Extend Snapshot Reference Lifecycle

Author: Szehon Ho

Notes: Due to potential confusion, this document uses 'purge' to mean removing a file from disk, and 'delete' to mean logically detaching a file from the table.

Motivation

Currently, a Snapshot's lifecycle is handled by 'ExpireSnapshots(long olderThan)'. This operation does the following:

- Choose a set of Snapshots to expire based on timestamp
- Remove these Snapshots references from TableMetadata
- Purge metadata files of these Snapshots
- Purge data deleted by these Snapshots.

Purging expired Snapshot's deleted data often requires a smaller timeline, due to strict requirements to claw back unused disk space, fulfill data lifecycle compliance, etc. In many deployments, this means an 'olderThan' timestamp that is set to just a few days before the current time (the default is 5 days).

On the other hand, removing expired Snapshot references from TableMetadata may be ideally done on a more relaxed timeline, such as months or more, to allow for meaningful historical table analysis.

But today, these are all purged together, and we cannot preserve just the Snapshot references for a longer period than that which is required for purging deleted expired Snapshot data.

Use Case 1 (Analyzing Files/Partitions Added or Removed Beyond a Few Days)

Data files are always associated with a snapshot_id that added them. Take for example DataFile A, which has been added many months ago by Snapshot A, and never rewritten or removed. If we always expire snapshots within a few days, then we will quickly lose contextual information about Snapshot A, even though DataFile A is part of the table.

More technically speaking, the manifest entry that refers to DataFile A has its 'adding' snapshot pointer broken, in particular the snapshot_id and sequence_number fields, as that snapshot no longer exists in TableMetadata.

We thus can no longer answer questions on data files and partitions beyond the olderThan timestamp (ie, a few days), such as:

```
-- when a data file is added
SELECT s.committed_at
FROM table.snapshots s JOIN table.entries
ON s.snapshot_id = e.snapshot_id
WHERE e.data_file.file_path='/my/path.parquet'

-- engine job that added a data file
SELECT s.summary['spark.app.id']
FROM table.snapshots s JOIN table.entries
ON s.snapshot_id = e.snapshot_id
WHERE e.data_file.file_path='/my/path.parquet'

-- when the partition is added (using entries table)
SELECT min(s.committted_at), f.partition
FROM table.snapshots s JOIN table.entries e
ON s.snapshot_id = e.snapshot_id
GROUP BY e.data_file.partition
WHERE e.data_file.partition.day = date('2024-07-15')

-- when the partition is removed (using snapshot summaries)
SELECT snapshot_id FROM iceberg.szho.test.snapshots
WHERE summary['partitions.day=2024-07-15'] IS NOT NULL
ORDER BY committed_at DESC LIMIT 1
```

Use Case 2 (Analyzing History of Table Beyond a Few Days)

A user may also want to track general table history for more general trends. But with an aggressive expire olderThan timestamp, we cannot answer following questions beyond a few days, like:

```
-- how many commits in time X
SELECT count(*)
FROM table.snapshots
WHERE committed_at BETWEEN date('2024-07-01') and date('2024-07-31')

-- how many files were added in time X
SELECT sum(summary['added-data-files'])
FROM table.snapshots
```

```
WHERE committed_at BETWEEN date('2024-07-01') and date('2024-07-31')

-- how many records were removed in time X
SELECT sum(summary['deleted-records'])
FROM table.snapshots
WHERE committed_at BETWEEN date('2024-07-01') and date('2024-07-31')

-- how many bytes were added in time X
SELECT sum(summary['added-files-size'])
FROM table.snapshots
WHERE committed_at BETWEEN date('2024-07-01') and date('2024-07-31')
```

Design

This proposal maintains an optional rolling log file of Expired Snapshots referenceable from TableMetadata, to preserve information of the expired snapshots after they are purged.

Table Metadata
Add an optional pointer field to the TableMetadata object.

V1	V2	Field name	Type	Description
optional	optional	expired-snapshots-path	string	Path to a file with references to expired snapshots

This is a pointer to a 'expired-snapshots-history' file, which will be a serialized JSON list of Snapshot objects that have been expired.

ExpireSnapshots Implementation

We add an additional method to ExpireSnapshots (Core, Spark) implementations.

```
/**
 * Removes all expired snapshots references older than the given timestamp.
 */
ExpireSnapshot removeExpiredRefsOlderThan(long oldestRefTimestamp)
```

If not called, the normal logic is run (all references to expired Snapshots are removed from TableMetadata, (all deleted data/metadata of expired snapshots are purged)

If set, we perform the following additional logic to save references to expired snapshots before purging all deleted data/metadata of expired snapshots:

```
// current logic (simplified for illustrative purposes)
List<Snapshot> toExpire = tableMetadata.snapshots().filter(s ->
s.timestampMillis() < olderThan);
tableMetadata.snapshots().removeAll(toExpire);

if (preserveExpired) {
  List<Snapshot> expiredHistory =
read(tableMetadata.getExpiredSnapshotPath());
  List<Snapshot> allExpired = snapshotLog.addAll(toExpire);
  List<Snapshot> newExpiredHistory = allExpired.filter(s ->
s.timestampMillis() > oldestRefTimestamp);
  String newFilePath = write(newExpiredHistory);
  tableMetadata.setExpiredSnapshotPath(newFilePath);
}

// current logic
commit(tableMetadata);
purgeFiles(toExpire);
```

Querying

We add the following method to TableMetadata. This will lazily load the 'expired-snapshots-history' file and return their information.

```
public List<Snapshot> expiredSnapshots()
```

We add another Metadata Table for query. This will have the same schema as the 'snapshots' table. It would return the TableMetadata's live snapshots, and also load the expired snapshots from the 'expired-snapshots-history' file. It would have an additional boolean 'expired' to mark those expired snapshots loaded from the file.

```
describe db.table.??total_snapshots;

+-------------+-----------------+-------+
|     col_name|        data_type|comment|
+-------------+-----------------+-------+
| committed_at|        timestamp|   NULL|
|  snapshot_id|           bigint|   NULL|
```

```
|    parent_id|              bigint|   NULL|
|    operation|              string|   NULL|
|manifest_list|              string|   NULL|
|      summary|map<string,string>|   NULL|
|      expired|             boolean|   NULL|
+-------------+--------------------+-------+
```

1. Upgrade

Breaking Changes/ Incompatibilities

This would not be breaking as its an optional pointer, and by default ExpireSnapshots would not retain any Expired Snapshots to store in this file.

Alternatives Considered

Currently, the way to preserve metadata along with running ExpireSnapshots with an aggressive olderThan timestamp is to copy the Snapshot metadata to a backup location manually.

- REST catalog implementations (or any catalogs) can optionally handle this
- External systems like LakeChime do this today

An earlier version proposed adding a boolean 'expired' to the Snapshot object and just extending how many are retained on TableMetadata's snapshots list. However this had the following drawbacks:

- Maintaining too many 'expired' snapshots in the main TableMetadata object will slow down Table writes and reads
- Some existing operations (Table time-travel, cherry-pick, and even DeleteOrphanFiles) would need to account for these new Snapshot types