databricks

Designing the Foundation

LECTURE

Introduction to Designing Foundation

In this lecture, we will cover fundamental concepts of designing a foundation, common performance bottlenecks, and strategies to avoid the small file problem.

Fundamental Concepts

Why some schemas and queries perform faster than others

- Number of bytes read
- Query complexity/computation
- Number of files accessed
- Parallelism

Number of Bytes ReadThe more data you have to read to satisfy a query, the longer the query takes. The data is getting pulled from a disk, transferred over the network, etc. So if you need to access a lot of data to satisfy a query, it will likely take some time or horsepower.

Query Complexity / Computation

The more complex the query or computation required, the longer the query takes. This may include aggregations, joins, etc.

Number of Files Accessed

The more files a query has to access, the slower the query. Each file access brings some overhead, and if the database has to access many files it may spend more time with the overhead of opening and closing files than actually processing the data. This principle is specific to databases that store their data across multiple files, like Databricks, but other databases will often have an equivalent (file system blocks, etc).

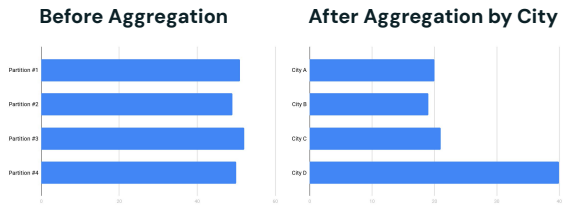
Parallelism

In MPP (Massively Parallel Processing) systems, such as Databricks, the ability to run computation in parallel is important to decreasing the clock time of a query.

Common Performance Bottlenecks

Encountered with any big data or MPP system

Bottleneck	Details
Small File Problem	• Listing and metadata operation for too many small files can be expensive • Can also result in throttling from cloud storage I/O limits
Data Skew	• Large amounts of data skew can result in more work handled by a single executor • Even if data read in is not skewed, certain transformations can lead to in-memory skew
Processing More Than Needed	• Traditional data lake platforms often require rewriting entire datasets or partitions



Performance bottlenecks can occur for several reasons. One is the **small file problem**—when data is spread across many tiny files, opening and moving them across the network slows queries and can even trigger cloud provider I/O throttling.

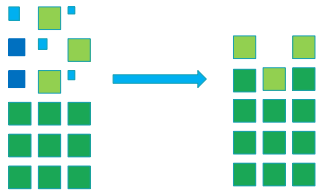
Another is **data skew**, where one partition is much larger than others or where transformations create imbalance. Since processing waits for the slowest executor, a single large partition can delay the entire job.

Bottlenecks also happen when **processing more data than needed**. Unlike traditional data lakes that often rewrite entire datasets, we can process only the required files and use techniques like **data skipping** to further improve performance.

Avoiding the Small File Problem

Automatically handle this common performance challenge in Data Lakes

- Too many small files greatly increases overhead for reads
- Too few large files reduces parallelism on reads
- Over-partitioning is a common problem
- Databricks will automatically tune the size of Delta Lake tables
- Databricks will automatically compact small files on write with auto-optimize



When working with large quantities of data that are file based, we run into the so-called “small file problem.” With numerous small files, performance degrades. Addressing the challenges posed by the 'small file problem' is pivotal for maintaining optimal performance in data processing. Our solution, Auto Optimize, has been designed to alleviate these issues through two essential features:

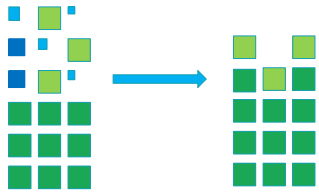
Optimize Write:

- This component operates dynamically within the same Spark job, adjusting Apache Spark partition sizes based on the actual data. The goal is to generate 128MB files for each table partition. This approach not only tackles the 'small file problem' but also optimizes the distribution of data within the Spark job.

Avoiding the Small File Problem

Automatically handle this common performance challenge in Data Lakes

- Too many small files greatly increases overhead for reads
- Too few large files reduces parallelism on reads
- Over-partitioning is a common problem
- Databricks will automatically tune the size of Delta Lake tables
- Databricks will automatically compact small files on write with auto-optimize



Auto Compact:

- Upon the completion of the Spark job, Auto Compact takes the optimization a step further. Launching a new job, it assesses whether additional compression can be applied to the files, with the ultimate aim of achieving a standardized 128MB file size. This post-processing step ensures that the data remains efficiently organized and compact.

Additionally, the ability to split files into partitions adds a layer of flexibility to our data management strategy. While partitioning is a powerful tool for parallelism and efficient processing, it's imperative to strike a balance. Over-partitioning can introduce complexities and challenges, emphasizing the need for careful consideration when implementing data partitioning strategies.By incorporating Auto Optimize and intelligently managing file partitions, our approach aims to provide a robust solution for handling large-scale file-based data, optimizing performance, and mitigating the adverse effects of the 'small file problem.

Thank you for completing this lesson and continuing your journey to develop your skills with us.