FLIP-309: Support using larger checkpointing interval when source is processing backlog

Discussion thread  https://lists.apache.org/thread/l1f7f30h72idjp6ow97y70dcthx7fl37
Vote thread  https://lists.apache.org/thread/1r1h7m7ssr9hyb20p7m20gzt6j245j8n
JIRA  [FLINK-32514 - FLIP-309: Support using larger checkpointing interval when source is processing backlog  CLOSED]
Release  1.19

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

[This FLIP proposal is a joint work between  Yunfeng Zhou and  Dong Lin]

Motivation

In some use-cases, a Flink job will process a bounded stream of backlog data first and then consume an unbounded stream of real-time data. And there is a different requirement for the frequency of checkpoints during these two phases.

For example, a job might want to consume data from HybridSource composed of an HDFS source and Kafka Source, and produce data using Paimon Sink (or any sink with exactly-once semantics whose throughput increases with smaller checkpointing interval). Users might want to trigger the checkpoint once every 30 minutes during the first phase so that the job will re-process at most 30 minutes worth of work after failover; then trigger the checkpoint once every 30 seconds during the second phase so that the sink will flush/commit/expose data to downstream applications once every 30 seconds (needed for data freshness). Note that there is no need to trigger checkpoint once every 30 seconds during the first phase because the records consumed in the first phase is already stale and users do not care about freshness of records produced during this phase.

It is currently hard to meet this requirement because users can only configure a static global checkpoint interval. If we set the checkpointing interval to be 30 seconds in the above example, the throughput of Flink job will be low during the first phase due to unnecessarily high frequency of checkpoints. And if we set the checkpointing interval to be 30 minutes, the data freshness of records produced in the 2nd phase will be worse than what is needed.

In this FLIP, we propose to enable source operators to report whether it is processing backlog. And we will allow users to specify a longer checkpointing interval that can be used when any source is processing backlog.

Note that although we use HybridSource as an example to describe the use-case, we also intend to address the use-case where the source is  MySql CDC. MySQL CDC internally has two phases that reads from a bounded snapshot and an unbounded binlog. There is generally no need for data freshness when reading from the snapshot. So we would also like to support users specifying different checkpointing interval upper-bounds for these two phases.

The approached described below can support this use-case as well.

High-level Overview

We propose to introduce a first-class concept called IsProcessingBacklog. In this section, we will describe its definition, semantics, as well as how this information should be generated and propagated throughout the job graph.

Definition:

Every source has a property, named IsProcessingBacklog, which is a boolean value that can dynamically change over time during the job execution. And every record (logically) has a property, named IsBacklog, which is a boolean value conceptually determined at the point when the record is generated. If a record is generated by a source when the source's IsProcessingBacklog is true, or some of the records used to derive this record (by an operator) has IsBacklog = true, then this record should have IsBacklog = true. Otherwise, this record should have IsBacklog = false.

Note: we will not add IsBacklog to every physical row/record/event. The logical information of the per-record IsBacklog will be propagated via the RecordAttributes (to be introduced in FLIP-327) so that its overhead will be much lower than watermark, suppose the IsBacklog status does not change too frequently.

Semantics:
The isBacklog information effectively tells Flink runtime whether the record should be generated and emitted with low processing latency.

In the stream mode, if a record has isBacklog = false, then the record should be generated and emitted with low processing latency, unless instructed otherwise by the Flink runtime (e.g., waiting for checkpoint triggering due to two-phase commit sink, or execution.end-to-end-latency > 0). If a record has isBacklog = true, then there is no processing latency requirement for this record, which means an operator is allowed to buffer this record arbitrary long until the end of its inputs.

**Rule of thumb for setting isProcessingBacklog = true for source operators:**

A source should have isProcessingBacklog = true if and only if the records it is emitting are allowed to be processed with arbitrarily long latency. This can be because the records from the source are already old/stale, or because the Flink job is lagging behind anyway where the end-to-end latency is bottlenecked by throughput rather than per-record processing latency.

Here are a few examples:

- The MySQL CDC source will read from a bounded stream of snapshot records followed by an unbounded stream of binlog records. Snapshot records are conceptually considered to be old and there is no need to process them with low latency. Therefore, MySQL CDC source can set isProcessingBacklog to true when it is reading snapshot, and set isProcessingBacklog to false when it is reading binlog.
- A Flink job is started to read from a Kafka topic starting at an offset corresponding to 5 days ago. There is no need to process records from 5 days ago with low latency. Therefore, Kafka source can allow user to specify a threshold (e.g. 5 minutes), and set its isProcessingBacklog to true iff the current_system_time - watermark > threshold.

**Rule of thumb for non-source operators to set isBacklog = true for the records it emits:**

An operator should emit RecordAttributes(isBacklog=true) (introduced in FLIP-327) if and only if the records it is emitting are allowed to be processed with arbitrarily long latency. If all inputs of an operator have isBacklog=true, then its output records should have isBacklog=true. Otherwise, if all inputs of an operator have isBacklog=false, then its output records should have isBacklog=false. Otherwise, it is update to the operator to determine its output records' isBacklog value based the operator-specific log.

