Approximate Data Stream Joins in Distributed Systems

Vassil Kriakov
Polytechnic University
Brooklyn, NY 11201
vassil@milos.poly.edu

Alex Delis
University of Athens
Athens, Greece 15771
ad@di.uoa.gr

George Kollios
Boston University
Boston, MA 02215
gkollios@cs.bu.edu

Abstract

The emergence of applications producing continuous high-frequency data streams has brought forth a large body of research in the area of distributed stream processing. In presence of high volumes of data, efforts have primarily concentrated on providing approximate aggregate or top-k type results. Scalable solutions for providing answers to window join queries in distributed stream processing systems have received limited attention to date. We provide a solution for the window join in a distributed stream processing system which features reduced inter-node communications achieved through automatic throughput handling based on resource availability. Our approach is based on incrementally updated discrete Fourier transforms (DFTs). Furthermore, we provide formulae for computing DFT compression factors in order to achieve information reduction. We perform WAN-based prototype experiments to ascertain the viability and establish the effectiveness of our method. Our experimental results reveal that our method scales in terms of throughput and error rates, achieving sub-linear message complexity in domains that exhibit a geographic skew in the joining attributes.

1 Introduction

In many modern applications, data from continually-emitting sources must be cross-referenced rendering the window join an operation of paramount importance. Typically, such streams from geographically dispersed sources must be cross-referenced at multiple locations in order to answer various queries. For instance, data traffic analysis for discovering and tracking malicious IP packets calls for join queries on packets flowing across multiple nodes and domains [25]. Very large sensor networks deployed for traffic monitoring, tracking of commercial goods, and temperature and humidity measurement for environmental monitoring pose similar requirements for joins [21]. In addition, many financial applications seek to exploit arbitrage situations by responding in a timely manner to real-time bid/ask offerings arriving from multiple high-frequency stock exchange streams. The efficient and accurate computation of the window join involving such data originating from multiple and highly dynamic streams is a challenging task. In this paper, we provide answers to the distributed sliding window join query [22]; here, the tuples of interest are only those which have arrived within a pre-defined window. The width of the window is defined in terms of either time duration, number of tuples, or a landmark (i.e. until a specific tuple is observed). Our approach is general, and is not affected by the specific type of definition used. Thus, without loss of generality, we assume that the window is measured in number of tuples. In the remainder of the paper, we refer to the window join as simply the join.

Recent work on approximating queries in the context of distributed streams has focused on aggregates which are single-state operators [3, 6, 9, 26]. Joins are notably more difficult to compute, especially in distributed environments [23]. Our work builds on previous centralized schemes [8] and, through different analytic techniques, extends them further to provide approximate materialized answers for distributed join queries. The approximation of window-join output has been analyzed in centralized environments, predominantly through sampling approaches based on various tuple replacement policies [11, 12, 27, 30]. To the best of our knowledge no prior research efforts have focused on the realization of join-queries in distributed stream processing (DSP) systems.

The distributed nature of processing the streaming data makes it problematic to compute joins as the latter require inter-node communications of high complexity. Namely, in a distributed system of N nodes, $N-1$ data transmissions per tuple are required in order to carry out the exact join computation. Consequently, our goal is to provide minimal inter-node communication when answering join queries over distributed streams. The intuition of our approach is that the frequency of intra-node messaging should be varied adaptively between different nodes, depending on their contribution to the final result set. To this end, our contributions include:

27th International Conference on Distributed Computing Systems (ICDCS'07)
0-7695-2837-3/07 $20.00 © 2007 IEEE

- mechanisms for constant or logarithmic message complexity for intra-node communications
- resource-dependent message complexity reduction in skewed data distributions
- worst-case scenario detection under uniform data distributions
- compression factor thresholds for join result-set estimation

Compared to a baseline scenario with $(N-1)$ communication cost, our method achieves:

1. sub-linear message complexity with respect to available dispersed resources
2. an ϵ-*error vs. communication* trade-off for window joins in distributed environments
3. *lossless* DFT coefficient compression up to a factor of 256 provides reductions in communication costs.

We formalize the problem in Section 2 and describe our architecture in Section 3. Section 4 gives background information on DFTs while justifying our choice for their use. The models and heuristics for communication resource management are described in Section 5. Our experimental results are presented and analyzed in Section 6. Related work is discussed in Section 7 and concluding remarks are made in Section 8.

2 Problem Definition

We provide a formal definition for the problem of approximately answering join queries over streams in a distributed environment. Figure 1 illustrates the distributed framework in which the stream processing is accomplished. We base our analysis and experimentation on the following

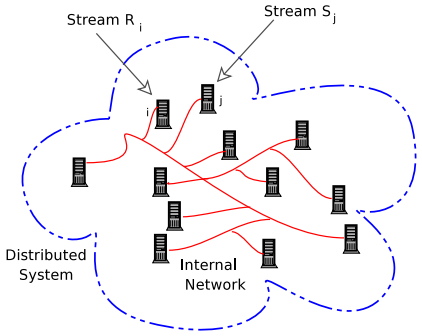


Figure 1. Framework for Distributed Stream Processing (DSP).

definition of *message complexity*:

Definition I. *Message complexity* is the number of messages transmitted by a single node for each arriving tuple.

In Figure 1, streams R and S are distributed among N processing nodes in such a manner that node N_i maintains a segment R_i and S_i of the entire window of each stream. We wish to compute an approximate result of the window join $R \bowtie S$, given a communication constraint. Assume that the window size is W at each node in the system. With N distributed sites participating in the computation, the effective window size is $\hat{W} = N \times W$. Each stream's window can be perceived as being partitioned into N discrete segments $R_{1..N}$ and $S_{1..N}$. To evaluate the exact result of $R \bowtie S$ in the distributed case, N^2 joins must be performed between the partitions [23]: $R \bowtie S = (R_1 \bowtie S_1) \cup (R_1 \bowtie S_2) \cup ... \cup (R_1 \bowtie S_N) \cup ... \cup (R_N \bowtie S_N)$. In accord with Definition I, this equates to a *message complexity* of $(N-1)$; i.e. when a tuple arrives at one node it must be forwarded to the remaining $N-1$ nodes in order to provide the complete result of the join. We propose a method for approximate join estimation by bounding the number of joins in the above equation in the range of $O(N)$ to $O(Nlog(N))$. We accomplish this by providing controls for adjusting the *message complexity* in the range $[O(1), O(log(N))]$.

