JCSE Journal of Computing Science and Engineering

Regular Paper

Journal of Computing Science and Engineering,
Vol. 5, No. 4, December 2011, pp. 294-304

Load Shedding for Temporal Queries over Data Streams

Mohammed Al-Kateb* **and Byung Suk Lee**
Department of Computer Science, The University of Vermont, Burlington, VT, USA
malkateb@cs.uvm.edu, bslee@cs.uvm.edu

Abstract

Enhancing continuous queries over data streams with *temporal* functions and predicates enriches the expressive power of those queries. While traditional continuous queries retrieve only the values of attributes, temporal continuous queries retrieve the valid time intervals of those values as well. Correctly evaluating such queries requires the *coalescing* of adjacent timestamps for value-equivalent tuples prior to evaluating temporal functions and predicates. For many stream applications, the available computing resources may be too limited to produce exact query results. These limitations are commonly addressed through load shedding and produce approximated query results. There have been many load shedding mechanisms proposed so far, but for *temporal* continuous queries, the presence of coalescing makes theses existing methods unsuitable. In this paper, we propose a new accuracy metric and load shedding algorithm that are suitable for temporal query processing when memory is insufficient. The accuracy metric uses a combination of the Jaccard coefficient to measure the accuracy of attribute values and $\mathcal{PQI}$ interval orders to measure the accuracy of the valid time intervals in the approximate query result. The algorithm employs a greedy strategy combining two objectives reflecting the two accuracy metrics (i.e., value and interval). In the performance study, the proposed greedy algorithm outperforms a conventional random load shedding algorithm by up to an order of magnitude in its achieved accuracy.

Categories: Ubiquitous computing

Keywords: Algorithms; Load shedding; Data streams; Temporal query processing

I. INTRODUCTION

Continuous queries [1] are standing queries, which typically run indefinitely. These queries are central to applications dealing with continuous and unbounded data streams [2]. Many of these applications handle data whose values may *change over time*. For this class of application, enhancing continuous queries with temporal functions and predicates [3] enriches the semantics of queries and, consequently, enables users to define temporal expressions in their queries. In this paper we refer to this class of queries as *continuous temporal queries*.

Unlike traditional continuous queries that typically return only the values of specified attributes, a continuous *temporal* query returns the *valid time interval* of each value as well the attribute values themselves [4]. The selection predicates in this type of query includes a temporal predicate for the time intervals associated with the attribute values satisfying the value predicate.

To guarantee the correctness of the result of a temporal query, adjacent or overlapping timestamps of value-equivalent tuples should be merged prior to the evaluation of the temporal function or predicate specified in the query. This merge process is referred to as *temporal coalescing* [5]. This coalescing is essential for temporal query processing because queries evaluated on uncoalesced data may generate incorrect answers [5, 6]. The same is true for *continuous* temporal queries over data streams [7].

For stream applications, it is not uncommon to have insufficient system resources to process or keep all tuples arriving from the input data stream in memory [2]. In these situations, discarding a fraction of tuples, called load shedding [8], often becomes necessary to resolve the problem. With load shedding

Open Access http://dx.doi.org/10.5626/JCSE.2011.5.4.294 http://jcse.kiise.org

Received 8 May 2011, **Revised** 14 August 2011, **Accepted** 9 November 2011
*Corresponding Author

 pISSN: 1976-4677 eISSN: 2093-8020

in place, however, the system cannot produce an exact result, thus ensuring the accuracy of an approximate query result is an important issue [9].

In this paper, we study the problem of load shedding from a data stream in the presence of temporal coalescing, with a particular focus on the case of insufficient memory to keep all tuples in the window specified in a continuous temporal selection query. The presence of coalescing as an integral step in temporal query processing poses two major challenges for the load shedding mechanism.

The first challenge is that the accuracy metrics used in the existing load shedding methods (e.g., the number of tuples in the approximate result as in [10-16] or the relative deviation between estimated and actual aggregate values as in [17, 18]) are not suitable for continuous temporal queries. Since a temporal query returns both attribute values and their valid time intervals, the accuracy metric should take both into account. The second challenge is a direct consequence of the first challenge, that is, the existing load shedding algorithms (whether random load shedding as in [17] or semantic load shedding as in [11]) are therefore inadequate for temporal queries.

To address the first challenge, we introduce a new accuracy metric that combines a value similarity metric and an interval similarity metric. The proposed metric is not only intuitive and semantically correct but also has theoretical backing. Specifically, the value similarity metric is based on the Jaccard coefficient [19], and the interval similarity metric is based on $\mathcal{PQI}$ interval orders [20]. The Jaccard coefficient is a common statistic used for comparing the similarity between two sets. $\mathcal{PQI}$ interval orders provide a similarity metric for comparing intervals (mapped from elements).

In response to the second challenge, we present a new "*coalescence-aware*" algorithm that keeps the effects of coalescing in mind when deciding which tuples to drop. Making an optimal load-shedding decision requires holding all tuples in memory but, because the available memory is limited, we propose a greedy algorithm that takes a quadratic running time and consumes linear memory space. It employs a greedy strategy that combines two greedy objectives, one targeting the value similarity and the other targeting the interval similarity. More specifically, it tries to minimize the number of extraneous tuples and the length of coalesced intervals lost in the approximate result as a consequence of dropping tuples.

A performance study compares the proposed coalescence-aware algorithm to a conventional random load shedding algorithm. (Note that other existing load-shedding algorithms are not comparable, since they do not work with coalescing.) The main purpose of the performance study is to compare the two mechanisms in terms of the achieved accuracy. The experiment results show that the accuracy improvement of our proposed mechanism over the random mechanism approaches an order of magnitude as the coalescing probability increases and the available memory size decreases. We have also examined the effects of the two greedy objectives on the performance of the proposed coalescence-aware algorithm and confirmed that the objectives are well balanced in their influence on the accuracy achieved by the greedy algorithm.

The main contributions of this paper can be summarized as follows.

- It introduces the new problem of load shedding for continuous *temporal* queries over a windowed data stream.
- It identifies coalescing as an important barrier to overcome, and proposes a new accuracy metric and a new algorithm for load shedding while taking coalescing effects into consideration.
- It presents a thorough performance study on the proposed coalescence-aware algorithm and two partially coalescence-aware algorithms (each reflecting one of the two greedy objectives).

The rest of this paper is organized as follows. Section II provides a relevant background and introduces the model of continuous temporal queries assumed in this paper. Section III presents the proposed accuracy metric and load shedding algorithm. Section IV presents the experiments and discusses the results. Section V reviews related work. Section VI concludes this paper and suggests future work.

II. CONTINUOUS TEMPORAL QUERIES OVER DATA STREAMS

We characterize continuous temporal queries by their support for temporal functions and predicates. These queries can be seen as a hybrid of traditional continuous queries over data streams [1] and temporal queries over temporal databases [4]. In this section, we first summarize the models of these two base query types and then introduce and motivate the model of the hybrid query type.

A. Continuous Queries Over Data Streams

A data stream is an unbounded sequence of tuples, and typical queries on data streams run continuously for an indefinitely long time period [1, 2]. We assume that each tuple arriving from a data stream is timestamped. Additionally, we assume that all tuples arrive in an increasing order of their timestamp and that tuples in a window are maintained in an increasing order of their timestamps.

In many cases, only tuples bounded by a window on a data stream are of interest at any given time [2]. A window may be tuple-based or time -- based depending on whether its size is the number of tuples or the window time span. The solutions presented in this paper --the proposed accuracy metric and the load shedding algorithm- work well with either window model.

