ACTA SCIENTIFIC COMPUTER SCIENCES

Volume 4 Issue 1 January 2022

Research Article

Query Optimization for Big Data Batch Processing and Stream Processing

Radhya Sahal*

CONFIRM Centre for Smart Manufacturing, University College Cork, Ireland

***Corresponding Author:** Radhya Sahal, CONFIRM Centre for Smart Manufacturing, School of Computer Science and IT, University College Cork, Ireland.

Received: November 11, 2021

Published: December 13, 2021

Abstract

Big data refers to huge and complex data sets made up of a variety of structured and unstructured data that are too big, too fast and too hard to be managed by traditional techniques. Big data exceeds the processing capacity of conventional database systems. Recently, new technologies have been invented to analyze and query this massive data. In this work, we have introduced two types of big data query optimization including batch processing and streaming processing.

Keywords: Query; Optimization; DBMS; Batch Data Processing; MapReduce; Stream Data Processing

Introduction

Nowadays, IT has been concentrated in all enterprises and companies, where the users deal with many computing applications. These applications allocate a large amount of data (e.g. facts, statistics, numbers, text and images) and information (that data has been processed and become meaningful for proper use). A collection of these data and information is called a database [1]. The database is controlled and managed by Database Management System (DBMS), such as Oracle, DB2, MS Access, MySQL, and PostgreSQL. It allows the users to store, update, organize, protect and retrieve their databases using a powerful query language [2]. A query language is a specialized language for retrieving information from a database [3]. It is written in a predefined form, such as SQL language. SQL stands for Structured Query Language, which is used for accessing and modifying the database. The query language contains three sequential phases to execute a given query, depicted in figure 1 [2,4].

The query phases are listed as follows [5]:

- Parsing (translation) translates the query into internal form and checks the syntax of relations in database.
- Optimization finds the best plan to execute the query. An execution plan is a set of sequence operations to implement a query with the lowest estimated cost. The query optimization phase chooses the execution plan based on statistics of database and resources (e.g. path, CPU, memory, and storage) [6].
- Evaluation executes the execution plan and returns the query answer.

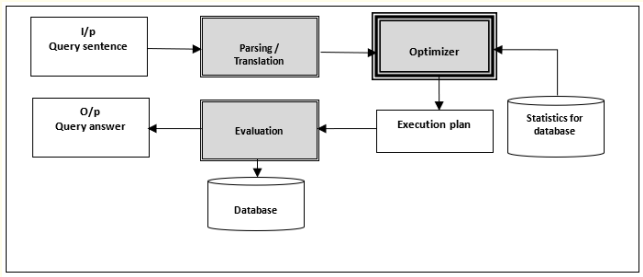

Figure 1: Query Processing.

One of Relational Database Management Systems (RDBMS) is query optimization [7]. It is a tool, which decides for the best plan to execute queries. A query optimizer determines the set of possible execution plans and chooses the best one according to the cost, time, and resource consumption [8,9].

The architecture of the query optimizer is depicted in figure 2. The planner is the main component of a query optimizer. It generates multiple query plans and chooses the best one for answering a query. Moreover, the planner looks up statistics tables (tables that hold the statistics and information of database tables such as catalogues) and uses search strategies that compare the execution plans based on their estimated costs to choose the cheapest plan [5,7]. This thesis focuses on the cost model in the query optimizer planner, which is used to profile the resource intensive of database workload. The resource intensive (e.g., CPU-intensive, Memory-intensive, and I/O-intensive) means how the performance of the workload affected by allocating/de-allocating more resources [6].

Big data has been spread rapidly in many domains, such as information systems. On the other hand, distributed computing is growing every day with the increasing of workstations power and the size of the dataset. Therefore, the development and implementation of the distributed system for Big Data applications are considered a challenge [10,11]. Two types of big data query processing are batch processing and streaming processing (see Figure 3). The details of query optimization for batch and streaming data processing are elaborated following.

Query optimization for batch data processing

The structure formatting and key-value data store will be concerned where the queries are transformed into data stages, including a set of jobs run on batch processing. Batch processing happens between blocks of data that have already been stored over some time. Hadoop MapReduce is the best framework for processing data in batches.

