Volume 4, Issue 4, October 2017 ISSN: 2348-8565 (Online)
International Journal of Modern Engineering and Research Technology
Website: http://www.ijmert.org Email: editor.ijmert@gmail.com

Performance Optimization of Hive by Two-Way Join

Manoj Dwivedi
M. Tech. Research Scholar
Takshshila Institute of Engineering & Technology
Jabalpur (M.P.), [INDIA]
Email: manojdwivedi590@gmail.com

Deepak Agrawal
Head of the Department
Department of Computer Science and Engineering
Takshshila Institute of Engineering & Technology
Jabalpur (M.P.), [INDIA]
Email:deepakagrawal@takshshila.org

ABSTRACT

Hive is disseminated data warehouse software. With the help of hive we can perform query processing and data analysis task. Hive is popular because it supports a bulk of the SQL operations in relational database management systems. To improve performance of database systems join has been the focus of several query optimization techniques. As a result the aim of this work is in two folds: Firstly we implement Two-way join technique and integrated in Hive and secondly performance is estimated, after perform join operation. we run relevant test queries on datasets generated using the industry standard benchmark, TPC-H. Our results indicate significant performance gain over highly selective queries.

Keywords:—*Big Data, Hive, two way Join Operation*

I. INTRODUCTION

Begning of Web 2.0**,** the various roles given to the users and to the web based application went through a high revolution. The passive view of the same shows that only users are the content creator. The facilities provided by the internet to the end users has dumped data from various sites like portals, blogs, social media and others, this has increased the load on the servers who was already overloaded with massive data. This change has raised the need of innovative solutions to store massive amount of data and which could also support querying over it. The raw data is queried to extract the meaningful information from it. This opens new opportunity for development of new algorithms, tools, and services to process queries over this massive amount of data in a sensible time slot. With the increase of amount of data dealt with in new and emerging applications, innovative solutions are required to not only store this massive of data, but also to process it efficiently. Hive [1] is a data warehouse solution for storing and processing such data is stored in a distributed system, Hadoop [2]. A programming model, called mapreduce [3], built on top of Hadoop system enables it to stream the data at a high bandwidth and perform massive manipulation of data. Join is an expensive operation in databases, which depending on the predicate, data, etc., allows information from different relations to be "joint". It also provides more data analysis and mining tasks important in the context of business intelligence for finding interesting and useful patterns in large amount of data. Therefore, improving various join operations can result in significant performance improvement. In relational databases, efficient join operations are supported through indexing or external sort techniques, without which the brute-force scan of the entire table is hopeless for large data. This is more crucial in particular when a small fraction of the tuples participate in a joint operation. Two major factors that influence the performance of join operations which are index based of Hive includes very high data volume and low index maintenance

cost. Though Hive is expected to work well with vast amount of data, indexing can further optimize the performance by reducing the amount of data accessed from the contributing tables. Having infrequent updates, as a characteristic of big data, makes the cost of index maintenance of less importance or affordable. Additionally, the index types proposed and developed in Hive take up a pretty small space.

II. HIVE

Hive is data warehouse software which is used for facilitates querying and managing large data sets residing in distributed storage. Hive language almost look like SQL language called HiveQL [4]. Hive is designed to enable easy data summarization. Hive also allows traditional map reduce programs to customize mappers and reducers when it is inconvenient or inefficient to execute the logic in HiveQL.

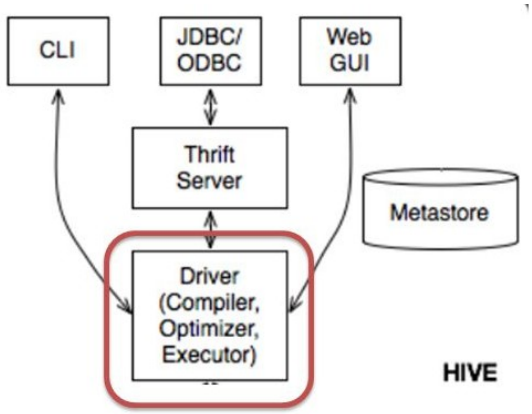

Figure 1: Hive Component

The first component is shell**;** Shell is the command line interface. It allows interactive queries like My SQL shell connected to database. Also supports web and JDBC clients [5].Driver, compiler and execution engine take the hiveql scripts and run in Hadoop environment.

