

Approximate Clustering on Distributed Data Streams

Qi Zhang ^{#1}, Jinze Liu ^{*2}, Wei Wang ^{#3}

[#]*Department of Computer Science, University of North Carolina, Chapel Hill
Chapel Hill, NC 27599-3175, USA*

¹zhangq@cs.unc.edu

³weiwang@cs.unc.edu

^{*}*Department of Computer Science, University of Kentucky
Lexington, KY 40506-0046, USA*

²liuj@netlab.uky.edu

Abstract—We investigate the problem of clustering on distributed data streams. In particular, we consider the k-median clustering on stream data arriving at distributed sites which communicate through a routing tree. Distributed clustering on high speed data streams is a challenging task due to limited communication capacity, storage space, and computing power at each site. In this paper, we propose a suite of algorithms for computing $(1 + \epsilon)$ -approximate k-median clustering over distributed data streams under three different topology settings: topology-oblivious, height-aware, and path-aware. Our algorithms reduce the maximum per node transmission to *polylog* N (opposed to $\Omega(N)$ for transmitting the raw data). We have simulated our algorithms on a distributed stream system with both real and synthetic datasets composed of millions of data. In practice, our algorithms are able to reduce the data transmission to a small fraction of the original data. Moreover, our results indicate that the algorithms are scalable with respect to the data volume, approximation factor, and the number of sites.

I. INTRODUCTION

Distributed data stream applications perform continuous, on-the-fly computations over geographically dispersed data streams, such as sensor-based applications, traffic management, network monitoring, location-tracking services, etc. Various tasks of continuous query and monitoring in distributed setting have been developed, including database queries [8], [6], [21], [10], monitoring simple statistics [7], [18], [22], event detection [2], [3], [25], etc. In this paper, we address the problem of clustering over distributed data streams. Particularly, we consider the problem of performing k-median clustering continuously in a distributed stream environment.

Consider a set of distributed sites, which push the data towards the base site periodically. We refer to each such period as an *update epoch*. Assume that all the sites communicate according to a routing tree rooted at the base site. The goal is to perform clustering on the data collected from all the sites. The clustering result is continuously updated at the root for each update epoch. Here we assume that the epoch is long enough so that all the data of one epoch is able to arrive at the root before next epoch ends.

Clustering over distributed streams is a challenging task. Difficulties lie in various issues: 1) *Communication*. Dis-

tributed stream system continuously produces large volume of data, which imposes prohibitive communication load if all the data are transferred to the root for centralized computation. In-network aggregation [19] is one of the techniques [21], [19], [24], [23] that push processing operators down into the network to reduce data transmission. It computes a local summary at each site and merges and summarizes further at each internal site towards the root. However, this approach cannot be immediately adopted to solve the problem of k-median clustering. The k-median clustering is known as a holistic computation [19], which cannot be readily decomposed into computations on data partitions. More specifically, the k-median clustering of the entire dataset cannot be accurately computed from the k-median centers of individual partitions. In order to get the exact answer, all the data need to be transmitted to a central site before the k-median clustering is performed. 2) *Latency*. Another approach to solve the k-median problem is to treat it as an iterative optimization problem. For each iteration, statistics [11] are computed and transmitted back and forth between the root and each site. The induced latency is unbearable for stream-based applications. 3) *Clustering quality*. Clustering quality is another concern if we want to trade accuracy for reduced communication. In this case, it is necessary that the error of the k-median solution is bounded. The bounded approximate in-network aggregation also raises new issues such as error propagation and topology sensitivity.

In this paper, we consider approximate in-network aggregation schemes for $(1 + \epsilon)$ -approximate k-median clustering over distributed data streams. We propose a suite of algorithms for both topology-insensitive and network-aware cases. Our contributions are as follows:

- For topology-insensitive case, we employ a multi-level structure of ϵ -kernel of k-median (ϵ -coreset [16]) as the local summary, and propose algorithms for constructing and merging local summaries. We prove that the size of the multi-level summary is a *polylog* function of the data volume and it guarantees a $(1+\epsilon)$ -approximate clustering.

- For network-aware cases, we propose height-aware and path-aware algorithms that utilize the topology information to achieve aggressive reduction of data communication.
- We perform experiments on both synthetic and real data sets to demonstrate the performance of the algorithms in terms of communication reduction, clustering quality and scalability.

The rest part of the paper is organized as follows. Section II reviews the related work. Section III introduces the preliminaries and background. Section VI discusses the detailed algorithms for topology-oblivious, height-aware, and path-aware cases. In Section V, we report the experimental results. Section VI concludes the paper.

II. RELATED WORK

In this section, we briefly review the related work in distributed clustering, approximate in-network aggregation, and stream clustering. We also review the notion of coresets, which is the ε -kernel for k-median clustering.

A. Distributed Clustering

Distributed clustering needs to address the problem of balancing between communication and precision. Forman and Zhang [11] proposed a technique to exactly compute several iterative center-based data clustering algorithms including k-means for distributed database applications. It sends sufficient statistics instead of the raw data to a central site. However, this approach involves each site in every iteration of k-means and transfers data back and forth, which is infeasible for stream applications. Januzaj *et al.* [17] proposed a distributed density-based clustering algorithm. The algorithm clusters the data locally at each site and computes the aggregation (representatives for local clusters) for the local site. All the aggregations are sent to the global server site where the global clustering is carried out. Their algorithm considers the flat two-tier topology instead of tree topology, and it does not provide a bound of the clustering quality.

B. Approximate In-network Aggregation

Bounded-error approximation is a desired property to have when trading accuracy for communication requirement. Cosdine *et al.* [9] proposed an approximate in-network aggregation scheme for sensor databases. They provided an algorithm for approximate duplicate-sensitive aggregates across distributed datasets, such as SUM. Their algorithm employs small duplicate-insensitive sketches for SUM which is generalized from similar technique for approximating COUNT. Greenwald and Khanna [14] proposed an algorithm for power-preserving computation of order statistics such as quantile. They proposed a scheme for computing ε -approximate quantile over the sensor network routing tree, where each sensor transmits only $O(\log^2 n/\varepsilon)$ data points opposed to the worst case of $\Omega(n)$.

C. Clustering Single Stream

A lot of work has been reported for clustering over a single stream [4], [5], [20], [16], [12]. The proposed algorithms perform continuous online clustering with small-space requirement. Guha *et al.* [12] proposed a constant factor approximation algorithms for k-median clustering on a single data stream. Their algorithm requires $O(n^\varepsilon)$ space with an approximation factor of $2^{O(\frac{1}{\varepsilon})}$. Aggarwal *et al.* [4] proposed an algorithm which considers both online statistical data collection and offline analysis, compared with the one-pass clustering algorithms.