Our approach is based on one fundamental principle: in the above equation, each of the joins of $R_{1..N}$ with $S_{1..N}$ may contribute a different number of tuples, depending on the data and frequency distributions of the arriving tuples. In centralized environments, approximation methods which seek the goal of maximizing the result-set are classified as *MAX*-subset [11]. Our method for approximating this final result falls in this category, as it takes into account the cross-correlation characteristics of the joining attributes in order to restrict the joins to partitions which contribute the largest number of tuples to the final result, In this regard, if the materialized result set of the join is Ψ, we measure the error ϵ as the percentage of tuples that were not reported in the approximate result set $\hat{\Psi}$:

$$\epsilon = \frac{(|\Psi| - |\hat{\Psi}|)}{|\Psi|} \quad (1)$$

Our goal is to minimize ϵ, while providing message complexities within the bounds of $O(1)$ to $O(log(N))$. Our algorithm provides best-effort ϵ reduction depending on resource availability. We provide analytical and empirical evidence of the algorithm's superior performance under skewed (Zipfian) data distributions of the joining attribute and show a detection mechanism for the worst-case scenario with uniform data distributions.

3 System Architecture

Our goal is to create a platform that can effectively handle multi-stream join queries while exhibiting superior performance under high-frequency streams in a distributed platform (Figure 1). Here, a set of interconnected nodes communicate in order to collaboratively answer window

join queries on a set of data streams. The communications architecture is such that every node is able to converse with every other node. This is different from previous research which employs a central coordinator for query processing [8].

For simplicity of the discussion, we assume that each stream R and S is distributed across all N nodes in the network and a join-query is disseminated to all networked nodes which receive streams relevant to the query. We analyze both cases where the joining attribute distribution is (1) uniform and (2) skewed across the nodes. For the remainder of the paper we refer to these two scenarios simply as uniform and skewed data distributions.

We measure the flow f between two nodes as the number of messages transmitted per arriving tuple. The straightforward implementation is not scalable since it requires the transmittal of $(N-1)$ messages per tuple, or $f = 1$. The net flow f_{NET} at each node is $(N-1) \times f$ and since there are N nodes, the net flow in the entire system is $F = N \times (N-1) \times f \approx O(N^2)$. Therefore, the growth of the total number of messages transmitted in the system is polynomial with the number of nodes. In order for a distributed system to provide for robust scalability this growth must be linear with the number of nodes in the system.

Our improvement to the baseline system is to allow each node to adapt its flow towards every other node in the system so that each node's net flow f_{NET} is in the bounds $[1, log(N)]$. A crude method is to limit f to $1/(N-1)$. However, this does not take into consideration tuple distributions and, can result in very large ϵ (Eqn. 1). To reduce the message complexity without dramatically increasing ϵ, we identify nodes whose streams exhibit correlated characteristics. Thus, we determine the flow factors for each node pair individually as shown in Figure 2. For instance, two

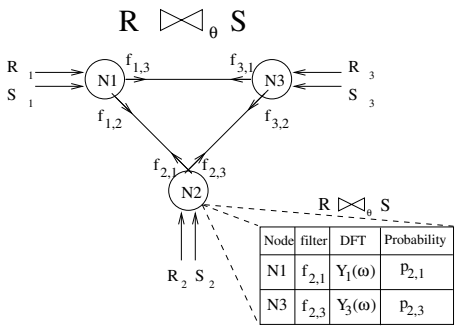


Figure 2. Three node system using restricted DSP. Each directional communication is restricted according to the statistical properties of the node's relationship with the other nodes.

nodes with very dissimilar streams will ultimately feature very few joining tuples. In this situation, mutual knowledge of the most up-to-date join operator state is not required for the two nodes. In this paper, we quantify a *similarity* measure, defined subsequently in Equation 4, for two streams and propose a technique to adaptively vary the rate of join state synchronization between the distributed nodes. This flow filtering is depicted in Figure 2 where the allowable rate of flow is limited by the probability $p_{i,j}$ that a tuple will be transmitted from node N_i to node N_j. The efficient calculation of $p_{i,j}$ is detailed in Section 5.

4 Background on DFTs

We give a brief overview of discrete Fourier transforms (DFTs) and justify our choice of DFTs for the proposed solution of the problem at hand. DFTs map time-series data into the frequency spectrum. Let us model the value of the joining attribute as a random variable x. The discrete Fourier transform of the random variable x is $X(\omega) = \sum_{n=-\infty}^{\infty} x[n]e^{-jn\omega}$, and for a finite data set of size W is defined as:

$$X(\omega) = \sum_{n=1}^{W} x[n]e^{-jn\omega} \tag{2}$$

Thus, the DFT can be expressed by the W coefficients produced from the summation. The original signal can be reconstructed from the DFT coefficients using the inverse DFT:

$$x[n] = \frac{1}{W} \sum_{k=1}^{W} X(k)e^{\frac{2\pi j}{W}kn} \tag{3}$$

Data reduction can be achieved by discarding low-energy coefficients of higher frequencies since they do not contribute significantly in the reconstruction process. A similar approach is shown to work well in [14], where time-series data is represented by just a few coefficients for addressing the subsequence matching problem. In that respect, DFTs can be viewed as means for efficient data compression. We exploit this property to avoid transmitting entire tuple sets.

