Adaptive Disorder Control in Continuous Data Streams

Hyeon Gyu Kim[1] Cheolgi Kim[2] Myoung Ho Kim[1]

[1]*Korea Advanced Institute of Science and Technology*
{hgkim, mhkim}@dbserver.kaist.ac.kr

[2]*Information and Communications University*
cheolgi@gmail.com

Abstract

A disorder control is the key factor regarding accuracy and latency of query results when processing sliding window aggregates over continuous data streams. Many stream systems maintain buffers or leverage punctuations for the control. However, current systems suffer from the lack of adaptivity in the measure for estimating buffer sizes or punctuations, which may lead to inaccurate or delayed query results. To address this problem, we propose a probabilistic approach to using an adaptive measure derived from the distributions of tuple generation intervals and network latencies. In our approach, the measure estimates buffer sizes or punctuations according to a drop ratio, which denotes a percentage of tuple drops permissible in run-time processing. The drop ratio can be defined declaratively in a query specification and it provides a way for users to control the tuple drops as their intention. The experimental results show that our adaptive measure estimates more appropriate buffer sizes than ad hoc measures, which means that the proposed measure provides a lower latency, while retaining accuracy by satisfying the given drop ratio.

1. Introduction

There is a substantial class of applications to process real-time results through long-running queries over continuous unbounded streams [3, 6]. In such applications, individual data items (hereafter tuples) are generated from multiple remote sources and continuously transmitted to a query processor, which is also known as *DSMS* (Data Stream Management System) [5, 10]. Well-known examples of the applications consist of sensor networks, network monitoring, financial analysis, telecommunication data management and manufacturing [6, 13].

In the applications handling streams, a sliding window is an important query feature [2, 7]. A window specifies a moving view that decomposes an unbound stream into finite subsets called *window extents*. The window extents are defined usually based on timestamps of tuples. There are two common ways in which timestamps are assigned to stream tuples [9]: Tuples are timestamped on an entry to the query processor, and remote sources timestamp the tuples before sending them to the processor. Timestamps of the former case are referred to as *system timestamps* and timestamps of the latter as *application timestamps*.

When evaluating extents of a sliding window, to achieve semantic correctness, a system usually needs to process tuples in an increasing timestamp order [2, 9]. However, when tuples are transferred from remote sources, their arrival may not be in an increasing order due to different network latencies. Such out-of-order arrival of tuples complicates the process of determining the window extents and may lead to inaccurate or delayed query results.

In order to resolve these issues, two common approaches have been widely used. One is to maintain buffers to sort tuples, and the other is to leverage *punctuations* [1]. The former case can be found in *Aurora* project [10]. In their research, they assume that a bound of network latency is known in advance, and from the assumption, the buffer has a fixed size that is large enough to cover the bound. The later case is discussed in *WID* approach [2] and *STREAM* project [9]. Briefly, a punctuation τ indicates that no more tuples having a timestamp greater than τ will be seen in the stream. When receiving τ, tuples having a timestamp less than or equal to τ can be processed in a sliding window. The punctuation can be either given by remote sources or estimated by a system.

There still remain some issues when applying the existing approaches to real-world stream applications.

This work was supported by grant No. (R01-2006-000-10809-0) from the Basic Research Program of the Korea Science & Engineering Foundation.

In buffer-based approaches, the bound of network latency may not be known in advance. Since the bound can grow larger than a pre-defined size of the buffer, any out-of-order tuples having a larger size than the bound are dropped, and such tuple drops may lead to inaccurate results. In punctuation-based approaches, the punctuation itself can be disordered by network latency when it is given by remote sources. In other cases, the estimation is usually conducted by ad hoc measures, not a theoretical one. Thus, in both cases, it is hard to expect an accurate query result.

This paper presents a generalized solution to support both ways of using buffers or punctuations in stream applications while resolving the issues stated above. For this purpose, we introduce a *disorder controller* which is placed at the entry to a sliding window. Disorder controllers can be configured to use either buffers or punctuations. When using buffers, if a prior knowledge about network latency is available, a fixed buffer is used. Otherwise, a dynamic buffer is used in the controller. When using punctuations, the controller conducts estimation using an adaptive measure which is theoretically derived. The measure is also applied for the controllers configured to use dynamic buffers.

