International Journal of Innovations & Research Analysis (IJIRA)
ISSN :2583-0295, Impact Factor: 5.449, Volume 03, No. 01(II), January- March, 2023, pp 47-54

JOIN ORDER QUERY OPTIMIZATION USING HIVE IN THE BIG DATA ENVIRONMENT

Ms. Nisha Jain*
Dr. Preeti Tiwari**

ABSTRACT

In the Big Data world, billions of internet users use various social platforms and generate large digital data from different resources every second. The main drawback of conventional database systems is the inability to manage huge and complex datasets along with big data challenges, measured in terms of petabytes, zettabytes so on. New mechanisms are needed to store and process this big data. Apache Hadoop framework uses the MapReduce programming model to handle big data that gave great performance by developing parallelism among processing nodes, which is utilized by many companies like Yahoo, Facebook, and so on. MapReduce has a few limitations for data analysis; it requires a high-level language like Hive for processing large datasets. Apache Hive is an open-source engine assembled on top of Hadoop that uses the HiveQL Structured Language, similar to relational SQL for query processing. Join is the most frequent and most expensive operation in Hive query processing which increases the processing cost and time. Optimization plays an important role to enhance query processing. Although join order query optimization can greatly improve the Hive query processing speed and cost, it is avoided by most researchers. This paper examines the different join-order techniques to find the optimal Query Execution Plan (QEP) with minimum cost in a large search space to increase query performance on Hadoop-Hive. We will also discuss the feasibility and limitations of the join order strategies for Hive queries which will assist the researcher in interesting directions for a highly efficient join order processing system in the future.

Keywords: *Big Data, Hadoop, Hive, Join Order Query Optimization Technique.*

Introduction

Billions of users around the world use mobile phones, internet, social media and generate huge amounts of data every second. As per an International Data Corporation (IDC) article [24], about 19 zettabytes of data were reached in 2018 and an expected 163 zettabytes will be reached in 2025. This huge range of heterogeneous digital data is called Big Data is defined as a 3'V that is Volume, Velocity and Variety and is measured in terms of terabytes or petabytes. This heterogeneous data may include structured, unstructured and semi-structured data. Earlier to Big Data, many industries and organizations were based on classical database management systems, like Facebook'sRDBMS-based data warehouse[1], to collect and evaluate their data. With respect to Big Data, such frameworks are lacking in handling 700 TB datasets and suffer from poor performance and limited storage space. Therefore, to overcome the limitation of these frameworks various industries and academia have developed several frameworks such as Google MapReduce[15], Yahoo, PNUTS, Microsoft SCOPE and Apache Spark[20].

Hadoop's Map-Reduce [15], one of the most popular programming models in Big Data tools, works on shared-nothing cluster-based commodity hardware for tremendous data handling [6] that gives better performance than Microsoft SQL Server when the data size increases, and it also reduces the hardware cost. That's the reason, many vendors of traditional warehouses and databases move towards Hadoop and its ecosystem for key features like data analysis, fault tolerance, scalability, and flexibility etc. Map-Reduce[7][8] is the java-based programming software on the Hadoop system which is designed

* Faculty, Department of Computer Science, S.S. Jain Subodh P.G. Mahila Mahavidyalaya, Jaipur, Rajasthan, India.
** Department of Computer Science, Associate Professor, IIIM, Jaipur, India.

for processing large-scale datasets in a distributed manner. Map-Reduce [20]programming framework uses map and reduce two functions to process large datasets. The map function takes data from the input file and split it into small forms and completes the process of shuffling and sorting on nodes before sending the data to the reduce function. The reduce function collects the output from the map function as an input and produced the result. This type of Map-Reduce programming creates new challenges for end-uses or programmers who aren't knowledgeable about tuning the Hadoop system because mostly programmers are familiar with the SQL[25] for data analytics tasks like filtering, joining, aggregation, grouping, sorting, and so on. than using the Hadoop framework. To resolve these challenges, recently some high-level languages for MapReduce have been offered like Pig[3] and Hive[2], and Hive has a shorter query processing time than Pig[9].

