INTERNATIONAL JOURNAL OF COMPUTER ENGINEERING &TECHNOLOGY (IJCET)

ISSN Print: 0976-6367
ISSN Online: 0976-6375

Publishers of High Quality Peer Reviewed Refereed Scientific, Engineering & Technology, Medicine and Management International Journals

PUBLISHED BY

IAEME

IAEME Publication

Chennai, India

https://iaeme.com/Home/journal/IJCET

International Journal of Computer Engineering and Technology (IJCET)
Volume 11, Issue 3, May-June 2020, pp. 106-115, Article ID: IJCET_11_03_013
Available online at https://iaeme.com/Home/issue/IJCET?Volume=11&Issue=3
ISSN Print: 0976-6367; ISSN Online: 0976-6375; Journal ID: 5751-5249
Impact Factor (2020): 16.11 (Based on Google Scholar Citation)
DOI: https://doi.org/10.34218/IJCET_11_03_013

DATA COMPRESSION AND EFFICIENT BIG QUERY LOGIC

Harshavardhan Chinthalapalli

Data Engineer, Cognizant, USA.

ABSTRACT

Data engineering is a critical field that supports the efficient handling, processing, and retrieval of largescale datasets. This paper explores two key aspects of data engineering: data compression techniques and efficient querying logic for Big Data environments. By examining contemporary methods and tools in these areas, we aim to provide a comprehensive overview of how data compression can optimize storage and how effective querying logic can enhance data retrieval performance. The discussion includes an analysis of existing technologies, current challenges, and potential future directions.

Keywords: Data Engineering, Compression, Big Data, Query Logic, Storage, Retrieval, Optimization, Tools, Challenges, Trends

Cite this Article: Harshavardhan Chinthalapalli. (2020). Data Compression and Efficient Big Query Logic. *International Journal of Computer Engineering and Technology (IJCET)*, 11(3), 106–115.

https://iaeme.com/MasterAdmin/Journal_uploads/IJCET/VOLUME_11_ISSUE_3/IJCET_11_03_013.pdf

1. Introduction to Data Compression

Data compression is the process of reducing the size of data to save storage space or reduce transmission time. In data engineering, compression is vital for managing large datasets efficiently, especially in environments where storage costs are high, and query performance is critical.

The rapid growth of data in various domains has necessitated advanced techniques in data engineering to manage and analyse large volumes of information effectively. Data compression and efficient querying are crucial aspects of this process, impacting both storage costs and retrieval times. This paper delves into the state of the art practices in data compression and Big Data querying, highlighting their importance in modern data systems.

1.1 Data Compression Techniques

Data compression involves reducing the size of data without losing significant information, thereby optimizing storage and transmission efficiency. This section covers:

1.2 Lossless Compression Algorithms

Lossless compression algorithms reduce the size of data without losing any information. The original data can be perfectly reconstructed from the compressed data. This is crucial for applications where data accuracy is paramount, such as in financial records, text files, and other data sensitive applications.

1.1.1 Lossless Compression Techniques:

- RunLength Encoding (RLE): A simple technique effective for data with repeated sequences.
- Huffman Coding: Utilizes variable length codes for more frequent symbols to reduce overall size.
- LempelZivWelch (LZW): A dictionary based method widely used in formats like GIF and TIFF.

Example: In a data pipeline where sensor readings are collected every second, a sequence like "11111122333444" could be compressed to "1x6, 2x2, 3x2, 4x3," significantly reducing storage needs.

1.1.2 Lossy Compression Algorithms

Lossy compression reduces data size by removing less critical information, which may result in some loss of fidelity. This type of compression is commonly used in multimedia applications where exact replication of the original data is not necessary.

