International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435(Online) Volume 7, Issue 4, April 2018)

Improving the Performance of Hive by Parallel Processing of Massive Data

Ankul Barman
Dept. of Computer Science & Engineering
Global Engineering College
Jabalpur, Madhya Pradesh, India

Deepak Paranjpe
Asst. Professor Dept. of Computer Science & Engineering
Global Engineering College
Jabalpur, Madhya Pradesh, India

Abstract- **Data refers to technologies and initiatives that involve data that is too diverse, fast-changing or massive for conventional technologies, skills and infra- structure to address efficiently is called Big data. Hive is a data warehouse infrastructure tool, well suited for query processing and data analysis. Hive is gaining popularity for its SQL like query language HiveQL and for supporting majority of the SQL operations in relational database management systems (RDBMS). Being the expensive operation in RDBMS, join has been the focus of many query optimization techniques to improve performance of database systems. This work proposes the use of Numerous Query Optimization (NQO) techniques to improve the overall performance of Hive. During parallel execution of numerous queries, many opportunities can arise for conjoint scan and/or computation tasks. Executing frequent jobs only once can reduce the total execution time of all queries remarkably. Our framework, transforms a set of interrelated HiveQL queries into new global queries that can produce the same results in remarkably smaller total execution times. It is experimentally shown that ConjointHive outperforms the conventional Hive by 30-60% reduction, depending on the number of queries and percentage of conjoint tasks, in the total execution time of correlated TPC-H queries.**

Keywords: Big Data, Hive, HiveQL, Numerous Query Optimization, ConjointHive

1. INTRODUCTION

Processing large-scale data in the amounts of hundreds of terabytes is a very difficult task. Solving the problems associated with high volume data requires can be achieved by dividing the data and work to many computers that will all work together in parallel to complete the task in a reasonable time. Map-Reduce [6] and Hadoop [1] have gained popularity in Parallel dataflow systems. These systems are extensively used for analytics and data warehousing, either directly or through the use of a high-level query language that is compiled down to a parallel dataflow graph for execution [1, 7].

Data warehousing is accompanied with batch oriented query workload that is relatively static. While ad-hoc queries must also be processed, much of the system's activity can be predicted in advance. Non-interactive nature of applications of data warehouses provides considerable freedom to reorder and optimize queries to improve overall performance.

The current parallel dataflow systems do not take advantage of these opportunities. Since Map-Reduce operations are expressed as user-defined functions therefore Hadoop performs no global analysis or optimization. Instead, a single Hadoop job is divided into smaller chunks of work called tasks, and each worker node is assigned one or more tasks and after the accomplishment of a particular task, another task is assigned by the Hadoop to execute, using some simple heuristics that try to place computations close" to their input data. However, no global analysis is performed to share work between similar jobs and to attempt to collocate data and computation, so as to predict the optimal schedule for jobs and tasks.

If we use Hive or Pig as a declarative query language for the execution platform of Map-Reduce there are more opportunities for optimization [8]. Only simple, conservative optimizations are deployed by the Current systems: for example, Pig can push distributive or algebraic aggregate evaluation beneath joins [9], and Hive applies projection and selection pushdown [4].

When users want to benefit from both MapReduce and SQL interface, then Mapping SQL statements to MapReduce tasks can become a very difficult job [9]. Hive is used to translate queries to MapReduce jobs, thereby exploiting the scalability of Hadoop, while presenting a familiar SQL abstraction [10]. These characteristics of Hive make it a suitable tool for data warehouse applications where data is not updated frequently, large scale data is analyzed, fast response times are not required [4].

The barrier of moving the applications to Hadoop is lowered by Hive which helps people who already know SQL to use Hive easily as most data warehouse applications are implemented using SQL based RDBMSs,. Similarly, Hive makes it easier for developers to port SQL-based applications to Hadoop. Since Hive processes each query independently and is based on query-at-a-time model , issuing numerous queries in close time interval decreases performance of Hive due to its execution model. From this perspective, it is important that there has not been any study that incorporates the Numerous Query Optimization (NQO) technique [10, 11] for Hive to reduce the total execution time of the queries.