Apache Hive is the first open-source ETL and data warehousing coined by Facebook in 2007, [4]on top of the Hadoop ecosystem that is intended to process and analyze huge volumes of data. Many companies like eBay, LinkedIn, Facebook, Taobao, Spotify and Yahoo! [10] uses Hive for data analytics and is also well-suited for batch processing. Currently, above 95% of Facebook jobs are produced by Hive[11]. Hive[25] supports the same query functionality as Relational Database Management System (RDBMS) like- tables, partition, etc. Hive [13] involves HiveQL as a SQL-like declarative language for query processing, which is compiled into a directed acyclic graph of map-reduce jobs to be executed on Hadoop, which stores the processed data in the Hadoop Distributed File System (HDFS). When the user sends their join queries to the corresponding query language HiveQL for evaluation, Sometimes the Hive system does not always perform well and results in intermediate data and execution time increased.

JOIN [25] is one of the most important, costly, and frequently used tasks in query processing that joins at least two relations (datasets) considering predefined conditions. This increases the I/O cost and execution time of the join queries. Therefore, it would require an efficient query optimization strategy to generate an optimal query execution plan for join queries in Hive. So, the challenge of finding a good optimization strategy for join operations is two-fold- First, to develop a good Join Algorithm for performing the join itself. Second- Join Order Algorithm that determines an appropriate join sequence in which joins are to be performed, minimizing the cost and time. This study will focus on an efficient join order as incorrect join ordering can lead to sub-optimal solutions rather than optimal solutions because as the relations in the query increase, the joins also increase in the Hive query. Therefore, there is required for the best optimization techniques that are able to hold complex join operations and find good plans that use less memory and improve execution time for large joins.

Join Order query optimization in Hive can be done by two approaches-First, Rule-based optimization [14] in MapReduce is like that of a traditional RDBMS, firstly every Hive query is represented as an expression tree then it transforms into a more efficient query plan. This transformation is led by a heuristic optimization rule [1] but rule-based optimization has a problem that is not always performed efficiently. So, this promoted the researchers to implement cost-based optimization and adaptive optimization techniques in Hive to come up with more efficient plans. Cost-based optimization calculates the cost of all possible query plans and chooses the lowest-cost plan similar to that of the query processing in RDBMS.

This paper is concerned with the multi-join order query optimization algorithms and will try to determine the best direction for the researcher to future betterment based on the literature review. The remains of this paper - Section II, Brief Introduction of Hive Architecture and HiveQL. Section III, Query Optimization in Hive. Section IV, Related Work, Section V & VI, Conclusion and References

Hive Architecture and HiveQL

Apache Hive is an open-source data warehousing that acts as a connection in the middle of the data analysts and the Hadoop-Hive ecosystem. Hive [1], also supports complex analytics, structure queries, window queries, etc. It uses HiveQL (Hive Query Language) similar to the SQL- like declarative language of RDBMS for query processing, which is compiled into map-reduce jobs. HiveQL is based on the SQL-92 standard[25] which makes it easy to migrate any RDBMS-based data analytic application to Hive. In addition, HiveQL supports primitive data types, arrays and maps, and nested compositions etc.

- **Hive Architecture:** External user interface involving the Command-line interface (CLI), Web UI, Thrift, and application programming interfaces (API) like ODBC and JDBC. The driver controls the processing of queries and coordinates with the compiler, optimizer and execution. The metastore is a repository system that contains the metadata to store the information about Hive tables and partitions in the data warehouse; users can access this information by API and it uses HDFS as disk storage for Hive metadata.

