OORJA
ISSN - 0974-7869 (Print)
ISSN - 2395-6771 (Online)

MULTI-JOIN-ORDERING QUERY OPTIMIZATION ALGORITHM FOR HIVE WAREHOUSE WITH MAPREDUCE

Ms. Nisha Jain, Research Scholar, Department of Computer Science, RTU, Kota
Dr. Preeti Tiwari, Associate Professor, Department of Computer Science, ISIM, Jaipur

Abstract

According to the Digital Report of July, 2021, Billions of users around the world uses Mobile Phones, Internet, social media every second. This huge range of heterogeneous digital data is called Big Data, and is measured in terms of terabytes or petabytes. It is difficult to the conventional relational databases to handle these heterogeneous data for data analytics, but is still in use significantly in the growth of Big Data. To handle SQL-based structured queries, Hadoop is one of the prominent and well-suited solution that allows Big Data to be stored and processed. Hive support SQL queries on Hadoop. Hive warehouse, is the oldest SQL-engine on the top of the Hadoop framework and to store the processed data, it uses HDFS (Hadoop Distributed File System). On the Hadoop, MapReduce is an execution engine that executes SQL-based queries. In the Query Optimization, join ordering always plays a significant role because when the order of tables in joining operation is changed, execution time of the query is reduced to a greater extent. The main problem of the Hive is that it does not enhance the order of the join for an SQL-query and also does not give assurance for an optimal execution plan. Its time complexity is measured in exponential (Shan, Y., & Chen, Y., 2015).The main focus of this paper is to discover the finest join ordering solution for a Hive query optimization problem through appropriate search algorithms and to improve SQL-based Hive queries performance with MapReduce–based system.

Keyword- Big Data, Hadoop, Hive, HDFS, MapReduce, Query Optimization Technique

Introduction

With the exponential growth of data with time in terms of volume, variety and veracity, it is extremely difficult for the traditional databases to handle complexity and demonstrate absolute performance in storing and querying data at high speed. Big Data Technologies play a significant role in handling heterogeneous data for better operational efficiency and intelligent decision making. In the age of the Internet, data is a treasure trove for any organization or system. Many organizations based on their data management strategies systematically believe in data warehouses which are vital for the growth of the organization. In the present scenario, huge amount of data (in terms of petabytes) is holed by the companies or organizations that cannot be managed by centralized servers. Due to these requirements, the data warehouse techniques have changed over the past few years and it uses high-end servers to holding huge amount of data to a set of commodity hardware machines in a distribution manner. The mapreduce programming paradigm from Google has facilitated this transformation by providing highly scalable and fault tolerant distributed systems for production level quality application software (Vissapragada, B., 2014). To handle SQL-based structured queries, Hadoop is one of the prominent and well-suited frameworks that allows Big Data to be stored and processed. Many traditional database venders such as Microsoft PolyBase, TeraData SQL-H, Oracles SQL and some researchers are also trying to migrate from SQL queries to the Hadoop

framework for faster response time. Apache Hive is open-source warehouse that is similar to SQL queries, which is used and discovered by several Hadoop users for their data processing requires. Apache Hive and its HiveQL (Hive Query Language) is one of the oldest SQL query engine on Hadoop framework and it converts HiveQL queries to MapReduce for processing and storing processed data in HDFS (Hadoop Distributed File System).

Hadoop: It is one of the prominent and well-suited solutions that allow Big Data to be stored and processed, for both unstructured data and for structured data. It is an open-source framework to store and analyze huge datasets on a cluster of machines in a distributed style. It stores large files into block. If these blocks compressed indivisibly, they improved throughput time and speedup the block performance. Basically, they used LZO compression algorithm for effective block processing. When the size of data increases, Hadoop MapReduce cost and execution time is much better than traditional database system such as Microsoft SQL server. Some significant features of Hadoop - Fault Tolerance, High Availability, Scalability, and Reliability so on. Hadoop Ecosystem is a platform that offers a wide range of services to solve Big Data problems. There are two important elements of Hadoop - HDFS and MapReduce.

MapReduce: It is a batch processing tool which works on Java programming. It has two parts- Map converts data into key-values and takes the data from HDFS for processing, while Reducer takes the data from Map and generates new key values as final result and submit to HDFS.

