RC24450 (W0712-005) December 3, 2007
Computer Science

IBM Research Report

LEEWAVE: Level-Wise Distribution of Wavelet Coefficients for Processing *k*NN Queries over Distributed Streams

Mi-Yen Yeh[1,2], Kun-Lung Wu[1], Philip S. Yu[1], Ming-Syan Chen[2]

[1]IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

[2]National Taiwan University
Taipei, Taiwan

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

LEEWAVE: Level-Wise Distribution of Wavelet Coefficients for Processing kNN Queries over Distributed Streams

Mi-Yen Yeh[♠,♢], Kun-Lung Wu[♠], Philip S. Yu[♠], and Ming-Syan Chen[♢]
[♠] IBM T. J. Watson Research Center, Hawthorne, NY
[♢] National Taiwan University, Taipei, Taiwan
miyen@arbor.ee.ntu.edu.tw, {klwu,psyu}@us.ibm.com, mschen@cc.ee.ntu.edu.tw

Abstract

We present LEEWAVE *– a bandwidth-efficient approach to searching range-specified k-nearest neighbors among distributed streams by level-wise distribution of wavelet coefficients. To find the top-k similar streams to a range-specified reference one, the relevant wavelet coefficients of the reference stream can be sent to the peer sites to compute the similarities. However, bandwidth can be unnecessarily wasted if the entire relevant coefficients are sent simultaneously. Instead, we present a level-wise approach by leveraging the multi-resolution property of the wavelet coefficients. Starting from the top and moving down one level at a time, the query initiator sends only the single-level coefficients to a progressively shrinking set of candidates. However, there is one difficult challenge in* LEEWAVE*: how does the query initiator prune the candidates without knowing all the relevant coefficients? To overcome this challenge, we derive and maintain a* similarity range *for each candidate and gradually tighten the bounds of this range as we move from one level to the next. The increasingly tightened similarity ranges enable the query initiator to effectively prune the candidates without causing any false dismissal. Extensive experiments with real and synthetic data show that, when compared with a naive one,* LEEWAVE *uses significantly less bandwidth under a wide range of conditions.*

1 Introduction

Processing data streams has become increasingly important as more and more emerging applications are required to handle a large amount of data in the form of rapidly arriving streams. Examples include data analysis in sensor networks, program trading in financial markets, video surveillance and weather forecasting. In response, many organizations [1, 3, 4, 8, 13, 28, 30] have started developing data stream processing systems (DSPS).

Finding k-nearest neighbors (kNN) is one of the most common applications in computing. Processing kNN queries has been one of the most studied problems in traditional database research. It is also believed to be the case in data stream processing [11, 16, 18, 20]. For a kNN query, the DSPS will find the top-k streams that have the most similar patterns to a given pattern contained in a reference stream. Compared to kNN query processing in traditional databases, stream-based kNN query processing is much more challenging. It must handle an endlessly growing amount of data with limited resources. Nevertheless, many researchers have started working on various aspects of stream-based kNN query processing [11, 16, 18, 20]. But, these works mainly focus on the case where data streams are collected and processed at a central site.

In many real-world applications, however, data streams are usually collected in a decentralized manner. For example, to forecast the weather and track global climate changes, meteorologists collect streams of measurements, like temperatures, from observation stations located over a wide area. In surveillance, video cameras are set up in many places and continuously capture images from various angles. Finally, readings from a sensor network are collected in a distributed fashion. In these cases, it is inefficient to gather all of the distributed streams to a central site before doing any query processing. It is even impossible to do so when the available network bandwidth is limited. Hence, there is a need to develop a bandwidth-efficient approach to processing kNN queries among distributed streams.

In this paper, we study the problem of processing distributed kNN (k-similarity) queries. We assume that the network is a mesh work; that is, all the sites can be connected to one another. Nevertheless, our proposed method can also be extended to other kinds of networks with appropriate modifications. The system model is shown in Fig 1, where there are M distributed sites, each monitoring one or more streams. For a given a reference stream S_{ref} maintained by an initiator site, P_{init}, the goal is to find the top-k

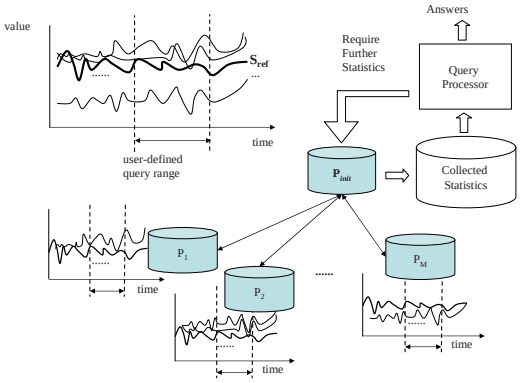

Figure 1. System model.

streams among all M sites with the highest similarities to S_{ref} in the user-defined time range T.

An obvious solution to distributed kNN query processing is to transmit all the relevant data of S_{ref} in the specified time range to all other sites. After receiving the data from P_{init}, each peer site computes the similarities between locally maintained streams and S_{ref}, and then reports its local kNN to P_{init}. Finally, P_{init} determines the true kNN after it receives results from all the peer sites. Unfortunately, this obvious solution is not a good one because it requires large bandwidth (total relevant data multiplied by $M-1$) and a significant part of it can be unnecessarily wasted, especially if $M \gg k$ or if T is large.

Searching for a better solution, we notice that summary sketches, instead of complete details, of the streams are usually maintained in a data streaming environment. Among various sketches, the wavelet-based one, especially the Haar wavelet summarization, has been widely adopted in many stream-oriented applications due to its efficiency and simplicity [5, 29, 32]. More importantly, the Haar wavelet decomposition provides multiple resolutions in time and frequency domains. In a coarser resolution, there are fewer wavelet coefficients, each representing a longer subsection of the original data; while in a finer resolution, there are more wavelet coefficients, but each represents a smaller subsection of the data. More specifically, coefficients in a lower resolution give a rough outline of the original data, while those in a higher resolution disclose more details. This multi-resolution property, which has also been explored for other applications [2, 19], gives us inspiration that we can use fewer coefficients in a coarser resolution to filter out many candidates and then use more coefficients in a finer resolution to refine the answers.

Armed with this insight, we present LEEWAVE − a bandwidth-efficient approach to processing kNN queries in a distributed streaming environment by level-wise distribution of wavelet coefficients. In essence, instead of simultaneously distributing all the relevant coefficients of the reference stream to other peer sites, the query initiator, P_{init}, sends the coefficients one level at a time, starting from the top (the coarsest) level. At each step, with returned level distances from the peer sites, P_{init} progressively prunes the candidates. As we progress to a lower level, which usually contains more coefficients, the number of candidates becomes much smaller. As a result, significant bandwidth savings can be realized because the wavelet coefficients at a lower level are sent to a much smaller set of candidate sites. More importantly, during the process of candidate pruning, we guarantee that there is no false dismissal.[1]

However, there is one difficult challenge in LEEWAVE: At each step, how does P_{init} prune the candidate streams without knowing all the relevant wavelet coefficients? In fact, it is impossible at an intermediate level to compute the true similarities. Without them, it is difficult, if not impossible, to prune the candidates. To address this challenge, we derive and maintain a *similarity range* for each candidate stream. The upper and lower bounds of a similarity range can be incrementally updated at P_{init} with level-wise distances returned from the peer sites. More importantly, a similarity range gradually becomes tighter as we move from one level to the next. These increasingly tightened similarity ranges enable P_{init} to effectively prune the candidate streams without causing any false dismissal.

To evaluate the effectiveness of LEEWAVE, we conduct extensive experiments using both real and synthetic data. For comparisons, we also implement a naive approach, which sends the entire relevant coefficients to all $M-1$ peer sites in one step. We measure the total bandwidth consumed in finding the top-k most similar streams to a reference one. The results show that, under a wide range of conditions, LEEWAVE consumes significantly less bandwidth, especially when $M \gg k$ or T is large.