Such configuration of disorder controllers can be defined declaratively in a query specification using a SQL-like language. Moreover, our approach provides a way for users to control tuple drops as their intention. It is achieved by declaring a *drop ratio* in the specification, which denotes a percentage of tuple drops permissible in run-time processing. Given a drop ratio, our measure conducts estimation based on it.

The rest of the paper is organized as follows. After a short review of related work in Section 2, Section 3 presents a brief overview of a disorder controller. Section 4 introduces query language features to specify the configuration of disorder controllers. Section 5 describes derivation steps of the proposed measure for adaptive estimation. Section 6 shows experimental results and Section 7 concludes our discussion.

2. Related Work

In *Aurora* project [10], operators having a sliding window can be defined with a *slack* parameter to deal with out-of-order tuples. The slack parameter denotes a size of the buffer, which is placed at the entry to a sliding window and sorts out-of-order tuples. In their research, they assume that a bound of network latency is always known in advance. From the assumption, the slack buffer has a fixed size of *n*. Any tuples that are more than *n* positions out of order are dropped. Note

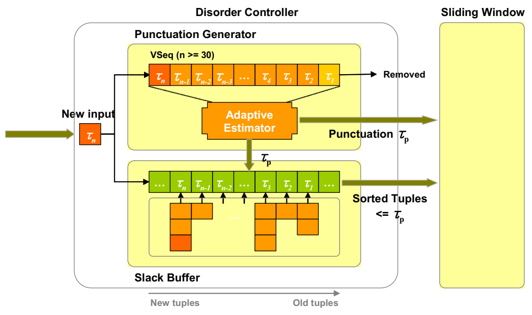

Disorder Controller
Sliding Window
Punctuation Generator
VSeq (n >= 30)
τ_n τ_{n-1} τ_{n-2} τ_{n-3} … τ_4 τ_3 τ_2 τ_1
Removed
New input
τ_n
Adaptive Estimator
Punctuation τ_p
τ_p
… τ_n τ_{n-1} τ_{n-2} … τ_3 τ_2 τ_1 …
Sorted Tuples <= τ_p
Slack Buffer
New tuples
Old tuples

Figure 1: A disorder controller

that, if prior knowledge about the bound is not available and the bound is fluctuated, it is hard to be used.

There are also a number of studies relevant to the disorder control based on the punctuations [1]. In *WID* approach [2], the punctuations are assumed to be given by stream sources. In *Niagara* [11], punctuations are either assumed to be provided within tuples, or if generated, are based on simple notions of slack as in Aurora. In *STREAM* [5], the proposed solution is based on the *heartbeats*, which can be thought of as special types of punctuations [8, 9]. The heartbeats are usually provided by internal estimation of a system. However, the estimation is conducted by an ad hoc measure, which may lead to inaccurate results.

3. Disorder Controller

This paper presents a generalized solution to support both ways of using buffers or punctuations in stream applications. For this purpose, we introduce a *disorder controller*, which is placed at the entry to a sliding window as depicted in Figure 1. The disorder controller can be configured to use one of the ways to control out-of-order tuples. Our approach enables users to define such a configuration declaratively in a query specification using a SQL-like language.

In Figure 1, if a disorder controller is configured to use punctuations, the controller has only an upper part called *Punctuation Generator* that includes an adaptive measure denoted as *Adaptive Estimator*. The measure estimates the punctuations adaptively according to network latencies which are continuously changed. To get information about network latencies, it maintains a circular list called *VSeq*, which accumulates an amount of the latest tuples arrived at the controller.

When a disorder controller is configured to use buffers, if a buffer size is fixed, the controller just makes use of a lower part denoted as *Slack Buffer* in the figure. In the lower part, the buffer maintains tuples in an increasing order of application timestamps.

Otherwise the buffer is of a variable size, the controller must decide when tuples in the buffer should be presented to a sliding window. To decide the point, the measure for estimating punctuations is reused, and in this case, both parts of the figure are used. Note that estimating punctuations implies to deciding buffer sizes in our model from this reason.

When dynamic buffers or punctuations are used, our approach enables users to define a *drop ratio* in a query specification, which provides a way to control tuple drops as intended by users. The drop ratio denotes how many percent of tuples are allowed to be dropped when query processing. Given the drop ratio, our measure references the ratio as a basis for run-time estimation.

