Onehouse Clustering Strategy Best Practices

Last Update: Nov 21, 2024

By Po Hong

Introduction

Onehouse offers customers a managed Universal Data Lakehouse (UDL) platform that enables rapid data ingestion and transformation in minutes, all at a reduced cost. Your data is stored in an auto-optimized, open-source data lakehouse, supporting a wide range of analytics and AI use cases, and is compatible with any query engine you prefer.

During the ingestion process, Onehouse leverages Apache Hudi to write the data, benefiting from Hudi's maturity, built-in indexes, and proven track record in providing low data latency for both batch and streaming workloads.

Onehouse offers customers fully automated small file handling and provides several sorting and ordering strategies, including linear ordering, Z-Ordering and others.

In this blog, we'll explore clustering strategies such as linear ordering and Z-Ordering. We'll explain the best use cases for each, discuss their deployment options, and share best practices to help you optimize your Onehouse experience.

What is Clustering in Apache Hudi?

In Apache Hudi, clustering typically involves two key tasks:

- **Managing small files**: During ingestion, small files are merged into optimally-sized files (~100MB) to prevent performance issues for query engines.
- **Re-organizing data files**: Sorting or organizing data files based on specific columns to improve query performance for certain patterns.

Small file handling works out of box with Onehouse. For more detailed information about how it works, see: https://hudi.apache.org/blog/2021/03/01/hudi-file-sizing/.

Clustering, in the context of data storage, stands as a valuable optimization technique to improve the storage layout by preserving data locality for better read efficiency.

Hudi offers three layout optimization strategies, namely linear, Z-Order, and Hilbert curves. Each of these strategies defines how records should be sorted during ingestion. The default strategy is linear, which performs lexicographical sorting. The other two, Z-Order and Hilbert curves, are known as space-filling curves; both sort entries and preserve good spatial locality, and they are similar in nature.

Purpose of Data Clustering

Optimizing Data Layouts

As you ingest data and build your data platform, it's crucial to efficiently organize the data layout for optimal storage and query performance. Without proper organization, the performance of your pipelines and queries can degrade

over time. Key data layout optimizations include partitioning, indexing, clustering, file-sizing, and cleaning. While these techniques are commonly supported in databases or data warehouses, they are often lacking in cloud object storage systems such as Amazon S3, Azure Data Lake Storage, and Google Cloud Storage (GCS).

In analytical systems, the write-to-read ratio is typically 1:100 or higher - that is, for every update, there are 100 or more queries. The reads serve various purposes such as real-time dashboards, weekly reports, and downstream analytics or AI/ML applications. Additionally, the data layout during ingestion is often different from the layout needed for efficient querying. For instance, it's easier to ingest customer orders by OrderDate, but querying is typically more efficient when the data is sorted by CustomerId.

Clustering in Hudi

Hudi's clustering framework offers a flexible strategy to reorganize data layouts and optimize file sizes. This enables you to enhance query performance without compromising data ingestion throughput. Clustering improves query efficiency by adjusting the data layout on disk, through sorting based on user-specified columns. This approach leverages the Parquet file format's ability to perform predicate push-down and skip irrelevant files and Parquet row groups, further improving performance and managing file sizes to avoid small file issues.

Clustering Strategies in Apache Hudi

In this blog, we will focus on two of the most common clustering strategies in Hudi: linear ordering and Z-Order. Hilbert curves are similar to Z-Order; in some cases, such as a large number of clustering columns, they may have better performance than Z-Order. For more details about the comparison between Z-Order and Hilbert curves, see Apache Hudi Z-Order and Hilbert Space Filling Curves.

Linear Order

Using this clustering or sorting strategy, the data files in each partition of a Hudi table - assuming it is a partitioned table - will be sorted by one or more columns, and the order of these columns plays a critical role.

As an example, let's assume that we use three columns (A, B, C) as the clustering/sort key. Then ONLY the following predicates (or group by) in a SQL query will benefit from this sort order: (A), (A, B), and (A, B, C). But queries with predicates on columns (B), (B,C) or (C) won't benefit from the sort key. A good mental model is to think of linear ordering/sort keys in Hudi as B-tree indexes in relational databases.

Z-Order

Z-Ordering is a data layout optimization technique that enhances query performance by sorting data across multiple columns. It maps multidimensional data into a one-dimensional space while preserving data locality, meaning that data points close in the original, multidimensional space remain close in the one-dimensional space. This allows for more efficient data retrieval and improves query performance by enabling more effective file skipping, complementing other data skipping techniques, such as using column statistics in Hudi.

Z-ordering is especially valuable when dealing with multiple columns. If you only need to sort by a single column, linear ordering is sufficient. However, if your query pattern involves multiple columns like (A, B, C), and these columns are frequently queried either individually or in pairs, applying Z-Ordering to (A, B, C) would generally provide better performance than a simple linear sort key.

As an example, let's perform a simple test on two Hudi tables which have the same data: table_1 and table_2. Table_1 is using a linear sort order on columns (A,B,C), while table_2 uses a Z-Order on the same columns (A,B,C).

Now consider the following three queries:

Q1: Select A, count(*) From table Group By A;

Q2: Select B, count(*) From table Group By B;

Q3: Select C, count(*) From table Group By C;

The query runtimes for the two different clustering/sorting strategies are summarized in Figure 1.

