International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 11 Issue: 11 | Nov 2024 www.irjet.net p-ISSN: 2395-0072

Optimizing Query Performance through Partitioning in Presto

Ajay Krishnan Prabhakaran

Data Engineer, Meta Inc

Abstract - *As data grows exponentially, optimizing query performance in distributed SQL engines like Presto becomes increasingly crucial. Partitioning, bucketing, and sorting strategies allow for efficient data querying and management. This paper explores these techniques in the context of Presto, providing insights into Presto's integration with Apache Spark and Hadoop Distributed File System (HDFS). Through a detailed examination of partitioning, bucketing, sorting, and other optimization techniques, the study highlights best practices to improve query performance. A series of diagrams and real-world case studies illustrate how organizations can enhance their data warehousing capabilities.*

Key Words: **Presto, partitioning, query optimization, HDFS, Spark, bucketing, distributed SQL, data warehousing**

1.INTRODUCTION

Distributed SQL engines like Presto have become essential for large-scale data querying in modern data ecosystems. Developed by Facebook and widely adopted by companies like Netflix and Airbnb, Presto enables real-time queries across distributed data sources without requiring data movement . One of the key techniques that Presto employs to optimize query performance is partitioning.

Partitioning divides large datasets into smaller, manageable pieces based on specific column criteria, such as dates or geographical regions, which Presto uses to reduce the scope of data scanned during a query. Additional techniques such as bucketing and sorting work alongside partitioning to further enhance query performance by improving data locality and reducing the complexity of data scans.

2. HADOOP DISTRIBUTED FILE SYSTEME (HDFS)

2.1 Overview of HDFS

The Hadoop Distributed File System (HDFS) is a scalable, fault-tolerant, distributed file system designed for storing large datasets across clusters of commodity hardware. HDFS is a core component of the Apache Hadoop ecosystem and is the backbone of many large-scale data processing applications, including Presto and Apache Spark.

HDFS breaks files into smaller blocks (typically 64MB or 128MB) and distributes these blocks across a cluster of nodes. This block-level distribution allows for parallel data processing, which is essential for handling large datasets efficiently. HDFS's architecture is built on two main components:

- **NameNode**: Manages the metadata of the file system (e.g., directory structure, file ownership, and block locations).
- **DataNodes**: Store the actual data blocks and perform read/write operations as directed by the NameNode

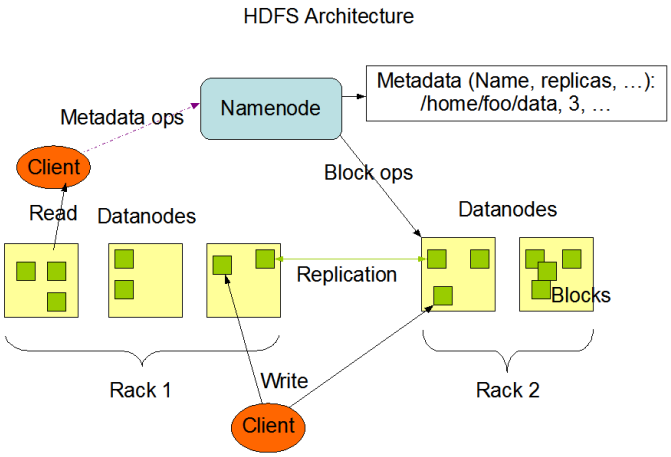

Fig -1: HDFS Architecture Overview

2.2 Presto Integration with HDFS

Presto natively supports querying data stored in HDFS, allowing users to execute SQL queries across massive datasets without needing to move the data. Presto achieves this by abstracting the data storage layer, enabling it to read from HDFS while distributing the query processing across multiple nodes.

Presto's ability to work with HDFS enables it to benefit from HDFS's fault-tolerant and distributed storage architecture. When Presto queries a dataset stored in HDFS, it accesses the relevant data blocks from DataNodes based on the query's requirements, leveraging partitioning and bucketing techniques to minimize the amount of data read.