An example continuous query is shown below, considering a wireless sensor network in which sensors are mounted on weather boards to collect timestamped topology information along with humidity, temperature, etc. [21]

Example 1. Continuous Query

Assume a query that monitors humidity values in the past 15 *minutes and continuously outputs, at every minute, humidity values and their associated timestamp values if it finds that the humidity exceeds* 75%. *This query can be expressed as follows using the Continuous Query Language (CQL) syntax* [22].

```
SELECT   S.humidity, S.timestamp
FROM     Stream S RANGE 15 MINUTE SLIDES 1 MINUTE
WHERE    S.humidity > 75;
```

In this query syntax, RANGE specifies the window size and SLIDES specifies the sliding window interval.

B. Temporal Queries Over Temporal Databases

Time is a ubiquitous aspect in almost all real-world phenomena that many real-world applications deal with data whose values may change over time. All these applications thus have intrinsic needs to model time, and it has motivated the database community to undertake an extensive study of temporal data management for relational databases [4], semi-structured databases (e.g., XML [23]), and data streams [24].

Temporal databases make it possible for the user to store the history of data and to retrieve this history through temporal queries. The importance of such capabilities has become evident with significant commercial interests ensuring that major DBMS products [25, 26] are currently supporting temporal data management as core features.

Two time dimensions are of interest in the handling of temporal data -- valid time interval and transaction time interval [4]. The former is the time interval during which the stored fact is true in reality. The latter is the time interval during which the fact is present in the database. A temporal database can support the former only (called a valid-time database), the latter only (called a transaction-time database), or both (called a bitemporal database). In this paper we consider a valid-time database.

In addition, temporal databases can support either attribute or tuple timestamping depending on whether the time interval is associated with individual attributes or with the entire tuple of all attributes [4]. In this paper we consider tuple timestamping.

Temporal databases can be queried using Temporal SQL (TSQL) [27], which is essentially SQL that has been extended with temporal predicates and functions. The correct evaluation of temporal predicates and functions demands temporal tuples to be coalesced first. As already mentioned, coalescing is a fundamental operation in any temporal data model, and is essential to temporal query processing since queries evaluated on uncoalesced data may generate incorrect answers [5, 6] (see the example below).

Example 2. Importance of Coalescing

Consider the temporal table Employee in Table 1, *where for simplicity only Andy's employment history records are shown. The second tuple reflects a change of his salary and the third tuple reflects a change of his department.*

Suppose the manager wants to know which employees worked for the R&D Department for more than six consecutive years. The query can be expressed as follows in TSQL [27].

```
SELECT   E.NAME, VALID (E)
FROM     Employee E (Name, Department) as E
WHERE    CAST (VALID (E) AS INTERVAL YEAR) > 6;
```

In this syntax, ***E(Name, Department)*** *specifies coalescing over* ***Name*** *and* ***Department****,* ***VALID(E)*** *returns the valid time interval of tuples in the result, and the* ***CAST*** *clause converts the valid time interval to the specified granularity, i.e., year. The query returns a pairs of values:* ***Name*** *and the interval during which the values of the coalescing attributes stay the same. With coalescing, the first two tuples in* Table 1 *(both of Andy in R&D)*

Table 1. Andy's employment records

Name	Department	Salary	Start	End
Andy	R&D	100 k	2001	2004
Andy	R&D	120 k	2004	2008
Andy	Sales	120 k	2008	Now

are merged because their valid time intervals are adjacent to each other. The merged time interval, [2001, 2008], is greater than six years, and so Andy is in the query result. Without coalescing, in contrast, Andy cannot be in the query result because the time interval of neither the first nor the second tuple alone is greater than six years.

C. Continuous Temporal Queries Over Data Streams

As already mentioned, we see the continuous temporal stream query model as a hybrid of the continuous stream query model and the temporal database query model. What this entails in the temporal representation of tuples is that, when a new tuple, s_i, is added to a window, we model the valid time interval of its preceding tuple, s_{i-1}, as (t_{i-1}, t_i), where t_{i-1} and t_i are the timestamps of s_{i-1} and s_i, respectively. Moreover, inherited from temporal database queries, continuous temporal stream queries output not only the values of the coalesced attributes but also their valid time intervals. As mentioned in Section II-B, the correct evaluation of temporal predicates and functions in a continuous temporal query requires that value-equivalent tuples with adjacent time intervals should be merged before a query can be evaluated for them.

In this paper, we consider continuous temporal *selection* queries (see Example 3 below) as the query type. A temporal selection query is adequate enough for our purpose of studying the effect of load shedding. Besides, we believe queries of this type are useful in a wide range of stream applications.

Example 3. Temporal Continuous Selection Query

In the application assumed in Example 1, consider another query for monitoring the stream of sensor readings across arriving temporal tuples. Every minute, the query processor outputs the humidity value and the associated time interval if it finds that the value has been the same for 3 consecutive minutes or longer. This query can be expressed as follows using the syntax borrowed from CQL [22] *and TSQL* [27].

```
SELECT   S.humidity, VALID (S)
FROM     Stream (humidity) as S RANGE 15 MINUTE
         SLIDES 1 MINUTE
WHERE    CAST (VALID (S) AS INTERVAL MINUTE) >= 3;
```

III. COALESCENCE-AWARE LOAD SHEDDING

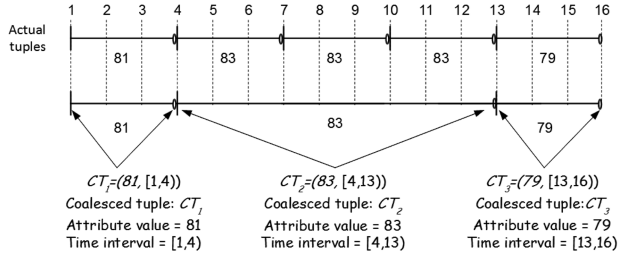
As mentioned in Section I, making load shedding coalescence-aware requires a new accuracy metric and a new load-shedding algorithm because the existing ones do not work with coalescing. In this section, we propose a new accuracy metric in Section III-B and a new algorithm in Section III-C.

A. Preliminaries

Definition 3.1 Coalesced tuple. A coalesced tuple CT_i is defined as a pair of an attribute value and a time interval of the coalesced tuple (Fig. 1).

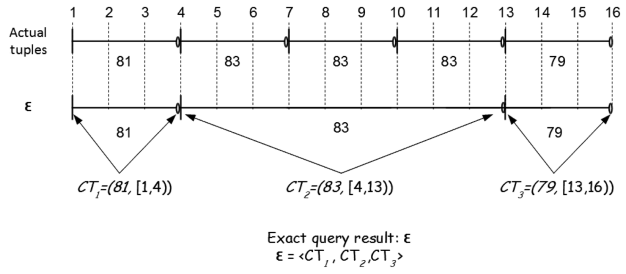
Definition 3.2 Exact query result. An exact result of a continuous temporal query, denoted as $\mathcal{E}$, is defined as a sequence of exact coalesced tuples $\langle CT_1, CT_2, \ldots CT_n \rangle$ (Fig. 2).

Definition 3.3 Approximate query result. An approximate result of a continuous temporal query, denoted as $\mathcal{A}$, is defined as a sequence $\langle A_1, A_2, \ldots A_n \rangle$ where A_i is the set of approximate coalesced tuples CT_{ij} resulting from the exact coalesced tuple CT_i due to load shedding (Fig. 3).