In contrast, the more commonly used wavelets require all n coefficients to fully reconstruct a signal [29]. One benefit of wavelets is that they can be computed incrementally and, in that respect, they are well suited for streaming data. However, in the case of DFTs, the advantages of transmitting $n/2$ coefficients can be combined with *incremental* computation at the cost of a small approximation error on the order of $O(10^{-16})$ per coefficient [4]. Analytical methods for determining an application-specific control vector for the trade-off between computational cost and approximation error in the DFT coefficients are presented in [28]. Based on [28], we set the control vector such that the arithmetic complexity is reduced by a factor of 10 with a probability for

completion of the DFT approximation greater than 0.95. In our empirical evaluation, computing DFTs did not represent a significant portion of the CPU utilization for windows of size up to 1,000,000 tuples. We compare the CPU cost of computing DFTs, incremental DFTs and AGMS[1] sketches [1], which are commonly used for join-size estimation. The average over 100 experiments of the times taken to compute the DFT, incremental DFT and AGMS sketches are reported in Table 1, where we can observe that the performance of the incremental DFT algorithm is comparable to that of AGMS sketches relative to the DFT algorithm. Therefore we believe that DFTs are viable means to compute approximate joins in a distributed stream processing system.

W	DFT	iDFT	AGMS
80,000	9	<1	<1
250,000	34	3.20	2.10
500,000	70	7.40	5.60
1,000,000	149	18.10	12.70

Table 1. CPU time in seconds to compute DFTs, incremental DFTs (iDFT), and AGMS sketches on a 400MHz UltraSPARC CPU. The updates are performed incrementally per tuple over a stream of 100,000,000 tuples.

Since the DFT is essentially a compressed representation of the original data, remote nodes can use this information to compute statistics of data which is not present locally. We use these statistics to create an efficient filtering mechanism for targeting tuples only to those nodes which can contribute the most results in the join query.

5 Algorithms

5.1 The Baseline Case

Our baseline model for performance comparison is the simple case when N nodes transmit $N-1$ messages for every tuple received. This provides complete results for joins, at a rather excessive cost of *message complexity*. Such costs are prohibitive in distributed environments where scalability is important. Even in networks with high-speed connections, such as clusters of workstations or blade servers, the computing resources cannot cope with applications that require the processing of very high frequency streams. Therefore, we propose methods that yield approximate answers in sub-linear message complexity.

5.2 Reducing Message Complexity

Let us assume that a new tuple arrives at node i via stream R. Node i runs the following algorithm: for each of the remaining $N-1$ nodes in the system, transmit the tuple to node j with probability $\{p_{i,j}|i \neq j\}$, as shown in Figure 2. We use a weighted factor model to adjust the probabilities for the desired message complexity bounds:

$$w_i \times p_{i,j} = w_i \times \rho_{i,j} = w_i \times \frac{\sigma_{i,j}}{\sqrt{\sigma_i \sigma_j}} \tag{4}$$

where $\rho_{i,j}$ is the cross-correlation *coefficient* and w_i is the weighing factor for node i. In the above equation, $\rho_{i,j}$ is expressed in terms of the cross-correlation $\sigma_{i,j}$ and the individual auto-covariances σ_i and σ_j of the set of tuples at nodes i and j, respectively [24]. The cross-correlation coefficient $\rho_{i,j}$ quantifies the similarity between stream segments at the two nodes. If two joining streams have high selectivity they are statistically similar and exhibit high cross-correlation, therefore having a high cross-correlation coefficient. We explain how w_i is computed in Section 5.2.2.

In order for node i to compute $p_{i,j}$, it first calculates the cross-correlation $\sigma_{i,j}$. To accomplish this, we first model the joining attributes of two streams at nodes i and j on which the join is performed as two discrete random variables $x[n]$ and $y[m]$ respectively. Therefore, the cross-correlation $\sigma_{i,j}$ can be expressed probabilistically in terms of the expected value of the random variables as:

$$\sigma_{i,j} = R_{xy}(n,m) = E\{x[n]y^*[m]\} \tag{5}$$

where $y^*[m]$ is the complex conjugate of $y[m]$. Notice however, that in order for node i to compute $\sigma_{i,j}$, it must attain knowledge of the tuples $y[m]$ at the remote node j. To avoid this, we compute R_{XY} from the individual DFTs of the two streams. This is possible since the DFT is a *linear* transformation of the input data [24]. If the DFTs of the random variables x and y are X and Y, respectively, the cross-correlation of the DFTs can be expressed as:

$$R_{XY}(u,v) = E\{X(u)Y^*(v)\} \tag{6}$$

By substituting Equation 2 into Equation 6 and redistributing the terms we obtain:

$$R_{XY}(u,v) = \sum_{n=1}^{W} e^{jn(v-u)} \times \sum_{m=1}^{W} E\{x[n+m]y^*[n]\}e^{-jmu} \tag{7}$$

Using Equation 5, Equation 7 becomes:

$$R_{XY}(u,v) = \sum_{n=1}^{W} e^{jn(v-u)} \times \sum_{m=1}^{W} R_{xy}[n+m,m]e^{-jmu} \tag{8}$$

We can see that the DFT cross-correlation $R_{XY}(u,v)$ is a *linear* function of the original data set's cross-correlation R_{xy}. Through the linearity property of the DFT, it is obvious that the cross-correlation $\sigma_{i,j} = R_{xy}$ of the two random variables can be computed from their DFT cross-correlation

[1] AGMS is an acronym of the last names of the authors of [1].

R_{XY}. In a similar fashion, the auto-covariances σ_i and σ_j can be computed from the respective DFTs of the joining attributes of nodes i and j. For brevity, we omit the details. Armed with $\sigma_{i,j}$, σ_i and σ_j, we can compute the cross correlation coefficient $\rho_{i,j}$ as defined in Equation 4.

