The Impact of Cluster Characteristics on HiveQL Query Optimization

Ognjen V. Joldzic, *Member, IEEE,* Dijana R. Vukovic, *Member, IEEE*

Abstract — **Huge amount of data is stored by different kinds of applications for further analysis. Relational databases were used for decades as data storages, but in some cases they are not suitable for Big Data processing. To solve the problem, non-relational databases were developed. As a help for transferring data from relational databases to non-relational databases, adequate tools were developed. In this paper, a tool named Sqoop is presented. The issue of query optimization should be addressed by all applications that deal with large amounts of data, regardless of their field of application and scope. The impact of cluster characteristics on HiveQL query optimization is analyzed in this paper.**

Keywords — **Apache Hadoop, distributed data storages, HiveQL, query optimization.**

I. Introduction

RELATIONAL databases are used for decades to store different kinds of data. As the quantity of data to be stored keeps growing (Big Data problem), it becomes harder to process it with relational databases. As a solution for the Big Data processing problems, non-relational databases were developed. Big Data problem can appear in applications already using relational databases. Different tools were developed for moving from relational data stores to non-relational data stores to improve Big Data processing.

Considering the fact that open-source database solutions are widely used, in this paper transferring data from relational database MySQL to distributed data storage HDFS (*Hadoop Distributed File System*) using Apache Sqoop is presented.

In section II, a short overview of relational databases in general is given. Section III covers distributed data storage HDFS. In section IV, the process of transferring data from relational databases to distributed data storage using Apache Sqoop is shown. To perform query analysis, after transferring data to HDFS, data was imported to non-relational database under Apache Hadoop framework, named Hive. Section V illustrates the comparative analysis of different queries in both data storages, MySQL and Hive. In section VI conclusion is given.

II. Relational databases

Relational databases are one of the most used data storages in the past few decades. According to [1] the relational database model can be described as a set of three basic concepts: a set of relations (tables), a set of relational operators and integrity rules that maintain the consistency of the data stored in a database. A database can have one or more tables. A table consists of columns and rows. Columns represent the attributes of one entity, and a row represents a concrete state for one entity. Table 1 represents one table named *person* in a relational database with four columns and two rows. Entity *person* has attributes: id, name, surname, year of birth and sex (row 1 in Table 1).

Table 1: TABLE PERSON

id	name	surname	year_of_birth	sex
1	Ognjen	Joldzic	1986	male
2	Dijana	Vukovic	1984	female

Basic operations for data manipulation in relational databases are known as CRUD operations: data can be created (inserted), retrieved (read), updated, or deleted. To perform CRUD operations, SQL (*Structured Query Language*) is used. Considering the fact that the first name for SQL was SEQUEL (*Structured English Query Language*), all statements written using SQL are reminiscent of sentences written in English. SELECT statement is used for reading data from database and it usually contain SELECT keyword in combination with keywords: FROM, WHERE and ORDER BY. If we want to read data from Table 1 about a person whose year of birth is 1986, the corresponding SQL statement is ***SELECT * FROM person WHERE year_of_birth=1984***.

The result will be data from first row in Table 1. SELECT statement is part of DML (Data Manipulation Language). DML is used to add (INSERT), read (SELECT), update (UPDATE) or delete (DELETE) data. DML also contains control statements, e.g., BEGIN TRANSACTION, COMMIT and ROLLBACK. For managing tables DDL (Data Definition Language) is used. DDL statements include e.g. CREATE, ALTER, and DROP. To manage database rights and permissions DCL (Data Control Language) is used. Major statements from DCL are GRANT and REVOKE.

Advantages of relational databases are: ease of use, flexibility, precision, security, data independence, and DML. Disadvantages of relational databases are: performance, physical storage consumption and slow extraction of meaning from data. [2].

Ognjen V. Joldzic, Faculty of Electrical Engineering, University of Banja Luka, Patre 5, 78101 Banja Luka, Bosnia and Herzegovina (e-mail: ognjen.joldzic@etfbl.net).