2. RELATED WORKS

The concept of Numerous Query Optimization (NQO) problem was introduced in 1980's and finding an optimal global query plan by using NQO was shown to be an NP-Hard problem [15]. A large number of works was done RDBMS since then 21]. The problem of identifying common subexpressions is an NP-hard problem [15]. Therefore, M.Jarke indicates that multirelation subexpressions can only be addressed heuristically [15]. The improvement of ad hoc query by comparing an incoming query with materialized results was shown by Finkel (intermediate results and final answer) produced from earlier queries. He deals only with equivalent expressions. M.Jarke discusses the common subexpression isolation in relational algebra, domain relational calculus, and tuple relational calculus. Chakravarthy and Minker identify the equivalence and subsumption of two expressions at the logical level, using heuristics [13]. An and/or graph is used to represent queries and detect subsumption by comparing each pair of operator nodes from distinct queries by Rosenthal and Chakravarthy [11]. Another issue in MQP is that the multigraph is proposed for representing numerous Select-Project-Join type queries in [13]. This multigraph can facilitate query processing by using Ingres' instantiation and substitution [13]. 8 In [14], the multigraph was modified for representing the initial state of numerous queries.

CPU utilization, memory usage, and I/O load variables in a study during planning numerous queries to determine the degree of intra-operator parallelism in parallel databases to minimize the total execution time of declustered join methods which was illustrated by DeWitt [13]. A proxy-based infrastructure for handling data intensive applications is proposed by Beynon [14]. This infrastructure was not as scalable as a collection of distributed cache servers available at multiple back-ends. Chen et al. considered the network layer of a data integration system and reduced the communication costs by a multiple query reconstruction algorithm [16]. In recent years, a significant amount of research and commercial activity has focused on integrating MapReduce and structured databases technologies.

Mainly there are two approaches: Either adding MapReduce features to parallel database or adding databases technology to MapReduce. The second approach is more attractive because there exists no widely available open source parallel database system whereas MapReduce is available as an open source project. Furthermore, MapReduce is accompanied by a plethora of free tools as well as cluster availability and support. Hive [5], Pig [5], and HadoopDB are the projects that provide SQL abstractions (SQL-to-MapReduce translators) on top of MapReduce platform to familiarize the programmers with complex queries. Recently, there are interesting studies to apply NQO to MapReduce frameworks for unstructured data. In spite of some initial NQO studies to reduce the execution time of MapReducebased single queries, to the best of our knowledge there is no study like ours that is related to optimize the execution time of multi-queries on SQL-to-MapReduce translator tools.

3. HIVE

Hive, an open source SQL-based distributed warehouse system is proposed to solve problems mentioned above by providing SQL like abstraction on top of Hadoop framework. Hive is a SQL-to-MapReduce translator and has an SQL dialect, HiveQL, for querying data stored in a cluster . In terms of storage Hive can use any file system supported by Hadoop, although HDFS is by far the most common. Hive provides its own query language HiveQL (similar to SQL) for querying data on a Hadoop cluster. It can manage data in HDFS and run jobs in MapReduce without translating the queries into Java. When MapReduce jobs are required, Hive doesn't generate Java MapReduce programs. Instead, it uses built-in, generic Mapper and Reducer modules that are driven by an XML file representing the "job plan". In other words, these generic modules function like mini language interpreters and the "language" to drive the computation is encoded in XML. Hive Queries are translated to a graph of Hadoop MapReduce jobs that get executed on your Hadoop grid. Hive Query Language (HQL) is based on SQL, and there are many of the familiar constructs such as "SHOW", "DESCRIBE", "SELECT", "USE" and "JOIN". Similar to an RDBMS in Hive there are "Databases" that contain one or more "Tables" that contain some data defined by a "Schema".

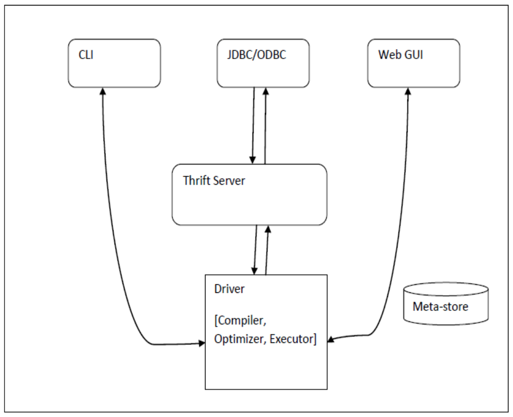

Figure 1: Hive Architecture.