5.2.1 Reducing the Overhead of Calculating DFTs

In practice, the summation $\sum_{m=1}^{W} R_{xy}[n+m,m]e^{-jmu}$ in Equation 8 is known as the power spectrum $S_{xy}(u)$ of two strict sense stationary random variables [24]. Furthermore, if we rewrite $\sum_{n=1}^{W} e^{jn(v-u)}$ as $2\pi\delta(u-v)$ with $\delta(u)$ = $\frac{1}{2\pi}\sum_{n=1}^{W} e^{-jnu}$ we obtain the simplified version of the DFT cross-correlation $R_{XY}(u,v) = 2\pi\delta(u-v)S_{xy}(u)$. For a DFT of W coefficients, the power spectrum $S_{xy}(u)$ can be estimated using fast Fourier transform (FFT) methods in $O(W)$ time [7, 19]. As detailed in Section 4, the DFT coefficients are computed incrementally in constant time complexity for each newly arrived tuple. At regular intervals, as specified by the control vector introduced in Section 4, the DFT is completely recalculated.

5.2.2 Establishing Bounds on *Message Complexity*

As previously described in Figure 2, node i transmits a tuple to node j with probability $p_{i,j}$. Therefore, for every newly arriving tuple the expected number of messages transmitted by node i is $T_i = \sum_{j=1}^{N-1} p_{i,j}$. To achieve our goal as set in Section 2, for every node i, we must constrain T_i to the range $[O(1), O(log(N))$. To accomplish this we adjust T_i individually at each node by introducing the previously discussed weighting factor w_i. This is necessary because the probabilities $p_{i,j}$ are derived from the correlation metric and are not controlled variables. Thus, T_i becomes $T_i = w_i \times \sum_{j=1}^{N-1} p_{i,j}$, such that the following inequality holds:

$$1 \leq w_i \times \sum_{j=1}^{N-1} p_{i,j} \leq log(N) \tag{9}$$

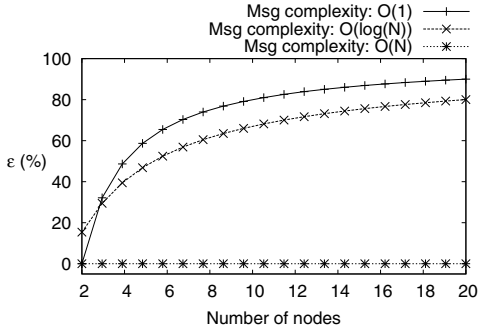
Theorem 1. For $T_i = 1$ ($i \in [1..N]$), under uniform data distribution, the upper bound on the error ϵ is $(1 - \frac{2}{N})$.

Proof. Because each tuple has an equal probability of joining with the same number of tuples at any node, the expected contribution of each node is equal, say ω. The size of the complete result-set is therefore $|\Psi| = \omega N$. The actual result set consists of the ω tuples produced from joining the two streams at the local node, and ω tuples from the join with the stream at the one remote site to which the tuple was sent. Therefore, the reported number of tuples is $|\hat{\Psi}| = 2\omega$ and using Equation 1, we get $\epsilon = \frac{\omega N - 2\omega}{\omega N}$. □

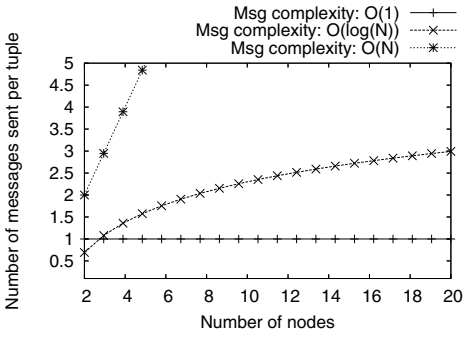
A uniformly distributed set of joining attributes is the worst case scenario since a tuple is sent to only one out of $N-1$ sites that contain $1/N$ of the joining tuples. This means that ϵ is unbounded and will grow with N. This is the worst case scenario for any distributed join algorithm. Fortunately, nodes in our system independently detect this case by observing the variance of $p_{i,j}$ for each neighbor j. A very small variance in the filter probabilities indicates equal correlation with all neighbors. Unfortunately, the only solution in this worst case scenario is to revert to a heuristics based method, such as round-robin, for directing tuples to neighbor nodes.

Theorem 2. For $T_i = log(N)$, under uniform data distribution, the error bound is $(1 - \frac{1+log(N)}{N})$.

The proof is along the same lines as in Theorem 1 and we omit it for brevity. Figures 3 (a) and (b) display the analytical error bounds and message complexities under uniform data distribution as defined in Theorems 1 and 2. Although



(a)



(b)

Figure 3. Message complexity and error bounds for our theoretical limits with uniform data distribution (worst case).

the error rate ϵ in Figure 3 (a) grows quickly with the number of nodes for both $T_i = 1$ *and* $T_i = log(N)$, in Figure 3 (b) we can see a three-fold reduction in the number of messages transmitted with $T_i = log(N)$. As previously explained, this is the worst case scenario and the error rates may be unacceptable for systems composed of large numbers of nodes. Our algorithm detects such a situation and

provides for a fall-back tuple distribution policy such as round robin.

The true benefit of our approach is revealed when the tuple distribution is non-uniform across the sites. This is commonly accepted as the case for real-world applications [13, 23, 24]. Under such distributions, assuming statistical significance of the correlation coefficients, the tuples are transmitted to sites that are more likely to produce the largest number of joining tuples.

Theorem 3. Under Zipfian data distribution with skew α the error bounds for message complexities $O(1)$ and $O(log(N))$ are $1.0 - \sum_{i=1}^{2} \alpha^i/N$ and $1.0 - (\alpha - \alpha^{log(N)+1})/(1-\alpha)$, respectively.

These limits can be observed in Figure 4 for a distributed system composed of up to 20 sites. The significance of this

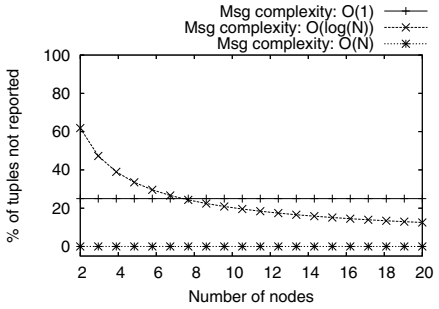


Figure 4. Error bounds for our theoretical limits with Zipfian data distribution ($\alpha = 0.4$).

observation is that, unlike in the case of uniform distribution, with $O(log(N))$ message complexity ϵ can asymptotically approach zero as the number of participating nodes increases.

