International Journal of LNCT
Volume-4 Issue-19 August -2020
ISSN: (Online):2456-9895

OPTIMIZING THE JOIN OPERATION PERFORMANCE OF HIVE MAPREDUCE USING DISTRIBUTED CACHE

Rajesh Kumar
Mtech Scholar
LNCT, Bhopal,India

Dr. Manish Shrivastava
Director R&D
LNCT, Bhopal,India

Sushil Kumar
Professor
LNCT, Bhopal,India

***Abstract*— Today when the internet technology is growing rapidly so, there is a need to store and process huge volume of data. The growing technology is generated data from which only 20% of data is in structured form and the remaining 80% of data is in unstructured form which is known as a big data problem. So there is a need of technology that can manage the big data efficiently. The Apache Hadoop is a framework that allows for the distributed processing of large data sets across machines. The Hadoop having two modules 1. Hadoop distributed file system and 2. Map Reduce. By using the hive query language on the Hadoop and increasing number of nodes the data will be processed fastest than with the fewer nodes. In this we explore MapReduce Join operation is used to combine two large datasets. Joining two datasets begins by comparing the size of each dataset. If one dataset is smaller as compared to the other dataset then either Mapper uses the smaller dataset to perform a lookup by Reducer for matching records from the large dataset and then combine those records to form output records. In this paper, we proposed a distributed cache , so the smaller dataset is stored into cache memory of every mapper node and all the lookups are perform by mapper to produce a final records. So by using distributed cache we don't use reducer means data load will be managed efficiently at mapper ends which produces a overall better performance as compared to normal join.**

Keywords*— *Big data, Hadoop, Mapreduce, performance, Hive, Query Optimization, joins Optimization, distributed cache.

I.INTRODUCTION

Big Data is that the term used for the gathering of dataset so large and sophisticated that it becomes difficult to process using traditional management or processing tools. Gartner defines Big Data in terms of three V's that's Volume, Velocity and Variety, that are the assets require for processing data [1]. Additionally, there are more V's within the literature like Validity, Veracity, Visibility, Value added by some researchers to elucidate Big Data more clearly [2]. There are sorts of applications and tools developed by various organizations to process and analyze Big Data. Hadoop is an open source Map- reduce project funded by Yahoo, emerged in year 2006 is employed to process Exabyte or Zettabyte of knowledge on cluster of commodity hardware connected by ethernet cables [3][4]. Hive may be a data warehousing tool of Apache Foundation built on top of Hadoop distributed framework wont to process and query data which is stored in HDFS during a similar manner as of traditional management system (RDBMS). Hive initially developed by Facebook, is now used and developed by other

companies like Netflix [5]. As Hadoop distributed file solution has been the simplest solution for parallel, execution and aggregating flat files, Hive also uses HDFS for storage and offers convenient data retrieval to users as if they were using traditional database engine [6]. Hive uses Derby Language (No-RDBMS schema) to process unstructured data as if it were structured. Hive process data and stores in sorts of tables and partitions, which may be accessed employing a Hive specific command language called HiveQL almost like SQL which is definitely handled by the people conversant in traditional management system. Since Hive is comparatively young project, query optimization may be a topic that comes into focus because Hive remains not during a stable state. On the Apache website [7], all recent releases are available which may be downloaded from mirrors, not necessarily during a stable state. As a result developers are enhancing the performance of Hive at this development phase. Performance of any database engine are often measured by its reaction time and amount of labor done by it. At an equivalent time, this language also allows programmers who are conversant in the MapReduce framework to plug their own mappers and reducers programs to perform more sophisticated analysis which will not be supported by the built-in capabilities of the language.

II. LITERATURE REVIEW

According to [1], Hadoop may be a widely adapted open-source map reduce implementation for storing and processing extremely large data sets. However, using Hadoop isn't easy for end-users, especially for those that weren't conversant in the map-reduce approach. Even for easy tasks, like getting raw counts or averages, users need to write map-reduce programs. Apache Hive, which may be a data warehouse infrastructure tool to process structured data in Hadoop helps users to simply query, summarize and analyze Big Data with SQL-like expressions called HiveQL. It supports various file formats to import and export data from-and-to the storage filing system . Hive's objective is to form it easy and performative when petabytes levels of knowledge must be processed. Unlike RDBMS, Hive stores data during a document-based structure so queries with JOINS degrade performance with high resource consumption. However, there are ways to enhance performance for the relational data by correctly configuring Hive. during this research, we implement several optimization techniques to enhance the query performance and evaluate the results to match their impact on the result .