Hadoop Distributed File System (HDFS): It is a cost effective and fault-tolerant storage system of Hadoop framework. It is a Java based data storage system that stores data in the form of small pieces (blocks). It creates many copies (replicas) of each data blocks for fast computation. It's also called Master-Slave model. It is further divided into two parts namely Namenode and Datanode. Namenode is also known as Master Node and it does not provide storage facility. It only stores and manages the Datanodes, The Datanode takes care of the data location, size & number of data, availability of data blocks etc., Data node is also known as Salvenode and is responsible for storing actual data block and performs various operations on it such as Read & Write, Replication, Deletion operations. Every Datanode consists of two segments, The first segment is used for storing data files and the second segment is used for storing the copied metadata.

Apache Hive: Many SQL supported open sources include Hive Stringer, Implala, Hadapt, Data HAWQ CitusDB Rainstor, MapR and Apache Drill etc (Pal,S., 2016). On Hadoop Framework, Apache Hive and its HiveQL language is considered to be one of the oldest SQL query engine. It was coined by Facebook in 2007, later taken over by the Apache Software Foundation and released the first version of Apache Hive in 2012. It is an ETL and data warehouse tool that uses HQL (Hive Query Language) similar to SQL-Relational Database Language for batch processing on structured data in a Big Data environment. It converts HiveQL queries to MapReduce for processing and stores processed data in HDFS.

Important Features of Hive are- File Styles -Avro files, Sequence File, ORC, Text File, RC File, Parquet, and LZO compression (Bagui, S., & Devulapalli, K. ,2018), all file format are supported by Hive. Sequential Files are suitable when user is storing intermediated data in the MapReduce joins, whereas ORC and Parquet are optimal for query processing in Hive and these files cannot be updated. Avro is the best option when there is a change the schema over the time but the

performance of the query is slower than ORC and Parquet. CSV files are perfect for extraction of data from Hadoop that work on Python or Java and also support different columns styles for partitioning the data in order to improve query performance.

Data Types- Various scalar data types like Integer, Float, Double, String to name a few and composite datatypes like Struct, Arrays, Maps, List etc are also used by Hive for executing complex queries. It stores data in tables similar to traditional database, where every table constructs rows and each row stated columns. Each column has a related type–Primitive Types (Integer- [Tinyint(1 bytes) int (4 bytes) , smallint (2 bytes), Bigint (8 bytes)], Float- 4 bytes, Double-8 bytes) and Complex Types which including Associative arrays – map <key-type, value-type>, Lists – list <element-type>, Structs – struct <file-name: field-type, ... > to name a few.

The next Section describes the Architecture and Query Processing of Hive, Section-III focuses on the Query Optimization for Hive Query. Section-IV focuses on the MapReduce Query Execution Engine. Section-V focuses on the Literature Review and the related work and the final section draws the Conclusion of this research paper.

Hive: Architecture and Query Processing

Hive Architecture and Query Processing by Hive-Hadoop Ecosystem has been described in **Figure-1.** Firstly, the user sends SQL- based queries through various database engines like JDBC, ODBC, Thrift Drivers, CLI Command Line Interface) and WUI (Web User Interface) to the Hive server and the Hive server sends these queries to Driver for execution. The Database Driver collects these queries from different sources and transfers it to the compiler where the execution of the queries takes place.

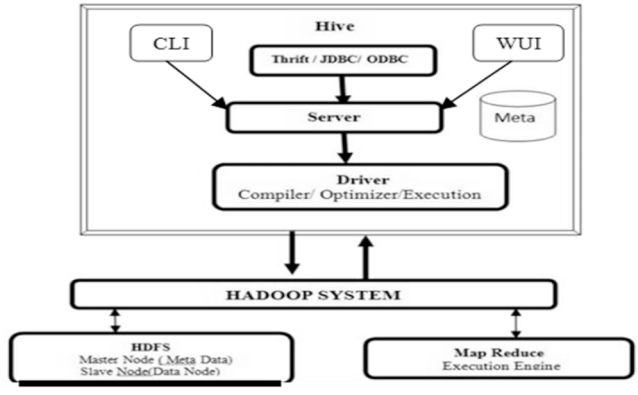

Figure 1: Hive-HadoopEcosystem [Bagui, S., & Devulapalli, K. (2018)]

