Approximate Quantile Computation over Sensor Networks*

Xifeng Yan[†], Jiong Yang[†], Wei Wang [‡], Jiawei Han[†]

[†]University of Illinois at Urbana-Champaign, {xyan, jioyang, hanj}@cs.uiuc.edu

[‡]University of North Carolina at Chapel Hill, weiwang@cs.unc.edu

Abstract

Sensor networks have been deployed in various environments, from battle field surveillance to weather monitoring. The amount of data generated by the sensors can be large. One way to analyze such large data set is to capture the essential statistics of the data. Thus the quantile computation in the large scale sensor network becomes an important but challenging problem. The data may be widely distributed, e.g., there may be thousands of sensors. In addition, the memory and bandwidth among sensors could be quite limited. Most previous quantile computation methods assume that the data is either stored or streaming in a centralized site, which could not be directly applied in the sensor environment. In this paper, we propose a novel algorithm to compute the quantile for sensor network data, which dynamically adapts to the memory limitations. Moreover, since sensors may update their values at any time, an incremental maintenance algorithm is developed to reduce the number of times that a global recomputation is needed upon updates. The performance and complexity of our algorithms are analyzed both theoretically and empirically on various large data sets, which demonstrate the high promise of our method.

1 Introduction

Sensor networks have become increasingly common in our everyday life. Wireless sensors are one type of sensors which gains popularity due to the fact that they can be deployed anywhere. Wireless sensor network usually consists of a large number of small, battery-powered, wireless sensors. Deployed in an ad-hoc fashion, these sensors collaborate to monitor physical environments at fine spatial and temporal scales [6], as illustrated below.

- Battlefield surveillance. The wireless sensors may be attached to soldiers and military vehicles in the battlefield. These sensors may record and report various status and conditions, e.g., the amount of remaining fuel and ammunition, the surface temperature, the blood pressure, the heart beat rate, etc. Based on such information, the field commanders can make timely and intelligent decisions.
- Weather monitoring. To monitor the changes in atmosphere, a large number of sensors are deployed to record various kinds of data, such as temperature, wind velocity, precipitation, etc., at different locations.
- Environment monitoring. In a large building or factory, it is important to keep track of the temperature or air quality of different parts of the building.

It is important to keep track of the basic statistics of the values measured by the sensors. For instance, it is extremely important for a commander to know the average or median amount of fuel left in his tanks in a battlefield.

A monitoring infrastructure is proposed in [14] for computing aggregates in a sensor network. Sensors are organized in a tree structure as shown in Figure 1. This organization also fits the communication model of sensors since they usually can only contact with their neighbors due to spatial and power constraints. In this architecture, every node in the tree is a sensor. To compute the aggregates, every internal node computes the aggregates over all the data from itself and its descendants and reports them to its parent. However, the authors in [14] admit that this method only works for *decomposable* functions, but not for non-decomposable functions. f is a *decomposable* function if there exists a function g such that for all k

$$f(v_1, \ldots, v_n) = g(f(v_1, \ldots, v_k), f(v_{k+1}, \ldots, v_n)).$$

Decomposable functions include min, max, $average$, etc. Unfortunately, median and other ϕ-quantiles are not decomposable, where the ϕ-quantile is the element ranking $\lceil \phi N \rceil$

*Submitted on July 2nd 2003, ICDE#274. The work was supported in part by National Science Foundation under Grant No. 02-09199, the Univ. of Illinois, and an IBM Faculty Award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

if the N values are sorted in the increasing order. Quantiles cannot be computed by the method proposed there. However, quantiles are very useful in understanding the statistics of the sensor data. For instance, a military commander may wish to know the amount of remaining fuel for the 10% of tanks with the least amount of fuel, whereas a safety monitor may wish to know the median inflammable gas concentration in a mine. These are the examples of quantile computation over sensor networks. Besides the undecomposable property, quantile computation over sensor networks has the following issues.

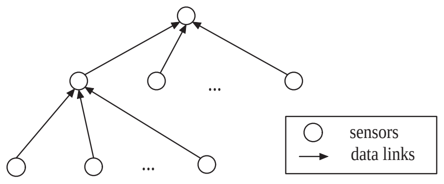

Figure 1. A Tree-Structured Sensor Network

- Limited memory: A sensor may only consist of several KB memory physically.
- Limited bandwidth: A sensor can only emit a limited volume of data in its life cycle.
- Highly distributed: The number of sensors can be in the range of hundreds to thousands or even more.
- Dynamically organized: At any given time, some sensors may leave the network due to various reasons, such as power outage, etc. New sensors may join the network too.
- Incremental maintenance required: The sensor values may change significantly over time. It is infeasible (both in time and energy consumption) to recompute the quantile value of the whole sensor network upon each update. Thus, an incremental update algorithm has to be worked out.

Due to all these constraints, it is infeasible to compute the exact quantile without incurring a great burden on the sensor network since it requires all sensors to send their data to the central server and be processed there. It would take too much energy and/or memory to relay the data to the server. A reasonable alternative is to compute an ϵ-approximate ϕ-quantile over N sensors which are organized in a tree structure like Figure 1, i.e., to compute an approximate quantile whose rank is within the range $[(\phi - \epsilon)N, (\phi + \epsilon)N]$ in a tree-structured sensor network (assuming each sensor provides one value).

The most common technique for computing quantile sketch is to use the traditional histogram. There are two main types of histograms: *equi-width* vs. *equi-depth*. Both methods cannot guarantee the error bound.

There have been several interesting proposals for online quantile computation [11, 9, 13]. However, most of them assume a centralized environment. Moreover, they assume either the data cardinality is known (in which case, the memory can be bound) or the memory is not bounded (i.e., it grows with the cardinality of the data set). Unfortunately, this is not the case in the sensor network where the memory is physically bounded, e.g., 4KB [10], but the number of sensors may be unknown due to the dynamic nature of the sensor network. Obviously, the existing works are not directly applicable to the scenario studied in this paper.

In this paper, a new algorithm is proposed that can minimize the error bound of the quantile. First, an existing algorithm ([13]) is extended to cover the distributed form. To address the problem of limited memory, a novel *adaptive buffer allocation* technique is invented. Previous work [11] proposes maintaining several buffers in the memory, called *sketch*, which is a summarization of the original data. In these approaches, both the number and the size of buffers are fixed at run time. In our approach, at lower levels of the sensor network tree, the memory is partitioned to a smaller number of buffers and the size of each buffer is large, which leads to less compression and higher accuracy. At higher levels, the compression is enhanced so that the number of buffers becomes larger and the size of each buffer gets smaller. This novel dynamic buffer allocation scheme enables us to accommodate all the data in a sketch with limited memory.