In [2], With the advancement of ever-growing online services, distributed Big Data storage i.e. Hadoop, Dryad gained far more attention than ever and therefore the fundamental requirements like fault tolerance and data availability become the concern for these platforms. Data replication policies in Big Data applications are shifting towards dynamic approaches based on the recognition of files. Formulation of dynamic replication factor paved the way of solving the problems generated
by existing data contention in hotspots and ensuring timely data availability. But from the empirical observations, it can be deduced that popularity of files is temporal instead of perpetual in nature and, after a particular period, content's popularity ceases most of the time which introduces the I/O bottleneck of updating replication within the disk.

In [3], Indexing provides fast searching over big data. It is a data structure technique to efficiently retrieve records from the database files supported some attributes on which the indexing has been done. With the assistance of indexing, queries usually leads to far better
performance. Hive is batch-oriented data warehouse software. With the help of hive we will perform query processing and data analysis task. Hive is popular because it supports a bulk of the SQL operations in electronic database management systems. to enhance performance of database systems join has been the main target of several query optimization techniques. As a result the aim of this work is in two folds: Firstly we implement index based join technique and integrated in Hive and secondly performance is estimated, after perform join operation.

III PROBLEM DEFINITION

To store, retrieve and analyze Big data in an efficient way using relational database. The Hadoop architecture encompassing Hive, HDFS and Mapreduce is chosen as the relational database. The data to be queried has to be loaded into the hdfs storage space provided by Hadoop. The database schema has to be created and the data should be loaded into the database used by Hive which provides the interface for the SQL like syntax. The queries have to be executed and the execution times are noted down.

MapReduce is a popular programming model for executing time-consuming analytical queries as a batch of tasks on large scale data clusters. In environments where multiple queries with similar selection predicates, common tables, and join tasks arrive simultaneously, many opportunities can arise for sharing scan and/or join computation tasks which will take memory loading and due to which overall performance degrades.

IV PROPOSED WORK

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. To improve Join operations performance a distributed cache is proposed , so the smaller dataset is stored into cache memory of every mapper node and all the lookups are perform by mapper to produce a final records. So by using distributed cache we don't use reducer means data load will be managed efficiently at mapper ends which produces a overall better performance as compared to normal join figure 1 shows the block diagram of proposed work.

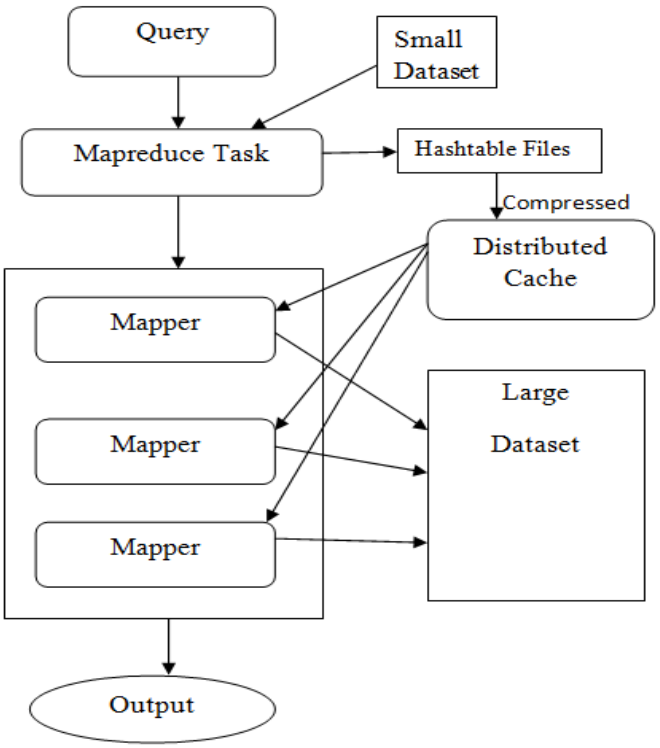

Figure 1. Proposed Flow Diagram

Step 1: first we collect dataset and load the dataset into HDFS in which the dataset is stored in various blocks. If queries frequently depend on small table joins, usage of map joins speed up queries' execution.

Step 2: before the original MapReduce task, its first step is to create a MapReduce local task. However, from HDFS this map/reduce task read data of the small table. Further save it into an in-memory hash table, then into a hash table file. Afterward, it moves the hash table file to the Hadoop Distributed Cache.

Step 3: while original join MapReduce task starts, which will populate the file to each mapper's local disk. Hence, in this way, all the mapper can load this hash table file into the memory and then do the join in Map stage.

Step 4: In this step we are analysing the time taken by the queries.

V EXPERIMENTAL & RESULT ANALYSIS

All the experiments were performed using an i5-2410M CPU @ 2.30 GHz processor and 4 GB of RAM running, and vmware is installed on it and on this vmware we can

configure ubuntu 14 on this vmware with 2 GB RAM is allocated to the vmware. After that we can configure hadoop in pseudo distributed mode in which all the hadoop daemons are running on a single machine and for joining we can use hive to generates a efficient mapreduce code for generation output. After that we can load the different size datasets into the HDFS by using hadoop put command. First we create a two table on which we perform a join operation and the table schema of both the tables are shown in figure 2.