The compiler then checks the syntax of the query and performs validation by Query Parsing Technique. After that semantic analysis is performed on the query and then the query is transformed from HiveQL statements into MapReduce Job. The Hive MetaStore is used as a storage area to store the structure of the data of the several tables and their partitions. It also contains metadata, which is used for read & write operations and the HDFS files is used for the data storage. (Bagui, S., & Devulapalli, K. (2018) **(Refer Figure 2)**

OORJA
ISSN - 0974-7869 (Print)
ISSN - 2395-6771 (Online)

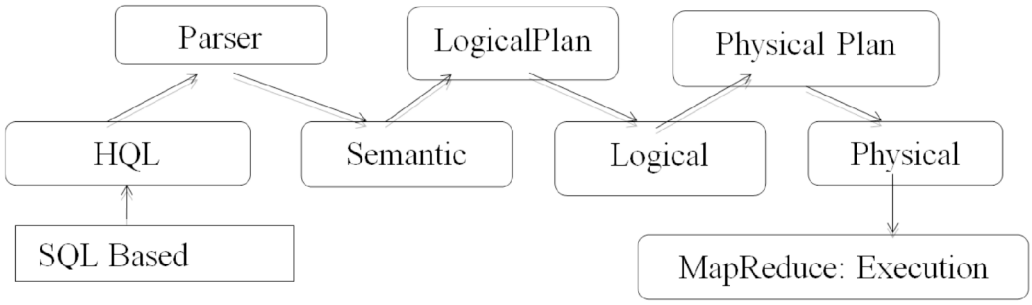

Figure 2 : Query Optimization Process in Hive (Pal, S., 2016)

There are three major components of HQL query. The first component is Query Optimization that is responsible for discovering the best Query Plan for the query. The second component is Query Execution Plan that indicated the best query plan and the third component is MapReduce Query Execution Engine which is responsible for query processing.

These three components can be explained via the given example.

UserQuery-1 (Shan, Y., & Chen, Y., 2015)

"Select * From R1, R2, R3, R4 Where R1.A=R2.A and R2.B=R3.B and R3.C=R .C;"

QP1 Left Deep Tree (Plan) **QP2 Bushy Tree (Plan)**

Figure 3 : Query Plan (QP) for Query (Wu, S., et.al. 2011)

MapReduce

According to the existing work (Wu, S., et.al. 2011) (Afrati, F. N., & Ullman, J. D., 2011) a query generates two query plans which are shown in Figure 3. The first Query Plan (QP1) is called Left Deep Tree Query Plan and the second Query Plan (QP2) is called Bushy Tree (BT) Query Plan. Bushy Tree is considered to be more effective and efficient as the query execution plans generated by optimizer can be involved in parallel binary join operations. In the present era, MapReduce is based on a shared-nothing model and it leverages commodity hardware to efficiently handle data of the large-scale (Shan, Y., & Chen, Y. 2015). It is finest option to execute parallel SQL query execution plan (QEP) (Okcan, A., & Riedewald, M., 2011). Although many studies state that there is no need to split multi-way join into binary join in MapReduce frame work because only one MapReduce is sufficient to perform multi-way join as compared to using set of binary joins (Zhang, X., et al., 2012) and it also improves I/O cost and execution time of the query. Several studied have also focused on how to apply join operations in the MapReduce (Shaikh, A., & Jindal, R., 2012) (Olston, C,. et al. 2008). MapReduce doesn't support directly join algorithm. So, there are two significant concepts that are used to improve query execution time and I/O cost with the help of Two-way and Multi-way join algorithm.

