Available online at www.sciencedirect.com

ScienceDirect

Theoretical Computer Science 370 (2007) 254–264

Theoretical Computer Science

www.elsevier.com/locate/tcs

A note on efficient aggregate queries in sensor networks[☆]

Boaz Patt-Shamir[*,1]

Department of Electrical Engineering, Tel Aviv University, Tel Aviv 69978, Israel

Received 15 May 2006; received in revised form 30 October 2006; accepted 31 October 2006

Communicated by D. Peleg

Abstract

We consider a scenario where nodes in a sensor network hold numeric items, and the task is to evaluate simple functions of the distributed data. In this note we present distributed protocols for computing the median with sublinear space and communication complexity per node. Specifically, we give a deterministic protocol for computing median with polylog complexity and a randomized protocol that computes an approximate median with polyloglog communication complexity per node. On the negative side, we observe that any deterministic protocol that counts the number of distinct data items must have linear complexity in the worst case.

Keywords: Sensor networks; Communication complexity; Median computation; Aggregate queries

1. Introduction

Most nodes in a sensor network must work with extremely constrained resources, which means that they must be very frugal in terms of the number of bits they communicate, the number of memory bits they require and the amount of processing they do. Typically, the largest power consumption is due to communication (sending or receiving a small message may consume as much power as a thousand processing cycles). Therefore, it is highly desirable to reduce the amount of data transmitted, possibly by doing more local processing. This basic motivation underlies sensor network systems like TAG [9], Cougar [15], and others (see, e.g. [16]). Intuitively, the idea is to view sensors as sources of data, and the goal of the system is to support aggregate queries formed in an SQL-like language. The setting envisaged is that a root node (connected to the user entity) issues queries regarding the data collected by the sensors, and the sensors collaborate in trying to generate an accurate response. In TAG, it is proposed to use a spanning tree of the network and let the sensors do some simple local aggregations so as to avoid sending all raw data to the root. In particular, in [9] the operations of finding the *maximum*, *minimum*, *count*, *sum* and *average* are identified as aggregate queries that can be carried out efficiently on a spanning tree. Finding the *median* and counting the number of *distinct elements* are explicitly classified in [9] as aggregates that require linear space and communication.

[☆] A preliminary version of this paper appeared in *23rd ACM Symp. on Principles of Distributed Computing*, July 2004.

[*] Tel.: +972 3 640 7036; fax: +972 3 643 6392.
E-mail address: boaz@eng.tau.ac.il.

[1] Research done while visiting HP Cambridge Research Lab, One Cambridge Center, Cambridge, MA 02142, USA.

0304-3975/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2006.10.032

Subsequent work did not quite settle the question of median and distinct elements. In [2], to improve robustness, the spanning tree condition is relaxed to allow for arbitrary duplication by the communication subsystem; [2] gives randomized algorithms that solve efficiently the aggregates of counting, sum and average. Attacking the robustness of computing these aggregates from another angle, Zhao et al. [16] finely-tune protocols for building spanning trees. Singh and Prasanna [14] give an algorithm for median computation in single-hop networks (i.e. all hear all), in which each node transmits only $O(\log N)$ bits, where N is the number of items in the system. Note that each node in in the algorithm of [14] receives $O(N \log N)$ bits. The best deterministic median computation in general sensor networks to date required some node to communicate $\Omega(N)$ bits in the worst case. Regarding randomized algorithms, the best known result is [6], where nodes are assumed to communicate by gossip; [6] presents an algorithm that finds, with high probability, the exact median (or any other order statistics) using $O((\log N)^3)$ bits of communication per node, assuming that the network has the best possible "diffusion speed" (a concept closely related to "mixing time").

Our results. In this paper we show that very little communication suffices to find the median element, but obtaining an exact count of the number of distinct elements requires a lot of communication. Specifically, we present the following results. First, for completeness, we give a very simple deterministic algorithm that computes the median value such that each node transmits and receives only $O((\log N)^2)$ bits, assuming that data items can be represented using $O(\log N)$ bits. (It has been brought to our attention that a very similar algorithm appears in [11].) Our main result is an algorithm that computes an *approximate* median, in which each node transmits $O((\log\log N)^3)$ bits. (Note that just sending a single item costs $\Omega(\log N)$ bits in the exact case!) Both algorithms are completely indifferent to the underlying communication mechanism: the first bound only assumes the existence of a deterministic protocol for counting with communication complexity $O(\log N)$ bits, and the second requires a randomized protocol for approximate counting with communication complexity $O(\log\log N)$ bits. Such protocols are known for various communication models (see, e.g. [13,3]). On the negative side, we give a simple reduction that proves a conjecture from [9]. Specifically, we show that computing the exact number of distinct elements in the data set indeed requires linear communication in the worst case, even if the algorithm is randomized and allowed to err with some probability. This result should be contrasted with known methods of approximating the number of distinct elements, where each node needs only to send $O(\log\log N)$ bits (see, e.g. [1,3]).

Concurrent results by others. At about the same time our results were announced in a conference [12], another algorithm for computing the median deterministically in sensor networks was discovered by Greenwald and Khanna [4], using the same communication cost model we use here. To compute the median (or any order statistic), their algorithm requires $O((\log N)^4)$ communication bits per node, as compared to our $O((\log N)^2)$ deterministic protocol. The algorithm in [4], however, can compute deterministically, after one pass over the data and $O((\log N)^3)$ communication bits, any *approximate* order statistic. In contrast, our randomized approximate algorithm computes only a single order statistic, but it does it using exponentially fewer communication bits.

Another result published concurrently with our initial publication is by Nath et al. [10], who advocate using "order- and duplicate-insensitive synopses" (similarly to [2]). Among other things, they propose using their tool to solve the median problem approximately by uniform sampling; in our terms, the complexity of that algorithm is $\Omega(\log N)$ communication bits per node, as opposed to our polyloglog approximate algorithm (both algorithms are randomized).

