Temporal Aggregation of Spanning Event Stream: A General Framework

Aurélie Suzanne[1,2(✉)], Guillaume Raschia[1], and José Martinez[1]

[1] Université de Nantes, 1 Quai de Tourville, 44035 Nantes, France
{aurelie.suzanne,guillaume.raschia,jose.martinez}@ls2n.fr
[2] Expandium, 15 Boulevard Marcel Paul, 44800 Saint-Herblain, France

Abstract. The Big Data era requires new processing architectures among which the stream systems that have become very popular. Those systems are able to summarize infinite data streams with aggregates on the most recent data. 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 an unified framework to deal with such stream mechanisms on spanning events. To this end, we formally define a spanning event stream with new stream semantics and events properties. We then review and extend usual stream windows to meet the new spanning events requirements. Eventually, we validate the soundness of our new framework with a set of experiments, based on a straightforward implementation, showing that aggregation of spanning event stream is providing as much new insights on the data as effectiveness in several use cases.

Keywords: Data stream · Spanning events · Temporal aggregates · Temporal database · Window query

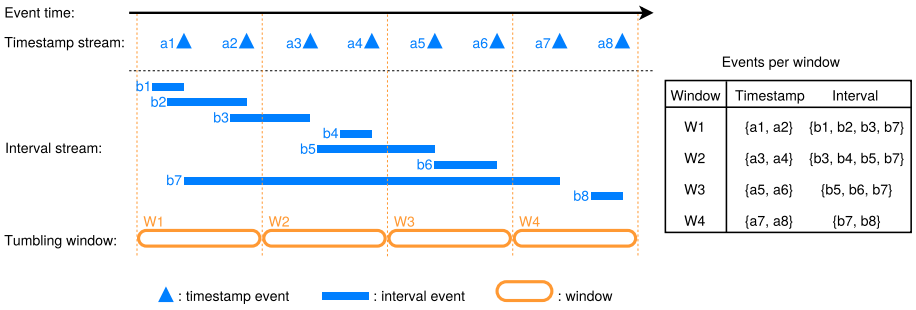
1 Introduction

Data stream processing has been widely studied in recent years [6,12], and many industrial systems are now using it [13] with applications such as monitoring systems for networks, marketing, transportation, manufacturing or IoT systems. Retrieving useful insights from this continuously produced data has hence become a key issue. A common way to process those streams is to aggregate events with respect to the instant they occurred or arrived in the system.

Time is a first class dimension for stream events as it determines how they will be aggregated, but up to now a strong assumption was made about point events, leaving aside spanning events. 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 on the 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 show their reception time, while b_i's show the full call duration. We want to analyse the load of the antenna every

S. Hartmann et al. (Eds.): DEXA 2020, LNCS 12392, pp. 385–395, 2020.
https://doi.org/10.1007/978-3-030-59051-2_26



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}

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

5 min showed by W_i's. With spanning events, stream intervals b_3 and b_5 span over resp. windows $\{w_1, w_2\}$ and $\{w_2, w_3\}$, while their matching timestamps a_3 and a_5 are uniquely assigned to resp. w_2 and w_3. Natively modeling event duration also allows to detect events which have no bounds in the window, like event b_7 crossing window w_2 and w_3. Spanning event stream (SES) hence allows to get not only information about (dis)connections, but also to the full connection information, providing more accurate results than point event stream (PES).

To be able to provide such results, interval comparison predicates, inspired by Allen's algebra [1], need to be properly set up to allow assignment of spanning events to windows. Furthermore, as a side-effect of spanning events, past windows can be affected by fresh new events and this without any delay in the stream: on Fig. 1, b_7 not only impacts w_4, but also past windows w_1, w_2, and w_3. Those simple observations motivate the need for a close review of the many windows types, and a new window-based aggregation framework to address SES.

The rest of this paper is organized as follows: Sect. 2 focuses on the formal requirements to elaborate the framework. Section 3 discusses adaptation of common windows to spanning events. We propose, in Sect. 4, a straightforward implementation of the framework. Finally, we review prior works in Sect. 5, before the general conclusion in Sect. 6.

