Extending standardized statistics in Iceberg

Gabor Kaszab (gaborkaszab@apache.org)
Denys Kuzmenko (dkuzmenko@apache.org) Initial analysis done here

Goals

This document reviews some of the query engines to identify commonly used statistics for primitive columns and compares them with the standardized statistics currently supported in Iceberg (v1.9). The ultimate goal is to propose new standardized statistics to address existing gaps, enabling Iceberg to serve as an engine-agnostic source of statistical data. This would foster an ecosystem where any engine - regardless of what catalog they use - can rely on Iceberg as the single source of truth for table statistics.

Statistics used by different engines

By clicking the links you can find the current statistics used by various query engines (as of July 2025). To compile this overview, I primarily relied on official documentation and supplemented it with occasional dives into the source code to verify or extend what I found in the docs.
Besides checking the used stats for Iceberg tables I also checked what stats are used for other table types, most particularly for Hive tables.
If you notice any missing metrics or engines, feel free to point them out. I'd be happy to update the list accordingly.

The covered engines:

- Trino (stat breakdown)
- Spark (stat breakdown)
- Hive (stat breakdown)
- Impala (stat breakdown)

Stats in Iceberg V3

Here are the statistics currently offered by Iceberg V3. There are a lot of changes happening in this area recently so this is to give a picture about how things are now (Iceberg 1.9).

Table stats	Table-level column stats	Partition stats	Partition-level column stats
- num rows - num files - file size in bytes + more	- Theta sketch / NDV in Puffin	- num rows (data, eq-del, pos-del, DVs) - num total rows - num files (data, eq-del, pos-del, DVs) - data file size in bytes	None

Iceberg also has another, more fine grained layer of statistics on a per-file basis:

File stats	File-level column stats
- num rows - size in bytes	- min value - max value - num values - null count - NaN count - size in bytes

General notes:

- File stats
  - Stored within the Manifest entries (spec)
  - Written when committing a new file to the table or when rewriting manifests
  - General file-level stats (like num rows, etc.) are required
  - File-level column stats (like min, max values) are optional
- Partition stats
  - Stored in auxiliary partition stat files referenced from table metadata (spec)
  - Written in the format of the table's 'write.format.default' (e.g. Parquet)
  - 1-1 mapping between snapshots and partition stat files
  - Writing partition stats is not part of the snapshot commit process. It is done on-demand after the snapshot is created
- Table stats
  - General stats
    - Iceberg offers a range of general table-level stats through the snapshot summary.
    - These are optional fields that describe what changed compared to the previous snapshot but also give metrics about the state of the table.
  - Table-level column stats
    - Single use is Datasketches Theta for NDV
    - Stored in auxiliary stat files referenced from table metadata (spec)

- Written as blobs in Puffin format
- 1-1 mapping between snapshots and stat files
- Writing stat files is not part of the snapshot commit process. It is done on-demand after the snapshot is created

Proposed stats

In general, the rule of thumb I used for identifying possible additions to the Iceberg spec is - mostly focusing on column stats - if at least 2 engines could use a particular statistics then it's a good candidate.
After browsing through previous conversations (here, here, here) I had the impression that some of the proposed stats already have some kind of consensus about the need or even about the "how" too. Hence, I've separated the proposed stats into two phases: Phase 1 contains the ones seemingly having more consensus, while Phase 2 holds the ones that require more further discussions, or might overlap with other proposals.

Let's see what the table of stats would look like with the candidate statistics!

I used the following colors codes:

- Currently available in Iceberg
- Proposed Phase 1
- Proposed Phase 2

Table stats	Table-level column stats	Partition stats	Partition-level column stats
- num rows - num files - file size in bytes + more	- Theta sketch / NDV (Puffin) - KLL sketch (Puffin) - min value - max value - avg length - max length - null count	- num rows (data, eq-del, pos-del) - num total rows - num files (data, eq-del, pos-del) - num Delete Vectors - data file size in bytes	- min value - max value - num values - null count - NaN count - size in bytes - avg length - max length - Theta sketch

File stats	File-level column stats
- num rows - size in bytes	- min value - max value - num values - null count - NaN count - size in bytes - avg length - max length

Phase 1:

1) Table-level KLL sketches for Histograms

Spark and Hive could currently use histograms for columns for better read cardinality estimates. Apache Datasketches KLL sketch could be used for this purpose.

- Details:
  - Store them as blobs in Puffin
  - For a particular snapshot the same Puffin file can hold Theta and KLL sketches
- There is already a PR from Denys for the Puffin spec changes.
- See proposed implementation details here.

2) Partition-level column stats

Partition stats currently hold information relevant for the partitions in general (like num data files, num data rows, etc.). It would be useful to have also the file-level column stats aggregated on the partition-level too.

