ISSN: 2634 - 8853

Journal of Engineering and Applied Sciences Technology

SCIENTIFIC Research and Community

Review Article

Open Access

Impact of Columnar Storage Optimizations in Redshift and BigQuery

Rameshbabu Lakshmanasamy

Senior Data Engineer, Jewelers Mutual Group, USA

ABSTRACT

With the data explosion, companies are utilizing "new age" data warehouses, such as Amazon Redshift and Google BigQuery, to process this information. Space-efficient storage structures (specifically column-oriented storage formats) are the path to improved query performance and lower costs. Columnar storage stores the data sorted by columns and not rows so that only the columns being queried are accessed when a query is made. Fewer disk reads mean faster execution time, and that's good for analytical processing.

***Corresponding author**

Rameshbabu Lakshmanasamy, Senior Data Engineer, Jewelers Mutual Group, USA.

Received: March 04, 2024; **Accepted:** March 11, 2024; **Published:** March 25, 2024

Keywords: Amazon Redshift, Google BigQuery, Columnar, Compression Techniques, JSON, Parquet, ORC

Introduction

Compression methods such as LZO, Zstandard, or auto-algorithm enable columnar storage to take up even less physical storage space. These optimizations are beneficial and crucial for accommodating different data types, namely structured, semi-structured, and unstructured data. As business demands real-time insights and the intricacy of the data is growing exponentially, storage optimization is a vital factor of performance and scalability [1,2]. Naturally, cloud data warehouses can accommodate these needs, but only if the right approaches are utilized to make them efficient and cost-effective.

Columnar Storage Overview

Two principal methods of organizing and storing data have emerged in data management: commercially available DBMSs, divided into row-based and columnar storage. Row-based storage locates data in rows; all the records' fields are stored in a strip. Columnar storage, on the other hand, works concerning columns; each column is stored continuously. It is crucial to make a distinction here as it makes much difference, mainly when focusing on the currently popular analytical databases like Amazon's Redshift or Google's BigQuery [2]. Knowing the differences between these storage models explains why columnar storage has emerged as the standard storage model in data analytical environments.