In summary, we have shown that individual nodes can efficiently and incrementally calculate DFTs of their stream segments. Once the initial DFTs are computed, each node sends its DFT coefficients to all sites that maintain segments of the joining streams. Consequently, each site uses its DFT and the remote site's DFT to compute the probabilities $p_{i,j}$. Thus, each node takes a probabilistic approach in determining how to distribute tuples to the remaining nodes.

5.3 Tuple Matching using DFTs

The second benefit that DFTs give us is the ability to reconstruct the original data set from the DFT coefficients as per Eqn. 3. By reconstructing a remote node's joining attributes, a node can test its tuples for membership in the remote data set. The only information exchanged is a set of W/κ DFT coefficients, where κ is the compression factor. We will further refer to this method as DFTT.

The process of calculating the inverse DFT from a set of coefficients produces results of decimal nature due to division and exponentiation operators. These values are approximations of the initial data set and are mapped to the discrete integer space. Therefore, we round off the output of the inverse DFT formula to generate the final approximated set. If the results from the inverse DFT deviate from the correct values by no more than 0.5, no information loss is accrued due to the round off and we can completely reproduce the initial attribute set. The amount of deviation, or error, can be controlled by *tuning* the number of coefficients used to represent the initial data set. For the remainder of this section we concentrate our discussion on how to address this trade-off and provide formulae for estimating a value for κ which maximizes compression for a given range of allowable error.

The above is not a straight-forward task as the allowable error in the coefficients is dependent on the data distribution. If the data distribution P is known a-priori, it is possible to use an estimation model to derive the correct *compression factor* necessary to provide an upper bound on the estimation error. Below, $\hat{x}$ is the estimate of the joining attributes reconstructed from the W/κ DFT coefficients, using the inverse DFT formula:

$$\hat{x}[n] = \frac{\kappa}{W} \sum_{k=1}^{W} \beta_{\frac{W}{\kappa}}(k) \hat{X}(k) e^{\frac{2\pi j}{W} kn} \tag{10}$$

where $\hat{X}$ is the DFT of the remote stream, approximated from W/κ coefficient, and $\beta_{\frac{W}{\kappa}}(k)$ is a binary function that equals to 1 for $k < \frac{W}{\kappa}$ and to 0 otherwise. The *mean square error* (*MSE*) of $\hat{x}[n]$ is

$$MSE = E[(x[n] - \hat{x}[n])^2] = \sum_{n} (x[n] - \hat{x}[n])^2 P(n) \tag{11}$$

Substituting x and $\hat{x}$ with their corresponding DFTs and combining the terms, Equation 11 becomes:

$$MSE = \sum_{n} \left[\frac{1}{W} \sum_{k=1}^{W} \left(X(k) - \kappa \beta_{\frac{W}{\kappa}}(k) \hat{X}(k) \right) e^{\frac{2\pi j}{W} kn} \right]^2 P(n) \tag{12}$$

where the difference inside the square brackets represents the residual $W(1 - \frac{1}{\kappa})$ DFT coefficients.

In Figure 5 we show the *absolute values* (i.e. not percentage) of the square errors for each join attribute value reconstructed from one of our sample stock data streams from $W/1024$, $W/256$, and $W/64$ coefficients. Since we represent the stream attribute values by a set of integers, it suffices to err in our DFT reconstruction by less than 0.5. This means that the expected value of the mean square error must be less than 0.25 (i.e. $E[MSE] < 0.25$) in order to *losslessly* reproduce the original values [24]. From Figure 5, we can see that when we use $1/256$'th of the original dataset's DFT coefficients we introduce marginal loss

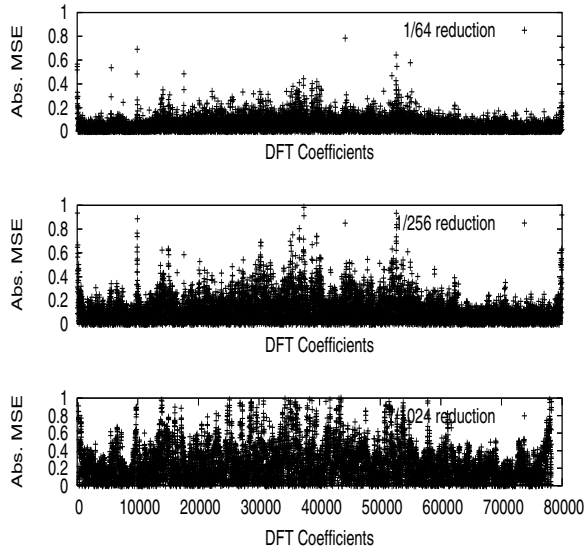


Figure 5. Absolute values of mean-square errors due to DFT compression of sample stock data stream with $W \approx 80000$.

of information. In Figure 6 we plot the mean and one standard deviation (y-error bars) of the mean square error for a range of compression factors. The horizontal line is drawn

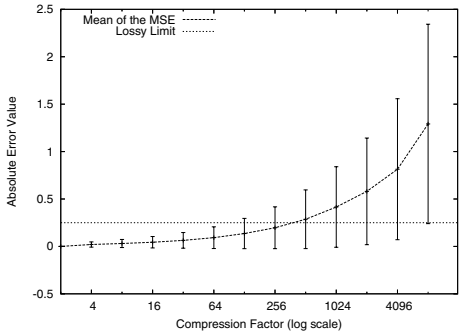


Figure 6. Mean-square errors due to DFT compression with one standard deviation y-error bars.

at $y = 0.25$. For $\kappa = 256$ the expected mean square error is below this line with the y-error bar extending to a 20% probability for errors. This corresponds to Figure 5 (middle) where 80% of the *MSE*s are below 0.25. In practice, this means that the transmittal of $\frac{1{,}000{,}000}{256} = 3,906$ coefficients will allow us to accurately estimate 800,000 of the original join attribute values. We believe that this trade-off justifies the use of DFTs for remote join attribute value reconstruction.