In previous algorithms [11], each buffer is associated with a weight. The weight indicates the degree of compression. Weight w means that one element in the buffer represents w elements in the original data set. The weight for all the elements in a buffer is the same. We use *non-uniform* weight mechanism to solve the incremental maintenance issue. When a sensor reports a new value, both the new value and the old one are propagated to the root - We assume that the root is a server which has significant computing power and large memory. The weight associated with the old value in the buffer decreases by one, while the new value is recorded in the server. We can also insert this new value into the buffer using the existing incremental histogram maintenance methods which may introduce additional error. When the error bound reaches a certain degree, a new global sketch recomputation may be invoked. Therefore, the *dynamic buffer allocation* and the *non-uniform weight management* for approximate quantile computation form the two major contributions of this paper.

The remainder of the paper is organized as follows. We introduce related work in Section 2. Sections 3 and 4 present the problem formulation and overview of our approach. Section 5 presents the sketch generation and quantile computation algorithm, and Section 6 describes the incremental maintenance algorithm. The empirical results are reported and analyzed in Section 7. We discuss further extensions in Section 8 and conclude the study in Section 9.

2 Related Work

There are two areas of research work related to this study: (1) *computing aggregates of sensor data*, and (2) *one-scan quantile computation*.

2.1 Aggregates Computation over Sensor Data

Zhao et al. proposes a new method to *compute decomposable aggregates over sensor networks* [14], where a tree is constructed for a set of sensors. Since the radius that a sensor can send messages to is limited by its power, it is likely that a sensor may not be able to send the message directly to the base station. Thus, it will first send the message to another sensor, and that sensor may relay the message to the third one, and so on until the message reaches the base station. To eliminate duplicate messages and message loss, the sensors are organized into a tree. Each sensor has a known parent, and it will send the messages to its parent. As a result, there will be no duplicate messages. Moreover, if there is a missing message, it is easy to identify which sensor needs to resend it.

After constructing the tree, the decomposable aggregates can be easily computed in a *divide-and-conquer* fashion. Each internal node will compute the aggregates based on the data from its children and report the aggregates to its parent. Thus, all decomposable aggregates can be calculated efficiently. However, this method cannot be applied to compute quantile because it is not decomposable.

2.2 One-Scan Quantile Computation

The problem of computing median or quantiles has been explored over decades. The theoretic results and methods are summarized in the survey by Mike Paterson [13]. We are interested in the space requirement for computing the exact and approximate quantiles. Munro and Paterson [12] show that $\Omega(N^{1/p})$ elements are required to calculate the exact ϕ-quantile in p scans. Exact quantile computation over stream data using one scan must require the memory size equal to the data size. Thus, approximate quantile computing algorithms raised great interests recently. Several studies have made progress at minimizing the memory consumption. Manku, Rajagopalan and Lindsay [11] reformulate the Munro-Paterson algorithm [12] and present a more efficient one-scan algorithm that constructs an ϵ-approxiamte ϕ-quantile in $O(\frac{1}{\epsilon}log^2(\epsilon N))$ space. A probabilistic algorithm is also given in [11]. Greenwald and Khanna [9] propose an approach which associates rank error with each element maintained in the sketch. Their algorithm only requires $O(\frac{1}{\epsilon}log(\epsilon N))$ space to calculate ϵ-approximate quantile. Other related work includes multiple passes [3], different error metrics [7], computation efficient [2, 5, 13], distributed streams [8], and other statistics [4, 1].

3 Problem Formulation

In this paper, we study the problem of computing quantiles over a set of remote sensors. Before formulating the problem, we summarize the notations that will be used throughout this paper in Table 1.

Notation	Description
N	the number of sensors
ϕ	the ϕ-quantile to be calculated
v_ϕ	the value of ϕ-quantile calculated
$\hat{v}_\phi$	the real value of ϕ-quantile
$\hat{\phi}$	$\hat{\phi}N$: the real rank of v_ϕ
ϵ	quantile error : $\lvert\phi - \hat{\phi}\rvert$
M	the maximum space in each sensor
b	the number of buffers
k	the size of each buffer
X_i	the i_{th} buffer
$w(X_i)$	the weight of X_i

Table 1. Notation Used throughout the Paper

The sensors are organized in a sensor network as in Figure 1. In this tree, each internal node (except the root) and leaf node may be a sensor. For simplicity, we assume that all the original data are submitted from leaf nodes. Non-leaf nodes are only responsible for processing data or sketches collected from lower level sensors.

The ϵ-approximate ϕ-quantile is the value ranking between $\lceil(\phi - \epsilon)N\rceil$ and $\lceil(\phi + \epsilon)N\rceil$. We want to generate the sketch in the root node that minimizes ϵ when we query the ϕ-quantile. We call this problem *approximate quantile computation problem over sensor networks*.

The second task is to *dynamically maintain the sketch in the root node*. The value at a sensor may change at any time. If the value changes significantly, then the old value will be replaced with the new one. It is extremely inefficient to recompute the sketch from the scratch whenever such replacement takes place. Assume there is a user-specified approximation bound ϵ'. We aim to develop an algorithm that dynamically changes the sketch in the root node if the resulting error bound (for all quantiles) is less than ϵ'.

4 Overview

We first illustrate the general framework of our algorithm. The computation conducted in each sensor can be divided into three operations: COLLECT, CONDENSE, and SUBMIT.

i. COLLECT. The sensor collects its descendant sensors' data (including sketches) into its local sketch.

ii. CONDENSE. The sensor condenses the data in its local sketch so that it can reserve the space for incoming data.

iii. SUBMIT. After all the data from its descendant sensors are collected and processed, the sensor submits its sketch content (or summarized content) to the upper level sensor (its parent).

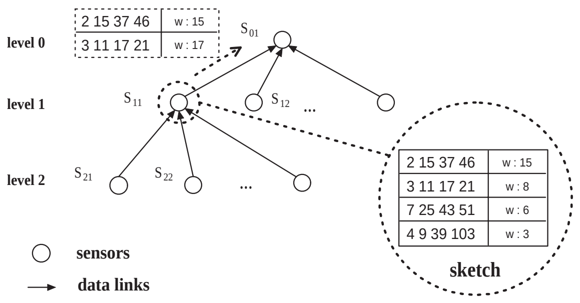

Figure 2. Distributed Quantile Computing

The first two operations are repeatedly conducted in every sensor. For the leaf sensor, the only step it has to do is to submit its value. Figure 2 shows the processing framework of a three-level sensor network (the *sketch* structure will be introduced in Section 5.1). Sensor s_{11} collects all the data from its underlying leaf sensors $s_{21}, \ldots, s_{2m}$ into its sketch. Assume the sketch structure has b buffers recording k elements each. Apparently, m may be larger than bk. Thus, we have to condense the buffers in order to leave space for new data. After condensing the buffers, some buffer may not represent k raw data any more. We assign weight w to each buffer, which shows how many raw data this buffer summarizes. When s_{11} collects all the data from leaf nodes, it will submit the data in its sketch to s_{01}. To conserve the power, it may not be good to submit all the data to the higher level sensor. Figure 2 shows an alternative, instead of submitting four buffers in the sketch to s_{01}, it condenses the four buffers into two buffers before submitting them. We discuss these alternatives in the following sections.

