A Scalable Approach to Approximating Aggregate Queries over Intermittent Streams *

Shanzhong Zhu and Chinya Ravishankar
Department of Computer Science and Engineering
University of California, Riverside
Riverside, CA 92521
{*szhu, ravi*}*@cs.ucr.edu*

Abstract

We present a novel approach to approximate evaluation of standing aggregate queries over streaming data, subject to user-specified error bounds. Our method models the behavior of aggregates as Brownian motions, and adaptively updates the model according to stream characteristics. This approach has two advantages. First, it greatly improves system scalability since we can defer query evaluation as long as the difference between the returned and true aggregate values remains within user-specified bounds. Second, we are able to provide approximate answers during stream interruptions by estimating the rate at which the streams and the aggregate drift during the blackout periods. We also study processor allocation issues in such approximate aggregate evaluation systems. Our experiments show that our model captures the behavior of real-world streams such as sensor data and stock traces with excellent fidelity, and scales very well for large numbers of standing queries.

1. Introduction

There has been increasing recent interest in managing *data streams* [4, 6]. Streaming data, unlike relational data, are continuous in principle, of unbounded extent, and need not be persistent. Examples of streaming data include real time tick-by-tick stock data [3] and sensor data (temperature, pressure, illumination, and so on) [2]. Applications of such streaming data abound, and include real-time financial applications [29], network traffic monitoring systems [8], sensor monitoring systems [20, 19], security systems, and manufacturing process.

* This work was supprted in part by grants from Tata Consultancy Services, Inc., the Digital Media Innovations program of the University of California, and by the Fault-Tolerant Networks program of the Defense Advanced Research Projects Agency, under contract F30602-01-2-0536.

Aggregates are commonly used to summarize large volumes of streaming data, and typically involve functions such as *AVG*, *SUM*, *COUNT*, and *MIN/MAX*. Evaluation of aggregate queries over data streams has already received some attention [13, 11, 23, 5, 20]. In our work, we consider *standing* (continuous) aggregate queries [4], which remain in effect on streams indefinitely. For example, a facilities manager who monitors the temperatures from a large number of sensors deployed in a building may wish to be notified when the average temperature of the third floor changes by more than $2^\circ F$. Similarly, a brokerage may manage several hundred thousands of stock portfolios over thousands of stocks, and each customer may wish to be notified when portfolio changes by a certain prespecified amount, say 1%.

In practice, it is generally unnecessary to rigidly evaluate aggregate queries each time a new stream item arrives, since users can typically tolerate some errors in the query results. Such flexibility can be used to enhance query processor performance. In the examples above, a naive approach would evaluate all aggregates each time a new item arrives, and generate notifications when the aggregate value exceeds the prespecified bound. If each standing query is assigned a thread, this approach clearly does not scale [7].

1.1. Approximate Query Evaluation by Deferral

Our work is a new approach to trading off aggregate result precision for query processor scalability, by deferring query evaluations as long as the error in a cached value for the aggregate remains within user-specified bounds.

Scalability is particularly important when the number of standing queries is large. For example, a real-time stock monitoring system may handle a few thousand stock streams, but monitor the values of millions of portfolios [3]. In such a system, re-evaluating all standing queries whenever a stream item arrives will be expensive, even if each aggregate is easy to compute. We will incur significant system overheads, including thread switching and cache

Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM'04)
1099-3371/04 $ 20.00 © 2004 IEEE

misses as the query processor switches between aggregates [7] (see Figure 7 for a comparison of scalability under the naive approach and our adaptive approach). The problem clearly worsens as the rate of incoming data streams increases.

A *train scheduling* technique is proposed in the *Aurora* data stream manager [6, 7] to improve system scalability. Their goal is to queue as many items as possible for each query operator, and process the complete train at once. This approach reduces the overall item processing costs, but may fail to deliver the results in a timely manner due to queuing delays. Our approach achieves high scalability by avoiding processing some stream items, and delivers results promptly.

1.2. Handling Stream Interruptions

An implicit, albeit unrealistic, assumption made commonly is that streams are unbroken and uninterrupted. In practice, however, sensors could malfunction or links could become congested temporarily, leading to data loss or delay. Query processing must continue regardless, especially for standing queries. Standard methods for interpolating data lost during the interruption is not always viable, since stream data usually contains a stochastic component.

In our work, we address two challenges, namely, how to improve system scalability given user tolerance to error, and how to continue approximate aggregate evaluation when stream interruptions occur.

1.3. Our Model

Our approach is to model numerical data streams as *Brownian motions* [12]. Brownian motion models are appropriate when increments in values are independent Normal distributions, and are widely used to characterize fluctuating data in fields like finance, engineering, telecommunication, and physics. We show in section 3 that many important streaming data such as stock traces and sensor data can be modeled as Brownian motions. We then show how to model stream aggregates, such as *AVG, SUM, MIN* and *MAX*, based on Brownian motion streams. This model yields parameters that allow us to estimate how far the current values of streams and aggregates are likely to have drifted from previous values.

Our approach avoids aggregate evaluations each time a new data item arrives. Each aggregate query A includes a user-defined error bound ϵ and a belief threshold p. We defer A's evaluation as long as our Brownian motion model estimates its value to be within ϵ of its previous value, with probability p. If some stream input to A is interrupted, our approach allows us to estimate the increments in aggregate values, and generate error bounds at probability level p. We show that this is an accurate and useful model for significant classes of streaming data, such as sensor and stock streams.

We update the Brownian motion model on-line, since stream characteristics can change over time. Fluctuations in sensed temperatures will depend on wind conditions, and the intra-day volatility of stocks may depend on market sentiment. When stream values have larger fluctuations, more frequent evaluations are required. Consequently, our approach raises some processor allocation issues. Given a set of aggregate queries with error bounds, and a query processor with fixed computing capacity, we must, at any time, be concerned about whether we can schedule all query evaluations so that their error bounds are met. If the processor is so overloaded that some evaluations cannot meet their error bounds, we aim to schedule these evaluations intelligently so that the total incurred error can be minimized.

