Big Data Query Optimization -Literature Survey

Anuja S. (anuja9624@gmail.com)
SRM Institute of Science and Technology
Malathy C.
SRM Institute of Science and Technology

Research

Keywords: Big data, Parallelism, optimization, hadoop, and map reduce.

Posted Date: July 12th, 2021

DOI: https://doi.org/10.21203/rs.3.rs-655386/v1

Abstract

In today's world, most of the private and public sector organizations deal with massive amounts of raw data, which includes information and knowledge in their secret layer. In addition, the format, scale, variety, and velocity of generated data make it more difficult to use the algorithms in an efficient manner. This complexity necessitates the use of sophisticated methods, strategies, and algorithms to solve the challenges of managing raw data. Big data query optimization (BDQO) requires businesses to define, diagnose, forecast, prescribe, and cognize hidden growth opportunities and guiding them toward achieving market value. BDQO uses advanced analytical methods to extract information from an increasingly growing volume of data, resulting in a reduction in the difficulty of the decision-making process. Hadoop, Apache Hive, No SQL, Map Reduce, and HPCC are the technologies used in big data applications to manage large data. It is less costly to consume data for query processing because big data provides scalability. However, small businesses will never be able to query large databases. Joining tables with millions of tuples could take hours. Parallelism, which solves the problem by using more processors, may be a potential solution. Unfortunately, small businesses cannot afford to operate on a shoestring budget. There are many techniques to tackle the problem. The technologies used in the big data query optimization process are discussed in depth in this paper.

1. Introduction

Big data is a term used to describe a large dataset that extends the capability of existing database management systems. Since the data is gathered from a variety of sources, it is noisy, heterogeneous, complex, and interconnected. The largest problem in querying big data is the overall amount of data that must be processed, which makes this task as a time-consuming process when the data size reaches petabytes. There is a need for effective query processing techniques for big data, which has a significant effect on businesses and research.

1.1Big Data Challenges for Query Processing

The underlying challenge today, regardless of the type of technologies used, is that we cannot process and make logical sense of any of this data quickly enough. This is indeed a direct result of most of the biases that existing data processing systems have inherited. To begin with, modern relational databases presume that there is always enough workload information and free time to tune the machine only with statistics, appropriate indexes and any other data structure that is needed to increase the speed of data access. With big data coming quickly and unpredictably, and the need to move quickly, we don't have the flexibility to take our time completely in tuning it. Also the, database systems are built on the premise that we can always ingest all data in order to have the most accurate and correct answer. If the amount of data increases, this becomes a much more costly mission.

Overall, we must first analyze a complicated and more time-consuming implementation of a process in order to (a) load the required data and (b) tune the entire system until we can use it for querying. These

sequential steps necessitate not only several hours for a reasonable database size, but also needs the best expert skills and extreme hard work. To put it another way, we need to know precisely what sort of queries can be posed so that we can tune the machine appropriately. If we need to explore a large data set, though, we do not really know what kind of queries can be posed before we start exploring; the response to any one query requires the formulation of the remaining queries.

2. Literature Survey

2. Literature Survey

When considering the scale of the data, query processing with acceptable performance on large data is frequently a concern. Scale independence is addressed to provide improved results when processing a big dataset, where we can provide a bound on the dataset needed to answer a question regardless of the size of the dataset and its changes. Several approaches achieving scale independence in big data, independent of size and range, have been suggested. The majority of database processing methods choose views that, on average, speed up query execution. However, the application's average output is also limited by the scale of the underlying database.

The method proposed [1] describes a scale-independent view selection and maintenance scheme that employs innovative static analysis strategies to ensure that newly generated views never become scaling inefficiencies. With novel static algorithms, scaling problems are avoided by ensuring that the provided views accept invariants on both the scale of the view and the amount of work needed for incremental maintenance. However, as the update of IMV's costs increases with the size of the application, a drawback occurs in the collection of incrementally maintained Materialized Views (IMV). This paper also discusses how to find hotspots and mitigate output loss using schema analysis and query rewrite techniques. This method combines load balancing and concurrent execution. To limit hotspots, this strategy combines load balancing and concurrent execution.

In [4], the accessible portions of the schema are defined for processing, and answering a query with both access and integrity constraints is thoroughly studied. The actual schema emphasizes a query-independent property called super-extractability. If at all we can extract a superset of values from a schema using the designed access pattern, it is said to be super-extractable.. A general definition of database access rewritability under constraints and conditions is that a query is logically access rewritable if it can be tested by running the query on the available data. A method is proposed for deciding whether a part of the schema can be extrapolated using the provided access methods while taking advantage of the constraints.

