FlameStream: Model and Runtime for Distributed Stream Processing

Igor E. Kuralenok[1], **Artem Trofimov**[1,2], **Nikita Marshalkin**[1,2], **and Boris Novikov**[1,2]

[1]JetBrains Research, [2]Saint Petersburg State University

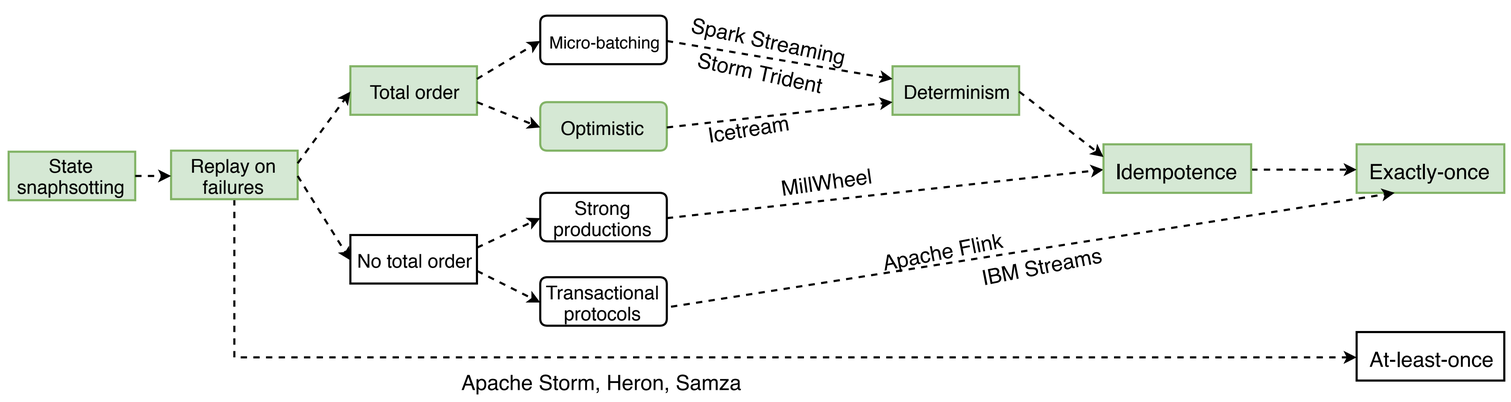

Figure 1: The roadmap of approaches for achieving exactly-once and at-least-once guarantees. Green elements indicate the path for our approach

Graph as a function

Determenistic exactly-once processing makes execution results predictable and repeatable as they are completely determined by input elements and a logical graph. This property is useful for preserving strong consistency in presence of failures and recoveries, because input replay does produce exactly the same results regardless the asynchronous nature of distributed systems.

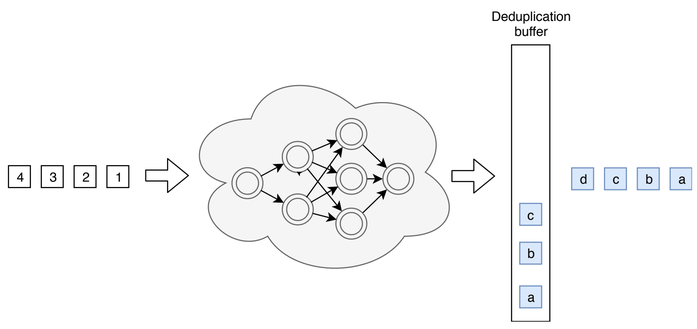

Figure 2: If computations are determenistic idempotance can be achieved by deduplication buffer at the end of a pipeline

Drifting state

Effectively, modern stream processing models allows managing an operation state in a form of

$$(in, state) \rightarrow (out, newState)$$

Even if systems do not provide such interface explicitly, it is implemented internally and exported via consistent state wrappers. We make such contract explicit by decomposing any stateful operation into two operations with a cycle.

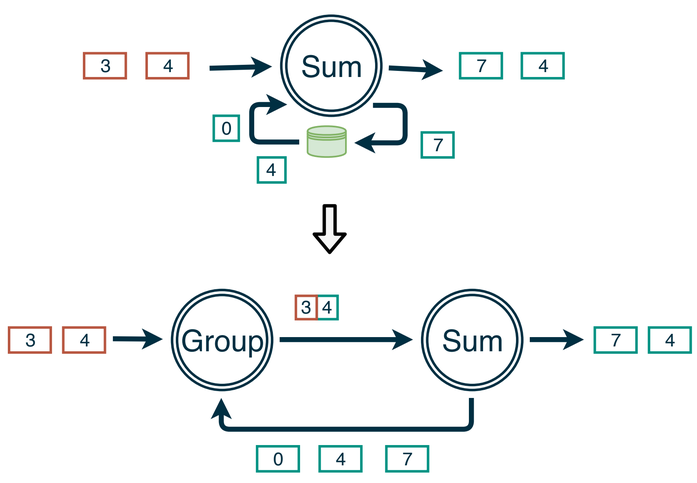

Figure 3: Any stateful transformation can be expressed by a combination of windowed grouping and stateless map operations

Any stateful pipeline can be constructed using a cyclic graph :

Map applies a user-defined function to the payload of an input item and returns a (possibly empty) sequence of data items with transformed payloads.

Grouping constructs a single item containing a set of consecutive items that have the same value of partition function. The maximum number of items that can be grouped is specified as a parameter $WindowSize$.

Groupings of different partitions are independent.

Ordering

To achieve deterministic results and to properly manage drifting state cycles, there is a need to impose strong processing order on data items.