We organize the rest of this paper as follows: Section 2 reviews some related work. In Section 3, we show that some numerical streams can be modeled as Brownian motions. Section 4 discusses how to adaptively trigger aggregate evaluations based on our Brownian motion models. Section 5 briefly discusses how to schedule aggregate evaluations according to system load. In Section 6, we conduct experiments that evaluate our adaptive aggregation scheme on both stock traces and sensor data. Section 7 concludes our work.

2. Related Work

A summary of models and issues in data stream system appears in [4], and discusses approximation techniques including sampling [22], histograms [17], and wavelets [15]. Such techniques focus on building bounded summary information on unbounded data streams, and have been shown to be quite efficient for traditional query operators.

Approximating aggregate queries over data streams has already received attentions [11, 13, 16]. Dobra et al. [11] propose techniques to approximately answer complex aggregate queries based on sketches. Gehrke et al. [13] propose single-pass techniques to approximate correlated aggregates based on histograms. Since the exact evaluation of the aggregates is expensive, both aim to provide imprecise answers. In contrast, our approximation scheme is based on deferral of query evaluations, so that aggregates are selectively reevaluated at times guided by stream characteristics and the user's error bounds.

Our system can also be regarded as a stream monitoring system [6, 18, 19, 29]. In *Aurora* [6], data flows through a graph of primitive operators, such as select, merge, and join, and an output stream is returned. Operator batching and tuple batching (train scheduling) techniques [7] are applied to reduce scheduling and operator overheads, and thus

improve system scalability. In the *StatStream* system [29], stream correlation and β value are monitored online, using *Discrete Fourier Transform (DFT)* approximation and grid data structure to reduce computation. While the goal of *StatStream* is to discover all pairs of streams with correlation bigger than the prespecified threshold, our goal is to report all significant changes (more than users' error bound) of stream aggregates with low overheads.

We also design a scheduler to schedule aggregate evaluations under real-time constraint. Load shedding techniques [10, 25] have been proposed to selectively shed loads when system is overloaded. Our scheduler aims to carefully sequence pending jobs so that the total incurred error can be minimized under overload.

TRAPP [5, 23, 24] investigates techniques to trade off precision (user-provided *precision constraint*) for communication overheads in replicated data environments. They consider how to reduce communication overhead in an environment where distributed data sources with limited resources continuously stream updates to a central server [23], and how to monitor streams with top k values in a similar environment [5]. Our work, unlike TRAPP, assumes no cooperation at remote sources. Besides, our focus is on query processor scalability instead of communication overhead, and how to provide best available query answers over intermittent streams.

3. Modeling the Streams

We consider data streams of numerical values, such as stock price streams, sensored temperature streams, and network traffic load streams. Numerical streams usually display random fluctuations, making it hard to predict their behaviors.

Brownian motion [12] is a stochastic model widely used to characterize fluctuation of random data. A stochastic process W_t is called a *Standard Brownian motion (SBM)* if it satisfies three conditions: (1) $W_0 = 0$, (2) $W_t - W_s$ is normally distributed with mean 0 and variance $t - s$, and (3) $W_t - W_s$ is independent of $W_v - W_u$ if (s, t) and (u, v) are non-overlapping time intervals. Property (2) is the key property of SBM, meaning every increment of SBM follows standard Normal distribution. In general, SBM is a Martingale process [14], meaning loosely that the best estimate for its future value is its current value.

We can generalize SBM to *Drifting Brownian Motion (DBM)* by introducing a secular drift in the expectation of the process (see Figure 1). A DBM process S_t can be modeled by the following difference equation

$$\Delta S_t \ = \ \mu_t \Delta t + \sigma_t \Delta W_t, \qquad (1)$$

where μ_t and σ_t are time dependent parameters. Fundamentally, DBM is a combination of a predictable linear trend

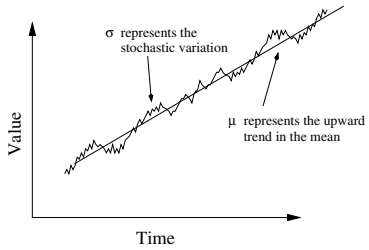


Figure 1: Brownian motion with drift

and a Brownian motion process. The term $\mu_t \Delta t$ represents the non-stochastic part of the process, and characterizes the current moving trend. The term $\sigma_t \Delta W_t$ is the stochastic or Brownian motion part, and represents the randomness in the data. At time t, the process increment ΔS_t follows the Normal distribution $(\mu_t \Delta t,\ \sigma_t^2 \Delta t)$.

In earlier work [28], we have shown experimentally that numerical streams such as stock price traces and sensed temperature streams can be modeled as DBMs. We applied the *Wilk-Shapiro* test [26] to verify that the stream increments were normal, and concluded that these numerical streaming sources can be modeled as Brownian motions with high confidence under small time intervals (μ_t and σ_t are relatively constant over small time intervals).

According to the DBM model, the behavior of each stream can be characterized by the *drift* and *diffusion* parameters μ and σ, respectively. As explained above, μ models a secular upward or downward trend in the mean of the stream data values, and σ models the variance or randomness associated with the stream. A higher σ means that stream values are likely to exceed the user-specified error bound sooner, so higher estimates for σ trigger more frequent aggregate evaluations. Stream behavior is determined by σ in the short term, and by μ in the longer term. Both parameters need to be estimated on a regular basis (see Section 4.6).

In the rest of this paper, we show how to predict the behavior of aggregates provided individual streams are modeled as DBMs.

4. Query Processing

Let $\mathcal{S} = \{S_1, S_2, \cdots, S_n\}$ be n continuous data streams streaming into the query processor (QP) (see Figure 2). Standing queries are of the form (A, ϵ, p), in which A is the aggregate query, and ϵ and p are the user-specified error bound and the belief threshold, respectively. The error bound is the error in query results that users can tolerate, and the belief threshold represents the desired confidence that this error is within bounds.

