Apache Iceberg Architecture: Managing 100 Million Records (2019-2024)

Scenario Overview

You are ingesting 100 million records spanning 2019 to 2024. These records are partitioned by month, with an even distribution of **1.6 million records per month** across **60 months**. Apache Iceberg provides efficient data storage, metadata management, and querying capabilities, ensuring scalability and high query performance for such datasets.

Considerations

Data Partitioning

1. **Partitioning Mechanism**:
    - Iceberg allows partitioning using transform functions like `month()` on a timestamp column.
    - With **hidden partitioning**, there is no need to explicitly create a column for partitioning. Iceberg dynamically applies the transformation at write time and stores the partition metadata efficiently in its metadata files.
2. **Partition Structure**:
    - The dataset is partitioned into **60 partitions**, one for each month from 2019 to 2024.
    - Each partition contains **1.6 million records**, ensuring even distribution of data.
3. **Partition Benefits**:
    - Queries that filter by `month` can directly leverage the metadata to scan only relevant partitions, drastically reducing query execution time and resource usage.

Step 1: Writing Data Files in Iceberg

Reading and Partitioning Data

1. The data for **100 million records** is ingested from the source (e.g., database, file system, or streaming input).
2. The data spans **5 years (2019-2024)**, partitioned into months using a transformation function like `month(timestamp_column)`.

Iceberg dynamically creates **60 partitions** (one per month) without requiring an explicit column for partitioning:
month=2019-01/
month=2019-02/
...

3. month=2024-12/

Writing Data Files by Partition

1. Each month's data consists of **1.6 million records**, evenly distributed.
2. Iceberg writes the data into **Parquet files**, a columnar storage format optimized for analytical workloads.
3. Each Parquet file can hold **100,000 rows**, so Iceberg splits the 1.6 million records per month into:
   1.6 million ÷ 100,000 = 16 Parquet files per month.
4. These Parquet files are written to the underlying storage system (e.g., ADLS Gen2, AWS S3, HDFS).

Folder and File Structure

Iceberg organizes the data into folders and files under the table's root directory:

```
root/
 month=2019-01/
  file1.parquet
  file2.parquet
  ...
  file16.parquet
 month=2019-02/
  file1.parquet
  ...
 ...
 month=2024-12/
```

- file1.parquet

- Each folder corresponds to a partition, and the files within the folder represent the actual data for that partition.

Total Number of Data Files

With **16 Parquet files per month** across **60 months**, the total number of data files is:

16 files/month × 60 months = 960 Parquet files.

These 960 files represent the entire dataset.

Step 2: Manifest Files

What Happens

1. After writing the data files, Iceberg creates **manifest files** to track metadata for the data files.
2. Each manifest file contains metadata about a group of data files, grouping files logically based on partitions or other criteria.

What Manifest Files Contain

Manifest files provide a detailed index for data files. For each data file, a manifest file contains:

- **File Path**: Location of the data file (e.g., `/dbfs/iceberg/sales/customer_table/file1.parquet`).
- **Row Count**: Number of rows in the data file (e.g., 100,000).
- **Bounds**: Minimum (`lower_bound`) and maximum (`upper_bound`) values of the columns to enable query pruning.
- **Null Counts**: Number of null values in each column.
- **Partition Information**: The partition value for the file (e.g., `month=2019-01`).
- **File Size**: Size of the data file in bytes, used for query optimization.
- **Metrics**: Additional statistics such as average row size, compression type, and column-level summaries.

Example Manifest File Entry

```
{
  "file_path": "/dbfs/iceberg/sales/customer_table/1gdj9fhf.parquet",
  "row_count": 100000,
  "null_count": 100,
  "lower_bound": "2019-01-01",
  "upper_bound": "2019-01-10",
  "partition": "2019-01",
  "file_size": 2048000
}
```

How Manifest Files Work

- **Metadata for Data Files**:
  - A manifest file indexes a set of data files, providing a quick reference for their location and statistics.
  - For instance, if one partition (e.g., `month=2019-01`) has 16 data files, one or more manifest files will hold metadata for all these files.
- **Query Planning**:
  - When executing a query, Iceberg scans manifest files to identify which data files are relevant to the query, skipping unnecessary files.
- **Scalability**:
  - Instead of storing metadata for all data files in a single location, Iceberg splits metadata across multiple manifest files, enabling efficient metadata management.

Why Manifest Files Matter

- Manifest files act as **an index for data files**, reducing the need to scan all files for query execution.
- They store column-level statistics, enabling fine-grained query pruning and efficient query planning.