- Column stats to introduce on partition-level:
  - min value
  - max value
  - num values
  - null count
  - NaN count
  - size in bytes
- Representation:
  - Add the new column stat fields to existing partition stats spec.
  - New fields are optional
  - As of Iceberg V3 use the same format from file_level:
    map<fieldID, stat type>
  - From Iceberg V4:
    - There are ongoing proposals to restructure file-level column stats.
    - V4 writers can follow the new format of column stats on partition-level too.
- See proposed implementation details here.

Phase 2:

In this phase I didn't go much into details because I think we need some initial conversations to see the exact need, and there might be some overlap with another proposal that we should sort out before drilling down further.
So this section is rather for soliciting feedback and kicking off discussions.

3) File- and partition-level avg length and max length

Spark, Hive and Impala store avg and max length metrics for variable length columns. This could come useful to better estimate the required memory usage to read such columns.

- Details:

        - Add avg length, max length stats to the file-level as optional fields.
        - When aggregating partition-level stats, include these too
    - This improvement might overlap with the restructuring of file-level column stats in V4.

4) Partition-level NDV sketches

Trino and Hive could use a partition-level NDV. With this we could give better cardinality estimates for selective queries especially if the predicates are on partition columns.

- Aggregation from file-level:
    - Partition stats aggregation now works in a way it doesn't read into the actual data files, but aggregates the stats present from file-level.
    - Following this logic, we should introduce Theta sketches on file-level, so that we can aggregate them to partition-level.
    - However, the storage usage of such a design would be unacceptable: One theta sketch size is around 50Kb on average, let's say we calculate it for 5 cols, it's already 250Kb per data file. It can easily grow into the tens or GBs of range (or even higher) per table.
    - So we can't use file-level NDVs for aggregation into partition-level.
- Dilemma on how to calculate:
    - We can't aggregate these sketches from file-level as seen above.
    - Scan of actual data would be needed to produce the partition-level NDV sketches. Either full- or incremental calculation requires scanning data
    - Partition stats calculation is not designed this way, it reads stats from manifest files and manifest entries, but doesn't scan data
    - Data scanning is usually left to the engines (see table-level Theta here)
- Dilemma on how to store:
    - Table-level NDV stats are Datasketches Theta sketches stored as blobs in Puffin files
    - However, currently Iceberg doesn't have Puffins on partition-level
    - Introducing it would also introduce extra complexity to cleanup/maintain
    - Could it be a byte array field in the manifest entry?
    - Users should be careful calculating partition-level Thetas because of storage requirements.
        - As number of partitions grow the size of Thetas on storage can reach the range of GBs (See above: 1 theta ~50KB => Theta for 5 cols is ~250KB per partition)
        - Partition stats are per-snapshot. Have to keep dropping them for older snapshots
        - Calculating for all cols blows up storage usage => only makes sense for a low number of columns
- This improvement might overlap with the restructuring of file-level column stats in V4.

5) Table-level column stats

All the investigated engines could use some table-level column stats. I'll add the intersection here to start a conversation whether they are needed.

- Table-level column stats used by engines:
    - min value
    - max value

    - avg length
    - max length
    - null count
- Dilemma on how to store:
    - Table-level stats now are stored either in Puffin (sketches) or in SnapshotSummary.
    - Puffin is mainly for storing sketches/blobs.
    - Maybe SnapshotSummary, then? Can we expect writers to have this information in commit time or should we calculate them 'offline' such as partition stats and table-level Puffins?

Implementation details

1) Implementation of table-level KLL sketches for Histograms

- 'compute_table_stats' Spark procedure to produce sketches
    - Produce Theta and KLL together => Single table scan
    - No extra parameter is needed, produce all the sketch types unconditionally
    - Discarded idea:
        - We could be smart here: If a writer can only produce one blob type but sees that there is another already written, then we can write the existing blobs together with the new ones into the new table stat Puffin file.
        - However this requires an extra initial read step of existing stats, and also could introduce (an unlikely but possible) race condition: if there is an external write of some stats between the read and write we'd discard the content of that external write.
- Column type support:
    - Datasketches offer KllDoublesSketch, KllLongsSketch, etc., and not a generic KLLSketch that fits all types
    - Gradually adding support is feasible starting with the most straightforward column types (int, log, etc.)
    - No support for nested column types

2) Implementation of partition-level column stats

- 'compute_partition_stats' Spark procedure
    - Also write column-stats into the partition stats files.
    - New optional boolean parameter: '*withColumnStats*'
        - 'False' by default
        - If 'true' then calculate column stats along with the regular stats
    - New optional String[] parameter: 'columns'
        - The names of the columns collect stats for (similar to compute-table-stats)
        - Cover all columns if not provided
- Incremental computation:
    - ~~min and max value could make it difficult to do incremental computation~~