2. Model and preliminaries

2.1. System model

The system is modelled as a set of *nodes* denoted V, of which one is called *root*. We do not make any specific assumption about the way communication is carried out: all we require is that the root can initiate some protocols and get back the results when a protocol terminates. The communication mechanism will be abstracted by the assumptions we make about the existence of protocols for primitive tasks in Section 2.2 below.

To formalize the problems we address, we assume that each node holds a multiset of non-negative integers called *input items*. The root node is assumed to have, in addition, a special *output register*. To simplify notation, we will be mainly concerned with the case where each node $i \in V$ holds a single input item, denoted x_i.[2] The collection of all input items is denoted by X. In general, X is a multiset. The cardinality of X including multiplicities is denoted by $|X|$, and by convention we also use the notation $N \overset{\text{def}}{=} |X|$. We denote the maximal possible value of X by $\overline{X}$, and assume $\overline{X}$ is known. We further assume that the input values are polynomial in N, i.e. that $\log \overline{X} = O(\log N)$. (All logarithms in this paper are to base 2.)

Let $\mathcal{X}$ denote all possible values the input multiset can take. A *task* is a function from $\mathcal{X}$ to the set of non-negative integers denoted $\mathbb{Z}^+$. A protocol is said to *solve* a given task $f : \mathcal{X} \to \mathbb{Z}^+$ if for all $X \in \mathcal{X}$, when the protocol is given input X, the value written to the output register at the root node is $f(X)$.

The complexity measures we use to evaluate the performance of a given protocol are worst-case measures per node. We will be mainly concerned with the communication complexity of a protocol, defined to be the maximum, over all inputs, of the number of bits *transmitted and received* by any node. We stress that our communication complexity measure is individual, in the sense that we measure the maximal number of bits communicated by any single node. The space complexity of a protocol is the maximum, over all inputs, of the number of bits used by any node during the execution of the protocol. The processing complexity of a protocol is defined to be the maximum, over all inputs, of the total number of computation steps taken by any node until output is made. For the purpose of accounting for computational complexity, we assume that each node is a classical RAM machine with access to an infinite tape of random bits. We use $P_A(N)$ to denote the processing complexity of an algorithm A as a function of the number of input items N. Similarly, $C_A(N)$ denotes the communication complexity of A, and $S_A(N)$ denotes its space complexity. The complexities of all protocols considered in this paper are nondecreasing functions of N.

2.2. *Primitive protocols: Max, counting, approximate counting*

Possibly the simplest tasks in our framework is computing $\max(X)$, $\min(X)$, and $|X|$. We call these task MAX, MIN and COUNT, respectively. The broadcast-convergecast protocol [13] with the natural corresponding aggregation rules gives the following trivial result.

Fact 2.1. *There exist protocols that compute* MAX, MIN *and* COUNT *with communication complexity $O(\log N)$, space complexity $O(\log N)$, and processing complexity $O(1)$.*

We remark in order to get the stated complexity bounds, one usually uses a bounded-degree spanning tree of the network [9] (bounded degree is required to maintain low individual communication complexity).

A much more interesting fact is that $|X|$ can be approximated with exponentially smaller communication complexity. Intuitively, the basis for the best approximate counting protocols is the following fact [1,3,7]: if each node samples an independent geometric random variable with parameter 1/2 (say, by counting random bits until the first "1" occurs), then the maximum of these samples is about $\log N$. Therefore, by using the MAX algorithm over these samples (each of which is $O(\log \log N)$ bits long), we get an estimate of the count while incurring only $O(\log \log N)$ communication complexity.

To formalize this result, we use the following concept.

Definition 2.1. A protocol APX_COUNT is said to solve the *α-counting* problem with variance σ^2 if for all possible inputs X of size N, its output is a random variable APX_COUNT(X) such that:

- $\frac{1}{N}|\mathbf{E}\,[\text{APX\_COUNT}(X)] - N| \le \alpha$, and
- $\frac{1}{N^2}\mathbf{Var}\,[\text{APX\_COUNT}(X)] = \sigma^2$,

for some $\alpha, \sigma \ge 0$.

Durand and Flajolet [3] analyze an algorithm based on the above idea, and prove a result which, cast in our framework, can be stated as follows.

Fact 2.2. *For any given parameter m, there exists an α-counting protocol with communication and processing complexity $O(m \log \log N)$. The protocol has $\alpha < 10^{-6}$, and its variance σ^2 satisfies $\sigma \le \beta_m/\sqrt{m} + 10^{-6} + o(1)$ for some sequence of constants $\beta_m \to 1.298$.*

[2] We consider nonsingleton local inputs only in Section 5.

Using the hash value of an item as the source of random bits, the algorithm of [3] can be used to count the number of distinct elements with small space (in fact, this is the intended use in [1,3]). Moreover, in this case the requirement for a spanning tree is not necessary [2].

2.3. Order statistics and median

Finally, we define the median problem and its generalization, the order-statistics problem. We use the following notation (recall that X is a multiset of integers):

Notation 2.2. *For any number y, $\ell_X(y)$ denotes the number of items $x_i \in X$ strictly smaller than y, i.e., $\ell_X(y) = |\{x_i \in X \mid x_i < y\}|$.*

We usually omit the subscript when X is clear from the context. Using the above notation, we define the following tasks.

Definition 2.3. For any given $1 \le k \le N$, a *k-order statistics of X*, denoted $\textsc{os}(X, k)$ is a number y such that $\ell(y) < k$ and $\ell(y+1) \ge k$. The *median* of X is defined by $\textsc{median}(X) \overset{\text{def}}{=} \textsc{os}(X, N/2)$.

We extend the definitions of order statistics and median to allow for approximations.