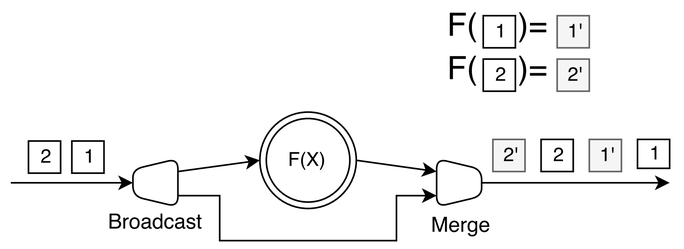

Figure 4: Data item with payload $1'$ is the derivative of the item with payload 1, according to operation F. The same is for items with payloads $2'$ and 2. After merge operation, the order between 1 and 2 is preserved. Furthermore, $1'$ follows 1, and $2'$ follows 2.

Optimistic out-of-order processing

To achieve system-wide ordering, it is sufficient to impose strong order only within grouping operations.

Grouping state has a well-defined simple structure that allows handling items speculatively without extra buffering.

It immediately processes input items as they were in-order. I case of reordering it is locally generates correct tuples and sends tombstones for incorrect ones.

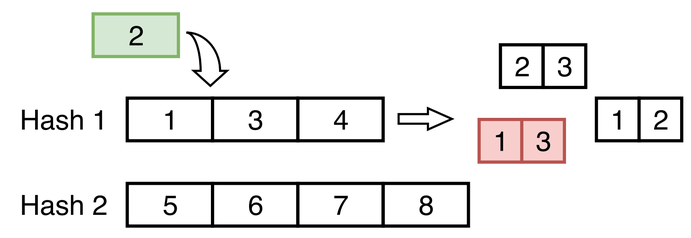

Figure 5: On arrival of out-of-order item, it is inserted in the correct location of the correcponding bucket. Valid elements containing new item are produced, and invalid ones are invalidated by sending a tombstone for it down the stream

Garbage filtering

In case of reordering invalid tuples are generated. In order to return only correct tuples to users there is a barrier at the end of the pipeline. It passes correct items and filters out incorrect ones.

Figure 6: Barrier collects elements in a buffer and removes item for which tombstones have arrived. Periodically it flushes buffer and returnes elements to the end-user. Punctuations used in OOP approach can be used as flush-trigger

Consistency

Model properties allows us to provide strong guarantees on processing results - exaclty-once semantics.

Total order and determinism allows us to filter out duplicates at the barrier by simply comparing elements timestamps with last evicted item.

Moreover state structures enables performing asynchronous snapshot of operations' states.

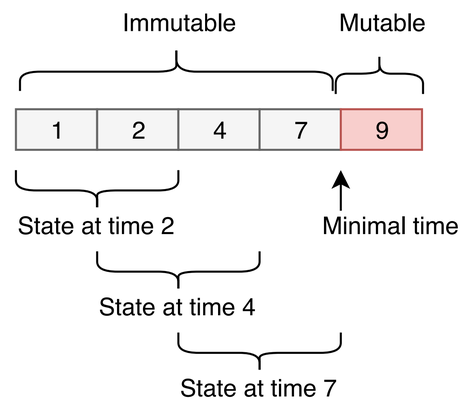

Figure 7: Grouping buckets can be divided into two parts: immutable and mutable. The separator is the time of the last received punction (we call it minimal time). State snapshot for any time before minimal time can be obtained by extracting window-sized sublist

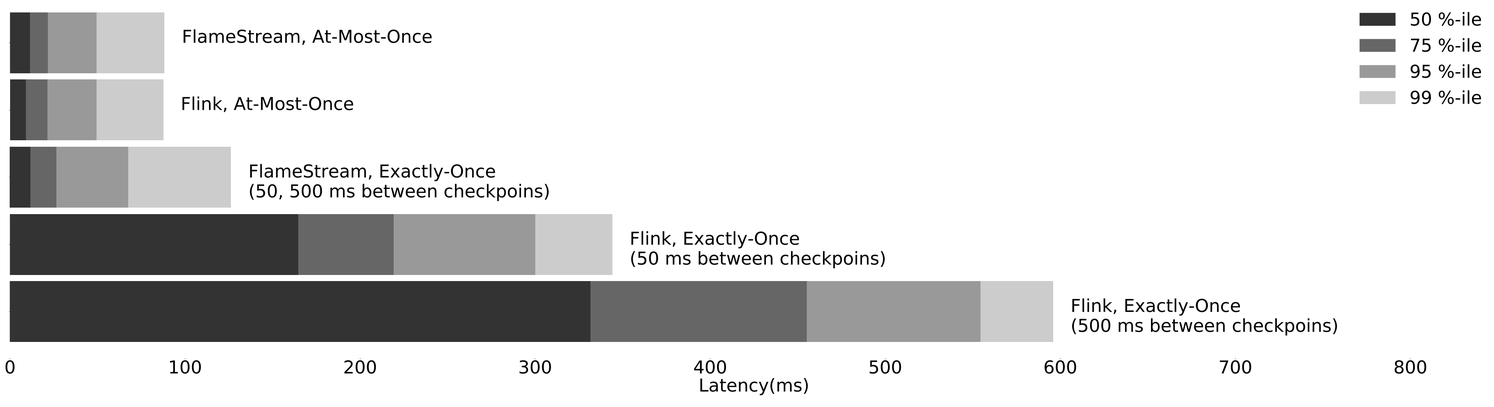

Figure 8: Comparison