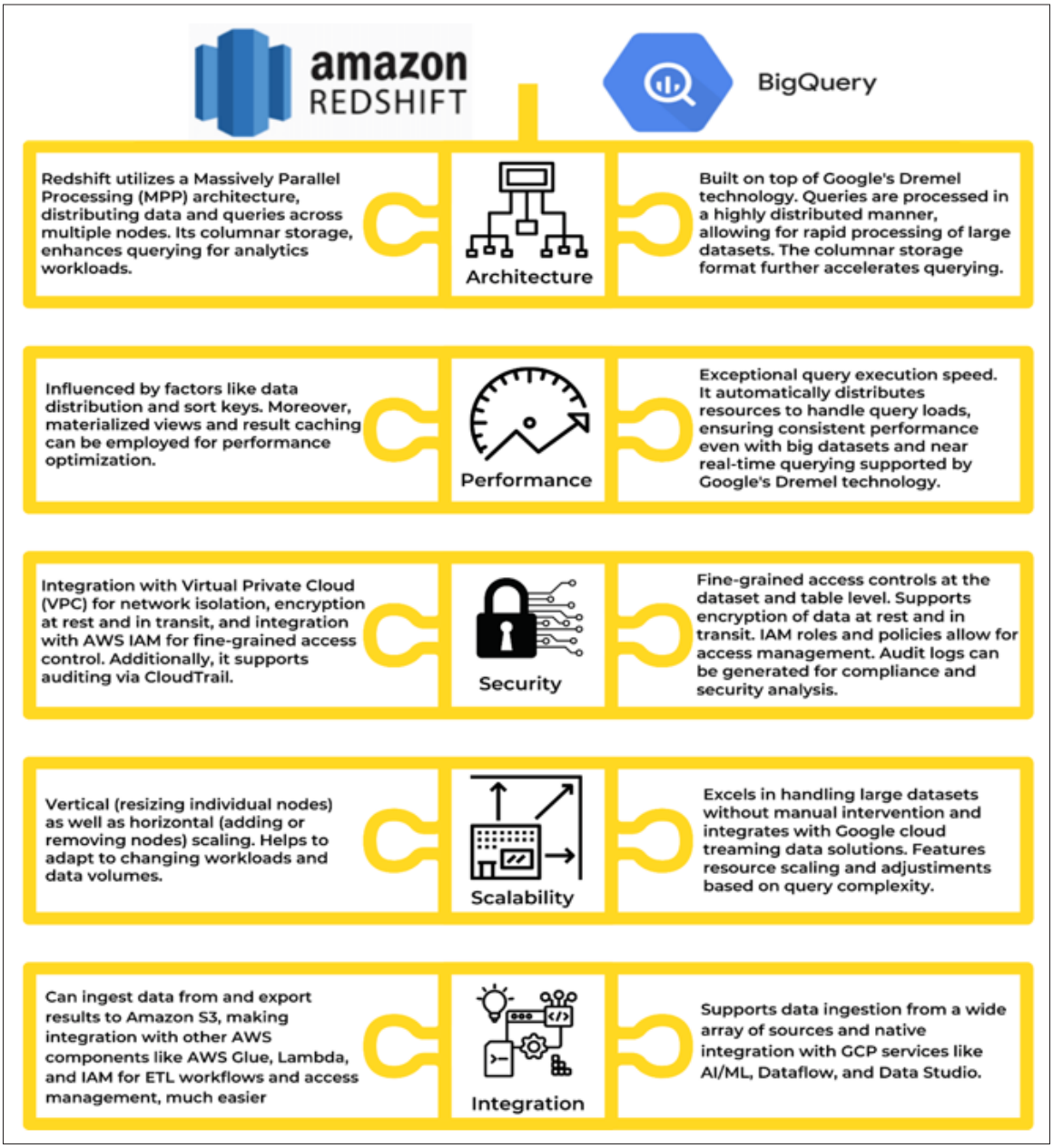

Figure 1

Row-Based Storage

Relational databases like MySQL and PostgreSQL use row-based data storage, meaning all data in a row is stored in one location. However, it is most potent when used in transactional database environments, mainly for inserting, updating, or deleting whole records. In such conditions, fetching complete records is optimal because all fields are scanned through. However, row-based storage is not as efficient when used with applications that integrate complex analytical operations [3]. Analytical queries primarily work with the computation of values from many rows but sometimes target a limited number of columns. For instance, a query may only need two or three columns out of hundreds of columns in a table. The row-based system reads all the columns in a row and, therefore, conducts unnecessary I/O operations, yielding the query slow.

Columnar Storage

Columnar storage, on the other hand, keeps data along the columns while storing it rather than along the rows. To be more precise, each column in the table is saved individually, so, for example, the values of the first and second columns are stored in two separate locations. For instance, the values in the "CustomerID" column are grouped successively with each other, followed by the "PurchaseDate" values and then the "Amount" values. If the extent of a query is typical of rows from specific columns, the database engine can read only the columns in question and ignore the rest, which may be unnecessary [4]. This minimizes I/O operations and improves the query, especially for big data case scenarios where typical queries implemented include filtering, summarizing, or even aggregation on big data sets.

Advantages of Columnar Storage

Columnar storage is particularly favorable for analytical workloads as it is implemented. First, it reduces I/O operations because only the required column can be accessed, optimizing query performances. Second, it supports high compression, which means less storage space and faster data computation [3]. Third, it is best suited for cases where massive amounts of data are present, but only a few columns need to be computed for reporting or data mining purposes.

Compression Techniques and Their Effect on Query Performance

Data compression is one of the most critical factors in today's data warehouses because it impacts storage costs and performance queries. To reduce the volume of data scanned at execution time, stored data is compressed, which helps minimize the time needed to provide responses and utilize resources. Like any other distributed data processing systems, Amazon Redshift and Google BigQuery utilize the enhanced provision of compression algorithms, each implemented in different manners and approaches [5]. As such, this section looks into some of the most prevalent compression methods within these platforms and scales on query performance.