Definition 2.4. Let $0 \le \alpha \le 1$ and $0 < \beta \le 1$ be given parameters. Given an integer k, we say that a number y is a k *(α, β)-order statistics of X*, denoted $\textsc{apx\_os}(X, k)$, if there exists a number y' such that
(1) $\ell(y') < k(1+\alpha)$ and $\ell(y'+1) \ge k(1-\alpha)$.
(2) $\frac{1}{\max(X)}|y - y'| \le \beta$.
An $N/2$ (α, β)-order statistics of X is called an (α, β)-median of X.

Both parameters are related to the density of input values in the vicinity of the order statistics (or median). The α parameter controls the allowed error in terms of rank, while the β parameter controls the allowed error in terms of value.

3. Efficient deterministic median and order statistics

In this section we give a deterministic algorithm to compute the median with communication complexity of $O((\log N)^2)$ bits. The algorithm extends directly to compute any desired order statistics; we explain below the median algorithm in the interest of clarity. We remark that a similar algorithm appears in [11].

First, we define the following convenient generalization of the counting problem.

3.1. The COUNTP protocol

The COUNTP protocol takes a predicate P as an input argument, and returns the number of elements x for which $P(x)$ is true. For example, the result of $\textsc{countP}(X, \text{“} > 5\text{”})$ is the number of elements in X whose value is strictly more than 5, and $\textsc{countP}(X, \textsc{true})$ is a long way to write $\textsc{count}(X)$.

If a predicate P is locally computable, then any implementation of COUNT can be used to implement $\textsc{countP}(P)$, by letting COUNT run only on the elements that satisfy P. It is important to note that in order for the asymptotic complexity of COUNTP to remain comparable to the underlying COUNT protocol, we need to ensure that P can be represented in $O(C_{\textsc{count}}(N))$ bits, and that its local processing and space complexities are $O(P_{\textsc{count}}(N))$ and $O(S_{\textsc{count}}(N))$.

3.2. Deterministic median algorithm

We now specify the median algorithm. The idea is extremely simple: we count the number of elements in a specified prefix of the range of values; doing a binary search on the upper bound of the prefix, we can find the median quickly. Pseudocode for the root node is given in Fig. 1. Nonroot nodes just follow the protocols (MIN, MAX, COUNT and COUNTP) initiated by the root.

```
Algorithm MEDIAN($X$)
1     Invoke protocols to compute $m \leftarrow$ MIN($X$), $M \leftarrow$ MAX($X$) and $n \leftarrow$ COUNT($X$).
2     $y \leftarrow \frac{M+m}{2}$; $z \leftarrow 2^{\lceil \log(M-m) \rceil - 1}$.      y is an integer or an integer $+\frac{1}{2}$
3     while $z > \frac{1}{2}$ do                                                       binary search
3.1       Invoke protocol to compute $c(y) \leftarrow$ COUNTP($X$, " $< y$").
3.2       if $c(y) < \frac{n}{2}$ then $y \leftarrow y + \frac{z}{2}$ else $y \leftarrow y - \frac{z}{2}$.
3.3       $z \leftarrow \frac{z}{2}$.
4     if $y \in \mathbb{Z}$ then output $y$
4.1       else if COUNTP($X$, " $< \lceil y \rceil$ ") $< n/2$ then output $\lceil y \rceil$ else output $\lfloor y \rfloor$.
```

Fig. 1. Algorithm for computing the median using COUNTP: code for the root node.

3.3. Analysis

We first prove the following loop invariant.

Lemma 3.1. *Let μ denote the median of X. Then whenever Line 3 of the code is executed, $\mu \in [y-z, y+z]$.*

Proof. By induction on the execution. The basis of the induction follows from Lines 1 and 2: by assumption about the correctness of the primitive protocols, we have that $m = \min X$ and $M = \max X$. Since $z = 2^{\lceil \log(M-m) \rceil - 1} \geq \frac{M-m}{2}$ and $y = \frac{M+m}{2}$, we have that in the first time Line 3 is executed, $y - z \leq m$ and $y + z \geq M$, and therefore, $\mu \in [y–z, y+z]$. For the induction step, let y, z and y', z' denote the value of the variables before and after executing the iteration. Consider first the case where $c(y) < n/2$. In this case, by correctness of the COUNTP protocol, $\ell(y) < n/2$. Then by definition, $\mu \in [y, M]$. Since by induction $\mu \in [y-z, y+z]$, we get that $\mu \in [y, y+z]$. It is sufficient to prove that $[y, y+z] \subseteq [y'-z', y'+z']$. This is true because by Line 3.2, $y' - z' = (y + z/2) - (z/2) = y$, and $y' + z' = y + (z/2) + (z/2) = y + z$. The case $c \geq n/2$ is similar. ∎

The following theorem summarizes the properties of the algorithm.

Theorem 3.2. *Algorithm* MEDIAN(X) *outputs the median of X with communication complexity $O((\log N)^2)$, processing complexity $O(\log N)$ and space complexity $O(\log N)$.*

Proof. Let μ denote the median of X. By Lemma 3.1 and the condition of Line 3, when Line 4 is reached, $\mu \in [y - 1/2, y + 1/2]$. Hence, if y is integer, then $y = \mu$. Otherwise, $\mu \in \{\lfloor y \rfloor, \lceil y \rceil\}$. Line 4.1 finds which case is it directly. Regarding complexity, note that the while loop is executed exactly $\lceil \log(M - m) \rceil + 1 = O(\log N)$ times. By Fact 2.1, each invocation of COUNTP has communication complexity $O(\log N)$ bits, since the predicate requires $O(\log N)$ bits to describe, and the result requires $O(\log N)$ bits. The processing and space complexities are trivially $O(\log N)$. ∎

3.4. Algorithm for order statistics

We note that it is straightforward to extend the algorithm to answer arbitrary k-order statistics queries: just replace the $n/2$ expression with k in Lines 3.2 and 4.1.