D. Coreset and Streaming k-median

A coreset is a small subset of points that approximates the original set with respect to some tasks (such as k-median, k-means, etc) [15]. Several coreset construction algorithms have been proposed for k-median and k-means clustering [15], [16], [13]. In [13], a coreset construction algorithm for streaming $(1 + \varepsilon)$ -approximate k-median and k-means is proposed. The coreset is computed using a QuadTree, and its space requirement is *polylog* in size of the data stream and the range of the data values. Har-Peled and Mazumdar [16] proposed another coreset construction algorithm for $(1 + \varepsilon)$ -approximate k-median and k-means, where the coreset takes *polylog* space. These two coreset-based streaming k-median algorithms achieve a better space-bound than Guha's algorithm [12] mentioned earlier. All of these algorithms consider the clustering over a single stream. In our paper, we study the stream clustering problem in a distributed setting.

III. PRELIMINARIES AND BACKGROUND

A. Problem Definition

Let $d(p, c)$ denote the Euclidean distance between any two points p and c . The goal of k-median clustering is to find a set C of k points representing the k cluster centers, which minimize the k-median cost of dataset P . Here the k-median cost is defined as $Cost(P, C) = \sum_{p \in P} d(p, C)$, where $d(p, C) = \min_{c \in C} d(p, c)$. If P is a weighted data set with $p_i \in P$ of weight w_i , then the weighted k-median of P is defined as the cluster center set C which minimizes $Cost(P, C) = \sum_{p_i \in P} w_i d(p_i, C)$.

Definition 3.1: $(1 + \varepsilon)$ -APPROXIMATE K-MEDIAN OF DATA SET P . Let C denote the k-median center set. Then C' is defined as a $(1 + \varepsilon)$ -approximate k-median center set if $Cost(P, C') \leq (1 + \varepsilon)Cost(P, C)$.

Consider a set of sites $\{S_i\}$ which communicate according to a routing tree¹. Suppose that during each update epoch e , site S_i receives a stream of data $P_i|_e$. There are two problems we consider: (1) how to compute the $(1 + \varepsilon)$ -approximate k-median on the set of data $\bigcup_i (P_i|_{e^*})$, where e^* is the latest update epoch; and (2) how to compute the $(1 + \varepsilon)$ -approximate k-median on all data received in past epochs. We shall see later that the solutions to both problems are similar except that

¹Each site is a node in the routing tree. In the following discussion, we will use the terms *site* and *node* interchangeably.

the root may perform some additional operations to solve the second problem. Therefore, we will focus on the first problem and discuss how the proposed algorithms can be modified to solve the second problem.

We define the first problem as below:

Definition 3.2: $(1 + \varepsilon)$ -APPROXIMATE K-MEDIAN OVER DISTRIBUTED STREAM SET $\{S_i\}$ DURING EPOCH e . The $(1 + \varepsilon)$ -approximate clustering of distributed stream set $\{S_i\}$ during epoch e is defined as the $(1 + \varepsilon)$ -approximate k-median over all the data received at all the sites during epoch e , which is $\bigcup_i (P_i|_e)$.

In the following discussion, we will simply use P_i or P to represent streams, assuming that we consider the streams during one epoch.

B. Local Summary Structure

Our algorithms employ approximate in-network aggregation schemes. Basically, each site computes a local summary and sends it to its parent. For internal nodes, the local summary is merged with the summaries received from children and another local summary operation is performed on the merged set before the summary is sent to the parent. When all the summaries reach the root, a k-median clustering procedure is performed at the root to get the $(1 + \varepsilon)$ -approximate clustering of the whole data.

An important element of the algorithm is the summary structure. A desirable summary structure SM should satisfy the following properties:

- **Property I: distributive (or decomposable):** The summary of a set P can be computed from the summaries of its partitions. For example, $SM(P) = f(SM(P_1), SM(P_2))$, where $P = P_1 \cup P_2$ and f is the function to combine the partition summaries.
- **Property II: compact:** the summary should have a much smaller size than the original data.
- **Property III: error bound:** the summary can deliver a k-median solution with bounded error, where the k-median solution on the summary guarantees a $(1 + \varepsilon)$ -approximate k-median solution on the original data. Here ε is the approximation factor associated with the summary.
- **Property IV: error accumulation:** the summary of the summary of data set P should still be a summary of P , only with a looser approximation factor due to error accumulation. Formally, $SM_{\varepsilon_1}(SM_{\varepsilon_2}(P)) = SM_{g(\varepsilon_1, \varepsilon_2)}(P)$, where $g(\varepsilon_1, \varepsilon_2)$ is a function to accumulate the error which satisfies $g(\varepsilon_1, \varepsilon_2) \geq \varepsilon_1$ and $g(\varepsilon_1, \varepsilon_2) \geq \varepsilon_2$.

The summary structure and the algorithms for constructing and merging this type of summaries are presented in Section VI.

C. Topology Dependence

The amount of accumulated error of in-network aggregation is determined by the number of merges, compressions and propagations of the local summaries occurring at each node in the network. Therefore, the approximation factor of the

summary is highly dependent on the network topology. In this paper, we provide three algorithms that utilize different topology knowledge for optimizing the data transmission.

- **Topology-oblivious algorithm** We propose a topology-oblivious algorithm without any prior knowledge of the tree topologies. Our algorithm computes a local summary structure of size $O(\frac{k}{\varepsilon} \log^3 N \sqrt{\log N})$ which maintains the same error bound at each node. In this topology-oblivious algorithm, only the merging operation performs at the internal node.
- **Height-aware algorithm** In this case, we assume that the height of tree is known. We present an improved algorithm which reduces communication compared with the topology-oblivious algorithm.
- **Path-aware algorithm** The height of the subtrees in a tree topology may vary significantly. Given the same approximation bound, the communication load at each node uniformly determined by the height of the entire tree may not be optimal. The path-aware algorithm adaptively computes the communication load for each node according to the height of its subtree. This approach minimizes the communication per node while still ensuring the same overall additive error bound. The algorithm is well-suited for reducing communication for unbalanced tree topologies.

IV. ALGORITHM

In this section, we describe in detail the algorithms for computing $(1 + \varepsilon)$ -approximate k-median over the distributed stream considering the different cases of topology dependency. We consider that sensors are organized into a routing tree. Specifically, we explain the design of local summary structure at each node, the construction and merging algorithms for summaries, and the error propagation schemes for the network-aware cases. In addition, we formally prove the error bound for the clustering result and the bound for max per node transmission.

A. Topology-oblivious Algorithm

This algorithm is designed for the scenario where the tree structure is unknown, *i.e.*, the size and the height of the routing tree is unknown. The algorithm computes $(1 + \varepsilon)$ -approximate k-median clustering with reduced maximum per-node transmission of $O(\frac{k}{\varepsilon} \log^3 N \sqrt{\log N})$.

The algorithm follows the in-network aggregation scheme. The key component is the construction and merging of summary structure.

1) *Local Summary at Each Node:* We use a level-wise structure of k-median's ε -coreset as the local summary at each site.

The ε -coreset is an ε -kernel defined on a point set P for certain geometric problems [1]. For k-median problem, the ε -coreset C is defined as follows:

Definition 4.1: ε -CORESET OF DATASET P . Let P be a weighted set of n points. A weighted point set C is an ε -coreset for the k-median problem, if for every set C_{tr} of k

centers:

$$(1 - \varepsilon)Cost(P, Ctr) \leq Cost(C, Ctr) \leq (1 + \varepsilon)Cost(P, Ctr)$$

Here C is usually a much smaller set than P , and each point $p \in P$ is uniquely represented by a point $c \in C$, where c 's weight is defined as the sum of the weights of the points in P it represents.