5 Approximate Quantile Computing

Previous work on computing quantiles focuses on a centralized environment: All raw data are directly sent to a central server for processing. The existing methods, e.g., one-scan algorithms over stream data, do not support distributed structures present in a sensor network. In this section, we first introduce the existing centralized quantile computation methods, and then present our algorithm which extends the Munro-Paterson algorithm [12] in the distributed environment. Further, we introduce the *adaptive buffer allocation mechanism* in the distributed environment where only limited memory is available in each node.

5.1 Centralized Environment

Manku et al. [11] builds a uniform framework for one-scan approximate quantile computation over stream data in a centralized system. Figure 3 illustrates a stream data processing model: data arrive at the central server one by one and are processed only once. A compact summarization structure (*sketch*) is maintained in the server to answer the quantile computation queries. In the centralized environment, there does not exist any distribution structure in the raw data. All the data go to the central sever once and then are discarded.

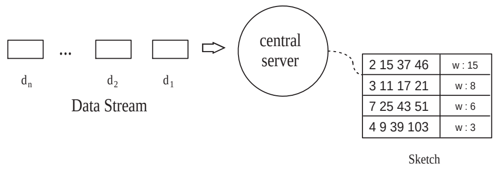

Figure 3. Centralized Quantile Computing

The framework proposed in [11] is parameterized by two integers b (the number of buffers) and k (the size of each buffer). A positive integer w (weight) is associated with each buffer, denoting how many raw input values are represented by each element in the buffer. For example, a $[2\ 15\ 37\ 46 \mid w : 15]$ buffer actually summarizes 60 (4×15) raw sensor values using 5 elements: four elements for sensor values, one for the weight. We can view these buffers as equi-depth histograms. Algorithms under this framework consist of three basic operations: NEW, COLLAPSE, and OUTPUT [11].

- NEW. The sensor allocates an empty slot in an unfilled buffer and records the input data. If the buffer is full and no space available for a new empty buffer, it will call the COLLAPSE operation to release some space.

- COLLAPSE. The sensor merges c full buffers, X_1, X_2,..., X_c into one buffer Y, where $w(Y) = \Sigma_{i=1}^{c} w(X_i)$. The detail of the collapse operation is described in [11]. Briefly, it first makes $w(X_i)$ copies of each element in X_i, puts all elements together, and sorts them into a sequence. Then it evenly segments the sorted sequence by width $w(Y)$ and selects the middle element in each segment as its representative in buffer Y. That is, each element is in position $jw(Y) + offset(Y)$ for $j = 0, \ldots, k-1$, where $offset(Y) = \frac{w(Y)+1}{2}$ if $w(Y)$ is odd, and $offset(Y)$ = $\frac{w(Y)}{2}$ or $\frac{w(Y)+2}{2}$ if $w(Y)$ is even.

  Figure 4 shows a collapse operation on two buffers a and b. Buffer a and b record the raw data: $w = 1$ means each element in a and b represents one raw value. We simply put the contents of these two buffers into a pool, sort them, evenly choose 4 elements to form a new buffer c. The result buffer c has weight 2 since each element in c represents two raw values. COLLAPSE operation discards a and b and retains c. Thus, the space of one buffer is released, which can be used for the NEW operation. Conceptually, a buffer is equivalent to an equi-depth histogram. However, the COLLAPSE operation does not involve any interpolating operation which is popularly used in merging two equi-depth histograms.

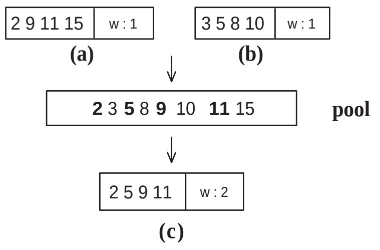

Figure 4. COLLAPSE **Operation**

- OUTPUT. The sensor makes $w(X_i)$ copies of each element in full buffers and 1 copy of elements in the unfilled buffer, sorts them into a sequence, and outputs the element in position $\lceil \phi N \rceil$ as the ϕ-quantile.

Figure 3 shows whenever new data arrive, we "squeeze" the sketch structure to accommodate them. We put data in the empty slots of an unfilled buffer if it exists. Otherwise, two or more buffers are collapsed so that new data can be stored in the newly freed buffer. In this framework, data is only processed in one scan. Algorithms vary according to different COLLAPSE policies. The Munro-Paterson algorithm implements one of the simplest policies: Select two buffers with the same weight and collapse them to one, the newly formed buffer has a doubled weight. Manku et al. [11] reviewed the Munro-Paterson algorithm [12] and proposed their own in a unified way. Recently, Greenwald-Khanna [9] proposed a different framework which consumes less memory than both of them. Although the Munro-Paterson algorithm is not as efficient as the Manku and Greenwald-Khanna algorithms, it is much easier to be applied in distributed environments. The analytical result about the error bound of the Munro-Paterson algorithm also holds straightforwardly in sensor networks. The simplicity comes from the fact that a tree structure as shown in Figure 1 does not affect the Munro-Paterson algorithm much. Therefore, our work is upon the Munro-Paterson algorithm. Certainly, it is an interesting problem to adapt the Manku and Greenwald-Khanna algorithms in the setting of sensor networks.

5.2 Distributed Environment

For the distributed environment shown in Figure 2, we apply the Munro-Paterson algorithm first in level-1 nodes (each level-1 node can be viewed as a local server) when they collect raw data from level-2 nodes as shown in Figure 2. After level-1 nodes build the internal sketches in their buffers, level-1 nodes submit their sketches to level-0 nodes. This process repeats between level-i nodes and level-$(i-1)$ nodes until it reaches the root node. The question is how to handle the COLLAPSE operations in the level-i node ($i > 0$) since level-i node may receive different weighted buffers from its children. That is, the problem setting in the distributed environment has changed from the original one where the Munro-Paterson algorithm was proposed. In the new context, data do not arrive as one raw element, but as a sketch. Data may come in a format like $[2\ 15\ 37\ 46]$ with weight 8, which represents underlying 32 (4×8) raw sensor values. The level-$(i-1)$ ($i < h$, where h is the height of the tree) nodes can only see the sketches instead of raw elements.