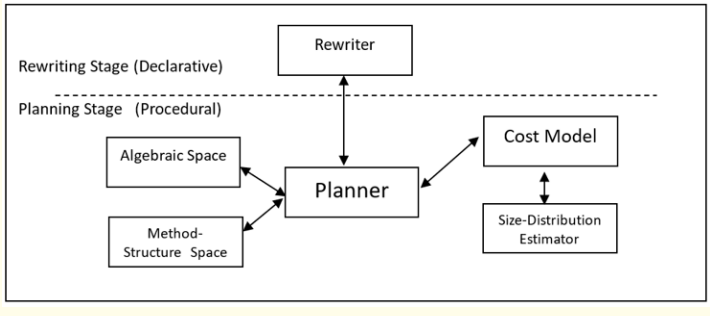

Figure 2: Query optimizer architecture: two stages (Rewriting and Planning).

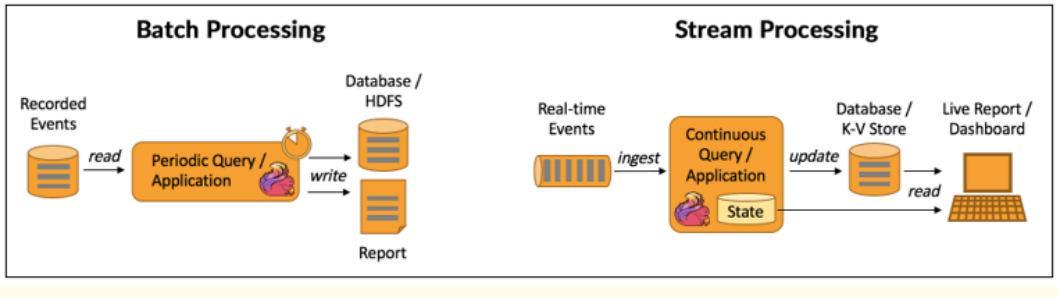

Figure 3: Query batch data processing vs. query stream data processing[1].

[1]https://k21academy.com/microsoft-azure/data-engineer/batch-processing-vs-stream-processing/

Besides efficiency, Hadoop MapReduce provides two simple, user-friendly interfaces; Map and Reduce function. Furthermore, Hadoop MapReduce supports query processing by integrating high-level declarative languages such as Hive and Pig to simplify data applications' programming [12-14]. 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. Regarding this thesis, Hive language is used where the HiveQL queries are translated into MapReduce jobs. 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, which includes 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 [15-20].

Query optimization for stream data processing

Industries realize a competitive advantage in being able to act on high volume and high-velocity streams of data. Stream processing allows the data analysis systems to process data in real-time as they arrive and quickly detect conditions within a small period from receiving the data. Different query streaming open-source engines have been developed to leverage windowing operations by repeatedly iterating over a series of micro-batches, in much the same way as static queries operate over stored data. Examples of the open-source streaming-based engines that use the windowing concept are Spark SQL[2], Flink Table API[3], StromSQL[4], SamzaSQL[5], and KSQL[6].

Typically, a window join query consumes infinite input data streams and produces output data streams (i.e., joined tuples) as long as the input streams continue streaming in [21]. Traditional cardinality based cost models for join optimization cannot produce joined results due to their mechanisms to estimate the required time to join query based on computations over the static dataset and query completion. However, stream queries have to handle unknown and potentially infinite data with no guarantee on when the query will eventually terminate. Different optimization techniques for stream query optimization are available; however, the work on optimizing the stream query based on window size has limited solutions. IoT devices and applications offer dynamic and loosely controlled networks with almost complete autonomy of devices and result in unpredictable behaviours such as data drifts, e.g. variable stream rate. No single optimal window configuration (one size fits all) is possible due to the variable input stream rate.

Furthermore, a limitation of computing resources and application requirements is also dynamic for IoT, making the query window configuration selection even further complicated. An optimal window configuration for a given query over a stream can become obsolete due to any changes in stream rate or infrastructure. The authors in have proposed a dynamic window-based selector, which provides a mechanism to select the proper window configuration considering stream attributes, changing stream rates, workload requirements, and resource capacity constraints during the query optimization process [22,23].