4. Query Specification

This section introduces query language features for specifying the configuration of disorder controllers. We adopt syntaxes proposed in WID approach [2] for specifying window queries. In the approach, a window specification consists of three parameters: *RANGE*, *SLIDE* and *WATTR*, which specify the length of the window, the step by which the window moves, and the windowing attribute – the attribute over which RANGE and SLIDE are specified.

To specify a disorder controller having a fixed buffer, we define an optional parameter called *SLACK* to denote a buffer size. The following shows how to use it in a specification.

```
Q1: SELECT MAX(speed)
    FROM Traffic [RANGE 5 minutes
           SLIDE 1 minute
           WATTR timestamp
           SLACK 10]
```

If a prior knowledge of the network latency is not available, it is desirable to use a dynamic buffer. In this case, a drop ratio should be provided as a basis for run-time estimation as stated earlier. To specify the drop ratio, we define an optional parameter called *DRATIO*. The parameter is followed by a percentage indicating the permissible ratio of tuple drops. The next example shows how to use it in a specification.

```
Q2: SELECT MAX(speed)
    FROM Traffic [RANGE 5 minutes
           SLIDE 1 minute
           WATTR timestamp
           DRATIO 5%
           SLACK 20]
```

In above example, SLACK is used together with DRATIO. When both parameters are declared, SLACK denotes a maximum buffer size which is available to a disorder controller. In this case, SLACK has a higher priority than DRATIO when evaluating in a run time. That is, in upper example, a disorder controller starts estimation based on the drop ratio of 5%, but if the estimated buffer size exceeds the given slack size of 20, the buffer size will be fixed to 20 and the drop ratio has no effect on estimation until the size returns to be smaller than 20.

To specify a disorder controller using punctuations, both parameters can be used again. But there is one restriction that the value of SLACK is set to be 0, since the controller does not maintain any slack buffer.

5. Adaptive Estimation

This section describes derivation steps of our measure for estimating punctuations, which is also applied for disorder controllers having dynamic buffers. The section starts with preliminaries such as problem statements and some assumptions, and then explains how we derive the measure based on the assumptions. At the end of this section, we give an algorithm for estimating punctuations and discuss time complexities.

5.1. Preliminaries

In our approach, tuple drops are controlled by a drop ratio which is defined in a query specification. A disorder controller should carefully estimate punctuations to keep a total number of tuple drops from violating the given drop ratio. Note that a tuple drop is presented whenever the tuple carries a timestamp less than or equal to a punctuation previously estimated. Let τ_p be an application timestamp of the punctuation to be estimated and $\mathcal{T}_{n+1}$ be a random variable for an application timestamp of tuple that will be arrived after the punctuation τ_p. Then an expression to estimate the punctuation can be written as follows.

$$\{ \tau_p \in \max(\mathcal{T}) \mid \Pr(\mathcal{T}_{n+1} < \mathcal{T}) < \Pr_d \text{ for } \mathcal{T}, \mathcal{T}_{n+1} \in \mathrm{T} \} \quad \ldots (1)$$

For convenience, in the remaining part of this paper, we use conventions that $\mathcal{T}_i$ denotes a random variable for the application timestamp of a tuple and T_i a variable for the system timestamp of the tuple. All of the timestamps are an element of a discrete and ordered time domain T. In addition, $\Pr_d$ denotes a drop ratio of DRATIO given in a specification.

In order to derive an estimation measure, we make two assumptions such that an interval of tuple generations in stream sources has an exponential distribution with a mean of θ (2), and a transmission delay from different network latencies follows a normal distribution with a mean of μ and a standard deviation of σ (3).

$$(\mathcal{T}_i - \mathcal{T}_{i-1}) \sim Exp(\theta) \quad \ldots (2)$$

$$(T_i - \mathcal{T}_i) \sim N(\mu, \sigma) \quad \ldots (3)$$

In above assumptions, the θ, μ and σ can be deducted by sensing a number of latest tuples. For this purpose, we introduce a circular list called *VSeq*, which accumulates timestamp information of the latest tuples arrived at a disorder controller. The size of VSeq is continuously changed according to estimation results from the upper distributions, and in our approach, it is always larger than or equal to 30.