Dijana R. Vukovic, Norwegian University of Science and Technology, Department of Telematics, O.S. Bragstads plass 2B, N-7491 Trondheim, Norway; (e-mail: dijanav@item.ntnu.no).

Relational databases in general can be divided into commercial and open-source databases. Depending of the system that will use the database for storing data, one can use different database type and different RDBMS in general. Today's most popular RDBMS in commercial group are Oracle [3] and SQL Server [4]. On the other side, most popular open-source RDBMS are: MySQL [5] and PostgreSQL [6].

III. Distributed data storage - HDFS

HDFS is a distributed file system developed as part of the Apache Hadoop platform for Big Data processing. All of its major features are specifically aimed at facilitating storage, management and processing of vast amounts of data. HDFS is a file system that has a specific purpose, and therefore implements several mechanisms that differ from the analogous mechanisms found in native file systems managed by the operating system. HDFS storage exhibits best results when used with data for which the number of reads is far greater that the number of writes ("write once, read many times" principle), and each of the reads requires most of the stored data. In other words, HDFS is designed primarily for batch processing, rather than direct access, and should be used in circumstances where it is more important for the file system to provide access times that are linearly proportionate to the amount of data, than to enable quick access to any single piece of data. This ability to "scale out" is one of the key features that is required not only from HDFS, but from any similar solution for data management. In order to achieve this, HDFS partially abandons some of the requirements imposed by traditional file systems, which makes it a POSIX-uncompliant file system [7].

HDFS storage is organized in blocks, which are a lot larger than ordinary file system blocks. By making the block larger (default size of the block is 64 MB), the time required to transfer the block from storage is larger than the time required to seek to the start of the block, making the transfer time close to disk rate for any single block of data. This makes HDFS more efficient when working with data files whose size is close to, or even larger than the size of a single block.

Distributed file systems such as HDFS are implemented (as per official recommendations) in the form of a cluster consisting of an arbitrary number of nodes made of commodity, inexpensive hardware. This feature gives HDFS storage the ability to scale to vast amounts of data, because new capacity can be seamlessly integrated into the existing infrastructure. Nodes within the cluster can assume one of two distinctive roles in the context of data storage: Name node and Data node. A Name node is a primary node of the storage cluster which tracks the location of all blocks within the storage. The number of Name nodes is also arbitrary, which provides for redundancy of metadata in case of hardware failure. A cluster is, in general, configured with replication capabilities, with each data block replicated across several nodes, as indicated by the replication factor. A client communicates only with the Name node through a special command line interface that enables HDFS management (copying, deleting, moving etc.). The Name node caches each input file locally until its size reaches the size of the configured block, after which it is replicated across Data nodes until each block is available on a number of nodes equal to the replication factor. For smaller clusters, replication factor should be set to 2, while for bigger clusters, the default value is 3. With larger replication factors, it is easier to locate a block (since more nodes contain it), but the efficiency of the storage is decreased (less total space is available for storage, and the replication process introduces more latency). Data nodes have no knowledge of the file system structure, and only employ mechanisms that enable them to access the files efficiently (local directory structure, transparent to the cluster). All information about the location of the blocks, as well as the tree structure created by the client is kept within the Name node(s).

Communication between the nodes is implemented by application-level protocols on top of TCP transport protocol. Data nodes report periodically to the Name node using keepalive messages, called *Heartbeats*. *Heartbeats* enable the Name node to keep track of the functioning Data nodes and, therefore, the status of the entire cluster. Several mechanisms have been implemented to enable re-balancing and re-replication of data blocks in case any single Data node should fail [8]. Furthermore, all platforms for Big Data processing try to minimize the amount of data that is transferred over the network, thus avoiding congestion, and instead try to execute computational tasks on the nodes that contain the actual data. Therefore, it may be efficient to relocate blocks that are accessed frequently, which is one of the functionalities implemented by the Name node.