We consider two categories of aggregate queries: *linear aggregates* and *min/max over a time window*. Each linear

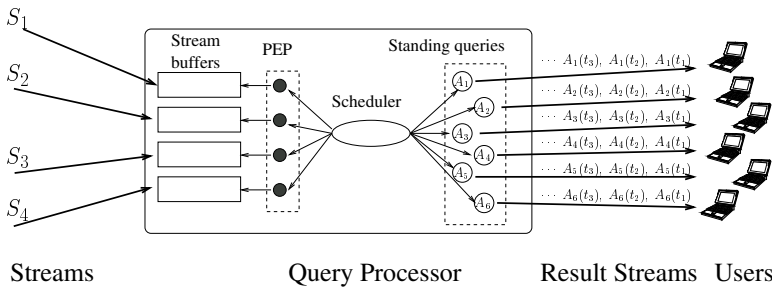


Figure 2: The architecture. The QP accepts standing queries and returns result streams

aggregate is a time dependent value:

$$A(t) \ = \ \sum_{S_i \in \Omega} a_i S_i(t), \tag{2}$$

computed as weighted sum over a subset of streams $\Omega \subseteq \mathcal{S}$. Aggregates such as *SUM*, *AVG*, *COUNT* are in this category.

We also discuss *min/max* aggregates over time windows (see Section 4.3). Consider the queries *"Return the maximum value of my stock portfolio in the last hour"*, and *"Give me today's minimum average temperature on the 2nd floor"*. The former is a *sliding window query*, and the latter is a *landmark window query* [13]. Both queries are very common in streaming applications.

Let $O(t)$ be the result stream of the aggregate A. Our approach evaluates A only at times $t_1 < t_2 < t_3 < \cdots$, so that $O(t)$ will comprise $\{A(t_1), A(t_2), A(t_3), \cdots\}$. These evaluation times for A are not equally spaced, since the characteristics of the component streams may change over time. Therefore, to make our scheme adaptive, we must decide $\Delta t_k = t_{k+1} - t_k$ as we evaluate $A(t_k)$ at time t_k.

Consider the following probability function:

$$F(t, \Delta t) = \Pr[|A(t + \Delta t) - A(t)| \leq \epsilon] \tag{3}$$

$F(t, \Delta t)$ is the probability that the increment of aggregate A is within bound ϵ at time $t + \Delta t$, given that the last evaluation time was t. Therefore, whenever Δt is such that $F(t, \Delta t) > p$, we believe that the aggregate value is still within ϵ of its previous value after time Δt. Clearly, A must be re-evaluated before $F(t, \Delta t)$ drops below the belief threshold p. In other words, the deadline for the next evaluation of A is determined by the smallest Δt for which $F(t, \Delta t) \leq p$.

Section 4.1 describes the architecture of our QP. Section 4.2 and Section 4.3 discuss how to apply the Brownian motion to approximately evaluate linear aggregates and windowed min/max aggregates, respectively. We discuss how to adaptively evaluate aggregates according to user-specified error bounds in Section 4.4 and how to compute approximate aggregates during streams interruptions in Section 4.5. Finally, we address the issue of on-line parameter estimation.

4.1. Query Processing Architecture

As Figure 2 shows, stream values are generated at remote sources (sensors, routers, etc), and pushed to the QP continuously. Each arriving stream item is of the form: (ts, val), where ts is the timestamp when the item is generated at the source and val is its value. Stream items are immediately delivered to the QP after they are generated at the sources. We assume constant network delays from remote sources to the QP. A buffer is associated with each stream, where the k most recent items of the stream are buffered. An on-line *Parameter Estimation Process* (PEP) repeatedly estimates μ and σ based on the item values in the buffer (see Section 4.6).

Standing aggregate queries are registered at the QP by users. The QP is responsible for adaptively evaluating the aggregates, and streaming results back to the users.

Each pending aggregate evaluation must be scheduled to complete before a deadline determined by its next evaluation. When streams have high σ, or when the number of aggregates is large, the query evaluation rate may be high enough to saturate the CPU. The scheduler must dynamically detect such overload, and behave appropriately. Thus, the QP must also address a real-time scheduling issue.

4.2. Linear Aggregates

We begin by showing that a linear aggregate is also a DBM when its component streams are DBMs. Stream S_i is governed by the equation:

$$\Delta S_i(t, \Delta t) = \mu_i(t)\Delta t + \sigma_i(t)\Delta W_t \tag{4}$$

As before, $\mu_i(t)$ and $\sigma_i(t)$ are the drift and diffusion parameters, respectively. W_t is a standard Brownian motion, and ΔW_t follows Normal distribution $(0, \Delta t)$. Equation 4 can be rewritten as

$$\Delta S_i(t, \Delta t) \sim N(\mu_i(t)\Delta t,\ \sigma_i^2(t)\Delta t), \tag{5}$$

showing that the increment of each data stream is normally distributed, and that the mean and variance are both dependent on Δt. A linear combination of Normals is also Nor-

mal, so Equations 2 and 5 together yield

$$\Delta A(t,\Delta t) \sim N\left(\mu_A(t)\Delta t,\ \sigma_A^2(t)\Delta t\right),$$
$$\mu_A(t) = \sum_{i=1}^{n}\left(a_i\mu_i(t)\right),\ \sigma_A^2(t) = \sum_{i=1}^{n}\left(a_i\sigma_i(t)\right)^2, \quad (6)$$

showing that the aggregate increments of DBM streams are also Normal, so that the aggregate $A(t)$ can be also modeled as a drifting Brownian motion.

Equations 3 and 6 clearly indicate that $F(t,\Delta t)$ is a decreasing function of Δt. Thus, it suffices to find Δt for which $F(t,\Delta t) = p$. We must hence solve,

$$\int_{-\epsilon}^{\epsilon} \frac{1}{\sqrt{2\pi\sigma_A^2\Delta t}} exp(\frac{-(x-\mu_A\Delta t)^2}{2\sigma_A^2\Delta t})dx = p. \quad (7)$$

Despite its imposing look, Equation 7 is a routine evaluation of the error function, and yields the next time to re-evaluate the aggregate to ensure that the expected error is within bounds. Since Brownian motions are Martingales [14], we can repeatedly calculate Δt on-line.

Equation 6 can be also rewritten as:

$$A(t+\Delta t) - (A(t) + \mu_A(t)\Delta t) \sim N\left(0,\ \sigma_A^2(t)\Delta t\right) \quad (8)$$

The term $A(t) + \mu_A(t)\Delta t$ is the expected value of A at time $t + \Delta t$. Thus, instead of only returning users the aggregate value $A(t)$ at those evaluation times, we can return users pair of values $((A(t), \mu_A(t))$ so that users can obtain better estimates of the aggregate, $A(t) + \mu_A(t)\Delta t$, at future time $t + \Delta t$. In this case, the next aggregate evaluation time can be determined using Equation 7 with $\mu_A = 0$.