Based on the information of VSeq, the θ can be calculated simply by the following equation (4), where n specifies a size of VSeq, T_1 a system timestamp of the earliest tuple in VSeq, and T_n a system timestamp of the latest one.

$$\theta = (T_n - T_1) / n \qquad \ldots (4)$$

The μ and σ can also be estimated from VSeq. The following steps show how μ is estimated. It simply removes the earliest delay and adds a new delay to get the sum of delays in VSeq, and then divide the sum with n to get μ. The steps for estimating σ are similar with these. In the below, *tail* and *head* denotes each pointer indicating a tail node and a head node in the VSeq, and r is the latest tuple arrived.

```
$sum \Leftarrow sum - (VSeq[tail].T - VSeq[tail].\mathcal{T}) + (r.T - r.\mathcal{T})$;
$\mu \Leftarrow sum / n$;
$VSeq[head].T \Leftarrow r.T$;
$VSeq[head].\mathcal{T} \Leftarrow r.\mathcal{T}$;
```

5.2. Derivation of the measure

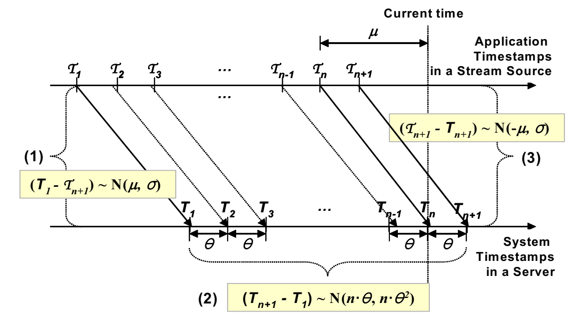
This part explains derivation steps of our measure for estimating punctuations based on above assumptions. Before discussing the steps, we present the distribution of an interval between the earliest system timestamp in VSeq and the latest one. In the below expression, T_1 is the earliest one and T_n the latest one.

$$(T_n - T_1) \sim N((n-1)\cdot\theta, \sqrt{(n-1)\cdot\theta^2}) \qquad \ldots (5)$$

Derivation:

Let $\mathcal{V}_i$ be an interval $(T_i - T_{i-1})$ between system timestamps of consequent tuples in VSeq, then $\mathcal{V}_i$ also has an exponential distribution from the assumption (2). The mean θ of the distribution can also be calculated by the equation (4).

Since number of $\mathcal{V}_i$ is equal to n-1 and the n is larger than or equal to 30 in our approach, it is large enough to approximate the sum of $\mathcal{V}_i$, that is $(T_n - T_1)$, to a normal distribution according to the *Central Limit Theorem* [12]. In the distribution of $(T_n - T_1)$, A mean is calculated by multiplying the mean of $\mathcal{V}_i$ n-1 times because of the independence of $\mathcal{V}_i$. A standard deviation is obtained in a same way, where the standard deviation of $\mathcal{V}_i$ is θ^2 from the fact that $\mathcal{V}_i$ follows an exponential distribution. □



Current time
μ
Application Timestamps in a Stream Source
$\mathcal{T}_1$ $\mathcal{T}_2$ $\mathcal{T}_3$ … $\mathcal{T}_{n-1}$ $\mathcal{T}_n$ $\mathcal{T}_{n+1}$
(1)
$(T_1 - \mathcal{T}_{n+1}) \sim N(\mu, \sigma)$
$(\mathcal{T}_{n+1} - T_{n+1}) \sim N(-\mu, \sigma)$
(3)
T_1 T_2 T_3 … T_{n-1} T_n T_{n+1}
θ
System Timestamps in a Server
(2)
$(T_{n+1} - T_1) \sim N(n\cdot\theta, n\cdot\theta^2)$

Figure 2: Decomposition of $(\mathcal{T}_{n+1} - \mathcal{T}_1)$

Based on the distribution (5) and the previous assumptions, it can be predicted whether a future tuple is dropped or not. When predicting the tuple drop, we assume that a generation interval and a network delay follow the current distributions. That is, the future tuple is generated after θ from the time that the last tuple is occurred and transferred to a system after μ with the variance of σ, where the θ, μ and σ are the parameters currently estimated.