Step 3: Manifest List (Snapshot File)

What Happens

1. After creating manifest files, Iceberg generates a **manifest list**, which serves as the **snapshot file**.
2. The manifest list contains high-level metadata about all manifest files.

What the Manifest List Contains

Each entry in the manifest list corresponds to a manifest file and includes:

- **Manifest File Path**: Location of the manifest file (e.g., `/FileStore/Iceberg/sales/customer_table/metadata/manifest1.avro`).
- **Manifest File Length**: Size of the manifest file in bytes.
- **Partition Spec ID**: Identifier for the partitioning scheme used.
- **Content Type**: Indicates whether the manifest file tracks data files or delete files.
- **Snapshot ID**: The unique identifier for the snapshot the manifest file belongs to.
- **File Counts**:

    - **Added Files**: Data files added in the current snapshot.
    - **Existing Files**: Data files carried over from previous snapshots.
    - **Deleted Files**: Data files removed during the current snapshot.
- **Row Counts**:
    - Tracks rows added, existing, or deleted.

Example Manifest List Entry

```
{
  "manifest_file_path": "/FileStore/Iceberg/sales/customer_table/metadata/manifest1.avro",
  "manifest_file_length": 10240,
  "partition_spec_id": 1,
  "content": "data",
  "snapshot_id": 1628785150952769302,
  "file_counts": { "added": 16, "existing": 0, "deleted": 0 },
  "row_counts": { "added": 1600000, "existing": 0, "deleted": 0 }
}
```

How the Manifest List Acts as a Snapshot

1. **Snapshot Representation**:
    - A manifest list represents the state of the table at a specific point in time.
    - It references all manifest files that are valid for the snapshot, indirectly pointing to the data files they track.
2. **Time Travel**:
    - Since every snapshot has its own manifest list, Iceberg enables time travel by querying specific snapshots.
    - Example Time Travel Query:
      SELECT * FROM customer_table VERSION AS OF 1628785150952769302;
    - This retrieves data as it existed at the time of the snapshot.
3. **Query Efficiency**:
    - By referencing only the relevant manifest files, the manifest list ensures that queries scan minimal metadata and access only the required data files.
4. **Atomicity and Isolation**:
    - Manifest lists ensure consistent snapshots, enabling atomic operations. If a write fails, the table remains in its previous state

Step 4: Metadata File

What Happens

Iceberg creates a **metadata file** (`metadata.json`) that serves as the central reference for table management.

What the Metadata File Contains

1. **Table Schema**:
    - Column names, types, and unique column IDs.
    - Enables schema evolution by maintaining column versions and changes.
2. **Partitioning Information**:
    - Specifies the partitioning scheme (e.g., `month`).
    - Tracks the `partition_spec_id` used for partitioning the table.
3. **Current Snapshot ID**:
    - Points to the latest snapshot for the table.
    - This allows Iceberg to know the most recent state of the table.
4. **Snapshot History**:
    - Maintains a log of all snapshots (e.g., append, overwrite, delete).
    - Enables time travel and version history of the table.
5. **Other Properties**:
    - **Compression Codec**: Indicates the codec used for data files (e.g., `zstd` for Parquet).
    - **Table Location**: Physical storage path for the table's data and metadata.
    - **Total File Counts and Sizes**: Tracks the total number of files and their cumulative size.

Example Metadata File Entry

```
{
  "format-version": 2,
  "table-uuid": "c62fbe64-8832-4e62-85c7-69c26d64f7ab",
  "location": "/FileStore/Iceberg/sales/customer_table",
  "last-sequence-number": 2,
  "current-snapshot-id": 1628785150952769302,
  "schemas": [
    {
      "id": 0,
      "fields": [
        { "name": "month", "type": "string", "id": 1 },
        { "name": "sales", "type": "double", "id": 2 }
      ]
    }
  ],
  "partition-specs": [
    { "spec-id": 0, "fields": [ { "name": "month", "transform": "identity" } ] }
  ],
  "snapshots": [
    {
      "snapshot-id": 1628785150952769302,
      "timestamp": "2023-01-25T10:00:00Z",
      "operation": "append",
      "added-files": 16,
      "added-records": 1600000
    }
  ]
}
```

How Metadata Files Work

1. **Centralized Management**:
   - The metadata file acts as the single source of truth for the table's schema, partitioning, and snapshot history.
2. **Consistency Across Queries**:
   - Query engines use the metadata file to ensure they operate on a consistent view of the table, even in a multi-user environment.
3. **Schema Evolution**:
   - Supports adding, removing, or updating columns without rewriting the entire table, ensuring backward compatibility with older snapshots.
4. **Efficient Query Planning**:
   - Metadata files reduce query planning overhead by storing information about snapshots and partitioning schemes.
5. **Integration with Catalogs**:
   - The metadata file is registered in the table catalog, allowing query engines (e.g., Spark, Flink) to quickly locate and access the table.