Either Equation 6 or Equation 8 can be used to adaptively predict aggregate evaluation times. In our experiments, we use Equation 6 to calculate Δt.

We so far only considered absolute error bounds, but relative error bounds are also frequently of interest. For example, a user may need to know the average temperature within a given percentage error bound. If we denote the relative error bound as $\epsilon_r\%$, at time t, we need to find Δt such that:

$$\Pr[|A(t+\Delta t) - A(t)| \le \epsilon_r\% \cdot |A(t)|] = p \quad (9)$$

Now treating relative error case is similar to treating absolute error case as described before.

4.3. Min/Max over Time Windows

We show how to apply the Brownian motion model to the deferred approximate evaluation of min and max aggregates over time windows, triggering reevaluation only when the error is expected to exceed ϵ . We only consider the max aggregate, since min is similar.

We first consider the max over *sliding windows*. If the window size is w and the current time is t, the max of linear aggregate $A(\tau)$ over the sliding window $[t-w, t]$ is defined as

$$M(t,w) = \max\{A(\tau) : t - w \le \tau \le t\} \quad (10)$$

As in the case of linear aggregates, our goal is to define a series of evaluation times $\{t_k\}$ so that we remain within the error bound ϵ. The following theorem presents how to adaptively calculate Δt, and is justified in [27].

Theorem 1. *Let M be the windowed* max *function defined over aggregate A, as in Equation 10. Let M have last been evaluated at t_k, and let A have attained its maximum A_m at time t_m during the last evaluation window $[t_k - w, t_k]$. To remain within error bound ϵ, M's next evaluation must occur at time t_{k+1}, where $t_{k+1} - t_k = \min\{\varphi_1, \varphi_2\}$, and*

$$\varphi_1 = \min\{\tau : M(t_k, w-\tau) \le A_m - \epsilon,\ 0 \le w - \tau \le t_k - t_m\} \text{ or } w \text{ if no such } \tau \text{ exists},$$
$$\varphi_2 = \min\{\tau : \Pr[A(t_k+\tau) \le A_m + \epsilon] \le p\}, \quad (11)$$

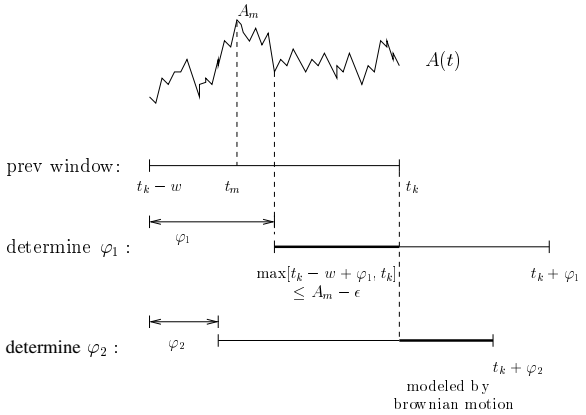


Figure 3: The next evaluation time for max **aggregate:** $\Delta t = \min\{\varphi_1, \varphi_2\}$

The significance of φ_1 and φ_2 is shown in Figure 3. In window $[t_k - w, t_k]$, M was determined to be A_m, attained at time t_m. As the current window slides forward, the trailing edge $t_k - w + \varphi_1$ is the time when the max aggregate over window $[t_k - w + \varphi_1, t_k]$ first drops below $A_m - \epsilon$. If the aggregate A continues to drop, the max value M over the current window will equal $A_m - \epsilon$ at this point, so that reevaluation must be triggered when its leading edge is at $t_k + \varphi_1$. If the max over $[t_k - w + \varphi_1, t_k]$ is always above $A_m - \epsilon$, we reevaluate after time w.

If however, A does not continue to drop, we must model its behavior into the future using the Brownian motion. Clearly, the next reevaluation must be triggered at time $t_k + \varphi_2$, before A is about to exceed $A_m + \epsilon$. Thus, the next reevaluation time t_{k+1} must be the smaller of φ_1 and φ_2, i.e., $t_{k+1} - t_k = \min\{\varphi_1, \varphi_2\}$. φ_1 can be computed from one scan of window $[t_m, t_k]$. φ_2 is equivalent to $\varphi_2 =$

$\min\{\tau : \Pr[A(t_k+\tau) - A(t_k) \leq A_m - A(t_k) + \epsilon] \leq p\}$, which can be similarly solved as Equation 3.

In [13], another type of time window based query, *landmark window* query, is introduced, defined as

$$M_L(t,l) = \max\{A(\tau) : l \leq \tau \leq t\}, \qquad (12)$$

where l denotes a landmark time. In this case, max aggregates over a growing window, whose trailing edge is fixed at l. Since max is monotonic over the landmark window, t_{k+1} is determined by φ_2 alone, the time when the aggregate A deviates beyond $A_m + \epsilon$.

4.4. Deferring Query Evaluations

As explained above, each time the QP evaluates aggregate A, it will also compute Δt (Equation 7 and 11) to determine the next evaluation time for the error bounds and probability constraints to be met. Since stream items are generated and delivered at discrete times, we must wait until the stream items with timestamps [1] most close to $ts_l + \Delta t$ have arrived to perform the next aggregate evaluation (ts_l is the timestamp of the current latest item involved in A). Since we assume constant network delays from the sources to the QP, the stream items in question will arrive at the QP within next Δt time. The adaptive aggregate evaluation scheme is shown in Algorithm 1.

t_{curr} is the current QP system time. t_{eval} is the next aggregate evaluation time, which is incremented by Δt at every evaluation instance. A_{apprx} and ϵ_{intr} are the approximate aggregate result and its associated error during stream interruptions (see Section 4.5). T is a time constant which dictates the interval between consecutive aggregate evaluations (and parameter estimations) during interruptions.