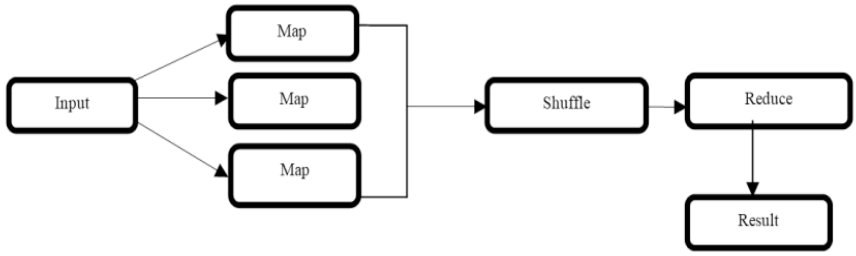

Figure 4 : MapReduce

Query Optimization

Query Optimization is an important method which always plays a vital role in query execution. There are many techniques of query optimization like Partitioning, Bucketing, Using Compression, Using ORC Format, Cost-based Optimizer and Join Optimization etc., that help to improve query performance. When queries use join operator between the multiple tables through their common attributes, join ordering will have a substantial impact on the reduction of execution costs. Query Optimizer is responsible for finding the efficient and minimum cost-effective query execution plan for speedy query execution. According to many researchers, Dynamic Programming Algorithm deals with relations efficiently when the number of joins in the join queries is usually less than 10. When the number of the relations increases the Dynamic Programming raises a problem a space and time complexity. In the big data environment, there are more than 100 tables that may be used in the join operation and hence this becomes one of the prominent reasons for proposing an effective Join Ordering Search Algorithm. Considering an example of joining three relations namely R1, R2, R3 of data size 10 tuples, 1000 tuples and 10,000 tuples, the query plan that joins R2 and R3 first take more execution time as compared to the query plan that joins R1 and R3 first (Kadkhodaei, H., & Mahmoudi, F., 2011). There are numerous of execution plans, but it will be important that the most appropriate plan is adopted, not all execution plans are equal in term of efficiency as the data size and the participating relations in the join operation increase or decrease. The Join Order Problem is extremely exigent as finding the optimal query execution plan for large scale queries is defined to be a NP-Hard problem.

Description of Join Ordering

There are four relations known as R1, R2, R3 and R4 that are used in the above-mentioned join query. Here, simple Select-Project-Join (SJP) query is considered without any aggregates and sub-queries as it involves large set of joins and SPJ queries are very time consuming. The Query plan is represented in the form of a structured tree which can be defined as either Left Deep Tree or Bushy Tree (**figure-3**). When a user enters Hive SQL query, the query parser checks the syntax and semantics of the input query and performs validation checks on the query. The input query is then forwarded to the query optimizer that generates multiple query evaluation plans in the form of the query graph. Each node in a graph represented as a relation (tables) and each edge represents a join operator between the connecting relations of the query. Through this query evaluation plan, the optimizer arranges the order of the nodes is known as Query Execution Plan (QEP). The focus of this paper is to unearth the strategies that help in generating sub-graphs from join query graph and assign an MR (MapReduce) to each sub-graph which executes all graph edges in minimum time.

OORJA
ISSN - 0974-7869 (Print)
ISSN - 2395-6771 (Online)

In addition to the Order of Joins that impacts the performance of the Input Query, the type of join operation is also considered to be an another parameter that has considerable effect and impact on the cost of final query execution. Broadly two types Join Algorithms are defined namely Two-way join algorithm (In Memory Join, Reduce-side Join, Map Side Join, Broadcast Join, Map–Merge Join, Repartition Join, Bloom Filter Join etc.) and Multi-way Join Algorithm (Reduce-Side Cascade Join, Map-Side Join, Reduce-Side One-Short Join, etc) (Thusoo, A., et al., 2010).

This paper focuses on the replicate join (Zhang, X., Chen, L., & Wang, M., 2012) that extends repartition join, and is represented as a Chain (R1 ⋈ R2 ⋈ R3 ⋈ R4) in relational algebra that support multi -way join and improve the efficiency of the query processing.

According the fig-3, there are two query plans of query1. First plan is a Left Deep Tree (LDT) and another is a Bushy Tree (BT). In QP1, it can be represented as R4 ⋈ R3 ⋈ R2 ⋈ R1in the relational algebra format and is represented as chain join because join G dependent on F, and F is dependent on E whereas QP2 is not a chain join, it can be represented as (R4 ⋈ R3) ⋈ (R2 ⋈ R1).

The Cost of (QP1) is | R4|+|R3 |+|R2|+|R|+|E|+|F|+|G|=1510 and cost of QP2 =1270.

