Mini design doc: dropDuplicates within watermark

Author: Jungtaek Lim
Date: 2023-03-24

Motivation

We got many reports that dropDuplicates does not clean up the state even though they have set a watermark for the query. We document the behavior clearly that the event time column should be a part of the subset columns for deduplication to clean up the state (link), but it cannot be applied to the customers as timestamps are not exactly the same for duplicated events in their use cases.

We looked into these reports, and figured out the common pattern across the majority of the reports. They set the watermark with a delay threshold, and expect Spark to deduplicate events accounting for the delay of the watermark.

That majorly falls under the scenario which the producer of events (pipeline) has somehow produced duplicate events (e.g. non-idempotent writers) but the event does not have an event time (or the value is incorrect/inconsistent), hence a different timestamp is issued/used for each event (e.g. ingestion timestamp). For the scenario, users would understand how long the difference among timestamps of duplicate events could be, and produce the delay threshold of watermark longer than max timestamp differences among duplicated events.

Let's look into one of examples:

```
df.withWatermark("eventTime", "4 hours").dropDuplicates("guid")
```

Given below input, their expectation is that Spark will deduplicate (ID: 1, TIME: 4) and (ID: 4, TIME: 5), because it's within 4 hours from the time Spark sees the same ID.

ID	TIME
1	1
2	2
4	3
1	4

4	5

This is challenging to address with current semantics of dropDuplicates. The existing dropDuplicates API aims for correct output - if there is no late event, the output must be the same between batch and streaming despite any streaming specific circumstances, including out-of-order events, etc. This does not allow us to evict the state if the event time column is not included in the subset.

For example, the operator just processed the event {"guid": "abc", "eventTime": "2023-03-17 00:00:00"}. The subset is specified to "guid", which means, all events should be considered as duplicates if they have the same value for "guid".

We can evict the state when it is guaranteed to not be referenced (read/update) in future, but there is "possibility" that the state can be referenced in any future date. For example, an event happening 4 years later like {"guid": "abc", "eventTime": "2027-03-17 00:00:00"} should also be deduplicated.

That requires us to come up with looser semantics.

Out of scope

The new API does not aim to solve the general issue of dropDuplicates. For example, we do not address the scenario where different events are produced with different timestamps and users want to deduplicate these events with certain criteria. For such scenario, they may want to retain the state as long as they are referenced, and evict the state in some criteria (e.g. it hasn't been referenced for specified duration). We classify this case as an "advanced" use case, and encourage users to implement via flatMapGroupsWithState. (Reference implementation) We can revisit deducing a new API once we have enough use cases demanding for this.

Proposal

We propose to deduce a new API of dropDuplicates which has following different characteristics compared to existing dropDuplicates:

- Weaker constraints on the subset (key)
  - Does not require an event time column on the subset.
- Looser semantics on deduplication
  - Only guarantee to deduplicate events within the watermark.

Since the new API leverages event time, the new API has following new requirements:

- The input must be streaming DataFrame.
- The watermark must be defined.
- The event time column must be defined in the input DataFrame.

More specifically on the semantic, once the operator processes the first arrived event, events arriving within the watermark for the first event will be deduplicated.
(Technically, the expiration time should be the "event time of the first arrived event + watermark delay threshold", to match up with future events.)

Users are encouraged to set the delay threshold of watermark longer than max timestamp differences among duplicated events. (If they are unsure, they can alternatively set the delay threshold large enough, e.g. 48 hours.)

The proposed method signatures are following (PySpark will have equivalent method):

```
// Scala/Java, all columns
def dropDuplicatesWithinWatermark(): Dataset[T]

// Scala friendly, specifying subset
def dropDuplicatesWithinWatermark(colNames: Seq[String]): Dataset[T]

// Java friendly, specifying subset
def dropDuplicatesWithinWatermark(colNames: Array[String]): Dataset[T]

// Scala friendly varargs subset
def dropDuplicatesWithinWatermark(col1: String, cols: String*): Dataset[T]
```

The method name starts with "dropDuplicates" to indicate that the new API is sibling (a variant) with existing dropDuplicates.

