Iceberg Flink Sink V2 migration

Summary

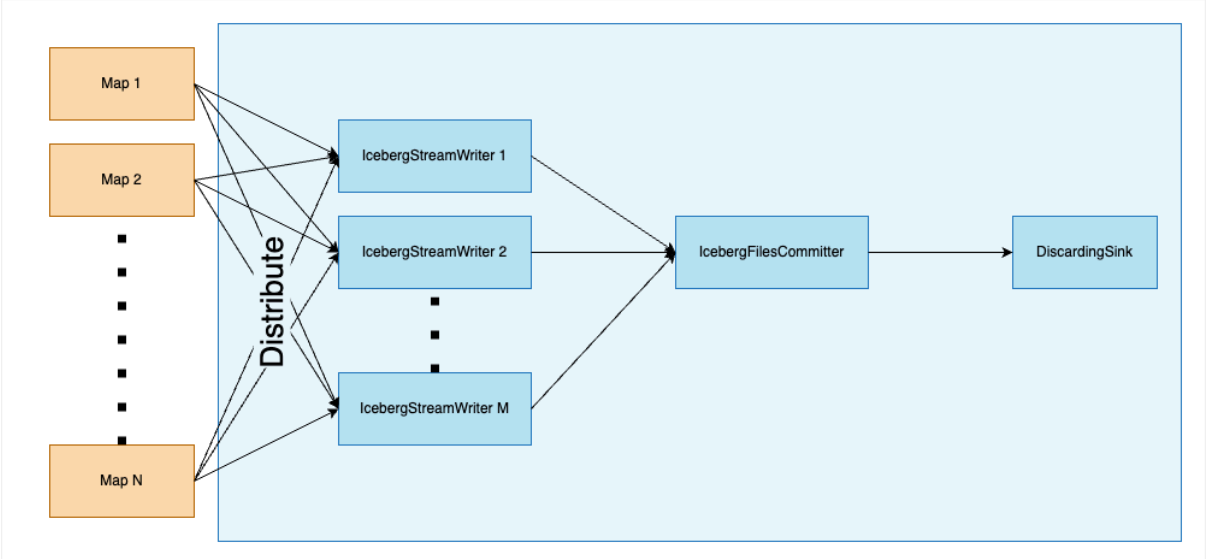
The Flink community created a new Sink specification in FLIP-143 [1] with the explicit goal to guarantee the unified handling of the bounded and unbounded data streams. Later it was enhanced in FLIP-191 [2] so there is a well defined place to execute small files compaction. The deprecation of the old SinkFunction is postponed to somewhere around Flink 2.0 based on the discussion on the dev mailing list [3], so the migration is not extremely urgent, but having the possibility to use the PostCommitTopology to execute the compaction of the small files could provide immediate benefits for the users of the Iceberg-Flink integration.

Previous work

There is an existing Iceberg PR #4904 [4] for the Sink migration by Liwei Li (https://github.com/hililiwei) and Kyle Bendickson (https://github.com/kbendick) with the related documentation [5] which is authored by the same team. The discussion there is stuck, and the PR has been out of date for almost a year now. **The current proposal builds heavily on their work and wants to keep them as the co-authors for the proposed change.**

Current Iceberg sink implementation (SinkFunction)

The current Iceberg sink creates its own operator chain to achieve the desired results to write and commit changes to a specific Iceberg table.



The steps are as below:

1. Distribute / redistribute the incoming flow if needed. This is defined by the Distribution Mode and the defined parallelism - Implemented by adding the required keyBy

function if needed (See: https://github.com/apache/iceberg/blob/a582968975dd30ff4917fbbe999f1be903efac02/flink/v1.17/flink/src/main/java/org/apache/iceberg/flink/sink/FlinkSink.java#L501-L576)
    a. Input: RowData
    b. Output: RowData
    c. Parallelism: Defined by the input
    d. Stateless
2. The writers are writing the records to the storage. The writer behavior (fileformat, compression, data file size, etc) is mostly defined by the table and the write properties - Implemented by the IcebergStreamWriter operator (See: https://github.com/apache/iceberg/blob/a582968975dd30ff4917fbbe999f1be903efac02/flink/v1.17/flink/src/main/java/org/apache/iceberg/flink/sink/IcebergStreamWriter.java)
    a. Input: RowData
    b. Output: WriteResult
    c. Parallelism: Defined by the user, fallback to the input stream parallelism
    d. Stateless
3. The committer is responsible to commit the data to the Iceberg table - Implemented in the IcebergFilesCommitter operator (see: https://github.com/apache/iceberg/blob/a582968975dd30ff4917fbbe999f1be903efac02/flink/v1.17/flink/src/main/java/org/apache/iceberg/flink/sink/IcebergFilesCommitter.java)
    a. Input: WriteResult
    b. Output: Void
    c. Parallelism: 1 - we have a global committer
    d. State: List of serialized DeltaManifests objects (see: https://github.com/apache/iceberg/blob/a582968975dd30ff4917fbbe999f1be903efac02/flink/v1.17/flink/src/main/java/org/apache/iceberg/flink/sink/DeltaManifests.java)
4. DiscardingSink just makes sure that the Flink considers the stream as a sink, since during the previous steps we already committed the data to the Iceberg table