Figure 5: Represent of the Query Plan 1 and Query Plan 2 (as shown in figure 3) In MapReduce (Chen, Y., et al., (2014)

Cost Model

The execution cost of the query with Replicate Join (Shaikh, A., & Jindal, R., 2012) (Olston, C,. et al. 2008) using MRA is defined as the summation of the MR_n, Cost (MRA)=$\sum_i$Cost(MR_i), The cost is equated as the sum of total mapper input size and shuffle data size. Total Cost of MRA={MR1, MR2, MR3,…MRn}and at the last combine all sub-queries and finds the minimum Cost (MRA). **Figure-5**, shows the two valid execution plans for query plan of **figure-3** (QP2) using MapReduce. For EP2 the sum od the total input cost is calculated as MR2 = |R1|+ |R2| + |F| = 130. To calculate the shuffle data size, the total number of times each tuple is replicated from mapper to reducer is calculated. Since, Cost (MR2) has 2 join keys so each tuple in table R1 and table F are replicated twice. Therefore the shuffle data size= 2*| R1|+|R2|+ 2*|F|= 220 and hence the total cost is Cost of MR2=350. Similarly for MR1 of EP2 in fig-4, Cost (MR1) =160, final result for the corresponding MRA, Cost (MRA) =510. The Time Complexity to search for Best Query Plan with Left Deep Tree and Bushy Tree for Polynomial algorithm is $O(n^2)$ (Vissapragoda, B., 2014). When this system is compared with others, AQUA (Ganguly, S. et al., 1992))and Hive, uses exponential algorithm for SQL queries in MapReduce. There are many scopes to improve the join order query performance through non-deterministic algorithms, metaheuristic algorithms etc.

Related Work

Several studies on Relational database have been conducted which focused on SQl Queries for Parallel Processing. These queries are first converted into executable Query Execution Plans by the Query Processor on the basis of binary join operation. (Afrati, F. N., & Ullman, J. D., 2011).

AQUA [Automatic Query Analyzer] generates the MapReduce jobs for every query to reduce the cost of query processing in two phases. In the first phase, the query is parsed into a group of join operations. Every group has many join operators and single MapReduce is used to calculate each group. In the second phase intermediate result generate the final result of the query and select the best plan which minimizes processing cost. This an optimization module that used replicated join algorithm to reduce I/O costs and it helps to avoiding to large number of intermediate results in MapReduce. In Hive, this method system represented the plan in expression tree. The Hive's analyzer receives this expression and applies metadata of tables to convert the tree into a MapReduce.

JOMR [Join request in MapReduce] (Chandar, J., 2010) algorithm focuses on the order of Join the tables in MapReduce because it helped to improve the performance of MapReduce. In the traditional database, join algorithms avoid the importance of order of tables but in many conditions, researchers observe the order of tables is important to find out the best plan of query execution. According to this method, researcher evaluates various multi join query execution plans and alters the join order of the tables and selects the minimum cost plan of join. For this reason, Travel sales problem (TSP) used MapReduce to improved join order plan and set minimum join cost. This method applies in Hive to improve the intermediate results of MapReduce by changed the order of multi join query. It uses some statistics like data size, count of rows and records etc in Hive to store the information in database to improve the query execution plan. In this paper JOMR method creates new join order of query of MapReduce. JOMR construct least cost of multi-join query by using TSP and cost between tables to evaluate and create the best order of join. This method enhanced more while ever-increasing the number of joins.

SQLMR[SQL-based and MapReduce]- Due to the increases of data, SQL based databases are not capable to handle scalability of the huge data but MapReduce is the best framework to solve large data processing and it also provide scalability and fault-tolerant. Most of the users are familiar with the SQL-like data processing so need for the solution to execute large scale data processing with SQL-based queries. SQL MR, is a hybrid approach of SQL-like databases and MapReduce data processing. The main components of SQLMR framework are: SQL-to-MapReduce, Compiler, Query Result, Partitioning and Indexing, and the Hadoop system. In this framework, a complier is transformed SQL-like queries to a MapReduce job and it stores SQL-based files directly in the HDFS. It uses minimum overhead data file construction technique to minimize conversion time between SQL and MapReduce in Hadoop system. In this framework the complier sends the query to the Manager to compare with the previous query. If the query is found in the cached, the result is directly sent to the user without re-processing the query. So, this design can reduce the pre-processing time extensively. Partition Index and B+ tree indexing techniques are used to improve the accelerate data accessing and searching in the cloud database for traditional database query. SQLMR performance also depends on the system where queries are executed. To improve Hadoop performance cross-rack application is used for optimization.

