Event Time and Watermarks in KafkaIO

(shared with Apache Beam community)

Background

KafkaIO source in Apache Beam by default uses processing time for event time and last processing time for source watermark, which is not very useful and inaccurate (e.g. when a pipeline falls behind realtime). KafkaIO lets user provide custom functions to extract event timestamps and to set watermark, but the API is does not have sufficient context to implement source watermark properly by the user. E.g. when the event time is in the past, user can not tell if it is due to high backlog or due to idle incoming traffic.

Kafka topics since version 0.10.1 can have either “server time” or “producer time” associated with each record. It would be convenient to support these event times in KafkaIO.

Proposal

The proposal below provides reasonable default implementations for handling both producer time and server time. In order delivery of records with in Kafka partition makes it possible to implement a near perfect watermark in the case of server time. Topics with with producer (or custom) time share the implementation of source watermark from PubsubIO, which keeps track of timestamp over last minute. Note that Dataflow pipelines use a native implementation of PubsubIO that has better tracking of event time watermarks both in the case of server side timestamps as well as with custom timestamps.

A Kafka topic consists of set of *partitions*. These partitions are distributed evenly among KafkaIO source splits. As a result, each split consumes from one or more partitions. The records in each partition are read in order.

Producer time (timestamp type = CREATE_TIME) or with **custom timestamps**

- The reader can not assume any bounds on timestamp spread. PubsubIO source keeps track of timestamps over last one minute’s worth of messages and returns the lowest timestamp as watermark. PubsubIO watermark implementation will be refactored and shared with KafkaIO. In some cases, a topic might have tighter bounds on drift, and user might want to adjust this duration from default 1 minute. This could be an option to the source. Both custom timestamps extracted by user handler and producer timestamps are treated the same way.

Server time *(timestamp type = LOG_APPEND_TIME)*

- As the records are read in the order they are appended to logs on Kafka, the timestamp monotonically increases, which makes it possible to track "*near perfect watermark*":

```
 partition_watermark =
   if (partition reader is not caught up)  // i.e. backlog > 0
     last record timestamp
   else
     last backlog check time               // see note below
split_watermark = min(partition watermarks)
```

- When the reader does not receive any records from server (e.g. idle topic), we can advance the watermark to current time if we are certain that there aren't any records left to read. I.e,

  ```
  Instant now = …
  fetch_latest_offsets();
  if (latest_offset = consumed_offset)
    partition_watermark = now - epsilon; // epsilon ~ 1 or 2 sec.
  ```

  This policy assumes synced clocks on both the workers and the servers, which is likely a reasonable assumption. But it does not know about any internal delays or in-flight records on Kafka servers. Setting watermark to 1 or 2 seconds behind '*now*' might be good enough in most cases. Another option is to make use of offsetsForTimes() interface, to check for any records with *timestamp* >= *current_watermark*, with the expectation that Kafka server takes any in-flight records into account. But the the JavaDoc does not explicitly guarantee. For now, simple policy to keep 1 or 2 buffer might be good enough.
    - Another aspect is frequency of polling for latest offsets. Existing implementation checks latest offsets every 5 seconds (mainly to calculate backlog), which would delay watermark by 5 to 7 seconds when the source is caught up. We could increase the frequency in such a case to once every second or so.