2 Problem Analysis and Definitions

2.1 About Time

Following the dominant view point, one defines the time domain as an infinite, totally ordered, discrete set $(\mathbb{T}, \prec_{\mathbb{T}})$, with units coined *chronons* [2]. As in temporal databases [2], two dimensions can be used: *valid time* and *transaction time*, being resp. the lifespan of an event in the real world and in the system. In streaming systems, transaction time is usually reduced to the start time point.

In this bi-temporal model, intervals are required to represent valid time. As an adjacent series of time points in $\mathbb{T}$, they are entirely defined by their lower and upper bounds, as pairs $(\ell, u) \in \mathbb{T} \times \mathbb{T}$ with $\ell \prec_{\mathbb{T}} u$. We denote by $\mathbb{I} \subset \mathbb{T} \times \mathbb{T}$ the

set of time intervals. For any t in $\mathbb{I}$, $\ell(t) \in t$ and $u(t) \notin t$ are resp. the lower and upper bounds of interval t, such that a chronon can be represented by $[c, c+1)$. Two intervals can be compared with the 13 Allen's predicates [1] which define precisely how the bounds compare to each other.

2.2 Spanning Event Stream

In this article, we extend the stream concept to incorporate spanning events with a lifespan as valid time. As a consequence, it is required to distinguish valid time (interval) and transaction time (point) of an event. Since data stream applications require near-real-time computation, valid and transaction time should be ideally connected, e.g., a phone call is recorded as soon as it ends.

We define a SES in Eq. 1, where Ω is any composite type that brings the content of each event $e \in S$; t is the valid time interval; and τ is the transaction timestamp of e. We denote by $t(e)$ and $\tau(e)$ those values for an event e.

$$S = (e_i)_{i \in \mathbb{N}} \quad \text{with } e_i = (x, t, \tau) \in \Omega \times \mathbb{I} \times \mathbb{T} \tag{1}$$

We assume that we record events when they finish, in a no-delay stream setting, such that $u(t(e)) = \tau(e)$. In the following, we denote by $S(.)$ any projection of stream S on one or more of its 3 components. For instance, $S(t)$ refers to the sequence of valid time intervals of all the events. We also denote τ_i the transaction time of the ith event in S, ie., $\tau_i = \tau(e_i)$.

2.3 Temporal Windowing

In data stream processing, data are transient and queries persistent, meaning that queries are continuously re-evaluated as data arrive. Blocking operators, such as aggregates, are a challenge for those *continuous queries* as they require to scan all the data set before producing the first answer [6]. A popular way to bypass this issue is to operate on a bounded sub-stream given by a *window* [6]. Indeed, closing a window unblocks the operation and an answer can be given with respect to the events "inside" the window. A common practice is to define infinite family of windows such like "each hour". Many window flavors exist [4,6,12], and in this paper, we propose a new categorisation for those windows.

Measures. Measures are useful to set up the shape and/or frequency of windows in a window family. Each measure is time related, to provide finite window bounds. We denote the measure set by $\mathcal{M} = \{\mathbb{T}\} \cup \{S(t)\} \cup \{S(\tau)\} \cup \{S(x, \tau)\}$ and consider it as extensive. Measures are the following:

- **Stream independent with a wall-clock time** $\mathbb{T}$: a system clock is used. Opening and/or closing windows are then independent from the stream.
- **Stream dependent with a stream projection**:
  - **Valid-time** $S(t)$ uses the valid time of events, e.g., sessions where bounds depend on the traffic flow.

- **Shape** $S(\tau)$ uses the transaction time of the events, more common example is *Count-based* which counts events arriving in the stream.
- **Data** $S(x, \tau)$ uses the data of events and orders them with their transaction time. Most common types are: *Delta-based* which uses a data part [6], e.g., a transaction counter; *Punctuation-based*: where window bounds are sent in the stream as special events [6].

Formal Definition. Formally, a temporal window is a regular time interval. A family of windows is $W = (w_k)_{k\in\mathbb{N}}$, $w_k \in \mathbb{I}$ ordered by increasing $\ell(w_k)$ or $u(w_k)$. $W \in \mathcal{W}$ is the set of windows families. We propose a couple (F_{bounds}, P_{insert}) to compute any window family.

$F^n_{bounds} : \mathcal{M}^n \to \mathcal{W}$, defines the bounds of the window. It uses one or several measures, and outputs a set of intervals (as shown in Table 1).