The following equation is to estimate a drop ratio of the future tuple based on the information of VSeq. In the below, $\mathcal{T}_{n+1}$ is a random variable for an application timestamp of the future tuple and $\mathcal{T}_1$ a variable for the earliest timestamp in VSeq, which is same as a current punctuation τ_p.

$$\Pr(\mathcal{T}_{n+1} < \mathcal{T}_1) = Z\left(\frac{n\theta}{\sqrt{2\sigma^2 + n\theta^2}}\right) \qquad \ldots (6)$$

Derivation:

In the upper equation, the left term can be rewritten as follows.

$\Pr(\mathcal{T}_{n+1} < \mathcal{T}_1) = \Pr(\mathcal{T}_{n+1} - \mathcal{T}_1 < 0)$

Again, $\Pr(\mathcal{T}_{n+1} - \mathcal{T}_1 < 0)$ can be decomposed with three sub terms as shown in Figure 2. In the figure, T_i is a system timestamp corresponding to an application timestamp $\mathcal{T}_i$.

$(\mathcal{T}_{n+1} - \mathcal{T}_1) = (\mathcal{T}_{n+1} - T_{n+1}) + (T_{n+1} - T_1) + (T_1 - \mathcal{T}_1)$

In the decomposition, both random variables relevant to the sub terms $(\mathcal{T}_{n+1} - T_{n+1})$ and $(T_1 - \mathcal{T}_1)$ follow a normal distribution from the assumption (3).

$(\mathcal{T}_{n+1} - T_{n+1}) \sim N(-\mu, \sigma)$ ... (6.1)
$(T_1 - \mathcal{T}_1) \sim N(\mu, \sigma)$ ... (6.2)

Also a random variable relevant to the sub term $(T_{n+1} - T_1)$ has a normal distribution from the derived distribution (5).

$(T_{n+1} - T_1) \sim N(n\cdot\theta, n\cdot\theta^2)$ … (6.3)

Using the derived distributions from (6.1) to (6.3) and *MGF* (*Moment Generating Function*) of a normal distribution [12], The term $(\mathcal{T}_{n+1} - \mathcal{T}_1)$ can be transformed as follows.

$$\begin{aligned}M_{\mathcal{T}_{n+1}-\mathcal{T}_1}(s) &= M_{\mathcal{T}_{n+1}-T_{n+1}}(s)\cdot M_{T_{n+1}-T_1}(s)\cdot M_{T_1-\mathcal{T}_1}(s)\\ &= e^{(\delta^2 s^2/2)-\mu s}\cdot e^{(n\theta^2 s^2/2)+n\theta s}\cdot e^{(\delta^2 s^2/2)+\mu s}\\ &= e^{\{(\delta^2 s^2/2)-\mu s\}+\{(n\theta^2 s^2/2)+n\theta s\}+\{(\delta^2 s^2/2)+\mu s\}}\\ &= e^{\{(2\delta^2+n\theta^2)s^2/2+n\theta s\}}\end{aligned}$$

The result is again in the form of normal distribution MGF. From this, a random variable ($\mathcal{T}_{n+1}$ - $\mathcal{T}_1$) has a normal distribution such as:

$$(\mathcal{T}_{n+1} - \mathcal{T}_1) \sim N(n\cdot\theta, \sqrt{2\sigma^2 + n\cdot\theta^2})$$

After normalization of the above, we can finally obtain an equation for estimating a probability of the future tuple drop.

$$\Pr(\mathcal{T}_{n+1} < \mathcal{T}_1) = Z\left(\frac{n\theta}{\sqrt{2\sigma^2+n\theta^2}}\right) \square$$

Given VSeq, the equation (6) estimates a probability of the future tuple drop. Observe that if we have a $\Pr_d$ from a query specification, it is possible to obtain a size of VSeq after slight modification of the equation (6). An equation to get the size has a form of the following: In the right term of equation (6), n becomes a variable intended to be estimated, while a given $\Pr_d$ becomes a constant instead of the left term of the equation.