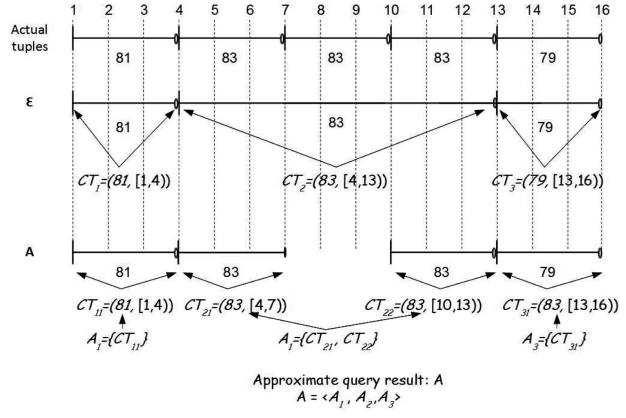
CT_1, CT_2, and CT_3 are coalesced tuples.

Fig. 1. An example of coalesced tuples.



$\mathcal{E}$ is an exact query result.

Fig. 2. An example of an exact query result.



$\mathcal{A}$ is an approximate query result.

Fig. 3. An example of an approximate query result.

Property 1. Bounding Property on Approximate Coalesced Tuples

As a result of load shedding, an exact coalesced tuple CT_i reduces to zero or more approximate coalesced tuples in $A_i \in \mathcal{A}$ such that the attribute values of all the approximate coalesced tuples are the same as that of CT_i and the time interval of every approximate coalesced tuple is contained in that of CT_i.

Proof: This property can be proved through a case analysis. As a result of load shedding, one and only one of the following four cases can happen to CT_i.

- Case 1: CT_i is retained without any alteration. This case happens when none of the tuples coalesced in CT_i is dropped.
- Case 2: An exact coalesced tuple disappears in its entirety. This case occurs when a tuple not coalesced with its adjacent tuples in the exact result is dropped.
- Case 3: An exact coalesced tuple is decreased in its interval. This case occurs when a tuple which is either the first or the last of the tuples coalesced in the exact result is dropped.
- Case 4: An exact coalesced tuple is split into two approximate coalesced tuples of shorter intervals. This case occurs when a tuple in the middle of the tuples coalesced in the exact result is dropped.

It is obvious in all four cases that the bounding property holds.

B. Accuracy Metric

Since the result of a continuous temporal query consists of both the values of coalesced attributes and their associated valid time intervals (recall Definitions 3.2 and 3.3), an accuracy metric for this class of queries should take both into account. In this regard, we propose an accuracy metric based on the concept of the *Jaccard coefficient* [19] and the concept of *$\mathcal{PQI}$ interval orders* [20].

The *Jaccard coefficient*, $J(D_x, D_y)$, measures the similarity between two data sets, D_x and D_y, and is defined as the ratio between the cardinality of their intersection and the cardinality of their union.

$$J(D_x, D_y) = \frac{|D_x \cap D_y|}{|D_x \cup D_y|} \tag{1}$$

We use the Jaccard coefficients to compare the coalesced attribute *values* between the exact and the approximate query result, that is, between $\mathcal{E} \equiv \langle CT_1, \ldots, CT_n \rangle$ and $\mathcal{A} \equiv \langle A_1, \ldots, A_n \rangle$. Note that there may be more than one coalesced tuple in A_i for each CT_i in $\mathcal{E}$ (recall Definitions 3.2, 3.3, and Property 1). We thus need to apply Equation 1 to multisets. That is,

$$J(\mathcal{E}, \mathcal{A}) = \frac{|\mathcal{E} \uplus \mathcal{A}|}{|\mathcal{E} \biguplus \mathcal{A}|} = \frac{\sum_{i:CT_i \in \mathcal{E}} \min(|E_i|, |A_i|)}{\sum_{i:CT_i \in \mathcal{E}} \max(|E_i|, |A_i|)}$$

$$= \frac{\sum_{i:CT_i \in \mathcal{E}} \min(1, |A_i|)}{\sum_{i:CT_i \in \mathcal{E}} \max(1, |A_i|)} \tag{2}$$

where $\uplus$ and $\biguplus$ are the multiset intersection and union operators, respectively (Let $f(\mathcal{D}_1, x)$ and $f(\mathcal{D}_2, x)$ be the multiplicity of

an element x in the multisets $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively. Then, $f(\mathcal{D}_1 \cap \mathcal{D}_2, x)$ and $f(\mathcal{D}_1 \uplus D_2, x)$ are defined as $\min\{f(\mathcal{D}_1, x), f(\mathcal{D}_2, x)\}$, and $\max\{f(\mathcal{D}_1, x), f(\mathcal{D}_2, x)\}$, respectively). Note $|E_i|$ equals 1 since it is a singleton set. The numerator term $\min(1, |A_i|)$ equals 0 only if the exact coalesced tuple disappears as a result of load shedding (case 2 in the proof of Property 1) but otherwise will always equals 1.

Equation 2 considers only the coalesced attribute values, and so it should be modified to consider the interval similarity factor as well. For this purpose, we adopt $\mathcal{PQI}$ interval orders used to compare intervals [20, 28]. In this scheme, given two intervals x and y, the similarity between them is defined as the length of their interval intersection ($\cap$) divided by the length of their interval union ($\cup$).

$$\mathcal{PQI}(x, y) = \frac{|x \cap y|}{|x \cup y|} \tag{3}$$

(See Appendix for more information about $\mathcal{PQI}$ interval orders.)

We use the $\mathcal{PQI}$ interval orders to compare an exact coalesced time interval with the *set* of approximate coalesced time intervals derived from it. Let I_i and I_{ij} be the coalesced intervals of CT_i and CT_{ij}, respectively. We know from Property 1 that I_{ij} is a subinterval of I_i for every j. Hence, the interval similarity for a given exact coalesced tuple is computed as follows.

$$\mathcal{PQI}(CT_i, A_i) = \frac{\sum_{j:CT_{ij} \in A_i} |I_{ij}|}{|I_i|} \tag{4}$$

Equation 4 essentially computes the interval reduction ratio of the exact coalesced tuple CT_i due to load shedding. Its value equals 1 only if CT_i is retained intact as a result of load shedding (the case 1 in the proof of Property 1), but otherwise is always less than 1. In the proposed accuracy metric, we use this ratio to scale the intersection term of CT_i in Equation 2.

$$\mathcal{ACC}(\mathcal{E}, \mathcal{A}) = \frac{\sum_{i:CT_i \in \mathcal{E}} \min(1, |A_i|) \times \mathcal{PQI}(CT_i, A_i)}{\sum_{i:CT_i \in \mathcal{E}} \max(1, |A_i|)} \tag{5}$$

Note that $\min(1, |A_i|)$ equals either 0 or 1 and that if $\min(1, |A_i|) = 0$ (i.e., the case 2) then $\mathcal{PQI}(CT_i, A_i) = 0$. Hence, Equation 5 can be further simplified as follows.

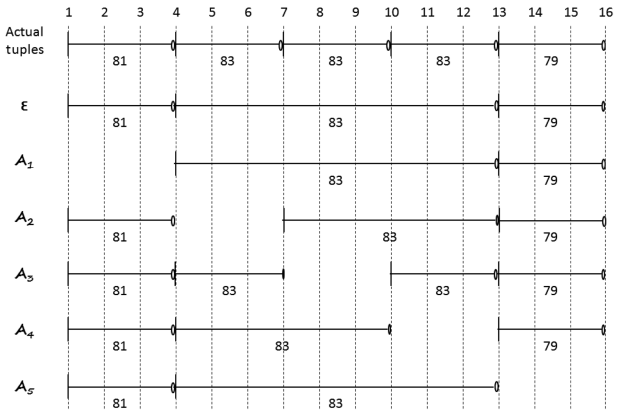
$$\mathcal{ACC}(\mathcal{E}, \mathcal{A}) = \frac{\sum_{i:CT_i \in \mathcal{E}} \mathcal{PQI}(CT_i, A_i)}{\sum_{i:CT_i \in \mathcal{E}} \max(1, |A_i|)} \tag{6}$$

This accuracy metric has the nice property of limiting the computed accuracy to a maximum of 1.0 (i.e., 100%) and allowing for intuitive interpretations as illustrated in the following example.