3.1 HiveQL

Hive defines a simple SQL-like query language to querying and managing large datasets called Hive-QL (HQL). It's easy to use if you're familiar with SQL Language. Hive allows programmers who are familiar with the language to write the custom MapReduce framework to perform more sophisticated analysis. In the current version, HiveQL makes it possible to CREATE and DROP tables and partitions, a table split into numerous parts on a specified portion key, as well as query them with SELECT statements. Not yet supported are UPDATE and DELETE functionalities. The most important functionalities that are supported through the SELECT statements in HiveQL are

- The possibility to join tables on a common key,
- To filter data using row selection techniques
- And to project columns.

These functionalities are similar to functionality provided to the user in a relational database system. A typical SELECT statement in HiveQL would for example look like this:

SELECT o_order, o_cust, c_cust
FROM customer c **JOIN**
orders o **ON** c.c_cust = o.o_cust **JOIN**
lineitem **ON** o.o_order = l.l_order;

In this specific example, a simple join between the tables customer, order and lineitem is initiated on their respective join keys, which are cust and order. The projections in the first part of the statement are pushed down to the table scan level by the framework. Hive also supports the execution of numerous HiveQL statements during one operation, parallelizing as many tasks as possible.

The example also shows that HiveQL statements with numerous tables require specific columns on which those tables are joined. In general, there are no cross products possible in Hive as commonly supported in database management systems such as DB2 or Oracle, SQL server.

4. CONJOINTHIVE SYSTEM ARCHITECTURE

In this section, we give brief information about architecture of ConjointHive which is the modified version of Hadoop Hive with new NQO component as shown in Figure 2. Inputs to compiler-optimizer-executer are pre-processed by a Numerous Query Optimizer component which examines incoming queries and produces a single global HiveQL command to execute a group of interrelated queries. System catalog and relational database structure (relations, attributes, partitions, etc.) are stored and maintained by Metastore. Once a HiveQL statement is submitted, it is maintained by Driver which controls the execution of tasks to answer the query. First, a directed acyclic graph is produced by HiveQL to define the MapReduce tasks to be executed. Next, the tasks are executed.

4.1. Query Processing for Numerous Query Optimizations

Hive Queries are submitted through the Command Line Interface (CLI) or the Web User Interface. In the architecture we proposed, before going to driver component NQO component receives the incoming queries. The set of the incoming queries are inspected, their common tasks (redundant join processes) are detected, and merged with a global HiveQL query that answers all the incoming queries.
The driver component passes the global query to the Hive compiler that produces a logical plan using information in Metastore and optimizes this plan using a single rule-based optimizer. The execution engine receives a directed acyclic graph of MapReduce tasks and associated HDFS tasks and executes it in accordance with the dependencies of the tasks.

4.2 The ConjointHive Layer

ConjointHive is a layer before Hive "query optimizer". It takes numerous Hive queries and builds their execution plan by using Hive "query parser". This parser generates query planin a tree structure. Then ConjointHive sends numerous query plans from numerous Hive queries to ConjointHive “optimization" layer. Conjoint Hive “optimization" layer is a plug-in based layer. New optimization rules can be added as new plugin to this layer and they are used for optimization of queries. After "optimization" layer processed query plans, there are global query plans less than original query plans. Suppose there are "q" original query plans, there will be a “p" global query plans after optimization with "q>=p" condition. Unfortunately, Hive doesn't support executing numerous queries at same time. For this reason, all optimized global queries are executed as sequential. For executing an optimized global query, its query plan is sent to Hive's "'semantic analyzer'" sub-layer of "'compiler'" layer directly by bypassing "parser" sub-layer. Figure2 depicts the architecture of Conjoint Hive with new NQO component as given below.