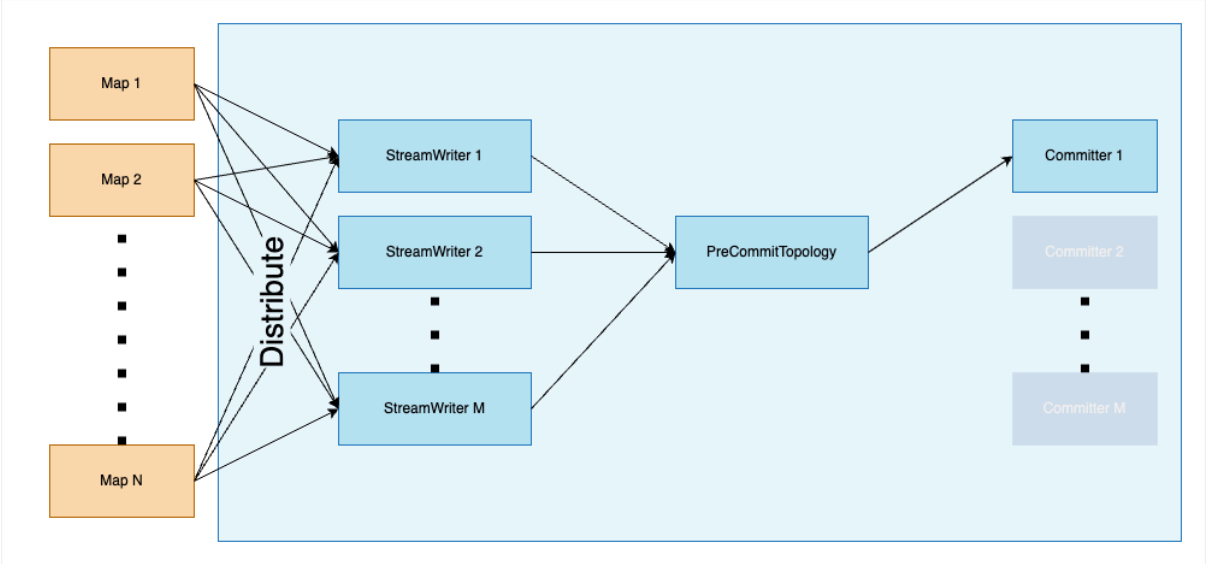
Highlights

I would like to highlight the following important features for the Iceberg V1 sink:

- The state is only handled in the IcebergFilesCommitter, the Writers are stateless
- The state is stored in a single DeltaManifests file for every checkpoint. As a result:
    - Relatively smaller state - only a single manifest file is checkpointed per checkpoint cycle
    - Consistent table view - the checkpoint is either committed, or not committed. We do not end up in a state where one writer's result is committed, but another writer's result for the same checkpoint is not committed to the Iceberg table.

PR 4904 SinkV2 architecture

In their PR [4] the authors propose the use the StatefulSink V2 implementation for the new Iceberg Sink.



The steps are as below:

1. Distribute / redistribute the incoming flow if needed - No change in the behavior
    a. Input: RowData
    b. Output: RowData
    c. Parallelism: Defined by the input
    d. Stateless
2. The writers are writing the records to the storage, and store the result in the state.
    a. Input: RowData
    b. Output: FilesCommittable - which contains DeltaManifests, but only files for the given writer instance
    c. Parallelism: Defined by the user, fallback to the input stream parallelism
    d. State: The StreamWriterState serializes the WriteResults to the DeltaManifests files. These files are written per writer.
3. The PreCommitTopology is used to decorate the committable objects with metadata (subtaskId, jobId), and makes sure that all of the committable records are routed to a single Committer
    a. Input: FilesCommittable
    b. Output: FilesCommittable - decorated
    c. Parallelism: 1
    d. Stateless
4. The FilesCommitter is responsible for committing the data to the Iceberg table and implements the SinkV2 Committer interface. SinkV2 implements the state handling for the operator.
    a. Input: FilesCommittable
    b. Output: FilesCommittable
    c. Parallelism: Same as the writer, but only the 1 is utilized
    d. State: List of serialized FilesCommittable objects - Handled by the SinkV2 operator