3. PRESTO AND APACHE SPARK INTEGRATION

3.1 Overview of Apache Spark

Apache Spark is a distributed data processing framework known for its speed, ease of use, and support for

a wide variety of data processing tasks, such as batch processing, stream processing, and machine learning. Spark operates on the concept of Resilient Distributed Datasets (RDDs), which are fault-tolerant and distributed across a cluster for parallel processing.

While Presto is designed for SQL-based querying, Apache Spark is used for more complex data transformation and analysis tasks. Together, Presto and Spark offer a comprehensive solution for data processing and querying.

3.2 Presto and Spark: Complementary Capabilities

Presto and Spark can be integrated to combine Presto's querying power with Spark's processing capabilities. For example, Spark can be used to perform ETL (Extract, Transform, Load) operations and complex transformations, while Presto can be used to run SQL queries on the processed data.

Spark is particularly useful for preprocessing raw data stored in HDFS. Once the data is preprocessed and partitioned, Presto can query it efficiently using SQL commands. This integration is beneficial for organizations that need both real-time analytics and complex data transformations.

4. PARTITIONING IN PRESTO

4.1 What is Partitioning?

Partitioning is the process of dividing a large dataset into smaller, more manageable pieces based on the values of one or more columns. In Presto, partitioning is often done on columns like date, region, or product_category. By splitting data into partitions, Presto can reduce the amount of data scanned during queries, improving performance. Example

```
CREATE TABLE sales (
   order_id BIGINT,
   customer_id BIGINT,
   total_amount DOUBLE,
   order_date DOUBLE
)
WITH (
   partitioned_by = ARRAY['order_date'],
);
```

This SQL statement creates a table partitioned by the order_date column, allowing Presto to optimize queries that filter by date by scanning only the relevant partitions.

4.2 How Partitioning Works in Presto

When a table is partitioned, each partition is stored as a separate directory in HDFS or another distributed storage system. Presto's query engine is able to scan only the directories (partitions) relevant to the query, thereby reducing I/O and improving performance. This becomes extremely valuable especially when dealing with large tables with billions of rows as scanning through all the records can become very expensive

Partitioning also allows Presto to take advantage of partition pruning, a technique where the query engine skips irrelevant partitions based on the WHERE clause in the SQL query. For example, if a query requests data from January 2024, Presto can skip all partitions that do not contain data for January 2024. Since just by reading the metadata Presto can figure out which files/partitions to read, its highly efficient and quick to fetch the results as it avoids wasteful searching for data that's of no relevance

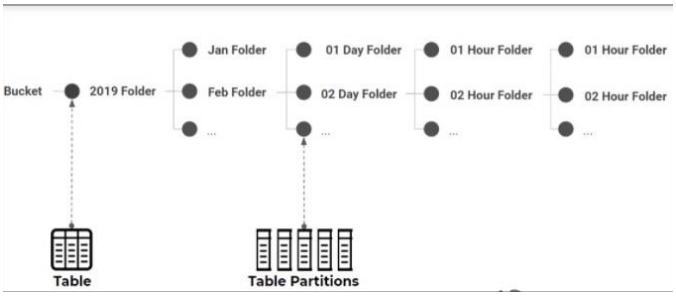

Fig -2: Partition in Presto

5. BUCKETING AND SORTING IN PRESTO

5.1 Bucketing

Bucketing is a technique that groups data within partitions into "buckets" based on the value of one or more columns. This allows for more efficient data retrieval during queries, especially when the query involves JOIN operations or aggregations. In a table with both partitioning and bucketing, data is first divided into partitions and then each partition is divided into buckets. Example

```
CREATE TABLE sales (
   order_id BIGINT,
   customer_id BIGINT,
   total_amount DOUBLE,
   order_date DOUBLE

)
WITH (
   partitioned_by = ARRAY['order_date'],
   bucketed_by = ARRAY['customer_id'],
   bucket_count = 10
);
```

In this example, the table is partitioned by order_date and bucketed by customer_id into 10 buckets. This enables

Presto to efficiently join tables or perform aggregations based on customer_id by scanning only the relevant buckets.

