Iceberg

A modern table format for big data

Ryan Blue @6d352b5d3028e4b
Owen O’Malley @owen_omalley
September 2018 - Strata NY

NETFLIX

HORTONWORKS®

Contents

Iceberg Performance

September 2018 - Strata NY

NETFLIX

HORTONWORKS®

Case Study: Netflix Atlas

- Historical Atlas data:
  - Time-series metrics from Netflix runtime systems
  - 1 month: 2.7 million files in 2,688 partitions
  - Problem: cannot process more than a few days of data

- Sample query:

```
select distinct tags['type'] as type
from iceberg.atlas
where
  name = 'metric-name' and
  date > 20180222 and date <= 20180228
order by type;
```

Atlas Historical Queries

- Hive table – with Parquet filters:
  - 400k+ splits, not combined
  - EXPLAIN query: **9.6 min** (planning wall time)

- Iceberg table – partition data filtering:
  - 15,218 splits, combined
  - **13 min** (wall time) / 61.5 hr (task time) / 10 sec (planning)

- Iceberg table – partition and min/max filtering:
  - 412 splits
  - **42 sec** (wall time) / 22 min (task time) / 25 sec (planning)

What is a table format?

September 2018 - Strata NY

You meant *file format*, right?

No. Table Format.

- **How to track what files store the table's data.**
  - Files in the table are in Avro, Parquet, ORC, etc.

- Often overlooked, but determines:
  - What guarantees are possible (like correctness)
  - How hard it is to write fast queries
  - How the table can change over time
  - Job performance

What is a good table format?

- Should be **specified**: must be documented and portable
- Should support expected database table behavior:
  - **Atomic changes** that commit all rows or nothing
  - **Schema evolution** without unintended consequences
  - **Efficient access** like predicate or projection pushdown
- Bonus features:
  - **Hidden layout**: no need to know the table structure
  - **Layout evolution**: change the table structure over time

Hive Tables

September 2018 - Strata NY

Hive Table Design

- Key idea: **organize data in a directory tree**
  - Partition columns become a directory level with values

```
date=20180513/
 |- hour=18/
 |    |- ...
 |- hour=19/
 |    |- 000000_0
 |    |- ...
 |    |- 000031_0
 |- hour=20/
 |    |- ...
 |- ...
```

Hive Table Design

- Filter by directories as columns

  `SELECT ... WHERE date = '20180513' AND hour = 19`

```
date=20180513/
 |- hour=18/
 |    |- ...
 |- hour=19/
 |    |- 000000_0
 |    |- ...
 |    |- 000031_0
 |- hour=20/
 |    |- ...
 |- ...
```

Hive Metastore

- HMS keeps metadata in SQL database
  - Tracks information about partitions
  - Tracks schema information
  - Tracks table statistics
- Allows filtering by partition values
  - Filters only pushed to DB for string types
- Uses external SQL database
  - Metastore is often the bottleneck for query planning
- Only file system tracks the files in each partition…
  - No per-file statistics

Hive ACID layout

- Provides snapshot isolation and atomic updates
- Transaction state is stored in the metastore
- Uses the same partition/directory layout
  - Creates new directory structure inside partitions

```
date=20180513/
 |- hour=19/
 |    |- base_0000000/
 |    |    |- bucket_00000
 |    |    |- ...
 |    |    |- bucket_00031
 |    |- delta_0000001_0000100/
 |    |    |- bucket_00000
 |    |    |- ...
```

Design Problems

- Table state is stored in two places
  - Partitions in the Hive Metastore
  - Files in a file system
- Bucketing is defined by Hive's (Java) hash implementation.
- Non-ACID layout's only atomic operation is add partition
- **Requires atomic move of objects in file system**
- Still requires directory listing to plan jobs
  - **O(n) listing calls, n = # matching partitions**
  - **Eventual consistency breaks correctness**

Less Obvious Problems

- Partition values are stored as strings
  - Requires character escaping
  - `null` stored as `__HIVE_DEFAULT_PARTITION__`
- HMS table statistics become stale
  - Statistics have to be regenerated manually
- A lot of undocumented layout variants
- Bucket definition tied to Java and Hive

Other Annoyances

- Users must know and use a table's physical layout
  - `ts > X` ⇒ full table scan!
  - Did you mean this?
    `ts > X and (d > day(X) or (d = day(X) and hr >= hour(X))`
- Schema evolution rules are dependent on file format
  - CSV – by position; Avro & ORC – by name
- Unreliable: type support varies across formats
  - Which formats support decimal?
  - Does CSV support maps with struct keys?