Redshift Compression Techniques

Amazon Redshift offers several compression techniques that reduce storage use while improving query performance. These techniques are particularly effective when applied to column-oriented databases, where data is stored in columns, and similar data types are combined to provide enhanced compression.

LZO (Lempel-Ziv-Oberhumer Compression) is one of many Redshift compression techniques, and it is used very often. In this aspect, LZO hash is a unique characteristic that provides an excellent middle ground regarding the speed and compression rates acquired. It may not offer the best compression rates, but it works very fast and does a fantastic job of shrinking the size of a dataset. Because LZO compresses data blocks independently, it is possible to read and write data in parallel, thus improving system queries. This makes it an excellent choice for all scenarios where query speed is more essential than the maximum extent of storage optimization.

There is also Zstandard (ZSTD), a commonly used method that offers similar compression ratios to LZO but with a faster decompression time. ZSTD is more efficient when dealing with data comprised chiefly of text or data with high similarity. It does so at a much lower compression ratio than LZO, implying that fewer kilobytes must be scanned and processed to complete a query.

Also, Redshift often employs Run-Length Encoding (RLE) when data is rather sequential or full of repeated values in the column. For example, a column of numbers that contains dates in a sequential order is perfect for RLE as it collapses the large sequences into ranges rather than the individual. This eliminates a lot of space needed to store data and also provides the benefit of faster execution time as less data is scoured. Redshift optimizes the storage space of significant and complex datasets using these compression techniques, and they don't use much memory and CPU power during the queries. The compressed data helps optimize the key parameters and provides higher scanning rates and improved analytical workloads.

BigQuery Compression Techniques

Google BigQuery also uses different types of compression techniques to reduce the space and time required to perform queries. More complicated than RedShift, BigQuery channels more of its efforts on internal optimization and automatically learning how to compress data.

Cap 'n Proto is a schema-based, optimal format used in BigQuery. It is optimized for structured data, so BigQuery stores data in the most compact way, occupying less space and time for recruiters. Cap' n Proto is capable of effectively handling complex data types, making it a major component of BigQuery's compression toolkit.

Moreover, BigQuery is based on well-known internal storage formats used by Google, including Protocol Buffers designed for specific data types. These formats work together with BigQuery's columnar storage system, and the best compression format for the data stored is chosen. This leads to efficient storage and improves the response time of queries that must be run on the database to generate the summarized information.

One of BigQuery's greatest strengths is that it has self-serve compression and data optimization. It also works natively at scale and can automatically apply compression techniques in the background depending on the patterns of use and the data types concerned. Note the absence of any penalty for this kind of data. This ensures that storage or query execution efficiency is not compromised.

Impact on Query Performance

There are specific data compression techniques in Redshift and BigQuery by which query performance is enhanced by minimizing the data scanned. Fewer I/ O operations are needed as there are fewer records. As a result, query response time is faster. For instance, data compression using Zstandard compression in Redshift can lead to reducing query times by 40% compared to the time needed to query uncompressed tables [6]. Second, indexing and self-optimization by BigQuery also lead to significant performance improvement, mainly when BigQuery works with large numbers of identical data sets.

Moreover, compressed data requires fewer resources to be managed: it decreases memory intensity and CPU utilization for queries. In platforms like BigQuery, pricing is based on the number of queries made, and compression can make querying large datasets cheaper by minimizing the amount of data processed. Therefore, compression techniques become mandatory for enhancing performance, decreasing overall costs, and increasing the efficiency of contemporary data warehouses.

Storage Cost Comparisons for Various Data Types and Sizes

Amazon Redshift and Google BigQuery differ in their pricing policies and features, which are directly tied to their prices, especially for storage at an increasing scale. It is crucial to understand how these platforms deal with the expense of archiving and querying large datasets depending on their dimensionality, sparsity, and whether those datasets have been compressed in some fashion.

