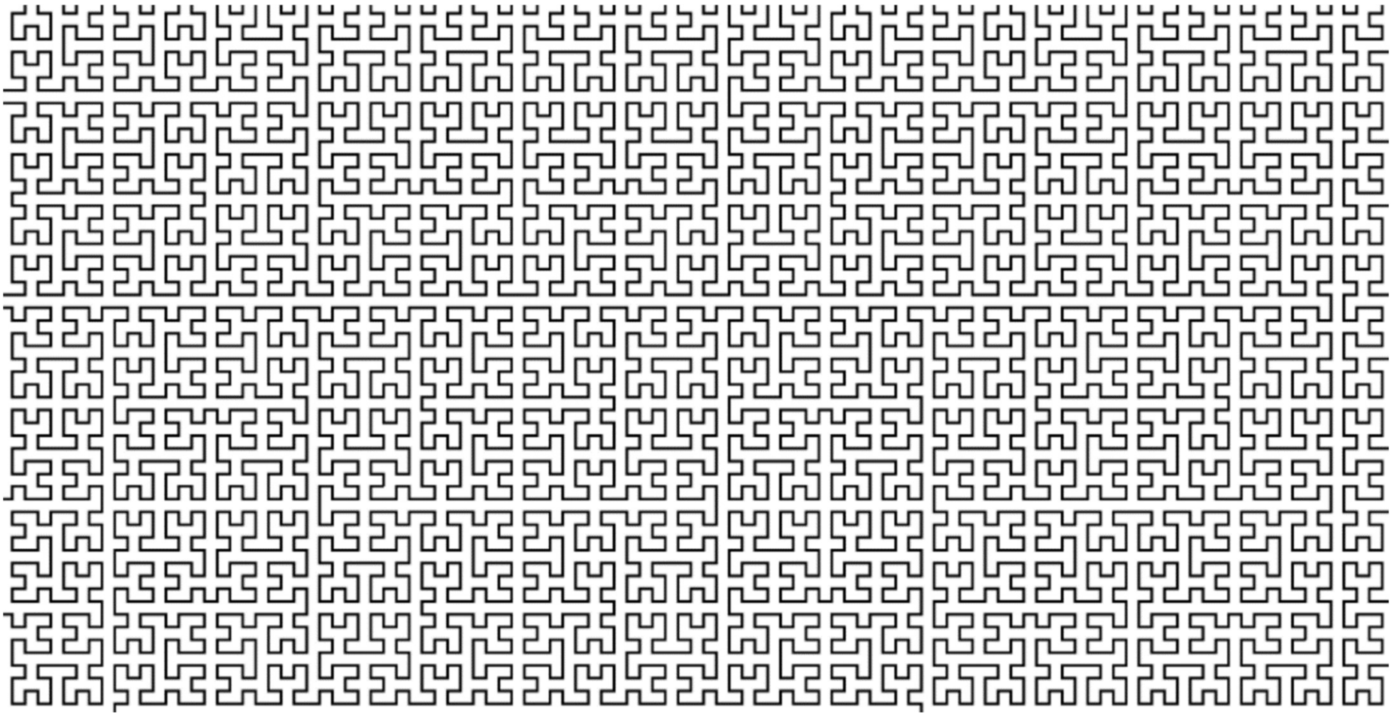
TV 2 Danmark

Comparative Analysis of Data Organization Techniques in Databricks

by Andreas Esmann Kristensen

September 2024



Abstract

The goal of this project is to optimize one of TV 2 Danmarks larger delta datasets containing **TV 2 Play tracking data** for faster reading capabilities.

The current dataset has multiple **critical performance flaws** which needs to be addressed as the **response times** for back-end transformation flows are challenged as well as the end users experience when querying on the dataset.

There are **four data organization techniques** available in Spark which are Hive Partitioning, Z-Ordering, Liquid Clustering and Bloom Filters. To test which data organization technique performs the best, the original dataset has been duplicated to **multiple variations** of each of the four techniques and then tested by three selected **performance perspectives**.

These performance perspectives are categorized by how well a dataset can be **pruned for irrelevant files**, how fast **individuals can be identified** and how fast larger **subsets of data can be aggregated** upon.

The test results are indicating that the newly developed **Liquid Clustering** techniques has the best performance and even seems superior to the other techniques with only a few exceptions.

Table of content

1. Introduction

The introduction chapter will be divided into the following sections:

- **Motivation**: The motivation for writing this project
- **Problem Statement**: Stating the goal of the project and relevant delimitations
- **Data Introduction**: The Lake House architecture, relevant metadata on the dataset and known dataset issues
- **Test Environment**: How the test setup was configured

1.1 Motivation

TV 2 Danmark launched **TV 2 Sputnik** back in 2004 which was a streaming and content on demand platform where subscribers could watch TV 2's content digitally through a web browser. The TV 2 Sputnik platform was, at the time, only a small product with a very limited set of development resources attached.

After some years the **ad revenue** of the traditional TV market was in a decline as "cable cutters" (cutting cable TV for alternative services e.g. streaming) started to emerge. This was an early warning for TV 2 Danmark to secure additional revenue elsewhere. TV 2 Danmark discovered a great potential for streaming and on demand services as it could generate paid subscriptions and ad revenue. This resulted in the launch of the **TV 2 Play** platform in 2012 as a rebranding of TV 2 Sputnik.

For analytical purposes, TV 2 Danmark purchased license to **Adobe Analytics** which was integrated with TV 2 Play as an analytics tool for accessing historical tracking data. Increasing demand for further analysis and modeling on the **TV 2 Play tracking data** resulted in data exports from Adobe Analytics to Azure storage accounts.

Due to the daily amount of large, compressed files it was necessary to advance the development of BI solutions to use Big Data utilization tools to enable an efficient transformation.

As the Data Platform team was already using Azure services it was a natural decision to use **Azure Storage Accounts** for file storage and **Azure Databricks** for computing.

Besides delivering aggregated data to tabular models, the transformed TV 2 Play tracking datasets have back then met the **expectations** of our stakeholders by simply presenting the data for Data Scientists and Data Analysts in a new controlled environment where Big Data was accessible and manageable.

Today this is not enough as data also needs to be delivered quickly as our stakeholders now have their own flows for **predicting models** or for **advanced analytics** which needs to reach their stakeholders early morning.

Because of these new challenges and demands from our stakeholders it is required that we increase performances for earlier access to the data and faster response times when using the data. This can be achieved by changing how the data is organized and stored through four different data organization techniques.

1.2 Problem Statement

This project will investigate how it is possible to **performance optimize** TV 2 Danmarks TV 2 Play tracking data by using different **data organization techniques** to optimize **reading capabilities** for further usage by Data Engineers, Data Scientists and Data Analysts. The writing process will be in mind but not in focus as it is more important that the recipients of this data have their query times reduced.

The goal is to specify one or multiple data organization techniques to use for optimized data reading capabilities. Trade-offs will be evaluated. Databricks has officially announced **Liquid Clustering** as their recommended technique, but this project will test whether Liquid Clustering is the optimized solution for TV 2 Play's tracking data.

1.2.1 Delimitations

TV 2 Play's transformed tracking data will be the focus of this project as it has some known performance issues and has multiple dependencies. More specifically will this project only contain performance analysis of the Curated Micro Data dataset which will be described in 1.3.2 Curated Micro Data.

PySpark (Python API for Apache Spark) and **SparkSQL** (Apache Spark module for using SQL) will be the only coding languages used in this project. TV 2 Danmarks Data Engineers are mostly using these specific languages, and it does not seem necessary to introduce additional languages as the complexity of our tasks should be able to be achieved from these two languages.

In addition, this project will not be going down on RDD levels for performance optimizations as the SparkSQL module is already optimized for utilizing RDD's (Databricks - Spark SQL, u.d.).

This project will only focus on the **Delta Parquet** file format as it is already the chosen format for our current hybrid Lake House solution. There are other popular options available, like Apache Avro or ORC (Optimized Row Columnar), which may have an impact on the performance of the reading and/or writing abilities.

Apache Avro is favored in relation to streaming services while Delta Parquet and ORC are primarily used for Data Lakes and analytical workload. ORC and Delta Parquet are very similar as they both use columnar storage methods and have **predicate pushdown support**. Delta Parquet has a small advantage over ORC as it has a variety of **compression codecs** which depending on the usage can improve the performance additionally (Vellaturi, 2023). This project will only focus on the default **snappy compression** codec.

1.2.2 Experiments

As mentioned above the dataset used for this project is the Curated Micro Data dataset. Most of the notebooks that depend on the dataset are daily calculating the distinct aggregations of the subscription video activity for the last 30 days. The dataset has been replicated to an isolated storage account with data from a two-month period which consists of 61 days. This has been specifically chosen to avoid some of the data organization techniques being favored for testing all data available.

An important aspect of the Apache Spark and Delta Parquet collaboration is the file skipping capabilities. Based on simple statistics generated by the Delta Parquet format, files can be skipped without being read which has a significant impact on query response times. Skipping capabilities will be an addition to the performance analysis of the data organization techniques.

The back-end Data Engineers use the data for aggregation for tabular models while end-users like Data Scientists or Data Analysts often use it for individual analysis (specific users tracking). The performance analysis of the data organization techniques will be divided into an aggregation experiment and an individual experiment as we need to accommodate both needs.

In total there will be 24 dataset variations that will be evaluated and measured by their skipping efficiency and aggregated- and individual performances. All datasets are derived from the original dataset on the production environment which has been duplicated to a new storage account. The dataset variations are organized and stored by one or multiple of the data organization techniques to test potential performance range of each technique. 13 of the 24 dataset variations are included in the project while the rest are available in the Appendix chapter. All the experiments will be discussed in further details in 3. Method.

Underneath are the three experiments briefly described:

- **Skipping Efficiency**: How well a data organization technique can skip scans and files when querying on a big dataset
- **Individual Performance Analysis**: How well a data organization technique can target individuals in a big dataset
- **Aggregated Performance Analysis**: How well a data organization technique can aggregate a larger subset of a big dataset

Configurations set for each of the 13 included dataset variation are illustrated in Figure 1 below. The figure shows each dataset with color coded dataset attributes which represent the data organization techniques being used. Some datasets are combining storage techniques which are also demonstrated by the color coding. An example is the Bloom Filter dataset where it uses Calendar and User as Z-Order attributes while having Video as the Bloom Filter. More information regarding the attributes used are available in 3. Method.

Hive Partition
Z Order
Bloom Filter
Liquid Clustering

Dataset	Calendar	User	Video	Site	Product
Hive Partition Base	X			X	X
Hive Partition YMD	X				
Hive Partition Video	X		X		
Hive Partition Video 2	X		X		
Z Order	X	X			
Z Order Video 2	X	X	X		
Hive Z Order	X	X			
Hive Z Order Video 3	X	X	X		
Bloom Filter	X	X	X		
Bloom Filter Video 3	X	X	X		
Bloom Filter Video 6	X	X	X		
Liquid Clustering 2	X	X			
Liquid Clustering Video 3	X	X	X		

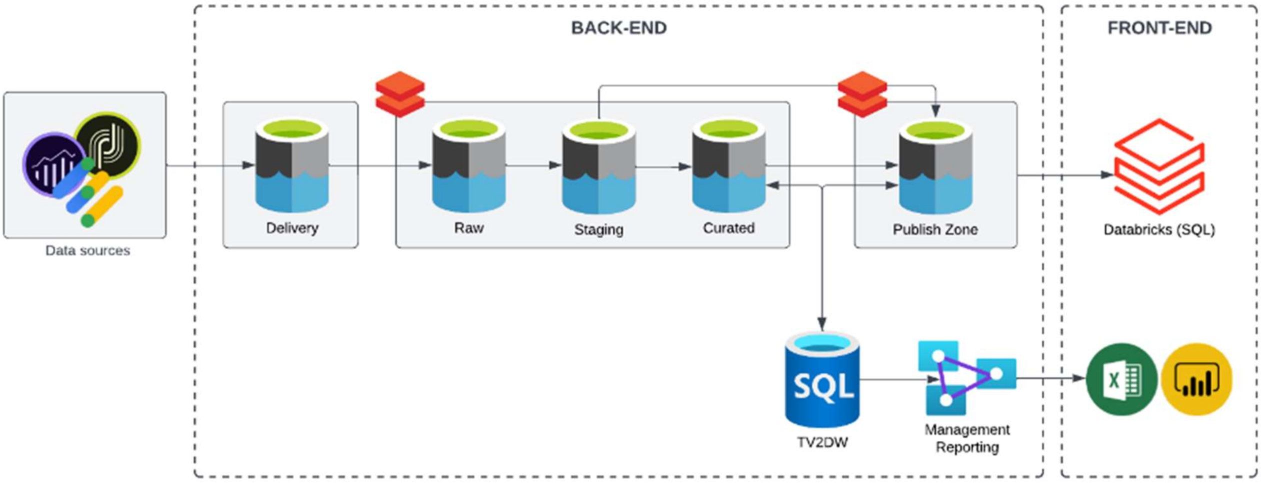
Figure 1: Dataset configurations

1.3 Data Introduction

This chapter will focus on introducing the architecture, the dataset and known issues with the dataset. This will give the reader a better understanding of the cause of the motivation and in what state the current dataset is in.