Our contributions can be summarized as follows:

- We introduce LEEWAVE as a bandwidth-efficient approach to processing kNN queries in a distributed streaming environment by level-wise distribution of wavelet coefficients. The relevant coefficients of the reference stream are sent to a progressively shrinking set of candidate streams/sites one level at a time.

- We derive and maintain a similarity range for each candidate stream and gradually tighten the range by incorporating level-wise distances computed and returned by a peer site. These increasingly tightened similarity ranges enable the query initiator to prune the candidate streams without any false dismissal.

[1] Note that not all the wavelet coefficients are retained in a steaming environment. As a result, the accuracy of our scheme is based only on the retained wavelet coefficients. Namely, if there is an error in the kNN w.r.t. the raw data, the error is solely due to the discarded coefficients. Our scheme does not introduce any extra error.

- We conduct extensive experimental studies to evaluate LEEWAVE. The results show that, when compared with a naive approach, LEEWAVE uses significantly less bandwidth under a wide range of conditions.

The remainder of this paper is organized as follows. Related work is discussed in Section 2. Preliminaries are given in Section 3, including wavelet decomposition and coefficient maintenance. The application of LEEWAVE to the processing of distributed kNN queries is described in Section 4. Section 5 shows the experimental results. Finally, the paper is concluded in Section 6.

2 Related Work

Wavelet transform plays an important role in the field of time series analysis [25]. Wavelet-based data summarization has been used for many data stream applications. Bulut et al. provided a wavelet-based index structure which incrementally summarizes data in multiple resolutions [5]. These indexes can then be used to answer point queries, range queries and inner product queries. Zhu et al. considered burst detection using a summary structure called SWT, which is a shifted-wavelet tree based on the Haar wavelet transform [32]. Teng et al. applied the Haar wavelet transform concept to discover frequent temporal patterns of data streams [29].

Searching k-nearest neighbors is an important research topic in a streaming environment. Many related works have discussed solutions in either a single-stream [11, 18] or a multiple-stream environment [20, 16]. In [11], the authors proposed to continuously retrieve the latest L points of a stream as a query pattern and then find its nearest neighbors from a time series database. Base on traditional indexing methods, the proposed scheme achieves efficient query response via prefetching. In [18], given an error bound, approximate k-nearest neighbors are searched among stream snapshots. In [20], the authors proposed a new indexing technique based on scalar quantization to provide efficient nearest-neighbor search among multiple streams. In [16], based on the Haar wavelet synopses, the authors provided an efficient approach to finding the k-nearest neighbors under an arbitrary range constraint. All these works assume that streams are collected and processed at a central site.

However, in practice, most streams are generated in geographically distributed places. Therefore, more and more research works have started to focus on distributed streams. These works include finding recently frequent itemsets [22], tracking approximate quantiles [9], processing aggregation and thresholding queries [21, 24, 26], indexing for inner product queries and similarity queries [6], and so forth. However, to the best of our knowledge, there is no prior work on processing kNN queries in a distributed streaming environment using wavelet coefficients.

Table 1. Haar wavelet decomposition.

	averages	wavelet coefficients
raw data	$\{4, 6, 7, 4, 8, 6, 5, 7\}$	-
high resolution	$\{5, 5.5, 7, 6\}$	$\{-1, 1.5, 1, -1\}$
mid resolution	$\{5.25, 6.5\}$	$\{-0.25, 0.5\}$
low resolution	$\{5.875\}$	$\{-0.625\}$

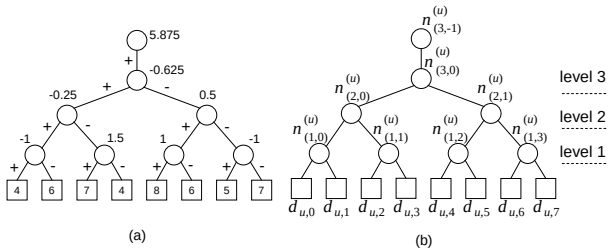

Figure 2. (a) The error tree for Example 1; (b) The notation of an error tree proposed in [16].

3 Preliminaries

3.1 Wavelet decomposition

Among different wavelet transformations, the Haar wavelet decomposition is the first and also the most popular one. It is achieved by averaging two adjacent data values of a sequence of data at different time resolutions. Then, only the overall average and differences are kept. We show a simple example here in Table 1 to illustrate the idea of the Haar wavelet decomposition. The original data are $\{4, 6, 7, 4, 8, 6, 5, 7\}$ and the final coefficients are $\{5.875, -0.625, -0.25, 0.5, -1, 1.5, 1, -1\}$. More details can be found in [15].

To better illustrate the Haar wavelet decomposition, a widely used data structure called *error tree* is proposed in [23]. The error tree for Example 1 is shown in Fig. 2(a). This tree is composed of wavelet coefficients as nodes and signs as edges. The root of this tree is the overall average and all the other non-leaf nodes are differences at various resolutions. The leaf nodes represent the raw data, but these raw data are not maintained. Instead, one can always reconstruct the raw data by tracing the error tree. Along each path from the root, each data value in a leaf node is equal to the sum of the value of a node multiplied by the sign below it. For example, the fifth data value, 8, can be reconstructed by $+5.875 - (-0.625) + 0.5 + 1 = 8$.

We also use an error tree to illustrate our idea, the same as the one used in [16], which is shown in Fig. 2(b). Here, each non-leaf node is labeled with an identifier with two attributes as subscripts: *level* and *placement*. A node with a

label of $n^{(u)}_{(l,p)}$ means that it is in the p-th placement of level l in the error tree corresponding to stream S_u. This notation can be efficiently maintained when data keep steaming in. Moreover, when not all wavelet coefficients are retained, we can easily find the relative positions of the retained coefficients in the error tree via the node labels.

3.2 Coefficient maintenance

Generally, not all the wavelet coefficients in an error tree are retained because the data volume tends to be huge and the memory space is limited in a streaming environment. To meet different error requirements between the raw data and the retained coefficients, many on-line approaches to selecting wavelet synopses have been proposed. These requirements include minimizing the L-norm average error [12], minimizing the maximum absolute/relative error [17], minimizing the weighted L^p-norm error [14], and providing a guaranteed accuracy [10], to name a few. In this paper, we assume that the wavelet synopses are dynamically maintained based on the one proposed in [12].

Given the retained coefficients of a stream, we can extract the relevant coefficients within any time range $[t_s, t_e]$. In this paper, we adopt an extraction procedure proposed in [16], which has $O(\log^2 N)$ complexity, where N is the total number of data values in a stream. Here we present a high-level outline of the procedure. More details can be found in [16]. When a time range $[t_s, t_e]$ is given, it first decomposes the range into several subranges where each of them corresponds to a complete error subtree. Using Fig. 3 as an example, suppose the given range is $[t_0, t_{11}]$, which contains the shaded triangular area in Fig. 3. We can decompose it into two complete error subtrees where one covers $[t_0, t_7]$ and the other $[t_8, t_{11}]$. For each complete error subtree, the new average node will be computed by traversing from the original root node $n^{(u)}_{(4,-1)}$ to the root of the subtree (see Fig. 3). The black nodes represent retained coefficients, while the white ones are those being discarded. When traversing the path to get the new average, the missing nodes are just treated as zero.

4 The LEEWAVE approach to processing distributed kNN queries

The central idea of LEEWAVE is as follows. The query initiator sends the coefficients of S_{ref} one level at a time. At each step, each peer site reports the level-l distance and other necessary information to the initiator site. The initiator then gradually prunes the candidates with the return level distances. However, the key challenge is how the initiator prunes the candidates.

