Acta Scientific COMPUTER SCIENCES

Volume 4 Issue 2 February 2022

Short Communication

How Apache MapReduce Handles Big Data Query?

Radhya Sahal*

Faculty of Computer Science and Engineering, Hodeidah University, Al Hudaydah, Yemen

***Corresponding Author:** Radhya Sahal, Faculty of Computer Science and Engineering, Hodeidah University, Al Hudaydah, Yemen.

Received: December 20, 2021

Published: January 18, 2022

Abstract

Apache MapReduce is the most popular framework for batch data processing. However, despite its merits, the critical challenge of Apache MapReduce is rapidly handling queries over large scale data. This review aims to provide the state-of-the-art of Apache Hive, a famous language to handle big query data on Apache MapReduce.

Keywords: Query Processing; Apache MapReduce; Hive; HiveQL

Apache MapReduce principles

Apache MapReduce is a programming model applied by Google for various purposes such as analytical data applications. In addition, it is a correlated implementation to process and generate Big Data sets [1,2]. Furthermore, it improves the performance of Big Data analytics by exploiting parallelism among several processing nodes [3]. At the programming level, developers define a Map function and a Reduce function as follows; the Map function is the phase that processes key-value pairs to produce a set of intermediate key-value pairs. Reduce function is the phase that joins all intermediate values associated with the same intermediate key. In other words, the programs are written in this functional style which is automatically parallelized and executed on a cluster of commodity machines. Furthermore, the runtime system considers the details of partitioning the input data, scheduling the program's execution on a set of machines, handling machine failures and managing the required inter-machine communication. This allows the programmers to efficiently use the resources of a massively distributed system without any experience with parallel and distributed systems [4].

Figure 1 depicts the execution of MapReduce phases which implement two functions; Map and Reduce. The Map phase takes an input pair and generates a set of intermediate key-value pairs. The MapReduce library combines all intermediate values with the same intermediate key I and passes them to the Reduce phase. The Reduce phase accepts an intermediate key I and a set of values. It joins together these values to form a probably smaller set of values where the intermediate values are provided to the user's Reduce function via an iterator. Typically, just zero or one output value is produced per Reduce invocation [1].

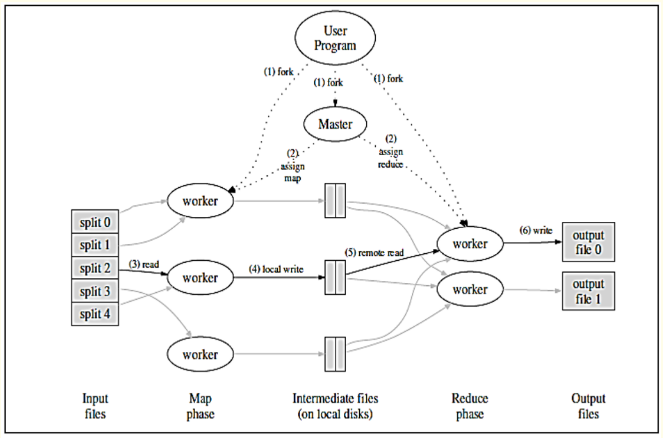

Figure 1: MapReduce execution overview [1].

Query processing in Apache MapReduce

Besides efficiency, Apache MapReduce provides two simple, user-friendly interfaces; Map and Reduce function (Figure 1). Apache MapReduce supports query processing by integrating high-level declarative languages such as Hive and Pig to simplify the programming of data applications [5-7]. These MapReduce-based query languages hide implementation details (e.g., access methods, query plan optimization) and offer developers and Big Data analysts an SQL-like interface [5,6]. Hive language is used in this work, where the HiveQL queries are translated into MapReduce jobs [7]. From the inside, each MapReduce job is flushed back to Hadoop Distributed File System (HDFS) as a backup for fault tolerance. The next MapReduce job reads the intermediate results of the previous job to continue processing. The HDFS I/O cost is significantly higher than local storage, including a network cost. So, exploiting the shared jobs within multiple queries can reduce HDFS I/O cost of intermediate results. Consequently, the shuffling cost of intermediate results can be cheaper than generating too large a size of intermediate results when no sharing data is exploited [7,8].