Conclusion

Multi Join Query Optimization (MJQO) plays a significant role in various application like-search engine, data mining, data warehouse and banking system etc. As the data size increases day by day and due to insufficient technology, multi join query optimization problem remains unsolved. The

Problem of MJQO include large number of join queries, high processing cost and long query execution time. There are limited studies that have been performed on join ordering for SQL-based queries on big data environment. Hive uses exhaustive algorithm for join ordering query optimization and taken long time to produce result in exponential time complexity. Basically, Hive and Pig are not providing optimize join sequence. Traditional methods like Exhaustive Search and Greedy Algorithms are not able to solve this optimization problem efficiently for Hive- SQL based queries. Therefore, there is need for the non-Exhaustive algorithms for optimal solution with better performance in Hive based SQL query.

References

- Shan, Y., & Chen, Y. (2015). Scalable Query Optimization for Efficient Data Processing using MapReduce. In 2015 IEEE International Congress on Big Data (pp. 649-652). IEEE.
- Vissapragada, B. (2014). Optimizing SQL Query Execution over MapReduce (Doctoral dissertation, International Institute of Information Technology Hyderabad.
- Chen, Y., Qin, X., Bian, H., & Chen, J. (2014). A Study of SQL-on-Hadoop Systems. Big Data Benchmarks, Performance Optimization, and Emerging Hardware-4th and 5th Workshops, pp. 154-166BPOE 2014, Salt Lake City, USA, LNCS, Vol. 8807.
- Pal, S. (2016). SQL on Big Data: Technology, Architecture, and Innovation. Apress.
- Bagui, S., &Devulapalli, K. (2018). Comparison of Hive's query Optimization Techniques. International Journal of Big Data Intelligence, 5(4), 243-257.
- Chande, S. V., & Sinha, M. (2011). Genetic optimization for the join ordering problem of database queries. In 2011 Annual IEEE India Conference (pp. 1-5). IEEE.
- Kadkhodaei, H., & Mahmoudi, F. (2011). A combination method for join ordering problem in relational databases using genetic algorithm and ant colony. In 2011 IEEE International Conference on Granular Computing (pp. 312-317). IEEE.
- Chen, M. S., Yu, P. S., & Wu, K. L. (1992). Scheduling and processor allocation for parallel execution of multijoin queries. In [1992] Eighth International Conference on Data Engineering (pp. 58-67). IEEE.
- Ganguly, S., Hasan, W., & Krishnamurthy, R. (1992). Query optimization for parallel execution. In Proceedings of the 1992 ACM SIGMOD international conference on management of data (pp. 9-18).
- Wu, S., Li, F., Mehrotra, S., & Ooi, B. C. (2011). Query optimization for massively parallel data processing. In Proceedings of the 2nd ACM Symposium on Cloud Computing (pp. 1-13).
- Afrati, F. N., & Ullman, J. D. (2011). Optimizing multiway joins in a map-reduce environment. IEEE Transactions on Knowledge and Data Engineering, 23(9), 1282-1298.
- Okcan, A., & Riedewald, M. (2011). Processing theta-joins using mapreduce. In Proceedings of the 2011 ACM SIGMOD International Conference on Management of data (pp. 949-960).
- Zhang, X., Chen, L., & Wang, M. (2012). Efficient multi-way theta-join processing using mapreduce. arXiv preprint arXiv:1208.0081.
- Shaikh, A., & Jindal, R. (2012). Join query processing in mapreduce environment. In

International Conference on Advances in Communication, Network, and Computing (pp. 275-281). Springer, Berlin, Heidelberg.

- Olston, C., Reed, B., Srivastava, U., Kumar, R., & Tomkins, A. (2008). 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).
- Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Zhang, N., & Murthy, R. (2010). Hive-a petabyte scale data warehouse using hadoop. In 2010 IEEE 26th international conference on data engineering (ICDE 2010) (pp. 996-1005). IEEE.
- Chandar, J. (2010). Join algorithms using map/reduce. Magisterarb. University of Edinburgh.