Journal of Universal Computer Science, vol. 15, no. 12 (2009), 2287-2310
submitted: 15/12/08, accepted: 25/6/09, appeared: 28/6/09 © J.UCS

Causality Join Query Processing for Data Streams via a Spatiotemporal Sliding Window

Oje Kwon
(Pusan National University, South Korea
kwonoj@isel.cs.pusan.ac.kr)

Ki-Joune Li
(Pusan National University, South Korea
lik@pnu.edu)

Abstract: Data streams collected from sensors contain a large volume of useful information including causal relationships. Causality join query processing involves retrieving a set of pairs (cause, effect) from streams of data. However, some causal pairs may be omitted from the query result, due to the delay between sensors and the data stream management system, and the limited size of the sliding window. In this paper, we first investigate temporal, spatial, and spatiotemporal aspects of causality join query processing for data streams. Second, we propose several strategies for sliding window management based on these results. The accuracy of the proposed strategies is studied via intensive experimentation. The result shows that we can improve the accuracy of causality join query processing in data streams with respect to the simple FIFO strategy.
Key Words: data stream, causality join query processing, spatiotemporal sliding window
Category: H.3.3

1 Introduction

Data streamed from diverse smart sensors contains a large volume of practical information for realizing intelligent environments. The causality analysis on data streams from sensors is often crucial for many applications, to understand environmental phenomena and achieve suitable control of actuators. Consider, for example, an automatic fire detection service in a massive building. Leakage data on gas is detected by gas sensors and that of electricity by electric sensors. The response to a fire should depend on where the fire originates from leakage of gas or electricity.

In data stream management systems (DSMS), one of the important analysis functions is to discover pairs (cause, effect) in a data stream from sensors. In this paper, this is called causality join query processing for data streams. Temporal, spatial, and spatiotemporal relationships are often very helpful for discovering the causal relationships in data streams, in addition to domain-specific knowledge. First, the cause always precedes its effect. Second, the locations of cause and effect often satisfy certain spatial conditions, such as the effect occurring within a given radius from the cause. Third, the cause is propagated at a given speed to the effect. These three examples correspond to temporal, spatial, and spatiotemporal relationships, respectively.

In most cases, the data collected at sensors is streamed to DSMS with a certain delay, for several reasons. The delay is an important factor, which degrades the accuracy of causality join query processing for data streams stored in limited sliding windows. For example, it is hard to rapidly respond to a fire if either the cause or effect arrives late at DSMS, due to the delay. However, the FIFO policy, which is a popular method of sliding window management, significantly degrades the accuracy of join query processing. In order to improve the accuracy, several issues must be carefully examined, in particular the temporal, spatial, and spatiotemporal relationships.

In this paper, we study the temporal, spatial, and spatiotemporal relationships between cause and effect. Based on these results, we propose three buffering methods for the sliding window, and compare them with the FIFO policy via intensive experimentation. The contributions of this paper are summarized as follows,

- Introduction of causality join query processing in data streams
- Study on temporal, spatial, and spatiotemporal relationships for causality join query processing and
- Buffering policies of the sliding window for causality join query processing, to improve the accuracy.

This paper is organized as follows. First we survey the related work in section 2. We introduce the motivations of the study in section 3 and give the definition of causality join query processing in data streams. In section 4, we investigate the temporal, spatial, and spatiotemporal properties of causal relationships in data streams collected from sensor networks. And, we analyze and compare the accuracy of the proposed methods and FIFO policy via intensive experimentation in section 5, and conclude the paper in section 6.

2 Related work

Sensor data transferred to DSMS forms an infinite stream. To deal with a sensor stream, DSMS has to consider two central critical constraints, 1) Limited buffer capacity and 2) Real-time query processing. In this section, we survey previous studies on these constraints.

2.1 Query processing with a limited buffer capacity

Since DSMS has a limited buffer, non-blocking operators for continuous partial processing are required, instead of operators for processing the entire data stream. A well-known method is the sliding window in STREAM [Arasu et al. 03] , Fjord [Madden and Franklin 02] and TelegraphCQ [Chandrasekaran et al. 03]. The performance of the sliding window is determined by the window size. The larger the window size, the more accurate the query result, because the volume of concurrently processed data is large, but query processing is more expensive.

Sliding window methods in [Arasu et al. 03], [Chandrasekaran et al. 03] and [Madden and Franklin 02] adopted the FIFO buffering policy, and only considered the transactional time of the data. But the valid time is more important than the transactional time for detecting causal relationships in the data stream.

D. Papadias and his fellows applied the sliding window method to processing k nearest neighbor monitoring in a stream data [Tao and Papadias 06] [Mouratidis and Papadias 07]. They proposed two methods; one finds k nearest neighbor pairs based on conceptual cells which are partitioned around the query point; the other finds the pairs based on the skyline that is a result of the time and distance between the query and each point. Nevertheless, the sliding window methods in [Tao and Papadias 06] and [Mouratidis and Papadias 07] were performed in a FIFO manner and were based on the transactional time.

In Joe Khor et al. applied the sliding window method to multiple routing protocols in ad-hoc environments [Khor et al. 05]. They proposed a solution to improve the security performance of multiple routing via sliding window. In particular, when there are node insertions/deletions to/from a window, they reset the security keys of nodes within the sliding window, without stopping transmission. Hua-Fu Li et al. utilized the sliding window method for mining changes of a data stream within a limited buffer [Li et al. 05]. However, the aforementioned methods did not consider the stream data sequence.

2.2 Real-time query processing

The second constraint dealing with infinite sensor streams is real-time query processing. A causality query is a type of join operator. STREAM proposed a binary join operator that uses a hash table. Fjord provided a zipper join, which joins two data stream elements if their transactional times are the same. T. Urhan and M. J. Franklin proposed the XJoin operator, which copes with a delay of the data stream [Urhan and Franklin 00]. This operator conducts a memory-based join first, and switches to a disk-based join when the stream has a slow delivery because of the delay. But binary join, zipper join and XJoin only focused on the transactional time of the data stream. They did not consider the valid time of the data stream, nor did they focus much on its spatiotemporal properties.

L. Ding et al. proposed a MJoin operator using the static and dynamic meta-data on the stream [Ding et al. 03]. In [Ding et al. 03], punctuation is used, which is the predicate that specifies the data that cannot appear after the punctuation mark. Non-prior data elements appearing after the punctuation mark are first purged from the window. But the punctuation conditions are not based on the valid time of the data, but rather the data values, and punctuation marking is included in the preprocessing steps. M. A. Hammad et al. proposed a multiple join operator, viz., Stream Window Join, based on the different delays of each stream [Hammad et al. 03]. They create a maximum threshold for delays among multiple streams, and continue processing the join until the join pair of different streams is completed. But, they only focused on the sequential order of the stream according to the transactional time. Buğra Gedik et al. proposed a GrubJoin [Gedik et al. 07] to decrease the CPU overhead during the processing join operation of multiple streams, and they applied it to the multiple stream monitoring system [Kun-Lung et al. 07]. They proposed two methods; an operator throttling method, which tunes the frequency of operations based on the transfer period of the stream; and, a window harvesting method which finds stream data that is directly related to join operation. However, they only considered the transactional time of the stream data.