4.5. Handling Stream Interruptions

At time t, assume we are trying to evaluate the query $(A(\Omega), \epsilon, p)$, $\Omega = \{S_1, S_2, \cdots, S_m\}$. Let the most recent estimates for stream parameters be $\{(\mu_i, \sigma_i), i = 1 \cdots m\}$, estimated at times $\{t_i, i = 1 \cdots m\}$, $(t_i \leq t)$, respectively. Further, let the subset $I \subseteq \Omega$ of streams be inactive at time t, having suffered interruptions at various times before t.

Though the stochastic components σ_i of the interrupted streams are completely unpredictable, the deterministic components μ_i provide good estimates for the secular trend in each interrupted stream. Let $\hat{S}_i(t)$ be the best available estimate for stream S_i at time t. Clearly, $\hat{S}_i(t) = S_i(t)$ for uninterrupted stream $S_i \in \Omega - I$, and $\hat{S}_i(t) = S_i(t_i) + \mu_i \cdot (t - t_i)$, for interrupted stream $S_i \in I$. The best approximate aggregate value (both for linear aggregate and min/max aggregate) that QP can provide at time t is

$$A_{apprx}(t) = A(\hat{S}_1(t), \hat{S}_2(t), \ldots, \hat{S}_m(t)). \qquad (13)$$

Along with this approximate result, we also return the expected error ϵ_{intr} associated with the result. For linear aggregates, the expected error in the aggregate is the weighted average of the expected errors in the interrupted component streams, which can be obtained from Equation 7. For min/max over time windows, the expected error is bounded by the expected error of its associated linear aggregate. We use the stochastic behavior of the linear aggregate to approximate the min/max error.

Algorithm 1 Adaptive Aggregate Evaluation Scheme

```
Standing Query (A, ε, p) Execution
loop
  if t_curr == t_eval then
    /* time to reevaluate aggregate value */
    if no stream interruptions detected then
      Evaluate the aggregate A_exact, return A_exact;
      Calculate Δt according to Equation 7 or 11;
      t_eval = t_curr + Δt;
    else
      /* some streams are interrupted */
      Calculate A_apprx according to Equation 13;
      Calculate ε_intr;
      Return pair (A_approx, ε_intr);
      t_eval = t_curr + T;
    end if
  else
    /* no evaluation needed since the error is within bound */
  end if
end loop
```

4.6. On-line Parameter Estimation

We now turn to the issue of estimating the time-dependent drift parameter $\mu_i(t)$ and diffusion parameter $\sigma_i(t)$ for each stream S_i. As mentioned in Section 4.1, The QP maintains the latest k items for each stream S_i in a buffer B_i: $[(ts_0, val_0), (ts_1, val_1), \cdots, (ts_{k-1}, val_{k-1})]$. If a new stream item of S_i arrives and sees B_i full, it simply displaces the oldest item in B_i. We use the contents of buffer B_i to estimate μ_i and σ_i at any time.

Let y_j be the increment of two adjacent stream items in B_i: $y_j = val_j - val_{j-1}$ $(1 \leq j \leq k-1)$, and δ_j be the interval between their generation times: $\delta_j = ts_j - ts_{j-1}$. Since y_j is a Normal sample with distribution $N(\mu_i\delta_j, \sigma_i^2\delta_j)$, buffer B_i provides $k-1$ independent Normal samples $\{y_1, y_2, \cdots, y_{k-1}\}$, and μ_i and σ_i are estimated using these samples. The issue of parameter estimation has been extensively studied in the statistics domain, and maximum-likelihood (ML) estimators are typically used [21].

Given independent Normal samples $y_1, y_2, \cdots, y_{k-1}$ of

1 Formally, stream items with $ts = \max\{t : t \leq ts_l + \Delta t\}$

Algorithm 2 On-line Parameter Estimation Process

on-line estimation process for stream S_i
loop
  if $t_{curr} == t_{estm}$ **then**
    / time to re-estimate parameters */*
    if no stream interruption detected **then**
      Update $\mu_i(t_{curr})$ according to Equation 15;
      Update $\sigma_i(t_{curr})$ according to Equation 16;
      $t_{estm} = t_{curr} + \min\{\frac{\sigma_i(t_{last})}{\sigma_i(t_{curr})} \times (t_{curr} - t_{last}), t_{max}\}$;
    else
      / stream* S_i *is interrupted */*
      $\mu_i(t_{curr}) = \mu_i(t_{last}),\ \sigma_i(t_{curr}) = \sigma_i(t_{last})$;
      $t_{estm} = t_{curr} + T$;
    end if
    $t_{last} = t_{curr}$;
  end if
end loop

stream S_i, the likelihood function of μ_i and σ_i is:

$$L(\mu_i, \sigma_i) = \prod_{j=1}^{k-1} f_{y_j}(y_j, \mu_i, \sigma_i),$$

$$f_{y_j}(y_j, \mu_i, \sigma_i) = \frac{1}{\sqrt{2\pi\sigma_i^2\delta_j}} exp(-\frac{(y_j - \mu_i\delta_j)^2}{2\sigma_i^2\delta_j}), \quad (14)$$

where $f_{y_j}(y_j, \mu_i, \sigma_i)$ is the density function for Normal sample y_j. The ML estimators for μ_i and σ_i can be derived by maximizing the likelihood function in Equation 14. After solving two partial derivative equations, we obtain the ML estimator for μ_i:

$$\hat{\mu_i} = \frac{\sum_{j=1}^{k-1} y_j}{\sum_{j=1}^{k-1} \delta_j} = \frac{val_{k-1} - val_0}{ts_{k-1} - ts_0}, \quad (15)$$

and the estimator [2] for σ_i^2:

$$\hat{\sigma_i^2} = \frac{1}{k-2} \sum_{j=1}^{k-1} \frac{(y_j - \delta_j\hat{\mu_i})^2}{\delta_j}. \quad (16)$$

Due to the lack of space, we omit the derivation of Equation 15 and 16 in this paper. $\hat{\mu_i}$ and $\hat{\sigma_i^2}$ are both unbiased estimators and easy to compute. If stream items are generated at constant rates at the sources, i.e., $\delta_1 = \delta_2 = \cdots = \delta_{k-1} = h$, Equation 15 and 16 can be simplified as:

$$\hat{\mu_i} = \frac{\bar{y}}{h}, \quad \hat{\sigma_i^2} = \frac{1}{(k-2)h} \sum_{j=1}^{k-1} (y_j - \bar{y})^2. \quad (17)$$

Since $y_1, \cdots, y_{k-1}$ become i.i.d samples from $N(\mu_i h, \sigma_i^2 h)$, when the generation rate is constant, the unbiased estimators for $\mu_i h$ and $\sigma_i^2 h$ are the sample mean $\bar{y}$ and sample variance $\frac{1}{k-2}\sum_{j=1}^{k-1}(y_j - \bar{y})^2$, respectively. Thus, by dividing both estimators by h, the resulting estimators for μ_i and σ_i^2 are consistent with what we derived in Equation 17.

