Optimization Techniques for Complex Queries in Redshift and BigQuery

Author:

Rameshbabu Lakshmanasamy, Senior Data Engineer, Jewelers Mutual Group

Abstract:

Query optimization becomes particularly relevant in big data analytics systems due to the need that is required. Such environments are faced with queries involving joins aggregation and sorting of terabytes or petabytes of data. If we have them in their nonoptimized forms, finishing these queries could take hours or days if the results are required in a real-time or near real-time fashion. Query optimization covers data retrieval processes and eliminates the difficulties of searching for data not required, hence making them faster and charging less computational power.

Keywords:

Amazon Redshift, Google BigQuery, Query Optimization, Performance, MPP

Introduction:

Query optimization is one of the most crucial steps in database management and deals with the optimization of the queries that are submitted to the database system. This is especially ideal for massively parallel processing complex structures such as Amazon Redshift and Google BigQuery, where data is queried at a large scale. Query optimization makes sure that these complicated operations take the least amount of utilization of, for instance, CPU time, memory, and I/O operations and, at the same time, take the shortest time possible. The purpose is to minimize query response time, cut costs, and be more efficient with the resources that the system has (Ibrahim & Aoun, 2022).

Although Redshift and BigQuery are both designed to handle large analytical queries at scale, both use distinctly different mechanisms and architectural structures to achieve high levels of performance. Redshift works on the architecture of a distributed, columnar storage system which directly impacts upon I/O operations with efficient techniques such as, sort keys as well as distribution keys. However, BigQuery works in the fully serverless mode, which means that the current existing resource allocation process depends on the query complexity. It does this in such a way as to allow for less intervention when scaling and so results in more seamless scaling. Additionally, the product has a complex query processing system based on materialized views and partitioning that contributes to the depreciation of the query time as well.

Comparative Study of Query Optimization Strategies

Optimizing queries is very important when handling big data since this determines the speed with which intricate data sets are processed. Amazon Redshift and Google BigQuery are two of the most widely used and highly effective cloud-based data warehouses that have splendid query optimization options available for use (Jindal et al., 2019). Despite the effective filtration of large datasets and the ability to provide quick query responses, both platforms have various optimization techniques resulting from architectural differences. Here below is a comparison between the optimization techniques of each.

Amazon Redshift

Columnar Storage

Amazon Redshift also uses a columnar storage format, and this will help in the performance of the queries. Redshift columnar storage is the opposite of the typical row-based storage, where the entire rows are read during a query; only the required columns are pulled. This significantly decreases the number of requests that require a disk read, enhancing I/O optimization as well as query processing, where numerous columns are in use on big datasets. Columnar storage is optimized for analytical I/O because analytical workloads ultimately will only process subsets of the column in these large complex concatenation and filtering computations.