The Performance Insightful Query Language (PIQL) is a declarative language proposed in [2] that enables the queries to be independent of the size of the dataset even if it scales by computing an upper limit on the number of key/value store operations for each query. Performance Insightful Query Language is a SQL (Structured Query Language) extension that provides additional bounding details to the programmer. For handling unbounded requests, PIQL has a paginate provision. With relationship cardinality

constraints in the database schema, it also provides bound in intermediate outcomes. It offers a service level objective adherence prediction model that calculates the feasibility of scale independence achieving service level goals using the query plan and process bounds.

The simplest SQL extension allows you to describe relationship cardinality and the corresponding size specifications. On such requests, the database programmer chooses a slower bounded plan than an unbounded plan to prevent output degradation. i.e., if the compiler is unable to generate a bound plan for a problem, it attempts to bound the computation in any way possible.PIQL in [3] sets strict limits on the number of I/O operations needed to answer any question. It was created with large-scale data-intensive applications in consideration. It is built on a subset of SQL, resulting in reliable output predictions and aid for modifying entities and relationships in web applications.

The methodologies in [5, 6] explore how to answer questions with limited access patterns. Especially the method in [5] discusses rewriting queries with restricted access patterns with integrity restrictions. For rewriting views, an algorithm is provided for identifying an exact executable plan, whether one exists, or a minimal containing plan for queries. In [6,] the complete answer for a query is computed using the binding pattern of a relation. A relation's binding pattern defines the limit on attributes that can be used to access the relation.

The technique in [7] proposes a Scalable Consistency Adjustable Data Storage (SCADS) architecture that allows users to specify application-specific requirements, uses utility computing to provide cost-effective scale-up and scale-down, and applies machine-learning algorithms to handle performance problems and predict the resource requirements of new queries before they are executed.

Performance-Safe Query Language, –Performance Tradeoff, Declarative Consistency and Scaling Up and Down Using Machine Learning are three technologies that include data scale independence. Since Performance-Safe Query Language is a scale-aware query language, it allows for effective web programming by ensuring scalability and predictability. Declarative Consistency –Performance Tradeoff is a declarative consistency and performance tradeoff language that enables developers to determine the accuracy of an application in terms of required performance SLAs (Service Level Agreement). Upgrading and Downgrading Machine Learning is the ability to use machine-learning models to effectively add and subtract capacity in order to meet SLAs.

Generally, optimizations based on Map Reduce functions are random computations written in a general-purpose programming language. In general, Map Reduce-based systems do not process iterative and complex queries that require access to a largest number of collections. Query optimization adapting the Map Reduce method isn't appropriate for all types of queries. In paper [8] the Hadoop framework is extended and the map-reduce framework is used to parallelize queries across nodes. HadoopDb has the advantage of fault tolerance and can operate in a variety of conditions. To have improved efficiency, the majority of query processing is done inside the database engine.

HadoopDb has a data storage layer and a data processing layer, with the storage layer being a block-structure framework that manages the core name node. The Data Loader portion partitions data globally based on a specified partition key. HadoopDb has a database connector that serves as a link between the task tracker and the independent database structure on the nodes. Any map reduce job provides a sql query as well as link parameters such as jdbc engine, query fetch size, and other query tuning parameters to the connector. The catalogue part keeps track of database metadata and stores it in HDFS as an XML (Extensible Markup Language) file (Hadoop Distributed Storage System).

HaLoop [9] is an updated Hadoop map reduce architecture that not only extends map reduce with programming language support for iterative queries, but also increases efficiency by rendering the task scheduler loop-aware with cache mechanisms. It offers a unique parallel and distributed system for large-scale iterative data processing applications.

The programming model and design of twister in [10] improves the map, minimizes runtime for facilitating iterative queries and computes efficiently. HaLoop is designed on top of Hadoop platform, and the optimizations include a loop-aware scheduler, loop-invariant caching, and caching mainly for fast fix point verification. Data is read from local disks associated with the worker node, and the intermediate data is stored in a distributed memory of the worker node. The outputs of the map tasks are moved to the corresponding reduce task through a broker network, where they are buffered before the reduce computation is executed. As a result, the intermediate fits into the distributed memory.

The approach in [11] proposes an optimization framework for SQL-like map-reduce queries. This paper focuses on a query language called Map Reduce Query Language, which is descriptive to catch much of the computation in declarative form and can also be optimized in a better way. It has been described how to map the algebraic forms extracted from queries to the optimization system. This paper describes how to map the algebraic forms obtained from queries to the optimization structure.