The ε -coreset is a good candidate for the local summary since it satisfies all requirements of the summary: 1) coreset is distributive. If C_1 and C_2 are the ε -coresets for two disjoint sets of points P_1 and P_2 respectively, then $C_1 \cup C_2$ is an ε -coreset for $P_1 \cup P_2$. 2) coreset only has size $O(\frac{k}{\varepsilon} \log N)$, where N is the size of P . 3) A $(1 + \varepsilon)$ -approximate k -median solution of P can be obtained by computing the exact k -median solution on its ε -coreset. 4) The ε_1 -coreset of the ε_2 -coreset of P is an $(\varepsilon_1 + \varepsilon_2)$ -coreset of P , which means

$$Coreset_{\varepsilon_1}(Coreset_{\varepsilon_2}(P)) = Coreset_{\varepsilon_1 + \varepsilon_2}(P)$$

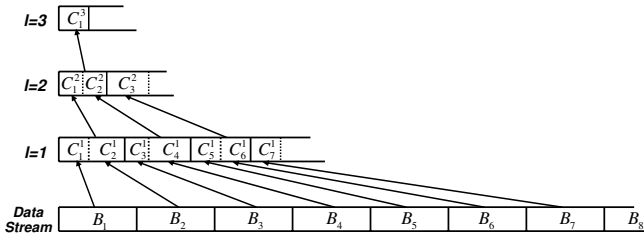


Fig. 1. **EH summary at a site:** This figure highlights the multi-level structure of our EH-summary. The incoming data is buffered in equi-sized blocks $B_1, B_2, \dots, B_j, \dots$, each of size $O(\frac{k}{\varepsilon})$. We compute the coreset C_j^1 for each block B_j and send it to level $l = 1$. At each level $l > 0$, whenever two coresets C_j^l, C_{j+1}^l come in, we merge and compute another coreset $C_{\lceil j/2 \rceil}^{l+1}$ on $C_j^l \cup C_{j+1}^l$ and send to level $l + 1$. There are at most $\log \frac{N}{k/\varepsilon}$ levels.

EH-Summary

At each site, we create an *EH*-summary which is a multi-level structure of coresets over the stream. *EH*-summary is constructed in the similar fashion to that of building an online exponential histogram. Assume that the stream arriving at a site is P . Whenever a block of B data points comes, we perform a coreset computation and append the points in the coreset to the stream one level above. Fig.1 illustrates the process of the algorithm. B_j represents the j^{th} block of the original stream data P . Whenever block B_j is filled up and the coreset C_j^1 of B_j is computed, C_j^1 is added to the stream at level $l = 1$. For all levels $l \geq 1$, whenever two coresets C_j^l, C_{j+1}^l come in, we merge them and compute another coreset $C_{\lceil j/2 \rceil}^{l+1}$ on top of the merged set and send to level $l + 1$. This process propagates until at level L , where there is only one coreset. Here we take block size $B = O(\frac{k}{\varepsilon})$. Let P_l denote the stream at level l , P_Set denote the set of streams at all levels. The algorithm is shown in Algorithm 1:

The algorithm generates a set of streams at different levels $P_Set = \{P_l\}, l = 0, 1, \dots, L$, where P_0 is the original data stream P . The computation propagates from the lowest level P_0 , towards higher levels $P_l, l > 0$ until at level L , there are

Algorithm 1 CompEHSummary(P, ε)

Input P : the original data stream; ε : the required error bound

- 1: $P_0 \leftarrow P, P_Set \leftarrow \{P_0\}$
- 2: Arrange P_0 into blocks B_1, \dots, B_j, \dots of equal size B
- 3: $P_1 \leftarrow \Phi$
- 4: Compute $\Delta\varepsilon_1$ -coreset C_j^1 on each block B_j , add C_j^1 into P_1
- 5: $P_Set = P_Set \cup \{P_1\}$
- 6: $l = 1$
- 7: **while** P_l contains more than one coreset, $P_l = \{C_j^l\}, j = 1, \dots, J_l, J_l > 1$ **do**
- 8: **if** $\neg(P_{l+1} \in P_Set)$ **then**
- 9: $P_{l+1} \leftarrow \Phi, P_Set = P_Set \cup \{P_{l+1}\}$
- 10: **end if**
- 11: Merge C_j^l and C_{j+1}^l , where $j = 1, 3, \dots$, and $j < J_l$
- 12: Compute $\Delta\varepsilon_{l+1}$ -coreset $C_{\lceil j/2 \rceil}^{l+1}$ on each $C_j^l \cup C_{j+1}^l$, and add $C_{\lceil j/2 \rceil}^{l+1}$ into P_{l+1}
- 13: $l \leftarrow l + 1$
- 14: **end while**

less than two coresets. At any level l , we compute $\Delta\varepsilon_{l+1} = \frac{1}{(l+1)\sqrt{l+1}}\frac{\varepsilon}{3}$ -coresets and send them to level $l + 1$.

Note that in Fig.1, a coreset C_j^l at level l represents a sequence of 2^{l-1} consecutive blocks in the original stream $\{B_{(j-1)2^{l-1}+1}, \dots, B_{j2^{l-1}}\}$, with an approximation factor of $\sum_1^l \Delta\varepsilon_i$. For example, coreset C_1^3 at level 3 covers the blocks $\{B_1, B_2, B_3, B_4\}$, C_3^2 at level 2 covers the blocks $\{B_5, B_6\}$ in P . We denote the union of the original data blocks in P covered by C_j^l as $CoverInterval(C_j^l)$. The following lemma can be derived:

Lemma 4.1: Each C_j^l is an ε_l -coreset of $CoverInterval(C_j^l)$ where $\varepsilon_l = \varepsilon - \frac{2}{3\sqrt{l}}\varepsilon$. According to the distributive and error accumulation properties of coreset, C_j^l is an ε_l -coreset of $CoverInterval(C_j^l)$, where

$$\begin{aligned} \varepsilon_l &= \sum_1^l \Delta\varepsilon_i \\ &= \sum_1^l \frac{1}{i\sqrt{i}} \frac{\varepsilon}{3} \\ &\leq \frac{\varepsilon}{3} + \frac{\varepsilon}{3} \int_1^l \frac{dx}{x\sqrt{x}} \\ &= \varepsilon - \frac{2}{3\sqrt{l}}\varepsilon. \end{aligned} \tag{1}$$

Consider any coreset C_j^l at level l . If $C_{\lceil j/2 \rceil}^{l+1} \in P_{l+1}$, we have $CoverInterval(C_j^l) \subset CoverInterval(C_{\lceil j/2 \rceil}^{l+1})$, since $C_{\lceil j/2 \rceil}^{l+1}$ is a coreset computed on either $C_j^l \cup C_{j+1}^l$ or $C_{j-1}^l \cup C_j^l$. Formally, we claim that C_j^l is an *obsolete* coreset if $C_{\lceil j/2 \rceil}^{l+1} \in P_{l+1}$, and an *active* coreset otherwise. In Fig.1, for example, the only three active coresets are C_1^3, C_3^2 , and C_7^1 . All the remaining coresets are obsolete coresets. Note that there could be at most one active coreset at each level.