1.3.1 Data Platform Architecture

TV 2 Danmarks Data Platform Architecture consists of the **Medallion Architecture** with the alternative naming convention Raw, Staging and Curated zones instead of the traditional Bronze, Silver and Gold zones.



Data source systems typically push raw and often zipped data to our Delivery zone where Databricks notebooks will unzip the data, if necessary, and convert it to Delta Parquet files which will be stored in the Raw zone.

The Raw, Staging and Curated zones are only containing Delta Parquet files. From each zone to the next, data will be transformed in some way. The ideal process would be to have raw data in the Raw zone, transformed data in the Staging zone and then have necessary joins in the Curated zone. As this setup was developed back in 2019 the process has not been consistent all the way through.

From the Curated zone, data will get exported to a regular Data Warehouse with **Azure SQL Server** and **Analysis Services**. Additionally, data is also exported from the Curated zone to another zone called Publish Zone. The

Publish Zone is an additional zone made for Data Analysts and Data Scientists to access centralized controlled Staging and Curated data on a secure platform without any risks of personal data. The Publish Zone storage account is linked to a **Databricks Premium Instance** where the Data Analysts can access data through the Databricks SQL Persona for simple analytics purposes while the Data Scientists can access data through the Data Engineering Persona for more advanced analysis with additional code languages available and orchestration possibilities.

All the processes are orchestrated from **Azure Data Factory** pipelines and not through Databricks own Workflows. This is a work in progress.

1.3.2 Curated Micro Data

The data platform has some good datasets available for performance analysis. One of the better choices is the Micro Data dataset from the Curated zone as this dataset has some known flaws with the file storage and the transformation code. These flaws will be described in 1.3.3 Identifying Performance Issues. In addition, there are **four notebooks depending** on the Micro Data dataset which see some challenges due to not pruning files efficiently from querying on the dataset.

All the data from the Micro Data dataset are stored in Delta Parquet format. The benefit of this format is that a **Delta Log** is created when writing the data to a directory. The Delta Log serves as a log for basic statistics of each data file like min and max values of each attribute which makes file pruning more efficient. The statistics are as default only logged on the **first 32 attributes** of the dataset. The Delta Parquet format and file pruning will be described further in 2.1 Data Skipping (Ionescu, 2018).

The Micro Data dataset consists mainly of TV 2 Play tracking data from the Adobe Analytics platform but also of subscription data from TV 2 Danmarks internal software solution called Play Commercial Services (PCS).

The purpose of the dataset is to have a **denormalized table** consisting of all the relevant user tracking events combined with the subscription data to make fast aggregations of the data which will get exported to a SQL server for id transformations and lastly loaded to a tabular model in Analysis Services.

The dataset was developed back in 2019 as one of the first Spark projects together with other datasets utilizing the Adobe Analytics tracking data. Back then it was a fairly new technology, which resulted in a discovering work process.

A subset consisting of a 61-day period has been duplicated from the Micro Data dataset from the production environment for this project. The subset of the dataset consists of:

- **64 attributes** of mainly integers and strings
- **5 Hive Partitions Levels** – Year, Month, Day, SiteKey and ProductId
- 20.384.372 average rows pr day
- **1.243.446.697** rows total
- **4.121 total files** and 68 files on average pr day
- **18,02 Gb data** and 0,3 Gb data on average pr day

Listed below are all the attributes used in this project with their respective description, datatype and an example of a value.

Native attributes used in the dataset:

- **Year**: The year of the event. Integer type. E.g. 2024
- **Month**: The month of the event. Integer type. E.g. 7
- **Day**: The day of the event. Integer type. E.g. 7
- **CalendarKey**: The date of the event as an integer type. E.g. 20240707
- **SiteKey**: The site used at the time of the event. String type. E.g. TV2PLAY, tv2.dk or TV2SportsApp
- **UserKey**: The user of the event which is a hexadecimal. E.g. “1bcbe8c253e74d5dbc413f45cd168312”
- **ProductId**: The subscribed products surrogate key end of day which can be related to a Subscription Product dimension. It is an integer type. E.g. 40 which corresponds to the Play product “Basis uden reklamer”

- **VideoViewsExist**: A flag set to true if a video view event exists which is if a user has started any content on the Play platform. Tv2.dk SiteKey events would often not be true as these events contain clicks on articles and not necessarily video content. It is formatted as an integer type and consists only of 1 or 0
- **VideoPlayCount**: Count of video plays of a day. Integer type. Can be any positive whole number
- **VideoTime**: The time spend on watching content on TV 2 Play. It is an integer type and can only be positive whole numbers

New attributes created for this project:

- **VideoFlag**: Flag being true when at least one of the video attributes (VideoViewsExist, VideoPlayCount or VideoTime) have valid data. Integer type consisting of either 1 or 0
- **VideoInfoCode**: Concatenation of a flag for each of the video attributes (VideoViewsExist, VideoPlayCount or VideoTime) having valid data. E.g. "010" which corresponds to no valid VideoViewsExist, a valid VideoPlayCount and no valid VideoTime

1.3.3 Identifying Performance Issues

Before starting to performance analyze the Curated Micro Data dataset, it is necessary to analyze the current performance issues with the data.

First, we need to identify how the Curated Micro Data dataset files are stored to know how the data can be queried most effectively. Then we need to analyze how the dataset is being queried as the common filter expressions might give away which attributes are more important.

When using the **DESCRIBE** function on the dataset it shows which attributes are within the first 32 and thereby included in the Delta Log statistics for effective file skipping. It also shows how the dataset is stored and by which attributes which can be utilized for further increasing how well files can be skipped.

The Curated Micro Data dataset is stored as a Hive Partitioned dataset and have five levels of partitioning. These five levels consist of a date hierarchy of Year, Month and Day and by SiteKey and ProductId.

The most common filter expressions made on the Curated Micro Data dataset from the back-end transformation of the four depended notebooks are either a singular date or a date range. The second most common filter expressions are having at least one valid or active video attribute. The video attributes are "VideoViewsExist", "VideoPlayCount" or "VideoTime" which define whether there has been an activation (an activity) on a subscription by viewing content on TV 2 Play.

Underneath in Figure 2 is a snippet of a filter expression used in one of the notebooks which query on the Micro Data dataset.

```
WHERE (cal.CalendarDate >= date_sub('$LongDate', 32)
        AND cast(cal.CalendarDate AS DATE) <= '$LongDate')
        AND  (
            VideoViewsExist > 0
            or VideoPlayCount > 0
            or videotime > 0
        )
```

Figure 2: Snippet of a filter expression on the Micro Data dataset

All three video attributes are represented in the filter expressions and none of them are partitioned or stored as a key in the dataset. None of them are included in the Delta Log statistics as they are beyond the 32 first columns in the dataset schema.

In the showcased filter expression in Figure 2, the date range is called through a joined calendar dimensions (cal) date formatted attribute and not directly through the Micro Data dataset to utilize partition pruning. This is presumably due to the partitions being hard to query on, as the Year, Month and Day partitions are integers and not strings which is a very important distinction as the string version can be directly concatenated (CONCAT(Year, Month, Day)) to become an integer calendar key. This is not the case for an integer formatted date hierarchy partitioning as this version needs additional formatting to fit an integer calendar key (Year * 10.000 + Month * 100 + Day).

When using the Calendar dimensions CalendarDate (yyyy-MM-dd) or the Micro Data datasets CalendarKey (yyyyMMdd) you are not enabling partition pruning in your query which results in full scan of all files through all partitions. Data will only be skipped based on min/max values of each file through the Delta Log statistics. If a concatenated string date hierarchy partitioning was used instead, the query would scan only the files in that specific partition and skip files further based on the Delta Log statistics.

In conclusion the main performance issues with querying on the Curated Micro Data dataset can be summarized as:

- The date hierarchy partitioning using integer values instead of strings (e.g. 7 and not "07"), which can't be directly concatenated to an integer date
- The video attributes are not included in the Delta Log statistics due to the attributes being further than the first 32 attributes
- The video attributes are not activated through partitions or stored keys as only a date hierarchy, SiteKey and ProductId partitioning exists
- The SiteKey and ProductId partitioning levels are not being used in any queries on the dataset which is unnecessary and might also issue some small file problems

1.4 Test Environment

This section will describe the configurations needed to set up the test environment on an Azure data platform. This will include the following subjects:

- Storage Account setup
- Authorizations
- Cluster Settings
- Data Configurations

Storage Account setup

The goal was to create a new test environment isolated as much as possible from the production environment. The new Azure Storage Account "tv2dwstorageproject" was therefore created for duplicated production data, new data storage methods and performance test results.

The "tv2dwstorageproject" storage account was upgraded to Data Lake Gen 2 as it is recommended by Azure on storage accounts that focus on Big Data and data analysis. The specific gain from this upgrade is stated by Azure to be "hierarchical namespace and unlock capabilities such as file and directory-level security and faster operations" (Microsoft - Data Lake Storage Gen 2, 2024). The hierarchical namespaces are required for the Hive Partitioning method as without this upgrade only one level of partitions could exist.

Authorizations

An established Service Principal account has been enabled for the service communication between the Azure Data Lake Storage Accounts and Databricks. This makes it possible for Databricks to read and write from relevant Storage Accounts in this project.

Cluster Settings

To execute tasks in Spark we need to configure a cluster. A cluster is a collection of virtual machines which can operate in parallel on the same task which increase the performance for query response times. The cluster is divided into a driver node and n amount of worker nodes. The driver node will distribute tasks and data between the workers for optimized execution times (Databricks - Concepts, 2024).

For this project an exact copy of our production environment cluster in terms of compute power has been created. This is created to have a direct performance comparison to what is already running in production. Furthermore, if some of the data storage changes are considered superior to what is already in production we might deploy parts of this solution to the production environment.

The cluster is a **Standard DS4 v2 cluster** with one driver and a scaling two to 10 workers with each 28 Gb Memory. It is running on the **Databricks Runtime 14.3 LTS** which has Spark 3.5 and Scala 2.12. One of the newer versions of the Databricks Runtime is required as some methods used in the project were first introduced recently. Liquid Clustering in its public preview is available on Databricks Runtime 13.3 or higher.

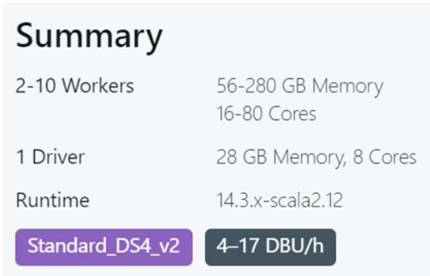


Figure 3: Cluster settings

Data Configurations

Even though the Service Principle has enabled a connection between the Azure Storage Account and Azure Databricks, the Service Principle still must be enabled from Databricks with every new Spark Session.

For enabling the Service Principle from Databricks I have created a Configuration notebook which retrieves the Service Principal information from a Key Vault and insert them into the client settings for our directory.

The Configuration notebook is called from all other notebooks in this project as this step is needed to access the data. Therefore, optimization options have been added to the notebook for centralized optimization through all operations in this project. This is e.g. caching, optimization, and adaptive options. The full notebook is located the appendix chapter.

2. Data Organization Techniques

In this chapter the data organization techniques used in the project will be described along with evaluating combinations of techniques and what Spark do when not using any.

There are currently four data organization techniques available:

- **Hive Partitioning** (partitioning)
- **Z-Ordering** (curve mapping)
- **Bloom Filter** (probability)
- **Liquid Clustering** (curve mapping)

While it is important to know how each data organization technique theoretically work it first needs to be established why it is important to work with data organization techniques.

When no data organization technique is enabled on a Delta dataset the default thing the Spark engine do is using the Delta Log statistics for Data Skipping irrelevant files. This will be elaborated further on in 2.1 Data Skipping. Data Skipping alone can do a lot for the query response times but helping the Data Skipping process with dividing the data into something more organized or frequently used can increase the query performance significantly.

2.1 Data Skipping

Data Skipping is a term used in Databricks for skipping files to improve performance at query times. The more data you can skip before loading data into memory, the faster the query can finish.

Data Skipping was mainly introduced in conjunction with the Delta Parquet file format as this format stored a log of statistics for each file in the directory the data was written to. These statistics include simple statistics like max, min, null counts and total records of each of the first 32 attributes in the dataset. Each nested elements in an attribute counts toward the limit of the first 32 attributes as well (Databricks - Data skipping Z-Order, u.d.).

The Delta Log statistics is a brilliant addition to the Parquet format as it makes data skipping a lot more efficient. Queries which use a filter expression on one of the first 32 attributes in a dataset will gain advantage of data skipping. The Spark Engine will look inside the Delta Log statistics, check the min and max values of each file and skip all files which is not within the accepted min/max threshold (Ionescu, 2018).

2.2 Hive Partitioning

The Hive Partitioning technique is a technique to divide your dataset into smaller datasets in folders in a directory based on **partitioning keys**. A partitioning key is an attribute in the dataset which the data is going to be divided into. Each partitioned attribute value will be encoded into the name of the folder. An example could be "…/Year=2023/Month=06/Day=15/" (Spark - Partition Discovery, u.d.).

Large datasets typically revolve around time series events which, if partitioned on e.g. a date attribute, can limit the file reads substantially. This is due to Data Engineering processes often using a single date of data in the daily transformations which would skip all folders and files except for the date partition with name of the date being filtered. The time series partitions are usually either based on a numeric calendar key (yyyyMMdd) or a year (yyyy), month (MM) and day (dd) partition hierarchy.