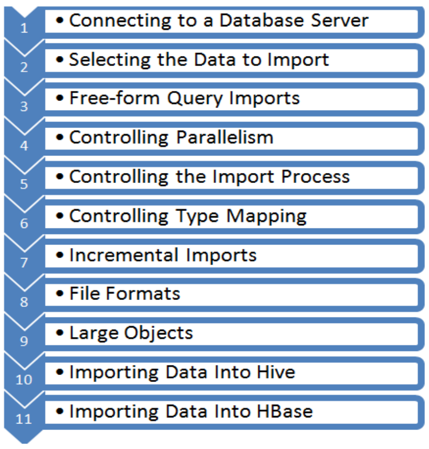
With proper configuration and optimal input data grouping, HDFS can provide a very efficient storage solution which is able to handle huge amounts of data. On the other hand, misconfiguration or inadequate dataset properties lead to sub-par results and performance penalties which make HDFS a bottleneck in an application. HDFS (and Hadoop, in general) is not meant as a universal storage system, and its used is actually discouraged in environments which could not benefit from its features [9].

IV. Apache Sqoop

The Big Data problem was the main reason for developing non-structured databases. Since structured databases, with the most used representative of this database group - relational databases, are used for years as data storage, there was a need for solving a problem of transferring data from structured to non-structured databases. Doing this job without some automation mechanism could be very complex and complicated, and it would require a lot of effort and a lot of time. As Apache Hadoop became one of the most used frameworks for distributed processing of large data sets, Apache foundation developed a tool for transferring data from relational databases to Apache Hadoop. A tool named Apache Sqoop [10] is: "*a tool designed for efficiently transferring bulk data between Apache Hadoop and* *structured data stores such as relational databases*". This tool can be used for importing data from relational database management systems (RDBMSs) into HDFS. Also, it can be used for reverse transfer, transforming the data with MapReduce and exporting the data back into chosen RDBMS. MapReduce is used by Sqoop to import and export data, which provides parallel operations and fault tolerance.

In the process of transferring relational databases to HDFS, as input data, Sqoop uses a database table, reading table row-by-row into HDFS. Because of parallel processing with MapReduce, the output of the tool will be a set of files containing the copied table data. Output files can be one of supported file types: delimited text files, or binary Avro or SequenceFiles with serialized data from one table record. Sub-product from this transfer is also one Java class that can encapsulate one table record. It can be used by the user in subsequent MapReduce processing of the data. Transfer process can be customized in a way of specifying delimiters, file formats, row ranges, columns etc.

Import from chosen RDBMS to HDFS can be done in 11 steps (Figure 2) [11].



1 • Connecting to a Database Server
2 • Selecting the Data to Import
3 • Free-form Query Imports
4 • Controlling Parallelism
5 • Controlling the Import Process
6 • Controlling Type Mapping
7 • Incremental Imports
8 • File Formats
9 • Large Objects
10 • Importing Data Into Hive
11 • Importing Data Into HBase

Fig. 2. Steps in importing from RDBMS to HDFS

In step 3, the user can add an SQL query to select one specific set from a database table to transfer it into HDFS, e.g. only data inserted in 2013. In step 4, the user can specify the number of map tasks (parallel processes) to use to perform the import. Some RDBMS have their own "fast" connector to for data transfer to other systems, so, instead of default JDBC, that other connector can be chosen in step 5 for making transfers in shorter periods of time. For now, the "fast" connector is only available for MySQL and PostgreSQL. In step 7 the user can specify if he wants to import just the data set that weren't impot earlier. Hive is Hadoop's data warehouse infrastructure that provides data summarization and ad hoc querying [12]. HBase is Apache's scalable, distributed database that supports structured data storage for large tables [13].