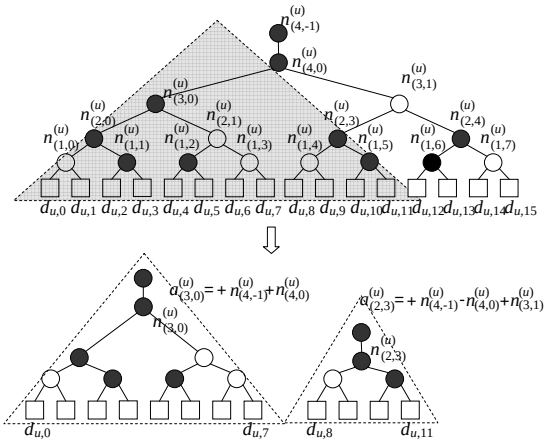

Figure 3. Extracting complete error subtrees and relevant coefficients from the whole error tree based on a desired time range.

4.1 Computing similarities using wavelet coefficients

Based on the maintained wavelet synopses, the goal of our kNN query is to find the k most similar streams to S_{ref}. We denote a kNN query as $Q(S_{ref}, k, t_s, t_e)$, where k is the desired number of top most similar streams, and $[t_s, t_e]$ defines the time range of interest. For the similarity measure between two streams, we adopt the commonly used Euclidean distance in this paper.

Given wavelet coefficients of two streams, we can compute the Euclidean distance directly from the coefficients themselves without doing inverse wavelet transform back to the original data [7]. By reformulating the distance computation proposed in [16], for a given time range T=$[t_s, t_e]$, the distance between two streams S_u and S_v can be computed as follows:

$$dst(S_u, S_v)|^{t_e}_{t_s} = [\sum_{T_i} \sum_{n^{(u)}_{(l,p)} \text{ or } n^{(v)}_{(l,p)} \in C_{T_i}} D^{(u,v)}_{(l,p)} \times 2^l]^{1/2}, \quad (1)$$

where

$$D^{(u,v)}_{(l,p)} = \begin{cases} [n^{(u)}_{(l,p)} - n^{(v)}_{(l,p)}]^2 & \text{if } n^{(u)}_{(l,p)} \in C_{T_i} \,\&\, n^{(v)}_{(l,p)} \in C_{T_i}, \\ [n^{(u)}_{(l,p)}]^2 & \text{if } n^{(u)}_{(l,p)} \in C_{T_i} \,\&\, n^{(v)}_{(l,p)} \notin C_{T_i}, \\ [n^{(v)}_{(l,p)}]^2 & \text{if } n^{(u)}_{(l,p)} \notin C_{T_i} \,\&\, n^{(v)}_{(l,p)} \in C_{T_i}, \end{cases}$$

T_i is one of the subrange in T with a complete error subtree, and C_{T_i} is the set of retained coefficients which are in the subrange T_i.

Eq. (1) computes the distance between two streams based on the complete error subtrees. Unfortunately, this

error-subtree-based distance computation is not useful in LEEWAVE because peer sites only receive the retained coefficients of S_{ref} one level at a time. We need a level-wise approach to computing the distance for LEEWAVE.

To better illustrate our idea, we introduce three definitions.

Definition 1 We define the square sum of Euclidean distance between stream S_u and S_v as $Dst(S_u, S_v)$. Namely,

$$Dst(S_u, S_v) = dst(S_u, S_v)^2.$$

Also, for ease of exposition, when we use the word "distance" from now on, we mean this square sum of Euclidean distance, $Dst(u, v)$.

Definition 2 We define the *level-l distance* between streams S_u and S_v as the distance of retained coefficients at level l.

$$Dst^l(S_u, S_v)|_{t_s}^{t_e} = \sum_p D_{(l,p)}^{(u,v)} \times 2^l,$$

where $D_{(l,p)}^{(u,v)}$ satisfies the same condition as in Eq.(1).

Definition 3 We define the accumulated distance between streams S_u and S_v from the highest level L down to level ρ (the top-down accumulated distance) as:

$$accDst^\rho(S_u, S_v)|_{t_s}^{t_e} = \sum_{l=\rho}^{L} Dst^l(u, v)|_{t_s}^{t_e}.$$

If we combine Definitions 2 and 3, and assume that the height of the whole retained error tree is L, then Eq.(1) can be reformulated as:

$$\begin{aligned} Dst(S_u, S_v)|_{t_s}^{t_e} &= dst(S_u, S_v)|_{t_s}^{t_e\,2} \\ &= \sum_p D_{(1,p)}^{(u,v)} \times 2^1 + ... + \sum_p D_{(L,p)}^{(u,v)} \times 2^L \\ &= \sum_{l=1}^{L} \sum_p D_{(1,p)}^{(u,v)} \times 2^l \\ &= \sum_{l=1}^{L} Dst^l(u, v)|_{t_s}^{t_e} \\ &= accDst^1(u, v)|_{t_s}^{t_e} \end{aligned} \quad (2)$$

Eq. (2) is important to LEEWAVE as it suggests a level-wise approach to computing the distance between two streams. LEEWAVE will use Eq. (2) to derive a similarity range for each candidate stream and gradually tighten this range in order to prune the candidates. Before we delve into the details, let us look at an example to understand the subtle, yet crucial, difference between Eq. (1) and (2). Note that, for ease of exposition, we will omit the notation of $[t_s, t_e]$ from now on in our description of the distance between two streams.

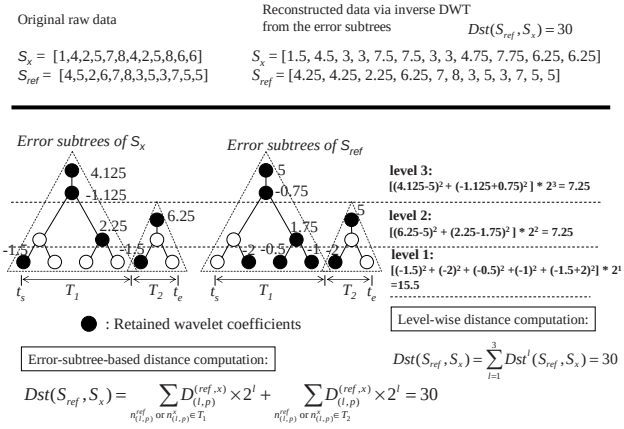

Figure 4. Example of level-wise vs. error-subtree-based distance computation.

Fig. 4 shows two different approaches to computing the distance between two streams, S_{ref} and S_x. Assume that during the time range $[t_s,t_e]$, S_x has 12 raw data values $[1, 4, 2, 5, 7, 8, 4, 2, 5, 8, 6, 6]$ and S_{ref} also has 12 raw data values $[4, 5, 2, 6, 7, 8, 3, 5, 3, 7, 5, 5]$. These raw data are transformed into separate error trees. Then a series of complete error subtrees and retained coefficients are extracted. To verify the correctness, we also show the reconstructed data and use them to compute the distance. However, in practice, we never need to do such reconstruction. The distance computation between two streams is done completely with retained wavelet coefficients.

Example 1 *Error-subtree-based computation:* Based on Eq.(1), the distance between two streams is the summation of the distance between each pair of their corresponding complete error subtrees. For the first pair of error subtrees in range T_1, the distance is: $((4.125 - 5)^2 + (-1.125 + 0.75)^2) \times 2^3 + ((2.25 - 1.75)^2+) \times 2^2 + ((-1.5)^2 + (-2)^2 + (0.5)^2 + (-1)^2) \times 2^1 = 23.25$. For the second pair of error subtrees in range T_2, the distance is: $(6.25 - 5)^2 \times 2^2 + (-1.5 + 2)^2 \times 2^1 = 6.75$. The final distance $Dst(S_{ref}, S_x)$ is $23.25 + 6.75 = 30$. If we check the distance from the reconstruction data by inverse DWT, we get exactly the same distance value. ■

Example 2 *Level-wise computation:* From Fig. 4, we can also compute the total distance in a level-wise manner. The total distance computed this way is also 30, which is the same as the computed distance in Example 1. ■

4.2 LEEWAVE for a kNN query

Note that, even with Eq. (2), the query initiator still cannot prune the candidate streams without risking some false dismissal. This is because, at a given level, it does not know how much those not-yet-seen coefficients of the candidate streams at lower levels will contribute to the final distances.