4. Approximate median algorithm

There are two main ideas in the approximate median computation algorithm we now describe. First, we replace the deterministic counting protocol with an approximate counting protocol; this forces us to develop a version of binary search that can tolerate errors. To further reduce the complexity, we change the target of the binary search: instead of looking for the *value* of the median, we look for the *length* (i.e. the logarithm) of that value.

To help exposition, let us first present the algorithm without the reduction of input items lengths. We assume the existence of a protocol APX_COUNT for α-counting with variance σ^2 such that $\alpha \leq \sigma/2$ (cf. Fact 2.2). Using such a

```
Algorithm APX_MEDIAN($X, \epsilon$)                                    $\epsilon$ is desired success probability
1       Invoke protocols to compute $m \leftarrow \min(X)$, $M \leftarrow \max(X)$.
2       Let $q \leftarrow \frac{\log(M-m)}{\epsilon}$; $n \leftarrow$ REP_COUNTP($\lceil 2q \rceil$, TRUE)       $q$ is convenient shorthand
3       $y \leftarrow \frac{M+m}{2}$; $z \leftarrow 2^{\lceil \log(M-m) \rceil - 1}$.
4       while $z > \frac{1}{2}$ do                                                    binary search
4.1         $c(y) \leftarrow$ REP_COUNTP($\lceil 32q \rceil$, "$< y$").                  $\sigma^2$ is variance of APX_COUNT
4.2         if $c(y) < n(\frac{1}{2} - \alpha_c - \sigma)$ then $y \leftarrow y + \frac{z}{2}$
4.2.1           else if $c(y) \geq n(\frac{1}{2} + \alpha_c + \sigma)$ then $y \leftarrow y - \frac{z}{2}$ else output $\lfloor y \rfloor$ and halt.
4.3         $z \leftarrow \frac{z}{2}$.
5       output $\lfloor y \rfloor$.

Subroutine REP_COUNTP($r, P$)
R1      Invoke $r$ independent instances of APX_COUNTP($P$)
R2      Return the average result.
```

Fig. 2. Algorithm for computing the median using COUNTP, assuming that APX_COUNT is an α-counting protocol with $\alpha = \alpha_c$ and variance σ^2 such that $\alpha_c < \sigma/2$.

protocol as a black box, we define a protocol REP_COUNTP, that takes a repetition parameter r, and a predicate P. Protocol REP_COUNTP simply applies APX_COUNT r times to count the elements that satisfy the predicate P. Pseudo code for the root protocol is presented in Fig. 2; nonroot nodes participate in the counting protocols initiated by the root.

4.1. Analysis

Throughout the analysis, we shall assume that APX_COUNT is an α-counting protocol with $\alpha = \alpha_c$ and variance σ^2 such that $\alpha_c < \sigma/2$. In the interest of brevity, let us denote an interval $[y-z, y+z]$ by $[y \pm z]$.

We start with the basic property of REP_COUNTP.

Lemma 4.1. *Let P be a predicate on X, and let r be a positive integer. Suppose that $|\{x \in X \mid P(x)\}| = g$ for some g. Then for any $t > 0$ we have that:*

$$\Pr\left[|\text{REP\_COUNTP}(r, P) - g| \geq t + \alpha_c g\right] \leq \frac{\sigma^2}{rt^2}.$$

Proof. REP_COUNTP applies APX_COUNTP independently r times. By definition of α-counting protocol,

$$|\mathbf{E}\left[\text{REP\_COUNTP}(r, P)\right] - g| \leq \alpha_c g,$$

and $\mathbf{Var}\left[\text{REP\_COUNTP}(r, P)\right] = \frac{\sigma^2}{r}$. The result follows from Chebychev's Inequality. ∎

Corollary 4.2. *After executing Line 2,*

$$\Pr\left[\frac{|N-n|}{N} > \alpha_c + \sigma\right] < \frac{1}{2q}.$$

Proof. Follows from Lemma 4.1 when we let $t = \sigma$ and $r = 2q$. ∎

The following lemma analyzes an iteration of the while loop that did not halt.

Lemma 4.3. *Let μ denote the median of X, and consider any execution of an iteration of the while loop that did not halt. Let y, z and y', z' denote the value of the variables before and after the iteration, respectively. If $\mu \in [y \pm z]$ and $\frac{|N-n|}{N} \leq \alpha_c + \sigma$, then $\Pr\left[\mu \in [y' \pm z']\right] \geq 1 - \epsilon/2\log(M-m)$.*

Proof. Suppose first that $\mu \in [y, y+z]$, i.e., $\ell(y) < N/2$, and consider the probability that the algorithm takes a "wrong turn", i.e. it assigns $y' \leftarrow y - z/2$: By Line 4.2.1, this happens only when $c(y) \geq n(\frac{1}{2} + \alpha_c + \sigma)$. But

$$\begin{aligned}\mathbf{P}\left[c(y) \geq n\left(\tfrac{1}{2} + \alpha_c + \sigma\right) \,\middle|\, \begin{matrix}\ell(y) < N/2 \\ n > N(1 - \alpha_c - \sigma)\end{matrix}\right] &\leq \Pr\left[c(y) \geq N\frac{1 + \alpha_c + \sigma}{2} \,\middle|\, \ell(y) < \frac{N}{2}\right] \\ &\leq \Pr\left[c(y) \geq \ell(y)\left(1 + \frac{\alpha_c + \sigma}{2}\right)\right] \\ &\leq \Pr\left[|c(y) - \ell(y)(1 + \alpha_c)| \geq \frac{\sigma - \alpha_c}{2}\right]. \qquad (1)\end{aligned}$$

Now, assuming $\sigma > \alpha_c/2$, we get from Eq. (1) and Lemma 4.1 that the probability that $y' \leftarrow y - z/2$ when $\frac{|N-n|}{N} \leq 1 + \alpha_c + \sigma$ and $\ell(y) < N/2$ is at most $\frac{16}{q} < \frac{\epsilon}{2\log(M-m)}$, and the lemma holds in this case. Similarly, if $\mu \in [y-z, y]$ then $\ell(y) \geq N/2$, and applying Lemma 4.1 and the assumption that $n < N(1 + \alpha_c + \sigma)$, we get that in this case too,

$$\Pr\left[c(y) \leq n\left(\frac{1}{2} - \alpha_c - \sigma\right) \,\middle|\, \ell(y) \geq \frac{N}{2}\ , n < N(1 + \alpha_c + \sigma)\right] \ \leq \ \frac{\epsilon}{2\log(M-m)},$$

and the result follows. ∎

The next lemma analyzes the case of termination in the middle of an iteration of the while loop.