Sort Keys and Distribution Keys

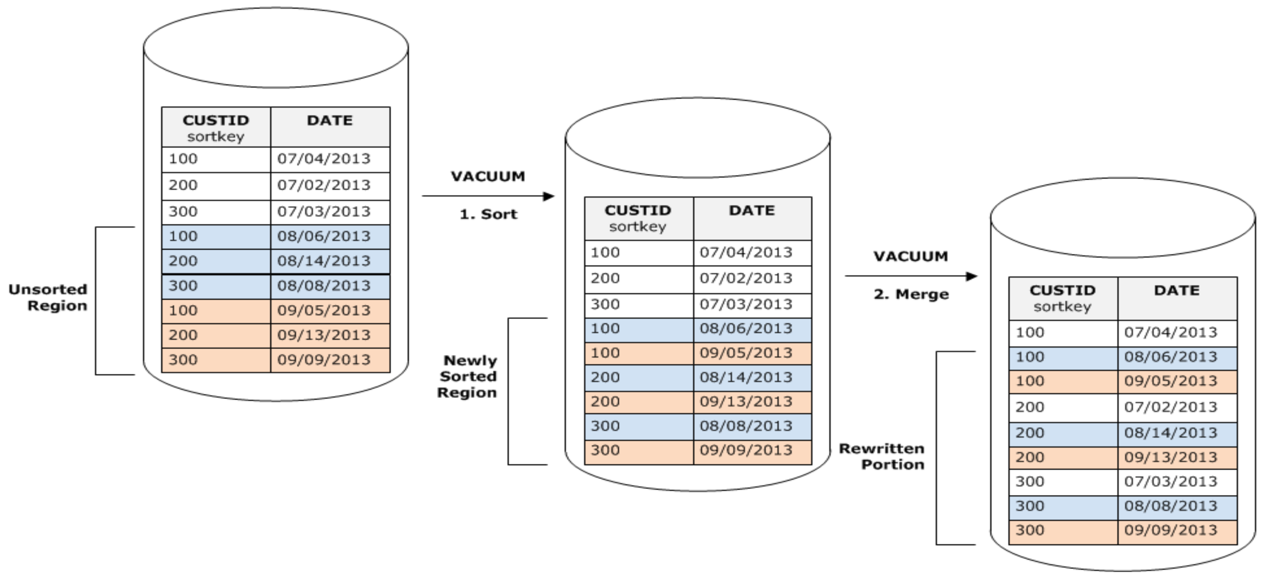
Sort and distribution keys play an important role in Redshift's data distribution and query optimization. Sort keys define the way data are arranged in the actual table so that inquiries can directly direct themselves to ascertain information without having to hunt for it physically. Sort keys save the time necessary to process a query because the data is grouped depending on how users are most likely to search for it.

Distribution keys, on the other hand, dictate the distribution of data within an environment of Redshift nodes. In Redshift, an appropriate distribution key is chosen to reduce the need to shuffle data between nodes (which can be time-consuming during join or aggregation). The targeted placement cuts access amounts and, therefore, quickens the execution time of the Query for huge joins and aggregations.

VACUUM and ANALYZE Commands

In Redshift, simple manual operations like the `VACUUM` and `ANALYZE` statements are needed to optimize query performance. Suppose data is deleted or updated within a table in Redshift. In that case, the data becomes fragmented and thus scattered throughout the system, which the `VACUUM` command will address by ensuring that queries point to data stored on contiguous blocks of storage. This process minimizes the I/O overhead and increases the rate of subsequent Query.

The `ANALYZE` command calculates statistics on table information used by the Redshift query planner to make improved choices when perusing queries. New statistics include information needed by the query optimizer to understand the data distribution and the number and size of tables.



CUSTID sortkey	DATE
100	07/04/2013
200	07/02/2013
300	07/03/2013
100	08/06/2013
200	08/14/2013
300	08/08/2013
100	09/05/2013
200	09/13/2013
300	09/09/2013

CUSTID sortkey	DATE
100	07/04/2013
200	07/02/2013
300	07/03/2013
100	08/06/2013
100	09/05/2013
200	08/14/2013
200	09/13/2013
300	08/08/2013
300	09/09/2013

CUSTID sortkey	DATE
100	07/04/2013
100	08/06/2013
100	09/05/2013
200	07/02/2013
200	08/14/2013
200	09/13/2013
300	07/03/2013
300	08/08/2013
300	09/09/2013