Figure 2. Table schema of both tables

After creating table we can load the data into the tables and then we retrieve the data from these two table by joining query. First we can launch the query over hadoop without distributed cache memory and the query we can launched and the mapreduce launches the jobs are shown in figure 3.

Figure 3. Query on default Hadoop Configuration

When the query is launched the mapreduce framework converts the query into mapreduce programs and when the program executed we can seen that total 2 mappers and 1 reducers task is launched by mapreduce to produce the final output which is shown in figure 4.

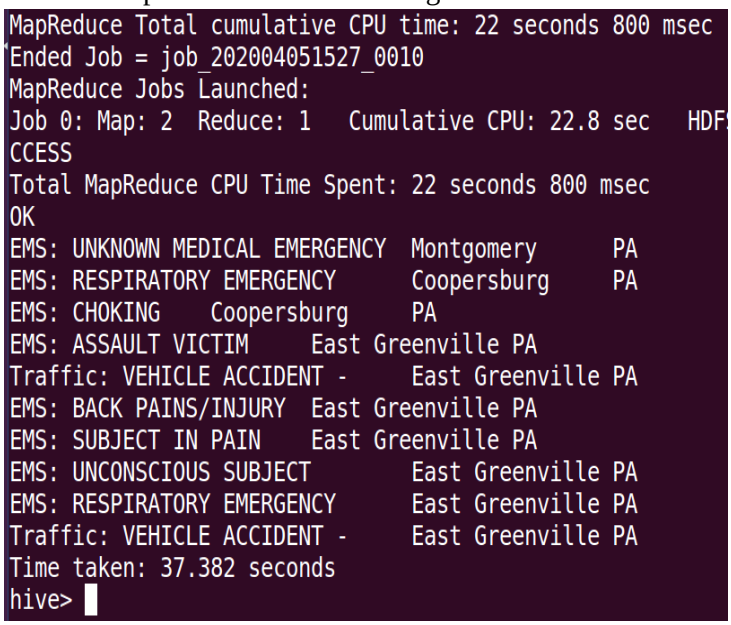

Figure 4. Output of Query

In the above figure we can get the total time taken by the query to produce the final output. Now we can implement a distributed cache memory in hadoop and once we enabled the configuration property of hadoop through which mapreduce framework knows details about cache memory. So whenever we load the data small dataset will automatically loaded into cache memory of hadoop. Now we can launch the same hive query on hadoop with enabled distributed cache memory which is shown in figure 5.

```
Total MapReduce jobs = 3
Ended Job = 714568077, job is filtered out (removed at runtime).
Ended Job = 917679945, job is filtered out (removed at runtime).
WARNING: org.apache.hadoop.metrics.jvm.EventCounter is deprecated. Please use org.a
oop.log.metrics.EventCounter in all the log4j.properties files.
Execution log at: /tmp/hadoop/hadoop_20200405170404_acc7b6a5-2f31-4980-9133-593ac0e
2020-04-05 05:04:20     Starting to launch local task to process map join;      max
ry = 932118528
2020-04-05 05:04:23     Processing rows:        43191   Hashtable size: 43191   Mem
:       13929176        rate:   0.015
2020-04-05 05:04:23     Dump the hashtable into file: file:/tmp/hadoop/hive_2020-04
-16_459_5052757955543093541/-local-10002/HashTable-Stage-3/MapJoin-mapfile01--.hash
2020-04-05 05:04:26     Upload 1 File to: file:/tmp/hadoop/hive_2020-04-05_17-04-16
757955543093541/-local-10002/HashTable-Stage-3/MapJoin-mapfile01--.hashtable File s
952
2020-04-05 05:04:26     End of local task; Time Taken: 5.685 sec.
Execution completed successfully
Mapred Local Task Succeeded . Convert the Join into MapJoin
Mapred Local Task Succeeded . Convert the Join into MapJoin
Launching Job 2 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_202004051527_0014, Tracking URL = http://localhost:50030/jobdeta
obid=job_202004051527_0014
Kill Command = /home/hadoop/work/hadoop-1.1.2/libexec/../bin/hadoop job  -kill job_
527_0014
```