Lemma 4.4. *If the algorithm halts at Line* 4.2.1, *and if* $\frac{|N-n|}{N} \leq \alpha_c + \sigma$, *then the output is an* (α, β)*-median with probability at least* $1 - \epsilon/2$, *for* $\alpha = 3\sigma$ *and* $\beta = 1/\overline{X}$.

Proof. If the algorithm halts at Line 4.2.1, then $n(\frac{1}{2} - \alpha_c - \sigma) < c(y) < n(\frac{1}{2} + \alpha_c + \sigma)$, and hence $\frac{N - 3\alpha_c - 3\sigma}{2} < c(y) < \frac{N + 3\alpha_c + 3\sigma}{2}$. Since by assumption $\alpha_c < \sigma/2$, we get from Lemma 4.1 that $\Pr\left[|\ell(y) - \frac{n}{2}| \geq 3\sigma\right] < \frac{1}{9q} < \frac{\epsilon}{2}$. For β, note that Line 4.2.1 changes y by at most 1. ∎

We can now summarize the properties of APX_MEDIAN.

Theorem 4.5. *Suppose that protocol* APX_COUNT *is an* α*-counting protocol with* $\alpha = \alpha_c$ *and variance* σ^2 *such that* $\alpha_c < \sigma/2$. *The output of Algorithm* APX_MEDIAN(X, ϵ) *is an* (α, β)*-median with probability at least* $1 - \epsilon$ *for* $\alpha = 3\sigma$ *and* $\beta = 1/N$. *The communication complexity of* APX_MEDIAN *is* $O((\log \max(X))^2 C_A(N)/\epsilon)$, *where* $C_A(N)$ *is the communication complexity of* APX_COUNT.

Proof. Let μ denote the median of X. Let $\mathcal{N}$ denote the event that $\frac{|N-n|}{N} \leq \alpha_c + \sigma$. We claim, by induction on the number of iterations of the while loop, that if $\mathcal{N}$ holds, and if $i - 1$ iterations have completed without halting, then before the ith iteration, $\mu \in [y \pm z]$ with probability at least $1 - \frac{(i-1)\epsilon}{2\log(M-m)}$. First, note that if $\mathcal{N}$ holds before an iteration, then it also holds after the iteration, so we need only to verify claim about μ. For the basis for the induction, $i = 1$, note that before the first iteration we have $\mu \in [y \pm z]$ because $[y \pm z] \supseteq [m, M]$ by Line 3 of the code. For the inductive step, consider the $(i+1)$st iteration. Let y, z and y', z' denote the value of the variables before and after the execution of the iteration. The induction is completed since by Lemma 4.3 we have

$$\begin{aligned}\mathbf{P}\left[\mu \in [y' \pm z'] \mid \mathcal{N}\right] &= \Pr\left[\mu \in [y' \pm z'] \mid \mathcal{N}, \mu \in [y \pm z]\right] \cdot \Pr\left[\mu \in [y \pm z] \mid \mathcal{N}\right] \\ &\geq \left(1 - \frac{\epsilon}{2\log(M-m)}\right)\left(1 - \frac{(i-1)\epsilon}{2\log(M-m)}\right) \\ &\geq 1 - \frac{i\epsilon}{2\log(M-m)}\ .\end{aligned}$$

Now, if the algorithm halted before completing $\log(M-m)$ iterations, then by Lemma 4.4 we have that for $\alpha = 3\sigma$ and $\beta = 1/N$, the output is an (α, β)-median with probability at least $1 - \epsilon/2$, provided that $\mathcal{N}$ holds. Otherwise, by applying the inductive claim with $i = \log(M-m)$, we obtain that the output is an (α, β)-median with probability at least $1 - \epsilon/2$. Since by Corollary 4.2, $\Pr[\mathcal{N}] \geq 1 - \frac{1}{2q} \geq 1 - \epsilon/2$, we can conclude that APX_MEDIAN(ϵ) computes an (α, β)-median with probability at least $1 - \epsilon$, for $\alpha = 3\sigma$ and $\beta = 1/N$.

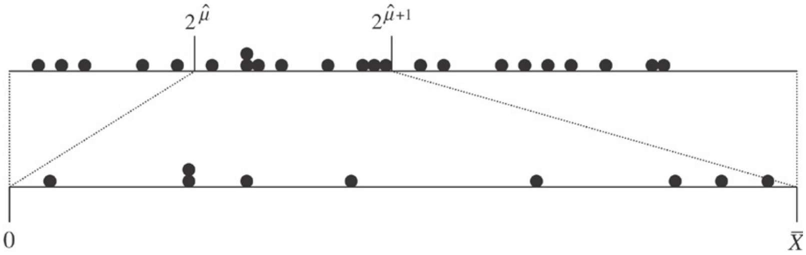

Fig. 3. Above: An example of the input values on the real line. Below: The values after the adjustment step (Line 3.2 of the algorithm). The algorithm "zooms" into the interval that contains the median.