The next expression is to get the size of VSeq when $\Pr_d$ is given. In the below, c is a constant which is a square of z-value corresponding to the given $\Pr_d$, and N denotes a set of natural numbers.

$$\{\, n_p \in \max(n) \mid n^2 - c\cdot n - 2\cdot c\cdot\sigma^2 / \theta^2 < 0 \text{ for } n \in N \,\} \ldots (7)$$

Derivation:

As stated earlier, this is a form of an inequality that n becomes a variable intended to be estimated in the equation (6). The n_p denotes the maximum value of n which satisfies the above expression.

$$Z\left(\frac{n\theta}{\sqrt{2\sigma^2+n\theta^2}}\right) < Z(\Pr_d) \rightarrow \frac{n\theta}{\sqrt{2\sigma^2+n\theta^2}} < Z(\Pr_d) \rightarrow$$

$$(n\theta)^2 < Z(\Pr_d)^2(2\sigma^2 + n\theta^2) \rightarrow \cdots$$

The desired result can be obtained by derivations from the above left-most inequality toward a right direction. $\square$

From the size of VSeq satisfying the given $\Pr_d$, a punctuation τ_p can be easily obtained. The equation for this purpose is described in the following. In the below, t_n denotes the latest system timestamp in VSeq.

$$\tau_p = \tau_n - n_p\cdot\theta \;\; (\, \tau_n = t_n - \mu \,) \qquad \ldots (8)$$

Derivation:

Remind that an interval between all consecutive tuples in VSeq is θ as discussed in earlier part. From this, given n_p, a region satisfying the current $\Pr_d$ can be calculated by $n_p\cdot\theta$.

```
Adaptive (tuple r)
1. VSeq(n_p) ← t and τ of r;
2. If (Ready(VSeq)==true) {
3.     μ, σ and θ ← VSeq(n_p);
4.     n_p ← Eqn7(c, σ, θ);
5.     τ_p ← Eqn8(t, n_p, θ);
6.     VSeq ← VSeq(n_p);
7.}
8. return τ_p;
```

Figure 3: Algorithm *Adaptive*

In addition, the maximum application timestamp τ_n can simply be obtained by (t_n - μ) from the assumption of network delay (3). Consequently, the punctuation τ_p satisfying the $\Pr_d$ is derived by subtracting the region $n_p\cdot\theta$ from the maximum application timestamp τ_n. $\square$

5.3. Algorithm

In the previous part of this section, we explain derivation steps to estimate punctuations based on the information maintained in VSeq. In the derivation steps, the size of VSeq satisfying the given $\Pr_d$ is calculated by the inequality (7), and from the size n_p, the punctuation τ_p is simply obtained by the equation (8). These two steps are continuously performed by a disorder controller whenever a new tuple arrives.

The algorithm described in Figure 3 shows these steps written in pseudo codes. In the algorithm, *Eqn7* denotes a function playing a role of the inequality (7) and *Eqn8* a function of the equation (8). VSeq(n) denotes a VSeq having a size of n. The function *Ready* checks whether VSeq is filled up. If *Ready* returns false, an estimation process is not activated and a previously calculated punctuation is simply returned.

Time complexity of our algorithm is O(n) for any tuple arrival. The steps from (1) to (5) take only a constant time apparently. However, the step (6) may have a time complexity of O(n) in some cases. If the estimated size of n_p is larger than or equal to the current size, there is only a processing effort to add new nodes in the circular list. Furthermore, no estimation will be occurred until the list is filled up. But, in a converse case, a number of nodes should be removed one by one from the list and a sum of the timestamps also be refreshed. Such a case requires an iteration, which means a time complexity of O(n).

6. Experimental Results

We implemented a disorder controller and connected it to *TinyDB* [18] for data generation. In our

```
Adhoc (tuple r)
1~3. … // same as the steps of algorithm Adaptive
3.     $\mu_t$, $\sigma_t$ and $\theta$ ← $VSeq(n_p)$;
4.     $\tau_p$ ← $\mu_t - \sigma_t + (\sigma_t * \Pr_d * 2)$;
5.     $n_p$ ← $N + (\sigma_t * \mathrm{zval}(\Pr_d) * \theta)$;
6~8. … // also same
```

Figure 4: Algorithm *Adhoc*

experiment, we configured TinyDB to generate data from 20 sensors in every second, and varied drop ratios of the disorder controller as to 15%, 10%, 5%, 2.5% and 1%. As a result, our measure didn't violate the given ratios in terms of accuracy.