Cost Models of Redshift vs. BigQuery

Amazon Redshift works under the pay-as-you-go model for both storage and computing. Customers can select between Spot instance pricing or reserved instances, which provide a lower price for dedicated use of a facility in a time interval (1 or 3 years) [7]. It offers fixed and sometimes underestimated costs for constant operations but can lead to resource wastage if not well monitored.

Google BigQuery, on the other hand, adapts a usage-based billing model that uses data processed by queries and the storage consumed by the queries. This model is ideal for random jobs but can prove very costly for searches, especially where the data being queried is significant and needs to be compressed for optimality.

Compression and Data Types

The compression efficiency in the two platforms differs in terms of data type. Some types of data are much more sensitive to the need for compression: text-intensive data and logs. Algorithms such as Zstandard in Redshift enable fees for storage space to be slashed, while big query's automatic compression allows the costs of queries to be slashed, too, because fewer bytes are scanned. Numeric data also compresses well, especially if using Redshifts' RLE, which adds to primary query performance, maintaining the benefits of less storage.

Both platforms suggest columnar storage formats for semi-structured data, such as JSON, Parquet, or ORC, which are more effective in terms of storage than a raw format. These formats shrink the data size and optimize the query time, especially when dealing with big data in Redshift and BigQuery.

Data Format Choices: JSON vs. Parquet or ORC

Parquet and ORC formats also have better compression and query performance than the raw JSON dataset. In Redshift, storing data in Parquet decreases costs and is also an appropriate approach in view of this platform's effective interaction. These formats result in lower query costs because BigQuery charges per byte processed and are ideal for large sets.

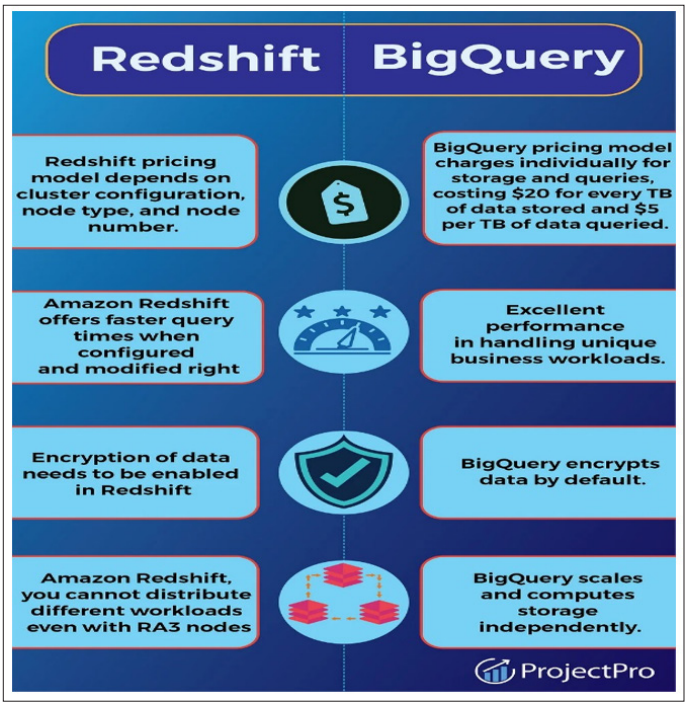

Figure 2

Impact of Data Growth on Storage Costs

When datasets increase, algorithms make the need to minimize storage facilities possible. In Redshift, storage cost is tied to the amount of data ingested, but using Zstandard will reduce the amount of space this data occupies. The reserved instances for large-scale workloads give the customer a predictable cost. In BigQuery, these storage costs increase with the data size, while pay-per-query model suggests that compressing the data into more efficient formats has reduced the price of querying large datasets, making compression critical in controlling costs.

Other Relevant Information

Apart from compression and storage cost models, two more factors affect Amazon Redshift and Google BigQuery, particularly when they have larger datasets. These are data ingestion, auto-scaling, performance tuning, and future storage trends.