Table 1. Examples of windows created with F_{bounds} and F^2_{bounds}

$\mathcal{M}$	Example of $\mathcal{W}$	$\mathcal{M}^2$	Example of $\mathcal{W}$
$\mathbb{T}$	15 min each 5 min	$\mathbb{T}, S(\tau))$	5 min each 100 events
$S(t)$	session	$\mathbb{T}, S(x, \tau))$	50 transactions each 5 min
$S(\tau)$	100 events each 10 events	$S(t), S(\tau)$	100 events each session
$S(x, \tau)$	50 transactions each punctuation	$S(t), S(x, \tau)$	Session each punctuation

$P_{insert} : \mathbb{I}^2 \to \mathcal{B}$, determines event belonging to a window with an Allen-like predicate. It takes as input two intervals: window bounds and event valid time, and outputs a Boolean. We define two specializations:

- $\mathrm{P}_{\triangle}$: True if $\ell(w) \leq u(e) < u(w)$ else False, deals with point events by using a chronon and a window interval
- $\mathrm{P}_{\cap}$: True if $u(e) > \ell(w) \wedge \ell(e) < u(w)$ else False, asserts if an event has at least one chronon in a window or not

Common Windows Review. We will now review window types which are often used in the literature [6,12]. Within PES, event valid time is a point, and hence for all those window types, $P_{insert} = \mathrm{P}_{\triangle}$.

Sliding window: window is defined by ω the range or size of the window, and β the step which sets up the delay between two successive windows. Most common measures are wall-clock time: $F_{bounds}(\mathbb{T}) = ((i, i+\omega) : i \mod \beta = 0)_{i\in\mathbb{T}}$ and count-based: $F_{bounds}(S(\tau)) = ((\tau_i, \tau_{i+\omega}) : i \mod \beta = 0)_{i\in\mathbb{N}}$

Tumbling window: tumbling windows can be seen as a specialization of sliding windows where $\omega = \beta$, meaning that only one window is open at a time.

Session window: a session is defined as a period of activity followed by a period of inactivity. Parameter ε gives the inactivity threshold time range. Session window family is $F_{bounds}(S(t)) = ((\tau_i, \tau_j) : i < j \wedge (\tau_i - \tau_{i-1} \geq \varepsilon \vee \tau_i = \tau_0) \wedge \tau_j + \varepsilon \leq \tau_{j+1} \wedge \forall p \in [\![1, j-i]\!], \tau_{i+p} - \tau_{i+p-1} < \varepsilon)_{i,j\in\mathbb{N}}$.

3 Assigning Spanning Events to Temporal Windows

3.1 Adaptation of Point Events Windows to Spanning Events

We now detail modifications that using SES implies to the previously defined windows. This impact depends on the measures used and we will review of all common windows in a mono-measure context, $F_{bounds} : \mathcal{M} \rightarrow \mathcal{W}$.

Within PES, we saw that we can always use $\mathrm{P}_{\triangle}$ for P_{insert}. With SES we have to take into account intervals. P_{insert} depends on the window bounds definition F_{bounds}. If it used only transaction time we can keep $\mathrm{P}_{\triangle}$, otherwise it needs to be modified to use an Allen's predicate (see Table 2).

Table 2. Most common spanning event window definition with (F_{bounds}, P_{insert})

Window type	Point		Spanning	
	$\mathcal{M}$	P_{insert}	$\mathcal{M}$	P_{insert}
Time sliding/tumbling	$\mathbb{T}$	$\mathrm{P}_{\triangle}$	$\mathbb{T}$	$\mathbf{P_{Allen}}$
Stream shape sliding/tumbling	$S(\tau)$	$\mathrm{P}_{\triangle}$	$S(\tau)$	$\mathrm{P}_{\triangle}$
Stream data sliding/tumbling	$S(x, \tau)$	$\mathrm{P}_{\triangle}$	$S(x, \tau)$	$\mathrm{P}_{\triangle}$
Session	$S(t) = S(\tau)$	$\mathrm{P}_{\triangle}$	$\mathbf{S(t)}$	$\mathbf{P_{Allen}}$

Hence, we claim that among the most popular windows, stream shape and data sliding/tumbling windows can be used straightforwardly with SES. Time-based sliding/tumbling and session windows must conversely be extended.