Our modified algorithm, termed DFTT, not only extracts a correlation coefficient for the two streams, but also recreates an approximation of the remote set of joining attributes. This set is used to test the local window for joins. Matching tuples must still be transmitted over the network in order to provide the complete result but this final step is necessary in any distributed system. The steps of the algorithm are depicted in Figure 7. First, we incrementally calculate the

```
PROCESSTUPLE(TUPLE τ, DFT myDFT,
    DFT DFT_{0..N-1}, FLOWFACTORS f_{0..N-1})
 1: newDFT ← IncrementalUpdate(myDFT, τ)
 2: newCoef ← ExtractCoeff(myDFT, newDFT)
 3: myDFT ← newDFT
 4: for i = 0 to N − 1 do
 5:    PiggyBackDFTChanges(i, newCoef)
 6:    joinAttributes_i ← InverseDFT(DFT_i)
 7:    if JoinEstimate(τ, joinAttributes_i) then
 8:       SendTuple(τ, N_i)
 9:    else if ChooseSite(newDFT, DFT_i, f_i) then
10:       SendTuple(τ, N_i)
```

Figure 7. Tuple processing DFTT algorithm.

new DFT coefficients and extract those that have changed since the last calculation (lines 1 and 2). These changes are piggy-backed onto tuple messages transmitted to each site in the system (line 5). If a tuple message was not sent to some site for a long period, the batch of updates are transmitted on their own. This period is dynamically determined as a multiple of the current rate of tuple arrivals. Lines 6-8 distinguish the DFTT algorithm from the DFT algorithm. The inverse DFT in line 6 is obtained from a lookup table of the current inverse DFTs of each site. This lookup table is updated as new coefficients arrive from other sites. The JoinEstimate() function in line 7 is responsible for allowing tuples to be forwarded only to those sites which produce the largest number of estimated joins as inferred from the inverse DFTs. The core of the probabilistic calculation based on the current set of weighted correlation coefficients is performed in line 9 with the function ChooseSite(), as per Equation 4. Thus, lines 9 and 10 are common to both the DFT and DFTT algorithm. For brevity, the algorithm does not display the case when a low variance in the flow factors $f_{0..N-1}$ is detected, which would trigger alternate distribution policies to handle this worst-case scenario.

6 Experiments

To evaluate empirically the performance of the proposed algorithms, we developed a working prototype of a DSP system in C++. We simulated a wide-area network (WAN) environment on a cluster of twenty Sun workstations interconnected through a 100 Mbps Ethernet network. To

simulate geographically distributed nodes, a latency of 20-100 milli-seconds was artificially imposed on each message transmitted, and a 90 kbps bandwidth was emulated by pausing every 90 kilobits transmitted for one second. Each workstation runs on a 400MHz UltraSPARC CPU with 128MB of main memory.

We explore two variations of our algorithm: DFT only performs flow filtering based on the DFT correlation coefficients, while DFTT also reconstructs tuples from the compressed coefficients. We compare the performance of these algorithms to those using different summarization techniques (BLOOM, SKCH) as detailed below and to the naive approach (BASE), which gives complete results. As a main indication for scalability and performance, we focus on three metrics: ϵ-*error* = percentage of tuples not reported in the join result; *messages per result tuple* = the total number of messages transmitted for each result tuple; *throughput* = number of tuples processed per second. The transmission of DFT coefficients is implicitly accounted for since the coefficients are piggy-backed onto regular messages. In size, the overhead of DFT coefficients constituted between 1.38 and 2.84% of the net data transmitted. Figure 8 shows how this ratio changed with the number of nodes in the system while running the DFT algorithm with Zipfian data distribution. The relative size of the DFT coefficient

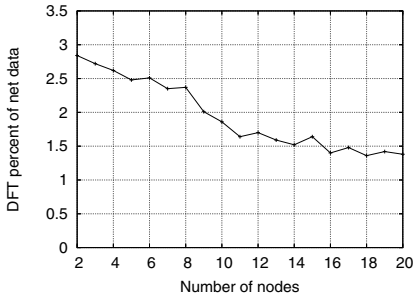


Figure 8. DFT coefficient updates as a percentage of the size of the net data transmitted ($\kappa = 256$).

updates compared to the net amount of data transmitted decreases with increasing number of nodes, we postulate that this overhead does not affect the scalability of the system.

We compare the performance of our algorithm with two algorithms which use set membership approximation (BLOOM) [5] and join-size approximation (SKCH) [1]. In BLOOM, a counting Bloom filter is constructed at each site. The Bloom filter is transmitted to remote sites where arriving tuples are tested for membership against the filter. The flow factors are determined from the number of positive filter hits that tuples generate. Another approach is to use randomized sketches to estimate the size of the joins between

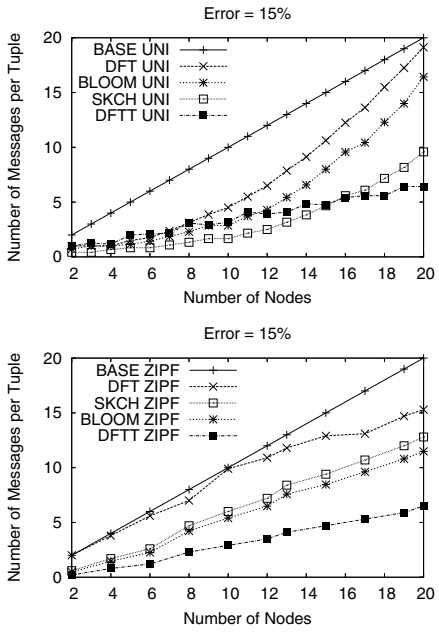


Figure 9. Number of messages per tuple with uniform (top) and Zipfian (bottom) data distribution.

different stream partitions. We use this estimate to weight the flow factors so that a tuple is more likely to be transmitted to those nodes which produce the most join results. We use the Sketches Library [20] for implementation of counting Bloom filters and AGMS sketches. For FFT calculations, we used a modified version of the FFTW library [15]. For all experiments, we adjust the size of the Bloom filters, sketches and DFT coefficients to be the same. For sketches, we preserve a 5:1 ratio between s_0 and s_1, for the chosen value of the sketch size s.