The second component driver, which receives the queries. This component implements the notion of session handles and provides execute and fetch APIs modeled on JDBC/ODBC interfaces of the table and partition metadata looked up from the meta store. The third component in hive is Execution engine which executes the execution plan created by the compiler. The plan is a DAG of stages. The execution engine manages the dependencies between these different stages of the plan and executes these stages on the appropriate system components. The last component is meta store that stores all the structure information of the various table and partitions in the warehouse including column and column type information, the serializes and deserialize necessary to read and write data and the corresponding hdfs files where the data is stored.

III. Related Work

We will present a few optimization techniques related to indexes in Hive.HIVE-1644 [6] is the implementation of processing the WHERE clause with the index. The new query replaces the table with the index table and looks for the address of the desired values. The relevant part of the predicate is the part that can be processed by the indexes, that is a conjunction of the binary expressions. The re-written query is compiled and the produced root tasks are added to the original query root tasks. Then the original query is executed over the intermediate results produced from the re-written query. All column references in HIVE-1644 must refer to the same table (no joins or sub-queries).

SELECT col_list

FROM tab_name

WHERE predicate;

and re-writes it into:

INSERT INTO intermediate

SELECT _BUCKETNAME, _OFFSETS

FROM tab_index

WHERE relevent_part_of_the_predicate;

Antony, S., Chakka, 2010[1] proposed accelerates queries with GROUP BY clauses here are a number of conditions to be met in HIVE-1694: the FROM clause must have only one table (no joins) in the query; there should be only one COUNT (index_key) function in the SELECT clause; and all column references must be in the index key.

In another research, Wang et Al 2010 [7] integrated indexing with a B+ tree structure into map-reduce framework. In this work, given a query, the index is accessed twice to locate the start point and the end point in the leaves. The nodes between these two positions satisfy the query. Map jobs are generated and attached to blocks of data covered between the start point and the end point. Each map first scans the index and then retrieves the records using the offset. Gruenheid, work proposed storing column-level meta-data in Hive tables to benefit from during query execution [8]. Column-level statistics or more specifically, histograms that exhibit value distribution within a table provide more accurate information than just the table size to estimate the output size. A new table is added to Hive meta-store that holds the number of distinct values, number of null value, min and max values and most frequent values as its fields. In presence of column statistics, an index-based join can determine whether it is an optimal approach before execution.

IV. TOW-WAY JOIN APPROACH

Single tables are involved in existing indices in Hive. A join index is a pre-computed access structure that maintains pairs of identifiers of tuples from two or more relations that would match in case of a join in RDBMS. This approach would be a suitable optimization approach in Hive where tables are updated infrequently. Index join concept is based on the fact that is to keep unique identifiers of the matched tuples in the same structure and cluster them on either of the unique identifiers of both tables. The current implementation of Hive does not support the concept of primary keys [9] which are considered the unique identifiers of tuples in RDBMSs. The aim of my work is to accelerate a two-way join query created in HiveQL as shown below:

SELECT col_list

FROM tab1 **JOIN** tab2

ON (table1.col1 = table2.col1)

[**WHERE ...**]

[**GROUP BY…**];

In the above mentioned query “WHERE” and “GROUP BY” clauses are optional. The same queries can be applied for joining “n” number of tables.

A. Design

When we execute simple query in Hive, it reads the whole dataset even if we have use ‘where ‘clause filter. This becomes a bottleneck for running Map-Reduce jobs over a huge table. We can overcome this problem by using partitioning in Hive. By using automatic partition method when the table is created. In Hive’s implementation of partitioning, data within a table is split across various small partitions. When the query is executed, only the required partitions of the table are execute, thereby minimizing the I/O and time required by the query. Because when external table is declared, default table path is changed to specified location in hive metadata which contains in meta store, but about partition, nothing is changed, so, we must manually add those metadata.