3.2 Time-to-Postpone

When dealing with a SES where events are released only once ended, lifespan yields two problems: (1) the system should be able to wait for (expected) event completion before closing any window; and (2) long-standing events may be assigned to multiple windows. Figure 1 shows an example of such duration constraint with, for instance, the event b_7 released in window w_4, but assigned to w_1, w_2, and w_3 as well. Therefore, working on a valid time interval requires to postpone the release date of the aggregates in order to accept events that started in, or before the window. We call this waiting time the Time-To-Postpone (TTP).

Of course, TTP is a patch to overcome limitations of exact aggregate computation for temporal windows, and can lead to approximate results since events may be ignored and aggregates already released. Advanced TTP techniques deserve to be explored in future works to leverage those approximations.

3.3 Time-Based Sliding and Tumbling Windows

Event assignment to a window depends on how their lifespan compare: an event e is assigned to window w if $P_{Allen}(t(e), w)$, with P_{Allen} any Allen-like predicate

searching for event in the window. Release time of the window τ_R depends on the TTP parameter δ, satisfying $\tau_R \geq u(w) + \delta$. A full overview of the needed changes to deal with SES is presented in Table 3.

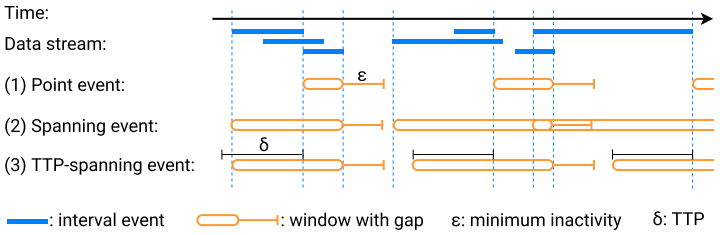


Fig. 2. 3 strategies for session windows.

Table 3. Comparison between PES and SES for sliding and session windows

Properties	Time sliding/tumbling		Session	
	Point	Spanning	Point	Spanning
Parameters	(ω, β)	$(\omega, \beta, P_{Allen}, \delta)$	$(\varepsilon, \omega_{\max})$	$(\varepsilon, \omega_{\max}, \delta)$
New window	$i \mod \beta = 0,\ i \in \mathbb{T}$		$\nexists i \in \mathbb{N}, \mathrm{P}_{\triangle}(\tau_i, \varepsilon(w))$	$\nexists i \in \mathbb{N}, \mathrm{P}_{\cap}(\Lambda(e_i), \varepsilon(w))$
$\ell(w)$	$i,\ i \in \mathbb{T}$		$(\min\{\tau(e)\})_{e \in S_w}$	$(\min\{\lambda(e)\})_{e \in S_w}$
$u(w)$	$i + \omega,\ i \in \mathbb{T}$		$(\max\{\tau(e)\} + 1)_{e \in S_w}$	$(\max\{u(t(e))\})_{e \in S_w}$
P_{insert}	$\mathrm{P}_{\triangle}$	P_{Allen}	$\mathrm{P}_{\triangle}$	$\mathrm{P}_{\cap}$
Release time	$u(w)$	$u(w) + \delta$	$u(w) + \varepsilon + 1$	$u(w) + \varepsilon + \delta$

3.4 Session Windows

In session windows, each received event either enters in the current window, or creates a new one. The upper bound of a session depends only on the end of the assigned events $u(w) = \max\{u(t(e))\}_{e \in S(w)}$. As for the lower bound, it must be chosen carefully. With PES, one can definitely decide the start of a session window as a fresh new event arrives. With SES, we must live-adjust this lower bound, since it requires to define an instant from a set of spanning events.

We model this problem as $\ell(w) = \min\{\lambda(e)\}_{e \in S_w}$, where $\lambda : S \to \mathbb{T}$ is a choice function that gives a reference point for an event. For a PES, this function is written as $\lambda(e) = u(t(e)) - 1$, using only the end bound as shown in strategy (1) in Fig. 2. When considering the lifespan of the event, a first estimate of the lower bound is $\lambda(e) = \ell(t(e))$, such that the event is starting in (and covers) the session. However, this strategy can lead to problems, as illustrated with strategy (2) where the last event leads to either re-opening or creating a session, causing impossible situations since the aggregate has already been released, and session

overlaps are not allowed. To overcome this problem, we apply the TTP parameter δ to restrict back-propagation of the update; with $\lambda(e) = \max(\ell(t(e)), \tau(e) - \delta)$ as with strategy (3) which makes the long-standing event problem disappear.