To overcome this problem, we maintain a similarity range into which the exact distance may fall for each candidate stream. To estimate the similarity range, we first decompose Eq.(2) into two parts: one is the accumulated distance so far and the other is the distance from those not-yet-seen coefficients.

$$\begin{aligned} & Dst(S_{ref}, S_x) \\ = & \sum_{l=1}^{L} Dst^l(S_{ref}, S_x) \\ = & accDst^{\rho}(S_{ref}, S_x) + \sum_{l=1}^{\rho-1} Dst^l(S_{ref}, S_x). \end{aligned} \quad (3)$$

At level ρ, the $accDst^{\rho}(S_{ref}, S_x)$ can be easily maintained by P_{init}. However, the second term of Eq. (3) is still unknown. Let us further decompose the second terms:

$$\begin{aligned} & \sum_{l=1}^{\rho-1} Dst^l(S_{ref}, S_x) \\ = & \sum_{l=1}^{\rho-1} \sum_{p} [n_{(l,p)}^{(ref)} - n_{(l,p)}^{(x)}]^2 \times 2^l \\ = & \sum_{l=1}^{\rho-1} \sum_{p} ([n_{(l,p)}^{(ref)}]^2 + [n_{(l,p)}^{(x)}]^2 - 2n_{(l,p)}^{(ref)} n_{(l,p)}^{(x)}) \times 2^l. \end{aligned} \quad (4)$$

In Eq. (4), P_{init} can easily compute the first term using its own coefficients. It can also compute the second term, $\sum_{l=1}^{\rho-1} \sum_p ([n_{(l,p)}^{(x)}]^2 \times 2^l)$, by first receiving $\sum_{l=1}^{L-1} \sum_p ([n_{(l,p)}^{(x)}]^2 \times 2^l)$ at the initial step and then gradually subtracting $\sum_p [n_{(\rho,p)}^{(x)}]^2 \times 2^{\rho}$ from it at each subsequent level ρ. The only problem left is the last term in Eq. (4), which involves the product of coefficients at levels below ρ. Neither P_{init} nor any candidate site can compute this term at the current level ρ. To guarantee no false dismissal, we would like to find a substitute for this term which is an overestimate but can be computed with level-wise coefficients.

Fortunately, we do find such a substitute. According to Cauchy-Schwarz inequality [27], we can find an upper bound of the inner product of two vector in real space, where this upper bound is the product of the linear square sum of each vector:

$$(\sum_{i=1}^{h} \alpha_i \beta_i)^2 \leq \sum_{i=1}^{h} \alpha_i^2 \times \sum_{i=1}^{h} \beta_i^2, \text{ where } \alpha_i, \beta_i \in \mathbb{R}.$$

Now we if let $\alpha_i = -[n_{(l,p)}^{(ref)}] \times 2^l$, and $\beta_i = [n_{(l,p)}^{(x)}]$, then the last term of Eq. (4) has an upper bound as follows:

$$\begin{aligned} & 2 \times \sqrt{\sum_{l=1}^{\rho-1} \sum_{p} (-[n_{(l,p)}^{(ref)}] \times 2^l)^2 \times \sum_{l=1}^{\rho-1} \sum_{p} [n_{(l,p)}^{(x)}]^2} \\ = & \quad 2 \times \sqrt{\sum_{l=1}^{\rho-1} \sum_{p} ([n_{(l,p)}^{(ref)}] \times 2^l)^2 \times \sum_{l=1}^{\rho-1} \sum_{p} [n_{(l,p)}^{(x)}]^2}. \end{aligned}$$

Therefore, at level ρ, the true distance between S_{ref} and S_x, which is described in Eq.(3), is bounded in the following range:

$$\begin{aligned} & accDst^{\rho}(S_{ref}, S_x) \leq Dst(S_{ref}, S_x) \leq \\ & accDst^{\rho}(S_{ref}, S_x) + \sum_{l=1}^{\rho-1} \sum_{p} ([n_{(l,p)}^{(ref)}]^2 + [n_{(l,p)}^{(x)}]^2) \times 2^l \\ & +2 \times \sqrt{\sum_{l=1}^{\rho-1} \sum_{p} ([n_{(l,p)}^{(ref)}] \times 2^l)^2 \times \sum_{l=1}^{\rho-1} \sum_{p} [n_{(l,p)}^{(x)}]^2}. \end{aligned} \quad (5)$$

With Eq. (5), it is now possible to maintain both the lower bound and the upper bound of this similarity range in a level-wise manner with $Dst^{\rho}(S_{ref}, S_x)$ and $\sum_p [n_{(\rho,p)}^{(x)}]^2$ returned by a peer site at each level ρ. This is because the term $\sum_{l=1}^{\rho-1} \sum_p [n_{(l,p)}^{(x)}]^2$ can now be incrementally computed at P_{init} by subtracting $\sum_p [n_{(l,p)}^{(x)}]^2$ from $\sum_{l=1}^{\rho} \sum_p [n_{(l,p)}^{(x)}]^2$, which was computed at the previous level $\rho + 1$. More importantly, as we move from one level to the next, the similarity range becomes tighter.

Theorem 1 *The upper bound of a similarity range is non-increasing when we move from level ρ to level $\rho - 1$.*

Proof: See the Appendix. ■

We are now ready to describe the details of how LEEWAVE processes a distributed kNN query in a level-wise manner and how P_{init} gradually prunes the candidates. Fig. 5 shows the algorithm of distributed kNN query processing using LEEWAVE. At the first step, P_{init} extracts the relevant wavelet coefficients of S_{ref} in the range of $[t_s, t_e]$. Then, it sends t_s, t_e and the level-L coefficients of S_{ref} to all other $M - 1$ peer sites. Each peer site then extracts relevant coefficients for each stream it monitors. And it returns 3 numbers for each local candidate stream, S_x. They are the level-L distance, $Dst^L(S_{ref}, S_x)$, and two other numbers that will be used by P_{init} to progressively tighten the similarity range:

$$\sum_{l=1}^{L-1} \sum_{p} [n_{(l,p)}^{(x)}]^2 \quad \text{and} \quad \sum_{l=1}^{L-1} \sum_{p} ([n_{(l,p)}^{(x)}]^2 \times 2^l).$$

Procedure: LEEWAVE for a kNN query
Input: P_{init}, S_{ref}, k, $T = [t_s, t_e]$
Output: The k most similar streams to S_{ref}

P_{init}:	*A candidate peer site P_x:*
1. Extract relevant coefficients of S_{ref} in [t_s,t_e].	
2. Send t_s, t_e, and coefficients of S_{ref} at level L to all other $M-1$ sites.	3. For each local candidate stream, S_x, compute and return a 3-tuple $(Dst^L(S_{ref}, S_x), \sum_{l=1}^{L-1}\sum_p [n_{(l,p)}^{(x)}]^2, \sum_{l=1}^{L-1}\sum_p ([n_{(l,p)}^{(x)}]^2 \times 2^l))$ to P_{init}.
4. Compute the upper and lower bound of the similarity range based on Eq. (5) for each candidate stream. Do the first pruning. Then, sort out a list of candidate sites and streams.	
5. **for** ($\rho = L-1$; (ρ! $= 0$ && !**done**); $\rho = \rho - 1$) {	
6. Send level-ρ coefficients of S_{ref} and the list of candidate streams to each candidate peer site.	7. Compute and return a 2-tuple ($Dst^\rho(S_{ref}, S_x)$, $\sum_p [n_{(\rho,p)}^{(x)}]^2$) for each local candidate stream, S_x.
8. Update the upper and lower bound of the similarity range based on Eq. (5) for each candidate stream. Do pruning. Set **done** to **true** if there are no more than k candidate streams left.	
9. }	

Figure 5. Algorithm for distributed kNN query processing using LEEWAVE.

