[Proposal] Add manifest-level statistics to Apache Iceberg for CBO estimation

Author: Xingyuan Lin (xingyuan_lin@apple.com)

Problem Statement

Cost-based optimizers (CBOs) need table statistics to estimate costs.

For example, to estimate the cost to broadcast a table `T` filtered by `c > 10`, the engine needs to know the projected column sizes, and the min/max values of the filtering column `c`. Essentially, the amount of data to be broadcasted can be estimated as

$$total\ size\ of\ projected\ column\ data\ \times \frac{(c_{max} - 10)}{(c_{max} - c_{min})}$$

As a real-world example, Trino fetches statistics including record counts, column min/max values, column NDV counts, column null counts, column nan counts, column sizes, etc. to do CBO estimation. (ref)

As of today (Iceberg v3 or below), these statistics are tracked at data file level.

Consequently, engines have to scan and process manifest files in order to get the stats, which can be an expensive operation when the tables have a lot of data files as well as a wide list of columns. Specifically, the complexity is $O(N * D * C)$, where N is the number of data files in a table, D is the table's dimensionality (number of columns), and C C is a constant value that refers to the number of column statistics types (5 in the Trino+Iceberg case). More specifically, let's talk about real-world scale. Consider a time-series table whose daily data volume has thousands of data files (let's say 5,000), and a wide list of columns (let's say 50), to get the stats for the past 3 month, compute scale would be $5000 * 90 * 50 * 5 \approx 113\ Million$.

High concurrency can make things worse. Warehouse engines like Trino, StarRocks and Snowflake are often deployed as a shared resource to ease the usage, increase resource utilization, etc. As such, high concurrency would put the metadata processing / query optimization component under a heavy load. If the metadata processing component is not horizontally scalable, the expensive computation of stats for CBO can slow down the entire system by jamming the metadata processing part.

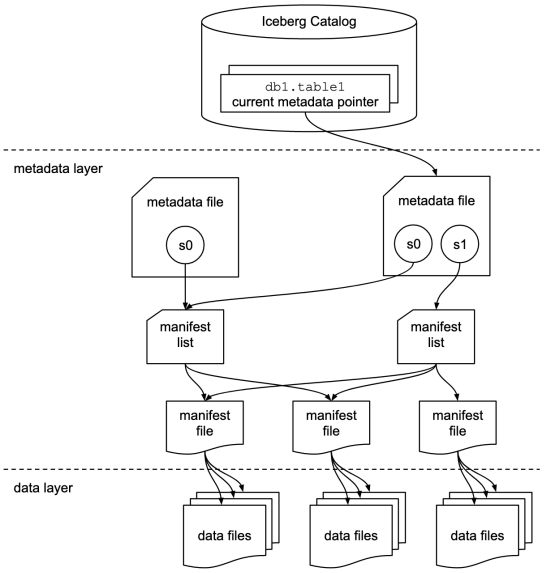
In summary, the problem is

- CBOs need these stats to estimate costs: column min/max values, column null counts, etc.
- These stats are tracked at data file level, so the CBO has to process manifest files to get aggregated results, which can be an expensive operation in large-scale cases.

- High concurrency can make things worse. Due to the expensiveness of stats computation for CBO, the metadata processing / query optimization part can become a bottleneck of the entire system.

Proposed Solution

Background – A recap of existing spec



- **Snapshot** -- The state of a table at some point in time, including the set of all data files.
- **Manifest list** -- A file that lists manifest files; one per snapshot.
- **Manifest** -- A file that lists data or delete files; a subset of a snapshot.
- **Data file** -- A file that contains rows of a table.
- **Delete file** -- A file that encodes rows of a table that are deleted by position or data values.

A **manifest list** is a valid Iceberg data file, which stores **`manifest_file`**, a struct with the following fields:

v1	v2	Field id, name	Type	Description
required	*required*	**`500 manifest_path`**	`string`	Location of the manifest file
required	*required*	**`501 manifest_length`**	`long`	Length of the manifest file in bytes
required	*required*	**`502 partition_spec_id`**	`int`	ID of a partition spec used to write the manifest; must be listed in table metadata `partition-specs`
	required	**`517 content`**	`int` with meaning: `0: data`, `1: deletes`	The type of files tracked by the manifest, either data or delete files; 0 for all v1 manifests
.....				
optional	*optional*	**`507 partitions`**	`list<508: field_summary>`	A list of field summaries for each partition field in the spec. Each field in the list corresponds to a field in the manifest file's partition spec.
.....				

