Framing the Question: Detecting and Filling Spatial-Temporal Windows

James Whiteneck, Kristin Tufte, Amit Bhat, David Maier, and Rafael J. Fernández-Moctezuma
Department of Computer Science
Portland State University
P.O. Box 751, Portland, OR
+1 (503) 725-2406
{whitenej, tufte, amitb, maier, rfernand}@cs.pdx.edu

ABSTRACT

We propose a new mechanism, which we term *frames*, for data-dependent windows. In contrast to traditional timestamp-based windows, frames represent just the boundary of a window and can be filled with data from secondary streams or historical data. Examples show how frames can be useful in network and sensor monitoring applications. We present frame definition and implementation in one dimension, discuss extension to multi-dimensional frames, and identify issues for further investigation.

Categories and Subject Descriptors
D.3.3 [**Database Management**]: Systems – *query processing*

General Terms
Algorithms, Design, Management, Performance

Keywords
Spatial-temporal windows, Data-stream management

1. INTRODUCTION

Data Stream Management Systems (DSMS) [2,3,7,9,10] process unbounded data streams by partitioning those streams into windows based on timestamps. Such windows have two distinguishing properties. First, they are based only on timestamps or tuple-counts hence cannot reflect spans of interest determined by data values in the stream. Second, these windows partition and analyze the same stream, so that the result associated with a window (typically an aggregate) is derived from the values in the same stream over which the windows are defined. However, many applications have queries that require windows to be defined dynamically based on properties of the data values. In this paper, we consider a variation on windows called *frames*—a mechanism for data-driven windows. A frame defines the extent or boundary of window, and can be used against the same stream or another stream to construct window content, or used on its own to monitor a condition.

Consider a network router that emits a stream of summary reports, one every 20 seconds, including among other things, the percentage of dropped packets. We want to know periods where a router has a packet loss above 0.3% for at least 3 consecutive reports (1 minute). The system administrators wish to correlate these periods of high loss with related data to understand the cause of the high packet loss. These results cannot be produced using traditional stream queries due to the nature of the window definition (packet loss above 0.3% for one minute) and the desired correlation with data from a different source. In contrast, frames support such behavior and will not report (unnecessary) results during periods of low loss.

A frame will demark the start and end of such a period of high packet loss. This frame could be combined with the same stream to get average packet volume (both count and combined length) for each such period. Or the frame could be used to select data from a stream of Border Gateway Protocol (BGP) messages to see if the router had a large number of routing revisions to deal with in the period. Or, the frame simply could trigger an alert to a system manager to investigate further.

We begin by describing our design and implementation of frames in one dimension, present possible extensions to two-dimensional frames, and then discuss issues for further investigation.

2. FRAMES IN ONE DIMENSION

We have observed a need to identify epochs of interest in data streams. These periods have similarities with windows, as they describe a stream subset of interest, but unlike windows, they are discovered by inspecting stream content rather than partitioning by time. We call such data-derived windows *frames*. A frame defines window boundaries based on data values.

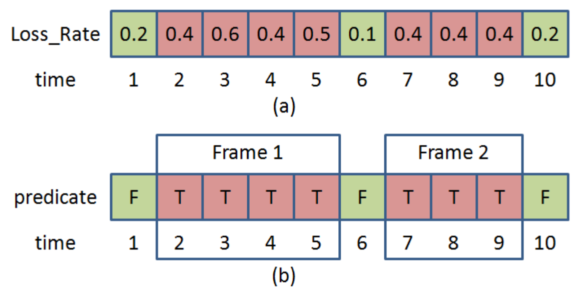
The specification for a frame consists of two parts—a predicate and a (time) duration: "Predicate P holds for duration at least D". P is tuple-wise predicate over the stream. D can be expressed as either a number of tuples or a minimum time interval. The frame specification for the router example is: "`tuple.Loss_Rate > 0.3` for at least 3 reports". We take the maximal period satisfying the condition. In the example, if there were 5 consecutive high-loss reports in a row, we would report it as a single frame instance (rather than three instances of length 3).

Our implementation of frames consists of an Apply operator to process the predicate P and a Frame operator, which looks for sequences meeting the duration condition, D. More specifically, Apply applies the predicate, P, to each input tuple and appends a true or false value to each. These tuples are fed to the Frame operator, which uses the duration D to form frames.

