Implementation of the Linear Road Benchmark on Kafka Streams

Tayfun Wiechert

Technische Universität Berlin

Table of Contents

1 Stream and Table Components

The following section summarizes the streams and tables that are generated to realize the benchmark.

1.1 Toll History Table

Input stream/topic	Toll History Topic
Implemented in	stream/historical/table/TollHistoryTableBuilder
Output Type	Table
Output key	XwayVehicleDay
Output value	Double

In order to respond to the daily expenditure request, we need the historical information of total expenditures per day, xway and vehicle. These historical information were generated in preparation and have to be made available to the application. The information is materialized to a table (KTable) which is created and fed before the actual benchmark starts. During runtime, the stream of daily expenditure request (1.5) is joined with this table. As the table is partioned by the tuple key, the operators will perform only local table lookups to respond to the queries.

1.2 Current Expenditure per Vehicle Table

Input stream/topic	Segment Crossing Position Report Stream, Current Toll Stream
Implemented in	stream/historical/table/ CurrentExpenditurePerVehicleTableBuilder
Output Type	Table
Output key	Integer
Output value	ExpenditureAt

This table is dynamically updated and holds the summed expenditures per vehicle. It is required to respond to the type 2 queries and is generated based on the stream of current tolls (keyed by $(xway, vehicle, day)$) and the segment crossing position reports. It has to be considered, that a toll is charged only once per segment. Thus, we use the reduced segment crossing position report stream as input stream. Additionally, it has to be considered that the toll assessment processed here is asynchronous to the toll notification that is sent before, i.e. the toll is assessed after leaving the respective segment: »Every time a position report identifies a vehicle as crossing from one segment into another, the toll reported for the segment being exited is assessed to the vehicle's account..«

When a position report of segment s and minute m arrives, we need to asses the toll that was valid at $s-1$ in that point in time, the driver has entered the segment $s-1$. For that reason, the segment crossing position report tuples carry the point in time $\overleftarrow{m}$ when $s-1$ has been entered by the same driver.

The segment crossing position report p and the current toll stream are joined, after re-keying p such that the segment s will be changed to $s-1$. Because we need the valid toll at minute $\overleftarrow{m}$, the event time of p is also modified. The joined streams are still keyed by $(xway, vehicle, day)$, which is changed by a map operation to a key only considering the vehicle id. That steam in turn represents a stream of accountable tolls per vehicle. Applying an unwindowed aggregation, that simply sums up the single tolls, yields a KTable which is then materialized.

1.3 Position Report Stream

Input stream/topic	PositionReport Topic
Implemented in	stream/PositionReportStreamBuilder
Output Type	Stream
Output key	XwaySegmentDirection
Output value	PositionReport

The stream of position reports is the starting point, as it used to determine average velocities in a certain segment or the number of vehicles both in a certain time value. In order to create that stream, the application reads from a Kafka topic that is fed after deployment and start of the stream topology. The stream is keyed per xway, segment and direction, which makes it directly reusable for the NOV, LAV and Accident Detection stream.

1.4 Segment Crossing Position Report Stream

Input stream/topic	PositionReport Topic
Implemented in	stream/SegmentCrossingPositionReportBuilder
Output Type	Stream
Output key	VehicleIdXwayDirection
Output value	SegmentCrossin

A toll is only charged, if the driver has crossed a segment. Multiple position reports in the same segment must not cause multiple charges. Therefore this stream is built on the basis of the position report stream. Firstly, it is re-keyed to $(vehicleId, xway, direction)$ tuples, then this stream of position reports per vehicle on a certain expressway and direction is reduced such that only the first position report within one segment is preserved. The underlying idea is visualized in figure 1.

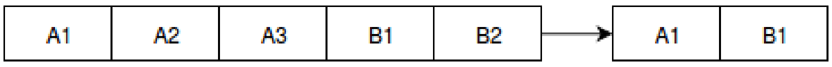

Fig. 1. Principle of stream reduction to preserve only the first position report from a segment.

One way to reduce the re-keyed stream to the segment crossing stream would be to apply a custom function on a data-driven window that creates a new window, whenever a position report of a new segment per considered key is observed. Neither data-driven windows nor the ability to apply a custom function on a window are available features in Kafka. Therefore the re-keyed position report stream p_1 is duplicated and the event times of the duplicate $p2$ are shifted by

$+30s$. Then p_1 is left-joined with $p2$ and the respective element from the first stream is emitted if $segment(p_1) \neq segment(p_2)$ holds (see figure 2).