K-L. Tan and his research team have performed numerous studies about realtime query processing for data streams. J. Wu et al. attributed the degradation of the join performance to the user-defined static window size [Wu et al. 07]. They proposed a memory-efficient algorithm to determine the window size, based on both the order of the streams and the delays among multiple streams. But, this join operator is performed in a FIFO manner, which only considers the transactional time. Yongluan Zhou et al. proposed COSMOS middleware, which process multiple queries in a distributed stream system using a publish/subscribe mechanism [Zhou et al. 08]. They proposed graph-based methods to increase the load-balance of each system and decrease the communication cost when queries are disseminated. However, they considered the transactional time of the stream data. HyperGrid, which provides common generic procedures to convert raw spatiotemporal sensor stream data into analytic data, was proposed in [Wu et al. 09]. This represents data with a grid model, and it provides customizable and generic operators for users. But, it does not distinguish between the valid time and the transactional time, because it fails to consider the communication delay in USN.

Causality in data mining involves inferring the causal relationships in information [Silverstein et al. 00] [Freedman 04] [Holland 86] [Pearl 00]. LTCCS statistically inferred causal relationships in data on truck accidents [LTCCS 07]. XinZhou Qin and Wenke Lee proposed a statistical causality analysis method for alerts in a security mechanism [Qin and Lee 03]. But causal relationships in

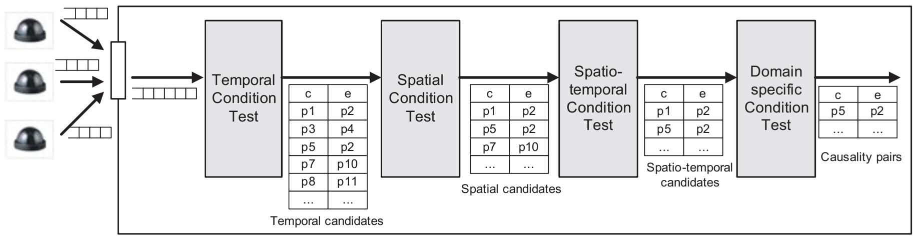


Figure 1: Conceptual evaluation procedure for causality join query processing in data streams

sensor streams must be analyzed in real-time, and statistical analysis is impossible, due to the limitations of the window size in DSMS. For this reason, new approaches are necessary for dealing with causal relationships in sensor streams.

3 Causality Join Query Processing in Data Streams

In this section, we give the definition of causality join query processing for data streams from sensors. And, we discuss the problems in processing a causality join query in sensor network environments.

3.1 Causality Join Query Processing for Data Streams

Causality join query processing in data streams is defined as the selection of pairs (cause, effect) from data streams satisfying the following causality condition;

Definition 1. Causality Join Query Processing for Data Streams
Given the data streams, the result of causality join query processing is defined as follows;

$$R\_CQ(X) = \{(c,e) | c \in W(X_i), e \in W(X_j),\ and\ P_{CQ}(c,e) = TRUE\} \quad (1)$$

where X=\{X_1, X_2, ... X_K\} is the set of streams, c and e represent cause and effect, respectively, and $W(X_i)$ denotes the window buffer of the X_i stream.

In this definition, $P_{CQ}(c,e)$ is the predicate describing the causality conditions, and it can be represented in conjunctive form as:

$$P_{CQ}(c,e) = P_{CQ.1}(c,e) \wedge P_{CQ.2}(c,e) ... \wedge P_{CQ.n}(c,e) \quad (2)$$

In order to process a causality join query, DSMS should evaluate a given set of predicates defined by the application. The predicates given in equation (2) are classified into four categories as follows.

- **Temporal predicate**: The cause and effect always satisfy the temporal condition that the cause occurs before the effect.
- **Spatial predicate**: The locations of cause and effect satisfy spatial conditions. For example, the distance between the cause and effect must be less than a given threshold.
- **Spatiotemporal predicate**: Since the phenomena of cause and effect are dynamic, the relationship contains spatiotemporal properties. One of the obvious spatiotemporal properties is the velocity of propagation. For example, a fire expands from the firing position at a certain speed, which is a useful fact in understanding fire phenomena. It means that the propagation delay from the firing point to the effect is a function of the velocity and distance.
- **Domain-specific condition**: While the aforementioned predicates are independent of the type of application, there are conditions defined by the type of application. We exclude these, since they are beyond the scope of the paper.

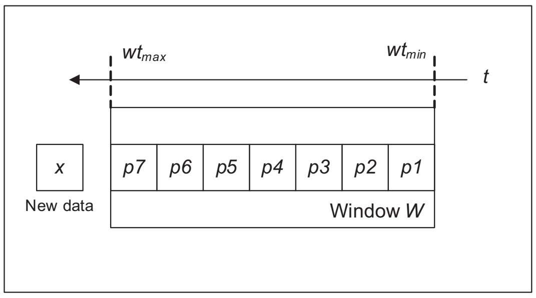
The conceptual evaluation procedure for causality query predicates is explained in figure 1. First, the predicates concerning the temporal condition are evaluated with the data stream gathered from the sensors, to produce the first set of candidate pairs (cause, effect) for the causality join query. Second, the spatial predicates are evaluated with the candidate pairs, to obtain the spatial candidates. In a similar manner, we evaluate spatiotemporal predicates and domain-specific predicates.

3.2 Buffering and Causality Join Query Processing

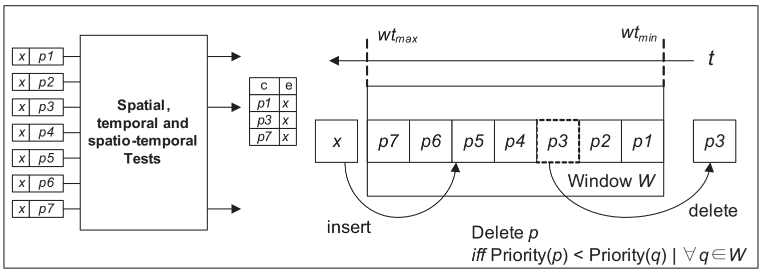
Unlike database management systems, join queries are processed with the data stored in a sliding window with a limited size, which is explained in figure 2. Note that for the sake of simplicity we assume a sliding window of a single stream for causality join query processing in this paper.