Example 4. Proposed Accuracy Metric.

As shown in Fig. 4*, consider an exact result ($\mathcal{E}$) – resulting from coalescing five actual tuples – and five possible approximate results ($\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$, $\mathcal{A}_4$, $\mathcal{A}_5$) – each resulting from dropping one of the five actual tuple. Using Equation 6, the accuracy of each approximate result is calculated as follows*.

$$ACC(\mathcal{E}, \mathcal{A}_1) = \frac{\frac{0}{4-1} + \frac{13-4}{13-4} + \frac{16-13}{16-13}}{1+1+1} = \frac{0+1+1}{1+1+1} = \frac{2}{3}$$



$\mathcal{E}$ is the exact query result whereas $\mathcal{A}_1$ through $\mathcal{A}_5$ are approximate query results.

Fig. 4. An example of load shedding.

$$ACC(\mathcal{E}, \mathcal{A}_2) = \frac{\frac{4-1}{4-1} + \frac{13-7}{13-4} + \frac{16-13}{16-13}}{1+1+1} = \frac{0+\frac{6}{9}+1}{1+1+1} = \frac{2.6}{3}$$

$$ACC(\mathcal{E}, \mathcal{A}_3) = \frac{\frac{4-1}{4-1} + \frac{(7-4)+(13-10)}{13-4} + \frac{16-13}{16-13}}{1+2+1} = \frac{1+\frac{6}{9}+1}{1+2+1} = \frac{2.6}{4}$$

$$ACC(\mathcal{E}, \mathcal{A}_4) = \frac{\frac{4-1}{4-1} + \frac{10-4}{13-4} + \frac{16-13}{16-13}}{1+1+1} = \frac{1+\frac{6}{9}+1}{1+1+1} = \frac{2.6}{3}$$

$$ACC(\mathcal{E}, \mathcal{A}_5) = \frac{\frac{4-1}{4-1} + \frac{13-4}{13-4} + 0\frac{0}{16-13}}{1+1+1} = \frac{1+1+0}{1+1+1} = \frac{2}{3}$$

We see that the resulting accuracy is the lowest in $\mathcal{A}_3$, where the third tuple, <83, [7, 10)> is dropped. In this case, it introduces a temporal gap between the second and the fourth tuples (which are coalesced with the third tuple in the exact result). This causes an extra attribute value (hence we get a 2 in the denominator) and a missing part of the exact temporal interval (hence we get 6/9 in the numerator) in the approximate result. On the other hand, the highest accuracy is achieved in either $\mathcal{A}_2$, where the second tuple is dropped, or in $\mathcal{A}_4$ where the fourth tuple is dropped. The reason is that dropping either tuple causes only the missing part of the exact temporal interval (as in $\mathcal{A}_3$) and neither has an extra attribute value (as in $\mathcal{A}_3$) nor is missing the entire interval (as in $\mathcal{A}_1$ and $\mathcal{A}_5$).

C. Coalescence-Aware Load Shedding Algorithm

The problem of coalescence-aware load shedding can be stated formally as follows.

Problem definition: Given the memory M (of size $|M|$ tuples) and a continuous temporal query which specifies coalescing tuples in a window W_S (of size $|W_S|$ tuples) over a data stream S, we discard $|W_S| - |M|$ tuples from the $|W_S|$ tuples with the

objective of maximizing the accuracy of the query output subject to the constraint that $|M| < |W_S|$.

Finding an optimal solution would require holding all $|W_S|$ tuples in memory to decide on the optimal load shedding decision, but the available memory can hold only $|M|$ tuples (as $|M| < |W_S|$). Even if it were possible (which it is not), it would be computationally intractable to find from $\mathcal{O}\left(\binom{|W_S|}{|M|}\right)$ possible subsets an optimal subset of tuples to discard. This problem is harder than the NP-hard nonlinear binary integer programming problem, as there is no well-formed functional form for computing the accuracy for a given set of binary integer assignments (e.g., 1 for retaining a tuple and 0 for discarding a tuple). So, we propose a greedy algorithm, which takes a quadratic running time in the worst case and consumes a linear amount of storage space.

The proposed algorithm decides which tuple to drop from memory upon the arrival of each new tuple from the input data stream. The algorithm uses a greedy strategy based on two objectives for achieving value similarity (Equation 2) and interval similarity (Equation 4).

Objective 1. Dropping a tuple may introduce an extraneous tuple in an approximate result if it causes a temporal gap between tuples that are coalesced in the exact result. Let $\delta^+_{s_j}$ denote the number of such extraneous tuples resulting from dropping a tuple s_j. Note that $\delta^+_{s_j}$ is 1 in the case 4 in the proof of Property 1 and is 0 in other cases. Evidently, we want to minimize the number of extraneous tuples, and, thus, we set the first objective to *drop a tuple with the smallest* $\delta^+_{s_j}$ *(i.e., = 0)* first. A tie is broken by random choice.

Objective 2. Dropping a tuple may also cause either all or part of the time interval of an exact coalesced tuple to be missing in the approximate result. Let $\rho^-_{s_j}$ be the ratio of the length of the time interval missing due to the dropping of s_j over the length of the time interval of the exact coalesced tuple. Note that $\rho^-_{s_j}$ is 0 in the case 1 in the proof of Property 1, is 1 in the case 2, and is bounded between 0 and 1 in other cases. Then, the second objective is to *drop a tuple with the smallest* $\rho^-_{s_j}$ *first* in order to reduce these missing intervals. Like Objective 1, a tie is broken by random choice.

There are different ways to combine $\delta^+_{s_j}$ and $\rho^-_{s_j}$ in the objective function of the greedy algorithm. The sum of them is used here (It can be a *weighted* sum if we want to give a bias in favor of either objective.). We believe summation is better than a product, as the resulting value is always greater than 0. The product gives 0 when $\delta^+_{s_j}$ or $\rho^-_{s_j}$ is 0.

For example, Table 2 shows the values of $\delta^+_{s_j}$ and $\rho^-_{s_j}$ for each of the five actual tuples in Example 4. It shows that the tuples that have a minimum value of $\delta^+_{s_j} + \rho^-_{s_j}$ are s_2 and s_4. The algorithm thus decides to drop either s_2 or s_4. Recall the conclusion in Example 4 that the highest accuracy is achieved in either $\mathcal{A}_2$, where s_2 is dropped, or in $\mathcal{A}_4$ where s_4 is dropped. This demonstrates how the accuracy metric introduced in Section III-B is reflected in our proposed load shedding algorithm.

Table 2. $\delta^+_{s_j}$ and $\rho^-_{s_j}$ for tuples in Example 4.