If the Spark engine must retrieve data from e.g. the date 15/6 2023 on a date partitioned Delta dataset, the filter expression would look like the example below in Figure 4. The filter expression is different depending on the date partitioning levels on the dataset.

Using the CalendarKey attribute in a filter expression on a date hierarchy partitioned dataset would force Spark to scan all Delta Log statistics in every partition and prune on file level instead of pruning on folder level. It is therefore essential to use the attributes which the dataset is partitioned by.

Partitioning	Filter Expression
CalendarKey	CalendarKey = 20230615
Year, Month, Day	Year = 2023 AND Month = '09' AND Day = '15'
	CONCAT(Year, Month, Day) = 20230615

Figure 4: Example of a Filter Expression which enable the date partitions

Physically in the Storage Account the data is written as Delta Parquet files and containing only a single date each. The files are distributed into folders in the directory which corresponds to the date or date part they hold. The date partitioning hierarchy is illustrated in Figure 5 as how it is physically stored in folder levels.

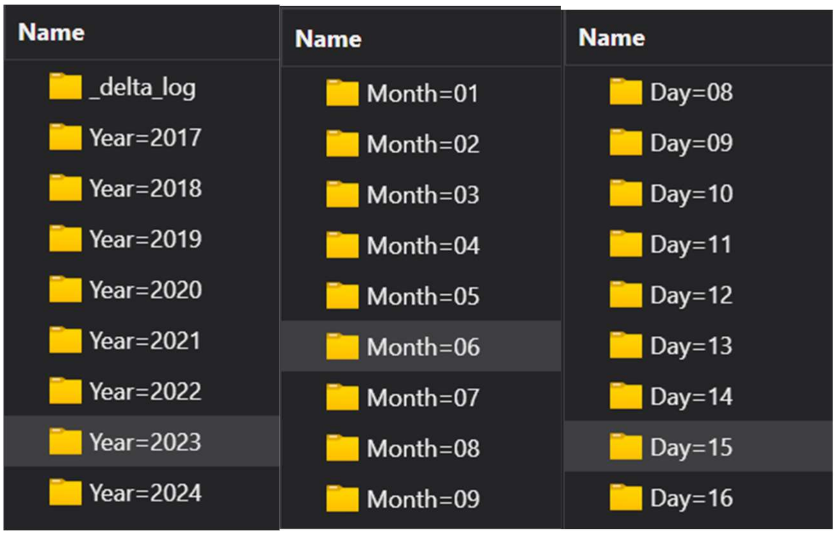


Figure 5: Example of a Storage Account date partitioning hierarchy

One of the advantages of using a date partitioning hierarchy consisting of Year, Month and Day compared to a Calendar Key is the ability to avoid date partition levels. If it is desired to retrieve data in January across all years, then a filter expression on Month = ‘01’ would be enough.

There are no limitations to partitioning keys, but it is advised to keep the “**small file problem**” in mind as the Spark engine will decrease in performance when reading too many small files compared to less bigger files.

2.3 Z-Ordering

Z-Ordering is a **Space-Filling Curve** (SFC) and is also known as the Morton Space-Filling Curve as it was first applied to a file sequence by Guy Morton in 1966 (Morton, 1966).

A Space-Filling Curve is a well-known method within the Computer Science area. It seeks to "map multi-dimensional points to one-dimensional values in a way that **preserves locality**" (Ionescu, 2018).

As a theoretical example, if a Data Analyst wants to query a specific date on a Delta Parquet dataset which has been written in regular order by a date key, it is easy for the Spark Engine to fetch the Delta Log statistics to find the files where a specific date exist and then skip all the other files. If the Data Analyst, later wants to filter for a specific user without the interest in the time aspect then it is problematic that the dataset is ordered only by the date as the users' events can potentially exists in all files as the user might watch TV 2 Play regularly. This potentially allows no data skipping which forces the Spark Engine to load all files. This of cause results in poor performance.

The Morton Space-Filling Curve tries to solve this by combining the binary representation of dimensional values into a single value. This is done by **interleaving bits** from all dimensions selected which will result in a combined binary value that the data will be sorted by when stored.

In Figure 6, a simplified example of the interleaved bits for a z-order curve is illustrated. Each dotted line represents a data file. The x- and y-axis represent a dimension each and the number along the axis is an integer and the 3-bit value of that integer. The binary values in the dimensional space are the interleaved bits in y- to x-axis order. The Z formed line represents the line when the interleaved bits are in order. To illustrate a regular Z shape in the visualization the y axis has the leading bit when interleaved.

To exemplify how the bits are interleaved we can use the y, x coordinates (100, 110) or (4, 6). The first pair of bits are 1 and 1 which is 11, the next two bits are 0 and 1 which are 01. The last two bits are 0 and 0 which are 00. This combined returns a Morton curve value of 110100 which is equivalent to 52.

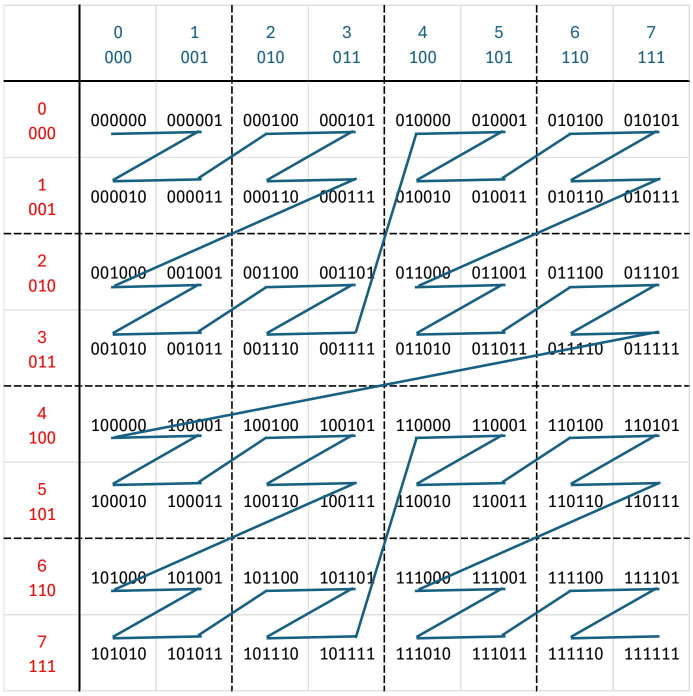


Figure 6: Morton curve illustrates with the interleaved bit values

As the bits are interleaved the data is ordered by all selected dimensions which ensures that spatially close points in the original space remain close in the resulting linear ordering (Chandra, 2023).

This simplified example shows how the Morton SFC is structured in a two-dimensional space. When increasing dimensionality in the multidimensional space the Morton SFC will run into locality issues as the spatially locality gets harder to preserve. This is due to the exponential growth in the number of possible combinations of coordinates, which makes it harder to maintain a linear ordering that accurately reflects the proximity of points in the original space. Consequently, while the Morton SFC is effective in preserving locality in two- and three-dimensional spaces, its effectiveness in maintaining spatial locality tends to diminish as the dimensionality increases.

Practically when a Delta Parquet dataset has been written with Z-ordering enabled, it will probably have new data appended regularly. The delta data will be added as new smaller files which will decrease the query performance of the Z-ordering curve as the curve has been "broken" and needs to be re-applied to minimize file reads. Therefore, the data needs to be optimized by the **OPTIMIZE** function. The OPTIMIZE function will reorder and unite the data and if stated it can reorder only a subset of the data instead of all of it. If Z-ordering has been enabled on a date column the OPTIMIZE function can be limited to a specific date range. If the Z-ordering has instead been enabled on an attribute where it is unknown which specific values has been updated, then the OPTIMIZE function must reorder all data.

The Z-ordering technique achieves optimal performance when ordering attributes with high cardinality as more files can be skipped. If a dataset would be Z-ordered by a low cardinality attribute which would have e.g. under 10 distinct values, the Morton SFC would give the same ordering value to 10 % of the dataset. One of the values could exist in multiple files while a high cardinality attribute with 100.000 distinct values is more likely to only exist in a single file if Z-ordered.

2.4 Hive Partitioning & Z-Ordering

This is the combination of the Hive Partitioning technique and the Z-Ordering technique which brings the best traits from both techniques in search for optimized performance.

Hive Partitioning strives when partitioning on attributes with low cardinality while Z-ordering strives from high cardinality attributes which can be particularly beneficial as the experiments in this project use filter expressions on the calendar and video related attributes which all have low cardinality and the UserKey attribute which has a very high cardinality.

2.5 Bloom Filter

The Bloom Filter was devised by Burton H. Bloom in 1970 as a new **hash coding technique** to determine which sets does not contain a certain element which could reject that set as relevant for the search.

Bloom introduced three factors for efficient hash coding searches which were **reject time** (average time to reject element from a set), **space** (hash area size) and **allowable errors** (false positives for elements in a set). The important part of these factors was that allowing a small number of false positives could reduce the hash area without increasing reject time. Due to this a search for an element in a set could be done much faster than not allowing any errors (Bloom, 1970).

The Bloom Filter excels at finding sets where an element is guaranteed not to be in while the rest of the sets are determined as a set where that element is probably present. The Bloom Filter can't produce false negatives but do produce false positive which is the exact trade-off of this data organization technique.

Many different Bloom Filter variations exists but for most variations three function parameters are used. Those are:

- n = Number of elements
- k = Number of hash functions used
- m = Size of bit array

The process of a Bloom Filter starts with an empty or false bit array of m bits (all values of 0). All elements in a set are hashed by each hash function and the hashed values will make that corresponding bit in the bit array return true (1).

The more hash functions used in the Bloom Filter the more time consuming the process gets but having too few hash functions would increase the probability of an element being a false positive match in the set (Mill, u.d.).

The following function will calculate the optimal number of hash functions to minimize the false positive probability:

$$k = \frac{m}{n} \ln 2$$

K needs to be a positive integer which forces m to be larger than n. In the following example I have used m = 15 which is the same value as in one of the upcoming examples of Bloom Filter indexes. As m need to be larger than n I have chosen n to be 5 which will return 2 as the optimal number of hash functions to use.

$$k = \frac{15}{5} \ln 2 = 2.08 \sim 2$$

To showcase how the Bloom Filter works (inspired by (Mill, u.d.)) we will start by creating an empty bit array consisting of m = 15 which is visualized in Figure 7. This example will use two hash functions or k = 2 which will be the FNV- and the Murmur hash function. Only a single element (n = 1) will be hashed by the two hash functions and stored in the bit array. The element hashed will be “Andreas”.

Bit value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit array	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Figure 7: An empty Bloom Filter bit array of 15 bits

When using the FNV hashing function on the element “Andreas” it returns 1 while the Murmur hashing function returns 13. Both hashed values will be set to true in the bit array which is visualized in Figure 8.

Bit value	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0
Bit array	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Figure 8: The FNV and Murmur hash functions populating the bit array

Now if an element is searched in the set, the element will be hashed by the same two hash functions. If at least one of the returned hashed values are false in the bit array the Bloom Filter determines that the element is not represented in the set. If both hashed values returned true in the bit array, the Bloom Filter would determine that it may be in the set but could be a false positive.

With the three function parameters m, k and n it is possible to determine the probability of a false positive encounter by the function (Kumar, 2023):

$$FPP = (1 - e^{\left(-\frac{-k*n}{m}\right)})^k$$

To finish the example, the FPP is calculated:

$$FPP = (1 - e^{\left(-\frac{-2*1}{15}\right)})^2 = 1.56\%$$

In Databricks the Spark engine only support selecting a single Bloom Filter attribute, setting the False Positive Probability (FPP) and the number of distinct values for the Bloom Filter attribute (n) that each file can contain. It is therefore important to configure these parameters correctly for optimized performance of the Bloom Filter index (Databricks - Bloom Filters, 2024).

In addition to the regular Delta Log on a Delta Parquet table the Bloom Filter will create a Delta Index in the same directory with an index file for each Delta Parquet file. Each file will correspond to a set and will contain the bit array for the Bloom Filter.

2.6 Bloom Filter & Z-Ordering

This is the combination of the Bloom Filter technique and the Z-Ordering technique. This combination is often seen in solutions with Bloom Filter datasets as the Bloom Filter itself is allowed only to have a single attribute set as its Bloom Filter attribute. This will result in some challenges for the dataset when not using that specific attribute in a filter expression.

When adding one or multiple attributes as Z-Ordered attributes to the dataset, the weaknesses are mostly covered as the dataset can handle a broader variety of filtering tasks.

2.7 Liquid Clustering

Liquid Clustering is the latest addition in Databricks for fast ingestion, efficient storage writing and reading capabilities. Currently as this project is in development the Liquid Clustering technique is in **Public Preview** and can only be used with some of the latest Databricks Runtimes, version 13.3 and above (Databricks - Clustering, 2024).

Liquid Clustering is using a **Space-Filling Curve** but preserves locality differently than the Morton SFC in the Z-Ordering technique. Liquid Clustering is using the Hilbert SFC which was discovered by the German mathematician David Hilbert in 1891 (Hilbert, 1891).

Even though Hilbert SFC was discovered over a century ago it has only been theorized for querying multi-dimensional data up until the early 2000 (Lawder, 2001).