Proposed Change of Spec

To simplify discussion, let's first consider situations without row-level deletes.

The proposed change is to add manifest-level statistics into the **`manifest_file`** records stored in the **manifest list** files. Specifically, the **`manifest_file`** struct will be extended to

v4	Field id, name	Type	Description
required	**`500 manifest_path`**	`string`	Location of the manifest file
.....			
Below are proposed to be added.			
optional	**`521 total_record_count`**	`long`	Total number of records in all data files (that are not in the “DELETED” status. Same applies to below.)
optional	**`522 total_file_size_in_bytes`**	`long`	Total size of all data files in bytes
optional	**`523 total_column_sizes`**	`map<int, long>`	Map from column id to the total size on disk of all regions that store the column. Does not include bytes necessary to read other columns, like footers. Leave null for row-oriented formats (Avro)
optional	**`524 total_null_value_counts`**	`map<int, long>`	Map from column id to number of null values in the column
optional	**`525 total_nan_value_counts`**	`map<int, long>`	Map from column id to number of NaN values in the column
optional	**`526 lower_bounds`**	`map<int, binary>`	Map from column id to lower bound in the column serialized as binary [1]. Each value must be less than or equal to all non-null, non-NaN values in the column for the file [2]
optional	**`527 upper_bounds`**	`map<int, binary>`	Map from column id to upper bound in the column serialized as binary [1]. Each value must be greater than or equal to all non-null, non-Nan values in the column for the file [2]

These statistics are available and valid only when the manifest file tracks data files.

Engine Behavior

During Cost-Based Query Optimizations

Engines process the **manifest list** file to get a list of **`manifest_file`** structs and aggregate the statistics together for CBO usage.

Engines should distinguish partitioning filters from non-partitioning filters. Partitioning filters are those applied on partitioning columns. They can be used to filter out **`manifest_file`** records. If

they are used to filter out **`manifest_file`** records, they shouldn't be used late for filter operator's statistics estimation. Otherwise "double filtering" would occur, making the statistics estimation less accurate.

Let's revisit the synthesized example in the Problem Statement section:

> *Consider a time-series table whose daily data volume has thousands of data files (let's say 5,000), and a wide list of columns (let's say 50), to get the stats for the past 3 month, compute scale would be* $5000 * 90 * 50 * 5 \approx 113\, Million.$

Assume that the table writer makes 24 commits and creates 24 manifest files every day. The compute scale to get the statistics would become $24 * 90 * 50 * 5 = 864k.$ Still non-trivial but much better than the previous one.

During Scan Planning

Engines can use manifest-level statistics to filter out manifest files directly. This is especially useful when rows tracked by one single manifest file have a certain level of clustering or sorting structures.

During Commits

Engines can optionally populate the manifest-level statistics during commits. To get them, the engines only need to aggregate the statistics of all the data files that are tracked by the manifest.

Considering Row-Level Deletes

Cost estimation with row-level deletes is tricky because the engine needs to distinguish between position delete files and equality delete files:

- For a position delete file, the cost estimator should deduct the original record count with the record count of the delete file, as an estimation of record count. And it should do similar things for column_sizes, null_value_counts, nan_value_counts, etc., while keeping lower_bounds and upper_bounds the same because it's too expensive to re-calculate them.
- For an equality delete file, the cost estimator should treat its records as query filters.

There is a design decision to be made on whether the CBO should consider row-level deletes or just ignore them to reduce computation during query planning.

Alternative Solutions

Partition-Level Statistics for CBO Estimation

One alternative solution is to store these statistics on the partition level. In fact, there have been efforts to do that (partition stats in Iceberg , Issue#8450, Issue#10791, Issue#11083). However, the current implementation doesn't include the statistics needed by CBO yet.

Furthermore, compared to partition-level statistics, manifest-level statistics have the following benefits:

- Writers make commits at the manifest granularity and Iceberg tracks data files at the manifest granularity, so it's more natural to aggregate statistics at the manifest level.
    - To maintain the statistics at the manifest level, the writers only need to aggregate the data files they're about to commit. However, to maintain them at the partition level, the writers need to "traverse back" to get existing statistics for the partitions they're going to modify. To make things worse, row-level deletes make it very hard to calculate accurate lower_bounds and upper_bounds for partitions. The writers might need to scan data files to do that. But it is very costly to do so.
    - The reader doesn't need to read the extra partition statistics files. It just needs to read extra stats when parsing the manifest list file.

Appendix

Email thread: https://lists.apache.org/thread/jkt1g4vzjgjbtd5b5dwqs9dzo1ndrwkt