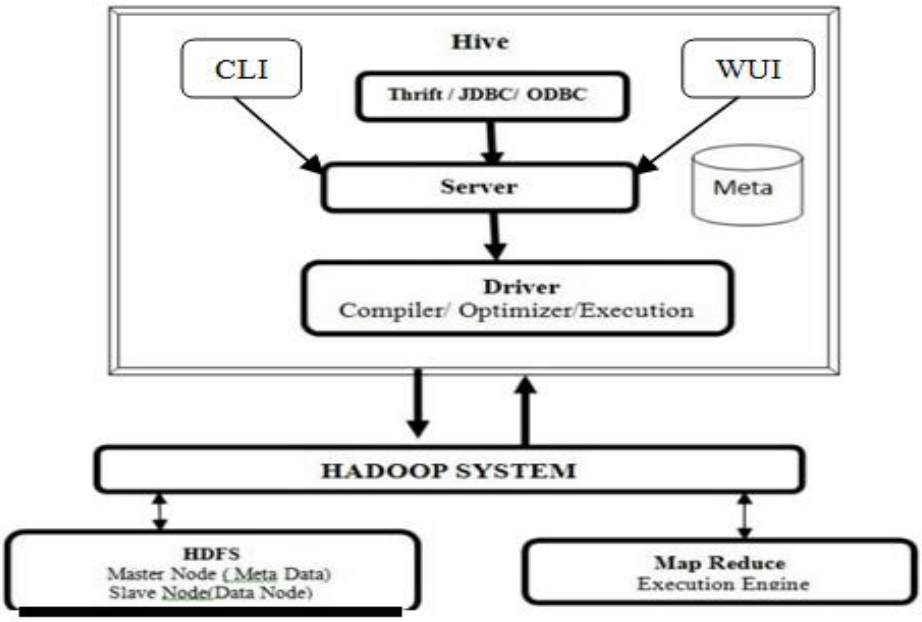

Fig. 1: Hive-Hadoop Ecosystem[1]

By default, Metastore and Hive services are performed in the same JVM into a standard format to save time and space for enormous datasets which are done with the support of custom SerDe (serialization/deserialization). When HiveQL statements send to the driver, the query compiler generates a directed acyclic graph (DAG) of map-reduce jobs for DDL statements. HDFS operations store tables and the last the execution engine executes the operations created by the compiler and cooperates immediately with Hadoop.

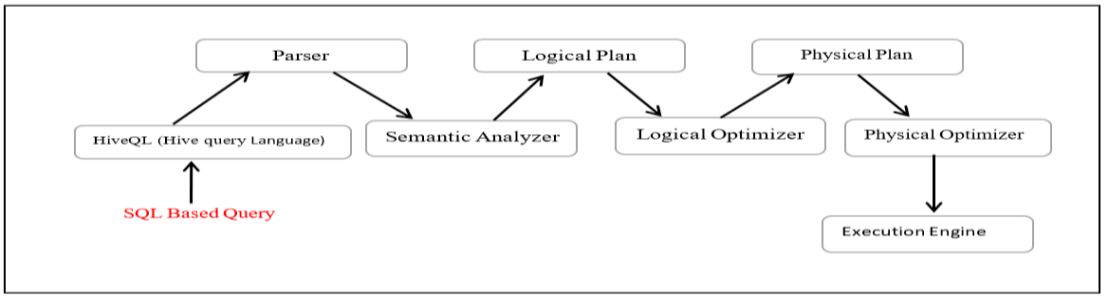

Fig. 2: Hive Query Processing and Compiling[4]

A HiveQL statement is submitted via the CLI, the web UI or an external client using the thrift, ODBC/ JDBC interfaces. The driver passes the query to the compiler and its parsing & semantic analysis and then generates an execution plan for the query [1]. The optimizer uses rule-based or cost-based techniques to optimize this execution plan. And finally, the executor runs the plan in the DAG form of a map-reduce job. The execution engine can be MapReduce Tez, Spark [4], and so on.

Query Optimization in Hive

Query optimization is an essential and most important challenging task in the present scenario where the query performance time has been reduced from seconds to milliseconds and nanoseconds. In general, the Hive optimizer also follows the same principles of query optimization approaches as have been studied in traditional database systems,[21] and parallel database systems [22], and distributed database systems [23]over the years, and various remarkable solutions have been proposed. However, query optimization in a MapReduce-based system [14] is different from other traditional methods. Therefore, need to investigate a cost-based good strategy and find the near-optimal plan out of the equivalent plans to produce the final result with minimum execution time.

Join the most expensive and overbearing factor in query processing that increases the execution time of Hive queries. The major challenge of the MapReduce system is to process the joins. Basically, join query evaluation strategies depending on two factors:

- Join Algorithm: require to design a good join algorithm
- Join Order Query optimization: make a proper join order to reduce the execution time and cost.

In the remaining discussion, we briefly introduce the join algorithm because this paper focuses on the join order algorithm. After all, a large-scale query involves many relations showings many joins, and these large numbers of joins in the query directly affect the performance of the query optimization process. For example, suppose we join 3 tables R1, R2, R3 of size 10 rows, 10,000 rows, 1,00,000 rows, if a query plan joins R2 and R3 it takes more time to execute than joins R1 and R3 first. Therefore, we examined the different join order query optimization strategies that improve the join performance in terms of execution time and I/O cost because the proper join sequence plays a significant role in improving the join performance.