Suppose a level i node S collects sketches from its children. There are c buffers, $X_1, \ldots, X_c$, from the sketches of its children. We can reuse the operations defined in Section 5.1: If S has space, insert X_i into it. If there are two buffers with the same weight in S, a COLLAPSE operation is performed. Repeat the new and collapse operations until all c buffers are put in the sketch of S. For each node, a specific buffer is prepared to accept the data from the unfilled buffers in its child nodes. When a buffer is full, it constitutes a buffer with weight 1 and insert it into the sketch. For simplicity, we will ignore this case in the following discussion. We call the above algorithm *distributed Munro-Paterson algorithm* (DMP).

5.3 Distributed Environment with Limited Memory and Bandwidth

As stated in the introduction section, sensors usually only have limited memory and bandwidth available for quantile computation. In this section, we introduce a new operation, called FOLD, into DMP to make it fit this particular setting.

DMP requires that the total number of sensors, N, must be known in advance so that k and b can be determined before the algorithm runs. That is, parameters k and b have to be preset in each sensor. However, this requirement cannot always be satisfied because low level sensors may not know the total number of sensors existing in the whole network. Furthermore, the makeup of sensor networks may dynamically change due to many factors. For example, some sensors may complete their life cycle and detach from the sensor network, while the sensor network may expand with newly added sensors in the mean time. In either case, N is an unknown value. We modify DMP in order to handle this situation. We assume the only parameter we know is M, the maximum memory that a sensor can contribute to quantile computing. This parameter is often physically determined when the sensor is deployed.

Given limited memory and bandwidth in sensors, we would like to minimize the error bound of quantile computation in the root node. For simplicity, we assume each sensor has M elements to store sketches, and it can only submit these M elements to its parent node. In practice, our proposed algorithm can handle any bandwidth constraint which may has less than M elements.

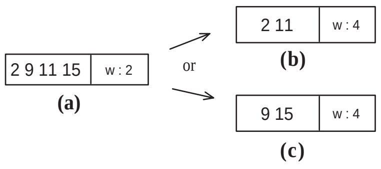

Figure 5. FOLD **Operation**

For a given k and b, the maximum capacity of DMP (including the original Munro-Paterson algorithm) is $k2^{b-1}$. The proof is given in Section 5.4. That is, when the total number of sensors exceeds $k2^{b-1}$, DMP cannot process them unless the values of k and b are adjusted. Therefore, we introduce a new operation FOLD: When $N > k2^{b-1}$, the FOLD operation will fold a buffer in half: only retain half of the elements and free the remaining ones. Suppose there are k elements $e_1, e_2, \ldots, e_k$ in each buffer. If k is even, we could either choose $e_1, e_3, \ldots, e_{k-1}$ or $e_2, e_4, \ldots, e_k$. If k is odd, we could either choose $e_1, e_3, \ldots, e_k$ or $e_2, e_4, \ldots, e_{k-1}$. That means, the new buffer has a half size of the original one, but its weight is doubled. Figure 5 shows that a FOLD operation is applied to buffer a. It sorts the elements in a and evenly splits them into buffer b and c. Either of them can be taken as the folding result.

If we apply the FOLD operations on every buffer [1] in the sketch, the maximum capacity of the sketch turns to $k2^{2b-1}/2$, which is larger than the previous capacity $k2^{b-1}$ if $b > 1$. We call the new algorithm *adaptive distributed Munro-Paterson algorithm* (AMP). Its pseudo-code is shown in Algorithm 1.

Algorithm 1 Adaptive Distributed Munro-Paterson Algorithm

```
Input: A series of buffers $B_i$, which has size $k_i$ and weight $u_i$
Output: A series of buffers $O_j$ with size $k$ and weight $v_j$

1: initialize $O_1 = B_1$, $k = k_1$, $v_1 = u_1$;
2: for each $B_i$ ($i \geqslant 2$) do
3:     if $k < k_i$, then
           FOLD $B_i$ $log(k_i/k)$ times,
           set $u_i = u_i k_i/k$ and $k_i = k$;
4:     if $k > k_i$, then
           FOLD all $O_j$ $log(k/k_i)$ times,
           set $v_j = v_j k/k_i$ and $k = k_i$;
5:     if $\exists$ an empty $O_j$, then insert $B_i$;
6:     else
7:         if $\exists$ $O_j$ s.t. $v_j \equiv u_i$, then merge $B_i$, double $v_j$;
8:         else FOLD all $O_j$, $k = k/2$, double $v_j$, goto line 3;
9: return non empty buffers $O_j$;
```

The maximum capacity we can achieve through FOLD operations is around $2^M (k = 1$ and $b = M)$. Considering M is usually larger than 32, it is large enough to accommodate any existing sensor network in the world.

The previous discussion only considers the memory limitation and assume each node can completely submit its sketch to the higher level node. If it is not the case, we can reduce the sketch size by FOLDing the buffers in the sketch. Thus, AMP can compute quantile in bandwidth-limited distributed environments too.

5.4 Analysis

In this section, we first briefly describe the error bound of the Munro-Paterson algorithm (given in [11]) and then analyze the error bound of DMP and AMP by comparing them with the Munro-Paterson algorithm.

[1]Actually we may fold the longest buffers on demand. This may produce higher accuracy because less compression is made on the buffers. Due to space limitation, we will not show the detailed proof and analysis of this variation.

5.4.1 Munro-Paterson Algorithm

With fixed value of k and b, the *Capacity*, the maximum number of elements that the Munro-Paterson algorithm can process is fixed.

Property 1 *The capacity of the Munro-Paterson algorithm is $k2^{b-1}$.*

Since the Munro-Paterson algorithm always collapses two equal-weighted buffers, the weights follow the geometric series with some values absent. The sketch reaches its maximum capacity when all the b buffers are full and the weights of buffers constitutes a sequence like, $1,\ 1,\ 2,\ ,\dots,2^{b-2}$. The first 1 is for the unfilled empty buffer, the second 1 is the full buffer with weight 1. In this case, the sketch cannot accept any new input data since there is no two buffers having the same weight and no COLLAPSE operation can be performed.

Next, let us examine the rank error between the approximate quantile and the real quantile. We denote the rank error by R_e, which defines the error bound of calculated quantile: $\epsilon = R_e/N$.

Property 2 *The rank error generated by the algorithms based on the above framework is at most $\frac{W-C-1}{2} + w_{max}$, where W denotes the weight sum of all the COLLAPSE operations performed during the whole process, C denotes the total number of COLLAPSE operations, and w_{max} is the maximum weight among the buffers before the OUTPUT operation is invoked [11].*

Please refer to [11] for the proof of Property 2.