After receiving the 3 numbers from a peer site, P_{init} updates the lower and upper bounds of a similarity range based on Eq. (5) for each candidate stream. It will then do some initial pruning, if possible, and sort out a list of candidate streams that might be the final top k similar streams. Then, it moves to the next level. For a given level ρ, P_{init} sends the level-ρ coefficients of S_{ref} and the list of candidate streams to a candidate site. A candidate site will compute and return two level-specific numbers: $Dst^\rho(S_{ref}, S_x)$ and $\sum_p [n_{(\rho,p)}^{(x)}]^2$. P_{init} will update the two bounds of a similarity range, making it tighter, with these two level-specific numbers. With increasingly tighter ranges, P_{init} can better prune the candidate list. The algorithm ends when there are no more than k candidate streams left.

To prune, P_{init} first sorts the candidate streams in an ascending order based on the upper bounds of their similarity ranges. Any candidate stream whose similarity lower bound is higher than the upper bound of the k-th streams in the sorted list cannot be the final answer, and thus is pruned. From Theorem 1, we can guarantee that there is no false dismissal under this pruning strategy.

Example 3 In this example, we use a concrete example to demonstrate the increasingly tightened similarity range for a candidate stream. Consider the case in Fig. 4. Table 2 shows the flow of data exchanges between P_{init} and P_x, the values of various terms in Eq. (5) maintained by P_{init}, and the similarity range of S_x. Each column shows the level currently in progress.

When $\rho = 3$, $Dst^3(S_{ref}, S_x) = 7.25$ is obtained as shown in Fig. 4, $\sum_{l=1}^{2}\sum_p [n_{(l,p)}^{(x)}]^2 = ((-1.5)^2 + (-1.5)^2 + 2.25^2 + 6.25^2) = 48.625$, and $\sum_{l=1}^{2}\sum_p [n_{(l,p)}^{(x)}]^2 \times 2^l = (((-1.5)^2 + (-1.5)^2) \times 2^1 + (2.25^2 + 6.25^2) \times 2^2) = 185.5$. The terms $\sum_{l=1}^{\rho-1}\sum_p [n_{(l,p)}^{(ref)} \times 2^l]^2$ and $\sum_{l=1}^{\rho-1}\sum_p [n_{(l,p)}^{(ref)}]^2 \times 2^l$ can be computed by P_{init} similarly. With above values, the first similarity range for stream S_x is $[7.25, 7.25 + 130.75 + 185.5 + 2 * (486 \times 48.625)^{1/2} = 630.95]$. This completes the values in the first column.

When ρ=2, P_x sends back $Dst^2(S_{ref}, S_x) = 7.25$, and $\sum_p [n_{(2,p)}^{(x)}]^2 = (2.25^2 + 6.25^2) = 44.125$. After receiving this 2-tuple, P_{init} then subtracts these two numbers from the maintained terms in the middle rows. Take ρ=2 as an example, $\sum_{l=1}^{2-1}\sum_p [n_{(l,p)}^{(x)}]^2 = 48.625 - 44.125 = 4.5$, and $\sum_{l=1}^{2-1}\sum_p [n_{(l,p)}^{(x)}]^2 \times 2^l = 185.5 - 44.125 \times 2^2 = 9$. For the sum related to coefficients of S_{ref}, P_{init} updates them similarly. Then, the second similarity range for S_x is $[7.25 + 7.25 = 14.5, 14.5 + 18.5 + 9 + 2 * (37 \times 4.5)^{1/2} = 67.81]$.

The last row in Table 2 shows the similarity range of S_x at each level. It clearly shows that the similarity range does indeed become tighter, $[7.25, 630.95]$ to $[14.5, 67.81]$ to $[30, 30]$, as we move from one level to the next. ■

5 Performance study

We conducted a series of experiments with both real and synthetic data to evaluate LEEWAVE. We compared LEEWAVE with a naive approach, which sends the entire coefficients to $M-1$ peer sites in a single step. Both approaches were implemented in Visual C++ and the experiments were run on a PC with 2.8GHz CPU and 2GB RAM.

We compared the total bandwidth requirements for LEEWAVE and the naive approach. We focused on the impacts of query range T, k and the total number of sites M on the bandwidth consumption. The total bandwidth consumption was calculated by adding up the data transmitted from P_{init} to all other candidate peer sites and those transmitted back

Table 2. Example of an increasingly tightened similarity range for a candidate stream.

	$\rho = 3$	$\rho = 2$	$\rho = 1$
P_{init} to P_x	$((5, -0.75), t_s, t_e)$	$(1.75, 5)$	$(-2, -0.5, -1, -2)$
P_x to P_{init}	$(7.25, 48.625, 185.5)$	$(7.25, 44.125)$	$(15.5, 4.5)$
$\sum_{l=1}^{\rho-1} \sum_p [n_{(l,p)}^{(ref)} \times 2^l]^2$	486.0	37.0	0
$\sum_{l=1}^{\rho-1} \sum_p [n_{(l,p)}^{(ref)}]^2 \times 2^l$	130.75	18.5	0
$\sum_{l=1}^{\rho-1} \sum_p [n_{(l,p)}^{(x)}]^2$	48.625	4.5	0
$\sum_{l=1}^{\rho-1} \sum_p [n_{(l,p)}^{(x)}]^2 \times 2^l$	185.5	9	0
similarity range for S_x	$[7.25, 630.95]$	$[14.5, 67.81]$	$[30, 30]$

from each peer candidate site. Each data value sent was counted as one unit of bandwidth. For coefficients in the same level, the total bandwidth was 2 (value and placement index) multiplied by the total number of coefficients at that level plus 1 (level number).

For the naive approach, the total retained coefficients plus 3 additional values were sent to $M - 1$ peer sites. The number of data values transmitted from the $M-1$ peer sites to P_{init} equals to the summation of the number of candidate streams (at most k) multiplied by 2, where 2 includes the stream index and the corresponding distances.

For LEEWAVE, we summed up the data transmitted at each level. For the highest level, only the level-L coefficients plus (t_s, t_e) were counted from P_{init} to the $M - 1$ peer sites. The number of data values sent back from each of the $M - 1$ peer sites for each locally maintained stream was 4, including the stream index, the level-L distance and two other values for pruning. For a subsequent level ρ, P_{init} sends the level-ρ coefficients and the candidate stream list for a candidate site. Then, for each local candidate stream, a candidate site only sends back the level-ρ distance and another data value needed for P_{init} to do the pruning.

The wavelet coefficients were retained using the method proposed in [12], which retains the B largest coefficients in terms of absolute normalized values. We randomly picked one stream from our dataset as the reference stream and performed kNN queries using both approaches. Since the total bandwidth used for processing kNN queries depends on the reference stream, we averaged the bandwidth consumption over a few different reference streams for each bandwidth value we reported.

5.1 Experiments with real data

The real data we used here were the daily average temperature data of 300 cities around the world, which were obtained from the Temperature Data Archive of the University of Dayton[2]. The data from each city was regarded as a stream. Each stream has $3,416$ data points. Unless otherwise specified, the default number of sites M was 150 and k was 10 for the experiments with real data. Streams were evenly distributed among the M sites for all the experiments.

The first experiment examined the impacts of query range T and k on bandwidth consumption for a given $M = 150$. The results are shown in Fig. 6. In this experiment, k varied from 5 to 25 and the query range was varied from the following set of values: 365, 730, $1,024$, $1,200$, $1,600$ and $2,048$. From Fig. 6(a), the bandwidth consumption of the naive approach increases significantly as the query range increases, because more coefficients need to be sent to the peer sites. In contrast, LEEWAVE continues to maintain a substantially smaller bandwidth requirement, even as the query range increases. Specifically, for $k = 5$ and $T = 2,048$, the bandwidth requirement of LEEWAVE is only about 16.67% that of the naive approach. Considering the impact of k, from 6(a), the bandwidth consumption of the naive approach is not sensitive to k, because it always sends the entire coefficients. On the other hand, the bandwidth requirement of LEEWAVE increases slightly as k increases, as shown in Fig. 6(b). This is because it uses the k^{th} lowest upper bound to do pruning. When k is larger, the upper bound is higher, which means the pruning ability becomes less effective.

