Temporal Aggregation of Spanning Event Stream: A General Framework (Abstract)

Aurélie Suzanne, Guillaume Raschia, José Martinez

Temporal Aggregation of Spanning Event Stream: A General Framework

Aurélie Suzanne
aurelie.suzanne@ls2n.fr
Université de Nantes
Nantes, France
Expandium
Saint-Herblain, France

Guillaume Raschia
guillaume.raschia@ls2n.fr
Université de Nantes
Nantes, France

José Martinez
jose.martinez@ls2n.fr
Université de Nantes
Nantes, France

CCS CONCEPTS

• **Information systems → Stream management**.

KEYWORDS

data stream, spanning events, temporal aggregates, temporal database, window query

1 INTRODUCTION

The Big Data era requires new processing architectures among which the stream systems that have become well-known. Those systems are able to summarize infinite data streams with aggregates on the most recent data, allowing keeping a limited but meaningful piece of the initial stream. However, up to now, only point events have been considered and spanning events, which come with a duration, have been let aside, restricted to the persistent databases world only. In this paper, we propose a unified framework to deal with such stream mechanisms on spanning events.

2 EXAMPLE

Let us consider a network monitoring system where we want to evaluate the load of an antenna, with spanning transactions, e.g., phone calls, happening continuously. In a classical streaming system, the load would be based either on the start or end time of the event. With a spanning event stream the full event duration would be interpreted.

Figure 1 models a series of calls: events a_i as point events show only their ending time, while b_i's as spanning events show the full-call duration. We want to analyze the load of the antenna every five minutes showed by windows W_i's. With spanning events, window w_2 contains 4 events: $\{b_3, b_4, b_5, b_7\}$, while point events would find only 2 events for the same window $\{a_3, a_4\}$. This results in more accurate results for spanning events.

Of course, it would be possible to handle both start and end times for each event and then, mimic the spanning event behavior with current streaming systems. However, this would come with some problems to solve: how to deal, for instance, with long-standing events? Or lost messages (never-ending or un-started events)? Or even reversed bound messages (end, then start timestamps)? Moreover, natively modeling event duration allows detecting events which have no bounds in the window, like event b_7 crossing window w_2 and w_3 on Figure 1. Spanning event stream hence allows getting not only information about (dis)connections to/from the antenna, but also to the full connection information.

Spanning events stream hence allows to modelize events in a similar way than the way they were in the real world, providing more accurate results than point events stream. Furthermore reproducing spanning event behavior with current streaming systems would be tricky and time consuming, thus the need for a specific system. In this paper, we propose a unified framework to deal with such stream mechanisms on spanning events.

3 SPANNING EVENT STREAM

A spanning event stream contains a possibly infinite number of spanning events. Those events are composed of a transaction time, a valid time and some data specific to the event.

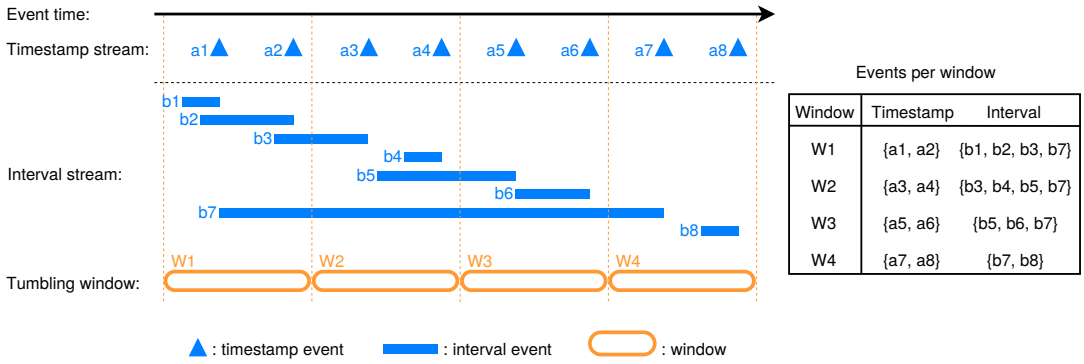
The transaction time corresponds to the moment when the event was received in the system, while the valid time is an interval corresponding to when the event was happening in the real world. Basically, a spanning event is a point event which valid time has been changed to an interval instead of a timestamp.

4 WINDOWING

A common solution to overcome the infinite stream problem with blocking operators, like aggregations, is to use windowing. Indeed, windows extract from the infinite stream finite sub-stream to feed the aggregation operators, which can then calculate results.

Those windows can be represented with intervals, and they are created with measures which set their size and frequency. Those measures can be independent of the stream, like a system clock stored on the streaming system server, or depend on the various parts of the events.

Windows are created with a function, which takes as input one or several measures and output intervals. Then, a predicate is used to fill them. It compares the event valid or transaction time with the window interval to tell if an event is in the window or not. For this, two predicates can be used, the first one comparing an event timestamp to a window interval, the second one using Allen's algebra to compare two intervals.



Events per window

Window	Timestamp	Interval
W1	{a1, a2}	{b1, b2, b3, b7}
W2	{a3, a4}	{b3, b4, b5, b7}
W3	{a5, a6}	{b5, b6, b7}
W4	{a7, a8}	{b7, b8}

Figure 1: End time vs. full-time events aggregation in a window-based stream system.

5 ASSIGNING SPANNING EVENTS TO WINDOWS

With the definition of spanning event stream and windows, the impact of spanning events can now be clearly studied. A first remark we should have is that normally a window is released as soon as its upper bound is reached, but spanning events duration makes to us no guarantee that we will receive an event before it ends (and in fact we might often receive it only when it ends). Hence we need to wait an additional time after the window has closed to actually release the window. We call this time a Time-To-Postpone, and it can as much be learned by the system as given by the user.

Once we know when to release the window, we still need to study precisely how spanning events impacts windows. For this we study here only the most common windows. Among them sliding windows is a good candidate. Those windows advance with the time, number of events received or data. Basically only the sliding windows using time are impacted as they use the valid time of the events to insert them into windows. Hence for them we need to use Allen predicates which compare two intervals together.

Then we can have some look at session windows. Those windows open with the arrival of an event and close when no event has been received for a certain duration. With spanning events we propose to adapt those windows to consider not only the end bound of the event, but the full event duration as being in the session. However we need to keep care of really long events, which could lead to two impossible choices: reopening windows which have already been released, or having overlapping session bounds. To avoid this, we limit the event size to the Time-To-Postpone which makes us sure to avoid clashes between the sessions which need to be created and those which have been released.

6 EXPERIMENTS

Eventually, we validate the soundness of our new framework with a set of experiments, based on a straightforward implementation. In those experiments we show that not only spanning events are more accurate than point events, but the Time-To-Postpone is also a good measure to deal with event duration. This Time-To-Postpone has on top only a limited impact on throughput. When comparing throughput of points events and spanning events on a real-like data set, we acknowledge a loss with spanning events which is, however, low and can be compensated with further optimization techniques.

7 CONCLUSION

In conclusion, spanning events can be used in streaming system, with a gain in accuracy for aggregation results. Indeed, they allow us to modelize the events the same way they were in the real world without making any truncation. However using spanning events implies that we modify the way we use windows. For this, the bound function needs to be adapted, in particular for session windows to acknowledge for the full event duration. Then, the insertion predicate needs to be adapted to compare two intervals together instead of asserting that a timestamp is in an interval. Finally, we need to postpone the window release to make sure we have received all the events before releasing a window. For this, the Time-To-Postpone answer to the question, but only partially, as we do not allow to go further in past than the Time-To-Postpone.