Tuple	$\delta^+_{s_j}$	$\rho^-_{s_j}$
s_1	0	1
s_2	0	1/3
s_3	1	1/3
s_4	0	1/3
s_5	0	1

Algorithm 1 outlines the steps of load shedding using the combined objective function. Upon the arrival of a new tuple s_i (Line 1), if the memory M is not full yet, s_i is added to W_S (Line 4), but otherwise a tuple s_m which gives a minimum value of $\delta^+_{s_j} + \rho^-_{s_j}$ among all $s_j \in M \cup \{s_i\}$ is found (Lines 7-14) and dropped from the window (Line 16).

Algorithm 1. Coalescing-aware load shedding (CALS)

```
Inputs: S //input data stream
        W_S //set of tuples currently stored in window on stream S
        M //set of tuples that can be stored in memory
 1: for each new tuple s_i arriving in the input stream S do
 2:   if |W_S| < |M| then
 3:     //memory is not full
 4:     insert s_i into W_S;
 5:   else
 6:     //memory is full
 7:     current_min := ∞;
 8:     for each s_j ∈ M ∪ {s_i} do
 9:       compute + δ+_{s_j} and ρ-_{s_j};
10:       if δ+_{s_j} + ρ-_{s_j} < current_min then
11:         current_min := δ+_{s_j} + ρ-_{s_j};
12:         m := j;
13:       end if
14:     end for
15:   end if
16:   remove s_m from M ∪ {s_i};
17: end for
```

It is easy to see that the running-time complexity is $O(|M|^2)$, since the algorithm scans the $|M|+1$ tuples in $M \cup \{s_i\}$ [15] linearly and, for each tuple, the computation of $\delta^+_{s_j}$ and $\rho^-_{s_j}$ takes $O(|M|)$ time in the worst case scenario. The complexity is lower in practice, as computing $\delta^+_{s_j}$ and $\rho^-_{s_j}$ involves only those tuples coalesced to form the same exact tuple. The storage space complexity is $O(|M|)$, as it suffices to have enough memory to hold the $|M|+1$ tuples and two numbers *current_min* and *m*.

In the experiments presented in Section IV, we use two additional algorithms, called coalescence-aware load shedding (CALS)-V and CALS-I. CALS-V is the CALS algorithm reduced to consider only the value similarity (hence using $\delta^+_{s_j}$ only), and CALS-I is that considering only the interval similarity (hence using $\rho^-_{s_j}$ only).

IV. PERFORMANCE STUDY

We conducted experiments to study the performance of the proposed load shedding algorithm. The main objective of the experiments was to observe the effectiveness of the proposed CALS algorithm. Additionally, we compare CALS with two

partially coalescence-aware algorithms, CALS-V and CALS-I, to observe how each of the two complementary greedy objectives affects the performance individually. We use the random load shedding (RAND) algorithm as the baseline approach (Under this random load shedding mechanism, tuples are discarded randomly such that each tuple has the same probability of being discarded from (or, equivalently, retained in) memory.). As anticipated, the experiment results show that CALS achieved the highest accuracy, RAND achieved the lowest, and CALS-V and CALS-I are in between.

The experiments were conducted using Matlab (Release R2010b) on a 64-bit machine with 4.00 GB internal main memory. We use the Matlab array/matrix data type to implement the window data structure.

A. Experiment Setup

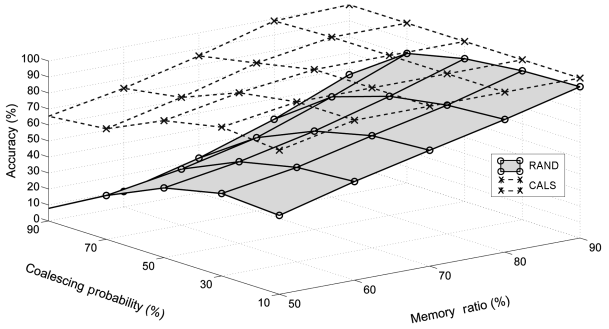
1) Control Parameters: We identified two key parameters influencing the accuracy ofs load shedding algorithms significantly: *memory ratio* and *coalescing probability.* The *coalescing probability* is the expected probability that a tuple in the input stream is coalesced with its preceding tuple upon arrival. The *memory ratio* is the ratio of available memory size over the specified window size, that is, the number of tuples that can be actually stored in the memory divided by the number of tuples that are supposed to be stored in the memory. In our experiments we set the window size to 500 tuples and vary the ratio from 50% to 90%. A natural expectation with these two parameters is that the benefit of coalescence-awareness will be more visible in the resulting accuracy when the coalescing probability is higher or the memory ratio is lower. In addition, we considered the *threshold* for time interval selection, which in our experiments is the lower bound on the selection interval specified in the query (see the selection query in Example 3). The experiment results, however, show that the influence of this parameter value on the accuracy is insignificant.

2) Data Sets: We use five *synthetic data sets* simulating streams with the coalescing probabilities 10%, 30%, 50%, 70%, and 90%, respectively. The timestamp value, t_i, of each tuple in the data sets is selected randomly from within the next 10 increments of time (i.e., within $[t_i, t_i + 10)$). (The particular length of the time unit and the size of the increment, whether constant or varying, are irrelevant for these experiments.) The *real data sets* contain weather measurement data collected from sensors deployed throughout the Intel Berkeley Research Lab to gather timestamped topology information along with humidity, temperature, light intensity, and voltage values. In order to make cases with different coalescing probabilities, we performed coalescing on different coalescing attributes such as humidity, voltage, and light intensity. The resulting coalescing probabilities are approximately 29% (for humidity), 64% (for voltage), and 86% (for light intensity).

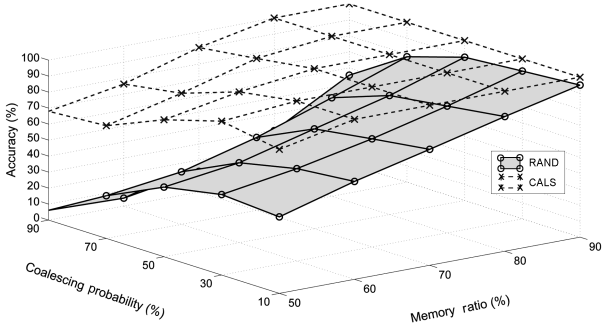
B. Experiment Results

For each algorithm, we measured the achieved accuracy as an average of the accuracies measured using each data set. The results are presented in two stages in this section. The first stage focuses on comparing CALS with RAND to see the effect of coalescence-awareness on the accuracy. The second stage focuses on examining the individual effects of the two partially coalescence-aware greedy objectives.

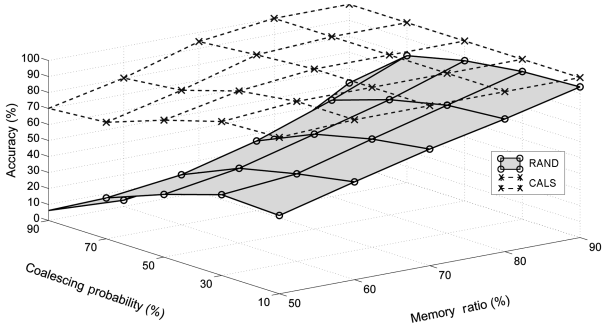
***1) The Effect of Coalescence-Awareness*:** Figs. 5 and 6 show the accuracies achieved by CALS and RAND using the synthetic and real data sets, respectively. Both graphs show that CALS outperforms RAND in the entire range of coalescing probabilities and memory ratios. Furthermore, it clearly shows that the performance advantage of coalescence-awareness increases as the coalescing probability increases and as the available memory size decreases, approaching an order of magnitude as they approach the largest coalescing probability (i.e., 90%) and the lowest memory ratio (i.e., 50%) used in the experiments. In addition, as the figures show, there is little difference in the accuracy for varying the interval selection threshold