Join Algorithm in Hive

Map-reduce framework is widely used for large-scale data processing. Hadoop's Map-Reduce structure supports HiveQL, a declarative language like SQL for storing and retrieving queries. Join is the most important and costly operation for data analytics, but the Map-Reduce system does not support join algorithms directly. Therefore, Map-Reduce Join algorithms are needed to perform the join operation that's why this paper briefly study on equi-join based map-reduce join algorithms for map-reduce jobs. This Map-Reduce joins algorithms are roughly classified into either in map-side join or in reduce-side join.

Map-Side Join Algorithm[14] is a default join algorithm in the Hadoop framework that uses only one map function to perform join operations and doesn't need to reduce function. It doesn't need to send intermediate results from the map function to the reduce function, that's why the map-side more efficiently reduces I/O cost and execution time but it can be applied in particular conditions. Map-Side join algorithms are classified into Map-Merge Join, which requires two fully Map-Reduce jobs and one only Map job. Map job performs the merge join and send the final result into HDFS. Map-Side Partition Merge Join [20] is an advance of the map-merge join. It's mainly used for large datasets and its helps in avoid memory overflow. Broadcast Join [20]. Map-Side join algorithm is a combination of Fragment Replicated Join and Reversed Map Join. when one dataset is fit in the cache memory in two datasets, it works as a Fragment Replica Join and if dataset is large, it works as a Reverse Map Join. Broadcast join more effectively if datasets are small enough. So, we can say that all map-side join algorithms are suitable for eliminating the overheads of intermediate results. Nevertheless, there are some issues related to dataset and time, so MapReduce Jobs are needed to meet these limits.

Reduce-Side Join Algorithm: When reduce function performs the join operation is known as reduce-side join. Map function creates a common join key corresponding to both relations and reduce function joins the records with the same join key. *Repartition Join* [26] is a reduce-side algorithm that requires only one Map-Reduce job for the join operation, but its main drawback is increased execution time which degrades performance whereas *Improved Repartition Join* is used to solve the buffering issues of the *Standard Repartition Join* but it also increases overhead. *Semi-Join*[18] join algorithm in Map-Reduce system are similar to those widely used in traditional DBMSs Semi-joins are often used when one dataset (R1) is much larger than another dataset (R2). The main objective of this algorithm is to remove duplicate data from one dataset (R1) using unique keys of another dataset (R2). It uses three map-reduce jobs

Multi-Join Order Query Optimization in Hive

Rule-based join order query optimization,[1] rule-based query optimization in MapReduce is similar to RDBMS's query optimization at a logical level, which is not sure to give the optimal solution for a multi-join order query plan. whereas *Cost-based join order query optimization*, [14] is based on a cost model and data statistics to evaluate efficient query plans for Hive queries. There are three main factors to making an optimal plan for a join query which are- Search Space, Cost Model, and Search Strategy.

- **Search Space:** The search space of the optimizer consists of multiple equivalent Query Evaluation Plans (QEPs) that differ from one another in the cost of query execution but they all compute the same result. The join operation follows commutative and associative properties so when there are N relations in the join operation there is an O(N!) possible QEP. There are two types of shapes defined for query trees viz. Left Deep Trees, and Bushy Trees; Left Deep Trees or Linear Trees can reduce the search space to $O(2^N)$ as they do not leave any degree of

freedom concerning the shape of the tree. Bushy trees do not restrict the cardinality of search space as they permit join nodes with both operands as composites i.e. without base relations. These trees are capable of implementing parallelism and are most suitable for parallel machines. For n base relations, there are (2(n-1)/(n-1)(n-1)! different solution. Cost Model: It includes the cost of operators, Data-Statistics and procedures to estimate the magnitude of intermediary results. The measured cost of the query is measured in terms of Total Execution Time or Query Response Time. we have two types of costs: the first type is I/O costs for example local disk I/O and network I/O and CPU costs. The second type is the total cost of processing a Map Reduce job. Search Strategy: The search strategy implemented by the database optimizer to extract appropriate query data in minimum computational time from the database is of great concern. There are various search strategies available for hive queries like deterministic strategies, randomized strategies,[32] etc. dynamic programming uses breadth-first search and greedy algorithm uses depth-first search, both of which are examples of dynamic programming. They create a runtime solution but when the size of the relation is greater than 10 the space complexity increases and the time exponentially as well whereas randomized algorithms are capable of reducing high costs but have no guarantee for the optimal solution for large joins.