In figure 2, it is supposed that all data elements p_i in the sliding window W were previously examined to determine whether they satisfy the causality predicates. When a new data element x arrives at DSMS, as shown in figure 2, the following tasks must be performed.

- **Causality join with data elements in W** : Each data element q in W must be examined with x, regardless of whether the causality predicate $P_{CQ}(p, x)$ is TRUE or not.



(a) Initial data stream in window



(b) Casuality join query procedure

Figure 2: Causality join query processing procedure for data streams

- **Removal of a data element from** W : Since the size of W is limited, one of the data elements in W will be removed. The data element must be selected such that the probability of causal relationships with the arriving data elements is minimal.

3.3 Problems in causality join query processing for data streams

In this paper, we assume computing environments consisting of sensor networks and a DSMS that collects data from sensors, where the data elements are transferred from a sensor to the DSMS via multi-hops. In order to process causality join queries in this environment, the following problems must be solved.

- **Limited size of window**: It is impossible to examine all data elements streamed to DSMS for the causality join conditions, since the size of the sliding window is limited, thus only a subset of data elements can be examined. This means that the result of causality join query processing is incomplete, and some pairs of cause and effect are likely to be omitted from the result.

- **Delay**: The sensors are connected to DSMS via multi-hops transfer. This means that a transfer delay from a sensor to DSMS is inevitable, for several

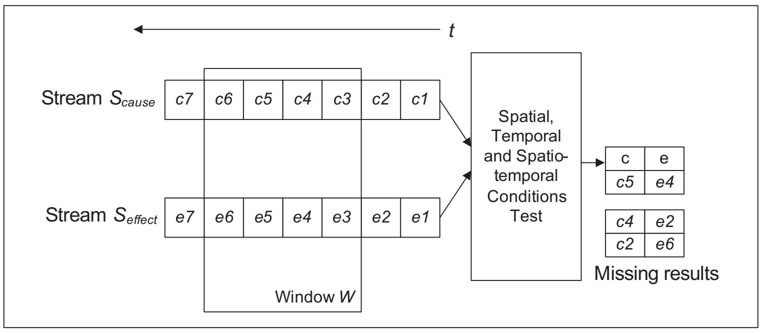


Figure 3: Example: Inaccurate query result due to the limited size of the window and transfer delay

reasons. And, we cannot guarantee that the arrival sequence of data elements is identical to that of the sensors. For example, a data element x captured at a sensor earlier than another data element y may arrive at DSMS later than y.

The two aforementioned problems are related, and they degrade the accuracy of the result of causality join query processing. Figure 3 explains the case where the system gives an inaccurate result of a query. In this figure, we suppose that $(c5, e4)$, $(c4, e2)$, $(c2, e6)$ is the query result of a causality join. Due to the transfer delay, $e3$, $e4$, $e5$ and $e6$ arrives at the DSMS later than $e2$, although they had been captured at sensors earlier than $e2$. If the size of window is four, and the policy of the sliding window is FIFO, $e2$ will be removed from the sliding window, and $(c4, e2)$ cannot be contained in the query result. For a similar reason, $(c2, e6)$ is not included in the query result, which contains only $(c5, e4)$, as shown by figure 3. Note that the FIFO policy in the figure is based on the *transactional time* (t_t) rather than the *valid time* (t_v). However, the valid time is more significant for causality join query processing than the transactional time.

In this paper, we propose several methods for improving the accuracy of query results. The goal of these methods is to ensure that probable causes and effects remain in the sliding window at the same time. To this end, we firstly analyze the temporal, spatial, and spatiotemporal properties. Secondly, we propose several policies for sliding window buffering based on this analysis.

4 Temporal, Spatial, and Spatiotemporal Relationships between Cause and Effect

In this section, we explore the temporal, spatial, and spatiotemporal properties between cause and effect in data streams from sensor networks, which are not specific to a given application domain. Based on these results, we propose several policies for buffer management of the sliding window.

4.1 Temporal relationships in causality

The most obvious property between cause and effect is the temporal relationship that a cause must precede an effect, and the temporal distance must be within a certain threshold. This property is described by the following predicate,

$$P_{CQ.T}(c,e) = \begin{cases} TRUE & \text{if } t_v(c) < t_v(e) < t_v(c) + \delta_T, \\ FALSE & \text{otherwise} \end{cases}$$

where $t_v(c)$ and $t_v(e)$ represent the valid time of cause c and effect e, respectively. The temporal property of causal relationships is summarized as

Property 1
Temporal property of causal relationships

$$P_{CQ.T}(c,e) = TRUE \textit{ if } c \textit{ is a cause and } e \textit{ is its effect.}$$

Consequently, the pairs (c, e) satisfying the temporal property must be selected from the data stream for causality join query processing.

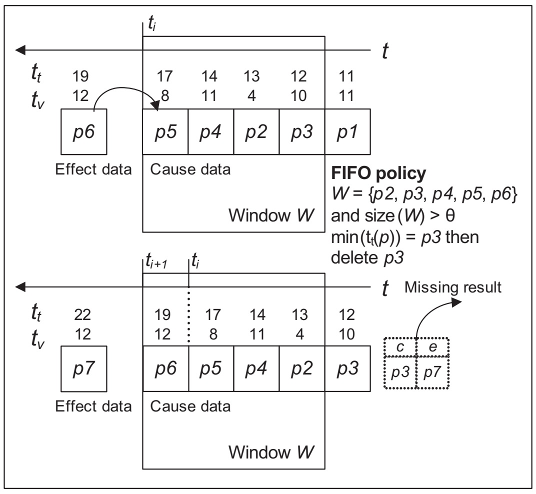
An important requirement of buffering is to remove a data element with the lowest probability of being a cause or an effect. The temporal property implies that the probability that (c, e) is a causal pair decreases as $t_v(e)-t_v(c)$ increases. This can be described by the following assumption.

Assumption 1
For two data elements p and q in the sliding window W,
if $t_v(p) < t_v(q)$, then

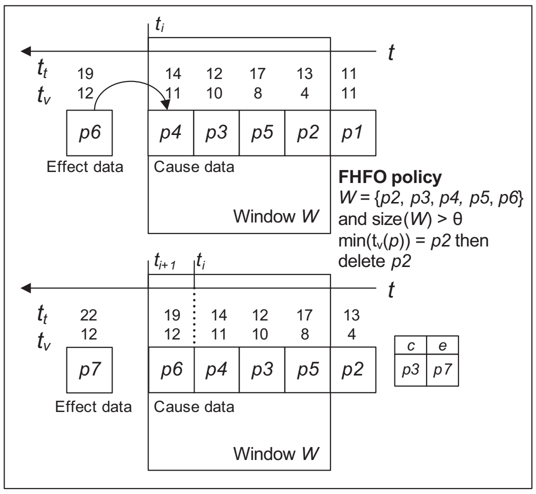
$$Prob(P_{CQ.T}(p,x) = TRUE) < Prob(P_{CQ.T}(q,x) = TRUE),$$

where x is a data element that arrives from the sensor network.