2 The actual ML estimator for σ_i^2 is $\frac{1}{k-1}\sum_{j=1}^{k-1}\frac{(y_j - \delta_j\hat{\mu_i})^2}{\delta_j}$, which is not unbiased. $\hat{\sigma_i^2}$ obtained in Equation 16 is an unbiased estimator.

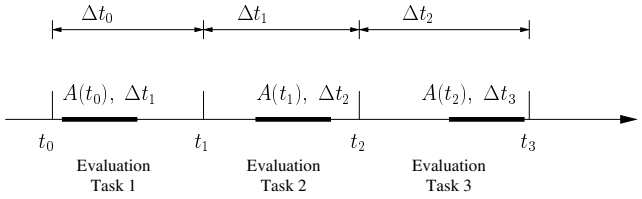


Figure 4: Query evaluation time constraints

The parameter estimation process for each stream is independent of each standing query execution. The query execution process simply grabs the most recent estimates of the parameters when it needs to calculate Δt. It is unnecessary to re-estimate these parameters as each data item arrives. If the current estimate of σ_i is low, indicating that the current variance of the increments is low, we may safely use the same μ_i and σ_i in the near term. We adopt a simple parameter estimation method based on this intuition, as shown in Algorithm 2. t_{last} denotes last parameter estimation time for S_i, and t_{max} denotes the maximum allowable estimation interval. When some streams suffers from temporary interruptions, the QP simply abandons the current estimation task, and uses the last estimates instead.

We also need to set the sample size k properly, since making it too large or too small will lead to bad estimated values. We experimentally adjust k to achieve the best performance.

5. Scheduling Query Evaluations

The central mission for the QP is to adaptively perform aggregate evaluations often enough that the user-specified error bound (ϵ) and the belief threshold (p) are met. As observed in Section 4.2, however, a very high value of σ or a huge number of standing queries may require aggregates to be evaluated very frequently, resulting processor overload. Under overload, some aggregates may fail to be evaluated by their evaluation times, leading errors to exceed the bounds. Thus, we need a mechanism to timely schedule all aggregate evaluations whenever possible, and minimize the overall errors incurred by missing evaluation times under processor overload.

We treat the problem as a real-time uniprocessor scheduling problem. Each real-time task consists of two parts: the aggregate evaluation itself, and the prediction of the next evaluation time. In Figure 4, $t_0, t_1, t_2, \cdots$ are the times the processor is to re-evaluate the aggregate. At time t_i, we must evaluate the aggregate $A(t_i)$ and calculate Δt_{i+1}, so that, by time t_{i+1}, we will know the next

Symbol	Company Name	β (Volatility)
NOVL	Novell	2.33
BRCM	Broadcom	3.91
SEBL	Siebel Systems	3.06
YHOO	Yahoo Inc	3.88
QCOM	Qualcomm	2.05
SUNW	Sun Micro	2.23
CSCO	Cisco	1.98
XLNX	Xilinx	2.07
GE	General Electric	1.1
PALM	Palm Computing	1.7

Table 1: Stock streams used in our simulation

evaluation time $t_{i+2} = t_{i+1} + \Delta t_{i+1}$. In standard scheduling terminology, t_{i+1} is the *deadline* for the evaluation task i, and *release time* for the task $i+1$. We assume our evaluation tasks are preemptible.

We adopt EDF (*earliest-deadline-first*) [9] algorithm to schedule evaluation tasks. EDF is optimal in underload condition [9]. The system becomes overloaded if some pending tasks cannot finish before their deadlines, i.e., tasks are late. Under overload, we aim to minimize the overall incurred errors. We define a penalty function which measures the extra error of a late task as follows:

$$P_i(C_i) = \begin{cases} 0 & C_i \leq d_i \\ |\mu_{Ai}|(C_i - d_i) + \sigma_{Ai}\sqrt{C_i - d_i} & C_i > d_i \end{cases} \quad (18)$$

where d_i and C_i are the deadline and the actual completion time for task i, respectively. In Equation 18, there is no penalty incurred if the task completes before the deadline, otherwise it incurs positive penalty according to the DBM model.

To minimize $\sum_i P_i$ under overload, we sequence all late tasks so that metric $\frac{(\mu_{Ai}+\sigma_{Ai}/\sqrt{s_i})}{p_i}$ forms a non-increasing order, where p_i is the processing time of task i, and s_i is the *slack* time at time t: $s_i = d_i - (t + p_i)$. This sequencing method guarantees the overall penalty is within a ratio bound of the minimum penalty, as proved in [27].

6. Experimental Evaluations

We conducted a series of experiments to demonstrate the applicability of our model and the performance of our approach to real-world streams such as real-time sensor data and stock price data. Our sensor time series are one-year's measurements (11/1991–11/1992) of *temperature (temp)* at various sea levels, taken from the *TAO* project [2] at the Pacific Marine Environmental Laboratory (PMEL). Each sensor stream contains about 10^4 values, sampled every minute. Our stock data are collected every minute for the entire year 06/2001–06/2002. Each stock stream corresponds to one stock symbol and contains about 10^5 values. The stock symbols we chose are listed in Table 1.

We deliberately chose streams with high volatility. Such streams show high uncertainty of movement, and display large fluctuations over short intervals. Highly volatile data are more challenging for our adaptive query evaluation model. For example, the β value is a measure of the relative volatility of a stock to the market (see Table 1). Generally, symbols with $\beta \in [1, 4]$ are considered to have high volatility. All stock traces we chose have β bigger than 1.