1.1.3 Lossy Compression Techniques:

- **Transform Coding:** Applied in audio and image compression (e.g., JPEG, MP3) to achieve higher compression ratios at the cost of some data loss.
    - JPEG (for images): JPEG compression reduces file sizes by discarding information that is less perceptible to the human eye.
    - MP3 (for audio): MP3 compression removes inaudible frequencies, compressing the audio data while maintaining a quality that is acceptable for most listeners.
- **Quantization:** Reduces the number of bits needed for representing data by approximating values.

Example: In a data engineering scenario, if an organization is storing a large number of images for a website, compressing these images using JPEG can save significant storage space.

2. Compression Algorithms in Big Data

In the context of big data, lossless compression algorithms like **Gzip, Snappy,** and **Zstandard** are commonly used because they allow for efficient storage without losing data accuracy. These algorithms are often integrated into data storage formats like **Apache Parquet** and **ORC**, which are designed for efficient storage and processing of large datasets in big data environments like Hadoop and Spark.

2.1 Data Compression in Big Data Contexts

Columnar Storage Formats: Parquet and ORC formats provide efficient compression for analytic workloads by storing data in columns rather than rows.

Compression in Distributed File Systems: Techniques used in Hadoop Distributed File System (HDFS) and other distributed storage solutions.

2.1.1 Apache Parquet

Parquet is a columnar storage format that provides efficient data compression and encoding schemes. It is optimized for reading and writing large datasets, making it a popular choice for big data applications.

Example: A Parquet file storing data from a retail application might compress string data using dictionary encoding and compress numeric data using delta encoding, significantly reducing file size.

2.1.2 Benefits of Data Compression

- **Reduced Storage Costs:** Compressed data requires less disk space, which can lead to significant cost savings, especially in cloud environments where storage is billed based on usage.
- **Improved Query Performance:** Compression reduces the amount of data that needs to be read from disk during queries, resulting in faster query execution times.
- **Efficient Data Transmission:** Smaller data sizes mean that less bandwidth is required for transmitting data over a network, which can be crucial in distributed systems.

2.1.3 Challenges and Tradeoffs

While data compression offers many benefits, it also comes with some challenges:

- **Processing Overhead**: Compressing and decompressing data requires additional processing power, which can increase CPU usage.
- **Complexity**: Implementing compression and ensuring it works seamlessly with other parts of the data pipeline can add complexity to the system.
- **Lossy Compression Risks:** When using lossy compression, there is a risk of losing important data, which may not be acceptable in all scenarios.

3. Efficient Big Query Logic

3.1 Introduction to BigQuery

Google BigQuery is a fully managed, serverless data warehouse that allows for superfast SQL queries using the processing power of Google's infrastructure. It is designed to handle large datasets efficiently, but optimizing query performance and managing costs are key challenges that require careful consideration of query logic. Efficient querying is essential for extracting valuable insights from large datasets.

3.2 Query Optimization Techniques for Query Performance

Partitioning and clustering tables can significantly improve query performance by reducing the amount of data that needs to be scanned. Efficient query logic is essential for minimizing costs and maximizing performance in BigQuery. Here are some techniques:

3.1.1 Partitioning:

This technique involves dividing a table into smaller, more manageable segments (partitions) based on a specific column, such as a date. Queries that filter on the partition column will only scan the relevant partitions, reducing query time and cost.

