre-joined with the fact table according to useful workload statistics in an adaptive manner before being transfered to Hive. Besides, a rewrite component enables the execution of incoming workloads with join operations over such pre-joined tables transparently. In this way, better performance can be achieved in Hive. Note that this approach is not restricted to any specific platform like Hive. Any MapReduce-based warehouse can benefit from it, as generic complex join operations occur in almost every analytical platform.

However, the experimental results also show that the performance improvement is not stable while the data volume grows continuously. For example, when the query is executed on one larger pre-joined table, the performance gain from eliminating joins is offset by the impact caused by the record parsing overhead and high I/O cost during the scan, which results in worse performance. This concludes that the total performance of complex data analytics is effected by multiple metrics rather than a unique consideration, e.g. join.

With the continuous growth of data, diverse frameworks and platforms (e.g. Hive, Pig) are built for large-scale data analytics and business intelligent applications. Data transfer between different platforms generally takes place in the absence of key information such as operational cost model, resource consumption, computational capability etc. within platforms which are autonomous and inherently not designed for data integration. Therefore, we are looking at a generic description of the operational semantics with their computational capabilities on different platforms and a cost model for performance optimization from a global perspective of view. The granularity we are observing is a single operator in the execution engines. Thus, a global operator model with generic cost model is expected for performance improvement in several use cases, e.g. federated systems.

Moreover, as an adaptively pre-joined fact table is regarded as a materialized view in a MapReduce-based warehouse, another open problem left is how to handle the view maintanence issue. The work from [9] introduced an incre-

[3]an open source tool for data transfer between Hadoop and relational database, see http://sqoop.apache.org/

mental loading approach to achieve near real-time datawarehousing by using change data capture and change propagation techniques. Ideas from this work could be taken further to improve the performance of total workload including the pre-join task.

8. REFERENCES

[1] F. N. Afrati and J. D. Ullman. Optimizing joins in a map-reduce environment. In *Proceedings of the 13th International Conference on Extending Database Technology*, EDBT '10, pages 99–110, New York, NY, USA, 2010. ACM.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[3] S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, and Y. Tian. A comparison of join algorithms for log processing in mapreduce. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, pages 975–986, New York, NY, USA, 2010. ACM.

[4] S. Chen. Cheetah: a high performance, custom data warehouse on top of mapreduce. *Proc. VLDB Endow.*, 3(1-2):1459–1468, Sept. 2010.

[5] A. Gruenheid, E. Omiecinski, and L. Mark. Query optimization using column statistics in hive. In *Proceedings of the 15th Symposium on International Database Engineering & Applications*, IDEAS '11, pages 97–105, New York, NY, USA, 2011. ACM.

[6] A. Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, Dec. 2001.

[7] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *SIGMOD Rec.*, 29(2):1–12, May 2000.

[8] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, SIGMOD '96, pages 205–216, New York, NY, USA, 1996. ACM.

[9] T. Jörg and S. Deßloch. Towards generating etl processes for incremental loading. In *Proceedings of the 2008 international symposium on Database engineering & applications*, IDEAS '08, pages 101–110, New York, NY, USA, 2008. ACM.

[10] Y. Kotidis and N. Roussopoulos. Dynamat: a dynamic view management system for data warehouses. *SIGMOD Rec.*, 28(2):371–382, June 1999.

[11] H. Mannila, H. Toivonen, and I. Verkamo. Efficient algorithms for discovering association rules. pages 181–192. AAAI Press, 1994.

[12] F. Özcan, D. Hoa, K. S. Beyer, A. Balmin, C. J. Liu, and Y. Li. Emerging trends in the enterprise data analytics: connecting hadoop and db2 warehouse. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, SIGMOD '11, pages 1161–1164, New York, NY, USA, 2011. ACM.

[13] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, SIGMOD '09, pages 165–178, New York, NY, USA, 2009. ACM.

[14] P. Scheuermann, J. Shim, and R. Vingralek. Watchman: A data warehouse intelligent cache manager. In *Proceedings of the 22th International Conference on Very Large Data Bases*, VLDB '96, pages 51–62, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[15] M. Stonebraker, D. Abadi, D. J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin. Mapreduce and parallel dbmss: friends or foes? *Commun. ACM*, 53(1):64–71, Jan. 2010.

[16] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy. Hive - a petabyte scale data warehouse using Hadoop. In *ICDE '10: Proceedings of the 26th International Conference on Data Engineering*, pages 996–1005. IEEE, Mar. 2010.

[17] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, and H. Liu. Data warehousing and analytics infrastructure at facebook. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, pages 1013–1020, New York, NY, USA, 2010. ACM.