KIP-655: Windowed Distinct Operation for Kafka Streams API

Status

Current state: Voting

Discussion thread: *here*

Voting thread: *here*

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

Pull request: PR-9210

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

Motivation

Message deduplication is a common task.

One example: we might have multiple data sources each reporting its state periodically with a relatively high frequency, their current states should be stored in a database. In case the actual change of the state occurs with a lower frequency than it is reported, in order to reduce the number of writes to the database we might want to filter out duplicated messages using Kafka Streams.

'Distinct' operation is common in data processing, e. g.

- SQL `DISTINCT` keyword,
- In standard libraries for programming languages
  - .NET LINQ `Distinct` method,
  - Java Stream `distinct()`,
  - Scala Seq `distinct()`,
- In data processing frameworks:
  - Apache Spark's `distinct()`,
  - Apache Flink's `distinct()`,
  - Apache Beam's `Distinct()`,
  - Hazelcast Jet's `distinct()`, etc.

Hence it is natural to expect the similar functionality from Kafka Streams.

Although Kafka Streams Tutorials contains an example of how `distinct` can be emulated, but this example is complicated: it involves low-level coding with local state store and a custom transformer. It might be much more convenient to have `distinct` as a first-class DSL operation.

Due to 'infinite' nature of KStream, `distinct` operation should be windowed, similar to windowed joins and aggregations for KStreams.

Public Interfaces

In accordance with KStreams DSL Grammar, we introduce the following new elements:

- **`distinct()`** parameterless DSLOperation on

- TimeWindowedKStream<K,V> DSLObject which returns KStream<Windowed<K>,V>
- SessionWindowedKStream<K,V> DSLObject which returns KStream<Windowed<K>,V>

The following methods are added to the corresponding interfaces:

```
KTable<Windowed<K>, V> distinct(final Named named);
KTable<Windowed<K>, V> distinct(final Materialized<K, V, WindowStore<Bytes, byte[]>> materialized);
KTable<Windowed<K>, V> distinct(final Named named,
                                final Materialized<K, V, WindowStore<Bytes, byte[]>> materialized);
```

The **distinct** operation returns only a first record that falls into a new window, and filters out all the other records that fall into an already existing window.

The records are considered to be duplicates iff serialized forms of their keys are equal.

Usage Examples

Consider the following example (record times are in seconds):

```
//three bursts of variously ordered records
4, 5, 6
23, 22, 24
34, 33, 32
//'late arrivals'
7, 22, 35
```

'Epoch-aligned deduplication' using tumbling windows

```
.groupByKey().windowedBy(TimeWindows.of(Duration.ofSeconds(10))).distinct()
```

produces

```
(key@[00000/10000], 4)
(key@[20000/30000], 23)
(key@[30000/40000], 34)
```

-- that is, one record per epoch-aligned window.

Note: hopping and sliding windows do not make much sense for `distinct()` because they produce multiple intersected windows, so that one record can be multiplied instead of deduplication.

SessionWindows work for 'data-aligned deduplication'.

```
.groupByKey().windowedBy(SessionWindows.with(Duration.ofSeconds(10))).distinct()
```

produces only

```
([key@4000/4000], 4)
([key@23000/23000], 23)
```

because all the records bigger than 7 are 'stuck together' in one session. Setting inactivity gap to 9 seconds will return three records:

```
([key@4000/4000], 4)
([key@23000/23000], 23)
([key@34000/34000], 34)
```

Compatibility, Deprecation, and Migration Plan

The proposed change is backwards compatible, no deprecation or migration needed.

Rejected Alternatives

The following was rejected during the discussion in favour of simpler approach:

Public Interfaces

In accordance with KStreams DSL Grammar, we introduce the following new elements:

- **`distinct`** DSLOperation on a `KStream<K, V>` DSLObject which returns another `KStream<K, V>` DSLObject,
- **`DistinctParameters<K, V, I>`** DSLParameter.

The type parameters are:

- K — key type
- V — value type
- `I` — the type of the record's unique identifier

With `DistinctParameters<K, V, I>` the following can be provided:

1. `KeyValueMapper<K, V, I> idExtractor` — extracts a unique identifier from a record by which we de-duplicate input records. If it returns null, the record will not be considered for de-duping and forwarded as-is. If not provided, defaults to `(key, value) -> key`, which means deduplication based on key of the record. **Important assumption:** records from different partitions should have different IDs, otherwise same IDs might be not co-partitioned.
2. `TimeWindows timeWindows` — tumbling or hopping time-based window specification. Required parameter. Only the first message with a given id that falls into a window will be passed downstream.
3. `Serde<I> idSerde` — serde for unique identifier.
4. `boolean isPersistent` — whether the `WindowStore` that stores the unique ids should be persistent or not. In many cases, non-persistent store will be preferrable because of better performance. Downstream consumers must be ready to accept occasional duplicates.

Proposed Changes

1. Add the following method to `KStream` interface:

```
<I> KStream<K, V> distinct(DistinctParameters<K, V, I> params);
```

Given the parameters, this method returns a new `KStream` with only the first occurence of each record in any of the time windows, deduplicated by unique id. Any subsequent occurences in the time window are filtered out.

2. Add and implement the following DistinctParameters class:

```
class DistinctParameters<K, V, I> extends Named {
    /** Windowing parameters only. {@code (k,v)->k} id extractor is assumed, and a persistent store with key
serde is used*/
    public static <K, V> DistinctParameters<K, V, K> with(final TimeWindows timeWindows);

   /** Windowing parameters and a store persistency flag. {@code (k,v)->k} id extractor is assumed and a key
serde is used*/
    public static <K, V> DistinctParameters<K, V, K> with(final TimeWindows timeWindows, final boolean
isPersistent);

    /** Windowing parameters, ID extractor, and a serde for unique IDs. A persistent store will be used.*/
    public static <K, V, I> DistinctParameters<K, V, I> with(final TimeWindows timeWindows,
                                                             final KeyValueMapper<K, V, I> idExtractor,
                                                             final Serde<I> idSerde);
    /** Windowing parameters, ID extractor, a serde for unique IDs, and a flag showing whether the {@code
WindowStore} should be
     * persistent or not.*/
    public static <K, V, I> DistinctParameters<K, V, I> with(final TimeWindows timeWindows,
                                                             final KeyValueMapper<K, V, I> idExtractor,
                                                             final Serde<I> idSerde,
                                                             final boolean isPersistent)
}
```