Apache hive

Hive is the first high-level query language for data warehouse, runs on top of the Hadoop framework to facilitate executing the query and managing large datasets [9]. It provides an interface similar to SQL, called HiveQL which makes MapReduce tasks more understandable and easier for programmers and developers [9,10]. Recently, Hive has become one of the Hadoop ecosystems tools and compatible with many massively parallel processing such as MapReduce and Tez. Therefore, most new research studies focus on high-level language regardless of the massively parallel processing [11-15]. According to these studies, Hive is considered a mature SQL-on-Hadoop engine. The user connects to the user interface and executes a HiveQL command sent to the driver. The driver creates a session and then sends the query to the compiler, which extracts metadata from the metastore and generates an execution plan. This logical plan is optimized by Hive query optimizer and then translated into an executable query plan which consists of multiple MapReduce phases. Finally, the MapReduce execution engine executes the plan consisting of one job tracker and possibly several task trackers per Map and Reduce phases [15].

HiveQL

HiveQL does not strictly follow the full SQL-92 standard. HiveQL offers extensions, not in SQL, including multiple INSERT and CREATE TABLE AS SELECT commands. Also, HiveQL supports transactions and materialized views. Furthermore, HiveQL provides different plans for INSERT, UPDATE, and DELETE with full ACID functionality. Hive compiler translates HiveQL statements into a directed acyclic graph (DAG) of MapReduce jobs that are given to be executed in Hadoop [16,17].

Bibliography

1. J Dean and S Ghemawat. "MapReduce: simplified data processing on large clusters". *Communications of the ACM* 51 (2008): 107-113.
2. R Lämmel. "Google's MapReduce programming model-Revisited". *Science of Computer Programming* 70 (2008): 1-30.
3. S Wu., *et al.* "Query optimization for massively parallel data processing". *In Proceedings of the 2nd ACM Symposium on Cloud Computing* (2011): 12.
4. J Dean and S Ghemawat. "MapReduce: a flexible data processing tool". *Communications of the ACM* 53 (2010): 72-77.
5. R Sahal., *et al.* "Exploiting Coarse-grained Reused-based Opportunities in Big Data Multi-Query Optimization". *Journal of Computational Science* 26 (2018): 432-452.
6. R Sahal., *et al.* "Comparative Study of Multi-query Optimization Techniques using Shared Predicate-based for Big Data". *International Journal of Grid and Distributed Computing* 9 (2016): 229-240.
7. R Sahal. *et al.* "iHOME: Index-based JOIN Query Optimization for Limited Big Data Storage". *Journal of Grid Computing* 16 (2018): 345-380.
8. X-Y Gao. *et al.* "Exploiting Sharing Join Opportunities in Big Data Multiquery Optimization with Flink". *Complexity* (2020): 2020.
9. A Thusoo. *et al.* "Hive: a warehousing solution over a map-reduce framework". *PVLDB* 2 (2009): 1626-1629.
10. A Thusoo. *et al.* "Hive-a petabyte scale data warehouse using Hadoop". *In 26th IEEE International Conference on Data Engineering* (ICDE) (2010): 996-1005.

11. J LeFevre. *et al.* "Opportunistic physical design for big data analytics". *In Proceedings of ACM SIGMOD international conference on management of data* (2014): 851-862.

12. HSA Azez. *et al.* "JOUM: An Indexing Methodology for Improving Join in Hive Star schema". *International Journal of Scientific and Engineering Research* 6 (2015): 111-119, 2015.

13. MN Abdullah. *et al.* "HOME: HiveQL Optimization in Multi-Session Environment". *In Proceedings of the 5th European Conference of Computer Science (ECCS14)* (2014): 80-89.

14. T Dokeroglu. *et al.* "Improving the performance of Hadoop Hive by sharing scan and computation tasks". *Journal of Cloud Computing* 3 (2014): 1-11.

15. A Gruenheid. *et al.* "Query optimization using column statistics in hive". *In Proceedings of the 15th Symposium on International Database Engineering and Applications* (2011): 97-105.

16. E Capriolo. *et al.* "Programming Hive". Data warehouse and query language for Hadoop, O'Reilly Media, Inc (2012).

17. R Kumar. *et al.* "Comparison of SQL with HiveQL". *International Journal for Research in Technological Studies* 1 (2014): 2348-1439.