It may be impossible to prove this assumption, but it is intuitively reasonable. And, this assumption means that we must remove the data elements in the sliding window according to the order of the valid time, rather than FIFO. Based on this observation, we propose a buffering policy for a sliding window, called FHFO (First Happens First Out) as follows,



(a) FIFO buffering policy



(b) FHFO buffering policy

Figure 4: Comparison of FIFO and FHFO

Definition 2. FHFO (First Happens First Out)
The buffering policy FHFO for a sliding window W is defined as

$$FHFO(W) : \text{remove } p \in W \text{ such that } t_v(p) = min(\{t_v(q)|q \in W\})$$

In comparison with FIFO, we reduce the omission of pairs, and consequently improve the accuracy of the query results, as explained in figure 4.

In this example, we assume that the pair $(p3, p7)$ is a causal pair. When a new data element $p6$ arrives, $p3$ is removed from the window according to the FIFO policy. When the next data element $p7$ arrives, $p3$ has been previously removed, and consequently the pair $(p3, p7)$ cannot be included in the result set. Compared with FIFO, $p3$ remains in the window, and $p2$ is removed from the window, since $t_v(p2) < t_v(p3)$. This means that $(p3, p7)$ may be included in the query result.

4.2 Spatial Relationships in Causality

Spatial relationships between cause and effect are also an important property, as well as temporal relationship. In general, spatial relationships are more complicated than temporal relationships, and expressed in several manners, including geometrical and topological relationships. However, in this paper, we focus on a general and obvious spatial relationship, in terms of the distance between the cause and effect.

$$P_{CQ.S}(c, e) = \begin{cases} TRUE & \text{if } dist(p(c), p(e)) < \delta_S, \\ FALSE & \text{otherwise} \end{cases}$$

$p(c)$ and $p(e)$ represent the locations of cause and effect, respectively. If these locations are too distant, they cannot be a cause and its effect. Note that the distance is defined according to the type of space and application.

The spatial property implies that the probability that (c, e) is a causal pair decreases as $dist(p(c), p(e))$ increases, and it can be described by the following assumption,

Assumption 2
For two data elements p and q in the sliding window W,
if $dist(p(p), p(x)) < dist(p(q), p(x))$, then

$$Prob(P_{CQ.S}(p, x) = TRUE) > Prob(P_{CQ.S}(q, x) = TRUE),$$

where x is a data element that arrives from the sensor network.

While we can sort the data elements in W according to the valid time, we cannot sort them according the position. In order to apply this assumption to buffering of a sliding window, we sort them according to the distance from DSMS. Although this distance may not fully reflect the spatial property for the buffering policy, it could affect the sequence of the data stream. This spatial buffering policy, viz. FCFO, involves removing the data element p with the smallest $dist(p, b)$, where b is the location of DSMS. This policy is summarized as follows,

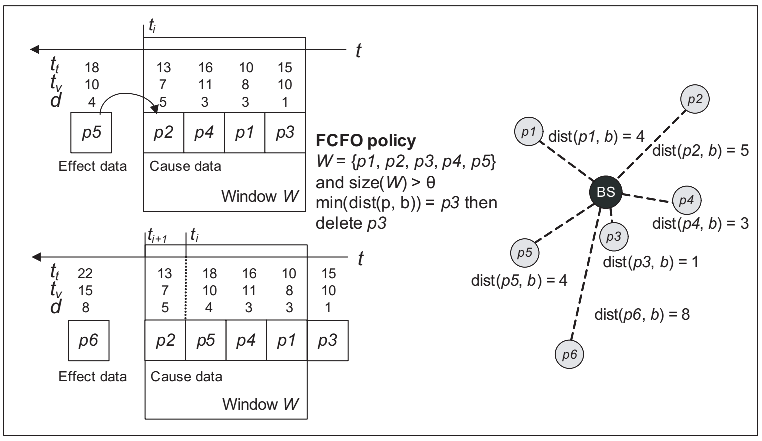


Figure 5: Example of FCFO

Definition 3. FCFO (First Closely located First Out)
The buffering policy FCFO, for a sliding window W, is defined as

$$FCFO(W) : \text{remove } p \in W \text{ such that}$$
$$dist(p, b) = min(\{dist(q, b) | q \in W\}) \text{ or } t_{cur} - t_t(p) > t_{max.stay},$$

where t_{cur} and $t_t(p)$ implies the current time and transactional time of p.

In this policy, we remove the data element if $t_{cur} - t_t(p) > t_{max.stay}$. This is to prevent the data element from remaining in the window indefinitely. This may happen when a data element arrives from a sensor very distant from b. If a data element remains in the window longer than $t_{max.stay}$, then we remove this element. An example of this policy is explained in figure 5.

We suppose that sensors are located as shown in figure 5. When a new data element $p5$ arrives, then the data element that has remained longer than $t_{max.stay}$ is removed. If no data element remains longer than $t_{max.stay}$, then the data element with the smallest distance from DSMS, which is $p3$ in figure 5, is deleted from the window.

4.3 Spatiotemporal Relationships in Causality

The third relationship between cause and effect is the spatiotemporal one. In this paper, we only focus on the spatiotemporal relationship concerning the

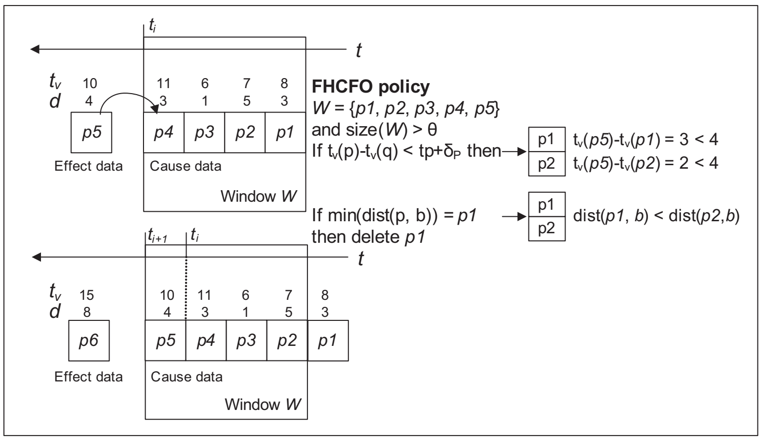


Figure 6: Example of FHCFO

propagation speed. Assuming that the propagation speed is v and the distance between the cause and effect is s, then the effect occurs at least $t_p = s/v$ later than the cause. This relationship is expressed as the spatiotemporal predicate $P_{CQ.ST}(c, e)$

$$P_{CQ.ST}(c, e) = \begin{cases} TRUE & \text{if } t_p < t_v(e) - t_v(c) < t_p + \delta_P, \\ FALSE & \text{otherwise} \end{cases}$$

In this relationship, δ_P represents the maximum tolerance of the propagation time from the cause and effect. This means that the event cannot be the effect of the cause if it happens after a $(t_p + \delta_P)$ delay. Note that the spatiotemporal predicate becomes a temporal predicate when t_p=0.