Iceberg Tables

September 2018 - Strata NY

NETFLIX

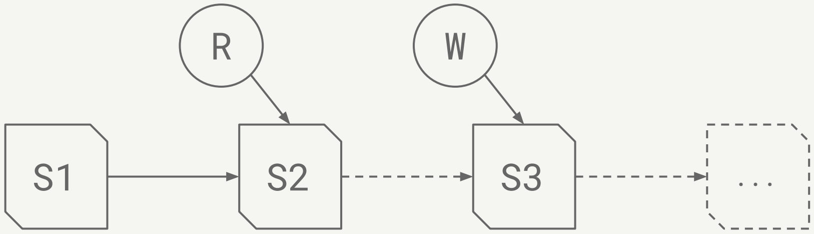
HORTONWORKS®

Iceberg's Design

- Key idea: **track all files in a table** over time
  - A **snapshot** is a complete list of files in a table
  - Each write produces and commits a new snapshot

Snapshot Design Benefits

- Snapshot isolation without locking
  - Readers use a current snapshot
  - Writers produce new snapshots in isolation, then commit

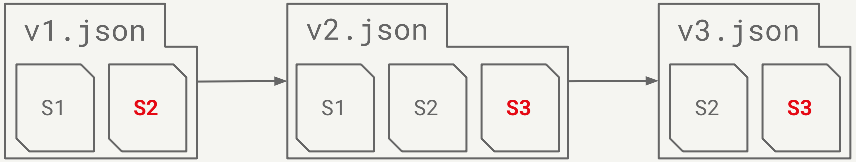


- Any change to the file list is an atomic operation
  - Append data across partitions
  - Merge or rewrite files

In reality, it’s a bit more complicated…

Iceberg Metadata

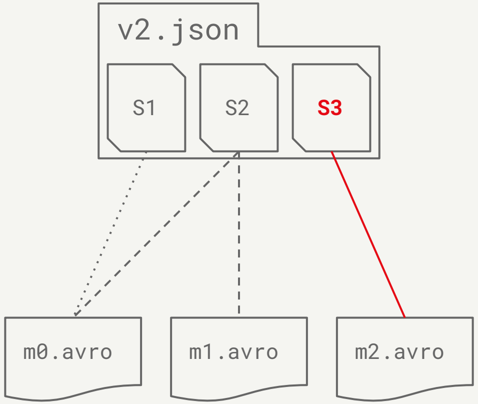
- Implements snapshot-based tracking
  - Adds table schema, partition layout, string properties
  - Tracks old snapshots for eventual garbage collection



- Each metadata file is immutable
- Metadata always moves forward, history is linear
- The current snapshot (pointer) can be rolled back

Manifest Files

- Snapshots are split across one or more **manifest files**
  - A manifest stores files across many partitions
  - A partition data tuple is stored for each data file
  - Reused to avoid high write volume



Manifest File Contents

- Basic data file info:
  - File location and format
  - Iceberg tracking data
- Values to filter files for a scan:
  - Partition data values
  - Per-column lower and upper bounds
- Metrics for cost-based optimization:
  - File-level: row count, size
  - Column-level: value count, null count, size

Commits

- To commit, a writer must:
  - Note the current metadata version – the base version
  - Create new metadata and manifest files
  - Atomically swap the base version for the new version
- This atomic swap ensures a linear history
- Atomic swap is implemented by:
  - A custom metastore implementation
  - Atomic rename for HDFS or local tables

Commits: Conflict Resolution

- Writers *optimistically* write new versions:
  - Assume that no other writer is operating
  - On conflict, retry based on the latest metadata

- To support retry, operations are structured as:
  - **Assumptions** about the current table state
  - **Pending changes** to the current table state

- Changes are safe if the assumptions are all true

Commits: Resolution Example

- Use case: safely merge small files
  - Merge input: `file1.avro`, `file2.avro`
  - Merge output: `merge1.parquet`

- Rewrite operation:
  - **Assumption**: `file1.avro` and `file2.avro` are still present
  - **Pending changes**:

    Remove `file1.avro` and `file2.avro`

    Add `merge1.parquet`

- Deleting `file1.avro` or `file2.avro` will cause a commit failure

Design Benefits

- Reads and writes are isolated and all changes are atomic

- No expensive or eventually-consistent FS operations:
  - No directory or prefix listing
  - No rename: data files written in place

- Faster scan planning
  - `O(1)` manifest reads, not `O(n)` partition list calls
  - Without listing, partition granularity can be higher
  - Upper and lower bounds used to eliminate files

Other Improvements

- Full schema evolution: add, drop, rename, reorder columns

- Reliable support for types
  - date, time, timestamp, and decimal
  - struct, list, map, and mixed nesting

- Hidden partitioning
  - Partition filters derived from data filters
  - Supports evolving table partitioning

- Mixed file format support, reliable CBO metrics, etc.

Contributions to other projects

- Spark improvements
  - Standard logical plans and behavior
  - Data source v2 API revisions

- ORC improvements
  - Added additional statistics
  - Adding timestamp with local timezone

- Parquet & Avro improvements
  - Column resolution by ID
  - New materialization API

Getting Started with Iceberg

September 2018 - Strata NY

Using Iceberg

- github.com/Netflix/iceberg
  - Apache Licensed, ALv2
  - Core Java library available from JitPack
  - Contribute with github issues and pull requests
- Supported engines:
  - Spark 2.3.x - data source v2 plug-in
  - Read-only Pig support
- Mailing list:
  - iceberg-devel@googlegroups.com

Future work

- Hive Metastore catalog (PR available)
  - Uses table locking to implement atomic commits
- Python library coming soon
- Presto support PR coming soon

Questions?

blue@apache.org
omalley@apache.org

September 2018 - Strata NY