Literature Review

Hadoop Map-Reduce[15] has been introduced by Jeffrey Dean and Sanjay Ghemawat in 2004 which is a cluster-based solution that is widely used in the current data warehouse system as the centralized servers do not provide scalable performance. We adopt Hadoop open-source as a processing engine. Hive (Thusoo et al.,2010) and Pig (C. Olston et al.,2008) are pure map-reduce batch processing SQL-like declarative query languages but Hive has more efficient and less time-consuming for data processing as compared to Pig (Choudhary, A.et al., 2015) (Kumar, S. et al., 2016).

Hive used Repartition Algorithm[1] for n-way join query optimization in large TPC-H benchmark-based datasets with three map-reduce jobs. It uses string file format which can be 20% optimized by Text File to improve Hadoop performance and for join ordering it use rule-based query optimization. To improve the performance of the Hive queries, we need a good join algorithm for n-way joins that reduce I/O overheads, a cost-based optimizer & an adaptive optimization technique for multi join order and a good file format. It plays a significant role in Hive query optimization as a good file format gives a better space in the system. Hive can store files in both format row and columnar in the earlier version of Hive, Text File, Sequence File is the type of row format and RC File (Record Columnar File) & Parquet file, ORC (Optimized Row Columnar) is the type of columnar file format. ORC file is developed to overcome restrictions of the RC File, Text File, Sequence File etc. ORC[17] reduces 75% of the size of the original data and increases data processing time. It also supports compressed (ZLIB, Snappy) as well as uncompressed storage.

This study proposed [1] Hive architectures and their query processing, the purpose of this study was to improve the performance of join operations on large datasets. It used a repartition join algorithm for a two-way Equi join with three map-reduce jobs. And it used a rule-based optimization algorithm for the query plan, similar to that of a traditional database, which is not sure to give the optimal solution for a multi-join order query plan. Two important issues have the outcome for this researched 1. Repartition Join Algorithm increases the cost of shuffle and second, we need cost-based optimization similar to RDBMS, which could improve the join order query plan.

This paper focuses[13] on cost-based optimization for join queries to overcome the limitations of rule-based query optimization because rule-based query optimization increases the overall execution time when the table is large. The purpose of this cost-based query optimizer is to determine a better join order based on column-level statistics to reduce query response times. It uses a permutation algorithm or exhaustive algorithm for the join order which does not give the optimal plan as it generates too many unusable join orders and this column-statistics based cost model or Histogram does not calculate the HDFS I/O cost that affects the query performance. Hence, there is a need for a good cost model that better estimates the cost of local disk I/O, network I/O, CPU costs and map-reduce jobs cost as well as search space strategy for multi join sequence that will perform efficiently on TPC-H datasets above 100 GB.

The main purpose of the AQUA[14] optimizer was to improve the efficiency and performance of the MapReduce data processing systems by generating efficient query plans based on cost models. AQUA is two-phase, In the first phase, replicated join was used to evaluate each group of join operators

to reduce overheads and I/O costs. In the second phase, a heuristic plan generator was used to avoid bad query plans and select one of the best plans with minimum processing cost, and a shared scan was also adopted to improve performance. It supports both the Left tree and Bushy tree plans. There is a need for a search algorithm for above 10 relations and a need for an n-way theta join algorithm for join operations.

This paper proposed [19]to optimize and improve the SQL-based HiveQL query performance on the Hadoop system. The sequence of join order plays an important role to improve the join query performance. For that a built cost-based JOMR optimizer used a parallel Travel Salesman Problem (TSP) based Tabu Algorithm to change the sequence of the join tables that enhance the performance of Hive's logical plan, when data size and joins increase. It supports Left Deep Tree plan for complex queries and the Hadoop DB hybrid system to include both MapReduce and DBMS.

This paper introduced a cost-based SOSQL[31]scalable optimizer for SQL query processing on MapReduce, where map-reduce jobs can be using cost-based two-way or multi-way equi or theta join algorithm. Here, the repartition join algorithm is a two-way equi join algorithm using MapReduce and the workload partition algorithm handles the two-way theta join algorithm using MR whereas it introduced a replicate join algorithm for multi-join. It applied cost-based Dynamic Programming or Polynomial Algorithm for optimal execution with $O(n^2)$ and performed better than Hive.