Based on this result, we apply this relationship to the buffering policy. Suppose that a new data element x arrives at DSMS. A data element p in W may be the cause of the subsequent data elements, if $t_v(p) - t_v(x) < tp + \delta_P$. This means that p must remain in W until the propagation delay with the given tolerance has expired. After the time has expired, the probability of a causal relationship decreases as time advances. Based on this result, we propose a buffering policy as follows,

Definition 4. FHCFO (First Happens and Closely located First Out)
The buffering policy FHCFO for a sliding window W is defined as

```
$FHCFO(W)$ : If $(t_v(p) - t_v(x) < t_p + \delta_P)$, then
        remove $p \in W$ such that $dist(p, b) = min(\{dist(q, b)|q \in W\})$
    Else
        remove $p \in W$ such that $t_v(p) = min(\{t_v(q)|q \in W\})$
```

This policy is in fact a hybrid of FHFO and FCFO. An example of this policy is shown in figure 6.

In this example, we assume that $t_p + \delta_P$=4 and the sensors are located as shown in figure 5. When a new data element $p5$ arrives, we search p in W such that $t_v(p) - t_v(p5) < tp + \delta_P$. Since every data element in W satisfies this condition, we select the data element with the minimum distance to DSMS. $p1$ is then removed, because $dist(p1, b)$ is the minimum.

5 Empirical Analysis

In this section, we present the results of experiments for analyzing and comparing the proposed methods and FIFO.

5.1 Experiment Setup

In order to perform the experiments, we prepared a data set with the following parameters.

- Spatial extent: $[0, 1]^2$
- Location of DSMS: (0.5, 0.5)
- Number of sensors: 100
- Number of data elements per sensor: 1000
- Data detection period per sensor: randomly generated by an exponential distribution with a given expected value λ_1
- Delay per hop: randomly generated by an exponential distribution with a given expected value λ_2
- Type of sliding window: tuple-based
- $t_{max.stay}$ of the FCFO: $n_W \times avg(\triangle\ t_t)$

Table 1: Parameters and their experimental values

Parameter	Description	Range
λ_1	Expected value of detecting period per sensor	0.1,0.15,...,0.5(sec)
λ_2	Expected delay per one hop	0.02,0.04,...,0.2(sec)
t_p	Minimal propagation time from cause to effect	0.2,0.4,...,2.0(sec)
δ_P	Tolerance of t_p	0.1,0.2,...,1.0(sec)
n_W	Size of sliding window	50,100,...,500
s	Distance between cause and effect	0.2

We apply two exponential distributions using the expected value λ_1 and λ_2 respectively, to generate realistic transactional and valid time for sensor streams. There are generally two types of sliding window: tuple-based and time-based. The objective of the proposed buffering methods is to ensure that the data elements that satisfy the causality query remain within the limited window over the long-term. This problem is particularly important when the limited sliding window over-flows. For this reason, we focus on the tuple-based sliding window rather than the time-based one. We set $t_{max.stay}$ of FCFO as the time when the window fills up. The parameters and their ranges are shown in table 1.

We measured the recall rate of the query results, to check the accuracy of the results. In this paper, the recall rate is defined as the number of pairs selected from the data stream with the proposed buffering policies over the number of all pairs satisfying the causality predicates.

$$Recall(P) = num_{Result}(P)/num_{Causality}$$

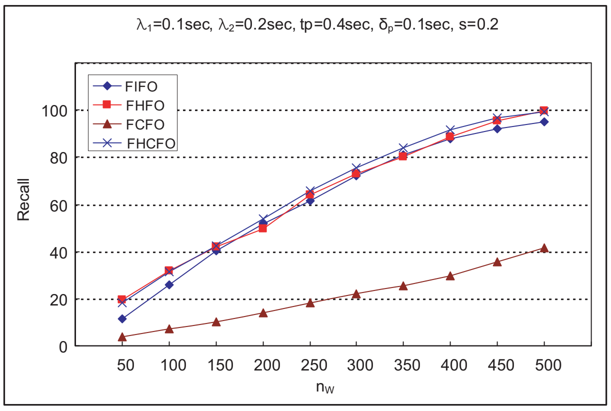
where $P = \{FIFO, FHFO, FCFO, FHCFO\}$

5.2 Results of experiments

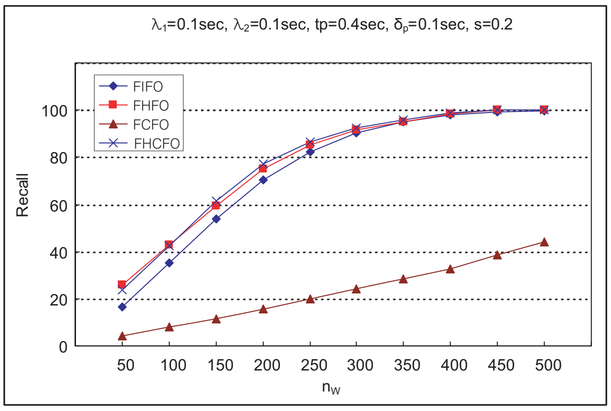
We ran the experiments on Athlon 2.6GHZ Processor with 2.00GB RAM. We performed several experiments with the parameters in table 1, to check the accuracy of each buffering policy. There are a number of combinations of parameter values shown in table 1. We performed most experiments for the possible combinations. But we only present the significant results in this paper, excluding results where the accuracy was nearly 100% for every method.