We used two types of data, synthetically generated and real-world data. The synthetic data consisted of 10,000,000 integers generated in the range $[1 : 2^{19}]$ according to two distributions: (1) UNI - uniform distribution, and (2) ZIPF - Zipfian distribution with parameter $\alpha = 0.4$. The two real-world data sets consist of (a) 1,800,000 traces of financial data including buy/sell trades (FIN) and (b) 2,200,000 network packet traces (NWRK) accumulated in a day [2]. Unless otherwise noted, for brevity, we display only the results of the real-data workloads, which were very similar to the Zipfian-distribution synthetic data sets with $\alpha = 0.4$.

In Figure 9 we show the number of messages transmitted per matching join tuple in the entire system with fixed error rate $\epsilon = 15\%$. This plot reveals how efficient each of the algorithms is. Under uniformly distributed data, all

[2] http://cs-people.bu.edu/jching/icde06/

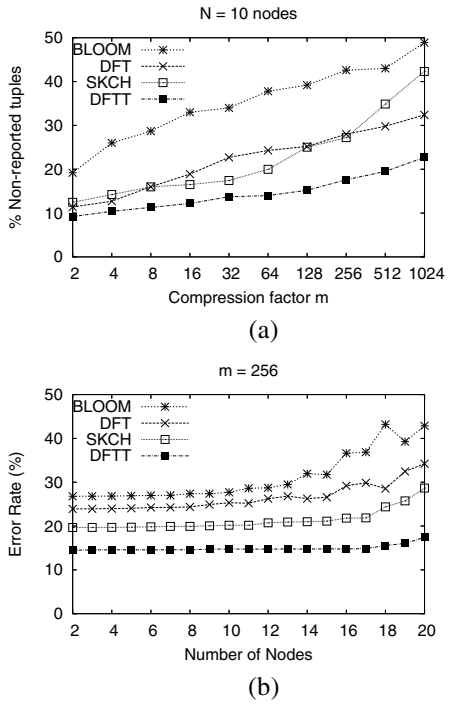


Figure 10. Evaluation of compression factor, error rate and throughput under Zipfian data distribution.

algorithms perform similarly, which agrees with our analysis. The obvious discrepancies occur when the data is skewed. We can see that, due to DFTT's ability to determine locally whether remote tuples join, much fewer messages are transmitted per tuple. DFT lacks this ability and performs worse than BLOOM and SKCH. Similarly, BLOOM has the advantage of testing tuples for membership in the remote stream locally and, therefore, incurs less messages than SKCH. DFTT is 1.6 to 2 times more efficient in tuple transmittal than any of its counterparts.

Next we analyze how the system performs under different compression factors κ. For the purpose, we fixed the window size at $W = 524,288 = 2^{19}$ and varied κ from 2 to 1024, resulting in summaries of sizes in the range $[512 : 262144]$. We can see in Figure 10 (a) that DFTT scales the best, while BLOOM performs the worst. SKCH performs similarly to DFTT but its error rate increases exponentially for smaller sketch sizes. Overall, with DFTT it is possible to obtain consistent error rates of 15% with only 4,096 DFT coefficients, which is 1.6 to 2.6 times better than any of the other algorithms.

To account for scalability in terms of expanding number of nodes, we fixed the compression factor at 256 and monitored the error rate as we increased the number of nodes

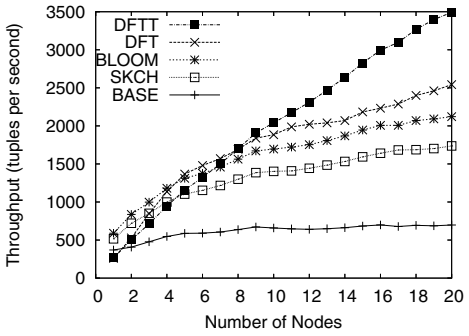


Figure 11. Throughput performance

from 2 to 20. The results are shown in Figure 10 (b). All four algorithms performed well for fewer than 15 nodes, with DFTT maintaining a 15% error as compared to 20-28% for the other algorithms. We can further see from Figure 10 (b) that beyond 15 nodes, again DFTT's rate of error increase was the slowest, performing 1.6 to 2.4 times better than the other algorithms.

We measured throughput as the number of joining tuples reported per second with ϵ fixed at 15%. The results are shown in Figure 11. With the efficiency of the DFTT algorithm, it is not surprising that it provides the best throughput of up to 3500 tuples per second. With the other algorithms, the large number of messages necessary to maintain this error rate contend for bandwidth with the actual result tuple messages. In the case of the BASE algorithm, the $(N-1)$ message complexity renders it ineffective in such an environment where scalability is important. It is worth indicating that only the DFTT algorithm is able to continuously sustain low error rates, while exhibiting throughput characteristics which outperform all other algorithms. We postulate that this is due to the provision for both mathematical stream correlation expression and set representation inherent in the discrete Fourier transform. Overall, our DFTT algorithm is able to maintain the lowest error rates while transmitting the least number of messages. This makes it a viable tool for performing approximate joins in a distributed system.

7 Related Work

Computing aggregates on a number of distributed streams using the idea of sliding windows appeared in [17]. A centralized method for approximate set cardinality estimation of distributed streams is proposed in [10]. Algorithms for *top-k* monitoring of streams that are transported data volume conscious in a distributed environment are examined in [2]. A distributed approach for computing holistic aggregates is discussed in [9]. Extensions for support of join and multi-join aggregates (COUNT) are presented in [8]. The use of randomized sketches in [8] is similar to

our approach in the sense that, just like DFT coefficients, sketches provide concise frequency representation of the data set. However, the sketches in [8] are transmitted to a central site responsible for tracking the aggregate queries. In [18], a rudimentary approach is taken by partitioning one of two streams among the processing nodes, whereas the other stream is replicated to all nodes. Previously, approaches based on precomputed relation synopses have only been explored in the context of OLAP queries [16]. Our approach has two main characteristics which differentiate it from previous research: it is completely distributed and does not rely on any central coordinators, and it is based on inexpensive incremental DFT calculations.