The proposed research work can be demonstrated by the following:

A Search for a JoinOperator is done by optimizer. If this step is omitted we can perform optimization for any query. Now

query is examined by optimizer for a two-way join. Further we get operator TableScan Operator that points to the table that has to be manipulated and verify that the table contain an index and check for its validity. The index is valid if it is compact index and it includes all the partitions of the table. If all the condition are fulfilled then the optimizer re-write the query:

```
SELECT col_list
        FROM index_table JOIN table2
                ON (tab1.col1 = tab2.col1)
        [WHERE]
        [GROUP BY];
```

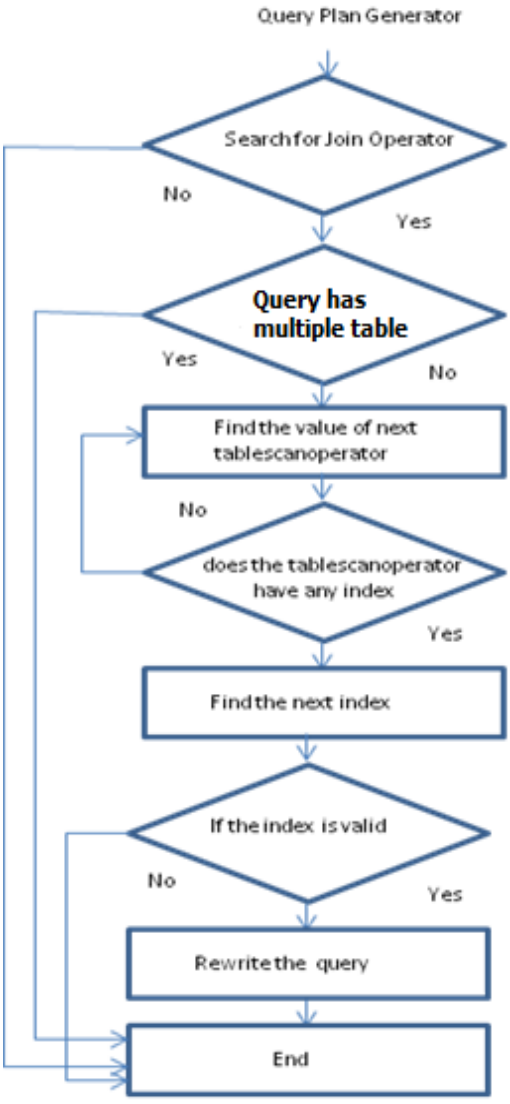

Figure: 2 Query Plan Generator Flow of Index-based approach

Otherwise This flow ends which means query is not executed successfully.

Any of the table (whichever that has the index) is replaced by its corresponding index table. This means that table must be removed from every internal data structure in the operator DAG and the new table must be added instead.

Hive query optimization is conforms by given flow show in figure 2.

B. Results

We used the standard benchmark TPC-H version 2.14.4 to generate data used in our experiments [10]. we considered only the supplier and nation tables.

C. Test Queries

selects.s_acctbal,s.s_name, s.s_suppkey from supplier s

join nation n

on(s.s_nationkey=n.n_nationkey).

select s.s_acctbal,s.s_name,s.s_suppkey from supplier s

join nation n

on(s.s_nationkey=n.n_nationkey) group by s.s_acctbal,s.s_name,s.s_suppkey

where c.c_acctbal>1000.

Select s.s_acctbal,s.s_name,s.s_suppkey from supplier s \

join nation

on(s.s_nationkey=n.n_nationkey) group by s.s_acctbal,s.s_name,s.s_suppkey

Group By s.s_acctbal,s.s_name,s.s_suppkey.