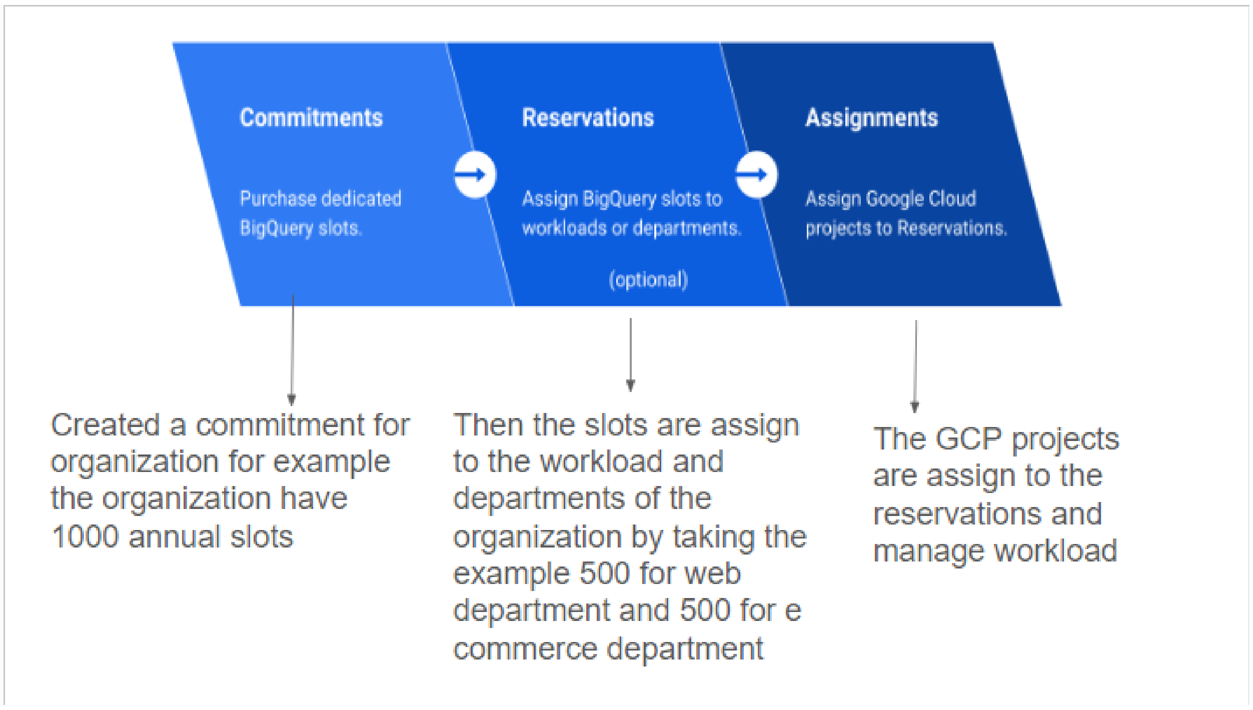
Google BigQuery

Serverless Architecture

Google BigQuery has a fully managed, serverless architecture to enable the scaling of query solutions and query demand. Users do not have to allocate resources or request capacity; rather, BigQuery allocates resources based on the size of the Query made by the user and the resulting demand for resources from the server. This serverless approach means that performance stays strong when posed with large datasets to search through. A traditional service requires manual fine-tuning to ensure premium performance during periods of increased processes to prevent degradation in service delivery; JIT does not have a lot of configurations that is associated with manual setup when workloads change from low to high and the other way around.

Slot Reservation

BigQuery uses computational slots which are portions of CPU and RAM that are engaged to run queries. Reserved slots enable users to predict query demands and schedule the slots to meet those demands for huge datasets or frequently running queries. The operation of slots allows users to work according to prioritized strategies where some important queries are given adequate numbers (Soma, 2022). Such a level of resource management provides manageable query characteristics in a multi-user or multiprocessor system environment.


Commitments
Purchase dedicated BigQuery slots.
Reservations
Assign BigQuery slots to workloads or departments.
(optional)
Assignments
Assign Google Cloud projects to Reservations.
Created a commitment for organization for example the organization have 1000 annual slots
Then the slots are assign to the workload and departments of the organization by taking the example 500 for web department and 500 for e commerce department
The GCP projects are assign to the reservations and manage workload

Clustered and Partitioned Tables

By using clustering and partitioning, BigQuery can improve the capabilities of queries when used on very huge datasets. Partitioning enables splitting a table into several sub-tables that can be managed easily based on a criterion like date. This allows BigQuery to read only those particular partitions when serving a query, which lowers the time and cost associated with such operations.