V. COMPARATIVE RESULTS OF DATABASE QUERYING

In order to achieve the performance and resilience that are the key features of HDFS-based storage cluster, several key factors must be taken into consideration. Some of the factors pertain to the configuration parameters which affect all applications using the cluster in the exact same manner. These options configure the global execution environment, and are particularly useful for specialized applications or a cluster that is custom tailored for a special purpose (or a single application). Another way of tuning the performance of the system is by optimizing the individual queries before they are submitted for execution. Unoptimized queries can significantly degrade the nominal performance of the data warehouse, thereby degrading the usability of the entire system.

Hive data warehouse offers a query language very similar to SQL (HiveQL [14]) which internally translates queries into one or more MapReduce jobs, depending on the complexity of the query itself. Several guidelines have been published in order to facilitate optimization of HQL [15]. Standard SQL parsers that are commonly contained within RDBMS systems are usually equipped with query optimizers which adjust the submitted queries so that the resources are efficiently utilized. On the other hand, successful HiveQL query optimization relies on a number of factors, some of which can change during the normal course of operation (the size of the dataset, for example), thus requiring dynamic runtime optimization in order for jobs to run efficiently and with acceptable execution times.

However, the infrastructure on which a Hive storage cluster is deployed can sometimes prevent optimal execution of MapReduce application. In other words, query optimization has to be performed in accordance with the platform which executes it. Namely, if a cluster is implemented using virtual machines connected to a common storage, execution time can be limited by the available disk I/O throughput. In order to prove this, Figure 3. shows a simple comparison between two queries, each executed on a Hive cluster and a common RDBMS. Both queries involve a full-text search within a sample GeoIP database [16] multiplied several times to provide an adequate volume of data. The first query is executed over a single unindexed table containing about 10^8 records and is composed of two nested subqueries (each performing a search on the same table). Due to a high disk I/O demand, Hive produces execution times that are several times longer than those of the compared RDBMS. High I/O requirements are not evident on the individual nodes – which could be misleading, but are present on the physical host machine. The second query contains an inner join between the same table used for the first query and a second table containing a much smaller dataset. Due to the lack of indexing structures within the relational database, its execution time is expectedly high. In this case, Hive was able to efficiently distribute the data across the nodes of the cluster, thereby enabling the distribution of the processing tasks, which, in turn, yielded much better execution times than the first query.

Furthermore, this type of query left enough room for additional tuning and optimization. Hive provides a very efficient way of minimizing disk I/O requirements when performing joins between a very large and a very small dataset by using a feature called MapJoin, which loads the smaller dataset as a hashtable in memory thereby reducing

I/O strain and effectively eliminating the Reduce stage of the job itself.

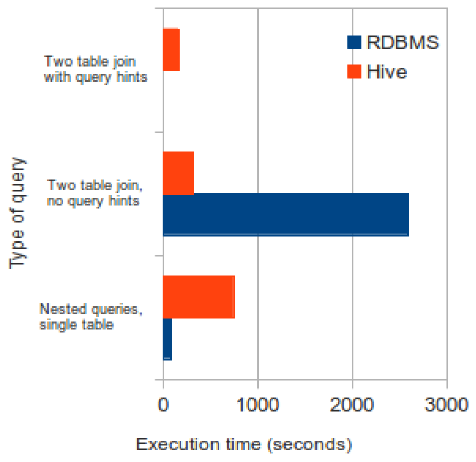


Fig. 3. Execution times for different types of queries

The MapJoin feature is activated by either specifying a query hint indicating which table should be loaded, or by activating a configuration option allowing Hive to determine if any of the tables involved in the join are suitable for such an operation. The second and the third column groups show the results of the same join query executed with and without the described query hint. It is important to note that the I/O bottleneck problem is still present, but is far less limiting than in previous cases. The relational database management system used in this test does not support explicit query hints that could be directly comparable to this feature, so the corresponding column is not shown in the chart.

If a Hive cluster is deployed on separate servers (with separate storage facilities), the performance of the queries improves even further. In such an environment, the order of the tables within a JOIN statement can also be an important factor for resource consumption. Specifically, Hive executes the mapping tasks separately for each of the tables involved in the join. During the reduce phase, the aggregate values for the first n-1 tables are buffered in memory, while the values from the last table in the list are “streamed” directly into the reducer [16]. Therefore, optimal execution requires that the largest table is listed last in the join clause of the query. This can also be achieved in two ways: manually – by ordering the list of tables according to their size, or by using a specialized query hint specifying the largest table explicitly.