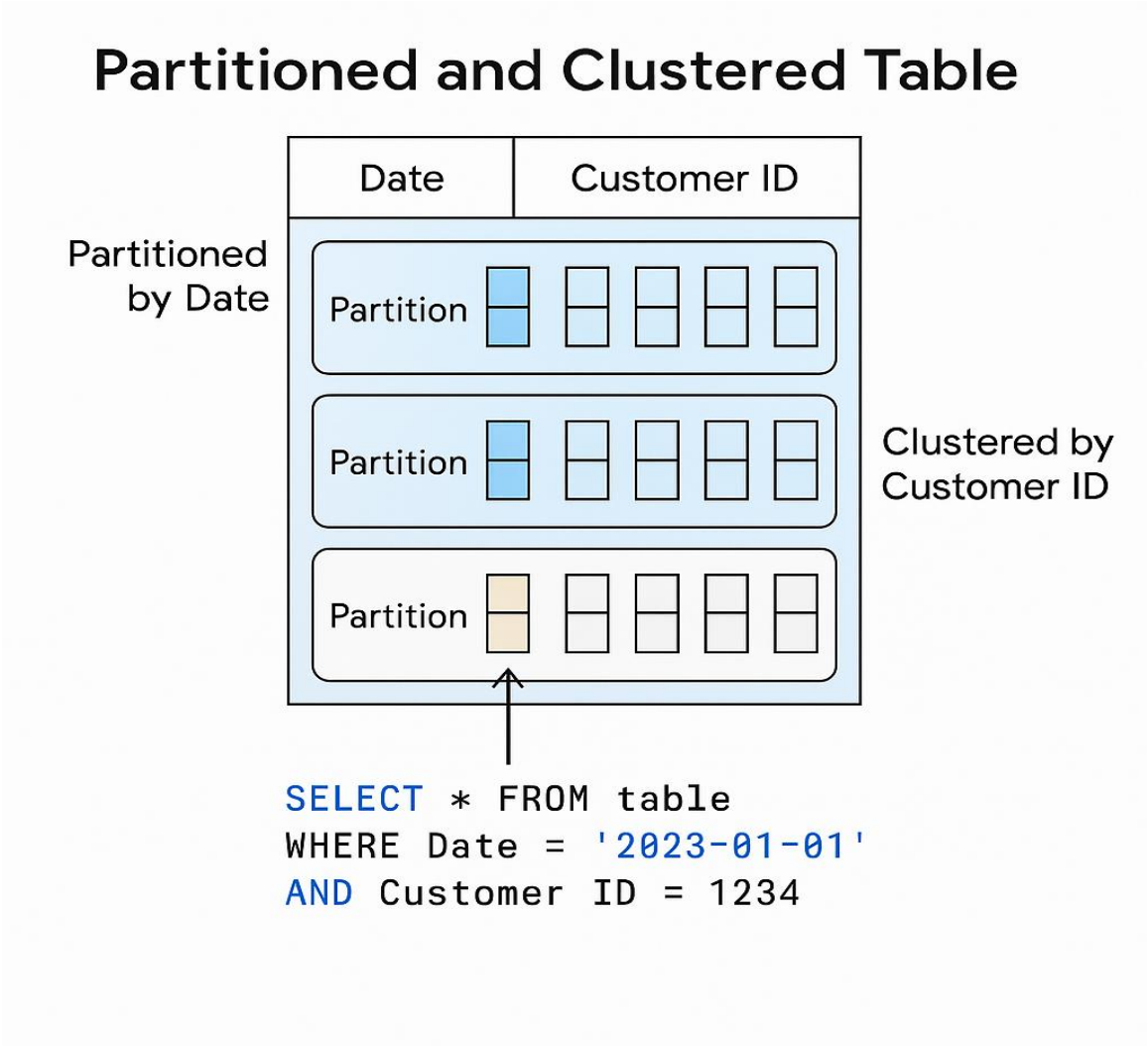
Example: A table containing log data can be partitioned by date, allowing queries to only scan logs from the specified date range.

3.1.2 Clustering:

Clustering organizes data within a table based on the values of one or more columns. It can improve the performance of queries that filter on clustered columns.

Example: A sales data table clustered by `customer_id` allows queries filtering by customer to be processed more efficiently.

Diagram: A diagram showing a partitioned and clustered table with example queries illustrating a partitioned and clustered table in BigQuery. The table is partitioned by 'Date' and further clustered by 'Customer ID'. The diagram shows how a query filtering by a specific date and customer ID scans only the relevant partition and cluster, optimizing query performance.