In addition, from Fig. 6(b) we observe that the 3D surface is not smooth for LEEWAVE, especially along the query-range axis. The reason is as follows. For a different query range, we extracted different series of complete error subtrees, with different heights and subranges. Hence, the relevant retained coefficients might be rather different for different query ranges. Since LEEWAVE computes the distance in a top-down, level-wise fashion, the retained coefficients at different levels under different query ranges have different influences on the pruning effectiveness. To see the details of such impacts, we collected the average number of candidate sites at each step during the query processing in LEEWAVE. In Fig. 7, we plotted the number of candidate sites at each step (level) when k=5 in Fig. 7(a) and k=25 in Fig. 7(b).

First we look at the case when $k = 5$. From Fig. 6(b), the bandwidth consumption is higher for $T = 1,024$ (the

[2]http://www.engr.udayton.edu/weather/

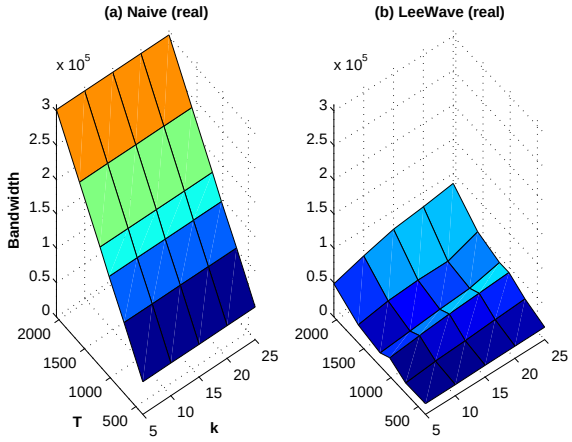

Figure 6. Impacts of T and k on bandwidth consumption with real data.

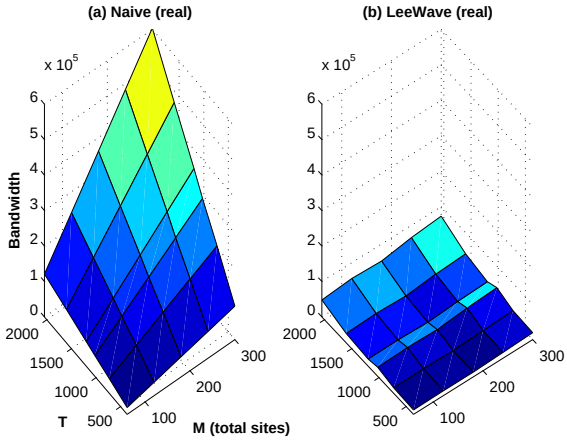

Figure 8. Impacts of T and M on bandwidth consumption with real data.

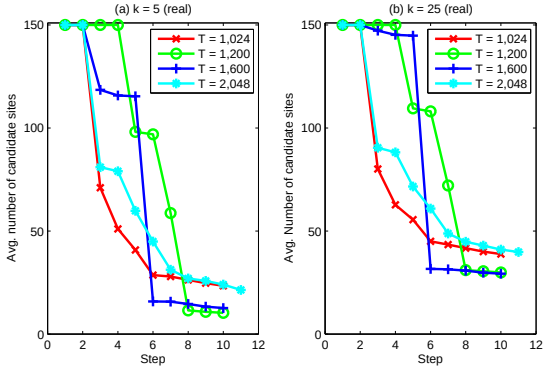

Figure 7. Size of candidate sites at each step of LEEWAVE in Fig. 6.

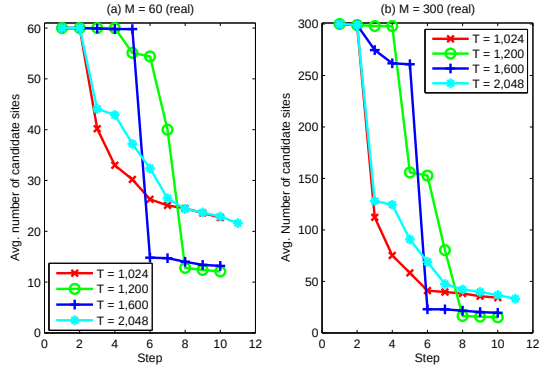

Figure 9. Size of candidate sites at each step of LEEWAVE in Fig. 8.

3^{rd} line along the k-axis) than for $T = 1,200$ (the 4^{th} line along the k-axis.) Then we examine the charts shown in Fig. 7(a). Although the size of candidate sites drops faster for $T = 1,024$ than for $T = 1,200$ at the initial few steps, the reduction is faster at the final few steps (step 8 to 10) for $T = 1,200$ than for $T = 1,024$. Note that there are usually more coefficients retained at the lower levels. Hence, the sizes of candidate sites at the final few steps dominate the total bandwidth consumption. As a result, the total bandwidth is smaller when $T = 1,200$ than when $T = 1,024$. For the case of $k = 25$, the final sizes of candidate sites are closer for both $T = 1,024$ and $1,200$. As a result, the bandwidth drop between these two ranges is less obvious (see Fig. 6(b)). For $T = 2,048$, for both k=5 and k=25 cases, although it generally has a smaller size of candidate sites than others, however, a lot more retained coefficients are involved for a larger range, consuming more bandwidth.

The second experiment, as shown in Fig. 8, examined the impacts of query range T and total number of sites M on bandwidth consumption. The time range settings were the same as those used in the previous experiment. The number of sites increased from 60 to 300. From Fig.8(a), the bandwidth consumption of the naive approach increases significantly not only as query range T increases, but also as the number of sites increases. In contrast to the naive approach, Fig.8(b) shows that LEEWAVE is not sensitive to the number of sites. This is because unnecessary coefficients are not distributed by the query initiator. The results under different query ranges behave similarly to those from the previous set of experiments. We also show the average number of candidate sites at each step for $M = 60$ and $M = 300$ cases in Fig. 9.

5.2 Evaluation with synthetic data

The synthetic data were generated by a random walk data model proposed in [31]. For a stream S_i, it was generated as follows:

$$S_i = 100 + \sum_{j=1}^{i}(u_j - 0.5),$$

where u_j was randomly picked from [0,1].

We generated 1,000 streams in total, where each stream has 20,000 data points. The default parameter settings, unless otherwise specified, were 500 for the number of sites and 10 for k. For a given number of sites, the 1,000 streams were evenly distributed among them.

The third experiment, shown in Fig. 10, studied the impacts of query range T and k on bandwidth consumption for a given $M = 500$. The query range was varied from the following: $3,000$, $6,000$, $8,192$, $10,000$, $14,000$ and $16,384$. k varied from 5 to 25. From Fig. 10, LEEWAVE consumes dramatically less bandwidth, when compared with the naive approach. The difference here is almost one order of magnitude and it is more significant than the difference in the real-data case. This is due to the nature of the data sets. The synthetic data were generated randomly. Hence, the deviations between streams were much larger than those between temperature streams of different cities. It is easier to separate apart those dissimilar streams in synthetic data set by using only the first few levels of coefficients in distance computation. This can be clearly seen in Fig. 11, where the size of candidate sites shrinks quickly after the first few steps. Sometimes the final answers can be obtained at an intermediate level. This is why the size of candidate sites approaches to zero in Fig. 11. When k is larger, the size of candidate sites is higher. This also shows that a higher upper bound has less pruning ability.

Finally, the fourth experiment, shown in Fig. 12, studied the impacts of T and M on bandwidth consumption. The query range settings were the same as those used in the third experiment. The number of sites increases from 200 to $1,000$. From Fig. 12(a), we observe the same phenomena as in real data. The bandwidth consumption of the naive approach increases significantly as both query range and the number of sites increase. In contrast to the naive approach, LEEWAVE saves a huge amount of bandwidth. From Fig. 13, the speed at which the size of candidate sites reduces is faster when $M = 1,000$ than when $M = 200$. This shows that LEEWAVE outperforms the naive approach, especially when the number of sites is large.