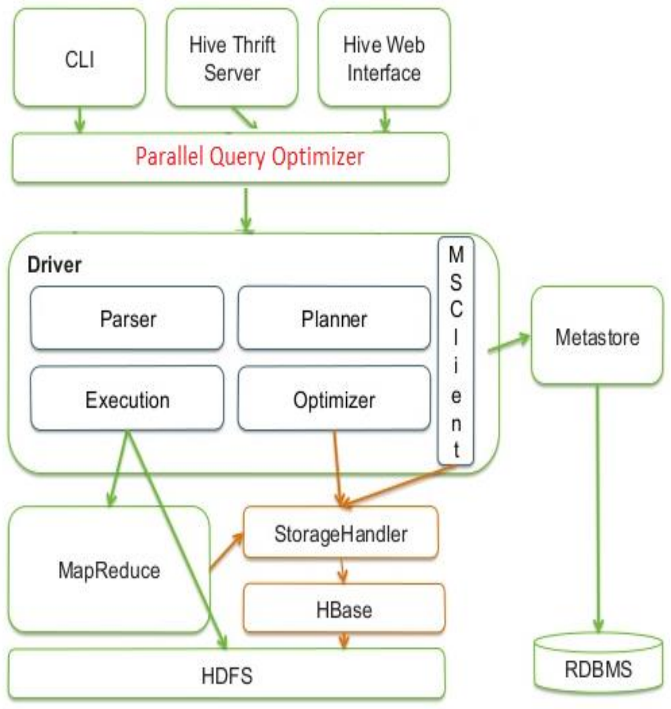

Figure 2: ConjointHive, Hadoop system architecture with NQO support.

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435(Online) Volume 7, Issue 4, April 2018)

5. NUMEROUS QUERY OPTIMIZATION ON HIVEQL

To evaluate how ConjointHive affects performance, we performed an experiment comparing a hand-optimized parallel schedule with a sequential schedule. We selected some queries from TPC-H (queries 1, 3, 6, 14, 19, 18), and arranged them into three groups. Within a group, queries were run in sequential; we then compared the performance of running groups in sequence or in parallel. We run the serial and parallel schedules, and report the total runtime for both variants in Table 1

Number of Queries	Execution time (sec)		
Data	**1GB**	**3GB**	**5GB**
Q14	151.21	163.68	177.62
Q19	68.91	74.12	78.63
Q11+Q19 (with NQO)	127.63	138.92	143.09

Table 1: Q14+Q19 Execution Results.

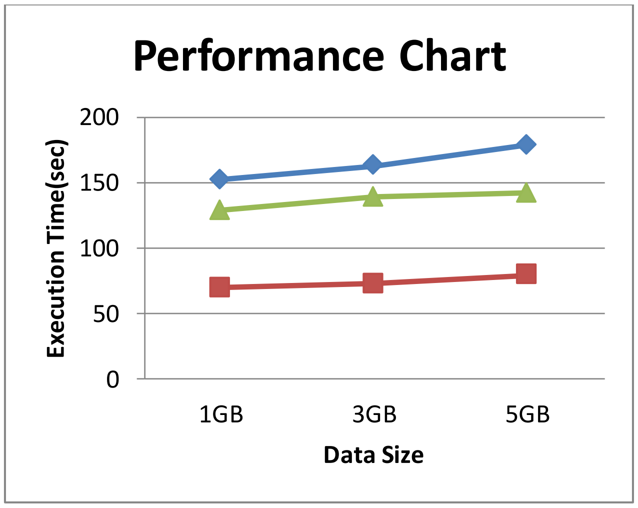

Figure 3: Performance Graph of TCP-H Queries Execution Time

Figure3 depicts performance graph of three TCP-H queries numbered as Q14, Q19 & Q14+Q19. Here the comparison has been focused between the Data size and execution time of the given queries.

ConjointHive yields approx 30% to 60% performance improvement for this test case. This is because parallel scheduling increases the utilization of the cluster: the sequential schedule leaves resources idle because the Hadoop job scheduler can only choose to schedule jobs from a single parallel-executing query. However, the magnitude of the performance advantage offered by **ConjointHive** is noteworthy, and suggests that single-user environments should consider using parallel scheduling when possible.