We simulated our system using the `csim` package [1] on an Intel Pentium 4 at 1.60GHz. We picked 10 stock streams and 10 *temp* sensor streams. Each stream was simulated as a `csim` *process*, which periodically generated data items read from our trace files. Each standing query was also a *process*, which was repeatedly invoked when its evaluation time was due, executed the query, and calculated the next evaluation time. The parameter estimation process for each stream updated μ_i and σ_i on a regular basis. The QP buffer size (k) of each stream was set to 100.

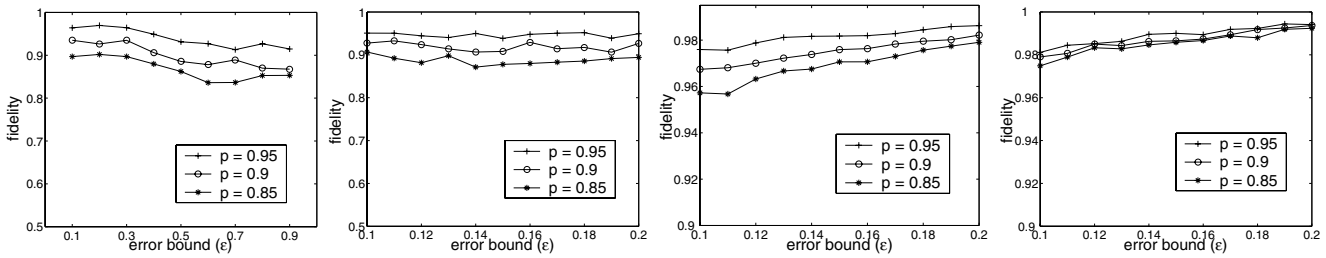
6.1. Applicability of Our Model

Our goal is to defer query evaluations as long as possible, but to execute them before the values have drifted so far that the error bound and belief threshold are not met. If our scheme schedules query evaluations at times $t_1, t_2, \ldots$, we may ask at each t_i whether it succeeds in meeting the error bound. We introduce the *fidelity* metric which measures how often our predictions of query evaluation times meet user-specified bounds.

$$fidelity(A) = \frac{\text{total time } A\text{'s cache-source errors} \leq \epsilon(\epsilon_r\%)}{\text{total simulation time}}$$

Figure 5 demonstrates the fidelity metric for our data sets, with various error bounds (ϵ) and belief thresholds p. For this experiment, we compute two types of aggregates: AVG over 9 streams, and sliding window $\max$ over AVG. Figures 5(a) and 5(b) show the AVG aggregate on stock data and *temp* data, respectively. Each point in the plot represents the average fidelity over 10 queries. We observe that higher p achieves higher fidelity, but needs more evaluations (see Figure 6). Figure 5(c) and 5(d) show the results of the sliding window $\max$. The window size is 30 mins.

Table 2 summarizes the average fidelity achieved for each p value. For *AVG*, the average fidelity closely matches to the corresponding p value for both datasets, showing that the drifting Brownian motion is the right model to capture the behavior of *AVG*. The sliding window $\max$ aggregate, however, achieves higher fidelity than the corresponding p for the following reason. In this case, Δt (see Equation 11) is determined by the two factors φ_1 and φ_2. While φ_2 is calculated to ensure that the $\max$ value is below $A_m + \epsilon$ with confidence p, φ_1 provides an absolute guarantee that the $\max$ is above $A_m - \epsilon$. The overall fidelity tends to be higher than p, due to the influence of φ_1.



(a) AVG for 9 stock streams (b) AVG for 9 *temp* streams (c) $\max\{AVG\}$ over sliding window (30 mins), stock streams (d) $\max\{AVG\}$ over sliding window (30 mins), *temp* streams

Figure 5: Fidelity

(agg type)/(data set)	$p = 95\%$	$p = 90\%$	$p = 85\%$
AVG/stock data	94.00%	89.92%	86.84%
AVG/*temp* data	94.66%	91.88%	88.74%
max/stock data	98.15%	97.47%	96.89%
max/*temp* data	98.90%	98.70%	98.53%

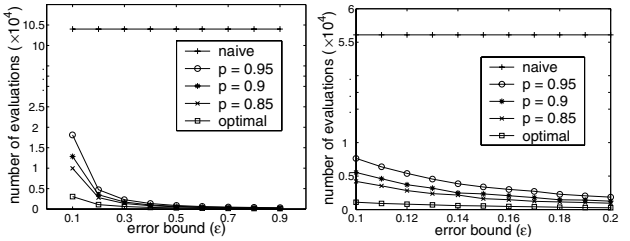
Table 2: Average fidelity for various p

Figure 6 compares the total number of *AVG* evaluations triggered by our scheme for different parameter values. The *naive* curve shows the number of evaluations that would have been performed were the aggregate to be naively reevaluated at each stream tick. The *optimal* curve, which is obtained from an off-line calculation, shows the minimum number of evaluations needed for a certain ϵ . Clearly, the savings of our scheme are significant, especially with bigger error bounds and lower belief thresholds.

6.2. Comparison of Scalability

Though our scheme clearly reduces the number of aggregate evaluations, it incurs the overhead of having to find the next evaluation time for each aggregate evaluation. Solving Δt (see Equation 6) for linear aggregates is equivalent to computing the standard erf function. For windowed $\min$/$\max$ aggregates (see Equation 11), it may also involve a scan of items within the window. We must ensure this cost is reasonable. Our experiments show that our scheme actually scales far better than the alternative scheme of evaluating all aggregates at each tick.

Figure 7 shows the percentage of aggregates that the system is unable to evaluate in time due to CPU saturation. Half the aggregates in the mix are linear aggregates, and half are sliding window $\max$ aggregates. The CPU is modeled as a *facility* at the speed of 1.6 GHz on a `csim` virtual machine. According to the real machine execution cost, AVG over 9 streams takes 0.1 milliseconds (ms), $\max$ over the window of 30 mins takes 0.53 ms, and erf takes 0.37 ms. To simplify our model, we assume all standing queries are in main memory with no disk access involved. With the naive scheme, the system starts missing aggregates when



(a) AVG for stock streams (b) AVG for *temp* streams

Figure 6: Number of evaluations

3,000 queries are in the system, and the performance worsens rapidly as the number of queries rises. In contrast, our adaptive scheme demonstrates much better scalability. Not surprisingly, the bigger the error bound, the higher scalability we can achieve, since fewer evaluations are triggered.