5.2 Sorting

Sorting is a powerful optimization technique in Presto that can significantly enhance query performance, particularly for range queries or queries with an ORDER BY clause. By storing sorted data within each partition or bucket based on a specific column, Presto can avoid additional sorting steps during query execution, thereby reducing the time required to complete queries.

When data is sorted within each partition or bucket, Presto can take advantage of this pre-sorted data to quickly retrieve relevant information. This is especially beneficial for range queries, where the goal is to find data within a specific range of values. With sorted data, Presto can simply scan the relevant partitions or buckets and return the results without needing to perform additional sorting operations. Similarly, when using an ORDER BY clause, Presto can leverage the pre-sorted data to efficiently sort the results. Instead of having to sort the entire dataset, Presto can focus on sorting only the relevant data within each partition or bucket. This reduces the overall processing time and makes the query execution more efficient.

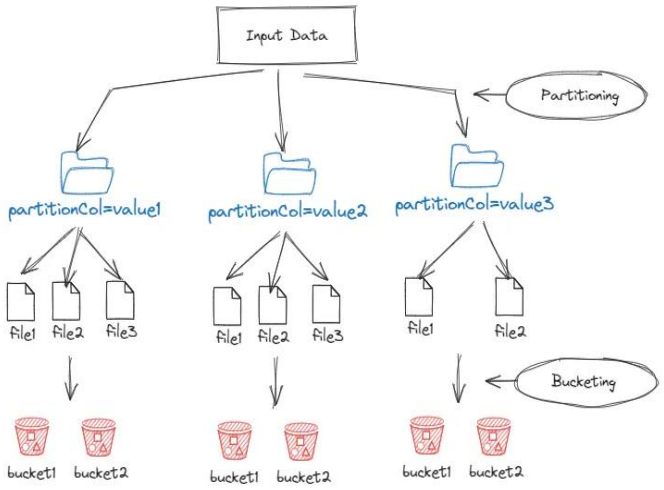

Fig -3: Bucketing in Presto

6. CHALLENGES OF PARTITIONING AND BUCKETING

6.1 Over-Partitioning and Metadata Overhead

One of the challenges of partitioning is over-partitioning, which occurs when the number of partitions grows too large. Too many small partitions can lead to significant metadata overhead, which can degrade query performance instead of improving it

6.2 Partition Skew

Partition skew occurs when data is unevenly distributed across partitions. For example, if most of the data falls into a few partitions, while others remain relatively empty, the benefits of parallelism are reduced. Skewed partitions cause some nodes in the distributed cluster to handle much more data than others, leading to an imbalanced workload and longer query times.

To mitigate partition skew, one option is to choose columns that ensure a more uniform distribution of data when partitioning. Another technique is to implement dynamic partitioning, where the partitions are generated based on incoming data patterns. This approach allows the system to adjust partitions as new data is ingested, improving distribution across partitions.

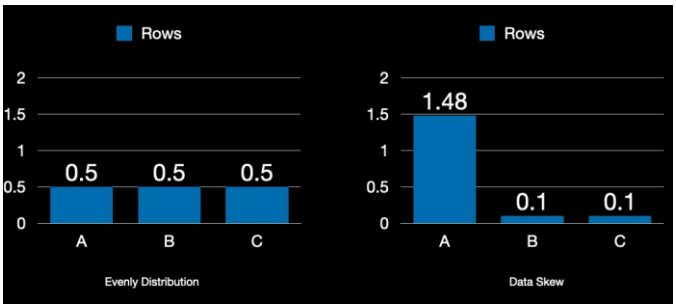

Fig -4: Partition skew

7. OPTIMIZATION STRATEGIES FOR PRESTO PARTITIONING

7.1 Partition Pruning

Partition pruning is a core optimization technique in Presto. It enables the query engine to scan only the necessary partitions based on filters in the WHERE clause. Presto's query optimizer analyzes the partitioning column and skips irrelevant partitions.