The ability of the application to automatically optimize the queries during runtime depends on the trends of data growth. Estimation of these trends is usually possible, so an application should be able to dynamically adapt its query structure to the actual condition of the cluster and, in combination with the features provided by Hive itself and a properly deployed infrastructure, provide acceptable performance for virtually any type of job.

VI. Related Work

Several existing papers in the field of Hive performance deal with the problem of query optimization [17, 18]. However, current optimization trends are generally based on input query rewriting, where the level of optimization is in close correlation to the number of rewrite rules involved in the process [19], without taking into account the specific characteristics of cluster deployment and their impact on the overall performance.

VII. Conclusion

Optimization is one of the key factors when implementing efficient data analysis systems. In order for any such system to produce results in acceptable times, its users must have a thorough understanding of not only the content, but also the deployment details of the system itself. The issue of query optimization should be addressed by all applications that deal with large amounts of data, regardless of their field of application and scope.

Acknowledgment

Acknowledgment goes to the Laboratory of Information Systems at the Faculty of Electrical Engineering, University of Banja Luka, for providing access to a server used for testing.

References

[1] N. Bhatnagar, "Security in Relational Databases", published in book "*Handbook of Information and Communication Security*", Springer, 2010.
[2] (Online Resource) "Advantages and Disadvantages of Using Relational Databases" (Available on: http://maxlogix.blogspot.no/2009/09/advantages-and-disadvantages-of-using.html).
[3] (Online Resource) Oracle (Available on: http://www.oracle.com/index.html).
[4] (Online Resource) Microsoft SQL Server (Available on: http://www.microsoft.com/en-us/sqlserver/default.aspx).
[5] (Online Resource) MySQL (Available on: http://www.mysql.com/).
[6] (Online Resource) PostgreSQL (Available on: http://www.postgresql.org/).
[7] (Online Resource) POSIX (Available on: http://standards.ieee.org/develop/wg/POSIX.html).
[8] (Online Resource) “HDFS Architecture Guide” http://hadoop.apache.org/docs/stable/hdfs_design.html
[9] Tom White, "Hadoop: The Definitive Guide, Third Edition“, O'Reilly/Yahoo Press, 2012.
[10] (Online Resource) Apache Sqoop (Available on: http://sqoop.apache.org/).
[11] (Online Resource) Apache Sqoop: User Guide (Available on: http://sqoop.apache.org/docs/1.4.0-incubating/SqoopUserGuide.html).
[12] (Online Resource) Hive (Available on: http://hive.apache.org/).
[13] (Online Resource) HBase (Available on: http://hbase.apache.org/).
[14] (Online Resource) HiveQL Language Manual (Available on: https://cwiki.apache.org/confluence/display/Hive/LanguageManual.
[15] (Online Resource) "Tip: Using Joins in Hive" (Available on: http://blog.safaribooksonline.com/2012/12/05/tip-using-joins-in-hive/).
[16] (Online Resource) "GeoIP databases and web services" (Available on: http://www.maxmind.com/en/geolocation_landing).
[17] Anja Gruenheid, Edward Omiecinski, Leo Mark, “Query Optimization Using Column Statistics in Hive”, Proceedings of the 15th Symposium on International Database Engineering & Applications, 2011.
[18] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Antony, Hao Liu, Raghotham Murthy, “Hive – A Petabyte Scale Data Warehouse Using Hadoop”, IEEE 26th International Conference on Data Engineering (ICDE), 2010.
[19] Yingzhong Xu, Songlin Hu2, “QMapper: A Tool for SQL Optimization on Hive Using Query Rewriting”, Proceedings of the 22nd international conference on World Wide Web companion, 2013.