Figure 8 shows how well our scheme handles stream interruptions. We compute the AVG aggregate over 9 stock streams, and study two cases, when interruptions occur in two, and three streams, respectively. As expected, the error increases as the interruption duration increases. As more streams are interrupted, the error incurred in the aggregate results increases.

7. Conclusion

We have proposed a novel model for approximate aggregate evaluation over streaming data, based on the DBMs. Given a user-specified error bound, evaluating the aggregate every time new data stream in is unnecessary. We exploit users' error tolerance to enhance QP's scalability by deferring aggregate evaluation as long as the error is within user bound. During stream interruptions, our model provides bounds for the missing values based on estimates for the drift and diffusion parameters

We study the suitability of our Brownian motion model, both theoretically and experimentally, and show that for large classes of stream data, such as stock prices and sensor

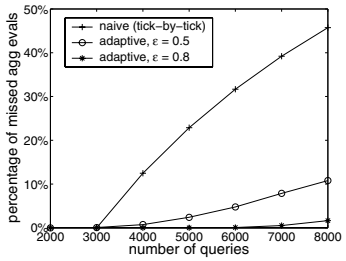


Figure 7: Percentage of missed aggregate evaluations due to CPU saturation (stock data, $p = 0.9$)

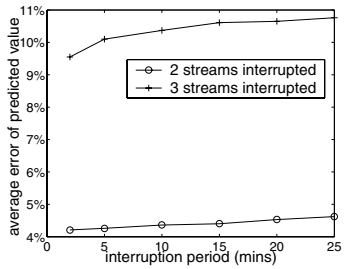


Figure 8: Prediction error($\epsilon = 0.5, p = 0.9$)

data, our model yields excellent performance. Our experiments also show that our methods are very scalable as the number of queries increases. We also study the issue of processor allocation in the context of our approximate query evaluation model.

How to extend our work to other types of queries, e.g., join, correlated aggregates, remains as a topic for our future work. we are also interested in other tradeoff problems in streaming applications. For example, how to trade query result precision for query response time.

References

[1] http://www.mesquite.com/htmls/guides.htm.

[2] http://www.pmel.noaa.gov/tao/index.shtml.

[3] http://www.traderbot.com.

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. 21th ACM SIGACT-SIGMOD-SIGART Symp. on Principle of Database Systems*, Madison, May 2002.

[5] B. Babcock and C. Olston. Distributed top-k monitoring. In *Proc. of the 2003 ACM SIGMOD Conf*, San Diego, 2003.

[6] D. Carney, U. Cetintemel, M. Cherniack, C. Convey, et al. Monitoring streams–a new class of data management applications. In *Proc. of the 28th VLDB Conf*, Hong Kong, 2002.

[7] D. Carney, U. Cetintemel, A. Rasin, S. Zdonik, et al. Operator scheduling in a data stream manager. In *Proc. of the 29th VLDB Conf*, Berlin, Germany, 2003.

[8] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. Niagracq: A scalable continuous query system for internet databases. In *Proc. of the 2000 ACM SIGMOD Conf*, Madison, May 2000.

[9] C.M.Krishna and K. G. Shin. *Real-Time Systems*. McGraw-Hill, 1997.

[10] A. Das, J. Gehrke, and M. Riedewald. Approximate join processing over data streams. In *Proc. of the 2003 ACM SIGMOD Conf*, San Diego, 2003.

[11] A. Dobra, M. Garofalakis, J. Gehrke, and R. Rastogi. Processing complex aggregate queries over data streams. In *Proc. of the 2002 ACM SIGMOD Conf*, Madison, 2002.

[12] M. Fisz. *Probability Theory and Mathematical Statistics, 3rd Edition*. John Wiley & Sons, Inc, 1963.

[13] J. Gehrke, F. Korn, and D. Srivastava. On computing correlated aggregates over continual data streams. In *Proc. of the 2001 ACM SIGMOD Conf*, Santa Barbara, 2001.

[14] G.Grimmett and D.Stirzaker. *Probability and Random Processes, 3rd Edition*. Oxford University Press, 2001.

[15] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proc. of the 27th VLDB*, Roma, Italy, 2001.

[16] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *Proc. of the 1997 SIGMOD Conf.*, Tucson, 1997.

[17] H.V.Jagadish, N. Koudas, S.Muthukrishnan, et al. Optimal histograms with quality guarantees. In *Proc. of the 24th VLDB Conf*, New York City, 1998.

[18] I. Lazaridis and S. Mehrotra. Capturing sensor-generated time series with quality guarantees. In *Proc. of the 19th ICDE Conf*, Bangalore, India, 2003.

[19] S. Madden and M. J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *Proc. of the 18th ICDE Conf*, San Jose, 2002.

[20] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *the 5th Annual Symposium on OSDI*, December 2002.

[21] A. M. Mood, F. A. Graybill, and D. C. Boes. *Introduction to the Theory of Statistics, 3rd Edition*. McGraw-Hill, 1974.

[22] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[23] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *Proc. of the 2003 ACM SIGMOD Conf*, San Diego, 2003.

[24] C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *Proc. of the 26th VLDB Conf*, Cairo, Egypt, 2000.

[25] N. Tatbul, U. Cetintemel, S. Zdonik, M. Cherniack, et al. Load shedding in a data stream manager. In *Proc. of the 29th VLDB Conf*, Berlin, Germany, 2003.

[26] H. C. Thode. *Testing for Normality*. Marcel Dekker, Inc., 2002.

[27] S. Zhu and C. Ravishankar. A scalable approach to approximating aggregate queries over intermittent streams. http://www.cs.ucr.edu/∼szhu/scal.pdf. Extended Version.

[28] S. Zhu and C. Ravishankar. Stochastic consistency, and scalable pull-based caching for erratic data sources. http://www.cs.ucr.edu/∼szhu/stochpull.pdf. Technical Report, Univ. of California, Riverside, 2003.

[29] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *Proc. of the 28th VLDB Conf*, Hong Kong, China, 2002.