Therefore, we define the *EH*-summary as follows:

Definition 4.2: EH-SUMMARY. An *EH*-summary of a

stream P is the set of active coresets at all levels (generated by Algorithm 1). $EHSummary(P) = \{\tilde{C}^0, \tilde{C}^{l_1}, \dots, \tilde{C}^{l_k}, \dots, \tilde{C}^L\}$, where \tilde{C}^{l_k} is the active coreset at level l_k , \tilde{C}^0 is the newest block in the original stream which is not full, and L is the maximum level generated by Algorithm 1.

The EH -summary covers the whole data stream P . Each \tilde{C}^{l_k} in the EH -summary covers a disjoint subset of consecutive complete blocks in P , specifically, \tilde{C}^{l_k} is an $(\varepsilon - \frac{2}{3\sqrt{l_k}}\varepsilon)$ -coreset of $CoverInterval(\tilde{C}^{l_k})$ (Lemma 4.1). We have

$$CoverInterval(\tilde{C}^{l_{k_1}}) \cap CoverInterval(\tilde{C}^{l_{k_2}}) = \Phi,$$

and

$$\left(\bigcup_{l_k} CoverInterval(\tilde{C}^{l_k})\right) \cup \tilde{C}^0 = P.$$

In Fig.1, for example, the EH -summary is $\{B_8, C_7^1, C_3^2, C_1^3\}$. C_1^3 at level 3 covers the blocks B_1, B_2, B_3 , and B_4 . C_3^2 covers B_5 and B_6 . C_7^1 covers B_7 . Together with B_8 , they cover the whole data stream.

Lemma 4.2: The union of coresets $(\bigcup_{l_k} \tilde{C}^{l_k}) \cup \tilde{C}^0$ in EH -summary $EH = \{\tilde{C}^0, \tilde{C}^{l_1}, \dots, \tilde{C}^{l_k}, \dots, \tilde{C}^{l_K}\}$, $l_K = L$, is an $(\varepsilon - \frac{2}{3\sqrt{L}}\varepsilon)$ -coreset of P .

According to Lemma 4.1, each \tilde{C}^{l_k} is an $(\varepsilon - \frac{2}{3\sqrt{l_k}}\varepsilon)$ -coreset of $CoverInterval(\tilde{C}^{l_k})$. Since $\varepsilon - \frac{2}{3\sqrt{L}}\varepsilon = \max(\varepsilon - \frac{2}{3\sqrt{l_k}}\varepsilon)$, each \tilde{C}^{l_k} is also an $(\varepsilon - \frac{2}{3\sqrt{L}}\varepsilon)$ -coreset of $CoverInterval(\tilde{C}^{l_k})$. Therefore, according to distributive property, $(\bigcup_{l_k} \tilde{C}^{l_k}) \cup \tilde{C}^0$ is an $(\varepsilon - \frac{2}{3\sqrt{L}}\varepsilon)$ -coreset of $CoverInterval(\tilde{C}^{l_k}) \cup \tilde{C}^0 = P$.

2) *Merging EH-Summary at Intermediate Nodes:* For any internal node, we need to combine the local EH -summary with any EH -summary it receives from its children. Consider two EH -summaries $EH = \{\tilde{C}^0, \tilde{C}^{l_1}, \dots, \tilde{C}^{l_k}, \dots, \tilde{C}^L\}$, $EH_* = \{\tilde{C}_*^0, \tilde{C}_*^{l_1}, \dots, \tilde{C}_*^{l_k}, \dots, \tilde{C}_*^L\}$. Denote the combined EH -summary as EH_{all} . Intuitively, the combination of EH and EH_* can proceed as follows: 1) At level 0, combine \tilde{C}^0 and \tilde{C}_*^0 , denote it as \tilde{C}_{all}^0 . If $|\tilde{C}_{all}^0| > B$, arrange \tilde{C}_{all}^0 into blocks B_c and B_r , where B_c is a complete block of size B , and B_r is the remaining part. Compute an $\Delta\varepsilon_1$ -coreset $\Delta\tilde{C}_{all}^1$ over B_c , and send it to level 1. Set \tilde{C}_{all}^0 to be B_r and add it to EH_{all} . 2) For any level $l \geq 1$, start from level 1. a) If both EH and EH_* contain coresets on level l , merge the two coresets, compute another coreset $\Delta\tilde{C}_{all}^{l+1}$ on top of it, and send to level $l+1$. Additionally, if there is a non-empty $\Delta\tilde{C}_{all}^l$ sent by the level $l-1$, add $\Delta\tilde{C}_{all}^l$ into EH_{all} as the coreset at level l . b) If only one of EH and EH_* contains coreset on level l , assume it is EH with \tilde{C}^l . If there is a non-empty $\Delta\tilde{C}_{all}^l$ sent by the lower level, we compute another coreset $\Delta\tilde{C}_{all}^{l+1}$ on top of $\tilde{C}^l \cup \Delta\tilde{C}_{all}^l$ and send it to next level. Otherwise, add \tilde{C}^l into EH_{all} as the coreset at level l . c) Finally, if neither EH nor EH_* contains coreset on level l , and there is a non-empty $\Delta\tilde{C}_{all}^l$ sent by the lower level, we add $\Delta\tilde{C}_{all}^l$ into EH_{all} as the coreset at level l . The algorithm is shown in Algorithm 2.

3) *Compute k-Median at the Root:* At the root node, we have an EH -summary for all the data during the last epoch e , P_e . Let the EH -summary at the root be $EH_{root} = \{\tilde{C}^0, \tilde{C}^{l_1}, \dots, \tilde{C}^{l_k}, \dots, \tilde{C}^L\}$, where $(\bigcup_{l_k} \tilde{C}^{l_k}) \cup \tilde{C}^0$ is an ε_L -coreset of P_e , and $\varepsilon_L < \varepsilon$, according to Lemma 4.2. Therefore, we can compute a $(1 + \varepsilon)$ -approximate k-median of P_e by computing the exact k-median on $(\bigcup_{l_k} \tilde{C}^{l_k}) \cup \tilde{C}^0$, using the algorithm in [16].

So far we have shown how to compute the $(1 + \varepsilon)$ -approximate k-median clustering for only one update epoch. If we want to continuously maintain the k-median clustering on data received in all previous epochs, we need to maintain an EH -summary at the root that covers all previous epochs. Let EH_{root}^{past} denote the EH -summary for all past epochs. We can incrementally update it by merging with the EH -summary EH_{root} for epoch e , using Algorithm 2. $EH_{root}^{past} \leftarrow CombineEHSummary(EH_{root}^{past}, EH_{root})$. This summary is efficient, with guaranteed approximation and can be incrementally maintained. Performing the exact k-median clustering on this summary will always produce a $(1 + \varepsilon)$ -approximate k-median clustering.