5.2.1 Experiments on sliding window n_W

Figure 7 shows the recall rate of each buffering policy with respect to the size of the sliding window n_W in DSMS. The x- and y-axis represent the size of the



(a) λ_2=0.2



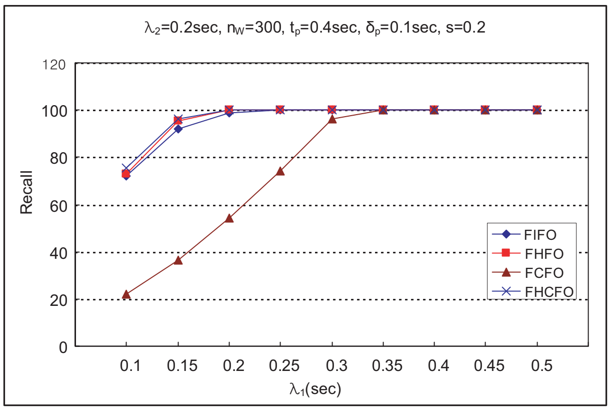
(b) λ_2=0.1

Figure 7: Recall of four policies according to window size n_W

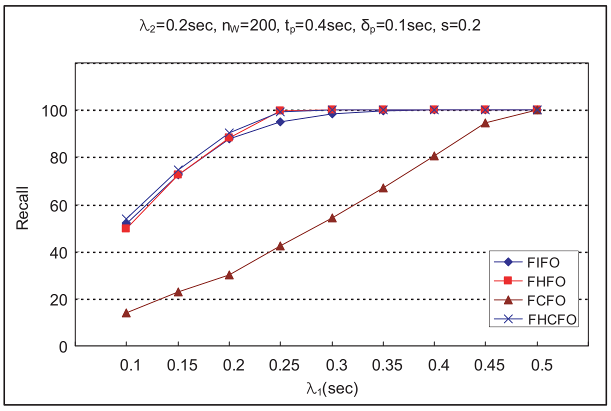
sliding window n_W and recall rate, respectively. It is obvious that the recall rate increases as the window size increases. The experiments show that FHCFO is up to 5% better than FIFO, and FHFO is up to 4% better than FIFO. FHCFO has an accuracy of up to 7% better than FIFO, especially with a small sliding window. This means that we must consider the spatiotemporal property of sensor streams, to improve the accuracy of causality query processing in small device with limited memory, such as mobile devices.

When the expected delay is small ($\lambda_2 = 0.1$, refer to figure 7(b)), the accuracy approaches 100% with a small sliding window. This means that the probability of the reverse of sequence is low, if there is no long transfer delay. This is clearly shown by figure 9. In this experiment, FHCFO also shows better accuracy than FIFO, especially with a small window. One important result from this experiment is that the size of the sliding window greatly affects the accuracy. When the size of window is insufficient, more than half the results may be omitted. And, we observe that FCFO gives the worst accuracy in most of our experiments. This means that we cannot improve the accuracy only via spatial considerations.

5.2.2 Experiments on detection period per sensor λ_1



(a) n_W=300

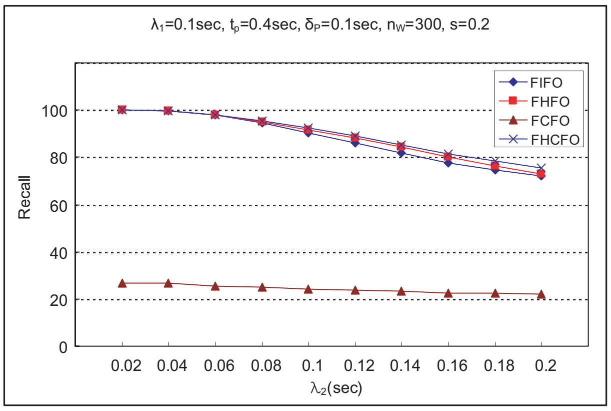


(b) n_W=200

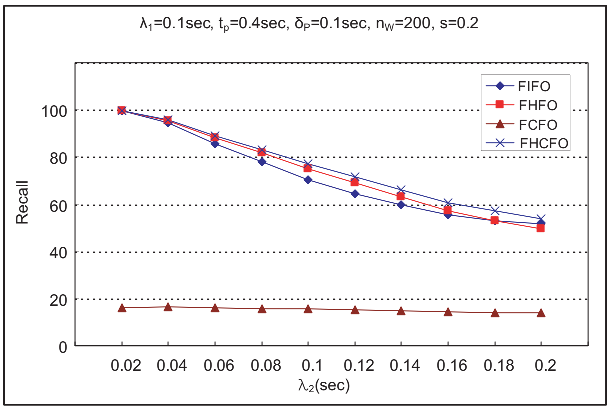
Figure 8: Recall of four policies according to detection period per sensor λ_1

Figure 8 shows the recall rate for each buffering policy with respect to the expected value of the detection period per sensor λ_1. The x- and y-axis represent the detection period λ_1 and recall rates, respectively. As the detection period increases, the probability that the sequence of the data stream does not satisfy that of the occurrences decreases. Figure 8 shows that FHCFO is up to 4% better than FIFO with the large window ($n_W = 300$, refer to figure 8(a)) and 3% better with the small window ($n_W = 200$, refer to figure 8(b)). FHFO is up to 3% better than FIFO as well. This is because FHCFO and FHFO control the buffer using the valid time of the stream data, to avoid the reverse sequence of the stream. In the case that n_W exceeds 450, the accuracy of each method approaches 100%, regardless of the detection period λ_1.

5.2.3 Experiments on transfer delay per hop λ_2



(a) n_W=300



(b) n_W=200

Figure 9: Recall of four policies according to transfer delay per hop λ_2

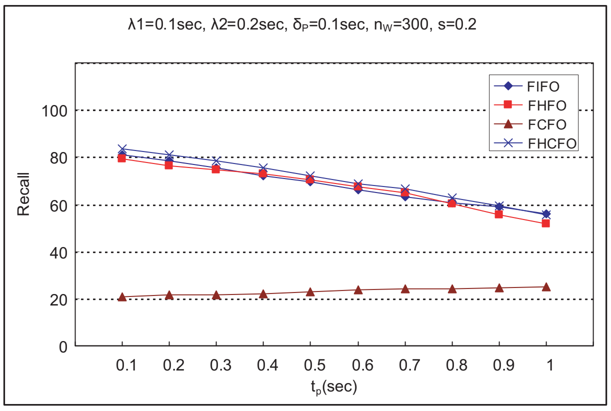
Figure 9 shows the relationship between the recall rate of each buffering policy and the delay per hop λ_2. The x- and y-axis represent the delay per hop λ_2 and recall rate, respectively. When the delay is short, the probability of the reverse sequence of the stream is also low, and consequently this gives a high accuracy. In the case that the delay λ_2 is long, where the probability of the reverse sequence of the stream is high, FHCFO is nearly 4% better than FIFO with the large window ($n_W = 300$, refer to figure 9(a)) and 7% better with the small window ($n_W = 200$, refer to figure 9(b)). FHFO is also nearly 3% and 5% better than FIFO with the respective windows. FHCFO shows better accuracy than FHFO, because FHCFO includes spatial as well as temporal consideration. And, we found a significant reduction in accuracy with a small window size (refer to figure 9(b)). This means that we must set a sufficiently large sliding window to avoid omitting results.