5.4.2 Distributed Munro-Paterson Algorithm

Let's examine the rank error generated by DMP. Interestingly, we found it is exactly the same as that of the original Munro-Paterson algorithm if the number of raw values are the same. These two algorithms can run in two settings: one for data stream, and the other for sensor network. Suppose the former has N raw values and the latter has N leaf nodes, we show that the error bounds of the quantile outputs are the same if both of them have the same value of k and b. We depict these two settings in Figure 6. Figure 6(a) is a setting of data stream: raw data arrive at the root node directly while Figure 6(b) is a setting for sensor networks: raw data arrive at low level sensors first and then their sketches are reported to the higher level sensors. Without loss of generality, we only show two levels here in Figure 6(b). We call the former setting $stream$*-setting*, and the latter $sensor$*-setting*.

We denote the sum of weights of all COLLAPSE operations in the $stream$-setting by W and the number of COLLAPSE operations by C. Correspondingly, these two values in the $sensor$-setting are W' and C' respectively.

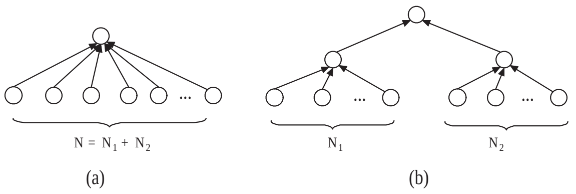

Figure 6. Munro-Patterson Algorithm and its Distributed Version

Lemma 1 $W = W'$ *and* $C = C'$.

Proof. Suppose the root node in the $stream$-setting has l full buffers, $X_1, \dots, X_l$ ($w(X_i) < w(X_{i+1})$) and the root node in the $sensor$-setting has l' full buffers, $X'_1, \dots, X'_l$ ($w'(X_i) < w'(X_{i+1})$). We have $l = l'$ and $w(X_i) = w'(X'_i)$ because the decomposition of $\lfloor N/k \rfloor$ to a sum of several elements in the geometric series is unique (we can view this decomposition as the binary representation of $\lfloor N/k \rfloor$). If two buffers share the same weight w, they must share the exact same history of COLLAPSE operations though the content of buffers may be different. That is, each buffer must be collapsed from two buffers with weight $w/2$, each of which must be collapsed from another two buffers with weight $w/4$, and so on. Thus, $W = \sum_{i=1}^{l} log(w(X_i)) \times w(X_i)$. Similarly, we have $W' = \sum_{i=1}^{l'} log(w'(X'_i)) \times w'(X'_i)$ too. Therefore, $W = W'$. For the total number of COLLAPSE operations, we can draw the similar result, $C = C'$. ■

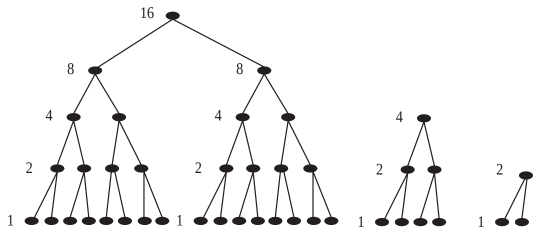

Figure 7. Collapse Trees: Stream-Setting

We illustrate Lemma 1 using Figure 7 and 8 which are the tree representations of operations carried by these two algorithms [11] in the $stream$-setting and $sensor$-setting, respectively. The leaf node represents a buffer with weight 1. The non-leaf node represents a buffer which is collapsed from two child buffers. The labels indicate their weights. W (also W') is equal to the weight sum of all the nodes. C is equal to the half number of the edges. The trees in Figure 8 are divided into two parts: one from the sketch construction in the n_1 partition (Figure 6(b)), the other from that in the n_2 partition (Figure 6(b)). Be careful that the trees in Figure 7 and 8 have different semantics from that in Figure 6. According to DMP, these trees finally are collapsed together. As we can see, the trees in Figure 7 and 8 are exactly the same although they are generated in different orders: in the $stream$-setting, the tree is generated from left to right and bottom to up whereas in the $sensor$-setting the trees for n_1 and n_2 parts are generated first and then collapsed. The exact same structure of COLLAPSE operations in both settings indicates the sum of weights and the total number of COLLAPSE operations are equal in both settings.

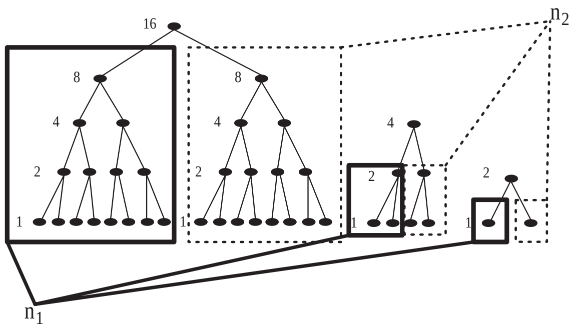

Figure 8. Collapse Tree: Sensor-Setting

Lemma 2 *The rank error generated by the distributed Munro-Paterson algorithms is also at most* $\frac{W-C-1}{2} + w_{max}$.

Proof. by Property 2 and Lemma 1. ■

Thus, the error bound given by the original Munro-Paterson algorithm can be also applied in the distributed Munro-Paterson algorithm. We have $W = (b-2)2^{b-1}$, $C = 2^{b-1} - 2$, and $w_{max} = 2^{b-2}$. As long as $k2^{b-1} \geqslant N$ and $kb \leqslant M$ are satisfied, we can derive the error bound: $\epsilon = ((b-2)2^{b-2} + 1/2)/N$ [11]. For the typical values of M, such as 256, 512, 1024, Table 2 lists several common combinations of k, b, the maximum capacity (N), and the theoretical error bound of quantile output (ϵ). Our experiments show that in practice the real error is less than this bound.

5.4.3 Adaptive Distributed Munro-Paterson Algorithm

In this subsection, we finalize the error bound of AMP. Suppose $M = 2^u$, for the lowest level non-leaf nodes in the sensor network, we set $k = 2^u$. The rank error of quantile output using AMP can be derived from a similar proof (similar to Property 2) described in [11].