To account for the communication complexity, note that each iteration of the while loop contains $O(\frac{\log(M-m)}{\epsilon}) = O(\log \max(X)/\epsilon)$ invocations of APX_COUNT, and that there are at most $O(\log(M-m)) = O(\log \max(X))$ iterations. The communication complexity of the MIN and MAX protocols invoked at Line 1 is $O(\log \max(X))$. ▮

Approximate k-order statistics. As was the case with the deterministic algorithm, Algorithm APX_MEDIAN can be easily extended to answer arbitrary k-order statistics queries with the same complexity.

Theorem 4.6. *Suppose that protocol* APX_COUNT *is an α-counting protocol with $\alpha = \alpha_c$ and variance σ^2 such that $\alpha_c < \sigma/2$. Then there exits an Algorithm* APX_OS(X, ϵ, k) *that computes the k (α, β)-order statistics with probability at least $1-\epsilon$ for $\alpha = 3\sigma$ and $\beta = 1-1/N$. The communication complexity of* APX_OS *is $O((\log \max(X))^2 C_A(N)/\epsilon)$, where $C_A(N)$ is the communication complexity of* APX_COUNT.

Proof Sketch. The algorithm is obtained from APX_MEDIAN by replacing the "$\frac{1}{2}$" expressions in Lines 4.2 and 4.2.1 by "$\frac{k}{N}$". ▮

4.2. Approximate median with polyloglog complexity

To further reduce the complexity of computing median, we use the following simple idea: Each node i will use, instead of its original input value x_i, the number $\hat{x}_i = \lfloor \log x_i \rfloor$. Now, since $\max_{i \in V}(\hat{x}_i) = O(\log N)$, by using an approximate counting algorithm with polyloglog complexity, we get from Theorem 4.5 that the complexity of the resulting algorithm for computing an (α, β)-median is polyloglog. However, the precision (expressed by β) of that algorithm is only constant. Using recursion, we can improve β almost arbitrarily. Specifically, suppose that the median of the $\hat{x}_i$ values is $\hat{\mu}$. Then the median of the original x_i values must be between $2^{\hat{\mu}}$ and $2^{\hat{\mu}+1} - 1$. We can therefore discard all items whose value is outside that range. Then, scaling the remaining items to be in the range $[1, \overline{X}]$ (recall that $\overline{X}$ is a known upper bound on the values in X), and setting $k \leftarrow \frac{N}{2} - \left|\left\{x_i \mid x_i < 2^{\hat{\mu}}\right\}\right|$, we can apply APX_OS to the new values. See Fig. 3 for a schematic example. Repeating this process for $O(\log \frac{1}{\beta})$ stages, we achieve the desired precision of β. Pseudo-code for the algorithm is presented in Fig. 4. Note that once an intermediate result is computed (denoted $\hat{\mu}^{(j)}$ in the jth iteration) and broadcast to nodes (Line 3.1), each node can locally find whether it is active in the next iteration, and if so, what is the scaled value it should use (Lines 3.2–3.3). The analysis is similar to the analysis of Algorithm APX_MEDIAN, and we only sketch it below.

Theorem 4.7. *Let A be an α-counting algorithm with communication complexity $C_A(N)$, $\alpha = \alpha_c$ and variance σ^2 such that $\alpha_c < \sigma/2$. Then for any given $\epsilon, \beta > 0$, an (α, β)-median can be computed with probability at least $1 - \epsilon$ with $O((\log\log \max(X))^2 C_A(N)(\log \frac{1}{\beta})^2/\epsilon)$ communication complexity for $\alpha = O(\sigma \log \frac{1}{\beta})$.*

Proof Sketch. Consider the jth iteration of the for loop. By Theorem 4.6, with probability at least $1 - \epsilon/2 \log \frac{1}{\beta}$ we have that after executing Line 3.1, $\hat{\mu}^{(j)}$ is a $k^{(j)}$ $(\alpha, \frac{1}{N})$-order statistics of $\hat{X}^{(j)}$. Line 3.2 just applies a linear transformation to the $x_i^{(j)}$ values, which maps the range $[2^{\hat{\mu}^{(j)}}, 2^{\hat{\mu}^{(j)}+1} - 1]$ to the range $[1, \overline{X}]$. For Line 3.3, note that since $\hat{\mu}^{(j)}$ is implicitly broadcast in Line 3.4, each node i can locally tell whether $x_i^{(j+1)} \in \hat{X}^{(j+1)}$. Line 3.4 computes the value of $k^{(j+1)}$ by subtracting from $k^{(j)}$ the number of elements in $X^{(j)}$ whose value is less

```
Algorithm APX_MEDIAN2($X, \beta, \epsilon$)          $\beta$ is desired precision, $\epsilon$ is desired success probability
1     Root invokes protocol to compute $n \leftarrow$ REP_COUNTP($X$, $\left\lceil 2\log\frac{1}{\beta}/\epsilon \right\rceil$, TRUE);
          $k^{(1)} \leftarrow n/2$.                                                        initialization
2     Each node $i$ sets $\hat{x}_i^{(1)} \leftarrow \lfloor \log x_i \rfloor$;
          Let $X^{(1)} \leftarrow X$, $\hat{X}^{(1)} \leftarrow \left\{ \hat{x}_i^{(1)} \mid i \in V \right\}$.
3     for $j \leftarrow 1$ to $\left\lceil \log\frac{1}{\beta} \right\rceil$ do
3.1       Root invokes protocol to compute $\hat{\mu}^{(j)} \leftarrow$ APX_OS($\hat{X}^{(j)}$, $\epsilon/2\log\frac{1}{\beta}$, $k^{(j)}$),
              and broadcasts $\hat{\mu}^{(j)}$ to all nodes.
3.2       Each node $i$ executes: if $2^{\hat{\mu}^{(j)}} \le x_i^{(j)} < 2^{\hat{\mu}^{(j)}+1}$ then          scale input value
              $$x_i^{(j+1)} \leftarrow 1 + \frac{(x_i^{(j)} - 2^{\hat{\mu}^{(j)}}) \cdot (\overline{X} - 1)}{2^{\hat{\mu}^{(j)}} - 1}.$$   otherwise $i$ becomes passive
3.3       $X^{(j+1)} \leftarrow \left\{ x_i^{(j+1)} \mid 2^{\hat{\mu}^{(j)}} \le x_i^{(j)} < 2^{\hat{\mu}^{(j)}+1} \right\}$;          redefine input set
          $\hat{X}^{(j+1)} \leftarrow \left\{ \left\lfloor \log x_i^{(j+1)} \right\rfloor \mid x_i^{(j+1)} \in X^{(j+1)} \right\}$.
3.4       $k^{(j+1)} \leftarrow k^{(j)} -$ REP_COUNTP($X^{(j)}$, $\left\lceil 2\log\frac{1}{\beta}/\epsilon \right\rceil$, "$< 2^{\hat{\mu}^{(j)}}$").          adjust $k$
4     Output $\mu^{(\lceil \log\frac{1}{\beta} \rceil)}$.          $\mu^{(j)}$ is the original value that corresponds to $\hat{\mu}^{(j)}$
```