4) *Overall Analysis:* The transmission cost for the topology-oblivious algorithm can be derived as follows. For each site, we transfer the EH -summary instead of the original stream data. Assume that the EH -summary is $EH = \{\tilde{C}^0, \tilde{C}^{l_1}, \dots, \tilde{C}^{l_k}, \dots, \tilde{C}^L\}$. The size of EH depends on the size of coreset \tilde{C}^{l_k} at each level l_k , and the number of levels. The size of any \tilde{C}^{l_k} is no more than $O(\frac{k}{\varepsilon_{l_k}} \log N) = O(\frac{kl_k \sqrt{l_k}}{\varepsilon} \log N)$, which is bounded by $O(\frac{k}{\varepsilon} \log^2 N \sqrt{\log N})$, since $l_k < \log N$. Totally, there are no more than $\log N$ levels. Therefore, the EH -summary size is $O(\frac{k}{\varepsilon} \log^3 N \sqrt{\log N})$. Overall, the communication for all the sites is no more than the number of sites times the max per node transmission bound.

B. Height-aware algorithm

In this section, we assume that the height of the tree h is known. We propose a height-aware algorithm to further reduce the transmission size. The basic idea of the algorithm is to compute a coreset on top of the EH -summary and use it as the local summary for transmission. Each internal node computes another coreset after merging all the coresets from the children with the local coreset. We incorporate both local coreset computation and coreset combination into the algorithm shown in Algorithm 3:

Theorem 4.1: The coreset C_T computed at the root node using Algorithm 3 is an ε -coreset of the data received over all streams.

At any node, let $EH = \{\tilde{C}^0, \tilde{C}^{l_1}, \dots, \tilde{C}^{l_k}, \dots, \tilde{C}^L\}$ be the EH -summary after the first step of Algorithm 3. According to Lemma 4.2, $(\bigcup_{l_k} \tilde{C}^{l_k}) \cup \tilde{C}^0$ is an $(\frac{\varepsilon}{2} - \frac{\varepsilon}{3\sqrt{L}})$ -coreset of P , since $\varepsilon_L = \sum_{l=1}^L \frac{1}{\sqrt{l}} \frac{\varepsilon}{6} \leq \frac{\varepsilon}{2} - \frac{\varepsilon}{3\sqrt{L}}$. In Step 2, we compute a $\frac{1}{3\sqrt{L}}\varepsilon$ -coreset C_{EH} on EH , which makes C_{EH} an $\frac{\varepsilon}{2}$ -coreset of P . Whenever we move up from a node to its parent in the tree, the combination of coresets in Steps 3 and 4 increases the error of coreset by $\frac{\varepsilon}{2h}$. Since the total height is h , the final

Algorithm 2 CombineEHSummary(EH, EH_*)

```

1:  $EH_{all} \leftarrow \Phi, l \leftarrow 0$ 
2:  $\tilde{C}_{all}^0 \leftarrow \tilde{C}^0 \cup \tilde{C}_*^0$ 
3: if  $|\tilde{C}_{all}^0| > B$  then
4:   Divide  $\tilde{C}_{all}^0$  into block  $B_c$  of size  $B$  and set the remaining
      part to be  $\tilde{C}_{all}^0$ . Add  $\tilde{C}_{all}^0$  into  $EH_{all}$ 
5:   Compute  $\Delta_{\varepsilon_1}$ -coreset  $\Delta\tilde{C}_{all}^1$  over  $B_c$ 
6: else
7:   Add  $\tilde{C}_{all}^0$  into  $EH_{all}$ 
8:    $\Delta\tilde{C}_{all}^1 = \Phi$ 
9: end if
10: for  $l = 1$  to  $L$  do
11:    $\Delta\tilde{C}_{all}^{l+1} = \Phi$ 
12:   if  $\tilde{C}^l \in EH$  and  $\tilde{C}_*^l \in EH_*$  then
13:     Compute  $\Delta\tilde{C}_{all}^{l+1}$  as the  $\Delta_{\varepsilon_{l+1}}$ -coreset over  $\tilde{C}^l \cup \tilde{C}_*^l$ 
14:     if  $\Delta\tilde{C}_{all}^{l+1} \neq \Phi$  then
15:        $\tilde{C}_{all}^{l+1} \leftarrow \Delta\tilde{C}_{all}^{l+1}$ , add  $\tilde{C}_{all}^{l+1}$  into  $EH_{all}$ .
16:     end if
17:   else
18:     if  $\Delta\tilde{C}_{all}^{l+1} \neq \Phi$  then
19:       if  $\tilde{C}^l \in EH$  then
20:         Compute  $\Delta\tilde{C}_{all}^{l+1}$  as the  $\Delta_{\varepsilon_{l+1}}$ -coreset over
            $\tilde{C}^l \cup \Delta\tilde{C}_{all}^{l+1}$ 
21:       else
22:         if  $\tilde{C}_*^l \in EH$  then
23:           Compute  $\Delta\tilde{C}_{all}^{l+1}$  as the  $\Delta_{\varepsilon_{l+1}}$ -coreset over
              $\tilde{C}_*^l \cup \Delta\tilde{C}_{all}^{l+1}$ 
24:         else
25:            $\tilde{C}_{all}^{l+1} \leftarrow \Delta\tilde{C}_{all}^{l+1}$ , add  $\tilde{C}_{all}^{l+1}$  into  $EH_{all}$ 
26:         end if
27:       end if
28:     else
29:       if  $\tilde{C}^l \in EH$  then
30:          $\tilde{C}_{all}^{l+1} \leftarrow \tilde{C}^l$ , add  $\tilde{C}_{all}^{l+1}$  into  $EH_{all}$ 
31:       else
32:         if  $\tilde{C}_*^l \in EH$  then
33:            $\tilde{C}_{all}^{l+1} \leftarrow \tilde{C}_*^l$ , add  $\tilde{C}_{all}^{l+1}$  into  $EH_{all}$ 
34:         end if
35:       end if
36:     end if
37:   end if
38: end for

```

Algorithm 3 CombineCoresetHW(P, h)

```

1: Compute the local  $EH$ -summary  $EH = \{\tilde{C}^0, \tilde{C}^{l_1}, \dots, \tilde{C}^{l_k}, \dots, \tilde{C}^L\}$  using Algorithm 1, with
    $\Delta\varepsilon_l = \frac{1}{l\sqrt{l}}\varepsilon$ 
2: Compute a  $\frac{1}{3\sqrt{L}}\varepsilon$ -coreset  $C_{EH}$  on  $(\bigcup_{l_k} \tilde{C}^{l_k}) \cup \tilde{C}^0$ 
3: Take the union of  $C_{EH}$  with all the coresets  $C_{EH}^j$  received from
   the child nodes  $C_U = C_{EH} \cup (\bigcup_j C_{EH}^j)$ 
4: Compute an  $\frac{\varepsilon}{2h}$ -coreset  $C_T$  on  $C_U$ 
5: Return  $C_T$  as the final coreset for transmission

```

coreset C_T at root will be a ε -coreset of P ($\frac{\varepsilon}{2} + \frac{\varepsilon}{2h}h = \varepsilon$). In this algorithm, we only need to know the height of the entire routing tree h . Sites do not know their locations in the tree.