3.2.3 Indexing:

Creating indexes on frequently queried columns to speed up search operations. While BigQuery doesn't use traditional indexes like relational databases, using primary keys and well selected columns for filtering and joining can effectively serve a similar purpose. Proper use of indexes (like partition keys) can help avoid full table scans, drastically improving query performance.

3.2.4 Partitioning and Bucketing:

Dividing data into smaller, manageable segments to reduce the amount of data scanned during queries.

3.2.5 Query Execution Plans:

Analysing and optimizing the steps taken by the query engine to retrieve results.

3.2.6 Query Caching:

BigQuery caches query results for 24 hours, so if the same query is run within that time, the cached results are returned quickly and at no additional cost.

Example: A weekly report that runs the same query on a static dataset can take advantage of query caching to avoid reprocessing the data.

3.2.7 Materialized Views:

Materialized views store the results of a query physically, making subsequent queries faster as they don't need to recompute the data. This is especially useful for frequently run, complex queries.

Example: A materialized view on a table of aggregated sales data allows for fast access to precomputed totals, reducing the load on the database.

3.3 Writing Efficient Queries

Crafting efficient SQL queries is crucial for performance and cost management in BigQuery. Here are some best practices:

3.3.1 SELECT Statements Optimization

Select Only the Columns You Need: Avoid using `SELECT *`, which scans all columns, even if they are not needed. Instead, specify only the columns required for the query.

Example: Instead of `SELECT * FROM orders`, use `SELECT order_id, customer_id FROM orders` if only those columns are needed.

3.3.2 Use of JOINs vs. Subqueries

JOINs: While joining tables is common, it can be resource intensive. Use `JOIN` only when necessary and ensure that the join keys are indexed or part of a clustered/partitioned column.

Example: Joining a large `orders` table with a small `customers` table should be done carefully, ensuring that the join key (e.g., `customer_id`) is indexed.

Subqueries: Subqueries can be optimized by limiting their scope and ensuring that they don't process more data than necessary.

Example: Instead of running a subquery on the entire dataset, use filtering conditions to limit the number of rows processed.

3.3.3 Aggregation Functions

Aggregation functions like `SUM`, `COUNT`, and `AVG` can be optimized by ensuring they are applied only to the necessary data subset.

Example: Applying `SUM(sales_amount)` only to rows where `status = 'completed'` rather than to the entire sales table.

4. Big Query Platforms

Google BigQuery: A serverless, highly scalable data warehouse that offers automatic optimization and parallel processing for large queries.

Apache Spark: An open source, distributed computing system with advanced in memory processing capabilities that enhance query performance.

4.1 Cost Management and Resource Utilization

Cost Estimation and Budgeting: Techniques to estimate and control query costs in cloud based environments.

Resource Allocation: Strategies for efficient resource utilization, including workload management and prioritization.

Reducing Data Scanned: Minimize the data scanned by filtering data early in the query and using partitioned tables to ensure only relevant partitions are scanned.

5. Query Cost Estimation

BigQuery provides a cost estimation feature that shows the amount of data a query will scan before running it. Reviewing this estimate can help identify costly queries and optimize them before execution.

5.1 Using Pricing Models (OnDemand vs. FlatRate)

BigQuery offers on demand pricing (pay per query) and flat rate pricing (fixed monthly cost). Choosing the right model depends on the query workload. High volume users may benefit from flat rate pricing, while occasional users may prefer on demand pricing.

6. Common Pitfalls and Best Practices

Avoiding common mistakes like unnecessary data scanning, overly complex queries, and ignoring query cache can lead to better performance and cost savings.

7. Challenges and Future Directions

Despite advancements, several challenges remain in data compression and querying:

Scalability: Ensuring that compression and query techniques scale effectively with increasing data volumes.

Data Integrity: Maintaining accuracy and completeness in lossy compression scenarios.

Real Time Processing: Improving the speed of data compression and query execution in real time analytics.