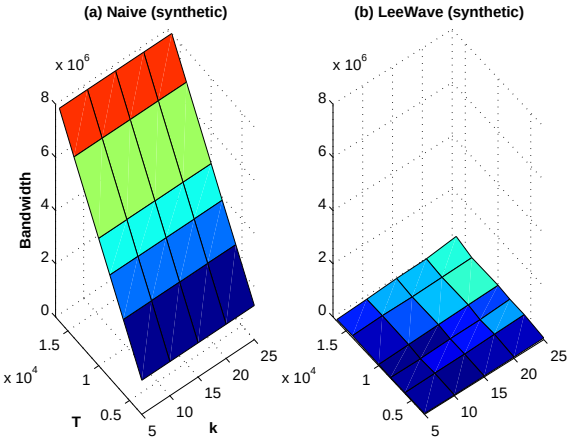

Figure 10. Impacts of T and k on bandwidth consumption with synthetic data.

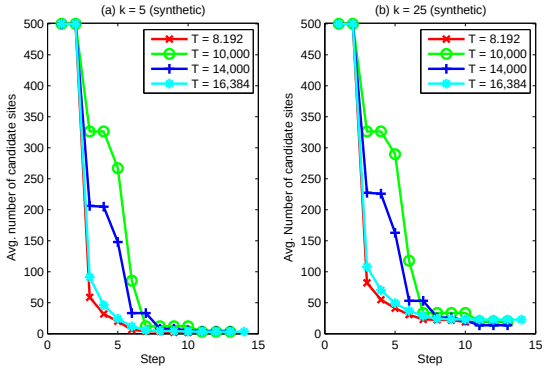

Figure 11. Size of candidate sites at each step of LEEWAVE in Fig. 10.

6 Conclusion

In this paper, we presented LEEWAVE - a bandwidth-efficient approach to processing range-specified kNN queries in a distributed streaming environment. Leveraging the multi-resolution property of wavelet coefficients, LEEWAVE distributes the relevant wavelet coefficients to the peer sites in a level-wise fashion. Starting from the top level and moving down one level at a time, the query initiator only sends single-level coefficients to a gradually reduced set of candidate sites. In order to overcome the challenge of pruning the candidates without knowing all the relevant coefficients, we devised and maintained a similarity range for each candidate stream. This similarity range is tightened with each returned level distance. This increasingly tightened similarity range enables the query initiator to effectively prune the candidates. Significant bandwidth savings are achieved by avoiding sending unnecessary coefficients. We conducted extensive experiments with both real and synthetic data. The results show that (1) When

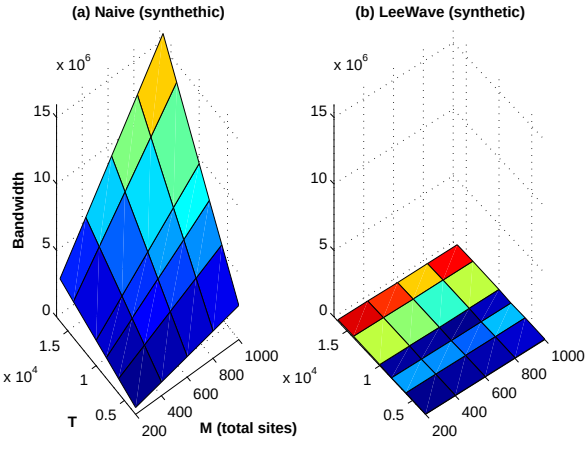

Figure 12. Impacts of T and M on bandwidth consumption with synthetic data.

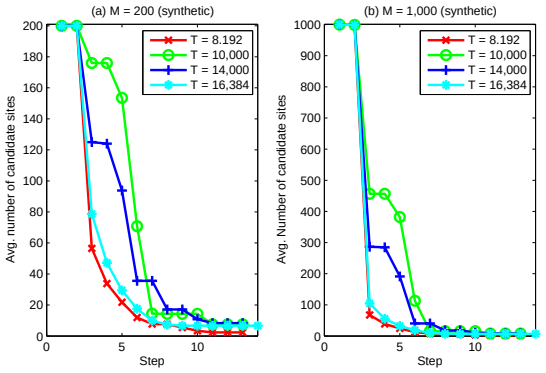

Figure 13. Size of candidate sites at each step of LEEWAVE in Fig. 12.

compared with a naive approach under a wide range of conditions, LEEWAVE uses significantly less bandwidth, especially when query range or the number of sites is large. (2) When the deviations among the streams are larger, the performance advantage of LEEWAVE is more significant.

References

[1] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The design of the Borealis stream processing engine. In *CIDR*, 2005.

[2] M. G. Albanesi, M. Ferretti, and A. Giancane. A compact wavelet index for retrieval in image database. In *Proc. of IEEE Int. Conf. on Image Analysis and Processing*, 1999.

[3] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thomas, R. Varma, and J. Widom. STREAM: The Stanford stream data manager. *IEEE Data Engineering Bulletin*, 26, 2003.

[4] H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, E. Galvez, J. Salz, M. Stonebraker, N. Tatbul, R. Tibbetts, and S. Zdonik. Retrospective on Aurora. *VLDB Journal*, 2004.

[5] A. Bulut and A. K. Singh. SWAT: Hierarchical stream summarization in large networks. In *IEEE ICDE*, 2003.

[6] A. Bulut, R. Vitenberg, and A. K. Singh. Distributed data streams indexing using content-based routing paradigm. In *IPDPS*, 2005.

[7] F. K.-P. Chan, A. W.-C. Fu, and C. Yu. Haar wavelets for efficient similarity search of time-series: With and without time warping. *IEEE TKDE*, 15(3):686–705, 2003.

[8] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, V. Raman, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

[9] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi. Holistic aggregates in a networked world: distributed tracking of approximate quantiles. In *ACM SIGMOD*, 2005.

[10] G. Cormode, M. Garofalakis, and D. Sacharidis. Fast approximate wavelet tracking on streams. In *EDBT*, 2006.

[11] L. Gao, Z. Yao, and X. S. Wang. Evaluating continuous nearest neighbor queries for streaming time series via prefetching. In *ACM CIKM*, 2002.

[12] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. One-pass wavelet decompositions of data streams. *IEEE TKDE*, 15(3):541–554, 2003.

[13] L. Girod, Y. Mei, R. Newton, S. Rost, A. Thiagarajan, H. Balakrishnan, and S. Madden. XStream: A signal-oriented data stream management system. In *IEEE ICDE*, 2008.

[14] S. Guha and B. Harb. Wavelet synopsis for data streams: minimizing non-euclidean error. In *ACM SIGKDD*, 2005.

[15] A. Haar. Zur theorie der orthogonalen funktionensysteme. *Mathematische Annalen*, 69, 1910.

[16] H.-P. Hung and M.-S. Chen. Efficient range-constrained similarity search on wavelet synopses over multiple streams. In *ACM CIKM*, 2006.

[17] P. Karras and N. Mamoulis. One-pass wavelet synopses for maximum-error metrics. In *VLDB*, 2005.

[18] N. Koudas, B. C. Ooi, K.-L. Tan, and R. Zhang. Approximate NN queries on streams with guaranteed error/performance bounds. In *VLDB*, 2004.

[19] J. Lin, M. Vlachos, E. J. Keogh, and D. Gunopulos. Iterative incremental clustering of time series. In *EDBT*, 2004.

[20] X. Liu and H. Ferhatosmanoglu. Efficient k-NN search on streaming data series. In *SSTD*, 2003.

[21] A. Manjhi, S. Nath, and P. B. Gibbons. Tributaries and deltas: efficient and robust aggregation in sensor network streams. In *ACM SIGMOD*, 2005.

[22] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (recently) frequent items in distributed data streams. In *IEEE ICDE*, 2005.

[23] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *ACM SIGMOD*, 1998.