M	k	b	N	ϵ
512	256	2	512	0.0010
	128	4	1024	0.0008
	64	8	8192	0.0469
1024	512	2	1024	0.0005
	256	4	2048	0.0042
	128	8	16384	0.0234
4096	1024	4	8192	0.0010
	512	8	65536	0.0059
	256	16	8388608	0.0273

Table 2. Total Memory (M), Number of Sensors (N), and Error (ϵ)

Property 3 *The rank error generated by the adaptive distributed Munro-Paterson algorithm is also at most* $\frac{W_c+W_f-C-1}{2} + w_{max}$, *$W_c$ is the weight sum of all* COLLAPSE *operations, W_f is the weight sum of all* FOLD *operations, C is the number of* COLLAPSE *operations, and w_{max} is the maximum weight among the buffers before the* OUTPUT *operation is invoked.*

Let's compare the error bound of AMP with DMP. In AMP, when the sketches are delivered up one level and the capacity has to be changed through the FOLD operation, k is shrunk to a half. Suppose the value of k in the root node is 2^v if we use AMP in a sensor network. We can also run DMP in the same sensor network and set k to 2^v and b to 2^{u-v}. In this way, we need not adjust the values k and b during the execution of DMP since the combination of $k = 2^v$ and $b = 2^{u-v}$ guarantees that the capacity is enough for this sensor network. Intuitively, the error bound of AMP is less than that of DMP. Let the sum of weights be W (DMP) and $W' = W'_c + W'_f$ (AMP) respectively.

Lemma 3 $W \geqslant W'$.

Proof. Suppose the root node in the DMP has l full buffers, $X_1, \ldots, X_l$ ($w(X_i) < w(X_{i+1})$) and the root node in AMP has l' full buffers, $X'_1, \ldots, X'_l$ ($w'(X_i) < w'(X_{i+1})$). We have $l = l'$ and $w(X_i) = w'(X'_i)$ because the decomposition of $\lfloor N/k \rfloor$ to the sum of several elements in the geometric series is unique (we can view this decomposition as the binary representation of $\lfloor N/k \rfloor$, where k is the same for both root nodes of DMP and AMP in our setting).

In DMP, $W = \sum_{i=1}^{l} f(X_i)$, each $f(X_i)$ can be calculated by the following recursive function:

$$f(1) = 1,$$
$$f(w) = w + 2f(w/2).$$

In AMP, $W' = \sum_{i=1}^{l} f'(X_i)$, each $f'(X_i)$ can be calculated by the following recursive function:

$$f'(1) = 1,$$
$$f'(w) = \begin{cases} w + 2f(w/2)(\textsc{Collapse}), \\ w + f(w/2)(\textsc{Fold}). \end{cases}$$

By induction, we can prove that $f'(w') \leqslant f(w)$ if $w' = w$. Overall, we have $W' \leqslant W$. ■

We can construct an operation tree similar to Figures 7 and 8, a COLLAPSE operation introduces two edges which merge two buffers having the same weight. A FOLD operation introduces one edge which fold one buffer into a half with weight doubled. The labels in the tree hold the weights of buffers. We illustrate the COLLAPSE and FOLD operations in Figure 9. As we can see, the weight sum of all nodes in Figure 9 is much less than that in Figure 8.

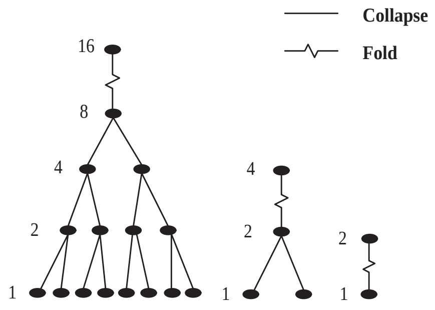

Figure 9. Collapse and Fold Tree

Based on a more sophisticated checking, we also can draw $W - C \geqslant W' - C'$, which finally shows that the rank error bound of DMP is the upper bound of the rank error which may be caused by AMP:

$$\epsilon = ((b-2)2^{b-2} + 1/2)/N \quad (1)$$

6 Incremental Maintenance

We already presented AMP which can construct a fresh sketch from the scratch. For applications which do not require real time quantile computation, we can run AMP periodically, e.g., in a daily basis. However, some applications may need up-to-date quantile information. It is more desirable to maintain the sketch in an incremental way if only a small number of sensors change their values.

Let us first check an example to see the potential issues in incremental updating. If the leaf sensor S_{21} (Figure 2) changes its value significantly, should we report the change to S_{11} and propagate this change to higher level nodes? The answer is if only S_{21} changes its value but other sensors do not change their values, it is unnecessary to propagate this change to upper level nodes since it only introduces additional error $\frac{1}{N}$. However, S_{21} does not know whether a large number of sensors in other branches already changed their values or not. Thus, we have to report any value change of S_{21}. We can either directly deliver the change to the root node through relay or change the sketch in S_{11} and propagate the sketch change to upper level nodes. The second approach is difficult to do since S_{01} does not keep track of the original sketch from S_{11}. We prefer the first approach, i.e., forwarding the pair $\langle v_{old}, v_{new}\rangle$ from the leaf sensor to the root node and let the root node, usually a server, to update its sketch.

We propose a *non-uniform* weight maintenance mechanism. By running AMP, the root node receives lots of sketch buffers from its child sensors. Elements in the same buffer share a uniform weight. For a buffer $\{e_1, e_2, \ldots, e_k\}$ with weight w, we construct k pairs $\langle e_i, w\rangle$ ($1 \leqslant i \leqslant k$) and put them into a histogram. We sort all pairs in this histogram by their first value e_i. We can query this histogram for quantiles with error bound given in Equation 1. When an updating pair $\langle v_{old}, v_{new}\rangle$ comes, we propose two operations:

- DELETE. Locate a pair $\langle v, w\rangle$ such that v is the closest one to v_{old} in H. Modify $\langle v, w\rangle$ to $\langle v, w-1\rangle$.
- INSERT. Insert a pair $\langle v_{new}, 1\rangle$ to H.

Through these two operations, the original elements in the same buffer may have non-uniform weights. Let the quantile error bound be ϵ and the histogram be H before the updating. After the updating, we query the new histogram, H', for ϕ-quantile. Suppose its real rank is within the interval $[(\phi - \epsilon')N, (\phi + \epsilon'')N]$. Say the ϕ-quantile is located in a bucket $\langle v_\phi, w\rangle$ of H'. We analyze the new error bound based on several cases.

Case 1: both v_{old} and v_{new} are less than or greater than v_ϕ. The above delete and insert operations will not affect the query result, i.e., the error bound of ϕ-quantile is still around ϵ.

Case 2: $v_{old} \leqslant v_\phi$ and $v_{new} \geqslant v_\phi$. Let the closest value to v_{old} be $\bar{v}_{old}$ in the old histogram H. We have $\bar{v}_{old} \leqslant v_\phi$, $\phi - \epsilon' \geqslant \phi + \frac{1}{N} - \epsilon$ and $\phi + \epsilon'' \leqslant \phi - \frac{1}{N} + \epsilon$. Thus, $\epsilon' \leqslant \epsilon - \frac{1}{N}$ and $\epsilon'' \leqslant \epsilon - \frac{1}{N}$. Overall, the new ϕ-quantile is still within the interval $[\phi - \epsilon, \phi + \epsilon]$.