Table 1: Query1 Response Time without Index Based Approach / with Index-Based Approach

	Without Index Approach Response Time(s)			With Index Approach Response Time(s)		
Data Size	**1 GB**	**3 GB**	**5 GB**	**1 GB**	**3 GB**	**5 GB**
	51.22	146.11	200.54	34.11	100.12	143.23
	52.56	142.25	201.56	32.34	100.34	143.22
	55.78	142.34	203.54	31.67	101.23	145.32
	52.44	141.26	203.55	30.00	101.45	142.23
	52.11	140.99	201.45	30.67	101.34	142.23
Avg.	**52.82**	**142.59**	**202.128**	**31.758**	**100.896**	**143.246**

Table 2: Query1 Response time without index based approach /with index-based approach

	Without index approach response time(s)			With index approach response time(s)		
Data Size	**1 GB**	**3 GB**	**5 GB**	**1 GB**	**3 GB**	**5 GB**
	54.22	141.11	193.54	32.11	110.12	133.23
	53.56	142.25	191.56	32.34	110.34	133.22
	53.78	142.34	191.54	31.67	111.23	135.32
	52.44	141.26	191.55	31.02	111.45	132.23
	52.11	140.99	191.45	31.23	111.34	132.23
Avg.	**53.11**	**141.59**	**191.928**	**31.674**	**110.896**	**133.246**

Table 3: Query1 Response Time without Index based Approach / with Index-Based Approach

	Without index approach Response time(s)			With index approach Response time(s)		
Data Size	**1 GB**	**3 GB**	**5 GB**	**1 GB**	**3 GB**	**5 GB**
	50.60	147.66	265.00	32.11	101.56	210.54
	49.81	146.87	265.78	32.45	101.43	211.34
	49.43	150.98	265.65	32.12	101.24	210.11
	49.76	147.01	265.78	32.43	101.23	210.45
	49.76	147.00	265.71	32.10	101.45	210.23
Avg.	**49.87**	**147.90**	**265.58**	**32.24**	**101.38**	**210.53**

Figure 3: Query1 response time without index based / with index-based approach on single node setup

Figure 4: Query2 response time without index based / with index-based approach on single node setup

Table 3: Query1 Response time without index based approach /with index-based approach

V. CONCLUSION AND FUTURE WORK

The expensive operation in RDBMS, join has been the focus of many query optimization techniques to improve performance of database systems. In partitioning, we create a partition for each unique value of the column. We investigate such techniques for join operations in Hive and develop an index-based join algorithm for queries in HiveQL. Indexes have been around for long time and the benefit of using them is obvious Though index size depends on the data distribution and the number of attributes for indexing, our experiments showed the Hive index space utilization is reasonable. With respect to accessing the index, current Hive indexes do not provide an instant access to values, which undoubtedly comes with heavy space overhead. What they offer instead is, scanning a huge amount of data is replaced with scanning a drastically small set of it that holds the desired values. Hive index maintenance cost is noticeably low, considering the infrequent updates and batch-mode data insertion as the characteristics of big data. The indexing technique in Hive is rather new and the progress has been limited to current index structure and also the query life cycle. As future work, first we plan to work on hash based indexing using bucket level because bucket is smallest data model in hive.

REFERENCES:

[1] Antony, S., Chakka, P., Jain, N., J., Liu, Murthy, R., Sarma, J. S., Thusoo, A., Zhang, N "Hive – A Petabyte Scale Data Warehouse Using Hadoop," IEEE 26th Intl. Conf. Data Engineering (ICDE), Long Beach, CA, 2010, pp. 996 – 1005.

[2] Apache Hadoop [Online]. Available: http://hadoop.apache.org

[3] Dean, J., Ghemawat, S. "MapReduce: Simplified Data Processing on Large Clusters," Mag. Commun. ACM 50thanniversary, vol. 51, issue 1, 2008, pp.107-113

[4] http://www.hadooptpoint.com/introduction-hive

[5] Yue Liu1, 6, 7, Songlin Hu1 "DGFIndex for Smart Grid: Enhancing Hive with a Cost-Effective Multidimensional Range Index" 40th International Conference on Very Large Data Bases, September 1st - 5th 2014, Hangzhou, China.