An optimizer [27]that made a hybrid system with PostgreSQL and MapReduce. This paper tried to utilize an index in the existing database system and use co-partition join to exploit the performance of join processing because it does not reallocate data. It used RNN Model to calculate the cost of all sub-plan and binary vector for TPC-H datasets. AQUA+ considered both left and bushy plans and it used a Greedy-based Heuristic approach to the sequence of query plans but a greedy algorithm is not optimal for complex queries. So, there is a need for a more efficient search algorithm to handle the complex query. This paper proposed ACO-GA algorithm and HDFS [29] MapReduce to enhance query optimization in Big Data. This paper proposed two methods for Big Data arrangement and optimization process. In the first method, SHA-512 algorithm is used to remove duplicate data in HDFS and the K-Means clustering technique is used to group N values into the number of clusters for Big Data arrangement which resolves the clustering problem to find the optimum objective. The second method used the Traveling salesmen problem (TSP) based ACO-GA Algorithm to find the shortest route to join the multiple nodes. The proposed system was 96.25% more accurate than the older method. In the future will use advanced optimization algorithms to reduce the execution time and improve the efficiency of the system.

The GHive [28] system proposed in this paper is used to exploit GPU parallelism to enhance the performance of OLAP in the database. This system used Hive for OLAP queries using CPU and GPU, SSB benchmark is used as a dataset, In GHIVE SYSTEM, SQL query is parsed, then Abstract Syntax Tree passed to Calcite-based Query Optimizer (which uses Dynamic Programming) to produce optimized logical plan after that Physical Optimizer to produce a vectorized physical plan and is converted into MapReduce jobs that are run on Apache Tez. The query execution of GHive is much better Hive. In the future, there is lots of space to optimize logical plan by advanced optimization algorithms.

Conclusion

Multi Join Query Optimization (MJQO) plays a significant role in many applications like search engines, data mining, data warehouse, and banking system, etc. As the data size increases day by day and due to insufficient technology, the multi-join query optimization problem ruins. It may include large join queries, high costs, and time-consuming execution time. In previous research work clearly shows that traditional methods like dynamic programming, exhaustive search, greedy algorithms, and randomized method are used for Hive query processing both are insufficient to execute the query and time-consuming to solve the large join queries it also enhanced execution costs and space complexity. This paper diverts the forces of researchers to use an appropriate cost-based advanced search strategy to find the optimal solution for large join order in Hive queries with better performance. This paper draws the researcher's attention to the use a of reasonable cost-based advanced search strategy to find an optimal solution for large join order in Hive queries with better performance.

References

1. Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Anthony, S. and Murthy, R. (2009) 'Hive: a warehousing solution over a map-reduce framework', *Proceedings of the VLDB Endowment*, Vol. 2, No. 2, pp.1626–1629.

