Support Writing Bucket Meta-info

Motivation

When using StreamFileSink, one requirement is to support writing meta info when a bucket terminates. For example, users may want to add a _SUCCESS file after the bucket terminated or write a new Partition record into the Hive meta store. Since for cases like Hive the meta store could not bear the pressure that if all tasks write to it, it is better to take this action globally for each bucket. Since a bucket spans multiple subtasks, writing meta-info should happen after all the sub-buckets are terminated and committed.

Sub-bucket Termination Detection (When to Terminate the Bucket)

We could detect whether the bucket has terminated on current subtask at some point:

1. After one record is written.
2. When processing time progressed.
3. When one checkpoint is finished.

Besides, all the buckets need to be terminated at the end of streams if the streams are bounded.

Global Termination Detection (How to Write Bucket Meta-info)

Unbounded Stream

Since there might be failovers during the whole process, there are several options to detect the global termination and ensures the global committer is performed:

(Option 1) A dedicated task guarantees the global commit:

Let Hive Sink to reuse Buckets and implemented as a non-sink operator, then add a new commit operator with parallelism equal to 1 after the Hive sink operator**.** The sink operator sends one message to the commit operator after the sub-bucket is terminated and committed. Since the messages are sent via normal edges, the commit operator could directly use the state to maintain the sub-task commit status.

Pro

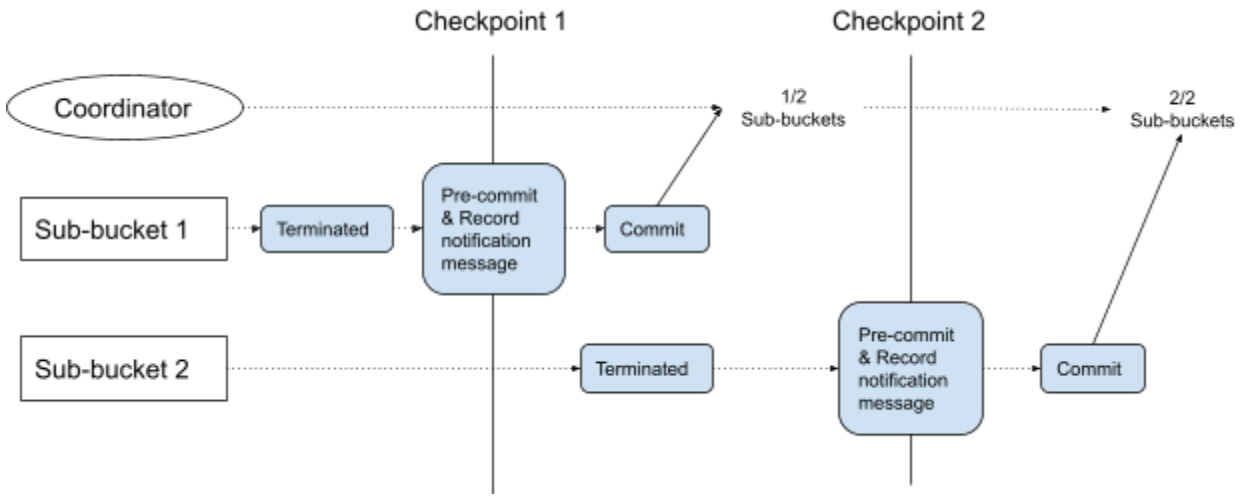
1. Dependency uncertainty is minimal.

Con

1. The dedicated task may connect multiple pipeline regions into one.
2. Table needs to change the logic to compile table sink into two operators.

(Option 2) Sink tasks guarantee the global commit

Similar to option 1, but Operator Coordinator is used instead of specialized tasks. tasks notify the coordinator after the sub-buckets' last commits. Meanwhile, the tasks bookkeeping these notification messages in the tasks' state so that these notifications could be resent after failover. The bookkept notification on task side could be removed after the operator after the global commit is performed and acknowledged.