The Hilbert SFC has solved some of the challenges seen in the Morton SFC. For the Hilbert SFC the next point on the curve is guaranteed to be one of its neighbors in the multi-dimensional space. For the Morton SFC the next point can be a direct or diagonal neighbor but after each $2^{(n-1)}$ Z-shape the distance between the two connecting points increases drastically. This can be seen in Figure 6.

The Hilbert SFC has a “evolutionary” behavior as it can continuously grow in **orders** which is like layers building on top of each other. The Hilbert SFC’s 1st order starts with a “bucket” shape consisting of four quadrants containing each a single data point in the multi-dimensional space. This is illustrated in the first subfigure of Figure 9. For the 2nd order each quadrant of the 1st order is divided into four quadrants with a new “bucket” shape in each quadrant with a line connecting the buckets. This continues for each order (Hilbert, 1891).

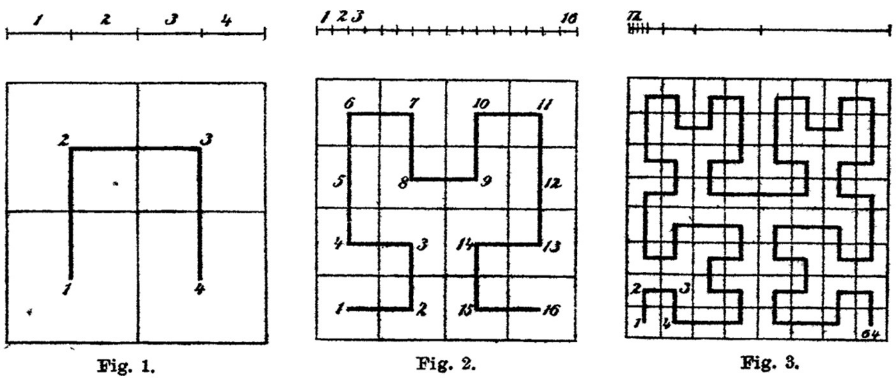


Figure 9: Hilbert curves 1st, 2nd and 3rd order (Hilbert, 1891)

Every order of the Hilbert SFC has a fixed path depending on the previous orders rotation. This is important due to the recursive schema always having a path with no crossing of previous data point and no dead ends. The directions for higher orders rotations are visualized in Figure 10 by Wirth.

If we view Figure 10 as a collection of key-value pairs separated by the colon, then the keys illustrate the baseline rotation while the values represent the “buckets” rotations and connection lines of the next order.

To exemplify the rotational behavior of Figure 10, we can use the first two subfigures of Figure 9. As seen in the first subfigure of Figure 9 we start the first order with a B key as the “bucket” is upside down. When observing the 2nd order in the second subfigure we find the points 1 through 4 which is the 1st orders first quadrant. These points correspond to a C key in Figure 10. Looking at the B key again we can see that the first value is a C key followed by an upward arrow. This illustrates that if we have a B key in the first order, then the second orders first quadrant needs to be a C key connecting to the next quadrant with an upward line and so on.

$$
\begin{aligned}
A:&\quad D \leftarrow A \downarrow A \rightarrow B\\
B:&\quad C \uparrow B \rightarrow B \downarrow A\\
C:&\quad B \rightarrow C \uparrow C \leftarrow D\\
D:&\quad A \downarrow D \leftarrow D \uparrow C
\end{aligned}
$$

Figure 10: Hilbert curve directions between orders (Wirth, 1976)

In Figure 11 the Hilbert SFC is illustrated in the same two-dimensional space as exemplified with the Morton SFC. Here we see the first three orders overlapping and illustrated by color. Due to the dimensional space being identical to the one in the Morton SFC example the buckets are reversed as the x and y axis start in the upper left corner and not in the lower left corner.

To identify the Hilbert curve value of a given x, y coordinate it is necessary to work through each order. Each order uses the corresponding interleaved bits of the x, y coordinates as the order level but will be depending on the quadrants order and rotation. Each quadrant is named from 0 to 3 in binary which makes the values 00, 01, 10 and 11 for each of the four quadrants in each order.

It is very important to notice that the values inside the dimensional space in Figure 11 are the Hilbert curve values of each coordinate and not the binary code for the interleaved bits.

In the rest of this section the term “quadrant” will be used a lot which can be confusing as quadrants can exist within other quadrants depending on the orders. To clarify which area of the graph a quadrant is in, x and y coordinate ranges will be used. If focus is on the lower right quadrant of the 1st order it will be located at x: 4-7 and y: 4-7.

If we want to know the Hilbert curve value of the x, y coordinates (110, 100) or (6, 4) in Figure 11, we must start by interleaving the first two bits. These are 1 and 1 which combined is 11.

The 1st order (green curve) starts by quadrant 0 in the upper left quadrant which locate 11 in quadrant 2 (x: 4-7 and y: 4-7) which in binary is 10. The first two-bit values in the Hilbert curve for the (110, 100) coordinate is 10.

The next two interleaved bits are 1 and 0 which combined is 10. In the 2nd order (purple curve) the "bucket" of area x: 4-7 and y: 4-7 starts in the upper left quadrant with quadrant 0. The interleaved bits 10 is in quadrant 3 (x: 6-7 and y: 4-5) which in binary returns 11. Concatenated with the results from the first order we have the binary value 1011 so far.

The last two interleaved bits are 0 and 0 which is 00. In the 3rd order (blue curve) the "bucket" of area x: 6-7 and y: 4-5 starts in the lower right quadrant with quadrant 0. The last interleaved bits 00 is in quadrant 2 (x: 6 and y: 4) which returns 10.

This gives a Hilbert curve value of 101110 which converted to an integer gives 46. This value represents where the two-dimensional value is going to be stored physically in files. Figure 11 is illustrated with 64 values in the dimensional space where this selected two-dimensional value is stored as the

46th element. If the dataset is divided into two files this value would be stored in the middle of the second file.

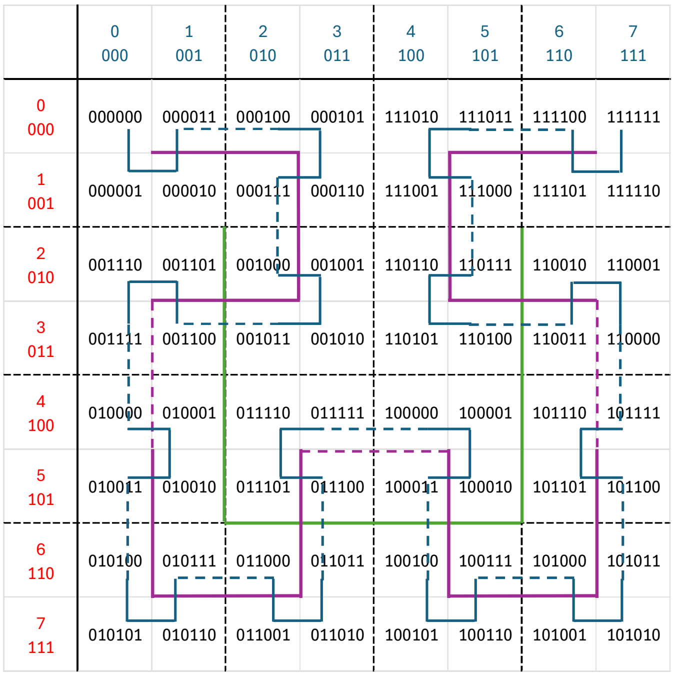


Figure 11: Hilbert curve with 1st (Green), 2nd (Purple) and 3rd (Blue) order

The Liquid Clustering technique performs better when the clustered keys are attributes with high cardinality. It is limited to **four clustered keys** but does a better job at preserving spatial locality when more clustered keys are added compared to Z-Ordering (Databricks - Liquid Clustering, u.d.).

3. Method

This chapter will contain all the methods used in the process of developing this project. The chapter will be divided into the following sections:

- Data Migration
- Performance tests
- Skipping Efficiency test
- Test results

3.1 Data Migration

All data migration happens in the notebook "002-Data Migration" which is included in the appendix. The data migration process has been created as multiple parameterized PySpark functions which each can create new datasets stored with one or multiple data organization techniques. These functions make it easy to generate new datasets variations for testing different techniques.

Hive Partitioning and Z-Ordering

The PySpark function "**copy_data**" copies data from a source table and write it to a target delta path in delta parquet format. If the partition or optimize parameters are set, the function can partition and/or z-order the data by the listed attributes.

The "copy_data" function has been used for the Base dataset, Hive Partition- and Z-Ordering datasets and the combination of Hive Partition and Z-Ordering datasets.

The Hive Partitioning happens in the write function while the Z-Ordering happens in an OPTIMIZE function after the data has been written to Delta Parquet format. The PySpark code of the "copy_data" function is illustrated in Figure 12 below.

```python
def save_dataframe(df, allow_schema_evolution, path, data_format, write_mode, partition_by=None):
  df_temp = df.write\
    .mode(write_mode)\
    .format(data_format)\
    .option("mergeSchema", allow_schema_evolution)

  if partition_by is not None:
    df_temp = df_temp.partitionBy(*partition_by)

  df_temp.save(path)

def copy_data(source: str, delta_data_path: str, partition_by: str=None, optimize_by: str=None):
  try:
    source_df = spark.table(source)
  except Exception as e:
    source_df = source

  save_dataframe(df=source_df,
                 allow_schema_evolution='true',
                 path=f'abfss://curated@tv2dwstorageproject.dfs.core.windows.net/{delta_data_path}',
                 data_format='delta',
                 write_mode='append',
                 partition_by=partition_by
                 )

  if optimize_by is not None:
    spark.sql(f'''
      OPTIMIZE DELTA.`abfss://curated@tv2dwstorageproject.dfs.core.windows.net/{delta_data_path}`
      ZORDER BY ({optimize_by})
    '''
    )
```

Figure 12: copy_data function illustrated

An example of the "copy_data" function being called is illustrated in Figure 13. This example shows the creation of the Hive Z Order dataset variation.

```python
#Hive Z Order
copy_data(
  source = micro_data, \
  delta_data_path = 'adobeanalyticsmicro_hivezorder', \
  partition_by = ['Year', 'Month', 'Day'], \
  optimize_by = 'UserKey'
)
```

Figure 13: copy_data function execution illustrated

Bloom Filters

Bloom Filter dataset variations are created with the "**bloom_filter_builder**" function to handle the process of creating a new table with a Bloom Filter index enabled.

The process of creating a Bloom Filter index starts by creating a new empty delta parquet dataset with a predefined schema which for this function is hardcoded to copy the schema from the Hive Partition Base dataset. Other datasets would suffice as well. The "bloom_filter_builder" function is illustrated in Figure 14 below.

```python
def bloom_filter_builder(bf_table_path, bf_columns, zorder_columns, fpp=0.1, numItems=1000000):
    root_path = "abfss://curated@tv2dwstorageproject.dfs.core.windows.net/"
    source_path = f"{root_path}adobeanalyticsmicro_hivepartition_base"
    source_df = spark.read.format("delta").load(source_path)
    source_schema = source_df.schema
    schema_str = ", ".join([f"{field.name} {field.dataType.simpleString()}" for field in source_schema.fields])

    spark.sql(f"CREATE OR REPLACE TABLE DELTA.`{root_path}{bf_table_path}` ({schema_str})")
    spark.sql(f"CREATE BLOOMFILTER INDEX ON TABLE DELTA.`{root_path}{bf_table_path}` FOR COLUMNS({bf_columns}
              OPTIONS (fpp={fpp}, numItems={numItems}))")
    spark.sql(f"INSERT INTO DELTA.`{root_path}{bf_table_path}` TABLE DELTA.`{source_path}`")
    spark.sql(f"OPTIMIZE DELTA.`{root_path}{bf_table_path}` ZORDER BY ({zorder_columns})")
```

Figure 14: bloom_filter_builder function illustrated

Next the Bloom Filter Index is created on the dataset by the parameterized attributes with default bloom filter options configured unless anything else is specified by the input parameters. The options available are the False Positive Probability (FPP) rate at default 0.1 and the Number of Unique Items at default 1 mil.

```python
# Bloom Filter
bloom_filter_builder(
  bf_table_path  = "adobeanalyticsmicro_bloomfilter", \
  bf_columns = "VideoFlag", \
  zorder_columns = "CalendarKey, UserKey"
)
```

Figure 15: bloom_filter_builder function execution illustrated

After creating the Bloom Filter Index, the new dataset must be populated with data to create the **Delta Index files** (hash function log). The "bloom_filter_builder" function is hardcoded to populate the new dataset with data from the Hive Partition Base dataset as it is already present in the function from the schema copy.

Lastly the "bloom_filter_builder" function has the option to Z-Order columns on the dataset. This is another performance possibility which is utilized in some of the Bloom Filter datasets.

In Figure 15 an example is shown of how the function is called to create the base Bloom Filter dataset. The function is creating the new dataset with a Bloom Filter on the VideoFlag attribute and Z-Ordering by the CalendarKey and UserKey attributes.