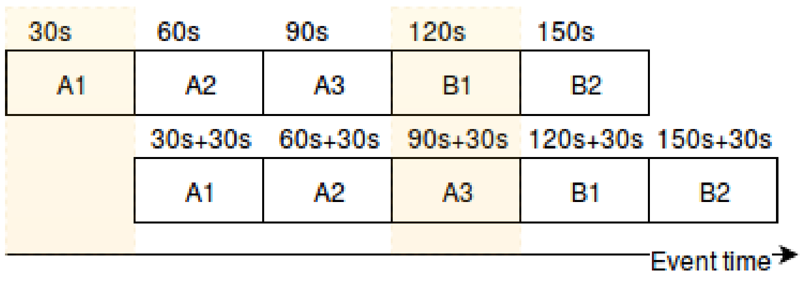

Fig. 2. Self-Join re-keyed stream with shifted duplicate to obtaib segment crossing stream.

The resulting stream can then be used to notify the driver about a toll, when the driver enters a new segment. One requirement of the benchmark is that the actual assessment, i.e. the point in time when the drivers account is debited is asynchronous to the notification. More precisely, the toll for segment s is assessed when the driver enters segment $s + 1$ (consider that there is no toll for the last segment).
Toll assessment involves adding every toll to a KTable that is keyed by vehicleId. In order to assess the toll for segment s when $s+1$ arrives, the tuple for $s+1$ needs to carry the point in time, the vehicle has entered s. The concrete realization of that table is covered in the respective section. However, this piece of information is incorporated into the segment crossing stream. Technically, this requires the operator to access the preceding data-item. A self-join as processed above is not applicable because it is unknown, when the vehicle has entered the preceding segment and thus is shift by a fixed amount of seconds would be useless. Instead a unwindowed, pairwise aggregation is applied to the stream, which is abused to connect a data-item with its predecessor. This approach assumes that elements are not out of order, i.e. a data-item that is sent to the streaming system at t_1 is processed later than an item t_0 if $t_1 > t_0$ if both have the same key. The required code snippet to incoprorate the timestamp of the predecessor is listed in 1.1. Note the call of *.toStream*() which transforms the changelog-stream to a record-stream.

Listing 1.1. Connecting data-item with predecessor.

```
.aggregateByKey(()-> new SegmentCrossing(),
(key, value, agg)-> new SegmentCrossing(value, agg.getTime())
.toStream();
```

The principle is visualized also in 3.

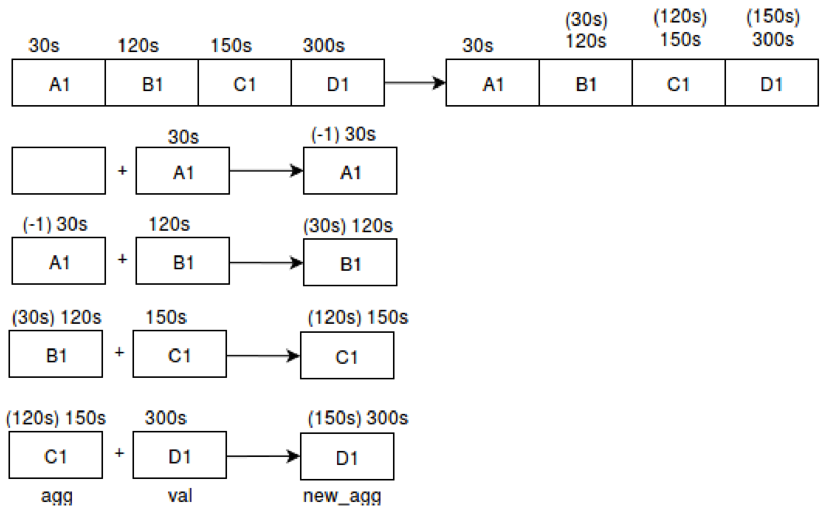

Fig. 3. Pairwise unwindowed aggregation to connect data-item with predecessor.

1.5 Daily Expenditure Request Stream

Input stream/topic	DailyExpenditure Topic
Implemented in	stream/DailyExpenditureRequestStreamBuilder
Output Type	Stream
Output key	DailyExpenditureRequest
Output value	Void

This stream reads from the respective Kafka topic and provides the daily expenditure requests as a stream. This stream is not materialized directly because it is re-keyed to create the corresponding response stream.

1.6 Daily Expenditure Response Stream

Input stream/topic	Daily Expenditure Request Stream, Toll History Table
Implemented in	stream/DailyExpenditureResponseStreamBuilder
Output Type	Stream
Output key	Void
Output value	DailyExpenditureResponse

In order to build this stream, the respective request stream (see above) is left-joined with the toll history table (1.1). For that purpose, the request stream must be re-keyed to $(xway, vehicle, day)$ tuples. It may be, that the toll table does not contain the respective entry. In that case a toll of 0 is responded.