5. PostCommitTopology is not implemented yet

Pros

I would like to highlight the following important features of this implementation:

- It uses the SinkV2 API
- The final result is committed to the Iceberg table correctly

Cons

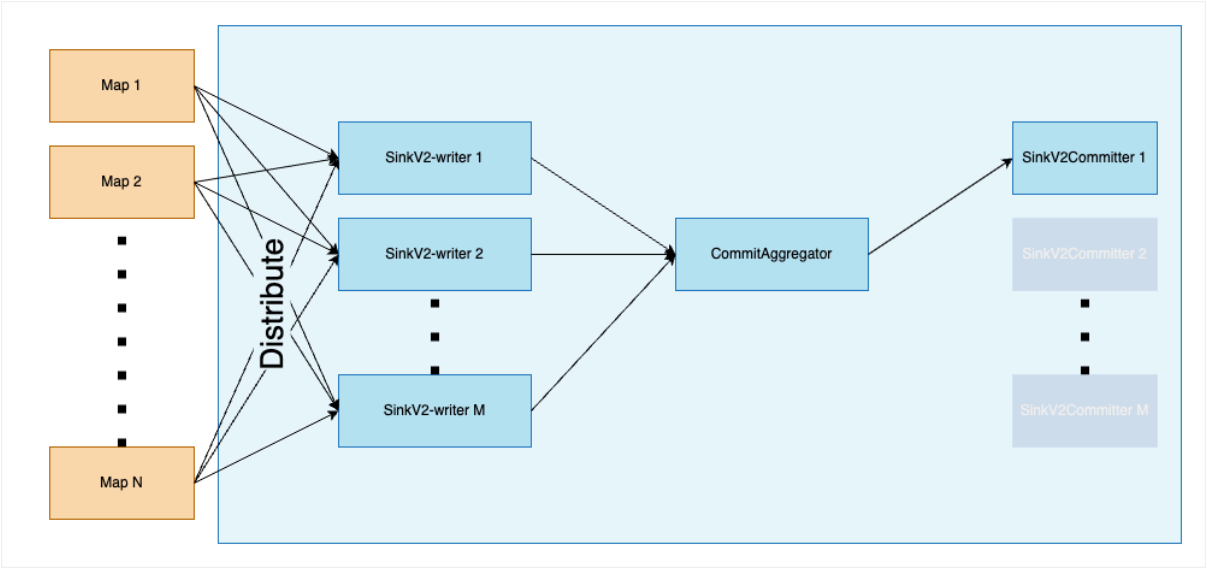
We should improve upon the following properties of the implementation:

- Checkpointing the write result in every writer is not necessary, and adds delay to the checkpoint generation, since we have to write/read M (where M is the writer parallelism of the Sink) number of files instead of 1, as we did in SinkV1.
- The implementation commits all of the changes in a single Iceberg commit. This is ok for append-only tables, but when equality delete files are present, then they should be applied only for data files created by previous commits. Imagine the following situation:
    - Checkpoint 1: +I(1, 'Aliz') - creates a new data file
    - Checkpoint 2: -D(1, 'Aliz') - creates an equality delete file

  The expected result is a table without the record, but equality deletes are applied before the inserts, so if these 2 files are committed in a single Iceberg transaction, then the result would be a table with the existing (1, 'Aliz') record.
- The code duplicates plenty of the current Sink code

Proposed SinkV2 architecture

To have a more performant and reliable SinkV2 implementation, and to also better fit it to the Iceberg needs, we propose the following architecture by implementing the TwoPhaseCommittingSink only instead of the StatefulSink. The proposed architecture from the high level is the same as the [PR 4904 SinkV2 architecture]



There are 2 main difference to the previous implementation

1. The SinkV2Writers are stateless
2. The CommitAggregator is a fully fledged Operator now, which has several responsibilities:
    a. Runs as global - single parallelism, which ensures that every writer result is collected
    b. Aggregates the writer results for every checkpoint
    c. Writes out the DeltaManifests file for every checkpoint

So the steps in proposed architecture would look like this:

1. Distribute / redistribute the incoming flow if needed - No change in the behavior
    a. Input: RowData
    b. Output: RowData
    c. Parallelism: Defined by the input
    d. Stateless
2. The SinkV2Writers are writing the records to the storage, but without keeping anything in state.
    a. Input: RowData
    b. Output: SinkV2Committable - which contains only the WriteResult object
    c. Parallelism: Defined by the user, fallback to the input stream parallelism
    d. Stateless
3. The CommitAggregator is used to aggregate the SinkV2Committables and serialize them to the storage for recovery reasons. Also makes sure that all of the committable records are routed to a single Committer
    a. Input: SinkV2Committable - which contains only the WriteResult object
    b. Output: SinkV2Committable - which contains only the DeltaManifests object
    c. Parallelism: 1
    d. Stateless
4. The FilesCommitter is responsible for committing the data to the Iceberg table and implements the SinkV2 Committer interface. SinkV2 implements the state handling for the operator.
    a. Input: FilesCommittable - which contains only the DeltaManifests object
    b. Output: FilesCommittable - which contains only the DeltaManifests object
    c. Parallelism: Same as the writer, but only the 1 is utilized
    d. State: List of serialized FilesCommittable objects - Handled by the SinkV2 operator
5. PostCommitTopology is not implemented yet

Extract common code

The new code will introduce several new classes for writers, and committers, so we propose to refactor the relevant code to /writer/* and to the /committer/* directory respectively.

Also the proposed solution contains some refactoring of the current code so there is no duplication. This would include:

- SinkBuilder - The base class for collecting the information which is needed for the Sink creation.

- SinkBase - The common initialization / validation code, and also contains the distributeDataStream method which is the same for both Sink implementations.
- CommonCommitter - The committer code which are used by both committers.

Pros

This almost flawlessly replicates the current SinkV1 behavior, and at the same time integrates with the new Sink interface adding the possibility for compaction etc later

Cons

There are some missing Flink features which would mean that the job plan could not be intuitively read (extra committer instances), also the missing features of the Committer API will mean that we lose commit metrics in the new V2 committer, and do some ineffective reinitializations in the committer code.
We propose to enhance the Flink V2 sink to accommodate this features

Missing Flink SinkV2 features

Committer API

Metrics

We would need to initialize the Committer when it is created with minimally a MetricsGroup

```
public interface Committer<CommT> extends AutoCloseable {
   /**
    * Initializes the committer before using it at the first time.
    *
    * @param context the runtime context.
    */
   void init(InitContext context);

   /** Context that {@link #init} can be used for getting additional data about the
runtime environment. */
   @PublicEvolving
   interface InitContext {

   /** @return The metric group this committer belongs to. */
   OperatorMetricGroup metricGroup();
   }
}
```

Parallelism

We need only a single instance of the Committer, so we would need the possibility to set the parallelism for the committer.

WithPreCommitTopology

Currently the PreCommitToplogy expects the operator to only decorate the messages. In the Iceberg sink we need to collect and transform the WriteResult objects, and emit a DeltaManifests object instead. As a workaround we created an union object containing only one of those, but this is clearly only a workaround.
We would like to have something like this:

```
public class IcebergSink extends SinkBase
  implements WithPreWriteTopology<RowData>,
    WithPreCommitTopology<RowData, WriteResult, DeltaManifests>,
    WithPostCommitTopology<RowData, DeltaManifests> {
```

When we start working on the WithPostCommitTopology we might need to transform the emitted records there as well, with something like this:

```
public class IcebergSink extends SinkBase
  implements WithPreWriteTopology<RowData>,
    WithPreCommitTopology<RowData, WriteResult, DeltaManifests>,
    WithPostCommitTopology<RowData, CommitResult> {
```


- [1] - FLIP-143: Unified Sink API - https://cwiki.apache.org/confluence/display/FLINK/FLIP-143%3A+Unified+Sink+API
- [2] - FLIP-193: Extend unified Sink interface to support small file compaction - https://cwiki.apache.org/confluence/display/FLINK/FLIP-191%3A+Extend+unified+Sink+interface+to+support+small+file+compaction
- [3] - [DISUCSS] Deprecate multiple APIs in 1.18 - https://lists.apache.org/thread/3dw4f8frlg8hzlv324ql7n2755bzs9hy
- [4] - Flink: new sink base on the unified sink API - https://github.com/apache/iceberg/pull/4904
- [5] - New FLIP-143 Compliant Flink Iceberg Sink Design - https://docs.google.com/document/d/1G4O6JidAoKgbIdy8Ts73OfG_KBEMpsW-LkXIb89I5k8/edit#heading=h.qqlw5ghn3vp7