6. CONCLUSION

In this study, we proposed a numerous query optimization (NQO) based framework, ConjointHive, to improve the performance of conventional Hadoop Hive. In ConjointHive, we detected and categorized sets of correlated HiveQL queries and merged them into optimized HiveQL statements to run on Hadoop. With this approach, we showed that significant performance improvements can be achieved.

In this proposed work, it is possible to process only those queries that have exactly the same datasources, not partially similar datasources. If we can detect similar common tasks (such as similar FROM statements), we can merge and optimize more TPC-H queries, increasing the potential benefits that can be achieved by ConjointHive. As future work, first we plan to work on detecting correlated queries having similar datasources (FROM statements) which need not match exactly and merging them into optimized global queries.

REFERENCES

[1] Hadoop project. http://hadoop.apache.org/.

[2] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wycko_, and R. Murthy. Hive A Warehousing Solution Over a MapReduce Framework. VLDB, 2009.

[3] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Jain, S. Anthony, H. Liu, and R. Murthy. Hive A Petabyte Scale Data Warehouse Using Hadoop. IEEE, 2010.

[4] The Hive Project. Hive website, 2009. http: //hadoop.apache.org/hive/.

[5] R. Stewart. Performance and Programmability of High Level Data Parallel Processing Languages: Pig, Hive, JAQL & Java-MapReduce, 2010. Heriot-Watt University.

[6] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In OSDI '04, pages 137{150, 2004.

[7] IBM Research. Jacl website. http://www.jaql.org.

[8] C. Olston, B. Reed, A. Silberstein, and U. Srivastava. Automatic optimization of parallel dataow programs. In USENIX 2008 Annual Technical Conference, pages 267{273, 2008.

[9] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In SIGMOD '08, pages 1099{1110, 2008.

[10] Sellis, T.K. (1988). Multiple-query optimization. ACM Transactions on Database Systems (TODS), 13(1), 23-52.

[11] Bayir, M. A., Toroslu, I. H., and Cosar, A. (2007). Genetic algorithm for the multiplequery optimization problem. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 37(1), 147-153.

[12] Y. Jia and Z. Shao. A Benchmark for Hive, PIG and Hadoop,2009 https://issues.apache.org/jira/browse/HIVE.

[13] Mehta, M. and DeWitt, D.J. (1995). Managing intra-operator parallelism in parallel database systems. VLDB (382-394).

[14] Beynon, M., et al. (2002). Processing large-scale multi-dimensional data in parallel and distributed environments. Parallel Computing, 28(5), 827-859.

[15] Jarke, M. (1985). Common subexpression isolation in multiple query optimization. In Query Processing in Database Systems (pp. 191-205). Springer Berlin Heidelberg.

[16] Chen, G., et al. (2011). Optimization of sub-query processing in distributed data integration systems. Journal of Network and Computer Applications, 34(4), 1035-1042.

[17] S. Cluet and G. Moerkotte. On the Complexity of Generating Optimal Left-Deep Processing Trees with Cross Products. International Conference on DatabaseTheory, 1995.

[18] J. Dean and S. Ghemawat. MapReduce: Simpli_ed Data Processing on Large Clusters. Operating Systems Design and Implementation, 2004.

[19] A. Ganapathi, Y. Chen, A. Fox, R. Katz, and D. Patterson. Statistics-Driven Workload Modeling for the Cloud. SMDB 2010, 2010.

[20] G. Graefe. Query Evaluation Techniques for Large Databases. ACM Computing Surveys 25:2, p. 73-170, 1993.

[21] M. Jarke and J. Koch. Query Optimization in Database Systems. ACM Computing Surveys, 1984.

[22] Y. Jia and Z. Shao. A Benchmark for Hive, PIG and Hadoop, 2009. https://issues.apache.org/jira/browse/HIVE-396.

[23] B. J. Oommen and M. Thiyagarajah. Rectangular Attribute Cardinality Map: A New Histogram-like Technique for Query Optimization. Proceedings of the International Database Engineering and Applications Symposium , IDEAS'99, Montreal, Canada, 1999.

[24] V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. J. Shekita. Improved Histograms for Selectivity Estimation of Range Predicates. Proceedings of the ACM SIGMOD International Conference on Management of Data, 1996.