[24] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *ACM SIGMOD*, 2003.
[25] I. Popivanov and R. J. Miller. Similarity search over time-series data using wavelets. In *IEEE ICDE*, 2002.
[26] I. Sharfman, A. Schuster, and D. Keren. A geometric approach to monitoring threshold functions over distributed data streams. In *ACM SIGMOD*, 2006.
[27] J. M. Steele. *The Cauchy-Schwarz Master Class: An Introduction to the Art of Mathematical Inequalities*. Cambridge University Press, New York, NY, USA, 2004.
[28] StreamBase Systems. http://www.streambase.com/, May 2007.
[29] W.-G. Teng, M.-S. Chen, and P. S. Yu. Resource-aware mining with variable granularities in data streams. In *SIAM Data Mining*, 2004.
[30] K.-L. Wu, P. S. Yu, B. Gedik, K. W. Hildrum, C. C. Aggarwal, E. Bouillet, W. Fan, D. A. George, X. Gu, G. Luo, and H. Wang. Challenges and experience in prototyping a multimodal stream analytic and monitoring application on system S. In *VLDB*, Sept. 2007.
[31] Y. Zhu and D. Shasha. Statstream: statistical monitoring of thousands of data streams in real time. In *VLDB*, 2002.
[32] Y. Zhu and D. Shasha. Efficient elastic burst detection in data streams. In *ACM SIGKDD*, 2003.

Appendix

Proof of Theorem 1: At level ρ, the upper bound is:

$$accDst^{\rho}(S_{ref}, S_x) + \sum_{l=1}^{\rho-1}\sum_{p}([n_{(l,p)}^{(ref)}]^2 + [n_{(l,p)}^{(x)}]^2) \times 2^l$$
$$+2 \times \sqrt{\sum_{l=1}^{\rho-1}\sum_{p}([n_{(l,p)}^{(ref)}] \times 2^l)^2 \times \sum_{l=1}^{\rho-1}\sum_{p}[n_{(l,p)}^{(x)}]^2}. \quad (6)$$

From level ρ to $\rho-1$, the upper bound is reduced by the amount of:

$$-Dst^{\rho-1}(S_{ref}, S_x) + \sum_{p}([n_{(\rho-1,p)}^{(ref)}]^2 + [n_{(\rho-1,p)}^{(x)}]^2) \times 2^l$$
$$+2 \times (\sqrt{\sum_{l=1}^{\rho-1}\sum_{p}([n_{(l,p)}^{(ref)}] \times 2^l)^2 \times \sum_{l=1}^{\rho-1}\sum_{p}[n_{(l,p)}^{(x)}]^2}$$
$$-\sqrt{\sum_{l=1}^{\rho-2}\sum_{p}([n_{(l,p)}^{(ref)}] \times 2^l)^2 \times \sum_{l=1}^{\rho-2}\sum_{p}[n_{(l,p)}^{(x)}]^2}). \quad (7)$$

To prove the upper bound is non-increasing, we need to prove that Eq. (7) is ≥ 0. For ease of exposition, we let $\alpha_{(l,p)} = [n_{(l,p)}^{(ref)}] \times 2^l$, and $\beta_{(l,p)} = [n_{(l,p)}^{(x)}]$. By expanding the term $Dst^{\rho-1}(S_{ref}, S_x) = \sum_p([n_{(\rho-1,p)}^{(ref)}]^2 + 2n_{(\rho-1,p)}^{(ref)} n_{(\rho-1,p)}^{(x)} + [n_{(\rho-1,p)}^{(x)}]^2) \times 2^l$ and using substitutes, Eq. (7) becomes:

$$2\alpha_{(\rho-1,p)}\beta_{(\rho-1,p)}$$
$$+2 \times (\sqrt{\sum_{l=1}^{\rho-1}\sum_{p}\alpha_{(l,p)}^2 \times \sum_{l=1}^{\rho-1}\sum_{p}\beta_{(l,p)}^2}$$
$$-\sqrt{\sum_{l=1}^{\rho-2}\sum_{p}\alpha_{(l,p)}^2 \times \sum_{l=1}^{\rho-2}\sum_{p}\beta_{(l,p)}^2}). \quad (8)$$

Now the task becomes to prove Eq. (8) ≥ 0. We divide Eq. (8) into two cases, which is when $\alpha_{(\rho-1,p)}\beta_{(\rho-1,p)} \geq 0$ and $\alpha_{(\rho-1,p)}\beta_{(\rho-1,p)} < 0$. If it is the former case, Eq. (8) ≥ 0 must be true. Therefore, we only need to prove the later case. By reformulating Eq. (8) and omitting the factor 2, to prove Eq. (8), we need to prove the following:

$$\sqrt{\sum_{l=1}^{\rho-1}\sum_{p}\alpha_{(l,p)}^2 \times \sum_{l=1}^{\rho-1}\sum_{p}\beta_{(l,p)}^2} \geq$$
$$-\alpha_{(\rho-1,p)}\beta_{(\rho-1,p)} + \sqrt{\sum_{l=1}^{\rho-2}\sum_{p}\alpha_{(l,p)}^2 \times \sum_{l=1}^{\rho-2}\sum_{p}\beta_{(l,p)}^2}. \quad (9)$$

Since both the left-hand side and the right-hand side of Eq. (9) are positive, we can square the terms of both side while the inequality still holds. The square value of left-hand side of Eq. (9) is:

$$\sum_{l=1}^{\rho-1}\sum_{p}\alpha_{(l,p)}^2 \times \sum_{l=1}^{\rho-1}\sum_{p}\beta_{(l,p)}^2$$
$$= (\alpha_{(\rho-1,p)}^2 + \sum_{l=1}^{\rho-2}\sum_{p}\alpha_{(l,p)}^2) \times (\beta_{(\rho-1,p)}^2 + \sum_{l=1}^{\rho-2}\sum_{p}\beta_{(l,p)}^2)$$
$$= \alpha_{(\rho-1,p)}^2\beta_{(\rho-1,p)}^2 + \alpha_{(\rho-1,p)}^2 \sum_{l=1}^{\rho-2}\sum_{p}\beta_{(l,p)}^2$$
$$+\beta_{(\rho-1,p)}^2 \sum_{l=1}^{\rho-2}\sum_{p}\alpha_{(l,p)}^2 + \sum_{l=1}^{\rho-2}\sum_{p}\alpha_{(l,p)}^2 \sum_{l=1}^{\rho-2}\sum_{p}\beta_{(l,p)}^2 \quad (10)$$

The square value of the right-hand side of Eq. (9) is:

$$\alpha_{(\rho-1,p)}^2\beta_{(\rho-1,p)}^2$$
$$-2 \times \alpha_{(\rho-1,p)}\beta_{(\rho-1,p)}\sqrt{\sum_{l=1}^{\rho-2}\sum_{p}\alpha_{(l,p)}^2 \times \sum_{l=1}^{\rho-2}\sum_{p}\beta_{(l,p)}^2}$$
$$+\sum_{l=1}^{\rho-2}\sum_{p}\alpha_{(l,p)}^2 \sum_{l=1}^{\rho-2}\sum_{p}\beta_{(l,p)}^2 \quad (11)$$

Compare Eq. (10) with Eq. (11), by eliminating the same terms, we only need to prove

$$\alpha_{(\rho-1,p)}^2 \sum_{l=1}^{\rho-2}\sum_{p}\beta_{(l,p)}^2 + \beta_{(\rho-1,p)}^2 \sum_{l=1}^{\rho-2}\sum_{p}\alpha_{(l,p)}^2$$
$$\geq$$
$$-2 \times \alpha_{(\rho-1,p)}\beta_{(\rho-1,p)}\sqrt{\sum_{l=1}^{\rho-2}\sum_{p}\alpha_{(l,p)}^2 \times \sum_{l=1}^{\rho-2}\sum_{p}\beta_{(l,p)}^2}. \quad (12)$$

By using the inequality of arithmetic and geometric means, Eq. (12) holds, and so does Eq. (9). Q.E.D.