Many algebraic optimizations are also addressed, such as fusion cascading map-reduce jobs into one single job and summarizing combine function structure from a map-reduce job's reduce function. For deductive and relational database structures, [12] proposes an incremental estimation algorithm for computing the view obtained from the relations. The total number of alternative derivations for each of the derived tuples in the view is tracked using a counting algorithm. In the query evaluation process, the algorithm operates for both set and duplicate semantics. In this article, negation and aggregation are used to explore another algorithm for nonrecursive views. Since it precisely computes the result, the algorithm produces an optimum result. Since it precisely computes the view tuples inserted in the database, the algorithm produces an optimal answer. Another algorithm, known as the rederive algorithm, has been developed for incremental recursive view management.

In paper [13] the discussion focuses more about recursive delta-based computing. The delta adjustments are primarily used to facilitate iterative computations between iterations, and the state is easily modified in an extensible manner. This paper presents a programming model that details the implementation and optimization of these queries in REX's runtime environment. In the REX runtime scheme, failures are

treated gracefully. A cost-based optimizer called Comet [14] is discussed that shares computations at the SQL level and at the Map Reduce level to eliminate redundancies. In [15], a rule-based optimizer takes advantage of similarities between input tables and operators of the same partition key. Almost all generic database optimizers don't pay attention to iterative queries in particular.

The architecture of effective map reduce algorithms for data processing, deep learning, and related joins are described in [16]. The main function of the Map Reduce system are often executed on a single master computer, where data has been preprocessed until the map functions are called. This paper discusses map reduce algorithms in the sense of data mining, such as frequent pattern mining, sorting, probabilistic modeling, and graph analysis. HiveQL, a SQL-like declarative language, has been thoroughly explored in [17], and it supports collections such as arrays and maps ,tables containing primitive types, and nested compositions.

It contains a metastore that holds the schema and statistics needed for data exploration, query compilation and query optimization. The query generator generates the execution plan using the metadata contained in the Metastore. Finally, the tasks are completed on the order on which they are dependent. Only after any of the dependent tasks' prerequisites have been completed, the task performed. A map/reduce task does serializing the portion of the plan into an xml file referred as plan.xml. This file was further added to the task's job cache, and then Hadoop instances of ExecReducers and ExecMapper are created. In [18], the incremental maintenance view is addressed. A database ring is built and used as the basis for query calculus architecture for efficient aggregate queries.

The properties of a ring with a regular form of polynomials and calculating inverses for delta queries are inherited by the query calculus. It also gets rid of costly query operators like joins, which are used to compute incremental view maintenance. The ring database's algebraic structure is required, and it is then bound to ring to form an aggregative query calculus.The technique in [19] discusses data intensive processing and addressing large-data issues in depth, and the distinct solution for those problems is illustrated.. If the related dataset was too massive to be stored in memory for data intensive retrieval, it was saved to disk.

As a result, it is preferable to prevent random data access and arrange data processing computations sequentially. This paper discusses data centre reliability .The suggested methodology [20] has a detailed discussion of a scalable distributed architecture for learning models from massive datasets.Parallel Learning of Tree Ensembles with Map Reduce is a distributed computation architecture that describes and implements distributed computations using the map reduce model.

In [21], a new architecture called Spark was proposed, which supports applications for maintaining MapReduce's scalability and fault tolerance. Resilient Distributed Datasets is a new abstraction introduced by Spark. It's a read-only set of objects that's partitioned around computers and can be restored if one of them is destroyed. The majority of technologies used to execute large-scale data-intensive applications use an acyclic data flow model, which is ineffective for these applications. The reuse of data through several parallel operations is the key subject of this paper.

3. Conclusion

In this literature, survey various approaches and methodologies used for query processing and optimization in big data is discussed and every approach has its own advantages and disadvantages. Query answering in big data should be made feasible regardless of the data size by incorporating the concept of scale independence. Query optimization in big data is mostly concerned with the probability of efficiently extracting a subset from a massive dataset. The complexity and heterogeneity of data, which includes both structured, semi-structured and unstructured data makes query optimization difficult. Big data queries are often programmed to process huge collections of structured, semi-structured and unstructured data, and it was built as a batch-processing framework that doesn't lend itself to quick data analysis. Most of the query engines in big data environments are not capable of maximizing speed when compared to relational databases at certain circumstances. The reason is that the queries are to be translated to map reduce for execution. The survey concludes that big data queries can be optimized at the processing level by applying scale independence and computation time can be reduced during the execution by using the approaches which helps in reducing the computation time.

Declarations

Ethics approval and consent to participateNot applicable.