To check adaptivity of our measure, we compared it with an ad hoc measure, which reflects characteristics of an input stream for estimation, but not theoretically derived. The algorithm for the ad hoc measure, which is shown in Figure 4, is similar with our algorithm in that it uses parameters such as μ, σ and θ deducted from an input stream as described in the steps 1 to 3 of Figure 3. But it is different in that it uses an intuitive measure for estimating punctuations which corresponds to the step 4 and 5.

Figure 5 shows estimation results in terms of buffer sizes and average waiting times between the two measures. The results show that our measure *Adaptive* is more adaptive than the *Adhoc* in that the proposed measure estimates smaller buffer size and provides a lower average waiting time.

7. Conclusions

In this paper, we presented a generalized solution to support both ways of using buffers or punctuations in stream applications. For this purpose, we introduce a *disorder controller* which configuration can be defined declaratively using a SQL-like language. When the controller is configured to use dynamic buffers or punctuations, it conducts estimation using an adaptive measure which is derived theoretically based on the distributions of tuple generation intervals and network latencies. Our experimental results show that the estimation measure has merits in terms of accuracy and adaptivity when compared with ad hoc measures. We believed that our method can be extended to cover other various causes of disorder in data streams such as merging unsynchronized streams or data prioritization, and are currently working on the issues.

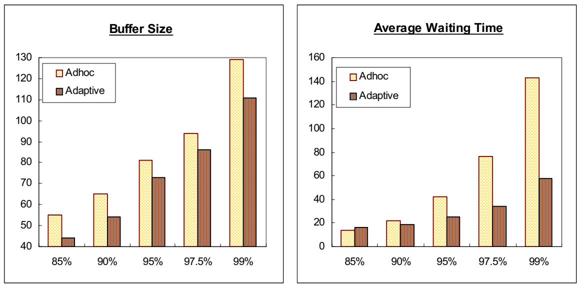


Figure 5: Estimation Results between *Adaptive* and *Adhoc*

8. References

[1] Peter A. Tucker, David Maier, Time Sheard, Leonidas Fegaras, "Exploiting Punctuation Semantics in Continuous Data Streams", *IEEE Transactions on Knowledge and Data Engineering*, May/June 2003.

[2] Jin Li, David Maier, Kristin Tufte, Vassilis Papadimos, Peter A. Tucker, "Semantics and Evaluation Techniques for Window Aggregates in Data Streams", *ACM SIGMOD*, June 14–16, 2005.

[3] S. Babu and J. Widom, "Continuous Queries over Data Streams", *ACM SIGMOD Record*, Sep. 2001.

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and Issues in Data Stream Systems", *ACM PODS*, June 2002.

[5] Arvind Arasu et al, "STREAM: The Stanford Data Stream Management System", *IEEE Data Engineering Bulletin*, Vol. 26 No. 1, March 2003.

[6] Rajeev Motwani et al, "Query Proessing, Resource Management, and Approximation in a Data Stream Management System", *CIDR*, Jan. 2003.

[7] A. Arasu, S. Babu and J. Widom, "The CQL Continuous Query Language: Semantic Foundations and Query Execution", Stanford University Technical Report, Oct. 2003.

[8] S. Babu, U. Srivastava and J. Widom, "Exploiting k-Constraints to Reduce Memory Overhead in Continuous Queries over Data Streams", In *ACM TODS*, Sep. 2004.

[9] U. Srivastava and J. Widom. "Flexible Time Management in Data Stream Systems", In Proc. of *PODS* 2004, June 2004.

[10] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, S. Zdonik. "Aurora: A New Model and Architecture for Data Stream Management", In *VLDB Journal* (12)2: 120-139, August 2003.

[11] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. "NiagaraCQ: A scalable continuous query system for internet databases". *ACM SIGMOD*, pages 379–390, May 2000.

[12] Dimitry P. Bertsekas and John N. Tsitsiklis, "Introduction to Probability: International Edition", Athena Scientific, Belmont, Massachusetts, 2002.

[13] Stream Query Repository: http://www-db.stanford.edu/stream/sqr/.

[14] TinyDB: http://www.tinyos.net.