International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 7, Issue 1, January 2017)

The Performance Optimization of Hadoop/Hive by Distributed Search and Computation Tasks

Neetu Patel[1], Brajesh Patel[2]
[1]M. Tech., [2]Professor, Dept. of Computer Science & Engineering, SRIT, Jabalpur, India

***Abstract*--In order to process structured data we use the tool Hadoop. To make querying and analyzing easy it is put on top of big data. Execution speed of queries is enhanced by hadoop MapReduce framework. Many opportunities can arise for distribution search and/ or computation tasks during simultaneous execution of multiple queries. Therefore, we propose to use Multiple Query Optimization (MQO) techniques to enhance the overall performance of Hadoop/Hive, an open source SQL-based distributed data warehouse system based on MapReduce. 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 DistributedHive outperforms the conventional Hive by 20-50% reduction, depending on the number of queries and percentage of shared tasks, in the total execution time of correlated TPC-H queries.**

***Keywords*-- Hadoop, Hive, Multiple-query Optimization, Distributed Data Warehouse**

I. 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 is 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 multiple 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 Multiple-query optimization (MQO) technique [10, 11] for Hive to reduce the total execution time of the queries.

II. RELATED WORKS

The concept of multiple query optimization (MQO) problem was introduced in 1980s and finding an optimal global query plan by using MQO was shown to be an NP-Hard problem [15]. A large number of work was done RDBMS since then. 21]. The problem of identifying common subexpressions is an NP-hard problem [15]. Therefore, Jark 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. Jark 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 multiple 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 multiple queries.

CPU utilization, memory usage, and I/O load variables in a study during planning multiple 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 MQO to MapReduce frameworks for unstructured data. In spite of some initial MQO 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.

III. 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 . Hive can store any file system although HDFS is by far the most common. Hive doesn't generate Java MapReduce programs when MapReduce jobs are required. Generic modules function like mini language interpreters and the "language" to drive the computation is encoded in XML. 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".

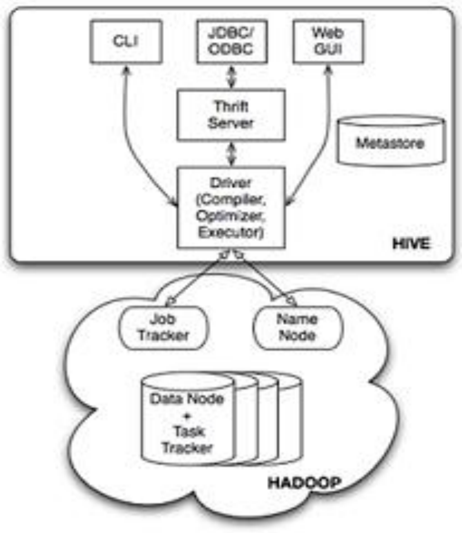


Figure 1: Hive Architecture.

A. HiveQL

HiveQL is a query language similar to SQL which is used to organize and query the data stored in Hive. It is possible to CREATE & DROP tables and partitions , split a table into multiple parts by HiveQL . UPDATE and DELETE functionalities are not yet supported. 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.

The relational functionality provided to user is same as that provided in this. A typical SELECT statement is as follows:

SELECT o_orderkey, o_custkey, c_custkey

FROM customer c **JOIN**

orders o **ON** c.c_custkey = o.o_custkey **JOIN**

lineitem l ON o.o_orderkey = l.l_orderkey;

The above query depicts join between table customer , order and line item. Hive also supports the execution of multiple HiveQL statements during one operation, parallelizing as many tasks as possible.

IV. Distridited-Hive System Architecture

In this Chapter, we give brief information about architecture of DistributedHive which is the modified version of Hadoop Hive with new MQO component as shown in Figure 2. Inputs to compiler-optimizer-executer are pre-processed by a Multiple Query Optimizer component which examines incoming queries and produces a single HiveQL command to execute a group of correlated queries. Once a HiveQL statement is submitted, it is maintained by Driver which controls the execution of tasks to answer the query.

A. Query Processing for Multiple Query Optimization

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 MQO 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 optimize 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.