Liquid Clustering

The data migration process for Liquid Clustering is created with the function "**liquid_cluster_builder**". This function creates a new empty delta parquet dataset with a target delta path and cluster columns parameterized. As the new clustered delta parquet dataset is empty it must be populated with data to cluster the data. It is hardcoded to retrieve the data from the YMD Hive Partitioned dataset as this dataset has simple configurations. Most other datasets could suffice as well. The code for the "liquid_cluster_builder" function is illustrated below in Figure 16.

```
def liquid_cluster_builder(lc_table_path, cluster_columns):
    root_path = "abfss://curated@tv2dwstorageproject.dfs.core.windows.net/"
    cluster_columns_str = ", ".join(cluster_columns)
    spark.sql(f"""
        CREATE OR REPLACE TABLE {lc_table_path} USING DELTA
        CLUSTER BY ({cluster_columns_str})
        LOCATION '{root_path}{lc_table_path}'
        AS SELECT * FROM DELTA.`{root_path}adobeanalyticsmicro_hivepartition_ymd`
    """)
```

Figure 16: liquid_cluster_builder function illustrated

In Figure 17 it is shown how the "liquid_cluster_builder" is called to create the dataset variations. This example shows the creation of the base Liquid Cluster dataset, which is clustered by the CalendarKey, SiteKey and UserKey.

```
# Liquid Clustering
liquid_cluster_builder(
  lc_table_path = "adobeanalyticsmicro_liquidclustering", \
  cluster_columns = "CalendarKey, SiteKey, UserKey"
)
```

Figure 17: liquid_cluster_builder function illustrated

3.2 Performance Tests

The code to the performance tests is in the notebook called "003-Performance Tests" which is included in the appendix. The performance tests are all created with a single parameterized PySpark function called "**performance_test**" which has a delta table, iteration number and test type as its input.

The **delta table** parameter is the suffix of the dataset name as all datasets are stored in the same storage account and container. The **iteration number** is the number of iterations a predefined SQL query should be executed against the datasets. The **test type** can be set to either "analytics" or "userkey" which corresponds to whether the iterations are executing a SQL query with a filter expression on a date period and active video for “analytics” or a date period, active video and user keys for “userkey”. There are other minor differences in the SQL query, but these are the main characteristics of the test type parameter. The “analytics” or “userkey” test type queries are illustrated in Figure 19 and Figure 20 below.

Through the performance tests it was discovered that Databricks is so efficient that performance most likely can’t be reliably measured in succession as a data cache or query plan must be stored somewhere due to all iterations except the first one resulted in almost the same response time across all datasets.

As only the first iteration returned valid data, the testing process was challenged as a data sample of at least 30 observations was expected to create a dataset representing a **Gaussian distribution** (LaMorte, 2016). Therefore, the number of iterations was reduced to 3-5 iterations instead of 30 as there still needs to be some iterations to be able to test whether the first iteration was valid compared to the rest.

The 30 observations would instead be achieved by running the notebook 30 times for each of the two test types in a Databricks workflow with a job cluster. This is very far from optimal but was the only option left to secure valid data.

```
i = 3
type = dbutils.widgets.get("type")
datasets = ['base', 'hivepartition_base', 'hivepartition_site', 'hivepartition_ymd', 'hivepartition_video', 'hivepartition_video2',
'zorder', 'zorder_video', 'hivezorder', 'hivezorder_video', 'liquidclustering', 'liquidclustering_video', 'liquidclustering_video2',
'bloomfilter', 'bloomfilter2', 'bloomfilter_video2', 'bloomfilter_video3', 'bloomfilter_video4']

for dataset in datasets:
  performance_test(dataset, i, type)
```

Figure 18: performance_test function illustrated

Many attempts have been made in search of an answer or a solution, but nothing returned any success. The following hypothesis has been made, evaluated and tested:

- **Exchange Reuse:** Shuffle files are stored for later use which will make iterations on the same dataset more efficient as some data is cached. This has been disproved to be the issue as the issue persists when clearing caches, unpersist dataframes and RDDs. Even when iterating over different periods or randomizing the order of the datasets to force shuffle overwrites, had no impact on the issue

- **Spark Optimization Settings:** Some of the Spark Optimization settings are automatically enabled and some are not. This category includes Caching, Optimizers and AQE (Adaptive Query Execution). By disabling all Spark Optimizations settings, the issue persists

Performance Test Queries

As previously stated, we have two different types of performance test against the generated datasets with different data organization techniques enabled. The two types are created to accommodate the needs of both the Data Engineering's aggregated transformations and the Data Analysts or Data Scientists individual search.

The two queries are visualized and described below. Both queries are similar in their structure but have different filters and requirements for aggregation. In addition, both have a dynamic calendar filter and video filter which are dataset depending as the partitions and keys are often different. The import part here is to keep both queries close to the original queries from the Curated Micro Data dependent notebooks.

Analytics Query

The Analytics query is illustrated in Figure 19. As the Analytics query focus on aggregation the query is grouped by the CalendarKey as a large period of data is collected in the filter expression. The query summarizes on the VideoTime attribute, counting on the VideoPlayCount attribute and lastly distinct count on the UserKey attribute.

```sql
SELECT f.CalendarKey
      ,SUM(f.videoTime) as VideoTime
      ,COUNT(f.VideoPlayCount) as VideoPlayCount
      ,COUNT(DISTINCT f.UserKey) as DistinctUserKey
FROM DELTA.`abfss://curated@tv2dwstorageproject.dfs.core.windows.net/adobeanalyticsmicro_{delta_table}` f
WHERE {calendar_filter} BETWEEN 20231119 AND 20231220
    AND {video_filter}
GROUP BY f.CalendarKey
```

Figure 19: Analytics performance query illustrated

Each of the aggregated attributes has the purpose of challenging the file pruning abilities of each dataset and especially the distinct count calculation is hard to utilize through the Delta Log statistics. The more files skipped by the data organization technique in the query filter the less files has to be aggregated on.

Individuality Query

The Individual focused query illustrated in Figure 20. The query is grouped by the UserKey attribute as there are two specific UserKeys being filtered in addition to the calendar- and video filter.

```sql
SELECT f.UserKey
      ,SUM(f.videotime) as videotime
FROM DELTA.`abfss://curated@tv2dwstorageproject.dfs.core.windows.net/adobeanalyticsmicro_{delta_table}` f
WHERE {calendar_filter} BETWEEN 20231119 AND 20231220
    AND {video_filter}
    AND f.UserKey IN (
        '125341c446fc3af86b9660ef16ddc971',
        '4823c6d95ad3443384fba248f1fa9a7a'
    )
GROUP BY f.UserKey
```

Figure 20: Individual performance query illustrated

Due to the large date period being filtered the UserKeys could very well exist in all dates depending on their user activity making it challenging for some of the data organization techniques.

Compared to the Analytics query this query does only have a summarize of the VideoTime attribute. It was not necessary to make other aggregations due to it not being the focus and due to tougher pruning effectiveness for the individual query.

3.3 Skipping Efficiency Test

The Skipping Efficiency Test consists of five metrics to help determine how a data organization techniques pruning capabilities perform. The terms and processes of the Skipping Efficiency test are inspired by the Tech Lead at Databricks Adrian Ionescu's blog post on Data Skipping and Z-Order Clustering (Ionescu, 2018). The metrics are:

- **Total Files:** The total number of files stored in a dataset. The number of total files will vary depending on the parameters used and the data organization technique's ability to compress files
- **Files Scanned:** The number of files a filter expression returns based on the Delta Log statistics. If a specific date is filtered in a query how

many files have a min and max date value which is between the specific date in the Delta Log statistics

- **Files Skipped:** The files skipped which did satisfy the filter expression based on the Delta Log statistics, but the files did not contain relevant data when read into memory. To exemplify this, a delta dataset with 20 total files is being filtered which determine ten of those files to be scanned. The files are read into memory and only four files contain the value being filtered, then six files have been skipped in the process. The files skipped are also called False Positive files as Spark must read them due to their Delta Logs statistics which is unnecessary as no relevant data exist in those files
- **Scan Skipped Efficiency:** This metric measures how well a Delta dataset can skip files when being filtered based on the Delta Log statistics. If a Delta dataset has 100 total files and 60 are scanned due to the Delta Log statistics, then the remaining 40 files are skipped, and the data organization technique of that dataset has a 40 % Scan Skipped Efficiency (40 / 100 = 0.4). The goal is having a Scan Skipped Efficiency as close to 100 % as we want to initially skip as many files as possible before reading any data. One file needed to be read compared to five will greatly affect the performance
- **File Skipped Efficiency:** This metric measures how well a Delta dataset can avoid files read being false positive reads. If 60 files are scanned due to Delta Log statistics and 40 of those files are skipped due to being false positive reads, then the data organization technique of that dataset has a 67 % File Skipped Efficiency (40 / 60 = 0.67). The goal of this metrics is to have a File Skipped Efficiency as close to 0 % as this would signal that all files read are relevant and Spark not wasting any compute power

Scan Skipped Efficiency and File Skipped Efficiency can't be evaluated and concluded upon individually. They must be used together and in collaboration with the Total Files. This is exemplified in the next section.

If a dataset is being filtered and has a 40 % File Skipped Efficiency, it might seem to be inefficient as most reads are false positive reads but if the Scan Skipped Efficiency is 90 % then most files are already skipped. Now if there are a total of 1,000 files then the 90 % Scan Skipped Efficiency still leaves 100 files to be read into memory while 40 % of those files are false positive reads as they do not contain any relevant data.

In the appendix the "003-Skipping Efficiency Test" notebook shows how the metrics are calculated. The filtered attributes in the Skipping Efficiency experiment are the date attributes (Year, Month, Day or CalendarKey), the video attributes (VideoFlag or VideoInfoCode) and the UserKey attribute. The exact attributes used are dynamically tunes based on which dataset is being tested.

3.4 Test Results

To evaluate the test results all results are pairwise mapped to calculate the p-value of their t-score for evaluating whether the datasets are **statistically significantly different** in that pairwise mapping.

This will evaluate whether two datasets or data organization techniques are statistically the same or not. If they are statistically the same there are no benefits of choosing one over the other and therefore the methods are considered equal in performance.

The p-value scores will be illustrated through a heatmap where only values above 0.05 are colored as this indicates those dataset combinations are considered statistically significant and therefore equal in terms of performance.

The individual and aggregated performance test results are scanned for outliers before getting illustrated. An outlier is a sample datapoint which has an abnormal distance to the rest of the dataset and most likely generated due to an external mechanism. The outlier detection method used in this project is the "**three-sigma rule of thumb**". This implies that an outlier is classified as an outlier if the distance from the mean exceeds the third standard deviation which will retain 99.97% of the data and exclude 0.015% at each end of the Gaussian distributed dataset (Osborne & Overbay, 2004).

When outliers have been detected and removed from the dataset the data is then illustrated in **Violin Graphs** as this type of graph combine the Boxplot with a Histogram as it becomes visible where the mean and quartiles are located and how well the datapoints complies to the Gaussian Distribution.

4. Analysis and Results

This chapter is divided into the three sections which each revolve around one of the experiments performed. The experiments are:

- Skipping Efficiency Experiment
- Individual Performance Experiment
- Aggregated Performance Experiment

For each experiment we will:

- Describe the purpose of the experiment
- Describe the expectations for the experiment
- Illustrate the outcome of the experiment
- Analyze the outcome and conclude

4.1 Skipping Efficiency Experiment

The purpose of this experiment is to illustrate how efficient each data organization technique is at pruning files at query time. Each dataset variation can show how the techniques differ from each other only by setting new partitions or keys, or even changing parameters.

This section is divided into skipping efficiency on an individual level and on aggregations. This is necessary as the pruning capabilities of each technique can vary significantly as some are efficient at finding specific values while others are great at locating and aggregate on larger subsets of the data. Specific values are usually queried by Data Scientists and Data Analysts while larger subsets of the data are needed by Data Engineers.

Furthermore, the Skipping Efficiency experiment search to test whether effective data skipping corresponds to high performance in terms of low query response times in the performance experiments.

Expectations:

This experiment focus on how well a dataset variation or data organization technique is at skipping files. The efficiency is depending on the number of files, their volumes and the Delta Log statistics. The aggregated experiment uses two attributes in the filter expression on each of the datasets which are the date attributes and a video attribute combination. The individual experiment filters the same attributes while also adding the UserKey attribute for the individual level.

The date attributes have low cardinality which should be advantageous for partitioning techniques. The disadvantage for the partitioning technique might be when on the individual level as the UserKeys have a high cardinality and does not fit as a partitioning attribute itself. Both experiments filter for a 30-day period out of the total 61 days. This is easy for the partitioning to prune for, but this only skip half of the files on date partitions. If there are not any other partitions on the dataset then Spark needs to read all the remaining files.

All variation of the video attributes has a cardinality lower than the date attributes which again should favor the partitioning techniques. Only issue is when the video attributes are not part of the partitions. Depending on the video variation it might cause a small file problem even if the video attributes are partitioned.

The UserKey attribute has a high cardinality which should really challenge the partitioning technique but should be advantageous for almost any other data organization technique as the Space-Filling Curves and Bloom Filters are efficient at high cardinality attributes.

The Z-Ordering, Liquid Clustering and Bloom Filter techniques should have a high Scan Skipped Efficiency as the SFC's focus on preserving locality and they all strive to have a low false positive rate compared to Hive Partitioning.

Results:

Below in Figure 21 are the results of the Skipping Efficiency Experiment on individuals and aggregations. The test results are illustrated in a horizontal bar chart with a legend containing the color code and metrics name which are "total files index", "scan skipped efficiency agg", "scan skipped efficiency user" and "file skipped efficiency user".

