KIP-93: Improve invalid timestamp handling in Kafka Streams

Status

Current state: *Accepted* *[VOTE] KIP-93: Improve invalid timestamp handling in Kafka Streams*

Discussion thread: *[DISCUSS] KIP-93: Improve invalid timestamp handling in Kafka Streams*

JIRA: Unable to render Jira issues macro, execution error.

Released: 0.10.2.0

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

Motivation

Currently, Kafka Streams does not handle invalid (i.e., negative) timestamps returned from the `TimestampExtractor` gracefully, but fails with an exception, because negative timestamps cannot get handled in a meaningful way for any time based operators like window aggregates or joins.

Negative timestamp can occur for several reason.

1. You consume a topic that is written by old Kafka producer clients (i.e., version 0.9 or earlier), which don't use the new message format, and thus meta data timestamp field defaults to `-1` if the topic is configured with `log.message.timestamp.type=CreateTime`
2. You consume a pre-0.10 topic after upgrading your Kafka cluster from 0.9 to 0.10: here all the data that was generated with 0.9 producers is not compatible with the 0.10 message format (and defaults to timestamp -1)
3. You consume a topic that is being written to by a third-party producer client that allows for embedding negative timestamps (`KafkaProducer` does check for negative timestamp an raises an exception for this case preventing invalid timestamp in the first place)
4. The user provides a custom timestamp extractor that extracts a timestamp for the payload data (i.e., key-value pair), and this custom extractor might return negative timestamps.

If **all** records happen to have negative timestamps (case 1 and 2), this KIP does not improve the situation much. However, if only **some** records have negative timestamps (could happen for case 3 and 4 above), we want to improve the situation and make it easier for the user to process such topics. Right now, it is not possible to process these topics at all because the Streams application will raise an exception at some point (only a global exception handler could get registered to prevent the application to die, however, this does not solve the issue as the `StreamThread` dies, partitions get reassigned and the next thread hits the same issue again, until not threads are left and the application is dead).

Public Interfaces

The signature of `TimestampExtractor` will be changed, to give the user a way to "infer" a timestamp from the current processing progress (i.e., internally tracked *stream-time*) if no valid TS can be extracted from the record.

```
// current interface
public interface TimestampExtractor {
    long extract(ConsumerRecord<Object, Object> record);
}

// new interface
public interface TimestampExtractor {
    long extract(ConsumerRecord<Object, Object> record, long previousTimestamp); // previousTimestamp provides
the TS from the latest extracted valid TS for the same partition the current record belongs to
}
```

Furthermore, default implementation of `ConsumerRecordTimestampExtractor` is replaces and new classes will be added to provide more predefined timestamp extractors (see details below):

- `FailOnInvalidTimestamp` (new default extractor, replacing `ConsumerRecordTimestampExtractor`; behavior stays the same)
- `LogAndSkipOnInvalidTimestamp`
- `UsePreviousTimeOnInvalidTimestamp`

Proposed Changes

We want to change Streams to an auto-drop behavior for records with negative timestamps (without any further user notification about any dropped records) to enable users to "step over" those records and keep the app running (instead of running into a runtime exception, which would typically bring down the whole application instance). To guard the user from silently dropping messages by default (and to keep the current fail-fast behavior), we change the default extractor `ConsumerRecordTimestampExtractor` to raise an exception if the embedded 0.10 message timestamp is negative, which includes the case where there is no 0.10 timestamp embedded in the message.

Furthermore, we want to add some reference implementations of timestamp extractor for covering common use cases (incl. case 3 above):

- a "drop-and-log" extractor: this extractor writes a WARN log message in case a negative timestamp is extracted (the record with negative timestamp will ultimately be skipped by Streams). The idea is that users want to skip over records with negative timestamps, but still get a some information about any such dropped records instead of silently "losing" data. Users can then leverage monitoring tools to track which, when, and how many messages are being dropped.
- a "timestamp inferring" extractor: this extractor tries to infer a new timestamp for a record if the originally extracted timestamp was negative. This allows to "fix" missing timestamps, enabling the processing of all records without data loss

For any other behavior, users can still provide a custom timestamp extractor implementation. As the `TimestampExtractor` interface will change, users cannot reuse old extractors and thus are made aware of the new behavior, thus case 4 is also covered.

Additionally, we want to add a new streams metric that reports the number of skipped record (as absolute count or percentage or both) to give the user a way to monitor if messages get skipped.

Compatibility, Deprecation, and Migration Plan

This is a breaking, incompatible change because `TimestampExtractor` interface gets changed. However, it only affect uses that provide a custom timestamp extractor. By default no code change is required and the overall behavior is the same as before this KIP (using default timestamp extractor user gets an exception in case of a negative timestamp).

- Even if the exception is throw from a different point and the exception message changes (the exception type is the same: `StreamsException`) the use cannot recover from the exception anyway (i.e., user cannot recover with current behavior and this will not change)

Required code changes:

- Custom timestamp extractors must be updated:
  - recompile to avoid runtime exception
  - code change to adapt to new interface (to make it compile)

Test Plan

The feature can be tested via unit tests.

Rejected Alternatives

- add a new error handler that is called if a negative timestamp is detected
  - this error handler raises an exception by default
  - user can provide custom error handle via `StreamsConfig`
  - rejected because it separates the root-case of negative timestamps from reacting/fixing those
    - for custom timestamp extractors, user can check for negative timestamps in the first place (directly after extracting it and before returning it to Streams) and react accordingly
    - thus, error handle is too clumsy to use and too hard to reason about from a user perspective