(a) Interval threshold = first quartile

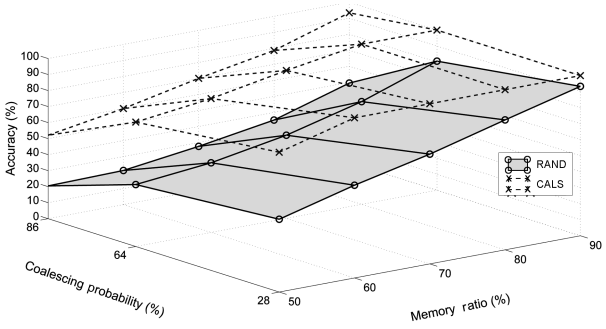


(b) Interval threshold = median

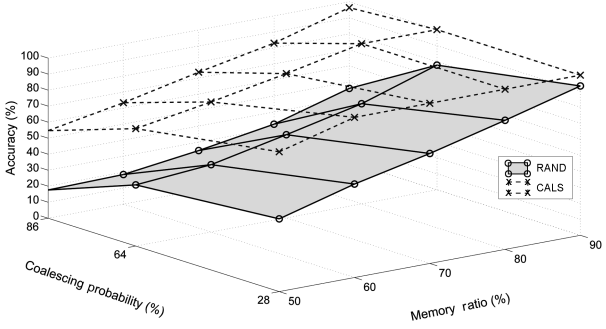


(c) Interval threshold = third quartile

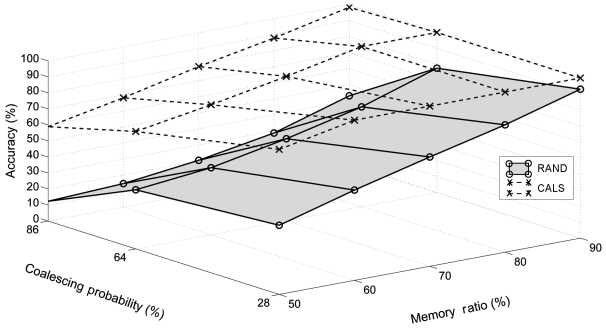
Fig. 5. Accuracies achieved by coalescence-aware load shedding (CALS) and random load shedding (RAND) on synthetic data sets.



(a) Interval threshold = first quartile



(b) Interval threshold = median



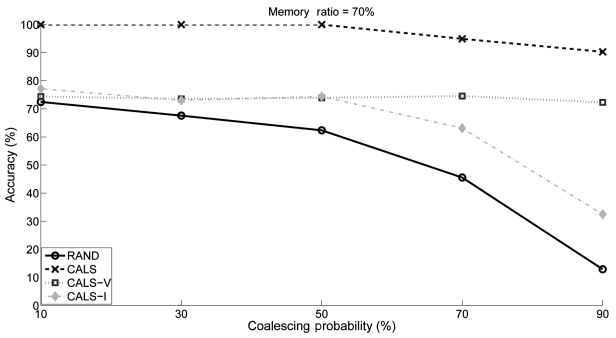
(c) Interval threshold = third quartile

Fig. 6. Accuracies achieved by coalescence-aware load shedding (CALS) and random load shedding (RAND) on real data sets.

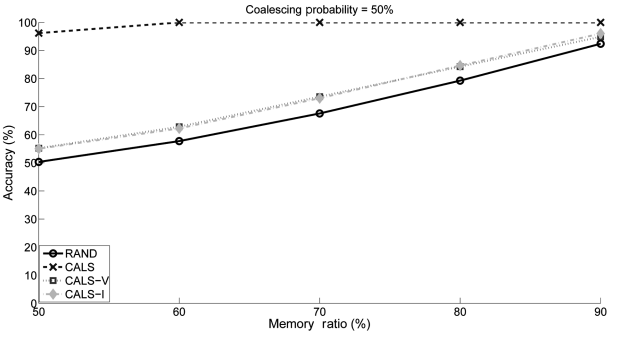
value. This indicates that, while the load shedding produces more tuples of shorter coalesced intervals, the relative difference between the sets of tuples selected in the case of exact and approximate query results does not change much for different threshold values.

***2) The Effects of the Two Greedy Bbjectives*:** Figs. 7 and 8 show the accuracies achieved by all four algorithms (i.e., CALS, CALS-V, CALS-I, RAND) for varying coalescing probabilities and memory ratios, respectively. As expected, the accuracies of CALS-V and CALS-I are bounded between the accuracy of CALS (upper bound) and the accuracy of RAND (lower bound).

In addition, we see that their performance is very close to each other regardless of the memory ratio. This is evident from the fact that the memory size determines the number of tuples to be dropped, and it has equitable effects on the proposed accuracy under both greedy objectives.

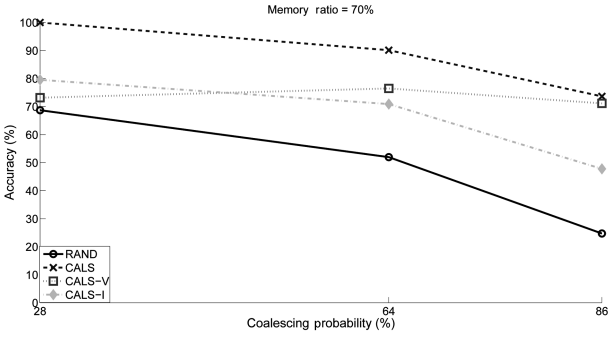


(a) Varying coalescing probability.

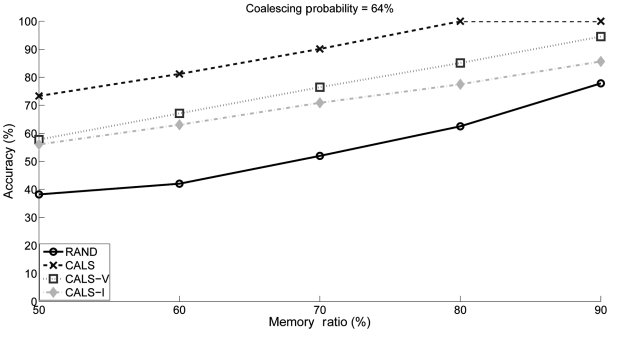


(b) Varying memory ratio.

Fig. 7. Accuracies achieved by coalescence-aware load shedding (CALS), CALS-V, CALS-I, and random load shedding (RAND) on synthetic data sets.



(a) Varying coalescing probability.



(b) Varying memory ratio.

Fig. 8. Accuracies achieved by coalescence-aware load shedding (CALS), CALS-V, CALS-I, and random load shedding (RAND) on real data set.

We also see that the performance of CALS-V changes little as the coalescing probability increases while CALS-I starts degrading as it increases beyond around 50%. Our reasoning is that CALS-V aims at deciding which tuples to retain that would otherwise cause time gaps (hence extraneous tuples) and so has little dependency on the coalescing probability itself, while in CALS-I it is more likely that tuples influencing the accuracy more (even if they have the shortest time interval) are dropped as the coalescing probability increases.

V. RELATED WORK

There are two separate aspects to the related works we look at: load shedding from a data stream and temporal coalescing over data streams.

A. Load Shedding from a Data Stream

The most common accuracy metric assumed in the research literature for load shedding over data streams is the size of approximate query result [10-16]. A few other existing studies, specific to aggregation queries, assume the accuracy metric of the relative error between approximate and actual aggregate values [17, 18]. For both of these accuracy metrics, the accompanying algorithms are generally categorized as random load shedding (i.e., drop tuples randomly, as in [17, 29]) or semantic load shedding (i.e., drop tuples based on their relative importance, as in [11]).