File Skipping Efficiency on aggregation is not represented in the figure as this metric is zero for all datasets. This is the result of the high cardinality attribute UserKeys being excluded from the filter, with only the date and video attributes remaining. Both attributes have very low cardinality which makes it near impossible to not satisfy the filter when looking inside the Delta Log statistics.

The “total files index” was added to illustrate how well a data organization technique is at compressing data into files. The bar shows the percentage of each datasets number of files compared to the total number of files. This bar is relevant to include as it will give the reader a quick indication of potential issues in the dataset variations like “small file problems”. The number at the end of each bar is that datasets total number of files.

The “scan skipped efficiency” bar for either individuals or aggregations shows the percentage of how many files compared to the total amount were skipped due to the Delta Logs statistics.

“File skipped efficiency” is the percentage of how many files were false positive file reads.

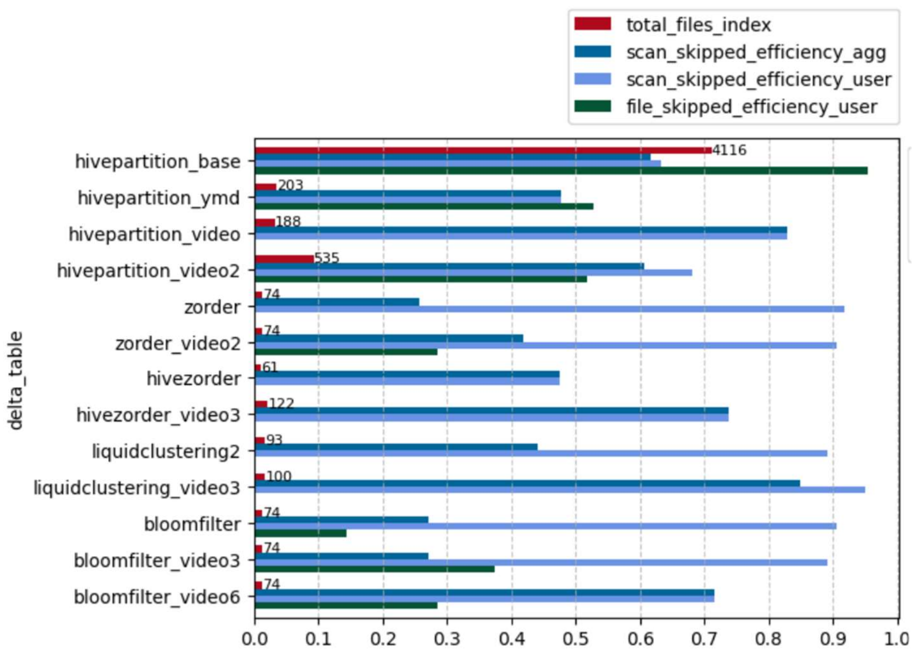


Figure 21: Illustrating the outcome of the Skipping Efficiency Experiment

It is important to mention that all the datasets in Figure 21 are not the full list of datasets which this experiment was performed on, but only a chosen subset as the excluded datasets either contained datasets which were identical or almost identical or not interesting for this project in retrospective. This subset of datasets has been chosen for all three experiments.

Analysis:

The original Base and the Hive Partitioning Base datasets are identical in this experiment and here it is illustrated that these datasets suffer from the Small File Problem as they have five levels of partitioning. There are 4.116 total files for each dataset which results in small files that can be hard for the Spark engine to collect while less files with higher volume of events are preferred in Spark.

It is observed that most of the datasets using the Hive Partitioning technique, or use it in combination with Z-Ordering, has a similar Scan Skipping Efficiency between individuals and aggregations. This is most likely because the file pruning process happens on partitions which are challenged by the UserKeys as this attribute has too high cardinality to be partitioned. Therefore, the individual and aggregated filter expressions scan skips the same partitioned files as UserKeys cannot be filtered through partitions. This creates a high level of false positive file reads or files with badly distributed data as all scanned files contains the needed data, but the majority of the data is unnecessary for the filter.

Furthermore, the datasets Hive Base, Hive YMD and Hive Zorder are all located near the 50 % mark on Scan Skipped Efficiency. The reason behind this are the datasets mostly being partitioned by the date attributes which is one partition for each day. As the filter used in this experiment search for a specific 30-day period and each day could contain one of the UserKeys being filtered, all 30 date partitions could satisfy the filtered requirements and therefore around half of the partition files are skippable.

The Hive Base dataset has a bit higher Scan Skipped Efficiency even though it is partitioned on Year, Month and Day. This might be because of the Hive Base dataset having five levels of partitions which results in many smaller files that will increase the possibility of the Delta Log statistics stating that the UserKeys being searched for are not in the min and max range. Therefore, it makes more files skippable. This does not correspond to better performance as this dataset still suffers from the small file problem which will be investigated in the performance experiments.

It is clear to see in Figure 21 that all datasets using a Space-Filling Curve, or a Bloom Filter has a well compressed file structure, and a high Scan Skipped Efficiency on individuals. These data organization techniques are highly efficient at collecting specific values in high cardinality attributes. When challenging these techniques to collect larger subsets of data based on low cardinality attributes their Scan Skipped Efficiency (aggregated) are significantly lower than their individual's counterpart.

The success criteria for an optimized data organization technique solution, based on its pruning capabilities, is to involve high individual and aggregated file pruning while having a well compressed file structure. Only a few of the datasets achieve these success criteria and will be described further below.

The Hive Partition Video dataset has a reasonable compressed data structure consisting of 188 files divided into four levels of partitions which are Year, Month, Day and VideoFlag (True when at least one of the three video attributes are active). Under the development of this project, it was discovered that nearly 75 % of the data does not have any active video attributes which makes it a great partitioning target. When filtering the dataset by the 30-date period and by active video attributes most files are pruned by scanning the Delta Log statistics as both filters are partitioned attributes with low cardinality. There are no false positive file reads as all remaining files contains at least one of the two UserKeys being filtered. This is most likely a coincidence.

The Liquid Clustering Video 3 dataset consists of 100 files which is fairly compressed compared to most Hive Partitioning dataset variants. This dataset is clustered on three attributes which are CalendarKey, UserKey and VideoFlag. All filters used in the individual and aggregated skipping efficiency experiment are satisfied by this dataset clustered keys. As Liquid Clustering datasets are using the Hilbert Space-Filling Curve it is very efficient at preserving the data locality from the multidimensional space to the one-dimensional space. The is why the dataset does not suffer for not using the UserKey clustered key in the individual experiment compared to Z-Order Video 2. The Z-Order Video 2 dataset has been ordered by the same attributes as the Liquid Clustering Video 3 dataset but suffers greatly with only around 41 % Scan Skipping Efficiency on the individual experiment compared to 85 % on the Liquid Clustering Video 3 dataset.

Experiment conclusion

The datasets "Hive Partition Video" and "Liquid Clustering Video 3" has the highest performing pruning capabilities as both are scan skipping over 82% of the data in both individual and aggregated experiment by checking the Delta Log statistics. Additionally, there are no false positive file reads in any of these datasets.

4.2 Individual Performance Experiment

The purpose of this experiment is to identify the most efficient dataset variation at filtering on an individual level. This is to assure business needs for efficient individual analysis are met. Data Analysts and Data Scientists in the TV 2 Danmark organization often wants to test the dataset on an individual level which with no efficient dataset data organization technique enabled can cause long inefficient query times. This experiment pursues the optimized solution for individual filtering in a Delta Parquet dataset. The best performing datasets will be evaluated for statistically significance by the p-value of a t-test.

Expectations:

In this experiment a standardized query is executed on each dataset which filters a 30-date range, a combination of the video attributes and two specific UserKeys which potentially can exist on each date with activated video. This task can be expensive for some Hive Partitioning datasets as more files needs to be read because of the UserKey attribute having too high cardinality for being partitioned. Therefore, the basic Hive Partitioning datasets should be struggling in this experiment.

It is expected that the Liquid Clustering technique will be among the best performing techniques in general as its use of the Hilbert Space-Filling Curve makes it easier to target specific values in big datasets. This is due to each file being ordered by the Hilbert Curve which order data with all clustered values in mind, opposed to the Hive Partitioning technique where all data in each file are written as inserted only divided by the partitions. The Z-Ordering technique should have similar performance to Liquid Clustering but not quite as effective as the Morton Space-Filling Curve should be somewhat inferior to Hilbert due to the regularly jumps in the one-dimensional space as illustrated in the 2.3 Z-Ordering chapter.

Results:

Below in Figure 22 are the results of the Individual Performance Experiment shown. As previously stated, the datasets displayed are only a subset of the tested datasets. The full list of datasets is included in the appendix chapter.

The figure shows each dataset on the y-axis and the performance in response time measured in seconds on the x-axis. The results are visualized by a violin chart which illustrates how the data points are distributed which can give insight on whether the data points follow a gaussian distribution or has a narrow or wide standard deviation.

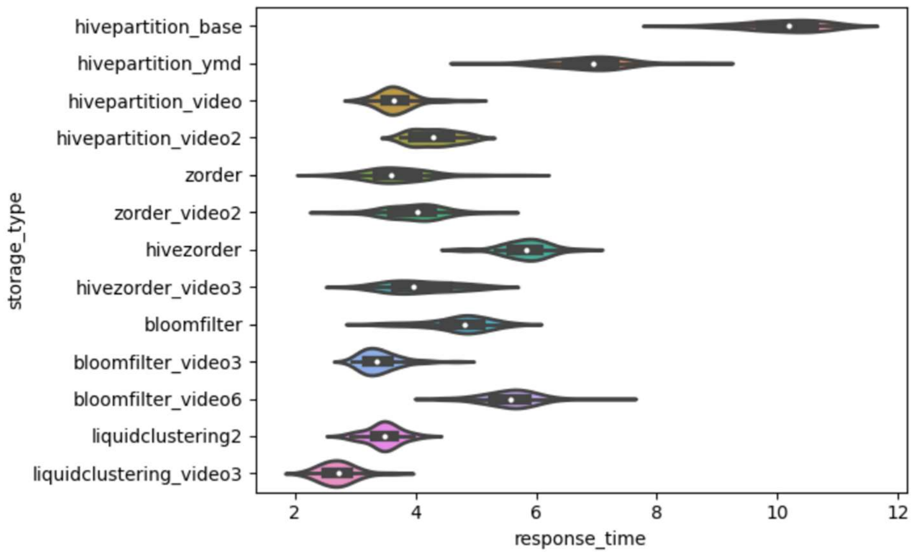


Figure 22: Individual Performance Experiment illustrated

In Figure 23 a heatmap is shown of the statistically significance between all the datasets variations in the individual performance experiment. Each axis displays all datasets, and the red areas reveal combinations of datasets which are statistically significant. If a combination of datasets is statistically significant it indicates that these datasets are statistically the same which makes none of them superior to the counterpart. Each grey combination indicates that these datasets are statistically different and therefore one is superior to the counterpart.

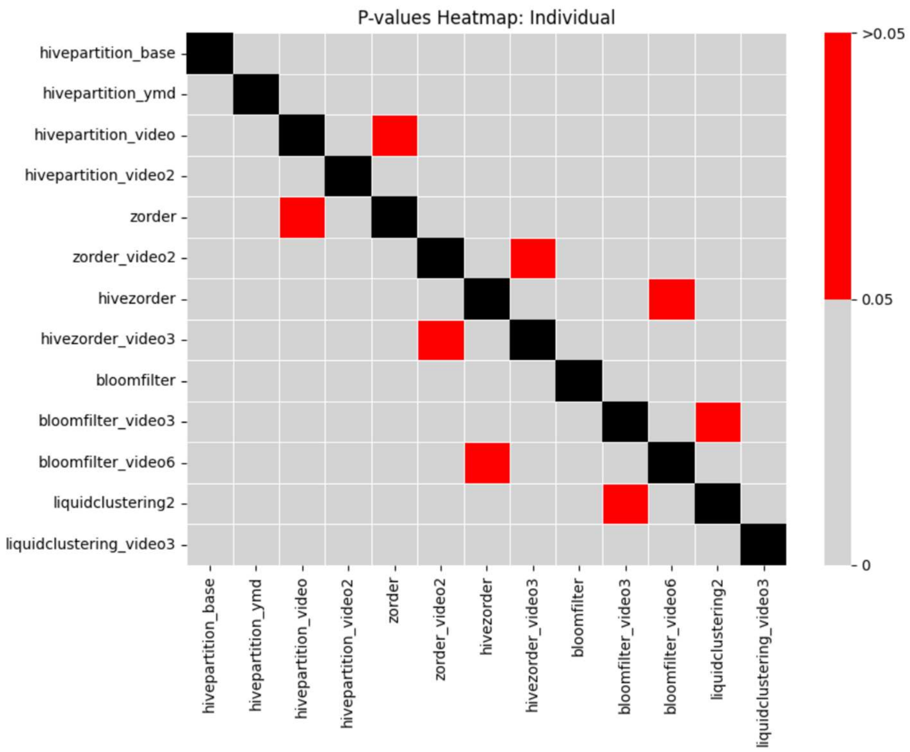


Figure 23: Heatmap of statistical significance between datasets on individual experiment

Analysis:

For the Individual Performance Experiments results shown in Figure 22 the performance of each dataset revealed to have large deviations between the data organization techniques. Each technique has some high and low performers in response time.