Future research could focus on integrating machine learning algorithms to dynamically adapt compression and querying strategies based on data characteristics and workload patterns.

8. Conclusion

Data compression and efficient querying are integral to effective data engineering, particularly in the context of Big Data. By leveraging advanced techniques and technologies, organizations can significantly enhance their data handling capabilities. Ongoing research and development in these areas promise to further improve the efficiency and effectiveness of data engineering practices.

9. References

Here's a list of references that would be relevant for the topics of data compression, efficient querying, and Big Data processing on industry-standard resources and foundational research papers.

[1] Salomon, D., & Motta, G. (2010). Handbook of Data Compression. Springer Science & Business Media. - Comprehensive guide on both lossless and lossy compression techniques.

[2] Ziv, J., & Lempel, A. (1977). A Universal Algorithm for Sequential Data Compression. IEEE Transactions on Information Theory. - Foundational paper on the Lempel-Ziv algorithm, which underlies LZW.

[3] Huffman, D. A. (1952). A Method for the Construction of Minimum-Redundancy Codes. Proceedings of the IRE. - Original research on Huffman coding for efficient, lossless data compression.

[4] Wallace, G. K. (1991). The JPEG Still Picture Compression Standard. IEEE Transactions on Consumer Electronics. - Describes JPEG compression, a widely used lossy method in image storage.

[5] Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM, 51(1), 107-113. - Foundational work on the MapReduce model, which supports distributed data processing and compression.

[6] Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). The Hadoop Distributed File System. In 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST). - Paper on HDFS, detailing its role in distributed storage and compression strategies.

[7] Lakshman, A., & Malik, P. (2010). Cassandra: A Decentralized Structured Storage System. ACM SIGOPS Operating Systems Review, 44(2), 35-40. - Insights into partitioning and clustering techniques for efficient querying.

[8] Chambers, C., Raniwala, A., Adya, A., et al. (2010). FlumeJava: Easy, Efficient Data-Parallel Pipelines. In ACM SIGPLAN Notices. - Discusses optimization techniques for large-scale parallel data processing.

[9] BigQuery Documentation. (n.d.). Query Optimization. Retrieved from [Google BigQuery Documentation](https://cloud.google.com/bigquery/docs/best-practices-performance-overview) - Official Google BigQuery resources for partitioning, clustering, and other optimization techniques.

[10] Melnik, S., Gubarev, A., Long, J., Romer, G., Shivakumar, S., Tolton, M., & Vassilakis, T. (2010). Dremel: Interactive Analysis of Web-Scale Datasets. Proceedings of the VLDB Endowment, 3(1-2), 330-339. - Details on Dremel, the basis of BigQuery, and query optimization techniques for large datasets.

[11] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster Computing with Working Sets. HotCloud, 10(10-10), 95. - Overview of Apache Spark's in-memory processing capabilities for efficient Big Data querying.

[12] BigQuery Pricing Documentation. (n.d.). BigQuery Pricing. Retrieved from [Google BigQuery Pricing](https://cloud.google.com/bigquery/pricing) - Details on BigQuery's on-demand and flat-rate pricing, relevant for cost management in query-intensive applications.

[13] Amazon Web Services (AWS) Pricing Documentation. (n.d.). Amazon Redshift Pricing. Retrieved from [AWS Pricing](https://aws.amazon.com/redshift/pricing/) - Information on Redshift's pricing for comparing Big Data warehouse costs.

[14] Fan, W., & Bifet, A. (2013). Mining Big Data: Current Status, and Forecast to the Future. ACM SIGKDD Explorations Newsletter, 14(2), 1-5. - Discussion on challenges in scaling Big Data querying and compression as data volume grows.

[15] Liu, Z., Zhang, W., & Chen, Z. (2018). Machine Learning-Based Big Data Query Optimization. Journal of Big Data, 5(1), 1-18. - A look at how machine learning may optimize querying and compression techniques dynamically.