Fig. 4. Algorithm for computing the approximate median with polyloglog complexity.

than $2^{\hat{\mu}^{(j)}}$. Regarding approximation, we use the union bound to estimate that with probability at least $1 - \frac{\epsilon}{\log\frac{1}{\beta}}$, both $\mu^{(j+1)}$ is within a factor of 3σ from the actual $k^{(j)}$-order statistics (by Theorem 4.6), and that $k^{(j+1)}$ is within a factor of $\alpha_c + \sigma < 2\sigma$ from $\left| \left\{ x_i \in X^{(j)} \mid x_i < 2^{\hat{\mu}^{(j)}} \right\} \right|$ (by Lemma 4.1). Now, assuming that $\overline{X} > 2$, the difference between any two distinct values is at least doubled with each additional iteration, i.e. for any j, if $x_{i_1}^{(j)}, x_{i_2}^{(j)} \in X^{(j+1)}$ then $|x_{i_1}^{(j)} - x_{i_2}^{(j)}| \le \frac{1}{2}|x_{i_1}^{(j+1)} - x_{i_2}^{(j+1)}|$. Since after a single iteration we have an (α, β)-median with $\beta = O(1)$, after $\log\frac{1}{\beta}$ iterations, the algorithm will achieve the desired precision β, and the result will be correct with probability at least $1 - \epsilon$. However, note that α is increased by $O(\sigma)$ in each iteration due to the inaccuracy in computing APX_OS and REP_COUNTP. Regarding the communication complexity, note that since for each j we have $\max(\hat{X}^{(j)}) \le \log\overline{X} = O(\log N)$, Theorem 4.5 guarantees that each invocation of APX_OS incurs only $O((\log\log N)^2 C_A(N)\epsilon/\log\frac{1}{\beta})$ bits to the communication complexity. Since the number of calls to APX_OS is $O(\log\frac{1}{\beta})$, the result follows. ∎

Theorem 4.7, in conjunction with Fact 2.2, has the following corollary.

Corollary 4.8. *For any given constants $\beta, \epsilon > 0$ and $\alpha > 10^{-6}$, an (α, β)-median can be computed with probability at least $1 - \epsilon$ in $O((\log\log N)^3)$ communication complexity.*

It is clear that the space complexity of the algorithm is dominated by the size of the input item, i.e. it is $O(\log N)$.

5. The COUNT_DISTINCT aggregate

The output of the COUNT_DISTINCT aggregate is the number of distinct elements in the input multiset X. In [9], this aggregate is classified as "unique", a term whose interpretation is that the size of the state required to compute it (and hence its communication complexity) is proportional to the number of distinct elements in X. We first note that if an approximate answer is sufficient, then extremely efficient algorithms exist. For example, given a parameter k, the algorithm of [3] uses communication complexity of $k^2 \log\log n$ bits, and guarantees with probability at least 99%, that its output is within a factor of $(1 \pm 3.15/k)$ from the true answer to COUNT_DISTINCT.[3] In this section we make the simple observation that if the *exact* answer is sought, then the communication complexity of COUNT_DISTINCT jumps to $\Omega(n)$.

The proof is based on a simple reduction from the decision problem of Set Disjointness, that asks whether two input sets are disjoint. Our result can be interpreted in two possible ways. The first is that if a node may have up to a constant fraction of the input items, then for *each* topology there exists an input instance that requires $\Omega(n)$ communication complexity; alternatively, in the case where each node may have at most one input item, then there *exist* some topologies, and input instances for these topologies, that require $\Omega(n)$ communication complexity.

Theorem 5.1. *The communication complexity of any deterministic algorithm for* COUNT_DISTINCT *is* $\Omega(n)$ *in the worst case.*

Proof. By contradiction. Consider the Two-Party Set Disjointness Problem (2SD), defined as follows. There are two players denoted A and B, and the input consists of a set to each player, where player A sees initially only the set X_A and player B sees initially only the set X_B. During the execution of a protocol, the players exchange bit strings and can compute arbitrary functions of their local input and the bits communicated so far. A protocol is said to solve 2SD if eventually, one of the players (doesn't matter which) outputs 1 iff $X_A \cap X_B = \emptyset$. The communication complexity of a protocol is the total number of bits communicated between the players before the output is made. It is well known that no deterministic protocol can solve 2SD with $o(n)$ bits in the worst case, where $n \overset{\text{def}}{=} |X_A| + |X_B|$ (see, e.g., [8]). So, suppose that a deterministic protocol P solves COUNT_DISTINCT with communication complexity $C_P(n)$, and assume for contradiction that $C_P(n) = o(n)$. Let X_A and X_B denote the inputs sets for the 2SD problem. Consider the following protocol $2SD(P)$ for 2SD defined using P as a subroutine.