Not applicable.

Consent for publication

Not applicable.

Availability of data and materials

Not applicable.

Competing interests

Not applicable.

Funding

Not applicable.

Authors' contributions

Not applicable.

Acknowledgements

Not applicable.

References

1. Armbust M, Liang E, Kraska T, Fox A, Micahel J, Franklin DA Patteson,"Generalized scale independence through incremental precomputation", Proceedings of Special Interest Group on Management Of Data, 2013, pp. 625–636.
2. Armbrust M, Curtis K, Kraska T, Fox A, Franklin MJ, Patterson DA, "PIQL: Success-tolerant query processing in the cloud", Proceedings of the Very Large DataBase Endowment, Vol. 5, Issue 3, 2011, pp. 181–192.
3. Armbrust M, Tu S, Fox A, Franklin MJ, Patterson DA, "PIQL: A Performance Insightful Query Language", Proceedings of the International Conference on Management of Data, 2010, pp.1207–1210.
4. Benedikt VBarany,M, and P. Bourhis,"Access patterns and integrity constraints revisited" Proceeding of the 16th International Conference on Database Theory, 2013, pp.213–224.
5. Deutsch A, Ludascher B, Nash A. Rewriting queries using views with access with access patterns under integrity constraints. Lecturer notes in Computer Science. 2005;3363:352–67.
6. Chen.Li. Computing complete answers to queries in the presence of limited access patterns ". The Very Large DataBase Journal. 2003;12(3):211–27.
7. Fox,D.A.Patterson.N.Lanham MArmbrust,A,B.Trusshkowsky,J.Trutna and Haruki.Oh,"SCADS:Scale independent storage for social compuitng applications",Proceedings of 4^{rth} Biennial conference on innovative data systems research,2009.
8. Abouzeid A, Bajda-Pawlikowski K, Abadi D, Silberschatz A, Rasin A, "HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads", Proceedings of Very Large DataBase Endowment,2(1):922–933, Aug. 2009.
9. Bu Y, Howe B, Balazinska M, and M. D. Ernst,"HaLoop: efficientiterative data processing on large clusters", Proceedings of Very Large DataBase Endowment 3(1–2)pp.285–296, Sept. 2010.
10. Ekanayake J, Li H, Zhang B, Gunarathne T, Bae S-H, Qiu J, andG. Fox, "Twister: a runtime for iterative MapReduce, Proceedings of High Performance and Distributed Computing",pp. 810–818, 2010.
11. Fegaras L, Li C, and U. Gupta,"An optimization framework for Map-Reduce queries",Proceedings of Extending Database Technology, pp. 26–37, 2012.
12. Gupta A, Mumick IS, and V. S. Subrahmanian,"Maintaining views incrementally", Proceedings of Special Interest Group on Management Of Data, pages 157–166, 1993.
13. Mihaylov SR, Ives ZG. and S. Guha,"REX: Recursive,delta-based data-centric computation",Publication of Very Large DataBase,5(11):pp.1280–1291,2012.
14. He B, Yang M, Guo Z, Chen R, Su B, Lin W, and L. Zhou,"Comet: batched stream processing for data intensive distributed computing",Proceedings of Symposium On Cloud Computing, pp.63–74, 2010.
15. Lee R, Luo T, Huai Y, Wang F, He Y, and X. Zhang,"YSmart: Yetanother SQL-to-MapReduce translator", Proceedings of International Conference on Distributed Computing System, pp.25–36, 2011.

16. K. Shim,"MapReduce algorithms for big data analysis", Publication of Very Large DataBase,5(12):pp.2016–2017, 2012.
17. Thusoo A, Sarma JS, Jain N, Shao Z, Chakka P, Zhang N, Anthony S, Liu H, and R. Murthy. Hive - a petabyte scale datawarehouse using Hadoop|, Proceedings of. International Conference on Data Engineering, pp.996–1005, 2010.
18. Christoph, Koch,"Incremental Query Evaluation in a Ring of Databases",Proceedings of Principle Of Database Systems,pp.87–88,2010.
19. Lin J and C. Dyer,"Data intensive text processing with MapReduce",Synthesis Lectures on Human Language Technologies,pp.177,2010.
20. Panda B, Herbach JS, Basu S. and R. J. Bayardo,"PLANET:massively parallel learning of tree ensembles with MapReduce". Publication of Very Large DataBase. 2009;2(2):1426–37.
21. Zaharia M, Chowdhury M, Franklin MJ, Shenker S, andI. Stoica,"Spark: cluster computing with working sets",Proceedings of HotCloud, pp.10–10, 2010.