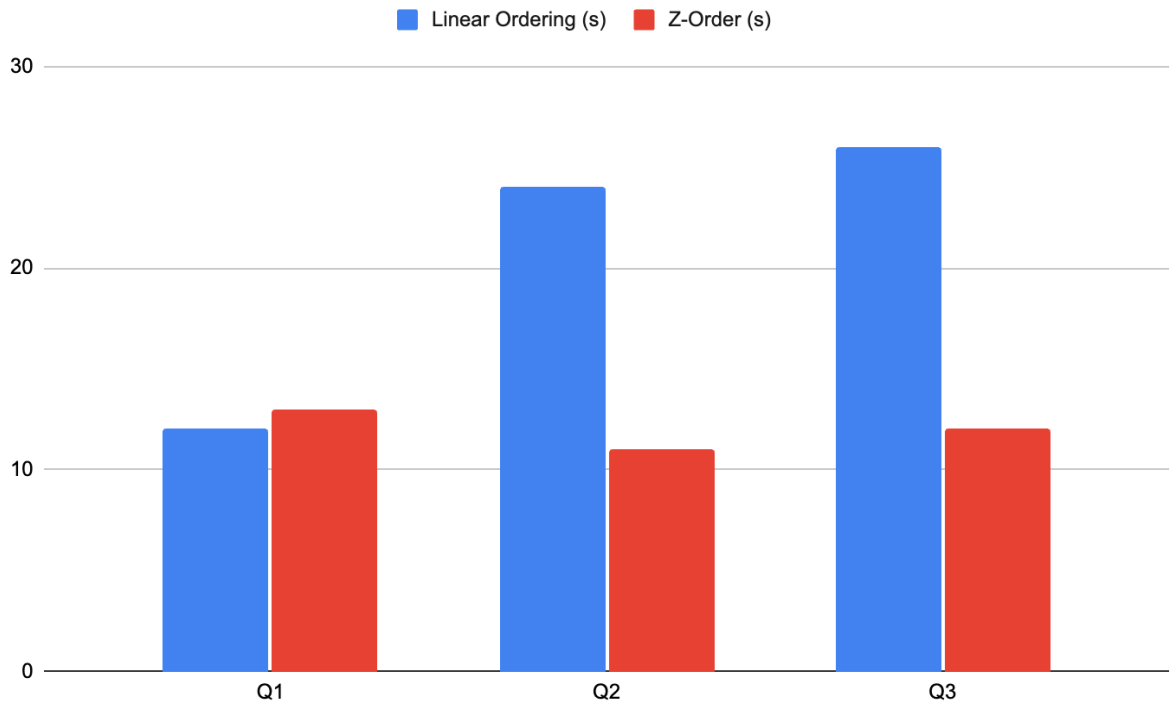

Figure 1: Query performance for linear ordering and Z-order (lower is better)

When a query uses column A, the leading column in the linear order key on Table_1, it performs as well as or even slightly better than on Table_2, which uses Z-order. However, when columns B and C are queried individually, linear ordering loses its effectiveness compared to Z-order.

Choosing the Right Clustering Strategy

Following are guidelines and best practices for choosing the best clustering strategy:

- Identify Clustering Requirements: Analyze data characteristics, usage patterns, and query workloads and compare them to your clustering options.

- Choose Appropriate Clustering Columns: Select columns that effectively represent data patterns and are relevant to query filtering and data access needs. Clustering columns should have a relatively high cardinality.

- Choose a Clustering Strategy:
    - Choose linear ordering if your query patterns are dominated by certain frequently-used predicates, and arrange them accordingly. In general, you should limit a linear ordering key to no more than four columns. Additionally, ensure that the ordering key as a whole has a sufficiently high cardinality to minimize data skew during Spark shuffling.
    - Choose Z-Ordering if the clustering columns are more or less equally frequent in the predicates.

- Utilize Onehouse's Incremental Clustering Feature: Leverage Onehouse's built-in incremental clustering capabilities to simplify the clustering process and integrate it with data ingestion and management workflows.

- Onehouse's Table Optimizer Feature: For Hudi tables created and managed outside of Onehouse, you can use Onehouse's Table Optimizer to run table

services (e.g. clustering) on these external Hudi tablesTable Optimizer: The Optimal Way to Execute Table Services in an async mode without impacting the writer's performance, helping to meet strict performance SLAs and reduce operational burden.

Conclusion

To optimize data layouts as you ingest data and build your data platform, it's important to consider how to efficiently organize the data on storage for both performance and cost savings.

Onehouse addresses this need by bringing essential database-like storage layout optimizations to a managed data lakehouse. In this blog, we have discussed the clustering strategies available in Hudi, such as linear ordering and Z-Order, explored their typical use cases, and provided guidance on selecting the optimal clustering strategy. If you already have a data lakehouse based on Apache Hudi and want to streamline management and automate Hudi table services such as clustering, compaction, and cleaning, you might consider Onehouse's Table Optimizer product.

If you are interested in learning more, please reach out to gtm@onehouse.ai or sign up for a free trial with $1000 in credits.