Figure 1 depicts frame detection. Apply converts `Loss_Rate` values shown in part (a) into Boolean values as shown in part (b).

Subsequently, Frame processes these Boolean values to detect sequences of true values that endure for the specified time period. In this example, there are two separate frames, one from time 2 through 5, and one from time 7 through 9.

Frame output consists of a unique frame id plus frame start and end times. The methods for detecting a frame depend on how and when the data arrives. All data is assumed to have a timestamp attribute, but is not assumed to arrive in order. If the data arrives on a known reporting schedule, (for example, a router reporting `Loss_Rate` every 20 seconds), the operator can take advantage of the known schedule and can use that information to output frames as they are detected and infer if data is missing. Missing data may be handled in various ways. In the router example, one might assume that a missing report is the same as a high-loss report. If data is not assumed to arrive in order, then a mechanism for communicating the progression of time is necessary. Punctuation is one such mechanism [11]. In the case of a reporting schedule, punctuation is used to determine when data is missing as opposed to just arriving out of order. If there is no reporting schedule, punctuation is necessary to determine the start and extent of frames.



Loss_Rate	0.2	0.4	0.6	0.4	0.5	0.1	0.4	0.4	0.4	0.2
time	1	2	3	4	5	6	7	8	9	10

(a)

		Frame 1					Frame 2			
predicate	F	T	T	T	T	F	T	T	T	F
time	1	2	3	4	5	6	7	8	9	10

(b)

Figure 1. (a) Loss_Rate by time. (b) Frames with Loss_Rate > 0.3% for 3 or more consecutive time periods.

3. EXTENDING FRAMES TO MULTIPLE DIMENSIONS

Consider Figure 2. It depicts data received from freeway loop detectors that report speed, volume (vehicle count), and occupancy (presence of vehicle over sensor) every 20 seconds. The chart represents spatial location on Interstate 5 North in Portland on the Y-axis. The numbers correspond to mileposts. The X-axis shows time of day. The color codes for speed, with red being slower and green faster.

One can see congestion in the figure, with slow speeds persisting over a region of time and space. The box shows a region with speeds below 20mph extending from mile 303.8 to 308 and lasting from 16:35 to 18:00.

We would like to extend our frame operator to capture such events that have both temporal and spatial extent. For example, we might want to identify periods where speed < 20 mph over distance of at least 2 miles for a period of at least 30 minutes. So each frame instance would have a start and end location in addition to a start and end time.

Given such a frame, we might combine it with the detector stream to calculate total time-delay experienced by drivers in the region, using volume, speed, and knowledge of "free-flow" speed. Or we could use it to select from a stream of GPS reports coming in from "probe vehicles" to look at travel time through the congestion. Or we might just take the frame alone and compare it to a collection of known "standing bottlenecks" that recur daily to see if the frame represents an unexpected spatial-temporal event that might indicate an accident or some other road blockage.

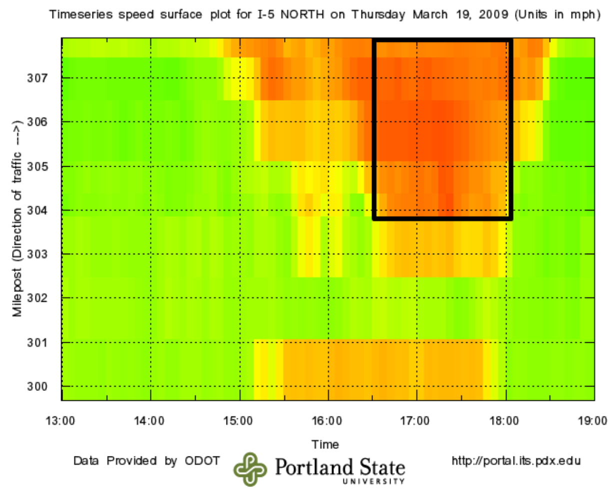
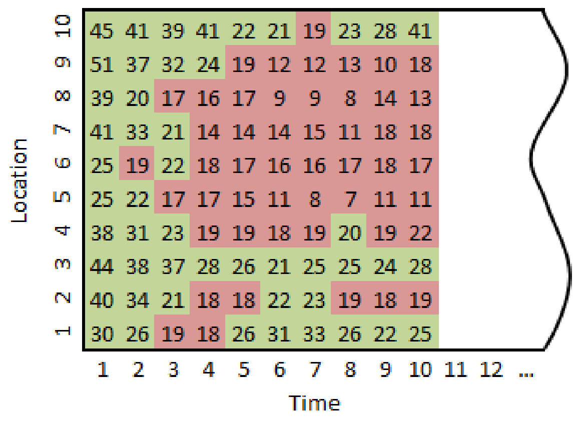


Figure 2. Highway speed contour plot.