Release of the session depends on the minimum inactivity period ε and the TTP δ, satisfying $\tau_R \geq u(w) + \varepsilon + \delta$. Several sessions can be active at the same time, and long events can yield to merge sessions. Table 3 details the adaptation between PES and SES sessions. We use $\varepsilon(w) = (u(w), u(w) + \varepsilon)$ as the inactivity interval, and $\Lambda(e) = (\lambda(e), u(t(e)))$ as the re-considered event with the TTP.

4 Experiments

Experimental Setup. In this series of experiments, data is not received at specific instant based on machine clock, but better "as fast as possible."

Data Set. We use 2 kinds of data sets: *Generated data set* allows fine-grained synthesis of SES with configurable parameters: event size, session duration, and inactivity. For each chronon, an event is created, which can be canceled with session creation. Each event size is generated by a normal distribution ($\mu = 100$, $\sigma = 10$) around the event size parameter. The generated set is 200K events. *SS7 data set* replays real-world-like data coming from a telephony network, assembling 1 min of communication with 3.2M events.

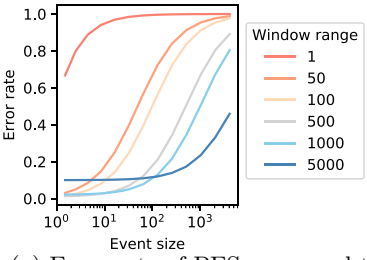
Aggregates. Aggregation in all experiments is a multi-measure of three aggregate functions: count, sum, and max.

Setup. All experiments were executed with an Intel(R) Core(TM) i7-8650U CPU @ 1.90 GHz with 16 GB RAM running under Linux Debian 10. Implementation is done in modern C++, using a single core.

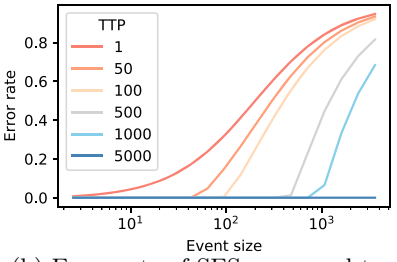
Implementation. Implementation uses an event-at-a-time execution. For PES windows an unique FIFO queue is used, with new events added and old ones removed each time the window is released. With SES, such an implementation is not possible. Instead, events pointers are stored in a *bucket* per window.

Results. All the scenarios chosen in this series of experiments have been motivated by industrial requirements, especially in the field of telecommunication.

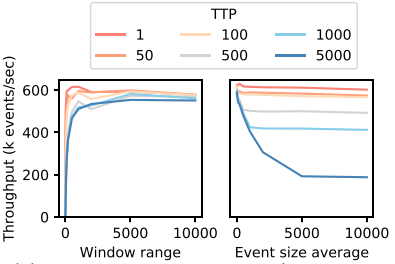
Time-Based Windows. The predicate used for event assignment is $\mathrm{P}_{\cap}$. As expected the error rate between PES and SES increases with the event size, and decreases with window range (see Fig. 3a). This validates the soundness of using PES but also the urge to choose wisely the window range. When using an Oracle, which knows all the stream, we can validate the need for a TTP within SES, which should be chosen accordingly to the event size (see Fig. 3b). Concerning the throughput, TTP has a restricted impact when the window range evolves (see Fig. 3c), which is not the case for increasing events size. SES yield to many duplicates among the windows, which comes with a cost in throughput. For increasing duplications, the throughput goes down, but it stays roughly the same



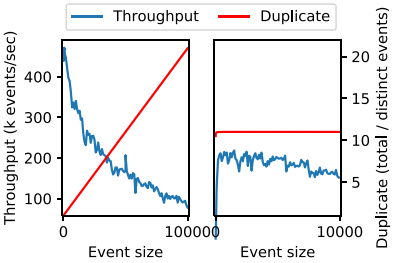
(a) Error rate of PES compared to SES (tumbling $\delta = 10000$)



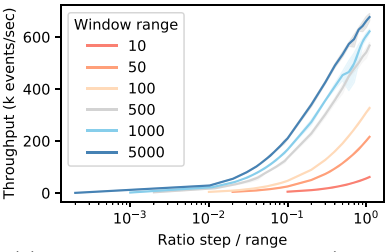
(b) Error rate of SES compared to oracle (tumbling $\omega = 200$)