1.7 Account Balance Request Stream

Input stream/topic	AccountBalanceRequest Topic
Implemented in	stream/AccountBalanceRequestStreamBuilder
Output Type	Stream
Output key	AccountBalanceRequest
Output value	Void

This stream reads from the respective Kafka topic created on the basis of the input tuples of query type 2. It is used as the trigger stream for the account balance response stream, which in turn responses the current account balance for the requested vehicle.

1.8 Account Balance Response Stream

Input stream/topic	Account Balance Request Stream, Current Expenditure per Vehicle Table
Implemented in	stream/historical/ AccountBalanceResponseStreamBuilder
Output Type	Stream
Output key	Void
Output value	AccountBalanceResponse

This stream represents the response stream of the type 2 queries. In order to return the desired account balance per vehicle, the request stream is firstly re-keyed to the corresponding vehicle id. The resulting stream is materialized and joined with the Current Expenditure per Vehicle Table and response messages are generated each.

1.9 Number of Vehicles Stream

Input stream/topic	PositionReport Stream
Implemented in	stream/NumberOfVehiclesStreamBuilder
Output Type	Stream
Output key	XwaySegmentDirection
Output value	NumberOfVehicles

This stream is supposed to count the number of distinct vehicles per $(xway, segment, direction)$ tuple in a time window of one minute without slide. Time refers to event time and is extracted from the first element of the position report value tuple. In order to recognize positions reports of the same car only once, the windowed position report is aggregated such that observed vehicle identifier are stored in a set. Subsequently, the set size is determined.

1.10 Latest Average Velocity Stream

Input stream/topic	PositionReport Topic
Implemented in	stream/LatestAverageVelocityStreamBuilder
Output Type	Stream
Output key	XwaySegmentDirection
Output value	AverageVelocity

This stream requires to emit every minute the latest average velocity of vehicles in the last five minutes per $(xway, segment, direction)$ tuple. For that reason, the position report can be consumed and does not have to be re-keyed. Generally, a time-window based aggregation is used to solve the problem. A sliding window of size 300 seconds and an advance of 60 seconds can be used to get every minute an aggregate of the latest five minutes. However, using the general sliding window, the first window and thus the first final aggregate would be emitted only after fives minutes. In order to be able to calculate tolls for minute 4, 3, 2 (in minute one there will never be a toll), we also need windows ending at these minutes. The Kafka streams API doesn't allow to specify an initial windowing start parameter, which then would have been set to -3. But Kafka allows to use custom time window implementations. These must, given a certain timestamp, return all times windows, in which this timestamp falls. The custom implementation that we use is adjusted, such that for all timestamps $t < 300s$, the first smaller windows are also returned (these which the timestamp falls into). The principle is illustrated in figure 4.

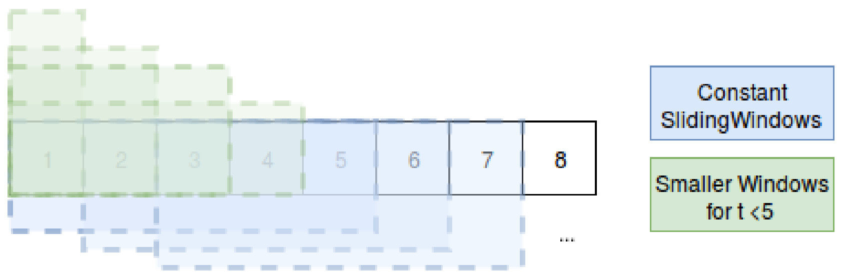

Fig. 4. Custom windowing in first five minues for LAV stream.

Because the stream DSL supports pairwise aggregation only, the average calculation is processed iteratively (see equation 1). For every tuple that falls into a window, the average is updated using the average value from before and a variable n that holds the number of processed elements.

$$\bar{X_n} = \frac{\sum_{i=1}^{n} X_i}{n} = \frac{\sum_{i=1}^{n-1} X_i + X_n}{n} = \bar{X_{n-1}} * \frac{(n-1)}{n} + \frac{X_n}{n} \tag{1}$$

1.11 Accident Detection Stream

Input stream/topic	PositionReport Stream
Implemented in	stream/AccidentDetectionStreamBuilder
Output Type	Stream
Output key	XwaySegmentDirection
Output value	Long

The current toll applicable for a certain $(xway, segment, direction)$ tuple depends on the occurrence of an accident. Two or more vehicles in the same segment, lane and position being stopped in at least four subsequent requests are considered to have an accident.
The stream is re-keyed to process $(xway, segment, direction, lane, positon)$ tuples instead and windowed using a window size of 4*30 seconds and a slide of 30 seconds. This is sufficient, because we can assume that every vehicle emits positions in a 30 seconds interval and thus we are able to catch all four consecutive occurrences in the stream. The slide is required, because we need to recognize four succeeding occurrences. The pairwise aggregation within these windows is processed such that when a new tuple arrives, the respective vehicle id is stored in a map representing the key and using the count of occurrences observed so far for that vehicle as the value. A filter operation will then analyze if the map contains at least two entries having counts equal to four.