For maximizing the size of an approximate result (known as the max-subset), Das et al. [11] proposed two heuristics for dropping tuples from window-join input streams. The first heuristic sets the priority of a tuple being retained in memory based on the probability that a counterpart joining tuple arrives from the other data stream. The second heuristic favors a tuple based on its own remaining lifetime. Kang et al. [14] proposed a cost-based load shedding mechanism for evaluating window stream joins by adjusting the size of the sliding windows to maximize the number of produced tuples. Ayad and Naughton [10] proposed techniques for optimizing the execution of conjunctive continuous queries under limited computational resources by placing random drop boxes throughout the query plan to maximize the plan throughput. Xie et al. [16] presented a stochastic load shedding mechanism for maximizing the number of result tuples in stream joins by exploiting the statistical properties of the input streams. Gedik et al. [12] proposed a load shedding approach for stream joins that adapts to stream rate and time correlation changes in order to increase the number of output tuples produced by a join query. Then, Gedik et al. [13] further addressed the problem of multiple-way joins over windowed data streams and proposed a window segmentation approach for maximizing the query output rate. Tatbul and Zdonik [15] proposed a load shedding operator that models a data stream as a sequence of logical windows and either drops or retains the entire window depending on the definition and characteristics of the window operator. For minimizing the relative error for aggregation queries, Babcock et al. [17] proposed a random sampling techniques for optimum placement of load shedding operators in aggregation query plans, measuring the resulting accuracy as the relative deviation between estimated and actual aggregate values. Law and Zaniolo [18] proposed a Bayesian load shedding approach for aggregation stream queries with the objective of obtaining an accurate estimate of the aggregation answer with minimal time and space overheads.

In contrast to all this existing work, we address a rather novel query model for data streams under which a query result is composed of both attribute values and the valid time interval of each value. Therefore, all the existing load shedding accuracy metrics and algorithms are not applicable to our research problem.

B. Temporal Coalescing Over Data Streams

Barga et al. [30] proposed to employ temporal coalescing in view of complex event processing over data streams, so that two events are represented as a single event if their valid-time intervals overlap. Zaniolo [31] examined how temporal coalescing can be expressed for a sequence of events using OLAP functions and Kleene-closure constructs. Kramer and Seeger [32] introduced coalescing as a physical operator for compacting the representation of a data stream by merging tuples with identical values and consecutive timestamps into a single tuple. Recently, we studied coalescing for a *windowed* temporal query processing over data streams and addressed the problems of updating a window's extent and optimally selecting between eager and lazy coalescing for concurrent temporal queries [7].

While all this existing work is pertinent to temporal coalescing over data streams, none of them addresses it from the viewpoint of memory being insufficient to retain all tuples. To the best of our knowledge, our work is unique in this regard.

VI. CONCLUSION

This paper addressed the problem of load shedding from a window for continuous temporal queries over data streams when memory is limited. The key challenges come from the fact that tuples need to be coalesced for temporal query processing and that this fact makes existing load-shedding algorithms and the accuracy metric used by them inapplicable. Thus, we proposed a new accuracy metric and a new algorithm that can handle coalescing challenges well. The accuracy metric combines a value similarity metric based on Jaccard coefficients and an interval similarity metric based on $\mathcal{PQI}$ interval orders. The algorithm takes a greedy approach in which our greedy strategy combines two objectives, each objective aims to maximize one of the two types of similarity. The algorithm takes polynomial running time and linear memory space, and the accuracy it achieves is far higher than that achieved by the baseline random load-shedding algorithm. Also, the two objectives combined into the greedy strategy make about equal contributions to the achieved accuracy.

For future work, there are several extensions possible. First, this paper focuses on the limited-memory case, and so the limited-CPU time case is another problem we could look into. Second, this paper considers the temporal selection query, and other query types like aggregation or join queries are possibilities for further work. For this, the accuracy metric and the greedy strategy may need to be adapted to the query type. Third, this paper

considers queries retrieving both the attribute values and their valid time intervals. Considering special cases, such as a snapshot query (retrieving only attribute values) and the valid-time query (retrieving only valid time intervals) [27], may offer simpler and more efficient solutions. This too could be an interesting study.

ACKNOWLEDGMENTS

The authors would like to thank the members of Intel Berkeley Research lab for graciously granting the permission to use their sensor data sets in the experiments. This research is based upon work supported by the National Science Foundation under Grant No. IIS-0415023.

Appendix. $\mathcal{PQI}$ Interval Orders

The $\mathcal{PQI}$ interval orders are used to compare intervals [28]. In this scheme, each element, x, of a given data set, D, is represented by an interval through a lower bound function, $\mathcal{L}(x)$, and an upper bound function, $\mathcal{U}(x)$, such that $\forall x \in D : \mathcal{L}(x) < \mathcal{U}(x)$. They provide three types of binary relations for interval comparison: *strict preference*, *weak preference* (or *hesitation*), and *indifference*.

The strict preference relation (denoted as $\mathcal{P}$) models the case in which the interval of one element precedes the interval of the other element (i.e., interval intersection of the two elements is empty) (Fig. 9a). Formally, given two elements x and y, x is said to be strictly preferred over y if $\mathcal{L}(x) > \mathcal{U}(y)$. The weak preference relation (denoted as $\mathcal{Q}$) is models the case in which the interval of one element overlaps the interval of the other element (i.e., non-empty intersection) (Fig. 9b). Formally, given two elements x and y, x is said to be weakly preferred over y if $\mathcal{U}(x) > \mathcal{U}(y) > \mathcal{L}(x) > \mathcal{L}(y)$. The *indifference* relation (denoted as $\mathcal{I}$) models the case in which the interval of one element contains the entire interval of the other element (Fig. 9c and 9d). Formally, given two elements x and y, x is said to be indifferent from (i.e., similar to) y if $\mathcal{U}(x) > \mathcal{U}(y) > \mathcal{L}(y) > \mathcal{L}(x)$ or $\mathcal{U}(y) > \mathcal{U}(x) > \mathcal{L}(x) > \mathcal{L}(y)$.

The degree of overlap between two intervals can be used as a measure of the *similarity* between them [20]. Formally, given two elements x and y, the similarity between x and y can be defined as follows.

$$\frac{\max\{0,\ \{\min\{\mathcal{U}(x), \mathcal{U}(y)\} - \max\{\mathcal{L}(x),\ \mathcal{L}(y)\}\}\}}{\max\{\mathcal{U}(x), \mathcal{U}(y)\} - \min\{\mathcal{L}(x),\ \mathcal{L}(y)\}} \quad (7)$$

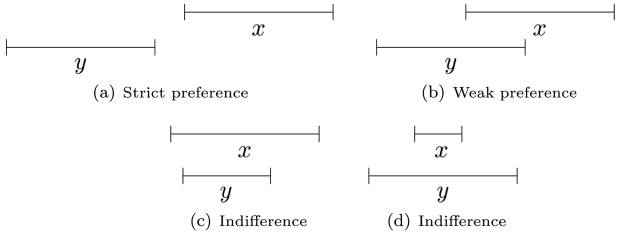


Fig. 9. Example of *PQI* relations between two intervals x and y.

This equation is equivalent to Equation 3 in Section III-B.