2. Thusoo, R. Murthy, J. S. Sarma, Z. Shao, N. Jain, P. Chakka, S. Anthony, H. Liu, and N. Zhang. Hive – a petabyte-scale data warehousing using hadoop. In *ICDE*,2010.
3. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *SIGMOD*, 2008.
4. Pal, S. (2016). SQL on Big Data: Technology, Architecture, and Innovation. Apress.
5. BARKHA, J., & Km, K. M. (2015). Query Optimization in Hive for Large Datasets. *Advances in Computer Science and Information Technology (ACSIT), India*.
6. Leu, J.S., Yee, Y.S. and Chen, W.L. (2010) 'Comparison of map-reduce and SQL on large-scale data processing', *International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pp.244–248.
7. Lee, K. H., Lee, Y. J., Choi, H., Chung, Y. D., & Moon, B. (2012). Parallel data processing with MapReduce: a survey. *AcMsIGMoD record*, *40*(4), 11-20.
8. *Abuqabita, Flasteen, Razan Al-Omoush and Jaber Alwidian. "A Comparative Study on Big Data Analytics Frameworks, Data Resources and Challenges."(2019).*
9. Choudhary, A., &Satsangi, C. S. (2015). Query Execution Performance Analysis of Big Data Using Hive and Pig of Hadoop. *International Journal of Computer Sciences and Engineering*, *3*(9), 91-97.
10. Huai, Y., Chauhan, A., Gates, A., Hagleitner, G., Hanson, E.N., O'Malley, O. and Zhang, X. (2014) 'Major technical advancements in apache hive', *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, Snowbird, UT, pp.1235–1246.
11. Lee, R., Luo, T., Huai, Y., Wang, F., He, Y. and Zhang, X. (2011) 'YSmart: yet another SQL-to-MapReduce translator', *ICDCS*
12. Comparison of Hive's query optimisation techniques
13. Gruenheid, A., Omiecinski, E., & Mark, L. (2011, September). Query optimization using column statistics in hive. In *Proceedings of the 15th Symposium on International Database Engineering & Applications* (pp. 97-105).
14. Wu, S., Li, F., Mehrotra, S. and Ooi, B.C. (2011) 'Query optimization for massively parallel data processing', *Proceedings of the 2nd ACM Symposium on Cloud Computing*, Cascais, Portugal, p.12.
15. J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. Operating Systems Design and Implementation, 2004.
16. Kumar, S., & Goel, E. (2016). Comparative analysis of mapreduce, hive and pig. *Research Cell: An International Journal of Engineering Sciences*, *17*(1), 51-63.
17. Y. He, R. Lee, Y. Huai, Z. Shao, N. Jain, X. Zhang, and Z. Xu, "Rcfile: A fast and space-efficient data placement structure in mapreduce-based warehouse systems," in *ICDE*, 2011, pp. 1199–1208.
18. Mohamed, M. H., Khafagy, M. H., & Ibrahim, M. H. (2018). From Two-Way to Multi-Way: A Comparative Study for Map-Reduce Join Algorithms. *WSEAS Transactions on Communications*, *17*, 129-141.
19. Shanoda, M. S., Senbel, S. A., &Khafagy, M. H. (2014, December). Jomr: Multi-join optimizer technique to enhance map-reduce job. In *2014 9th International Conference on Informatics and Systems* (pp. PDC-80). IEEE.
20. Al-Badarneh, A. F., &Rababa, S. A. (2020). An analysis of two-way equi-join algorithms under MapReduce. *Journal of King Saud University-Computer and Information Sciences*.
21. S. Chaudhuri. An overview of query optimization in relational systems. In *PODS*, pages 34–43, 1998.
22. S. Ganguly, W. Hasan, and R. Krishnamurthy. Query optimization for parallel execution. *SIGMOD Rec.*, 21(2),1992.
23. P. A. Bernstein, N. Goodman, E. Wong, C. L. Reeve, and J. B. Rothnie, Jr. Query processing in a system for distributed databases (sdd-1). *ACM Trans. Database Syst.*, 6(4):602–625, 1981.

24. Reinsel, D., Gantz, J. and Rydning, J., 2018. Data Age 2025: The Digitization of the World From Edge to Core. International Data Corporation (IDC) White Paper, [Online] Available at: https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf.

25. Bagui, S., &Devulapalli, K. (2018). Comparison of Hive's query optimisation techniques. *International Journal of Big Data Intelligence*, *5*(4), 243-257.

26. Blanas, S., Patel, J.M., Ercegovac, V., Rao, J., Shekita, E.J., Tian, Y., 2010. A Comparison of Join Algorithms for Log Processing in MapReduce. Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, pp. 975–986.

27. Pang, Z.,Wu, S., Huang, H., Hong, Z., &Xie, Y. (2021). AQUA+: Query Optimization for Hybrid Database-MapReduce System. *Knowledge and Information Systems*, *63*(4), 905-938.

28. Liu, H., Tang, B., Zhang, J., Deng, Y., Zheng, X., Shen, Q., ... & Luo, Z. (2022, June). GHive: A Demonstration of GPU-Accelerated Query Processing in Apache Hive. In *Proceedings of the 2022 International Conference on Management of Data* (pp. 2417-2420).

29. Kumar, D., & Jha, V. K. (2021). An improved query optimization process in big data using ACO-GA algorithm and HDFS map reduce technique. *Distributed and Parallel Databases*, *39*(1), 79-96.

30. Margoor, A., & Bhosale, M. (2020, December). Improving Join Reordering for Large Scale Distributed Computing. In *2020 IEEE International Conference on Big Data (Big Data)* (pp. 2812-2819). IEEE.

31. Shan, Y., & Chen, Y. (2015, June). Scalable query optimization for efficient data processing using mapreduce. In *2015 IEEE International Congress on Big Data* (pp. 649-652). IEEE.

32. Azhir, E., Jafari Navimipour, N., Hosseinzadeh, M., Sharifi, A., &Darwesh, A. (2019). Deterministic and non-deterministic query optimization techniques in the cloud computing. *Concurrency and Computation: Practice and Experience*, *31*(17), e5240.

□❍□