Data Ingestion and Processing

Redshift: Data ingestion can be done in batch by using COPY commands from Amazon S3, AWS Glue ETL or streaming with Kinesis Data Firehose. That is why higher compression algorithms such as Zstandard and RLE are used after ingestion, but one has to tweak to make configurations for compression and then the performance of queries. Redshift Streaming for real-time processes enables the direct consumption and processing of streaming data for real-time analysis [8].

BigQuery: Data ingestion is easier, streaming APIs for an instantaneous connection and data loading, alongside Google Cloud Storage for batch data loading. BigQuery has the feature of compression, where it does it based on data structure, and it does not need much intervention. It works seamlessly with Google Cloud Pub/Sub and can pipeline data continuously with almost real-time rates without requiring manual processes to configure compression.

Auto-Scaling and Performance Tuning

Redshift: Redshift allows for Resource Scaling to be done Manually, in addition Redshift has Concurrency Scaling for bursting workloads and Elastic Resize. While workload manager facilitates the retasking of query handling, the AutoVacuum tool recovers storage space from effaced rows. Still, some manual operations are required to perform node management and query optimization.

BigQuery: BigQuery is a true serverless platform that operates by dynamically scaling compute on the fly and storage based on usage. Its adaptable design is based on a slot model, which distributes processing abilities resourcefully. This makes BigQuery work with automatic partitioning and clustering, and users usually do not need to fine-tune their access to the data for optimal query performance.

Future Trends in Storage Optimization

Hybrid Transactional/Analytical Processing (HTAP): Redshift and BigQuery are already transitioning to HTAP, which allows transactional datasets to be analyzed in real-time without needing a different system.

Machine Learning Optimization: Supported by Redshift's Automatic Table Optimization and BigQuery's ML capacities, storage, and query are being enhanced, and these improvements are being made dynamically in both platforms.

Conclusion

Amazon Redshift and Google BigQuery have made numerous enhancements to columnar storage to improve query response time and lower storage costs per query. Other operations of the same platforms apply even more refined strategies, such as Zstandard and RLE optimization in Redshift and Cap'n Proto with auto-compression in BigQuery.

Redshift: Extremely good for companies with clear, planned, and large volume workloads, Redshift provides a very fine-grained level of control over the storage and CPU parts of the cluster. Due to its reserved pricing model, Concurrency Scaling, and Workload Management, it is cost-effective for large datasets, and the compression cannot be configured from SQL.

As we have seen, BigQuery is better in dynamic and pay-per-use environments, where it has a serverless model architecture that self-scales. The pay-as-you-go model fits well in organizations with fluctuating work schedules. It eases the management of semi-structured data and allows real-time analysis without having to involve technical support.

References

1. Amazon Redshift (2019) Columnar Storage - Amazon Redshift. Amazon.com https://docs.aws.amazon.com/redshift/latest/dg/c_columnar_storage_disk_mem_mgmnt.html.
2. Armenatzoglou N, Basu S, Bhanoori N, Cai M, Chainani N, et al. (2022) Amazon Redshift re-invented. In Proceedings of the 2022 International Conference on Management of Data 2205-2217.
3. BigQuery (2022) Optimize storage in BigQuery. Google Cloud https://cloud.google.com/bigquery/docs/best-practices-storage.
4. Stitch (2022) Amazon Redshift vs. Google BigQuery: a comparison. Www.stitchdata.com https://www.stitchdata.com/resources/redshift-vs-bigquery/.
5. Mucchetti M (2020) BigQuery for Data Warehousing https://link.springer.com/content/pdf/10.1007/978-1-4842-6186-6.pdf.
6. Manjunath TN, Pushpa SK, Hegadi RS, Ananya Hathwar KS (2023) A Study on Big Data Engineering Using Cloud Data Warehouse. Data Engineering and Data Science: Concepts and Applications 49-69.
7. Soma V (2022) Comparative Study of Big Query, Redshift, and Snowflake. North American Journal of Engineering Research http://najer.org/najer/article/view/34.
8. Sprinkle (2023) Amazon Redshift vs Google BigQuery: A Comprehensive Comparison. Www.sprinkledata.com https://www.sprinkledata.com/blogs/redshift-vs-bigquery.