REFERENCES

1. S. Babu and J. Widom, "Continuous queries over data streams," *SIGMOD Record*, vol. 30, no. 3, pp. 109-120, 2001.
2. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, Madison, WI, June 3-5, 2002, pp. 1-16.
3. J. Allen, "Maintaining knowledge about temporal intervals," *Communications of the ACM*, vol. 26, no. 11, pp. 832-843, 1983.
4. A. U. Tansel, Temporal Databases: Theory, Design, and Implementation, Redwood City, CA: Benjamin/Cummings Publishing Co., 1993.
5. M. H. Bohlen, R. T. Snodgrass, and M. D. Soo, "Coalescing in temporal databases," *Proceedings of the 22th International Conference on Very Large Data Bases*, Mumbai, India, September 3-6, 1996, pp. 180-191.
6. C. E. Dyreson, "Temporal coalescing with now, granularity, and incomplete information," *ACM SIGMOD International Conference on Management of Data*, San Diego, CA, June 9-12, 2003, pp. 169-180.
7. M. Al-Kateb, S. S. Kunta, and B. S. Lee, "Temporal coalescing on window extents over data streams," *IEICE Transactions on Information and Systems*, vol. E94-D, no. 3, pp. 489-503, 2011.
8. N. Tatbul, "Load shedding," *Encyclopedia of Database Systems*, L. Liu, Ed., Ney York, NY: Springer, 2009, pp. 1632-1636.
9. R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma, "Query processing, approximation, and resource management in a data stream management system," *First Biennial Conference on Innovative Data Systems Research*, Asilomar, CA, January 5-8, 2003.
10. A. M. Ayad and J. F. Naughton, "Static optimization of conjunctive queries with sliding windows over infinite streams," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Paris, France, June 13-18, 2004, pp. 419-430.
11. A. Das, J. Gehrke, and M. Riedewald, "Approximate join processing over data streams," *ACM SIGMOD International Conference on Management of Data*, San Diego, CA, June 9-12, 2003, pp. 40-51.
12. B. Gedik, K. L. Wu, P. S. Yu, and L. Liu, "Adaptive load shedding for windowed stream joins," *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, Bremen, Germany, October 31-November 5, 2005, pp. 171-178.
13. B. Gedik, K. L. Wu, P. S. Yu, and L. Liu, "A load shedding framework and optimizations for M-way windowed stream joins," *Proceedings of the 23rd International Conference on Data Engineering*, Istanbul, Turkey, April 15-20, 2007, pp. 536-545.
14. J. Kang, J. F. Naughton, and S. D. Viglas, "Evaluating window joins over unbounded streams," *Proceedings of the 19th International Conference on Data Engineering*, Bangalore, India, March 5-8, 2003, pp. 341-352.
15. N. Tatbul and S. Zdonik, "Window-aware load shedding for aggregation queries over data streams," *Proceedings of the 32nd International Conference on Very Large Data Bases*, Seoul,

Korea, 2006, pp. 799-810.
16. J. Xie, J. Yang, and Y. Chen, "On joining and caching stochastic streams," *ACM SIGMOD International Conference on Management of Data*, Baltimore, MD, June 14-16, 2005, pp. 359-370.
17. B. Babcock, M. Datar, and R. Motwani, "Load shedding for aggregation queries over data streams," *Proceedings of the 20th International Conference on Data Engineering*, Boston, MA, March 30-April 2, 2004, pp. 350-361.
18. Y. N. Law and C. Zaniolo, "Improving the accuracy of continuous aggregates and mining queries on data streams under load shedding," *International Journal of Business Intelligence and Data Mining*, vol. 3, no. 1, pp. 99-117, 2008.
19. P. Jaccard, "Etude comparative de la distribution orale dans une portion des alpes et des jura," *Bulletin del la Socit Vaudoise des Sciences Naturelles*, vol. 37, pp. 241-272, 1901.
20. M. Ozturk and A. Tsoukias, "Valued hesitation in intervals comparison," *Lecture Notes in Computer Science Vol. 4772: Scalable Uncertainty Management (First International Conference, SUM 2007, Washington,DC, USA, October 10-12, 2007. Proceedings)*, H. Prade and V. Subrahmanian, Eds., Heidelberg, Germany: Springer Berlin, 2007, pp. 157-170.
21. "Intel Lab Data," http://berkeley.intel-research.net/labdata.
22. A. Arasu, S. Babu, and J. Widom, "The CQL continuous query language: semantic foundations and query execution," *VLDB Journal*, vol. 15, no. 2, pp. 121-142, 2006.
23. F. Wang, C. Zaniolo, and X. Zhou, "Temporal XML? SQL strikes back!," *Proceedings of the 12th International Symposium on Temporal Representation and Reasoning*, Burlington, VT, June 23-25, 2005, pp. 47-55.
24. U. Srivastava and J. Widom, "Flexible time management in data stream systems," *Proceedings of the 23rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, Paris, France, June 14-16, 2004, pp. 263-274.
25. "An Oracle White Paper Sep 2009: Oracle Database 11g Workspace Manager Overview," http://www.oracle.com/technetwork/database/twp-appdev-workspace-manager-11g-128289.pdf.
26. "Teradata," http://teradata.us/t/database/teradata-temporal/.
27. R. T. Snodgrass, I. Ahn, G. Ariav, D. Batory, J. Clifford, C. E. Dyreson, R. Elmasrik, F. Grandik, C. S. Jensen, W. Kafer, N. Kline, K. Kulkarni, T. Y. C. Leung, N. Lorentzos, J. F. Roddick, A. Segev, M. D. Soo, and S. M. Sripada, "TSQL2 language specification," *ACM SIGMOD Record*, vol. 23, no. 1, pp. 65-86, 1994.
28. A. Tsoukias and P. Vincke, "A characterization of PQI interval orders," *Discrete Applied Mathematics*, vol. 127, no. 2 SPEC., pp. 387-397, 2003.
29. B. Mozafari and C. Zaniolo, "Optimal load shedding with aggregates and mining queries," *Proceedings of the 26th IEEE International Conference on Data Engineering*, Long Beach, CA, March 1-6, 2010, pp. 76-88.
30. R. S. Barga, J. Goldstein, M. Ali, and M. Hong, "Consistent streaming through time: a vision for event stream processing," *Third Biennial Conference on Innovative Data Systems Research*, Asilomar, CA, January 7-10, 2007, pp. 363-374.
31. C. Zaniolo, "Event-oriented data models and temporal queries in transaction-time databases," *Proceedings of the 16th International Symposium on Temporal Representation and Reasoning*, Bressanone-Brixen, Italy, July 23-25, 2009, pp. 47-53.
32. J. Kramer and B. Seeger, "A temporal foundation for continuous queries over data streams," *Proceedings of the 11th International Conference on Management of Data*, Athens, Greece, 2005, pp. 70-82.

Mohammed Al-Kateb

Mohammed Al-Kateb received his Ph.D. from the Department of Computer Science, University of Vermont. He received his BS and MS degrees in Information Systems from Cairo University. His research interests include data streams processing and temporal data management.

Byung Suk Lee

Byung Suk Lee is Professor of Computer Science at the University of Vermont. His main research interests are database systems, data stream processing, query processing, and event processing. He held several positions in industry and academia: previously at Gold Star Electric, Bell Communications Research, Datacom Global Communications, and University of St. Thomas, and currently at the University of Vermont. He was also a visiting professor at Dartmouth College and a participating guest at Lawrence Livermore National Laboratory. He served on international conferences as a program committee member, a publicity chair, a special session organizer, and a workshop organizer, and also on the review panels of US federal funding agencies. He holds a B.S. degree from Seoul National University, M.S. from KAIST, and Ph.D. from Stanford University.