B The DistributedHive Layer

DistributedHive is a layer before Hive "query optimizer". It takes multiple Hive queries and builds their execution plan by using Hive "query parser". This parser generates query plan in a tree structure. Then DistributedHive sends multiple query plans from multiple Hive queries to DistributedHive "optimization" layer. DistributedHive "optimization" layer is a plug-in based layer. New optimization rules can be added as new plug-in 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 "m" original query plans, there will be "n" global query plans after optimization with "n <= m" condition. Unfortunately, Hive doesn't support executing multiple 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

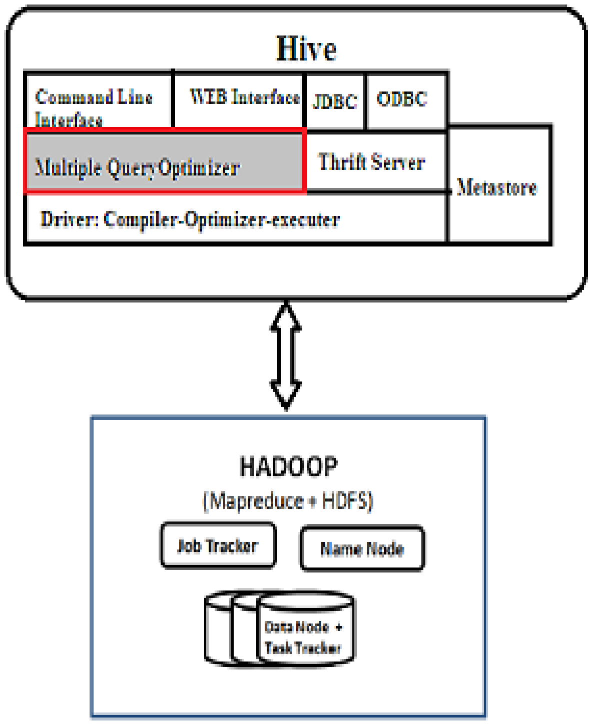


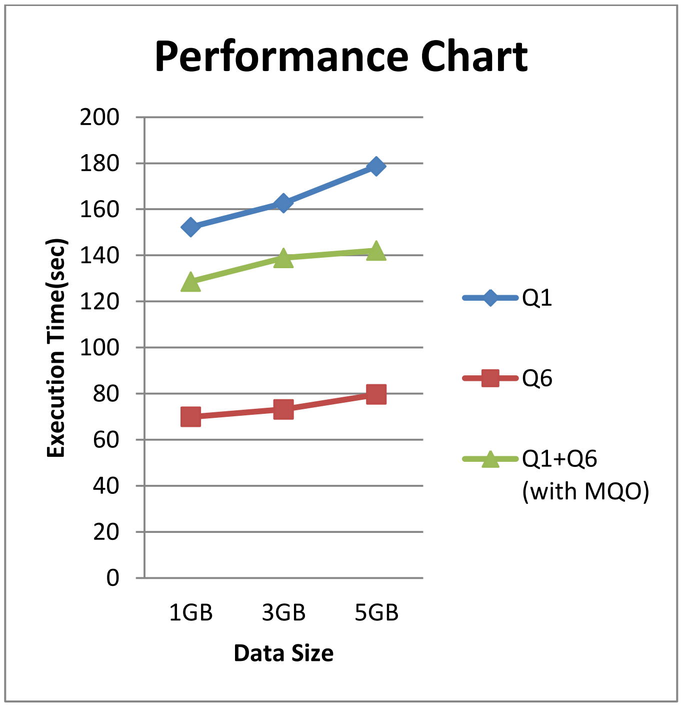
Figure 2: DistributedHive, Hadoop system architecture with MQO support.

V. MULTIPLE-QUERY OPTIMIZATION ON HIVEQL

To evaluate how **DistributedHive** affects performance, we performed an experiment comparing a hand-optimized parallel schedule with a sequential schedule. We selected 6 queries from TPC-H (queries 1, 3, 6, 14, 19, 18), and arranged them into 3 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

Table 1:
Q1+Q6 Execution Results.

Number of Queries	**Execution time (sec)**		
Data	**1GB**	**3GB**	**5GB**
Q1	152.21	162.68	178.62
Q6	69.91	73.12	79.63
Q1+Q6 (with MQO)	128.63	138.92	142.09



DistributedHive yields approx 50% 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 currently-executing query. However, the magnitude of the performance advantage offered by **DistributedHive** is remarkable, and suggests that single-user environments should consider using parallel scheduling when possible.

VI. CONCLUSION

In this study, we proposed a multiple query optimization (MQO) based framework, Distributed Hive, to improve the performance of conventional Hadoop Hive. In DistributedHive, 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 thesis, 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 DistributedHive. 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 Database Theory, 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.