        - ~~What to do when deleting a file with the actual min/max value? We won't know the new value then.~~
        - ~~To resolve such a situation there could be multiple ways:~~
            1) ~~When removing a file from the "edge" of the min/max range, fallback to full computation~~
            2) ~~Always do full computation from the beginning when columns stats are required~~
            3) ~~After reading manifests but before reading manifest entries we can check if files have been deleted by any of the manifests. If yes, do full computation, if no then we can do it incrementally~~
            - ~~I think 3) can the less complex implementation that avoid situations where we realize the need to fallback to full computation right when we almost finished incremental computation~~
    - Getting accurate values within incremental computation is not easy for min/max, because there can be a delete that deletes the exact min/max value. However, flexibility with the accuracy is acceptable here, so we don't deal with the deletes when aggregating stats, including min/max stats, we only look at data files.
- Missing fields
    - Column stats are optional fields on the file-level
    - When one or more files miss a particular column stat then this stat won't be aggregated to the partition-level
    - Log can be written about missing column stats for observability

Per-engine breakdown of statistics coverage

I used the following colors to indicate the availability of the particular stats:

- Available in Iceberg
- Not available but proposed in Iceberg
- Not covered

Trino

(Table stats) (Hive connector stats) (Iceberg connector stats) (TableStatisticsReader), (IcebergStatistics)

Table stats	Table-level column stats	Partition stats	Partition-level column stats
- num rows - num files - size in bytes	- NDV - size in bytes - min value - max value - null count - nulls fraction - NaN count	- num rows - num files - size in bytes	- NDV - size in bytes - min value - max value - null count - nulls fraction - NaN count

Spark

Table stats (link)	Table-level column stats (link)	Partition stats	Partition-level column stats (link)
- num rows - size in bytes	- NDV - min value - max value - null count - avg length - max length - histogram	?	- min value - max value - null count - row count - size in bytes

Hive

(Stat docs)

Table stats	Table-level column stats (ColumnStatisticsObj)	Partition stats	Partition-level column stats (link)
- num rows - size in bytes - num files - num partitions	- NDV - min value - max value - null count - avg length - max length - num trues - num falses - histogram - bit vectors	- num rows - size in bytes - num files	- NDV - min value - max value - null count - avg length - max length - num trues - num falses - histogram - bit vectors

Impala

Table stats	Table-level column stats (from ColumnStatisticsObj)	Partition stats	Partition-level column stats
- num rows - file size in bytes	- NDV - min value - max value - null count - avg length - max length	- num rows - num files - file size in bytes	None

Cleanup of orphaned statistics files

This is a non-goal for this document to solve, however I found it important to mention and raise attention to at this point. When new stats are added besides the existing ones it will be a valid use-case to re-calculate certain stats to include ones that haven't been calculated the

last time (maybe because written by an older writer). Because of how Iceberg references stats now, this would result in the old stat file being unreferenced from the table while adding a reference to the new one, making the old stat file an orphan file.

Details:

- dev@ conversation. Also was brought up in Community Sync
- Community opinion: Unpleasant that normal behaviour creates orphan files, but situation doesn't require intervention at the moment
- Solution is orphan file cleanup

Thoughts to consider:

- Since table location ownership is not guaranteed, orphan file cleanup might not be an option for all the users
- Since we have more ways of creating orphan auxiliary files (puffin files, partition stat files, maybe later indices and who knows what else), we might want to find a way to take care of them automatically within the table library
- Maybe a similar tracking mechanism what we have for metadata.jsons could be considered for auxiliary files like stat files in the future

Other proposals for Iceberg V4

There is a lot of movement around Iceberg V4 and in particular around statistics at the moment. When writing this document the communication around these topics has just started, so just for the record I mention the proposal I'm aware of that might have some interference with this one.

1) Column Stats Improvements (doc)

With this improvement the way file-level column stats are stored would be changed to optimize for better reading performance (projection by column and by stat).
Currently, these stats aren't aggregated on partition-level, but in case we decide to do so then that work has to be coordinated with the restructuring of V4 manifest stats.

2) Iceberg Single File Commits (doc) and Iceberg V4 Adaptive Metadata Tree (doc)

Both of these proposals aim to have a new way how metadata, in particular manifest are organised. This could be relevant when aggregating manifest-level stats into a higher level, for instance for partitions.
How I understand manifest stats could be pre-aggregated into some higher level manifest structures, so aggregating them to partition-level could be a cheaper operation.

Relevant links

- Ajantha's proposal doc for initial partition stats
- Github issue to ask for partition-level min max stats
- Denys' PR for adding KLL and Hive's ColumnStatsObj into Iceberg Puffin spec. Note, the scope of the PR was reduced to add KLL sketch into Puffin spec.