The most variation in performance is seen in the Hive Partitioning technique as it has the lowest performing dataset and one of the best performing datasets. The Hive Partitioning Base dataset has the longest response time among the illustrated datasets while the Hive Partitioning Video dataset has a very promising response time of 3.7 seconds on average.

Among the not illustrated datasets in the figure we have the original base dataset which is excluded in this visualization due to its extremely poor performance with an average response time of 53.8 seconds. If included it would cause visualization problems as the other datasets would become hard to distinguish between as they would be stuck in a narrow visual space.

The Hive Partitioning datasets are in most cases not performing very well which is mainly caused by the poor file pruning as the UserKey attribute has too high cardinality to become a partitioning attribute. Therefore, this technique only relies on the date attributes and the video attributes if enabled as a partition on the dataset.

For the two Hive Video datasets, files can be pruned by excluding all partitions not satisfied by the query's calendar and video filter which leaves a small subset of files to read for the specified UserKeys. The only difference between Video and Video 2 is that the Video dataset is partitioned by the VideoFlag attribute (1 or 0) while the Video 2 dataset is partitioned by the VideoInfoCode (e.g. 001 or 101) which is a concatenation of flags for the three video attributes. As the VideoInfoCode has a higher cardinality than VideoFlag there are more partitions and therefore more files to scan and read which explains the difference in their performance.

The datasets not having the video attributes as a partitioning attribute rely solely on the Delta Log statistics to skip files based on video and UserKeys. This can be an expensive process as the UserKeys might exist in all date partitions depending on the chosen users' behavior.

The two Z-Ordering datasets are among the better performing datasets and both datasets are Z-ordered by CalendarKey and UserKey. The Video 2 version of the Z-Ordering dataset has VideoFlag as an additional ordered attribute. As illustrated in Figure 22 the regular Z-order dataset has a mean response time a bit faster than the Video 2 version which is positively due to the VideoFlag addition. The VideoFlag attribute has a low cardinality and is the third Z-ordered attribute on the dataset which makes the locality harder to preserve opposed to only two ordered attributes.

The better performance in response time for the Z-Ordered datasets are caused by the filtering quickly being satisfied as both CalendarKey and UserKey are among the ordered attributes. When the dates and the specific UserKeys are isolated in the files it only must identify the active video events when reading.

The Hive Partitioning and Z-Ordering combined datasets contains Year, Month and Day as the partition levels and has UserKey as the Z-ordered attribute. The Video 3 version has VideoFlag as an additional partitioned attribute which is observed to make a difference in the mean response time. The VideoFlag is a strong partitioning choice as previously stated due to its great file skipping potential in an **unbalanced partition** which often is distributed with 25 % of the data in one partition and 75 % in the other.

Both datasets have a single Z-ordered attribute which basically corresponds to a one-dimensional space being projected into a one-dimensional space. This is the same as if that attribute was regularly ordered by the Z-ordered attribute and therefore does not benefit from the Morton Space-Filling Curve.

There are three Bloom Filter versions illustrated in Figure 22 with three very different mean response times and standard deviations. The regular Bloom Filter dataset has the VideoFlag attribute as its Bloom Filter key while being Z-Ordered by CalendarKey and UserKey. The VideoFlag has a low cardinality which is not recommended as a Bloom Filter key as the hash functions can only be tested on 1 or 0. Therefore this dataset is mostly relying on the Z-ordered attributes which should result in response times similar to the regular Z-Order dataset.

Even if they theoretically should be similar, then this is not the case as the Bloom Filter- and the Z-Order datasets are statistically significantly different as seen in Figure 23. The reason for this is revealed by the Skipping Efficiency Experiment. In the experiment it is shown that both datasets have 74 files, but the Z-Order dataset has a bit higher Scan Skipping Efficiency on individual level than the Bloom Filter dataset. Furthermore, the Bloom Filter dataset have near 13 % false positive file reads in the Files Skipping Efficiency while the Z-Order dataset has none. This gives the regular Z-Order dataset a minor performance advantage.

The Bloom Filter Video 3 dataset has a much better performance than the Bloom Filter dataset. This dataset has the VideoInfoCode attribute as its Bloom Filter key and is Z-ordered by the CalendarKey and UserKey as the regular Bloom Filter dataset. The VideoInfoCode attribute has a low cardinality but a bit higher than the VideoFlag. The increase in cardinality for the Bloom Filter key has made it easier for the hash functions to help the Spark engine skip files.

Bloom Filter Video 6 uses the high cardinality attribute UserKey as the Bloom Filter key and is Z-ordered by CalendarKey and VideoInfoCode. Furthermore, the Bloom Filter key is limited to 10.000 unique values in each file. The dataset should benefit from the high cardinality Bloom Filter key compared to the other Bloom Filter datasets as these use low cardinality Bloom Filter keys. The reason for this is that the hash functions can do a better job at excluding files which does certainly not contain a specified UserKey. As this experiment is filtering each dataset on users, a period and active video events, only the period and active video events are remaining in the filter. Both CalendarKey and VideoInfoCode are low cardinality attributes which does not benefit the Z-ordering technique. This might be the reason for the dataset performing this poorly.

The Liquid Clustering datasets are the top performing technique in mean response times for the Individual Performance Experiment. The Liquid Clustering 2 and Liquid Clustering Video 3 datasets both have CalendarKey and UserKey as their cluster keys. The only difference is the Video 3 dataset having the VideoFlag attribute as a cluster key as well which is the reason for Video 3 being the highest performer. All cluster keys in Video 3 are used by the filter expression and the Hilbert SFC is not suffering in performance in the same way the Morton SFC (Z-Ordering) does for higher dimensionality.

Furthermore, the Video 3 dataset is statistically significantly different from all other datasets while Liquid Clustering 2 is statistically significant to the Bloom Filter Video 3 dataset.

Experiment conclusion

The dataset “Liquid Clustering Video 3” has the lowest mean response for the individual experiment by only 2.3 seconds. The reason behind this performance is due to all filtered attributed are cluster key in the dataset which fully utilize the Hilbert Space-Filling Curve for file pruning.

4.3 Aggregated Performance Experiment

The purpose of this experiment is evaluating all the data organization techniques to identify the most suitable technique and settings in terms of lowest response times in TV 2's Data Platform transformations for fast updated tabular models or datasets to the end-users. This means that this experiment is focused on data aggregation on larger subsets of the data and does not care for individual filtering. The best performing datasets will be evaluated for statistically significance by the p-value of a t-test.

Expectations:

This experiment is filtering each dataset by a larger date range and on active video events and aggregate the data by sum, count and distinct count functions. This task should therefore be favored by the partitioning technique as all files not satisfying the specified date range and active video events can be skipped.

All other techniques should have difficulties locating all the data as it could be spread across more files. If a dataset is using a Space-Filling Curve the data is ordered by the curve which could spread a single date into multiple files. Additionally, most datasets using a Space-Filling Curve has the UserKey as one of its keys. As the UserKey has the highest cardinality it should only perform the best when filtered. When not filtering on UserKey but instead on the low cardinality keys it might cause some issues for those data organization techniques.

The Bloom Filter technique is used in combination with the Z-ordering technique which, as described above, should cause some issues for the performance. The Bloom Filter itself should also have trouble being utilized as most files will contain relevant data.

Results:

Underneath in Figure 24 and Figure 25 are the same illustrations as seen from the Individual Performance Experiment. Instead, they are now populated with data from the Aggregated Performance Experiment. As the illustrations are the same, they will not be elaborated further on as this would be redundant. Only new relevant information will be added.

In Figure 24 it is important to note that the x-axis is not starting at zero, but at six while each tick is two seconds in the response time. This makes the difference between each dataset much more impactful as it is noticeable for a user when waiting eight or twelve seconds for a response compared to the previous experiment on individuals where the wait time would be less impactful when comparing three seconds to four seconds.

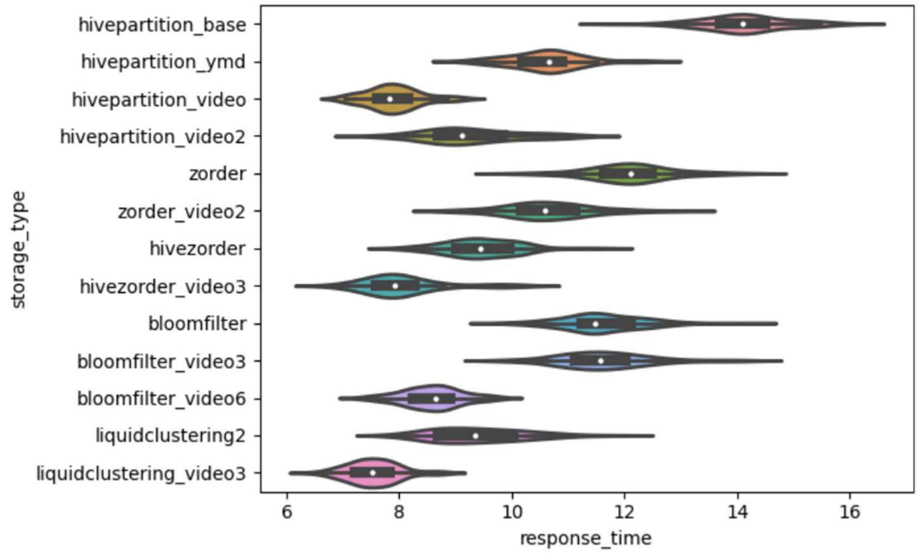


Figure 24: Aggregated Performance Experiment illustrated

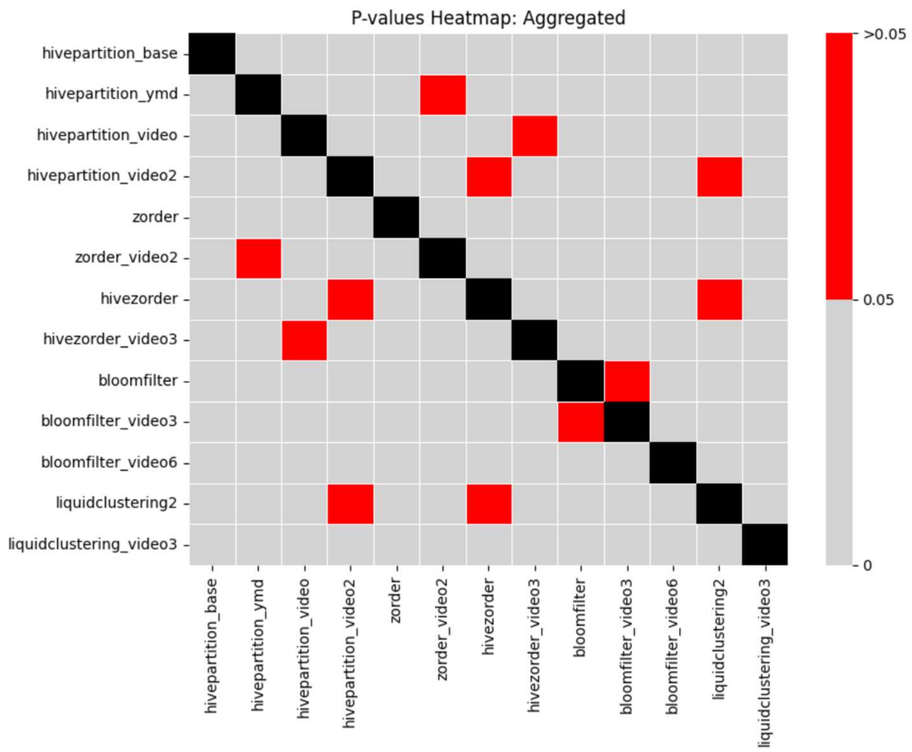


Figure 25: Heatmap of statistical significance between datasets on aggregated experiment

Analysis:

For the Aggregated Performance Experiment results shown in Figure 24 it is noticeable that all data organization techniques except the Z-order datasets have a top performer among their variants.

The Z-order technique seems inferior to others when aggregating larger subsets of data. It is likely that a variation with only date and active videos as its Z-order keys would perform better and maybe on par with the other top performers of each data organization technique, but this would cause challenges for the Individual Performance Experiment as all dataset needs to be optimized for both experiments. By not having UserKey as a Z-Ordered attribute the dataset would have a low Scan Skipped Efficiency, and a high false positive file read as the UserKey values could exist in almost all files.

As expected, we see great performance on the Hive Partitioning datasets which has active video enable either as a partition or with z-order. With the Hive Partitioning Video and Video 2 datasets both skip all files not satisfied by the date period which prune around half the files. Then the last level of partitioning on the datasets are related to active video which for the Video data is the VideoFlag and for Video 2 is VideoInfoCode. The VideoFlag has only two options which are heavily unbalanced as most events does not have any active video. Compared to the VideoInfoCode from Video 2 which has six different active video variants the VideoFlag from the Video dataset is in this case superior.

The Hive Zorder Video 3 dataset performs almost identical to the Hive Partitioning Video dataset. This is because they have the same partitioning levels. The only difference is the files with the last partition level has been Z-ordered by the UserKey attribute. When looking at the heatmap in Figure 25 we see that these two datasets are statistically significant and therefore considered equal in performance which would be expected.