Overall Analysis The transmission cost for the height-aware algorithm can be derived as follows. For each site, we compute and transfer a single coreset C_T (Algorithm 3, Step 4) instead of an EH -summary as in topology-oblivious algorithm. C_T is an ε -coreset of C_U , where $|C_U| < N$, thus $|C_T| = O(\frac{k}{\varepsilon/2h} \log N)$. Therefore, the max per node transmission for height-aware algorithm is $O(\frac{kh}{\varepsilon} \log N)$, which is smaller compared with the topology-oblivious case considering small h . However, we need to know the height of the tree beforehand.

C. Path-aware algorithm

In height-aware algorithm, the additive approximation factor $\frac{\varepsilon}{2h}$ is uniformly assigned to each site. In this section, we assume that each site is aware of the height of the subtree rooted at itself. This information can be obtained during the routing process. With the extra information about the topology, we propose a path-aware algorithm which assigns approximation factor uniformly along each *path*. The path-aware algorithm further reduces data transmission compared with height-aware algorithm.

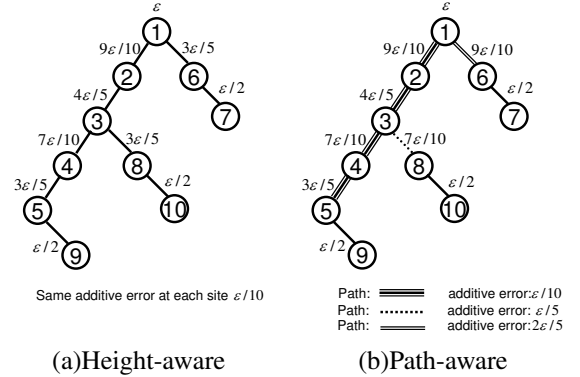


Fig. 2. Error accumulation of Height-aware and Path-aware algorithms. This figure compares the different strategies of assigning additive approximation factors at each site of the tree for height-aware and path-aware algorithms. Height-aware algorithm assigns the additive error uniformly to $\frac{\varepsilon}{2h}$, where h is the height of the tree. Path-aware algorithm assigns the additive error uniformly inside each sub-path, but differently for different sub-paths.

In the height-aware algorithm, each site computes an $\frac{\varepsilon}{2}$ -coreset of the local data, merges all coresets from the children, and computes another $\frac{\varepsilon}{2h}$ -coreset on top of it. It works well for balanced tree. However, for unbalanced tree, the transmission of the sites on a shorter path in the tree can be further reduced. For example, in Fig.2(a), the height of the tree is 5, so that the additive approximation factor at each site is $\frac{\varepsilon}{10}$, which is determined by the longest path in the tree (path $9 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1$). However, for a shorter path such as $7 \rightarrow 6 \rightarrow 1$, we can actually take additive approximation factor as $\frac{2\varepsilon}{5}$ instead of $\frac{\varepsilon}{10}$ without affecting the final approximation factor ε at the root. This approach saves communication because the coreset size is inversely proportional to the approximation factor. Therefore, we may determine the additive error on each

path separately. In the following discussion, we propose a top-down algorithm for assigning additive approximation factor $\Delta\epsilon$ in a piecewise fashion along each path, assuming that each node knows the height of the subtree rooted at itself.

Algorithm 4 AssignEps(S_i, ϵ_i)

Input S_i : current site; ϵ_i : the maximum possible approximation factor of the coreset sent by S_i 's children

```

1: if  $S_i$  is leaf then
2:   return
3: end if
4: if  $S_i$  is root then
5:    $\Delta\epsilon_{S_i} = \epsilon_i/h$ , where  $h$  is the height of the entire routing tree
6:    $\epsilon_i = \epsilon_i - \epsilon_{S_i}$ 
7: end if
8: for each site in  $S_i$ 's children set  $\{SC_j\}$  do
9:   if  $SC_j$  is a leaf then
10:    Assign the additive approximation factor of  $SC_j$  to be
         $\Delta\epsilon_{SC_j} = \epsilon_i$ 
11:   else
12:    Assign the additive approximation factor of  $SC_j$  to be
         $\Delta\epsilon_{SC_j} = \frac{\epsilon_i}{\text{height}(SC_j)}$ , where  $\text{height}(SC_j)$  is the height
        of the subtree rooted at  $SC_j$ .
13:   end if
14:   AssignEps( $SC_j, \epsilon_i - \epsilon_{SC_j}$ )
15: end for

```

The algorithm proceeds as follows (Algorithm 4). Initially, we call AssignEps($root, \frac{\epsilon}{2}$) and the algorithm runs recursively and traverses the tree to assign the proper additive approximation factor for each site. In Fig.2(b), for example, the algorithm starts with root node S_1 AssignEps($S_1, \epsilon/2$), and set the additive error $\Delta\epsilon_{S_1}$ at root to be $\epsilon/10$ (height of the tree $h = 5$). Two children of S_1 are S_2 and S_6 , where $\text{height}(S_2) = 4$, and $\text{height}(S_6) = 1$. Therefore, we assign the additive approximation factor at S_2 as $\Delta\epsilon_{S_2} = \epsilon/10$. Similarly, the additive approximation factor at S_6 is $\Delta\epsilon_{S_6} = 2\epsilon/5$. Note that $\Delta\epsilon_{S_6}$ is assigned to be $\epsilon/10$ in height-aware algorithm. $\Delta\epsilon_{S_8}$ is also assigned to be $\epsilon/5$ instead of $\epsilon/10$.

After the additive error assignment phrase, the remaining part of the path-aware algorithm is the same as that of the height-aware algorithm.

V. EXPERIMENTS AND ANALYSIS

We have simulated our algorithms using different tree topologies with up to 50 nodes. We generated the routing tree by placing the sites on a 100×100 grid. In our experiments, we assumed that the sites can communicate with other sites within a distance of 5. Based on this assumption, we generated a site graph and used the distances between the sites as the weights of the site graph edges. A spanning tree algorithm is then applied to generate the routing tree.

A. Benchmark Data

We tested our algorithms on both real and synthetic datasets with up to millions of data points. Specifically, we applied our algorithms on the following two data sets:

- **New York stock Exchange (NYSE) Data:** We used an archived data set representing data volumes at the end of each day in the New York stock exchange². The data volume information is collected at the stock exchange over a hundred years. The overall dataset has over 30K observations. The trading data volume can vary significantly from day to day.
- **Synthetic data:** We generated synthetic data using a weighted combination of normalized distributions at a given set of centers. 15 centers are randomly chosen in the user-specified data range. We tested our algorithms on up to 2 million observations.