(1) A sends $|X_A|$ to B, and B sends $|X_B|$ to A.
(2) A and B run P to compute $c \leftarrow$ COUNT_DISTINCT on $X_A \cup X_B$.
(3) Output YES iff $c = |X_A| + |X_B|$.

Obviously, $2SD(P)$ solves 2SD since $X_A \cap X_B = \emptyset$ iff $|X_A \cup X_B| = |X_A| + |X_B|$. To complete the description, we specify the mapping of the two parties to network nodes. In the case the model allows for multiple items per node, we can take any topology with more than a single node, let A simulate the root node, and let B simulate all other nodes, and distribute the input accordingly. Otherwise, if only one input item can be held by a node, we can take a line graph of length $2n$, let A simulate the left n nodes and let B simulate the right n nodes. In any case, the communication complexity of $2SD(P)$ is $O(\log n + C_P(n))$, which is $o(n)$ if $C_P(n) = o(n)$, contradiction. ▮

We note that it is known [5] that the $\Omega(n)$ lower bound on communication complexity holds also for any *randomized* algorithm that solves 2SD correctly with probability at least $1 - \epsilon$, for any constant $\epsilon < 1/2$. This may seem to contradict the fact that COUNT_DISTINCT can be solved approximately using $O(\log \log n)$ bits. Intuitively, the explanation is that in order to solve 2SD, an algorithm must distinguish, with non-negligible probability, between some two answers of COUNT_DISTINCT that differ by as little as 1: this is because a difference of 1 in the result of COUNT_DISTINCT can flip the result of 2SD. Therefore, the proof above, in conjunction with the randomized lower bound for 2SD, implies that if an approximation algorithm for COUNT_DISTINCT produces the exact result with some significant probability, then its communication complexity is $\Omega(n)$.

6. Conclusion

In this paper we looked at the communication complexity of some natural distributed aggregate queries. We showed that contrary to some initial conjectures, the median can be computed efficiently once we allow multiple passes over the network, and even extremely efficiently if we only seek an approximate answer. However, the exact complexity of both the exact and the approximate versions of the median task are left open. It would be very interesting to see a superlogarithmic lower bound on the complexity of deterministic median computation; it would be also nice to reduce the complexity of approximate median computation to better than $O((\log N)^3)$.

[3] Under the assumption that good hash functions exist.

Acknowledgments

The author wishes to thank George Kollios and Mike Franklin for proposing the problem, and Mark Tuttle for useful discussions. Additional thanks are due to the anonymous reviewers of a previous version of this paper, who pointed out the existence of [11], and helped to clarify some confusion around the exact conditions under which Theorem 5.1 holds.

References

[1] Noga Alon, Yossi Matias, Mario Szegedy, The space complexity of approximating the frequency moments, J. Comput. System Sci. 58 (1) (1999) 137–147. Prel. version in STOC'96.

[2] Jeffrey Considine, Feifei Li, George Kollios, John Byers, Approximate aggregation techniques for sensor databases, in: Proc. 20th Int. Conf. on Data Engineering, ICDE, April 2004, pp. 449–460.

[3] Marianne Durand, Philippe Flajolet, Loglog counting of large cardinalities, in: Proc. 11th European Symposium on Algorithms, ESA, September 2003, pp. 605–617.

[4] Michael B. Greenwald, Sanjeev Khanna, Power-conserving computation of order-statistics over sensor networks, in: Proc. 23rd ACM Symp. on Principles of Database Systems, PODS, June 2004, pp. 275–285.

[5] Bala Kalyanasundaram, Georg Schnitger, The probabilistic communication complexity of set intersection, in: 2nd Annual Structure in Complexity Theory Conference, STRUCTURES, 1987, pp. 41–49.

[6] David Kempe, Alin Dobra, Johannes Gehrke, Gossip-based computation of aggregate information, in: 44th Annual Symposium on Foundations of Computer Science, 2003, pp. 482–491.

[7] Peter Kirschenhofer, Helmut Prodinger, A result in order statistics related to probabilistic counting, Computing 51 (1993) 15–27.

[8] Eyal Kushilevitz, Noam Nisan, Communication Complexity, Cambridge University Press, 1997.

[9] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, TAG: A tiny aggregation service for ad-hoc sensor networks, in: 5th Ann. Symp. on Operating Systems Design and Implemntation, OSDI, December 2002, pp. 131–146.

[10] S. Nath, P.B. Gibbons, S. Seshan, Z.R. Anderson, Synopsis diffusion for robust aggregation in sensor networks, in: SenSys'04: Proc. 2nd International Conference on Embedded Networked Sensor Systems, November 2004, pp. 250–262.

[11] Alberto Negro, Nicola Santoro, Jorge Urrutia, Efficient distributed selection with bounded messages, IEEE Trans. Parallel and Dist. Systems 8 (4) (1997) 397–401.

[12] Boaz Patt-Shamir, A note on efficient aggregate queries in sensor networks, in: Proc. 23rd Ann. ACM Symp. on Principles of Distributed Computing, July 2004, pp. 283–289.

[13] David Peleg, Distributed Computing: A Locality-Sensitive Approach, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[14] Mitali Singh, Viktor K. Prasanna, Optimal energy balanced algorithm for selection in single hop sensor network, in: IEEE International Workshop on Sensor Network Protocols and Applications (SNPA) ICC, May 2003.

[15] Yong Yao, Johannes Gehrke, The Cougar approach to in-network query processing in sensor networks, ACM SIGMOD Record 31 (3) (2002) 9–18.

[16] Jerry Zhao, Ramesh Govindan, Deborah Estrin, Computing aggregates for monitoring wireless sensor networks, in: The First IEEE International Workshop on Sensor Network Protocols and Applications, SNPA'03, Anchorage, AK, May 2003.