Figure 5. Query on Cache Hadoop Configuration

When the query is launched the mapreduce framework converts the query into mapreduce programs and when the program executed we can seen that total 3 mappers and 0 reducers task is launched by mapreduce to produce the final output which is shown in figure 6.

```
527_0014
Hadoop job information for Stage-3: number of mappers: 1; number of reducers
2020-04-05 17:04:32,581 Stage-3 map = 0%,  reduce = 0%
2020-04-05 17:04:38,799 Stage-3 map = 100%,  reduce = 0%, Cumulative CPU 4.6
2020-04-05 17:04:39,805 Stage-3 map = 100%,  reduce = 0%, Cumulative CPU 4.6
2020-04-05 17:04:40,811 Stage-3 map = 100%,  reduce = 0%, Cumulative CPU 4.6
2020-04-05 17:04:41,817 Stage-3 map = 100%,  reduce = 100%, Cumulative CPU 4
MapReduce Total cumulative CPU time: 4 seconds 630 msec
Ended Job = job_202004051527_0014
MapReduce Jobs Launched:
Job 0: Map: 1   Cumulative CPU: 4.63 sec   HDFS Read: 4329 HDFS Write: 381 S
Total MapReduce CPU Time Spent: 4 seconds 630 msec
OK
EMS: BACK PAINS/INJURY  Gilbertsville   PA
EMS: DIABETIC EMERGENCY Lansdale        PA
Fire: GAS-ODOR/LEAK     Norristown      PA
EMS: CARDIAC EMERGENCY  Norristown      PA
EMS: HEAD INJURY        Lansdale        PA
EMS: NAUSEA/VOMITING    Horsham PA
EMS: RESPIRATORY EMERGENCY      Collegeville    PA
EMS: SYNCOPAL EPISODE   Harleysville    PA
Traffic: VEHICLE ACCIDENT -     Plymouth Meeting        PA
Traffic: VEHICLE ACCIDENT -     Conshohocken    PA
Time taken: 25.495 seconds
hive>
```

Figure 6. Output & Time taken by query

In Figure 6 it is clear that query takes less time when we implement a distributed cache which tables more mapper but no reduce task which means the smaller dataset is loaded into cache memory and lookups are perform by mapper not from reducer so by using cache memory lookups will be faster and overall time taken of query is less, we can launch various queries and the time taken we have got are shown in table 1.

Time Taken in Seconds	Without Distributed Cache	With Distributed Cache
Query-1	37.38	25.49
Query-2	87.85	78.80
Query-3	49.26	45.99

Table-1. Time taken by Queries

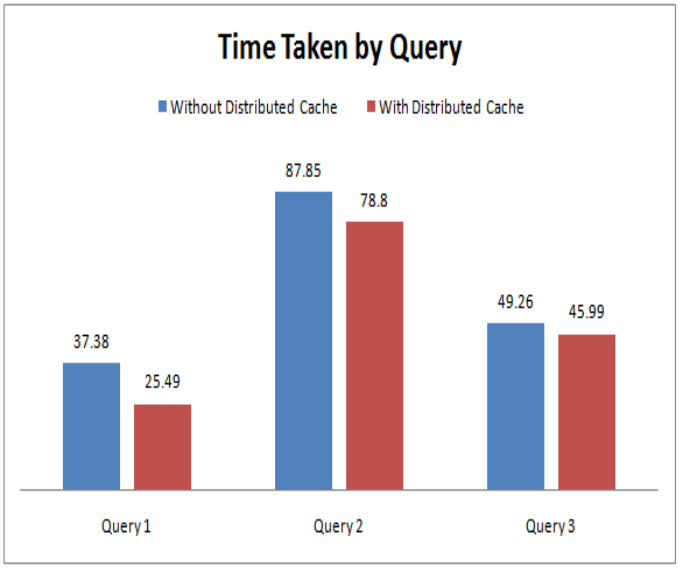

Figure 7. Time taken by Queries

VI CONCLUSION

We have executed and reviewed various queries implementation on Hive for large datasets by using query implementation techniques like Join operations. Mapping and reducing functionalities of Map-reduce and HDFS helped Hive to process datasets. In this paper, we proposed a distributed cache , so the smaller dataset is stored into cache memory of every mapper node and all the lookups are perform by mapper to produce a final records. So by using distributed cache we don't use reducer means data load will be managed efficiently at mapper ends which produces a overall better performance as compared to normal join. In future we can perform operation on fully distributed cluster mode.

REFERENCES

[01] Melih Gunay ; M. Numan Ince ; Alper Cetinkaya' "Apache Hive Performance Improvement Techniques for Relational Data" in IEEE 2019.

[02] Pinchao Liu, Adnan Maruf, Farzana Beente Yusuf, Labiba Jahan, Hailu Xu, Boyuan Guan, Liting Hu, Sitharama S. Iyengar "Towards Adaptive Replication for Hot/Cold Blocks in HDFS using MemCached" in IEEE 2019.

[03] Akshay Kumar Suman, Dr.Manasi Gyanchandani "Improved Performance of Hive using Index-Based Operation on Big Data" in IEEE 2018

[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/Index De

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

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

[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