We are exploring ways to enhance our frame implementation to handle multi-dimensional frames. We describe here two alternative approaches. For this description, we assume our input tuples arrive at predetermined intervals in time and space, as shown in Figure 3.



Location												
10	45	41	39	41	22	21	19	23	28	41		
9	51	37	32	24	19	12	12	13	10	18		
8	39	20	17	16	17	9	9	8	14	13		
7	41	33	21	14	14	14	15	11	18	18		
6	25	19	22	18	17	16	16	17	18	17		
5	25	22	17	17	15	11	8	7	11	11		
4	38	31	23	19	19	18	19	20	19	22		
3	44	38	37	28	26	21	25	25	24	28		
2	40	34	21	18	18	22	23	19	18	19		
1	30	26	19	18	26	31	33	26	22	25		
Time	1	2	3	4	5	6	7	8	9	10	11	12 …

Figure 3. Highway speeds < 20 mph.

Both approaches begin by applying the predicate to each incoming tuple. We can view the result as a 2-D Boolean array, as shown in Figure 4.

Method I applies our existing 1-D technique at each spatial location. At a given point in time, we have the duration of the predicate for each location – the number of contiguous prior time periods when the predicate was true. For example, in Figure 5, the 7 entry for location 6 at time 10 indicates that the predicate has been true for the past 7 reports. The durations are easily updated at each new reporting time. The existence of a frame instance with a given temporal and spatial extent is easily detected from the duration values at a given time instant. For example, if we are looking for a frame of spatial extent at least 4 units and temporal extent at least 5 units, we just need to see whether there is a stretch of at least 4 duration values at a given instant where the minimum value is at least 5. In Figure 5, for example, locations 5-8 meet this criterion at time 8.

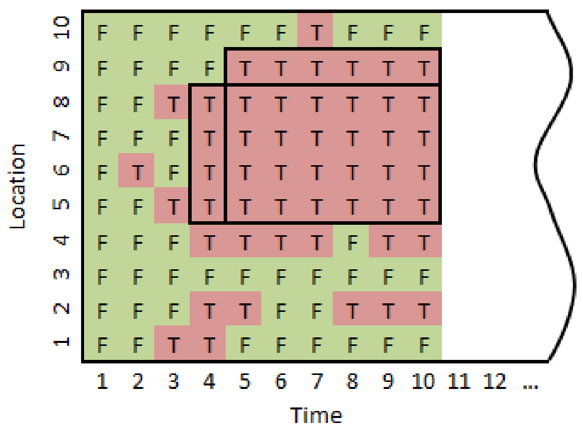


Figure 4. Array of Boolean predicate values for Figure 3.

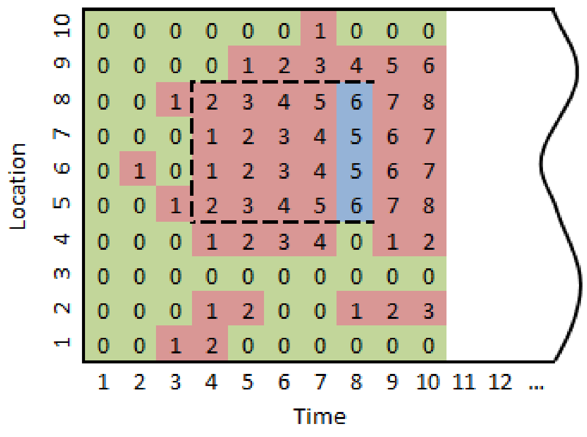


Figure 5. Method I: Duration of Predicate Values.

Method II keeps track of partial frames as a list of "spans" in the spatial dimension, along with a temporal "depth" of each span. As before, we are looking for frames of spatial extent at least 4 units and temporal extent at least 5 units. Figure 6 shows the spans computed at time 7, one of depth 3 for locations 4-9, and a second of depth 4 for locations 4-8. These spans are demarcated by dotted lines. We only need to maintain spans that have the minimal spatial extent. Spans can be updated incrementally; they might shrink spatially as their depth increases. For example, the 4-9 span at time 7 shrinks to locations 5-9 at time 8. A span of depth equal or greater than the minimum temporal extent represents a frame instance.