Here is the walkthrough example.

```
clean_answers
  .withWatermark("eventTime", "48 hours")
  .dropDuplicatesWithinWatermark("guid")
```

With the above query, watermark is set to accept late events up to 48 hours, which means the operator will produce a first occurrence event among deduplicated events for 48 hours.

Batch 1
Input: {“guid”: “abc”, “eventTime”: “2023-03-17 00:00:00”}
Watermark: 0

dropDuplicatesWithinWatermark

- There is no deduplication data in state with the key.
- The operator puts {“guid”: “abc”, “expiresAt”: “2023-03-19 00:00:00”} into the state.
    - “2023-03-17 00:00:00” + 48 hours
- The operator emits the event {“guid”: “abc”, “eventTime”: “2023-03-17 00:00:00”}, as it’s the first occurrence.
- The operator checks the state for eviction, there is no deduplication data in which the value of expiresAt is earlier than the watermark.

——————
Batch 2
Input: {“guid”: “abc”, “eventTime”: “2023-03-16 00:00:00”}
Watermark: “2023-03-15 00:00:00” (“2023-03-17 00:00:00” - 48 hours)

dropDuplicatesWithinWatermark

- There is a deduplication data in state: {“guid”: “abc”, “expiresAt”: “2023-03-19 00:00:00”}
- The operator deduplicates the event, hence nothing is emitted.
- The operator checks the state for eviction, there is no deduplication data in which the value of expiresAt is earlier than the watermark.

——————
Batch 3
Input: {“guid”: “abc”, “eventTime”: “2023-03-13 00:00:00”}
Watermark: “2023-03-15 00:00:00” (didn’t advance)

dropDuplicatesWithinWatermark

- The operator drops the event as a late record.
- The operator checks the state for eviction, there is no deduplication data in which the value of expiresAt is earlier than the watermark.

——————
Batch 4
Input: {“guid”: “abc”, “eventTime”: “2023-03-20 00:00:00”}
Watermark: “2023-03-15 00:00:00” (didn’t advance)

dropDuplicatesWithinWatermark

- There is a deduplication data in state: {“guid”: “abc”, “expiresAt”: “2023-03-19 00:00:00”}

- The operator deduplicates the event, hence nothing is emitted.
- The operator checks the state for eviction, there is no deduplication data in which the value of expiresAt is earlier than the watermark.

———
Batch 5
Input: {"guid": "def", "eventTime": "2023-03-25 00:00:00"}
Watermark: "2023-03-18 00:00:00" ("2023-03-20 00:00:00" - 48 hours)

dropDuplicatesWithinWatermark
- There is no deduplication data in state with the key.
- The operator puts {"guid": "def", "expiresAt": "2023-03-27 00:00:00"} into the state.
    - "2023-03-25 00:00:00" + 48 hours
- The operator emits the event {"guid": "def", "eventTime": "2023-03-25 00:00:00"}, as it's the first occurrence.
- The operator checks the state for eviction, there is no deduplication data in which the value of expiresAt is earlier than the watermark.

———
Batch 6
Input: {"guid": "def", "eventTime": "2023-03-25 01:00:00"}
Watermark: "2023-03-23 00:00:00" ("2023-03-25 00:00:00" - 48 hours)

dropDuplicatesWithinWatermark
- There is a deduplication data in state: {"guid": "def", "expiresAt": "2023-03-25 00:00:00"}
- The operator deduplicates the event, hence nothing is emitted.
- The operator checks the state for eviction, figures out {"guid": "abc", "expiresAt": "2023-03-19 00:00:00"}, and evicts it.

———
Batch 7

Input: {"guid": "abc", "eventTime": "2023-03-24 01:00:00"}
Watermark: "2023-03-23 01:00:00" ("2023-03-25 01:00:00" - 48 hours)

dropDuplicatesWithinWatermark
- There is no deduplication data in state with the key.
- The operator puts {"guid": "abc", "expiresAt": "2023-03-26 01:00:00"} into the state.
- The operator emits the event {"guid": "abc", "eventTime": "2023-03-24 01:00:00"}, as it's the first occurrence.
- The operator checks the state for eviction, nothing is earlier than the watermark.