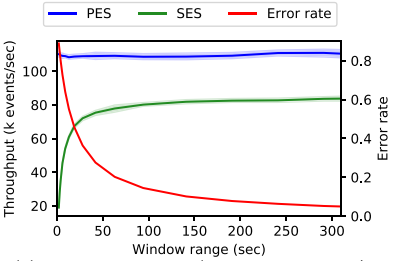
(c) Impact of TTP (tumbling $avg(|t(e)|) = 500$ vs $\omega = 500$)



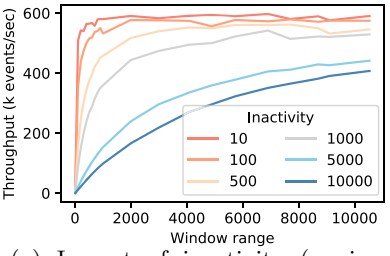
(d) Duplication (tumbling $\omega = 500$ vs. fixed duplicate factor)



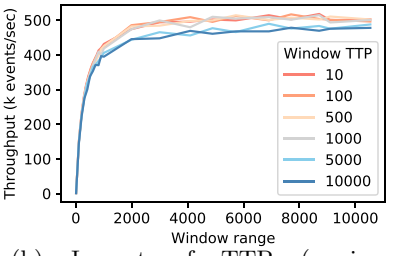
(e) Impact of step and range (sliding $avg(|t(e)|) = 500$, $\delta = 1000$)



(f) SS7 data set (sliding $\beta = \omega/5$, $avg(|t(e)|) = 15sec$, $\delta = 2h30$)



(g) Impact of inactivity (session $avg(|t(e)|) = 10$, $\delta = 500$)



(h) Impact of TTP (session $avg(|t(e)|) = 10$, $\varepsilon = 500$)

Fig. 3. Temporal windows metrics

for increasing window range with the same duplication rate (see Fig. 3d). Duplication induced by overlapping in sliding windows also has a strong impact on throughput (see Fig. 3e). Concerning real-world applications, on a naïve implementation the throughput of SES is only 30% slower than PES, and the error rate is still around 20% for 1 min windows, as we can see on Fig. 3f.

Session Windows. As shown on Figs. 3g and 3h, reducing window size as a negative impact on throughput. This is in accordance with time-based windows and refers to how often aggregates should be computed. Figure 3g highlights the impact of inactivity duration on throughput which is quite high. When maintaining inactivity periods at a same level, we can observe that TTP has a small influence on the throughput (see Fig. 3h).

Summary. This series of experiments shows that our framework is consistent with the all required assumptions for window-based aggregation on SES, and in particular the TTP. It then deserves to be pushed further in order to gain efficiency and completely meet the industrial requirements.

5 Related Work

The work done in this paper elaborates on previous work on data stream processing and temporal databases.

Window Aggregation in Data Stream Processing. Windowing is a common technique and a common categorization of window characteristics is given in [4,12] with CF, FCF and FCA classes. Depending on those characteristics, several optimisations techniques have been proposed, such as sub-aggregating the input stream, and using aggregate tree indexes [4,12]. Studied in the context of PES, we believe that the extension of such methods would be of great interest to fasten window-based aggregation of SES. Nevertheless, the window approach has been criticized for its inability to take into account delayed or out-of-order streams. Some methods have been proposed to fix the delay issue, among which an allowed waiting time (TTP in this paper), the use of punctuation in the stream [7], or even the generation of heartbeats [11].

Temporal Databases. Queries in a temporal database can be of various forms [2]. Among those, sequenced queries, where the query spans over a time range, are close to our temporal window-based aggregates. However, pure sequenced query is resource demanding and barely evaluated with a one-pass algorithm [8]. Several methods were proposed to evaluate sequenced queries, mainly with graph or indexes [2,9], but as an open issue, only few market databases implement them [3,5,10]. Temporal aggregates on spanning events have not been widely studied. In [14], the authors combine windows and full history with a fine-to-coarse grain along the timeline, using SB-tree structure to index events and evaluate the queries. However, the approach is out-dated w.r.t. recent advances in window-based stream processing and temporal databases.