[6] ANTLR [Online]. Available: http://www.antlr.org/

[7] An, M., Wang, W., Wang, Y., "Using Index in the MapReduce Framework,", 12th Intl. Asia Pacific Web Conf. (APWEB),Beijing, China, 2010, pp. 52-58

[8] Dean, J., Ghemawat, S. "MapReduce: Simplified DataProcessing on Large Clusters," Mag. Commun. ACM 50th anniversary, vol. 51, issue 1, 2008, pp.107-113.

[9] Capriolo, E., Rutherglen, J., Wampler, D. Programming Hive: Data Warehouse and Query Language for Hadoop, 1st ed, O'Reilly Media, 2012

[10] TPC-H[Online]. http://www.tpc.org/tpch/

[11] HIVE 1694[Online]. Available: https://issues.apache.org/jira/browse/HIVE-1694

[12] Hive index design doc [Online]. Available: https://cwiki.apache.org/confluence/display/Hive/IndexDe

[13] Hive JIRA [Online]. Available: https://issues.apache.org/jira/browse/

HIVE

[14] HIVE-1644 [Online]. Available: https://issues.apache.org/jira/browse/HIVE-164

[15] N. Jain, L. Tang, "Join strategies in Hive", Facebook, Rep. Hadoop summit 2011, 2011 [Online].

[16] Li, Z., Ross, K. A. "Fast joins using join indices", in The International Journal on Very Large Data Bases, vol. 8, issue 1, 1999, pp.1–24

[17] Lou, W., Ren, K., Wang, C., Wang, Q. Privacy-Preserving Public Auditing for Storage Security in Cloud Computing, Proc. 30th IEEE Int'l Conf. Computer Communications (INFOCOM 10), IEEE Press, San Diego, CA, 2010, pp. 525–533.

[18] S. Madden: "From Databases to Big Data," IEEE Internet Computer., vol.16, issue 3, pp. 4-6, May-June, 2012

[19] MapReduce Tutorial [Online]. Available (date): http://hadoop.apache.org/docs/mapreduce/r0.22.0/mapred_tutorial.html

[20] MongoDB[Online]. Available: http://www.mongodb.org/

[21] Neo4j[Online]. Available(write date): http://www.neo4j.org/

[22] Gruenheid, A., Mark, L., Omnecinski, E. "Query Optimization using column statistics in Hive," in Proc. 15th Symp. Intl. Database Engineering & Applications (IDEAS), Lisbon, Portugal, 2011, pp. 97-105, 2011

[23] Dean, J., Ghemawat, S. "MapReduce: Simplified Data Processing on Large Clusters," Mag. Commun. ACM 50th anniversary, vol. 51, issue 1, 2008, pp.107-113

[24] Eaton, C., Deroos, D., Deutsch, T., Lapis, G., Zikopoulos, P. Understanding Big data: Analytics for Enterprise Class Hadoop and Streaming Data, 1st ed, McGraw, 2011

[25] Garcia-Molina, H., Ullman, J., Widom, J. Database Systems: The Complete Book, 1st ed, Upper Saddle River, NJ, Prentice Hall Inc., 2002

[26] Gilbert, S., Lynch, N. A. "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services", in Newslett. ACM SIGACT, vol. 33, issue 2, pp. 51-59, June 2002

[27] Grance, T., Mell, P. "The NIST Definition of Cloud Computing," - NIST. Gaithersburg, MD, Rep. Recommendations of the National Institute of Standards and 107 Technology, 2011

[28] Yin Huai, Ashutosh Chouhan "Major Technical Advancements in Apache Hive" SIGMOD, June 2014

[29] H. V. Jagadish : "Big Data & Science Myths and Reality" Science Direct June 2015.

* * * * *