Pro

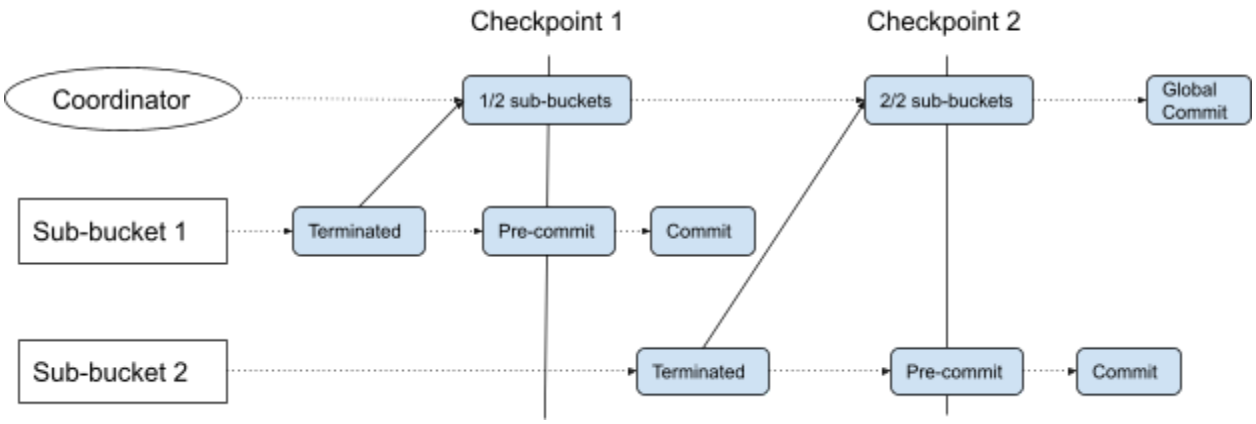
1. Avoid the trouble of connecting multiple pipeline regions and changing the Table compile logic.

Con

2. Complex retrying logic for notification messages between tasks and coordinator.

(Option 3) Operator Coordinator guarantees the global commit

Use Operator Coordinator and make sure the coordinator gets notified of sub-bucket termination in the same checkpoint cycle that the termination is detected. Coordinator will then bookkeep the state of each sub-bucket directly. After failover, coordinator could recover the sub-bucket status directly from the snapshot. For this option, additional care needs to be taken to ensure the global commits happen after the sub-bucket commits, e.g., make the last commits of each sub-bucket also happen in the coordinator together.



Pro

1. Simple logic to ensure the bucket will always be committed.

Con

1. Relying on the coordinator taking snapshots after tasks when checkpointing.
2. Requires additional logic to enable tasks and coordinator to failover separately.

Bounded Stream

For bounded streams, all buckets should be terminated after the stream ends. With the current checkpoint mechanism, there will not be checkpoints after some tasks finish. Therefore, we have to commit the active buckets with no checkpoint for per-commit. This will cause that we might receive the committed records if there are failures during this process.

We might solve this issue by (more detailed implementation is also under investigation):

1. For option 1 and option 2 we could allow checkpoint after tasks finished and make sinks wait for several more checkpoints to finish the final commits before they exit.
2. For option 3, we could simulate a checkpoint when tasks finished. However, since we might proxy the last commits to the operator coordinator, we could also allow tasks to finish in advance and let the coordinator to perform the last commits. This may have scalability problems if sink parallelism is high and we may allow for a multi-instance coordinator.

For batch scenarios that checkpoint is not enabled, since the checkpoint related actions could not be triggered, no buckets could be committed until EOP is received. Therefore, for batch scenarios we also need to commit all the buckets when EOP is received.

Other Questions

There are some remaining questions:

1. The bucket might not span all the tasks, for tasks that do not report sub-bucket termination, it is not easy to distinguish if it will receive records in the future. we could let

the tasks also send notification after a new sub-bucket is created. After all the known sub-buckets are terminated, the coordinator/commit could use active query or timeout to decide if the other empty sub-tasks are finished.

Proposed Changes

Currently all the three options could not resolve the bounded stream problem. At this moment, we might

1. Continue ignoring the issue and let Hive continue to use the original *FileSystemOutputFormat* for batch jobs.
2. Simply commit the sub-buckets and buckets at the end of the stream. However, it might introduce the repeat record problem.

Then we could choose one option for global termination det.

The interface of sub-bucket termination condition is declared as follows:

```
public interface SubBucketTerminationCondition<IN, BucketID> {
  boolean shouldTerminateOnEvent(BucketContext<BucketID> bucketContext, IN
element);

  boolean shouldTerminateOnProcessingTime(BucketContext<BucketID>
bucketContext, long currentTime);

  boolean shouldTerminateOnCheckpoint(BucketContext<BucketID>
bucketContext);
}
```

The logic of the global commit could be declared with

```
public interface BucketTerminationListener<BucketID> {

  void onBucketTermination(BucketContext<BucketID> context);

}
```