While Method I is simpler, Method II should accommodate irregularity in the spatial dimension, for example, if speed reports come from vehicles rather than fixed sensors.

4. ISSUES

We have identified several issues for further investigation, which we describe below. The first three arise from the multi-dimensional aspect. The last two also arise in the 1-D case, though are likely simpler to address there.

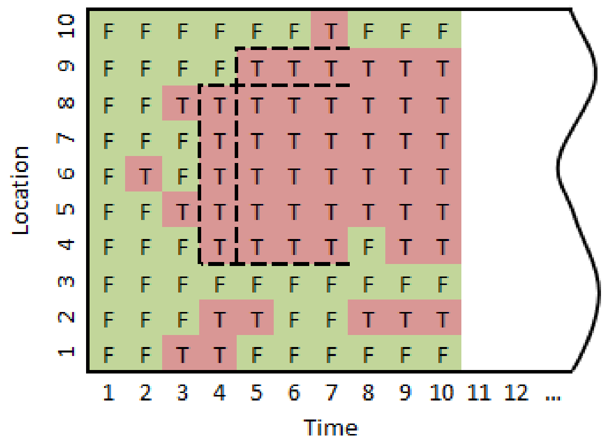


Figure 6. Method II: Spans computed at time 7.

A. Maximality: In the 1-D case, requiring maximum duration for frames means that frame instances do not overlap. However, in two or more dimensions, maximal frames can overlap. For example, in Figure 4 there is a 4-by-7 frame at locations 5-8 starting at time 4 and a 5-by-6 frame at locations 5-9 starting at time 5, neither of which contains the other. While we are not opposed in principle to overlapping frames, we are concerned that a) The proliferation of frames may lead to computational overhead and b) multiple frames representing the same "conceptual" event may be confusing to users. One possible approach is to define a preference relationship among frames, though it might be challenging to come up with an intuitive definition that is transitive.

B. Incremental Reporting: In the one-dimensional case, our implementation is capable of reporting frame fragments. Going back to the router example, suppose the period of high loss for a router extends over 98 reports. We might not want wait until the situation ceases to issue an alert or start filling the frame. Thus, once we have reached the minimum threshold, we can report an initial fragment, and then report continuation fragments, and ultimately a final fragment, as shown in Figure 7. In the multi-D case, there is an issue that once we have determined that a frame instance will exist, we do not know its final spatial extent. Going back to Figure 3, we have an initial 5-by-6 region starting at time 5, indicating there will definitely be a frame instance, but we do not know at that point whether the frame might end up as, say, 5-by-9 or 4-by-9.

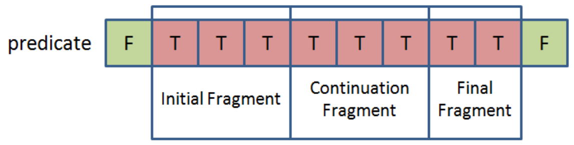


Figure 7. Incremental Frame Reporting.

C. Non-rectangular Frames: Referring back to the traffic diagram in Figure 2, the upper congested region actually has a triangular shape. (Such a shape is not unusual in traffic analysis; for example, see Cassidy *et al.* [5]) Thus, we might want to allow frames that are not rectangles or that are not parallel to the axes. We believe both methods we describe could be extended to handle convex polygons with additional mechanisms, but are not certain they would extend to more complex shapes.

D. Richer Specifications: There are obviously other ways to specify a frame besides a predicate holding over a minimal temporal and spatial extent. We might specify a frame based on a certain density of a particular kind of event. For example, we might want to determine areas in an office building where the number of people per square foot (say as detected with RFID readers or motion detectors) exceeds a threshold for more than 10 minutes, so that the HVAC system can be directed to provide more cooling or ventilation to the area. For such a specification, it is not clear we want a maximality condition on frames. Perhaps we want the smallest region meeting the density threshold, such that expanding the region drops the average density.

E. Noisy Data: Traffic sensors give notoriously dirty data, both in terms of missing and corrupted reports. We might want to allow imputation of a default value for a missing or "bad" report for purposes of frame detection. We might want to distinguish "real" and imputed reports for frame initiation. For example, a frame for a congested region must start with real reports, but can be extended with imputed reports.