5.2.4 Experiments on effect propagation time t_p

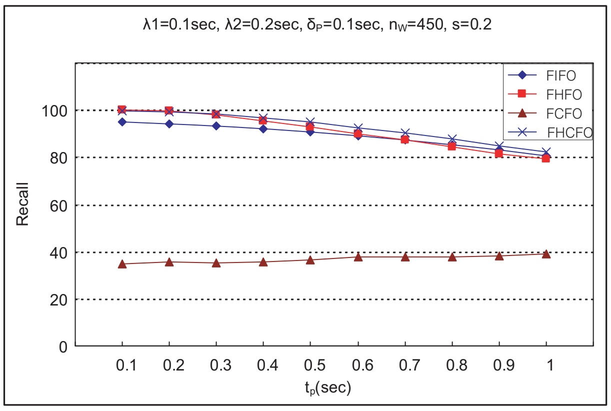
Figure 10 shows the relationship between the recall rate and effect propagation time between the cause and effect. The x- and y-axis represent effect propagation time t_p and recall rate, respectively The decrease of the recall rate according to the increase of the propagation time is as we expect. However the reduction in accuracy is significant when the delay is large (refer to figure 10(a)). This means that setting the size of the sliding window appropriately is very important to avoid omitting results. We can verify this in figure 10(b), where the size of window is sufficiently large. The reduction in accuracy is slight when the size of window is sufficiently large. When n_W is sufficiently large, FHCFO is up to 5% better than FIFO, and FHFO is up to 4% better than FIFO.

5.2.5 Experiments on the effect propagation tolerance δ_P

Figure 11 shows the relationship between the recall rate and effect propagation tolerance δ_P. The x- and y-axis represent effect propagation tolerance δ_P and recall rate, respectively. A high value of the effect propagation tolerance means that the difference between t_v(cause) and t_v(effect) is large. Then, the probability that the cause and effect are in the buffer at the same time is low, as shown in figure 11. Both FHCFO and FHFO are up to 5% better than FIFO with a sufficiently large window size, as shown in figure 11(b). However, FHFO does not give better accuracy than FIFO, especially when the size of window is not sufficiently large, as shown in figure 11(a) as well as figure 10(a). These results show that maintaining a sufficiently large sliding window is very important for controlling the sensor stream data.



(a) $n_W = 300$

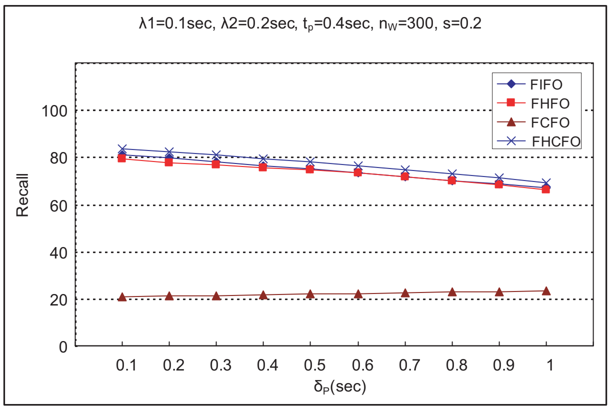


(b) $n_W = 450$

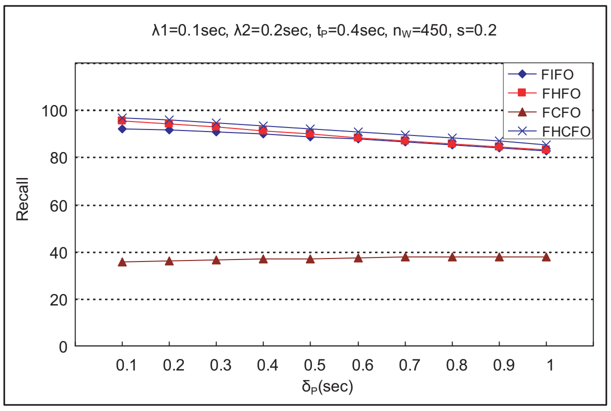
Figure 10: Recall of four policies according to the effect propagation time t_p

5.2.6 Experiments on CPU time

In order to perform causality query processing in sensor streams, only data elements in the sliding window at the same time are valid . Therefore the processing time of buffering methods is entirely related to the size of the sliding window. We measured the CPU processing time of each buffering method according to the size of the window n_W (refer to figure 12). The x- and y-axis represent the size of the sliding window n_W and CPU time, respectively. It is obvious that the CPU time increases as the size of window n_W increases. The CPU time of FCFO FHCFO is larger than that of the others. This is because FCFO and FHCFO calculate the distance between DSMS and each data element in the window, to test the spatial constraints of causality query processing. However, the CPU



(a) n_W=300



(b) n_W=450

Figure 11: Recall of four policies according to the effect propagation tolerance δ_P

time is negligible, and has no effect on real-time causality query processing.

In order to conclude the experiments, we summarize the important results on the accuracy.

- The FHCFO policy consistently gives high accuracy causality join query processing for data streams from sensors.
- The FHFO policy is better than FIFO in most cases, but not always.
- The FHCFO policy shows better performance than FHFO. This means that we must include both spatial and temporal considerations of a data stream, to deal with causality query processing.

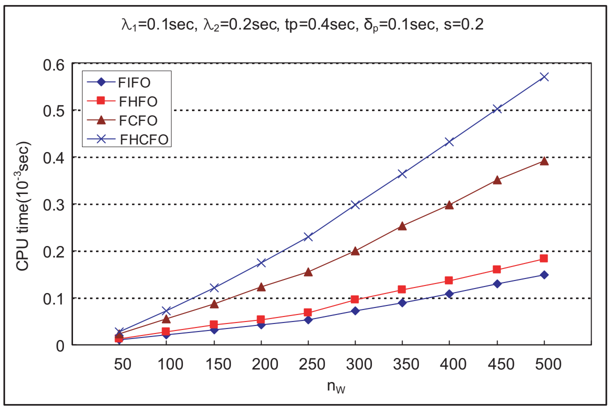


Figure 12: CPU time of four policies according to the size of windown_W

- In some cases, the accuracy falls below 50%. In order to guarantee reasonable accuracy, the size of the sliding window must be sufficiently large. But the FHCFO policy guarantees more accurate results with a small sliding window.
- The FHCFO policy requires more CPU time than other buffering methods, but it is still negligible, thus causality query processing can occur in real-time.