It is based on the idea that one can order table data by selected columns so that BigQuery can easily find the data for filtering operations. In large queries, clustering and partitioning allow only a fraction of the data to be scanned while improving queries' results and response times.

Best Practices for Writing Efficient Queries

Efficient query optimization both in Amazon Redshift and Google BigQuery is a necessary prerequisite for dealing with truly enormous data volumes and keeping the costs of data processing in check. Both platforms have their advantages, and each can be used in different extended ways, but some common practices can make complex queries work more efficiently (Soma, 2022). Here are the detailed tips for Redshift as well as BigQuery to get a fast and scalable query performance.

Amazon Redshift

Use Appropriate Distribution Keys

Selecting a good distribution key is crucial to prevent the shuffling of data from one node to another during query processing. Redshift also spreads data across multiple compute nodes. If the distribution key is not chosen correctly, there will be a lot of network interconnect I/O, and a major part of the Query, say join, will exhibit low performance (AWS Big Data, 2020). If the distribution key is chosen as a column often involved in joins or aggregations, the data will be colocated on one node, which means a lot of movements are avoided so that the system will be fast.

Leverage Compression

Redshift supports columnar compression, where the data is compressed on columns using the `COPY' command and the LZO (Lempel-Ziv-Oberhumer) algorithm. Due to small disk space, it occupies a minimum amount of I/O during the query execution; hence, it leads to optimum results. As for the big data, the unused might should be preloaded with the compression enabled, as it decreases the time needed to read info into the memory and makes the general inquiry reaction much faster.

Limit the Use of `SELECT `

A classic error when writing queries is using `SELECT ` which means getting all records from this table; this will make Redshift conduct a scan on all those columns that are not required. Wherever possible, you should always declare only the columns you want to use in a query because the rest will only slow down the Query (Soma, 2022). By reducing the portion of the dataset that is assessed, query response time is faster, and the overall workload is minimized.

Batch Updates

Small and recurrent `INSERT` operations could be problematic because each transaction involves some overhead in Redshift. However, loading data in batch fashion before inserting it into Redshift has shown to have a big impact on improving performance. When operating on large data sets, larger batch sizes mean fewer I/O operations and less time to commit the data. As a result of this approach, large-scale data changes and continuous data update environments are effectively dealt with while maintaining the system's responsiveness.

Google BigQuery

Avoid Repeated Queries

When using complex queries, increased resource use may translate into higher costs and decreased efficiency when the same Query is run several times. To this end, BigQuery has query result caching, which is at the center of this case. Big Query caches a query, and if the same Query is run and the data has remained unaltered, then this database will not run the Query again but will provide the result that was run before (Google Cloud-BigQuery, 2022). This, in turn, minimizes the time taken to generate the Query and response time in subsequent querying, hence more beneficial for queries that are frequently used on dashboards and reporting.

Use Partitioning

When dealing with large volumes of records, especially those with time-related data, partitioning tables by an often-filtered field like date has a huge optimization effect. By partitioned definition, when an analyst performs a query, only that partition is searched, and hence, few bytes are traversed, which makes it fast (Google Cloud-BigQuery, 2022). This is especially advantageous if the analytics workloads relate to data' certain temporal periods or any other pre-identified data sub-sets.

Optimize JOINs

JOIN operations are generally costly concerning the volume of computations needed to be performed. The feature of BigQuery is that, unlike some other SQL bases, it permits the utilization of the `WITH` clause that defines subordinate queries as CTEs, which can be referenced several times in the primary Query. Big Query allows subqueries to be precomputed using `WITH`, thus making it easier to optimize and minimize on redundancy of initial queries (Smallcombe, 2022). Making the content read-only helps in making the Query easier to understand and, at the same time, will help in the execution of the Query with BigQuery's use of CTEs.