For example, consider a table partitioned by order_date. If a query filters for a specific date range, Presto can avoid scanning partitions outside that date range. This dramatically reduces the amount of data scanned, improving both performance and resource usage.

7.2 Combining Partitioning with Bucketing

When partitioning is combined with bucketing, query performance can improve even further. Partitioning limits the number of partitions that need to be scanned, while bucketing allows for efficient JOIN operations within each partition. For instance, in the case of a table partitioned by date and bucketed by customer_id, Presto can perform a join

on the customer_id column by scanning only the relevant buckets within each partition, reducing query execution time.

7.3 Repartitioning for Data Evolution

As data evolves, previously efficient partitioning schemes may become less effective due to changes in data volume or distribution. Repartitioning involves redefining partition boundaries based on updated data patterns, ensuring continued performance improvements over time. This periodic repartitioning, often combined with sorting within partitions, can help avoid issues like partition skew or over-partitioning.

8. CASE STUDIES: PRESTO PARTITIONING IN REAL-WORLD APPLICATIONS

8.1 Case Study 1: Netflix

Netflix leverages Presto to query petabyte-scale datasets stored in HDFS and Amazon S3. Initially, many of Netflix's datasets were not partitioned, which led to slow query performance as large amounts of unnecessary data were scanned.

By implementing partitioning, particularly based on the date and region columns, Netflix significantly improved query times. For example, when querying video stream logs, Netflix partitioned the data by the stream_date column. This allowed Presto to scan only the partitions related to specific dates, reducing query times by over 60%. In addition, bucketing by user_id helped improve JOIN performance in analytics workflows where users' activity data was analyzed.

8.2 Case Study 2: Airbnb

Airbnb uses Presto to run interactive analytics on large datasets stored in HDFS. One of the primary challenges faced by Airbnb was ensuring low-latency queries on high-traffic datasets. Through a combination of partitioning and bucketing, Airbnb was able to improve query performance significantly.

Airbnb partitioned its customer activity logs by the date column and bucketed the data by customer_id. This enabled Presto to efficiently retrieve data for specific date ranges, while JOIN operations (such as those comparing customers' activity across multiple tables) became faster due to bucketing.

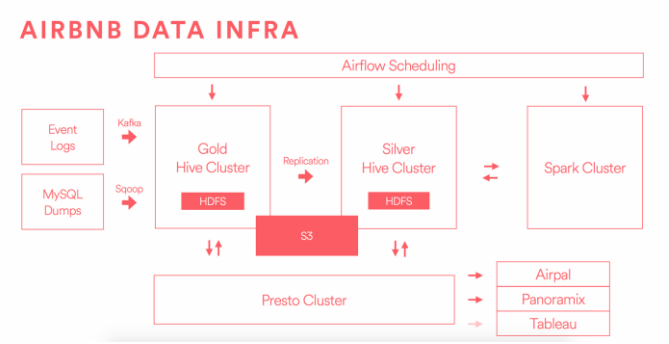

Fig -5: AirBnB Data Model

9. COMPARING PARTITIONING, BUCKETING, AND SORTING IN PRESTO

Partitioning, bucketing, and sorting are all powerful techniques for optimizing query performance in Presto, but they serve different purposes and are suited to different use cases.

- Partitioning: Best for filtering large datasets based on a specific column, reducing the number of rows scanned during queries. It is highly effective for range queries or queries involving a specific subset of the data.
- Bucketing: Improves query performance by grouping rows with similar values into buckets. This is particularly useful for JOIN operations, as it allows for faster data retrieval when joining tables on a bucketed column.
- Sorting: Useful for range queries and ORDER BY operations, sorting within partitions or buckets reduces the need for additional sorting during query execution.