The occurrence of an accident at segment s is not only of importance for vehicles that are currently in that segment, but also for vehicles moving in up to four segments upstream, i.e. these in segments $s-1, s-2, s-3, s-4$. The calculation of the toll for segment s at a certain minute requires the accident information for the downstream segments $s, s+1, s+2, s+3, s+4$. Also the vehicles in one of segments upstream have to be notified about that accident. Thus, it is useful to flatmap the occurrence of an accident at segment s to segments $s, s-1, s-2, s-3, s-4$.

1.12 Accident Notification Stream

Input stream/topic	Accident Detection Stream, Segment Crossing Position Report Stream
Implemented in	stream/AccidentNotificationStreamBuilder
Output Type	Stream
Output key	Void
Output value	AccidentNotification

The pure accident detection stream only informs about occurred accidents, but does not refer to the vehicles, that may be affected by that accident. The actual notification is processed by this stream, that generally joins the occurred accidents with segment crossing position reports, which is possible because both streams are keyed by $(xway, segment, direction)$ tuples. It has to be considered that the simple position report stream cannot be used as the input trigger stream, because the authors explicitly require that a position report can only trigger an accident notification, if the preceding position report of that vehicle has not been emitted from the same segment. The corresponding predicate is defined as $q.Seg \neq \overleftarrow{q}.Seg$

Vehicle drivers must be informed of that accident, if they are moving upstream the segments up to four segments. Again, because the accident detection stream has been flat-mapped such that accidents on segments s are declared as accidents on segments $s, s-1, s-2, s-3, s-4$, this works implicitly. Additionally, the authors require that the notification must happen only if the respective position report q was emitted no earlier than the minute following the minute when the accident occurred, and no later than the minute the accident is cleared. Thus, a position report at minute m will not trigger an accident notification of an accident occurred at the same minute (see figure 5). Technically, this is approached by modifying the event time of the position report source stream. One minute is subtracted from the position report stream (see figure 6).

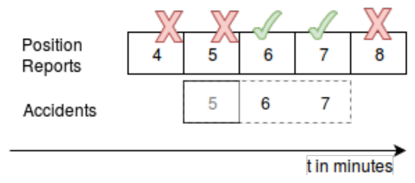

Fig. 5. Triggering of Accident Notification.

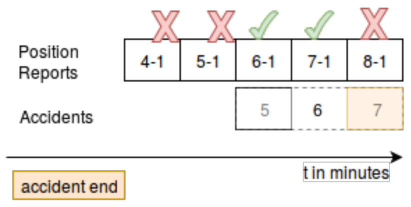

Fig. 6. Triggering of Toll Notification after re-keying trigger stream.

1.13 Current Toll Stream

Input stream/topic	Accident Detection Stream, Latest Average Velocity Stream, Number of Vehicles Stream
Implemented in	stream/CurrentTollStreamBuilder
Output Type	Stream
Output key	XwaySegmentDirection
Output value	CurrentToll

The current toll valid at a certain minute and a $(xway, segment, direction)$ tuple depends on the occurrence of an accident downstream to the current segment, the number of distinct vehicles of past minute and the latest average velocity of vehicles during the last five minutes. Because the toll valid for minute m requires all the before-mentioned values observed at minute $m-1$, we declare the calculated toll for the observations of minute m valid for minute $m+1$. Applying this trick, we do not need to change the event time of the source streams.
Firstly, the LAV stream is joined with the NOV stream and then the resulting stream is left-joined with the accident detection stream. This is necessary, because the accident stream does not emit values every minute, but only in case of an accident occurrence. A toll is generally not charged, if an accident occurred in any downstream $(-0..4)$ segment. Because accidents in any segment s have been flat-mapped to segments $s, s-1, s-2, s-3, s-4$, we can be sure to catch all of these, in case of an accident (see section 1.11). In a subsequent map-operation the toll formula is applied to the values and the current toll is emitted.

1.14 Toll Notification Stream

Input stream/topic	Current Toll Stream, Segment Crossing Position Report Stream
Implemented in	stream/TollNotificationStreamBuilder
Output Type	Stream
Output key	Void
Output value	TollNotification

The current toll stream does not address the vehicle drivers. The Linear Road specification requires the system to notify drivers about the current toll immediately after entering a segment. A driver must not be notified more than once about the same toll and within one segment only one toll can be charged. Thus, is is feasible to us Segment Crossing Position Report Stream as the triggering input stream. That stream is basically joined with stream of current tolls. Before that, a filter is applied, that checks if the car's position is not located an exit lane, which must not cause any tolls.