Step 5: Table Catalog

What Happens

1. The table is registered in a catalog (e.g., Hive Metastore, AWS Glue, or Hadoop catalog).
2. The catalog entry stores the path to the **latest metadata file**.

What the Catalog Contains

- **Table Name**: A unique name for the table (e.g., `sales.customer_table`).
- **Metadata File Path**: Points to the latest metadata file for the table.
- **Namespace**: The logical grouping of the table within the catalog (e.g., `sales` namespace).

Example Catalog Entry

```
{
  "table_name": "sales.customer_table",
  "metadata_file": "/FileStore/Iceberg/sales/customer_table/metadata/v1.metadata.json"
}
```

Why the Catalog Matters

1. **Centralized Table Management**:
   - The catalog serves as a global registry for all Iceberg tables, allowing query engines to discover and interact with them.
2. **Seamless Querying**:
   - Query engines use the catalog to locate the latest metadata file, ensuring they always query the most recent table state.
3. **Multi-Tenant Support**:
   - Catalogs allow logical separation of tables using namespaces, enabling multi-tenant use cases.

Step 6: Query Planning and Execution

What Happens

When you run a query, Iceberg processes it in the following steps:

1. **Metadata File Lookup**:
   - The query engine reads the metadata file to locate the current snapshot.
2. **Snapshot Resolution**:
   - The snapshot ID in the metadata file points to the manifest list for the latest state of the table.
3. **Manifest List Scanning**:
   - The manifest list provides a list of manifest files to be scanned based on the query filters.
4. **Manifest File Scanning**:
   - Relevant manifest files are scanned to identify which data files match the query criteria.
5. **Data File Access**:
   - Only the required data files are read to execute the query.

Why Query Planning is Efficient

1. **Metadata-Driven Pruning**:
   - Iceberg's metadata structure allows the query engine to prune irrelevant data files without scanning the entire dataset.
2. **Hidden Partitioning**:
   - Users do not need to explicitly specify partitioning columns, simplifying query writing while still leveraging partition-based pruning.
3. **Time Travel**:
   - Iceberg enables querying historical data by referencing specific snapshots or timestamps:
     SELECT * FROM customer_table VERSION AS OF 1628785150952769302;
4. **Column-Level Statistics**:
   - Manifest files store statistics like min/max values and null counts, enabling predicate pushdowns and further reducing I/O.

USE CASE 2 : WHAT HAPPENS WHEN YOU INSERT MORE DATA:

Insert Operations and Data Files

What Happens

1. **New Data Files Are Created**:
   - When you perform an insert operation, new rows are written into **new data files**. Existing data files remain untouched because Iceberg treats data files as immutable.
   - These new data files are stored in the table's underlying storage (e.g., ADLS Gen2, AWS S3, HDFS) according to the partitioning scheme of the table.
2. **File Organization**:
   - If the table is partitioned, the new data files are written to directories corresponding to the partition values. For example, data for January 2023 might be stored in: /path/to/table/month=2023-01/datafile1.parquet
   - In object storage, Iceberg may hash file paths to distribute the data files evenly across directories.
3. **Efficient Storage**:
   - The size of the new data files is governed by Iceberg's configuration, ensuring that each file is optimized for analytical workloads (e.g., `128 MB` by default).

Summary

Insert operations result in the creation of **new data files** for the inserted rows, while existing data files remain unmodified.

Manifest Files

What Happens

1. **New Manifest Files Are Created**:
    - After the new data files are written, Iceberg creates **new manifest files** to track them. Manifest files index a subset of data files and include metadata for efficient query planning.
2. **Content of Manifest Files**:
    - Each manifest file contains:
        - **File Path**: The location of the data file.
        - **Row Count**: The number of rows in the data file.
        - **Partition Information**: The partition key and value for the data file (e.g., `month=2023-01`).
        - **Column Statistics**: Minimum and maximum values for each column, null counts, etc.
        - **File Size**: The size of the data file in bytes.
3. **Immutable Design**:
    - Manifest files are immutable. If new data is added, Iceberg writes new manifest files instead of modifying existing ones.