5. RELATED WORK

Window definitions are typically based on a progressing attribute such as time or tuple-count. For example, Li, *et al.* [11] describe window semantics in which a window-id is attached to each tuple based on tuple-count or some other progressing attribute such as application timestamp. Frames can be thought of as data-driven windows, where the boundaries are dynamic. Past research on data-driven windows includes work on predicate-windows by Ghanem *et al.* [8]. These windows slide neither by time nor by tuple-count. Tuples enter into and expire from a predicate-window depending on whether they satisfy the predicate associated with the window. The semantics allow users to write queries such as 'Continuously, report the sensor-identifiers for sensors that have temperature greater than 90'. As opposed to detecting sets of tuples meeting a predicate, frames detect boundaries of sequences of tuples meeting a predicate.

Another way to think of framing is in the context of pattern matching. Various pattern matching languages have been proposed in the previous 5 to 6 years. Agrawal, *et al.*, propose the language SASE+ [1]. Another system that uses automaton and buffer for handling pattern matching queries over even streams is Cayuga [4]. SQL-TS is a database counterpart; it is an extension to SQL that supports searching for complex patterns in database systems [12]. S-OLAP is a flavor of online analytical processing system that supports grouping and aggregation of data based on patterns [6]. To enable real-time responses to pattern-based queries, S-OLAP maintains sets of pattern-based aggregate values in structures called s-cuboids.

The aforementioned systems are interested in the data driving the pattern matching. In contrast, our approach focuses on detecting the onset and duration of the pattern, enabling us to use this information to join with data from another stream to "fill the frame". Our technique also handles out-of-order data, can be used with reporting schedules, and can also be parameterized to handle missing data.

6. ACKNOWLEDGMENTS

The authors thank Ted Johnson for his router packet-loss example. This work is supported in part by the National Science Foundation (grant IIS-0917349) and CONACyT México (Fellowship 178258).

7. REFERENCES

[1] Agrawal, J., Diao, Y., Gyllstrom, D., and Immerman, N. Efficient pattern matching over event streams. In *SIGMOD 2008.* (Vancouver, Canada, June 2008).

[2] Babu, S., and Widom, J. StreaMon: An Adaptive Engine for Stream Query Processing. In *SIGMOD 2004.* (Paris, France, June 2004).

[3] Balakrishnan, H., *et al*. *Retrospective on Aurora*. The VLDB Journal: The International Journal on Very Large Databases, 13(4). (2004), pp. 370–383.

[4] Brenna, L., *et al.* Cayuga: a high-performance event processing engine. In *SIGMOD 2007*. (Beijing, China, June 2007).

[5] Cassidy, M.J., and Bertini, R.L. *Some traffic features at freeway bottlenecks.* Transportation Research Part B: Methodological, 33(1). (February 1999), pp. 25–42.

[6] Chui, C., Kao, B., Lo, E., and Cheung, D. S-OLAP: an OLAP system for analyzing sequence data. In *SIGMOD 2010*. (Indianapolis, Indiana, USA, June 2010).

[7] Cranor, C., Johnson, T., Spataschek, O., and Shkapenyuk. V. Gigascope: A Stream Database for Network Applications. In *SIGMOD 2003*. (San Diego, California, USA, June 2003).

[8] Ghanem, T. M., Aref, W. G., and Elmagarmid, A. K. *Exploiting predicate-window semantics over data streams.* SIGMOD Rec. 35(1). (Mar. 2006), pp. 3–8.

[9] Goldstein, J., Hong, M., Ali, M. and Barga, R. *Consistency Sensitive Operators in CEDR*. Technical Report MSR-TR-2007-158, Microsoft Research. (2007).

[10] Li, J. Tufte, K., Shkapenyuk, V., Papadimos, V., Johnson, T., and Maier, D. *Out-of-Order Processing: a New Architecture for High-Performance Stream Systems*. Proc. VLDB Endow. 1(1). (2008) pp. 274–288.

[11] Li, J., Maier, D., Tufte, K., Papadimos, V., and Tucker, P. A. Semantics and evaluation techniques for window aggregates in data streams. In *SIGMOD 2005*. (Baltimore, Maryland, June 2005).

[12] Sadri, R., Zaniolo, C., Zarkesh, A., and Adibi, J. *Expressing and optimizing sequence queries in database systems*. ACM Trans. Database Sys. 29(2). (June 2004), pp. 282–318.