Bloom Filter Video 6 is the highest performer among the Bloom Filter dataset variants. This dataset is using the UserKey as its Bloom Filter key while being Z-ordered by CalendarKey and VideoInfoCode. As we do not filter on UserKey in this experiment the Bloom Filter is not used or enabled for this dataset.

The dataset is solely utilizing the Z-ordered attributes which does not include UserKey like in the Z-ordered datasets which is why this dataset sees better performance.

The other two Bloom Filter datasets both utilize the Bloom Filter key in the experiments but as the Bloom Filter keys are either VideoFlag or VideoInfoCode. Both are low cardinality attributes, and the performances are inferior compared to the Bloom Filter Video 6 dataset.

Liquid Clustering Video 3 is the top performing dataset of this experiment and is significantly different from all other datasets. This dataset compared to Liquid Clustering 2 uses CalendarKey, UserKey and VideoFlag as its cluster keys while the Liquid Clustering 2 dataset does not have any active video attribute enables as a key. This is the reason for the better performance in Liquid Clustering Video 3 as all filtered attributes are activated in by the cluster keys which gives effective file pruning.

Experiment conclusion

The "Hive Partitioning Video", "Hive Z-Order Video 3" and "Liquid Clustering Video 3" datasets are those which has the lowest mean response time and are therefore the best performing datasets. All three datasets are very close around the 7.5 seconds for retrieving active video aggregations on 30 days of data.

5. Related Work

There are not many articles that tries to compare performance of multiple data organization techniques and multiple variants of each technique like this project does. Most articles that I have managed to find have a lack of details in their experiments, does it only on a theoretical level or use dummy data which often are generated to contain only dates and random integers. This does not reflect a realistic representation of how the various data organization techniques will perform in a real environment.

Here are some general subjective takes on why these observations seems to be the case for comparisons of storage techniques:

- Clusters can be expensive for testing
- Duplicating real big data is expensive as well
- Theoretical performance is trusted and generalized

Underneath is a list of the observations made from related articles:

Liquid Clustering: First Impressions (Martins, 2023):

- Martins compares a combination of Hive Partitioning and Z-Ordering with Liquid Clustering. The source code repository is linked to the article
- The article lacks details as it is unknown to the reader whether each performance experiment has been iterated over to simulate a gaussian distribution
- Additionally, Martins uses dummy data consisting only of dates and random integers. This is not a realistic scenario as real data often contains complex datatypes and NULL values
- The queries used against the datasets are partially unknown as only two queries are present in the source code. Both queries return all attributes with a filter on a single clustered or partitioned attribute. Both datasets are limited to return 1.000 rows
- The article showcase two variants of Liquid Clustering datasets as the first dataset has two cluster keys while the other has the same keys but inversed. Databricks has stated that the order of the cluster keys is not relevant. The experiment concludes the same

- The experiments in the article are generally focused on ingesting time and file size and less on reading performances

Delta Lake Z Order (Powers, 2023):

- Powers focuses on Z-Ordering and performs a couple of experiments, first the comparison of using nothing vs. one Z-ordered attribute and later how datasets with one and two Z-ordered attributes can perform on different types of queries
- A theoretical comparison is made of Hive Partitioning versus Z-Ordering which mainly describe the 'small file problem' for the Hive Partitioning technique as partitioning is not suitable for high cardinality attributes or attributes which create partitions which contain less than 1 Gb data.

  This is a general rule of thumb but as seen in this project the Hive Partitioning Video dataset variation had four levels of partitioning which resulted in partitions of around 120 Mb data. Despite this the Hive Partitioning Video dataset variation was one of the best performing datasets across all experiments
- Lastly Powers briefly describe the possibilities of applying the combination of Hive Partitioning and Z-Ordering to a dataset but still only on a theoretical level

Liquid Clustering with Databricks Delta Lake (Siciliani, 2023):

- Siciliani suggests using Hive Partitioning on datasets over 1 Tb and only on low cardinality attributes which create partitions over 1 Gb in size. Additionally, Z-Ordering should only be on high cardinality attributes
- Siciliani states that the Liquid Clustering 'dynamically adjust to changing query patterns' as cluster keys can be changed without rewriting the data, hence the name "liquid"

6. Conclusion

This project has tested the performance of the four storage techniques Hive Partitioning, Z-ordering, Bloom Filters and Liquid Clustering. All experiments were tested on real data from TV 2 Danmarks TV 2 Play tracking data.

The performance was measured primarily in query response times and in skipping efficiency. This was tested on 24 different variations of the four storage techniques with different combinations. 13 of these dataset variations were chosen to be included in the project to showcase the best performers of each technique.

Each data organization techniques dataset variation was tested on three different perspectives, which were chosen to measure the best performing solution for a broader variety of tasks.

The first perspective was measuring the datasets skipping efficiency as this would show how well a dataset could skip files initially for less necessary file readings. This is also defined as the false positive readings and would be divided into individual filter skipping and aggregated filter skipping.

The last two perspectives chosen was a performance experiment on an individual level and on an aggregated level as both are relevant for solving tasks in a data driven environment.

The result of each experiment is outlined below:

Skipping Efficiency Experiment

In this experiment it was discovered that the generated VideoFlag attribute was false for most of the data which would prune a large part of the data when filtering active video events. This made the VideoFlag attribute very suitable as an additional partitioning level. Therefore, we saw great performance on the Hive Partitioning Video data as this dataset could initially skip 83% of the files on both the individual and aggregated perspective.

This was the only real competitor to the Liquid Clustering Video 3 dataset which had even better performance as all filtered attributes were included as a cluster key in the dataset. The Liquid Clustering Video 3 dataset could skip 95% of all files on individual filtering and 85% on aggregated filtering.

Individual Performance Experiment

In this experiment there was a dataset which outperformed the others which was the Liquid Clustering Video 3 dataset. This dataset had the lowest mean response time of 2.7 seconds while the runner up had 3.4 seconds. This performance was achieved by having all filtered attributes as cluster keys in the dataset.

The Hilbert Space-Filling Curve has done a great job at projecting the data from a three-dimensional space into a one-dimensional space while preserving the original datapoints locality. This have resulted in data being distributed into files which has made them very accessible for the most common filters.

Aggregated Performance Experiment

In this experiment there were three datasets which had an edge against the remaining ones. The Liquid Clustering Video 3, Hive Partitioning Video and Hive Z-Order Video 3 datasets had a mean response time near 7.5 seconds for filtering and aggregating on a larger period of the data.

The Hive Partitioning Video and the Hive Z-Order Video 3 datasets performed identically in this experiment as the Z-Order attribute was irrelevant due to not being filtered upon.

Overall conclusion

This project will recommend TV 2 Danmark to use the Liquid Clustering technique on the Curated Micro Data dataset as it has been evaluated to be the best performing technique at finding individuals in big datasets while also being able to quickly aggregate large subsets of data for analytical purposes.

The Hive Partitioning technique showed that if you know your data well enough, it is possible to achieve impressive query response times with levels near the best performing Liquid Clustering variant.

7. Bibliografi

Advancing Analytics - Databricks Execution Plans. (n.d.). Retrieved from Advancing Analytics: https://www.advancinganalytics.co.uk/blog/2021/10/11/databricks-execution-plans

Bloom, B. H. (1970). *Space/time trade-offs in hash coding with allowable errors.* Retrieved from ACM Digital Library: https://dl.acm.org/doi/pdf/10.1145/362686.362692

Chandra, N. (2023, july 7). *Medium - Blog - Z order indexing*. Retrieved from Medium: https://medium.com/@nishant.chandra/z-order-indexing-for-efficient-queries-in-data-lake-48eceaeb2320

Data Flair - Hive Partitions. (n.d.). Retrieved from Data Flair: https://data-flair.training/blogs/apache-hive-partitions/

Data Ninjago - Adaptive Query Execution. (n.d.). Retrieved from Data Ninjago: https://dataninjago.com/2022/02/14/spark-sql-query-engine-deep-dive-19-adaptive-query-execution-part-1/

Databricks - Bloom Filters. (2024, march 15). Retrieved from Databricks: https://docs.databricks.com/en/optimizations/bloom-filters.html

Databricks - Clustering. (2024, july 26). Retrieved from Databricks: https://docs.databricks.com/en/delta/clustering.html

Databricks - Concepts. (2024, august 29). Retrieved from Databricks: https://docs.databricks.com/en/getting-started/concepts.html

Databricks - Data skipping Z-Order. (n.d.). Retrieved from Databricks: https://docs.databricks.com/en/delta/data-skipping.html#delta-zorder

Databricks - Delta Optimize. (n.d.). Retrieved from Databricks: https://docs.databricks.com/en/sql/language-manual/delta-optimize.html

Databricks - Liquid Clustering. (n.d.). Retrieved from Databricks: https://docs.databricks.com/en/delta/clustering.html

Databricks - Medallion Architecture. (n.d.). Retrieved from Databricks: https://www.databricks.com/glossary/medallion-architecture

Databricks - Partitions. (n.d.). Retrieved from Databricks: https://docs.databricks.com/en/tables/partitions.html

Databricks - Spark SQL. (n.d.). Retrieved from Databricks: https://www.databricks.com/glossary/what-is-spark-sql

Hilbert, D. (1891). *Ueber die stetige Abbildung einer Linie auf ein Flächenstück.* Retrieved from The University of Edinburgh: https://www.maths.ed.ac.uk/~v1ranick/papers/hilbert.pdf

Ionescu, A. (2018, july 31). *Databricks - Blog - Skipping Efficiency*. Retrieved from Databricks: https://www.databricks.com/blog/2018/07/31/processing-petabytes-of-data-in-seconds-with-databricks-delta.html

Kumar, H. (2023). *Medium - Bloom Filters*. Retrieved from Medium: https://medium.com/@harsh11csb/bloom-filters-a14ca050119a

LaMorte, W. W. (2016). *Boston University - Central Limit Theorem*. Retrieved from Boston University: https://sphweb.bumc.bu.edu/otlt/mph-modules/bs/bs704_probability/BS704_Probability12.html

Lawder, J. K. (2001, march). *Querying Multi-dimensional Data Indexed Using the Hilbert Space-Filling Curve.* Retrieved from Sigmod Record: https://sigmodrecord.org/publications/sigmodRecord/0103/3.lawder.pdf

Martins, G. (2023, september 23). *Liquid Clustering: FIrst Impressions*. Retrieved from Medium: https://medium.com/closer-consulting/liquid-clustering-first-impressions-113e2517b251

Medium - Liquid Clustering. (n.d.). Retrieved from Medium: https://medium.com/@tsiciliani/liquid-clustering-with-databricks-delta-lake-57dc251d7870

Medium - Spark Logical and Physical Plans. (n.d.). Retrieved from Medium: https://medium.com/datalex/sparks-logical-and-physical-plans-when-why-how-and-beyond-8cd1947b605a

Medium - Z-Ordering from A to Z. (n.d.). Retrieved from Medium: https://yousry.medium.com/delta-lake-z-ordering-from-a-to-z-315063a42031

Microsoft - Data Lake Storage Gen 2. (2024, january 18). Retrieved from Microsoft: https://learn.microsoft.com/en-us/azure/storage/blobs/upgrade-to-data-lake-storage-gen2-how-to?tabs=azure-portal

Mill, B. (n.d.). *Github - Bloom Filter Tutorial*. Retrieved from Github: https://llimllib.github.io/bloomfilter-tutorial/

Morton, G. M. (1966). *A computer Oriented Geodetic Data Base; and a New Technique in File Sequencing.* Ottawa.

MsSQLTips - Bloom Filter Indexes. (n.d.). Retrieved from MsSQLTips: https://www.mssqltips.com/sqlservertip/6968/bloom-filter-indexes-using-databricks-delta/

Osborne, J. W., & Overbay, A. (2004). *The power of outliers (and why r The power of outliers (and why researchers should AL chers should ALWAYS.* Retrieved from Core: https://core.ac.uk/download/pdf/239583894.pdf

Powers, M. (2023, june 3). *Delta Lake - Z-order*. Retrieved from Delta Lake: https://delta.io/blog/2023-06-03-delta-lake-z-order/

Siciliani, T. (2023, july 3). *Medium - Liquid Clustering with Databricks Delta Lake*. Retrieved from Medium: https://medium.com/@tsiciliani/liquid-clustering-with-databricks-delta-lake-57dc251d7870

Spark - Partition Discovery. (n.d.). Retrieved from Spark Apache: https://spark.apache.org/docs/latest/sql-data-sources-parquet.html#partition-discovery

Vellaturi, R. (2023, september 18). *Medium - Blog - Big Data file format*. Retrieved from Medium: https://blog.det.life/choosing-the-right-big-data-file-format-avro-vs-parquet-vs-orc-c868ffbe5a4e

Wikipedia - Z Order Curve. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Z-order_curve

Wirth, N. (1976, February). *Algorithms + Data Structures = Programs.* Retrieved from CL72: https://www.cl72.org/110dataAlgo/Algorithms%20%20%20Data%20Structures%20=%20Programs%20%5BWirth%201976-02%5D.pdf

Appendix

The appendix can be viewed from the attached Notebook.zip file which contains the following files in both .py and .html extension:

- 001-Configurations
- 002-Data Migration
- 003-Performance Tests
- 003-Skipping Efficiency Test
- 004-Test Results