Here are a few examples:

- A map operator should emit RecordAttributes(isBacklog=true) if and only if it has received RecordAttributes(isBacklog=true) from its input.
- A join operator should emit RecordAttributes(isBacklog=true) if and only if it has received RecordAttributes(isBacklog=true) from any of its inputs. This is because the record generated by join is "fresh" only if records from both sides are "fresh".
- A union operator should emit RecordAttributes(isBacklog=true) if and only if it has received RecordAttributes(isBacklog=true) from all of its inputs. This is because it can still output "fresh" records if there are "fresh" records from any of its inputs.

**Propagation of the isProcessingBacklog/isBacklog information:**

The isBacklog information should be generated by the source operators and be propagated throughout the job graph via RecordAttributes, in a similar way as how watermark/watermark-status is propagated throughout the job graph. Each operator should determine the isBacklog status for its output records based on the isBacklog status of its inputs based on the semantics and the rule of thumb described above.

The figure below shows an example job that joins records from two inputs with different isProcessingBacklog status. The information travels through the job graph via the RecordAttributes event.

**Public Interfaces**

1) Introduce a new concept named “isProcessingBacklog”. And add API for source operators to report its value of isProcessingBacklog during job execution.

The "backlog" is defined as described below:
A job is said to be under backlog status if and only if any source running in the job reports "isProcessingBacklog=true".

A source should report "isProcessingBacklog=true" if and only if it does not need downstream operators to optimize the record processing latency.

When a source reports "isProcessingBacklog=true", it effectively allows downstream operators to focus on throughput rather than latency. It is up to the downstream operators to decide whether to take advantage of this information. For example, an aggregation operator can optimize throughput by buffering/sorting received records before processing and emitting the results.

For example, HybridSource should report "isProcessingBacklog=true" when it is executing the first N-1 sources (suppose it is composed of N sources). And MySQL CDC source should report "isProcessingBacklog=true" when it is in the snapshot stage. In the future, we might introduce rules for sources to dynamically report "isProcessingBacklog=true" when its watermark is considerably behind the current system time.

```java
@Public
public interface SplitEnumeratorContext<SplitT extends SourceSplit> {
    /**
     * Reports to JM whether this source is currently processing backlog.
     *
     * When source is processing backlog, it means the records being emitted by this source
     * is already stale and there is no processing latency requirement for these records. This
     * allows downstream operators to optimize throughput instead of reducing latency for
     * intermediate results.
     *
     * If no API has been explicitly invoked to specify the backlog status of a source, the
     * source is considered to have isProcessingBacklog=false by default.
     */
    void setIsProcessingBacklog(boolean isProcessingBacklog);
}
```

2) Add the job-level config execution.checkpointing.interval-during-backlog

Here is the specifications of this config:

- Name: execution.checkpointing.interval-during-backlog
- Type: Duration
- Default value: null
- Description: If it is not null and any source reports isProcessingBacklog=true, it is the interval in which checkpoints are periodically scheduled. Checkpoint triggering may be delayed by the settings execution.checkpointing.max-concurrent-checkpoints, execution.checkpointing.min-pause. Note: if it is not null, the value must either be 0, which means the checkpoint is disabled during backlog, or be larger than or equal to execution.checkpointing.interval.

3) Update the description of execution.checkpointing.interval to mention that it can be delayed by execution.checkpointing.interval-during-backlog.

Here is the updated description:

Gets the interval in which checkpoints are periodically scheduled. This setting defines the base interval. Checkpoint triggering may be delayed by the settings execution.checkpointing.max-concurrent-checkpoints, execution.checkpointing.min-pause, and execution.checkpointing.interval-during-backlog.

Proposed Changes

1) Update CheckpointCoordinator to use execution.checkpointing.interval-during-backlog as the checkpointing interval if there exists a source operator that is running and has reported isProcessingBacklog=true.

Note that if a source operator has reported the isProcessingBacklog's value multiple times, the last reported value is used.

2) Update HybridSource to report isProcessingBacklog based on the underlying source it is running.

Suppose a HybridSource is composed of N sources where N >= 2. It should report isProcessingBacklog=true when it starts to run the first source. And it should report isProcessingBacklog=false when it starts to run the last source.

3) Update other sources maintained by community (e.g. MySQL CDC source) to report isProcessingBacklog as appropriate.

For example, MySQL CDC source should report isProcessingBacklog=true at the beginning of the snapshot stage. And it should report isProcessingBacklog=false at the beginning of the changelog stage.
Example Usages

Let’s re-use the example use-case specified in the motivation section. Suppose a job needs to consume data from HybridSource composed of an HDFS source and Kafka Source (or any sink with exactly-once semantics). Users want to trigger the checkpoint once every 30 minutes during the first phase so that the job will re-process at most 30 minutes worth of work after failover; then trigger the checkpoint once every 30 seconds during the second phase so that the sink will flush/commit/expose data to downstream applications once every 30 seconds (needed for data freshness)

This use-case can be addressed by using the following configs in flink-conf.yaml.

```yaml
execution.checkpointing.interval: 30sec
execution.checkpointing.interval-during-backlog: 30min
```

Compatibility, Deprecation, and Migration Plan

The new config execution.checkpointing.interval-during-backlog is set to null by default, which means it is disabled by default. This the changes proposed in this FLIP is backward compatible.

No deprecation or migration plan is needed.

Future Work

We plan to introduce more strategy for source to determine whether its isProcessingBacklog should be true. For example, we plan to work on FLIP-328 very soon after this FLIP is approved to support determining backlog status based on the event-time watermark lag.