8 Conclusions

We propose an approach which exploits the combination of correlation characteristics of distributed streams and compressed tuple-attribute representation to provide best-effort error estimates under a pool of geographically distributed nodes. Our approach is based on discrete Fourier transforms and provides a clean trade-off for message complexity versus approximation error in the size of the output, while scaling linearly with the addition of resources to the cluster. We verify our hypotheses by experimenting with a C++-based prototype system functioning in a cluster of twenty SUN-workstations.

Acknowledgments: we thank the anonymous reviewers and Ricardo Jimenez-Peris for their valuable comments.

References

[1] N. Alon, P. Gibbons, Y. Matias, and M. Szegedy. Tracking Join and Self-Join Sizes in Limited Storage. In *PODS*, pages 10–20, 1999.

[2] B. Babcock and C. Olston. Distributed top-k monitoring. In *SIGMOD*, pages 28–39, 2003.

[3] S. Babu, R. Motwani, K. Munagala, I. Nishizawa, and J. Widom. Adaptive Ordering of Pipelined Stream Filters. In *SIGMOD Conference*, pages 407–418, 2004.

[4] D. H. Bailey and P. N. Swarztrauber. A Fast Method for the Numerical Evaluation of Continuous Fourier and Laplace Transforms. *SIAM J. on Scientific Computing*, 15(5):1105–1110, 1994.

[5] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey, 2002.

[6] J. Considine, F. Li, G. Kollios, and J. W. Byers. Approximate Aggregation Techniques for Sensor Databases. In *ICDE*, pages 449–460, 2004.

[7] J. W. Cooley and J. W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Math. Comput.*, 19:297–301, 1965.

[8] G. Cormode and M. N. Garofalakis. Sketching Streams Through the Net: Distributed Approximate Query Tracking. In *VLDB*, pages 13–24, 2005.

[9] G. Cormode, M. N. Garofalakis, S. Muthukrishnan, and R. Rastogi. Holistic Aggregates in a Networked World: Distributed Tracking of Approximate Quantiles. In *SIGMOD Conference*, pages 25–36, 2005.

[10] A. Das, S. Ganguly, M. N. Garofalakis, and R. Rastogi. Distributed Set Expression Cardinality Estimation. In *VLDB*, pages 312–323, 2004.

[11] A. Das, J. Gehrke, and M. Riedewald. Approximate Join Processing Over Data Streams. In *SIGMOD Conference*, pages 40–51, 2003.

[12] L. Ding, E. A. Rundensteiner, and G. T. Heineman. MJoin: a Metadata-Aware Stream Join Operator. In *DEBS*, 2003.

[13] C. Faloutsos, Y. Matias, and A. Silberschatz. Modeling Skewed Distribution Using Multifractals and the '80-20' Law. In *VLDB'96*, pages 307–317.

[14] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast Subsequence Matching in Time-Series Databases. In *SIGMOD Conference*, pages 419–429, 1994.

[15] M. Frigo and S. G. Johnson. FFTW v3.1. http://www.fftw.org.

[16] V. Ganti, M. Lee, and R. Ramakrishnan. ICICLES: Self-Tuning Samples for Approximate Query Answering. In *VLDB*, pages 176–187, 2000.

[17] P. B. Gibbons and S. Tirthapura. Distributed Streams Algorithms for Sliding Windows. In *SPAA*, pages 63–72, 2002.

[18] X. Gu and P. S. Yu. Adaptive Load Diffusion for Stream Joins. In *ACM/IFIP/USENIX 6th International Middleware Conference*, 2005.

[19] W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. Power Spectrum Estimation Using the FFT. In *Numerical Recipes in FORTRAN, 2nd ed.*, pages 542–551, 1992.

[20] M. Hadjieleftheriou. Sketches Library v0.16b. http://u-foria.org/marioh/sketches/index.html.

[21] M. A. Hammad, W. G. Aref, and A. K. Elmagarmid. Stream Window Join: Tracking Moving Objects in Sensor-Network Databases. In *SSDBM*, pages 75–84, 2003.

[22] J. Kang, J. F. Naughton, and S. Viglas. Evaluating Window Joins over Unbounded Streams. In *ICDE*, pages 341–352, 2003.

[23] M. T. Ozsu and P. Valduriez. Principles of Distributed Database Systems, Second Edition. Prentice-Hall, 1999.

[24] A. Papoulis and S. U. Pillai. Probability, Random Variables and Stochastic Processes . McGraw Hill, 2002.

[25] K. Shanmugasundaram, H. Brönnimann, and N. D. Memon. Integrating Digital Forensics in Network Infrastructures. In *IFIP Int. Conf. Digital Forensics*, pages 127–140, 2005.

[26] M. A. Sharaf, J. Beaver, A. Labrinidis, and P. K. Chrysanthis. Balancing Energy Efficiency and Quality of Aggregate Data in Sensor Networks. *VLDB J.*, 13(4):384–403, 2004.

[27] U. Srivastava and J. Widom. Memory-Limited Execution of Windowed Stream Joins. In *VLDB*, pages 324–335, 2004.

[28] J. Winograd and S. Nawab. Probabilistic Complexity Analysis for a Class of Approximate DFT Algorithms. *The Journal of VLSI Signal Processing*, 14(2):193–205, 1996.

[29] Y.-L. Wu, D. Agrawal, and A. E. Abbadi. A Comparison of DFT and DWT based Similarity Search in Time-Series Databases. In *CIKM*, pages 488–495, 2000.

[30] J. Xie, J. Yang, and Y. Chen. On Joining and Caching Stochastic Streams. In *SIGMOD Conference*, pages 359–370, 2005.