Acknowledgements

This research has emanated from research supported by a research grant from Science Foundation Ireland (SFI) under Grant Number SFI/16/RC/3918 (CONFIRM), and Marie Skłodowska-Curie grant agreement No. 847577 co-funded by the European Regional Development Fund.

Bibliography

1. HFK Abraham., *et al.* "Database System Concepts". The 6th Edition. McGraw-Hill (2011).

2. R Sahal., *et al.* "Automatic calibration of database cost model in cloud computing". in 2012 8th International conference on informatics and systems (INFOS) (2012): CC-25-CC-34.

[2]https://spark.apache.org/sql

[3]https://ci.apache.org/projects/flink/flink-docs-master/dev/table/tableApi.html

[4]http://storm.apache.org/releases/1.1.2/storm-sql.html

[5]https://github.com/milinda/samza-sql

[6]https://www.confluent.io/product/ksql

3. G Papakonstantinou and J Kontos. "A query-oriented file organization technique". *International Journal of Systems Science* 5 (1974): 743-751.

4. H Turtle and J Flood. "Query evaluation: Strategies and optimizations". *Information Processing and Management* 31 (1995): 831-850.

5. S Chaudhuri. "An overview of query optimization in relational systems". in Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, Seattle, Washington, United States (1998): 34-43.

6. R Sahal S., *et al.* "GPSO: An improved search algorithm for resource allocation in cloud databases". in 2013 ACS International Conference on Computer Systems and Applications (AICCSA) (2013): 1-8.

7. JC Freytag. "The Basic Principles of Query Optimization in Relational Database Management Systems". in Proceedings of 11th IFIP World Computer Congress, San Francisco (1989): 801-807.

8. S Chu., *et al.* "From theory to practice: Efficient join query evaluation in a parallel database system". in Proceedings of ACM SIGMOD International Conference on Management of Data (2015): 63-78.

9. FA Omara., *et al.* "Optimum Resource Allocation of Database in Cloud Computing". *Egyptian Informatics Journal* 15 (2014): 1-12.

10. R Akerkar. "Big data computing". *CRC Press* (2013).

11. Gkoulalas-Divanis and A Labbi. "Large-Scale Data Analytics". *Springer* (2016).

12. Thusoo A., *et al.* "Hive: a warehousing solution over a map-reduce framework". *PVLDB* 2 (2009): 1626-1629.

13. Thusoo A., *et al.* "Hive-a petabyte scale data warehouse using Hadoop". in 26th IEEE International Conference on Data Engineering (ICDE) (2010): 996-1005.

14. Olston B., *et al.* "Pig latin: a not-so-foreign language for data processing". in Proceedings of ACM SIGMOD international conference on management of data (2008): 1099-1110.

15. S Wu., *et al.* "Query optimization for massively parallel data processing". in Proceedings of the 2nd ACM Symposium on Cloud Computing (2011): 12.

16. R Sahal., *et al.* "SOOM: Sort-Based Optimizer for Big Data Multi-Query". *Big Data* 8 (2020): 38-61.

17. X-Y Gao., *et al.* "Exploiting Sharing Join Opportunities in Big Data Multiquery Optimization with Flink". *Complexity* 2020 (2020).

18. R Sahal., *et al.* "Exploiting Coarse-grained Reused-based Opportunities in Big Data Multi-Query Optimization". *Journal of Computational Science* 26 (2018): 432-452.

19. R Sahal., *et al.* "iHOME: Index-based JOIN Query Optimization for Limited Big Data Storage". *Journal of Grid Computing* 16 (2018): 345-380.

20. R Sahal., *et al.* "Big data multi-query optimisation with Apache Flink". *International Journal of Web Engineering and Technology* 13 (2018): 78-97.

21. R Sahal., *et al.* "Big data and stream processing platforms for Industry 4.0 requirements mapping for a predictive maintenance use case". *Journal of Manufacturing Systems* 54 (2020): 138-151.

22. R Sahal., *et al.* "On Evaluating the Impact of Changes in IoT Data Streams Rate over Query Window Configurations". in Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems (2019): 262-263.

23. R Sahal., *et al.* "Industry 4.0 towards Forestry 4.0: Fire Detection Use Case". *Sensors* 21 (2021): 694.