6 Conclusion

Data streams collected from sensors contain a large volume of useful information including causal relationships. In this paper, we dealt with causality join query processing for data streams. To perform causality join query processing on sensor streams, it is important to study temporal, spatial, and spatiotemporal relationships between cause and effect in sensor streams. But the traditional FIFO policy, which only considers the transactional time of the sensor data, is not suitable for causality query processing, because of the limitations of the sliding window and delay. In this paper, we carefully examined the temporal, spatial, and spatiotemporal relationships, to satisfy causality join query processing, and proposed new buffering policies for the sliding window based on the results. The intensive experimentation performed in our study shows that the proposed policies are better than the traditional FIFO strategy. The contributions of our work are summarized as follows;

- Causality join query processing on sensor streams. Causality query processing can be used in various applications based on sensor networks.
- Definition of temporal, spatial, and spatiotemporal predicates. These predicates do not deal with transactional time, but rather the valid time of the sensor data, to clearly reflect the causal relationships between cause and effect.
- Novel buffering policies based on temporal, spatial, and spatiotemporal considerations. The experiments show that the spatiotemporal buffering policy is better than FIFO.

Acknowledgements

This work was supported by the Brain Korea 21 Project, 2008, and the Korean Land Spatialization project.

References

[Arasu et al. 03] Arasu, A., Babcock, B., Babu, S., Cieslewicz, J., Datar, M., Ito, K., Motwani, R., Srivastava, U., Widom, J.: "Stream: The Standford Data Stream Management System"; IEEE Data Engineering Bulletin 4, 1 (2003).

[Babcock et al. 02] Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: "Models and Issues in Data Stream Systems"; Proc. 2nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles and Data Systems, ACM, NY (2002), 1-16.

[Chandrasekaran et al. 03] Chandrasekaran, S., Copper, O., Deshpande, A., Franklin, M. J., Joseph M, H., Hong, W., Krishnamurthy, S., Madden, S., Raman, V., Reiss, F., Shah, M.: "Telegraphcq: Continuous Dataflow Processing for an Uncertain World"; Proc. of the Conference on Innovative Data Systems Research (2003), 11-18.

[Ding et al. 03] Ding, L., Rundensteiner, E. A., Heineman, G. T.: "MJoin: A Metadata-aware Stream Join Operator"; Proc. 2nd International Workshop on Distributed Event-based Systems, ACM, NY (2003), 1-8.

[Freedman 04] Freedman, D. A.: "Graphical Models for Causation and the Identification Problem"; Evaluation Review 28, 4 (2004), 267-293.

[Gedik et al. 07] Gedik, B., Wu, K., Yu, P. S., Liu, L.: "GrubJoin: An Adaptive, Multiway, Windowed Stream Join with Time Correlation-Aware CPU Load Shedding"; IEEE Transactions on Knowledge and Data Engineering 19, 10 (2007), 1363-1380.

[Golab and Özsu 03] Golab, L., Özsu, M. T.: "Issues in Data Stream Management"; ACM SIGMOD Record 32, 2 (2003), 5-14.

[Hammad et al. 03] Hammad, M. A., Aref, W. G., Elmagarmid, A. K.: "Stream Window Join: Tracking Moving Objects in Sensor-network Databases"; Proc. 15th International Conference on Scientific and Statistical Database Management, IEEE Computer Society, Washington, D.C., USA (2003), 75-84.

[Holland 86] Holland, P. W.: "Statistics and Causal Inference"; Journal of the American Statistical Association (1986).

[Khor et al. 05] Khor, I. J., Thomas, J., Jonyer, I.: "Sliding Window Protocol for Secure Group Communication in Ad-Hoc Networks"; Journal of Universal Computer Science 11, 1 (2005), 37-55.

[Kun-Lung et al. 07] Wu, K., Yu, P. S., Gedik, B., Hildrum, K. W., Aggarwal, C. C., Bouillet, E., Fan, W., George, D. A., Gu, X., Luo, G., Wang, H.: "Challenges and Experience in Prototyping a Multi-Modal Stream Analytic and Monitoring Application on System S"; Proc. 33th International Conference on Very Large Data Bases, VLDB Endowment (2007), 1185-1195.

[Li et al. 05] Li, H., Lee, S., Shan, M.: "Online Mining Changes of Items over Continuous Append-only and Dynamic Data Streams"; Journal of Universal Computer Science 11, 8 (2005), 1411-1425.

[LTCCS 07] LTCCS (The Large Truck Crash Causation Study) (2007). `http://www.loc.gov/marc/specifications/spechome.html`.

[Madden and Franklin 02] Madden, S., Franklin, M. J.: "Fjording the Stream: An Architecture for Queries over Streaming Sensor Data"; Proc. 18th International Conference on Data Engineering, IEEE Computer Society, Washington, D.C., USA (2002), 555-566.

[Mouratidis and Papadias 07] Mouratidis, K., Papadias, D.: "Continuous Nearest Neighbor Queries over Sliding Windows"; IEEE Transactions on Knowledge and Data Engineering 19, 6 (2007), 789-803.

[Mozer 99] Mozer, M. C.: "An Intelligent Environment Must Be Adaptive"; IEEE Intelligent Systems and Their Applications 14, 2 (1999) 11-13.

[Pearl 00] Pearl, J.: "Models, Reasoning and Inference"; Cambridge University Press (2000)

[Qin and Lee 03] Qin, X., Lee, W.: "Statistical Causality Analysis of INFOSEC Alert Data"; Proc. 6th International Symposium on Recent Advances in Intrusion Detection, LNCS 2820, Springer Berlin/Heidelberg (2003), 73-93.

[Silverstein et al. 00] Silverstein, C., Brin, S., Motwani, R., Ullman, J.: "Scalable Techniques for Mining Causal Structures"; Data Mining and Knowledge Discovery 4, 2-3 (2000), 163-192.

[Tao and Papadias 06] Tao, Y., Papadias, D.: "Maintaining Sliding Window Skylines on Data Streams"; IEEE Transactions on Knowledge and Data Engineering 18, 3 (2006), 377-391.

[Urhan and Franklin 00] Urhan, T., Franklin, M. J.: "XJoin: A Reactively-scheduled Pipelined Join Operator"; IEEE Data Engineering Bulletin 23, 2 (2000), 27-33.

[Wu et al. 07] Wu, J., Tan, K., Zhou, Y.: "Window-oblivious Join: A Data-driven Memory Management Scheme for Stream Join"; Proc. 19th International Conference on Scientific and Statistical Database Management, IEEE Computer Society, Washington, D.C., USA (2007), 21-30.

[Wu et al. 09] Wu, J., Zhou, Y., Aberer, K., Tan, K.: "Towards Integrated and Efficient Scientific Sensor Data Processing: A Database Approach"; Proc. 12th International Conference on Extending Database Technology: Advances in Database Technology, ACM, New York, USA (2009), 922-933.

[Zhou et al. 08] Zhou, Y., Aberer, K., Tan, K.: "Toward Massive Query Optimization in Large-Scale Distributed Stream Systems"; Proc. 9th ACM/IFIP/USENIX International Conference on Middleware, Springer-Verlag, New York, USA (2008), 326-345.