B. Results and Analysis

We performed our experiments in different configurations and analyzed the data transmission as a function of the total stream data size, the approximation bound in k-median computation, and the number of sites. In this section, we present our experimental results and compare with the theoretical bounds of our algorithms.

1) *Stream Data Size:* The max per node transmission is asymptotically bounded by a *polylog* function of total stream size N . For topology-Oblivious algorithm, the max per node bound is $Bound_{max} = O(\frac{k}{\epsilon} \log^3 N \sqrt{\log N})$; for height-aware algorithm, the max per node bound is $Bound'_{max} = O(\frac{kh}{\epsilon} \log N)$. Thus, the total data transmission size is asymptotically bounded by the number of sites times $Bound_{max}$. The advantage of our algorithms is more prominent for large data volume.

Fig.3 shows the overall and max per node transmission for synthetic and real data with varying per-epoch data volume of the input streams. For the experiments on both real data and synthetic data, the parameters we used are $k = 10$, $\epsilon = 0.05$. For real data, we tested all three proposed algorithms on a simulated stream system of 5 nodes, with total stream size varying between 20K and 28K. For synthetic data, we tested all three algorithms on a simulated stream system of 10 nodes, with total stream size varying between 1M and 9M.

Figs. 3(a) and 3(b) demonstrate the overall communication load of all three algorithms. In 3(a) and 3(b) we observe that the total communication of topology oblivious algorithm is more than height-aware algorithm, and height-aware algorithm is more than path-aware algorithm. All three algorithms are well below the corresponding theoretical bounds for total data transmission. Figs. 3(c) and 3(d) highlight the maximum per node data transmission of our algorithms on the real and synthetic data. The plots also indicate a *polylog* relationship between data transmission and total stream size. In addition, height-aware algorithm and path-aware algorithm have much smaller max per node transmission than topology-oblivious algorithm. In our experiments, we observed a significant reduction in both the total and max per node data transmission, and more reduction for larger stream size. We noticed that the max per node transmission of path-aware algorithm is only a

²<http://www.nyse.com/marketinfo/datalib/1022221393023.html>

small fraction ($\approx 20\%$) of the overall stream size for real data (see Fig. 3(c)) and ($\approx 2\%$) for synthetic data (see Fig. 3(d)).

2) *Number of Sites*: We demonstrate the total transmission and max per node transmission by varying the number of sites. The total data transmission is asymptotically linear to the number of sites. Figs. 4(a) and 4(b) highlight the total data transmission as a function of the number of sites for both NYSE dataset and synthetic dataset. We fix the total input stream size to $30K$ for the real data and $5M$ for the synthetic data. We observe that the total data transmission by all three algorithms are below the theoretical bound. The graphs indicate that the height-aware algorithm performs better than the topology-oblivious algorithm, and the path-aware algorithm performs better than the height-aware algorithm. Figs. 4(c) and 4(d) highlight the maximum per node data transmission as a function of the number of sites in the network. The height-aware and path-aware algorithms slowly increase with larger number of sites due to the increases of the height of the tree. The topology-oblivious algorithm does not exhibit increasing tendency with the change of the number of sites. The ups and downs of the topology-oblivious curve are due to the different distribution of the total stream data in different tree topologies (with increasing sensor numbers). The graphs demonstrate the scalability of the algorithms in terms of the number of sites.

3) *Approximation Error*: The data transmission is expected to reduce when a larger approximation error is allowed. Using the same datasets, our experiments demonstrate that the total transmission in all three algorithms declines slowly as the approximation error increases on both synthetic and real data, as shown in Figs. 5(a) and 5(b). Figs. 5(c) and 5(d) highlight the maximum per node data transmission as a function of the approximation error, which also decreases slowly with larger approximation error. We observe that the path-aware algorithm requires the least total data transmission and the maximum data transmission per node.

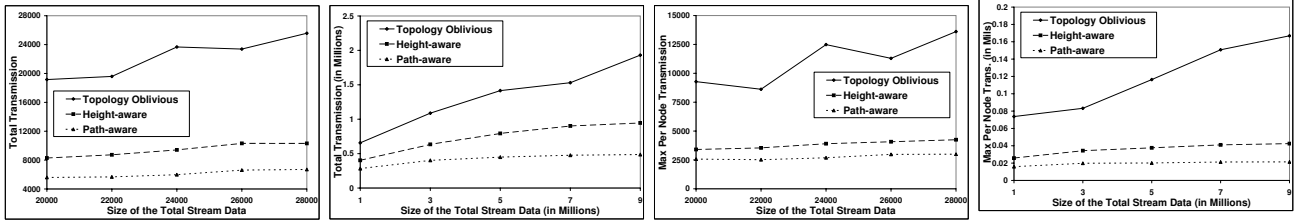
VI. CONCLUSIONS

We present algorithms for approximate k-median clustering over distributed data streams in three different settings: topology-oblivious, height-aware, and path-aware. Our algorithms reduce the max per node data transmission to $\text{polylog}(N)$. The topology oblivious algorithm runs without any prior knowledge of topology and the distribution of the stream data. We propose a multi-level summary structure and efficient algorithms to compute bounded-error approximate k-median. The performance can be further improved if the height of the tree is given. This leads to the development of our height-aware algorithm. If each site knows the path it is on, a path-aware algorithm is introduced to deliver even better performance than the height-aware algorithm, especially for unbalanced tree structure. In practice, our methods significantly reduce the data transmission requirements on both synthetic and real data sets to a small fraction of the overall volume of the streams and are also well below the theoretical bounds.

There are many promising avenues for future work. We would like to extend our algorithms to perform sliding window computations in streaming and sensor network model. Another interesting avenue is to extend the notion of bias to adapt to data distribution. We would also like to extend our clustering algorithms to higher order primitives for spatial computations, as well as other distributed applications.