6 Conclusion and Future Work

This paper aimed to introduce a brand new consideration in stream data with the integration of spanning events. To do so, we first introduced notions common to temporal databases with a valid time range and a transaction time point for events in a bi-temporal model. Then, a common solution to overcome the infinite stream problem with blocking operators is to use windowing. To that extent, we conducted a careful review that yielded to a new categorization of usual measures and the definition of a pattern (function, predicate) to define every popular window family as well as the forthcoming ones. We showed that, among the real-life window families, only time-based sliding/tumbling and session windows need to be adapted to handle spanning events.

Among those changes, we introduced pairwise interval comparison, as for Allen's algebra, for event assignment to windows. We also had to define a Time-To-Postpone parameter that allows for long-standing events to be properly assigned to past windows. In the experiments, we showed that spanning events can be processed by a stream system. We demonstrated that our framework is effective for fixing PES errors. We also pointed out some behaviors, like assignment duplication of events, which is a great challenge for real-life applications.

As future work, we anticipate that the implementation should use more advanced techniques to share parts of computations among windows. Delay also should be studied in more details. Finally, extension of the aggregation, such as new operations like grouping or filtering, should also be considered with the ultimate goal of making the system fully operational in real-world conditions.

References

1. Allen, J.F.: Maintaining knowledge about temporal intervals. Commun. ACM **26**(11), 832–843 (1983). https://doi.org/10.1145/182.358434
2. Böhlen, M.H., Dignös, A., Gamper, J., Jensen, C.S.: Temporal data management – an overview. In: Zimányi, E. (ed.) eBISS 2017. LNBIP, vol. 324, pp. 51–83. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96655-7_3
3. Böhlen, M.H., Dignös, A., Gamper, J., Jensen, C.S.: Database Technology for Processing Temporal Data. In: 25th International Symposium on Temporal Representation and Reasoning, TIME 2018 (2018, Invited Paper). https://doi.org/10.4230/lipics.time.2018.2
4. Carbone, P., Traub, J., Katsifodimos, A., Haridi, S., Markl, V.: Cutty: aggregate sharing for user-defined windows. In: CIKM 2016, pp. 1201–1210 (2016). https://doi.org/10.1145/2983323.2983807
5. Dignos, A., Glavic, B., Niu, X., Bohlen, M., Gamper, J.: Snapshot semantics for temporal multiset relations. Proc. VLDB Endow. **12**(6), 639–652 (2019). https://doi.org/10.14778/3311880.3311882
6. Gedik, B.: Generic windowing support for extensible stream processing systems. Softw. - Pract. Exp. **44**(9), 1105–1128 (2014). https://doi.org/10.1002/spe.2194
7. Kim, H.G., Kim, M.H.: A review of window query processing for data streams. J. Comput. Sci. Eng. **7**(4), 220–230 (2013). https://doi.org/10.5626/JCSE.2013.7.4.220

8. Moon, B., Lopez, I.F.V., Immanuel, V.: Efficient algorithms for large-scale temporal aggregation. IEEE Trans. Knowl. Data Eng. **15**(3), 744–759 (2003). https://doi.org/10.1109/TKDE.2003.1198403
9. Piatov, D., Helmer, S.: Sweeping-based temporal aggregation. In: Gertz, M., et al. (eds.) SSTD 2017. LNCS, vol. 10411, pp. 125–144. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-64367-0_7
10. Snodgrass, R.T.: A case study of temporal data. Teradata Corporation (2010)
11. Srivastava, U., Widom, J.: Flexible time management in data stream systems. In: PODS 2004, pp. 263–274 (2004). https://doi.org/10.1145/1055558.1055596
12. Traub, J., et al.: Efficient window aggregation with general stream slicing. In: EDBT 2019, pp. 97–108. OpenProceedings (2019). https://doi.org/10.5441/002/edbt.2019.10
13. Yang, P., Thiagarajan, S., Lin, J.: Robust, scalable, real-time event time series aggregation at Twitter. In: SIGMOD 2018, pp. 595–599 (2018). https://doi.org/10.1145/3183713.3190663
14. Zhang, D., Gunopulos, D., Tsotras, V.J., Seeger, B.: Temporal aggregation over data streams using multiple granularities. In: Jensen, C., et al. (eds.) EDBT 2002. LNCS, vol. 2287, pp. 646–663. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45876-X_40