Streaming Inserts

Streaming small inserts in BigQuery in real-time might have some problems as the system has to process each record. To increase the efficiency of data loading, it is recommended to split these inserts into bigger units before sending them to BigQuery. This makes the overhead involved in streaming minimal as well as making data ingestion even easier. While processing streaming workloads act to collect data in increments (e.g., by incorporating a staging table or a storage) before submitting them to BigQuery for efficiency.

Additional Optimization Techniques

Indexing in Amazon Redshift

Unlike other related databases, Amazon Redshift does not use indexes as they are defined in other associated databases; however, it uses sort keys. If sort keys are described well on the current most-executed columns, then performance like that of indexes is possible. While storing the data, sort keys order the data set according to how it is patterned within the Query so that Redshift can read the data conveniently (Kulkarni, 2023). When queries include sort key columns, then Redshift can avoid reading through irrelevant blocks of data, which has resulted in reduced disk read-in operations. Not only does this reduce the query response time, but it also reduces I/O operations, which is a very important feature in this storage structure.

Denormalization

Denormalization is a process that is one of the most commonly applied in data warehousing, the goal of which is to transform the data structure with the help of which it becomes simpler to implement queries. This is true in both Redshift and BigQuery because denormalized data means fewer joins on the fly during query processing. For example, star schema design, which combines fact tables that are directly related to more than one-dimension tables, is efficient and fast to query.

This means that the amount of joining can be reduced when users apply flattened data structures to get the desired data. Denormalization also has a positive effect on query writing because there is no need to join multiple tables;

joining is done on one table. However, what can be gained through higher read performance through denormalization—duplication of data and possibly extra work with modification of records—cannot go unmentioned.

Conclusion:

In conclusion, Amazon Redshift and Google BigQuery have different approaches to query optimization based on each platform's architecture. Redshift can use sort keys with configuration instead of automation; on the other hand, BigQuery is a serverless solution with slot reservations. LIA and MIA best practices concern avoiding data scans, using features that are available exclusively for one or another platform, and setting data distribution. Others, like denormalization and analysis of data skew, have an even better impact. The integration of all the highlighted approaches makes it possible for organizations to work with queries in a more efficient, cost-effective way, allowing for the scaling up of analysis to meet the current and future scale of demand while at the same time trying to facilitate the data-driven decision-making process.

References:

1. AWS Big Data. (2020, August 28). Top 10 performance tuning techniques for Amazon Redshift. Amazon Web Services. https://aws.amazon.com/blogs/big-data/top-10-performance-tuning-techniques-for-amazon-redshift/
2. Google Cloud-BigQuery. (2022). Introduction to optimizing query performance | BigQuery. Google Cloud. https://cloud.google.com/bigquery/docs/best-practices-performance-overview
3. Ibrahim, F., & Aoun, M. (2022). Improving query efficiency in heterogeneous big data environments through advanced query processing techniques. Journal of Contemporary Healthcare Analytics, 6(6), 40-64. https://publications.dlpress.org/index.php/jcha/article/view/48
4. Jindal, A., Viswanathan, L., & Karanasos, K. (2019). Query and Resource Optimizations: A Case for Breaking the Wall in Big Data Systems. arXiv preprint arXiv:1906.06590. https://arxiv.org/abs/1906.06590
5. Kulkarni, A. (2023). Amazon Redshift: Performance Tuning and Optimization. International Journal of Computer Trends and Technology, 71(2), 40-44. https://www.researchgate.net/profile/Amol-Kulkarni-23/publication/369211993_Amazon_Redshift_Performance_Tuning_and_Optimization/links/66005630a4857c7962741127/Amazon-Redshift-Performance-Tuning-and-Optimization.pdf
6. Smallcombe, M. (2022). 15 Performance Tuning Techniques for Amazon Redshift. Integrate.io. https://www.integrate.io/blog/15-performance-tuning-techniques-for-amazon-redshift/
7. Soma, V. (2022). Comparative Study of Big Query, Redshift, and Snowflake. North American Journal of Engineering Research, 3(2). http://najer.org/najer/article/view/34