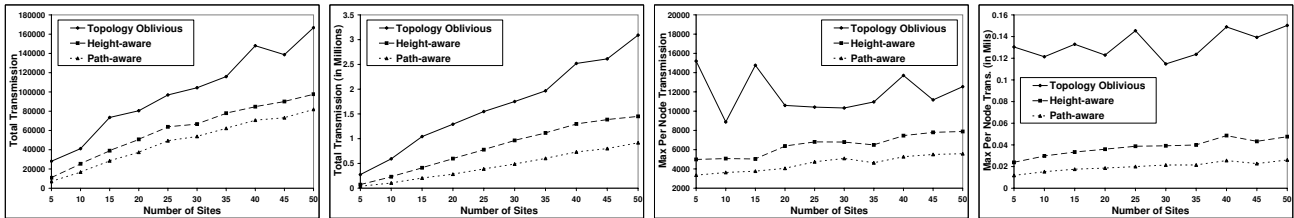
REFERENCES

- [1] Pankaj K. Agarwal, Sarel Har-Peled, and Kasturi R. Varadarajan. Geometric approximation via coresets. 2005.
- [2] Charu Aggarwal. On abnormality detection in spuriously populated data streams. In *Proceedings of ACM SIAM Conference on Data Mining*, April 2005.
- [3] Charu C. Aggarwal. A framework for diagnosing changes in evolving data streams. In *Proceedings of SIGMOD*, June 2003.
- [4] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for clustering evolving data streams. In *Proceedings of 29th VLDB Conference*, 2003.
- [5] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for projected clustering of high dimensional data streams. In *Proceedings of 30th VLDB Conference*, 2004.
- [6] Yanif Ahmad and Ugur Cetintemel. Network-aware query processing for stream-based applications. In *Proceedings of VLDB*, August 2004.
- [7] Brian Babcock and Chris Olston. Distributed top-k monitoring. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, June 2003.
- [8] Mitch Cherniack, Hari Balakrishnan, Magdalena Balazinska, Donald Carney, Ugur Cetintemel, Ying Xing, and Stan Zdonik. Scalable distributed stream processing. In *Proceedings of First Biennial Conference on Innovative Data Systems Research (CIDR 2003)*, January 2003.
- [9] Jeffrey Considine, Feifei Li, George Kollios, and John Byers. Approximate aggregation techniques for sensor databases. In *Proceedings of the International Conference on Data Engineering (ICDE04)*, March 2004.
- [10] Graham Cormode and Minos Garofalakis. Efficient strategies for continuous distributed tracking tasks. In *Proceedings of IEEE Data Engineering Bulletin*, pages 33–39, March 2005.
- [11] George Forman and Bin Zhang. Distributed data clustering can be efficient and exact. In *ACM KDD Explorations special issue on Scalable Data Mining Algorithms*, January 2001.
- [12] Gereon Frahling and Christian Sohler. Clustering data streams. In *IEEE Symposium on Foundations of Computer Science*, pages 359–366, 2000.
- [13] Gereon Frahling and Christian Sohler. Coresets in dynamic geometric data streams. In *Proc. 37th ACM Symposium on Theory of Computing*, pages 209–217, 2005.
- [14] Michael B. Greenwald and Sanjeev Khanna. Power-conserving computation of order-statistics over sensor networks. In *PODS*, June 2004.
- [15] Sarel Har-Peled and Akash Kushal. Smaller coresets for k-median and k-means clustering. In *Proceedings of the 21st annual symposium on computational geometry*, pages 126–134, 2005.
- [16] Sarel Har-Peled and Soham Mazumdar. Coresets for k-means and k-median clustering and their applications. In *ACM Symposium on Theory of Computing*, June 2004.
- [17] Eshref Januzaj, Hans-Peter Kriegel, and Martin Pfeifle. Towards effective and efficient distributed clustering. In *Workshop on Clustering Large Data Sets (ICDM2003)*, 2003.
- [18] Ram Keralapura, Graham Cormode, and Jeyashankher Ramamirtham. Communication-efficient distributed monitoring of thresholded counts. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, June 2006.
- [19] Samuel Madden, Michael J. Franklin, Joseph Hellerstein, and Wei Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *Proceedings of 5th Symp. Operating Systems Design and Implementation (OSDI 02)*, April 2002.
- [20] Liadan O’Callaghan, Nina Mishra, Adam Meyerson, and Sudipto Guha. Streaming data algorithms for high-quality clustering. In *Proceedings of IEEE International Conference on Data Engineering*, March 2002.
- [21] Chris Olston, Jing Jiang, and Jennifer Widom. Adaptive filters for continuous queries over distributed data streams. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, June 2003.



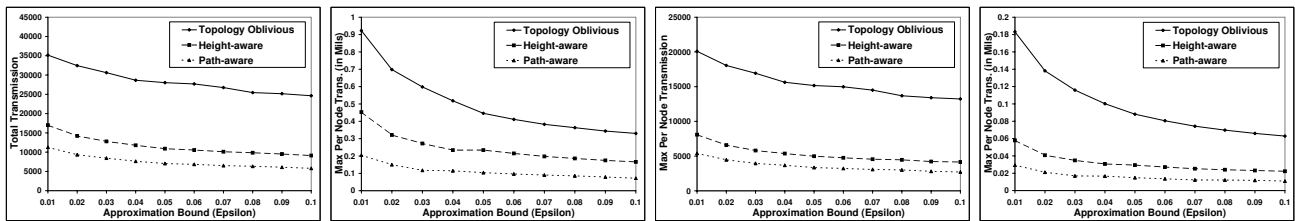
(a) Overall communication on real data (b) Overall communication on synthetic data (c) Maximum per node communication on real data (d) Maximum per node communication on synthetic data

Fig. 3. Performance of our algorithms as a function of the total stream size. We measured the overall communication among the sensor network nodes to perform k-median clustering on sensor network nodes using real and synthetic data. We performed our experiments on the NYSE data consisting of up to 28K records and synthetic data with up to 9 million data values. In our experiments, we used an approximation error threshold of 0.05. For real data, we tested all three algorithms on a 5-node system. For synthetic data, we tested on a 10-node system. Figs. 3(a) and 3(b) demonstrate the overall data communication of our algorithms as a function of input data size. Figs. 3(c) and 3(d) demonstrate the max per node data communication of our algorithms as a function of input data size. Our experiments demonstrate a significant reduction in the overall and max per node communication.



(a) Normalized overall communication on real data (b) Normalized overall communication per node on synthetic data (c) Normalized maximum per node communication on real data (d) Normalized maximum per node communication on synthetic data

Fig. 4. Performance of our algorithms as a function of the number of sites. We measured the data communication of our algorithms as a function of the number of sites on NYSE and synthetic data. The total input stream size is fixed to be 30K for the NYSE data and 5K for the synthetic data. Figs. 4(a) and 4(b) highlight the total data transmission as a function of the number of sites. Figs. 4(c) and 4(d) demonstrate the maximum per node data transmission as a function of the number of sites.



(a) Overall communication on real data (b) Overall communication on synthetic data (c) Maximum per node communication on real data (d) Maximum per node communication on synthetic data

Fig. 5. Performance of our algorithms as a function of approximation error. The overall communication of our algorithm decreases as the error increases. Figs. 5(a) and 5(b) highlight the overall data communication as the error increases. Figs. 5(c) and 5(d) demonstrate the max per node data communication as the error increases. In our experiments, we used 10 centers and performed approximate clustering on NYSE data with 30K data records and synthetic data with 5 million observations. As the error tolerance increases, we observe that both the height-aware and path-aware algorithms perform better than the topology-oblivious algorithm and can further reduce the communication by additional 10 – 30%.

[22] Izchak Sharfman, Assaf Schuster, and Daniel Keren. A geometric approach to monitoring threshold functions over distributed data streams. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, June 2006.

[23] Rebecca M. Willett, Aline M. Martin, and Robert D. Nowak. Adaptive sampling for wireless sensor networks. In *Proceedings of ISIT04*, 2004.

[24] Adam Silberstein Rebecca Braynard Jun Yang. Constraint chaining: On

energyefficient continuous monitoring in sensor networks. In *Proceedings of 5th Symp. Operating Systems Design and Implementation(OSDI 02)*, June 2006.

[25] Yunyue Zhu and Dennis Shasha. Efficient elastic burst detection in data streams. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 336–345, 2003.