Case 3: $v_{new} \leqslant v_\phi$ and $v_{old} \geqslant v_\phi$. Similar to Case 2, the new ϕ-quantile is also within the interval $[\phi - \epsilon, \phi + \epsilon]$.

In summary, the new error bound of ϕ-quantile computed from the new histogram is equal to or less than the old one. Thus, we can freely repeat the DELETE and INSERT operations. However, INSERT operation is costly because the root node has to maintain every sensor's new value in its histogram, and it grows larger and larger as time goes on. To construct a compact histogram, one solution is to replace the INSERT operation with the following one: Locate a bucket $\langle v_i, w\rangle$ such that v_i is the closest one to v_{new} in H and then modify $\langle v_i, w\rangle$ to $\langle v_i, w+1\rangle$. This method will increase the error of ϕ-quantile by $\frac{1}{N}$ if the ϕ-quantile is retrieved from the bucket $\langle v_i, w+1\rangle$. Assume the newly introduced error is ϵ_i. The overall error bound is $\epsilon + \epsilon_i$. Thus we need to monitor all such buckets which have increased their weights. If one of these buckets gets over the error bound that users desire, a new global recomputation is needed.

7 Experiments

We perform several experiments to compare the performance of three algorithms: equi-width, equi-depth, and AMP. Assume the maximum memory in each sensor is M, the sensor value range is [a, b].

In the equi-width algorithm (EquiWidth), we maintain a histogram with M buckets. We record the number of sensor values within the interval $[a + \frac{i-1}{M}(b-a), a + \frac{i}{M}(b-a)]$ in the i_{th} bucket ($1 \leqslant i \leqslant M$). We need to sum the counts in the corresponding buckets when we merge two equi-width histograms.

In the equi-depth algorithm (EquiDepth), we divide the memory evenly into two halves: one is used for an equi-depth histogram, the another is for a data buffer. When the buffer is full, we merge the equi-depth histogram with the buffer. The equi-depth histogram has $M' = M/6$ buckets (the histogram occupies $M/2$ space, each bucket needs three elements). We store three values: u_i, v_i, and c_i in each bucket B_i, where u_i is the lower bound of this bucket, v_i is the upper bound, and c_i is the total number of sensor values within the interval $[u_i, v_i]$. We also require $u_i \leqslant v_i$ and $v_i < u_{i+1}$. When we merge two equi-depth histograms, we redistribute the frequency in the old histogram to the new one by assuming uniform distribution in each bucket.

Dataset. We tested these three algorithms on a series of synthetic datasets. The datasets have several adjustable parameters as shown in Table 3: N is the total number of leaf sensors in the sensor network as shown in Figure 1; d and D define the minimum and maximum in-degrees of non-leaf sensors; Z is a parameter of Zipf distribution [15] which controls the data partitions. If $Z \to 0$, the tree (network partitions) shown in Figure 1 is balanced. If $Z \to \infty$, it is extremely unbalanced. The data generator works as follows: First, it sets N the number of leaf sensors covered by the root. It splits the root into m branches, where m is randomly selected within the range $[d, D]$. It divides N into m partitions that follow the Zipf distribution. This procedure repeats until it reaches a node which only covers 1 leaf sensor.

We choose Gaussian distribution as the sensors' data distribution and limit their minimum and maximum value.

abbr.	meaning
N	the total number of leaf sensors
d	the minimum in-degree of non-leaf sensors
D	the maximum in-degree of non-leaf sesnors
Z	Zipf parameter
σ	standard deviation of Gaussian distribution

Table 3. Parameters for Synthetic Datasets

Gaussian distribution supports the data model popular in natural environments where many sensor networks are deployed. In all the following experiments, we set the standard deviation $\sigma = 1$. Each dataset is mixed by two Gaussian distributions with different means.

Error Measurement. We are interested in the maximum real error of quantiles outputted by these three algorithms:

$$\epsilon = \max_{\phi \in [0,1]} \{|\phi - \hat{\phi}|\} \tag{2}$$

, where $\hat{\phi}N$ is the real rank of v_ϕ.

We first conduct experiments on two large scale sensor networks each consisting of 100k sensors: one is constructed in a balanced tree, the other is in an unbalanced tree. The Zipf parameter is set to 0 and 0.5 for the balanced and unbalanced partition respectively. Each non-leaf node has 2-10 in-degrees. We change the maximum memory (M) to check the maximum error defined in Equation 2. Figure 10 and 11 show the result. The error is reduced with the increased memory size. A steep error increase appears when the memory size is reduced from 256 to 64. It gives a clue to find the best trade-off between the maximum memory and the error rate. As depicted in the figure, AMP achieves better accuracy than EquiWidth and EquiDepth approaches in both balanced and unbalanced sensor networks except one case.

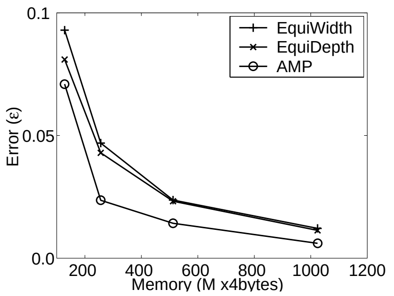

Figure 10. N100kM?d2D10Z0.0: Balanced

We also conduct experiments on small scale sensor networks each consisting of 1,000 sensors. These small scale

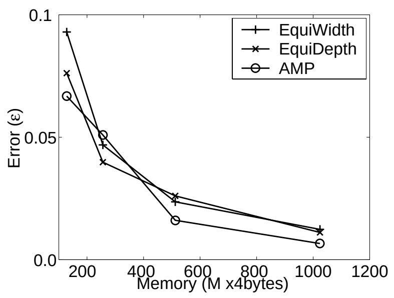

Figure 11. N100kM?d2D10Z0.5: Unbalanced

sensor networks are popularly used in many real applications. They may be deployed in a large geographic area. We also check two situations: a balanced network and an unbalanced network. We only allow 2-4 in-degrees in non-leaf nodes. Figure 12 and 13 show the result. In both cases, AMP has a competitive accuracy compared with EquiDepth. Both of them are better than EquiWidth.