Example Manifest File Entry

```
{
  "file_path": "/path/to/table/month=2023-01/datafile1.parquet",
  "row_count": 500000,
  "partition": "2023-01",
  "null_count": 0,
  "lower_bound": "2023-01-01",
  "upper_bound": "2023-01-31",
  "file_size": 134217728
}
```

Summary

New manifest files are created to track the newly added data files, while existing manifest files remain unchanged.

Manifest List (Snapshots)

What Happens

1. **New Snapshot Creation**:
    - Iceberg creates a new **snapshot** of the table after the insert operation.
    - A **manifest list** (snapshot file) is generated to track all manifest files associated with the current snapshot, including both new and existing manifest files.
2. **Content of the Manifest List**:
    - The manifest list includes metadata about all manifest files in the snapshot, such as:
        - **Manifest File Path**: Location of the manifest file.
        - **File Counts**: The number of files added, existing, or deleted in the snapshot.
        - **Row Counts**: The number of rows added, existing, or deleted in the snapshot.
        - **Partition Statistics**: Upper and lower bounds for partitions covered by the manifest file.
3. **Immutable Manifest Lists**:
    - Like data and manifest files, manifest lists are also immutable. Each snapshot gets its own manifest list, preserving the table's historical state.

Example Manifest List Entry

```
{
  "manifest_file_path": "/path/to/table/metadata/manifest1.avro",
  "file_counts": {
    "added": 10,
    "existing": 50,
    "deleted": 0
  },
  "row_counts": {
    "added": 1000000,
    "existing": 50000000,
    "deleted": 0
  },
  "snapshot_id": 567890123456789,
  "partition_spec_id": 1
}
```

Key Characteristics

1. **Tracks New and Existing Manifest Files**:
   - The manifest list references new manifest files created during the insert operation and retains old manifest files for data that remains valid.
2. **Supports Time Travel**:
   - Each snapshot is associated with a manifest list, allowing queries to access the table as it existed at any point in time.

Summary

Each insert operation creates a new manifest list (snapshot file) that points to all manifest files (new and old) relevant to the current state of the table.

Metadata Changes

What Happens

1. **New Metadata File**:
   - Iceberg creates a new **metadata file** for every table change, including insert operations.
   - The metadata file serves as the central repository for high-level information about the table.
2. **Content of the Metadata File**:
   - **Current Snapshot ID**: Points to the latest snapshot (manifest list).
   - **Snapshot History**: Tracks all previous snapshots, enabling time travel.
   - **Table Schema**: Includes column names, types, and unique column IDs.
   - **Partitioning Information**: Specifies the partition scheme.
   - **Previous Metadata Files**: Tracks the lineage of metadata files.
3. **Immutable Design**:
   - Metadata files are never modified. Instead, a new metadata file is created for every table update.

Example Metadata File Entry

```
{
  "format-version": 2,
  "current-snapshot-id": 567890123456789,
  "schemas": [
    {
      "id": 1,
      "fields": [
        { "name": "month", "type": "string", "id": 1 },
        { "name": "sales", "type": "double", "id": 2 }
      ]
    }
  ],
  "snapshots": [
    {
      "snapshot-id": 567890123456789,
      "timestamp": "2025-01-24T10:00:00Z",
      "operation": "append",
      "added-files": 10,
      "added-records": 1000000
    }
  ]
}
```

Summary

A new metadata file is created for every insert operation, capturing the table's latest state and maintaining a complete history.

Catalog Update

What Happens

1. **Catalog Update**:
   - After the insert operation completes, the catalog is updated to point to the new metadata file.
2. **Atomicity**:
   - The catalog ensures atomic updates, preventing inconsistencies when multiple writers access the table.
3. **Content of the Catalog**:
   - **Table Name**: A unique name for the table.
   - **Metadata File Path**: The location of the latest metadata file.

Example Catalog Entry

```
{
  "table_name": "sales.customer_table",
  "metadata_file": "/path/to/table/metadata/v2.metadata.json"
}
```

Summary

The catalog is updated atomically to reflect the latest metadata file, ensuring consistency and reliability.

Overall Summary of the Insert Process

1. **Data Files**: New data files are created for the inserted rows.
2. **Manifest Files**: New manifest files track the newly created data files.
3. **Manifest List (Snapshot)**: A new manifest list references both the new and existing manifest files.
4. **Metadata File**: A new metadata file captures the latest table state and maintains the history of previous snapshots.
5. **Catalog Update**: The catalog is updated atomically to point to the new metadata file.

By using this layered metadata approach, Iceberg ensures efficient and reliable table management, enabling robust query performance and time travel capabilities. Let me know if further refinements or examples are needed!