By combining these techniques, organizations can tailor their data structures to specific query patterns, ensuring that query performance remains high even as data grows.

10. PRESTO PARTITIONING BEST PRACTICES

10.1 Selecting the Right Partitioning Column

Choosing the right column for partitioning is crucial. The column should be one that is frequently used in queries' WHERE clauses. For example, partitioning by date is effective in datasets where queries often filter by time ranges. In some cases, partitioning by geographic region (e.g., country) is also beneficial, depending on the nature of the data.

10.2 Managing Partition Overgrowth

To avoid the issues associated with over-partitioning, such as excessive metadata and slow query performance, it is important to periodically monitor and tune partitions. If too many small partitions are created, it may be

necessary to consolidate them by adjusting the partitioning scheme.

10.3 Balancing Bucketing and Partitioning

While partitioning allows for efficient scanning of large datasets, bucketing adds another layer of optimization for queries involving JOINs or groupings. However, using both techniques together should be done with careful consideration. Overuse of bucketing can lead to excessive data fragmentation, so it's important to strike a balance based on the specific query patterns and dataset characteristics.

11. FUTURE TRENDS IN PRESTO PARTITIONING AND QUERY OPTIMIZATION

As Presto continues to evolve, new techniques and optimizations will be introduced to handle ever-growing datasets and more complex query patterns. One area of focus is adaptive partitioning, where Presto dynamically adjusts partition boundaries based on the incoming data and query patterns, further reducing query times without manual intervention.

Additionally, machine learning is being increasingly integrated into query optimization strategies, allowing for intelligent decisions regarding partitioning and bucketing schemes based on data access patterns. This shift towards automated data optimization will make it easier for organizations to maintain high query performance without constant manual tuning.

12. CONCLUSION

Partitioning, bucketing, and sorting are key optimization techniques that improve query performance in Presto, particularly when working with large datasets stored in distributed systems like HDFS and S3. By effectively leveraging these techniques, organizations can significantly reduce query times, minimize resource consumption, and improve overall data management.

However, these techniques require careful management to avoid pitfalls such as over-partitioning, skew, and excessive metadata. The combination of partitioning with advanced techniques like bucketing and sorting, alongside regular monitoring and optimization, ensures that Presto continues to deliver high-performance querying at scale.

As data volumes continue to grow, future developments in Presto, including adaptive partitioning and AI-driven query optimization, will further enhance its capabilities, making it an even more powerful tool for data analytics in the distributed computing ecosystem.

REFERENCES

[1] M. Han et al., "Presto: SQL on Everything," Proceedings of the VLDB Endowment, vol. 6, no. 12, pp. 1972–1975, 2013.

[2] B. Travers, "Scaling Data with Presto at Netflix," Netflix Tech Blog, 2021

[3] J. Borthakur et al., "The Hadoop Distributed File System," Proceedings of the IEEE, vol. 107, no. 9, pp. 1651–1670, 2019.

[4] G. Sakr, "Improving Data Warehouse Performance Using Partitioning Techniques," Journal of Big Data, vol. 9, no. 1, 2022.

[5] A. Colbert, "Partitioning Best Practices in Distributed SQL Engines," Dremio Blog, 2023.

[6] K. Kumar, "Presto Query Optimization," AWS Blog, 2022.

[7] R. Smith, "Managing Big Data with Presto Partitioning," Databricks Blog, 2021.

[8] A. Patel, "Challenges of Over-Partitioning," Snowflake Documentation, 2023.

[9] L. Zhang et al., "Handling Data Skew in Partitioning," Journal of Distributed Systems, 2020.

[10] S. Wiggins, "Partition Pruning Techniques," Data Management Journal, 2022.