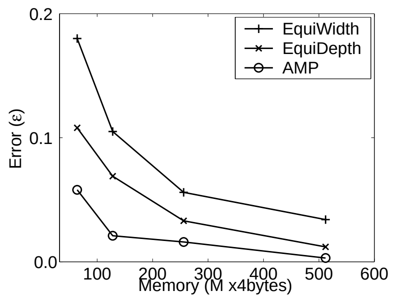

Figure 12. N1kM?d2D4Z0.0: Balanced

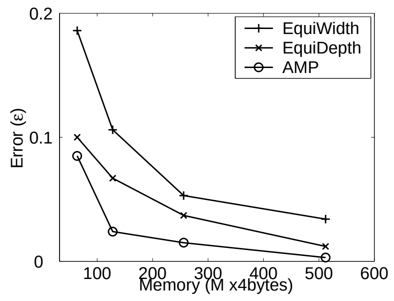

Figure 13. N1kM?d2D4Z0.5: Unbalanced

Next we change the network size to illustrate the scalability of each algorithm. The experimental results are shown in Figure 14. The parameters of the datasets in Figure 14 are N?M1024d2D10Z0.0. Figure 14 shows when the number of sensors increases, the accuracy of AMP deteriorates a little bit. However it still achieves the best accuracy among these three algorithms. When $N = 400k$, the maximum error of AMP is around 0.6% which is affordable in many applications. The error of EquiWidth and EquiDepth in the figure are just accidently similar in this dataset.

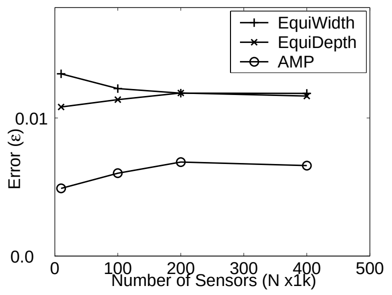

Figure 14. Accuracy vs. Network Size

It is well known that the equi-width histogram does not work well in very skewed data distributions. Due to the space limitation, we do not demonstrate those cases where the equi-width histogram definitely fails.

The experiments demonstrate that AMP performs better than EquiWidth and EquiDepth practically. Furthermore, AMP has an analytical error bound which can guide the setting of maximum memory based on the real requirement. This is one of the major advantages to use AMP instead of EquiWidth and EquiDepth.

8 Discussions on Further Extensions

In this paper, our proposed model is used when each sensor only submits a single number. However, in sensor networks, the value reported by a sensor may not be a single number, but rather an interval. As explained before, a sensor cannot afford to send the server a message whenever the value changes. A sensor may choose to report the new value only if it is significantly different from the original one. In this model, let v_i be the last reported value by sensor i. If the difference between the current value v_i' and v_i is greater than some threshold U_i, i.e., $|v_i - v_i'| \geq U_i$, then the sensor will notify the server the new value. Otherwise, the sensor will not do so. Under this scheme, we have an interval (l_i, u_i) for each senor value where l_i and u_i are the lower and upper bound of the interval for the value. Instead of finding a single quantile value, we are finding the minimum and maximum bound of the quantile intervals. For instance, 75% quantile is an interval (l, u) where l is the 75% quantile for all l_i and u is the 75% quantile for all u_i.

When the interval range is the same for all sensors, (i.e., $\forall i, j\ u_i - l_i \equiv U$, we can adapt our algorithm easily. We first build a quantile sketch on the center of all intervals. To find the ϕ-quantile of the lower bound of the interval, we use the following method. Let v_ϕ be the quantile computed based on the sketch. Since there is an error bound ϵ of the quantile, we will look for $\phi - \epsilon$ quantile. We know for sure that $v_{\phi-\epsilon}$ is less than or equal to the real ϕ-quantile of the center value. Then $v_{\phi-\epsilon} - U$ will be the lower bound of ϕ-quantile. For the same reason, $v_{\phi+\epsilon} + U$ will be the upper bound.

9 Conclusion

We have studied the problem of computing quantiles over sensor networks. Unlike centralized systems, the sensor network is usually highly distributed, with limited memory, and being update frequently. We devise an efficient algorithm to generate the sketch for quantile computing. Since the memory is limited and the network size may grow or shrink dynamically, we develop the technique of dynamic buffer allocation to adapt to the various sizes of network and memory space. Since it is energy inefficient to re-compute the entire sketch once a few sensors change their values, an incremental update algorithm is developed, using a non-uniform weight technique, to reflect the small amount of changes in the sensor network without incurring the global sketch recomputation. Our comprehensive experiments demonstrate the advantages of our quantile computing algorithm. Further, our approach provides an analytical error bound which can guide the setting of maximum memory based on the users' requirement.

There are many interesting research problems that should be pursued further. The Munro-Paterson algorithm is not the most efficient algorithm in quantile computation over stream data. Both Manku and Greenwald-Khanna algorithms [11, 9] perform better. it is a challenging task to adapt these two algorithms in the setting of sensor networks. Besides, the study in this paper only considers memory and bandwidth consumption, rather than the computation cost, which is also critical to the sensors' durability.

References

[1] B. Babcock, M. Datar, R. Motwani, and L. O'Callaghan. Maintaining variance and k–medians over data stream windows. In *Proc. 18th ACM Symp. Principles of Database Systems (PODS'99)*, San Diego, California, June 2003.

[2] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7:448–461, 1973.

[3] S. Chaudhuri, R. Motwani, and V. Narasayya. Random sampling for histogram construction: how much is enough? In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98)*, pages 436–447, Seattle, WA, June 1998.

[4] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *Proc. of Thirteenth Annual ACM-SIAM Symp. on Discrete Algorithms (SODA'02)*, pages 635–644, San Francisco, CA, January 2002.

[5] D. Dor and U. Zwick. Finding the αn^{th} largest element. *Combinatorica*, 16:41–58, 1996.

[6] D. Estrin, R. Govindan, and J. Heidemann. Embedding the internet. *Comunications of the ACM*, 43(5):39–41, 2000.

[7] P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental mainenance of approximate histograms. In *Proc. 1997 Int. Conf. on Very Large Data Bases (VLDB'97)*, pages 466–475, Athens, Greece, August 1997.

[8] P. B. Gibbons and S. Tirthapura. Distributed streams algorithms for sliding windows. In *Proc. of ACM Symp. on Parallelism in Algorithms and Architectures (SPAA'02)*, Winnipeg, Canada, August 2002.

[9] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *Proc. 2001 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'01)*, pages 58–66, Santa Barbara, CA, May 2001.

[10] M. Horton, D. Culler, K. Pister, J. Hill, R. Szewczyk, and A. Woo. Mica, the commercialization of microsensor motes. *Sensor Magzine*, pages 40–48, 2002.

[11] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98)*, pages 426–435, Seattle, WA, June 1998.

[12] J. I. Munro and M. S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12:315–323, 1980.

[13] M. S. Paterson. Progress in selection. In *Scandinavian Workshop on Algorithm Theory*, pages 368–379, Reykjavk, Iceland, July 1996.

[14] J. Zhao, R. Govindan, and D. Estrin. Computing aggregates for monitoring wireless sensor networks. In *First IEEE International Workshop on Sensor Network Protocols and Applications*, 2003.

[15] G. K. Zipf. *Human Behaviour and the principle of least effort*. Addison-Wesley, 1949.