8

Summaries over Distributed Data

Big data is often distributed. This can be due to space concerns—a single machine just cannot hold all the data, or efficiency concerns. Even if the entire data set can be stored on one machine, it is often more efficient to split it up into many pieces, and process them in parallel. The UPDATE and MERGE algorithms that have been provided in the previous chapters serve this purpose well: For each piece, we repeatedly UPDATE the summary with each individual data record, and then send the resulting summary to a central entity, who can then perform MERGE operations on all the summaries received.

In a distributed setting, communication is usually considered as the most important measure of cost. The approach above incurs a communication cost of $k \times s$, where s is the individual summary size and k is the number of pieces. However, sometimes one can improve upon this standard approach, reducing the communication cost significantly. In this chapter, we will see a few important examples where this is the case. For some other cases, like distinct count (Section 2.5 and 2.6) and F_2 estimation (Section 3.6), it is known that the approach of sharing summaries is already optimal [229].

To make the presentation more precise, we adopt the following simple communication model, also called the *flat model* or the *star network* in the literature. We assume that the whole data set is partitioned into k pieces, each held by a node. There is a dedicated central node, or the *coordinator*, who has a two-way communication channel with each of the k nodes. These k nodes do not communicate to each other directly, but they can pass messages via the coordinator if needed. Note that algorithms designed in the flat model can usually be applied to more general network topologies, with a communication cost that is h times that in the flat model, where h is the diameter of the network. In some cases, more efficient algorithms are known for general network topologies than directly applying the flat model algorithms, and proper references are provided in the **History and Background** section.

8.1 Random Sampling over a Distributed Set

Brief Summary. Recall from Section 2.2 that the goal of random sampling is to draw a random sample of size s without replacement from a set of n items. In the distributed setting, the whole set A is partitioned into k subsets $A_1, \ldots, A_k$, each of size $n_1, \ldots, n_k$, respectively. If we draw a random sample of size s from each subset, and then MERGE them together using the algorithm in Section 2.2, the total communication cost will be $O(ks)$. Here, we describe an algorithm with communication cost $O(k + s)$.

Operations on the summary. The k nodes first send the sizes of their subsets, i.e., $n_1, \ldots, n_k$, to the coordinator. Then the idea is for the coordinator to simulate the sampling process using only these counts, and then retrieve the actual sampled items from the nodes. Specifically, the coordinator first decides the number of sampled items from each node using the following procedure. Let $n = n_1 + \cdots + n_k$. For the first sampled item, with probability n_i/n, it should come from A_i. So with the right probabilities, it selects an i from which the sample should come from. Then it decrement n_i by one, and repeat the process. After s steps, the coordinator knows s_i, the number of sampled items it should retrieve from A_i for $i = 1, \ldots, k$. Then it sends s_i to the i-th node, who then returns a random sample of size s_i from A_i. Finally, the random sample of the entire set is simply the union of the random samples returned from the k nodes. The pseudocode is given in Algorithm 8.1.

Further Discussion. The communication cost of this algorithm is clearly $O(k + s)$. The $O(k)$ part is due to the cost of communicating the subset sizes $n_1, \ldots, n_k$. If these are already known, e.g., when the algorithm is repeatedly executed to draw random samples multiple times, then the cost is only $O(s)$. The correctness of the algorithm can be easily established using the principle of deferred decisions.

Algorithm 8.1: Random sampling over distributed data

```
1  foreach $i$ do
2      Node $i$ sends $n_i$ to Coordinator;
3  $n \leftarrow n_1 + \cdots n_k$;
4  $s_i \leftarrow 0, i = 1, \ldots, k$;
5  for $j \leftarrow 1$ to $s$ do
6      Pick $r$ uniformly from $\{1, \ldots, n\}$;
7      Find $i$ such that $\sum_{\ell=1}^{i-1} n_i < r \leq \sum_{\ell=1}^{i} n_i$;
8      $s_i \leftarrow s_i + 1$;
9      $n_i \leftarrow n_i - 1$;
10     $n \leftarrow n - 1$;
11 foreach $i$ do
12     Coordinator sends $s_i$ to node $i$;
13     $S_i \leftarrow$ a random sample of size $s_i$ from $A_i$;
14     Node $i$ sends $S_i$ to Coordinator;
15 Coordinator returns $S = S_1 \cup \cdots \cup S_k$;
```

8.2 Point Queries over Distributed Multisets

Brief Summary. In this section, we consider a multiset that is stored in a distributed fashion. We assume that the items in the multiset are drawn from a bounded universe $[u] = \{1, \ldots, u\}$. The multiset is partitioned into k pieces, held by k distributed nodes. We denote the *local count* of item i at node j by $x_{i,j}$, and the *global count* of item i is $y_i = \sum_j x_{i,j}$. Let $n = \|y\|_1 = \sum_i y_i$. A point query for item i returns an approximation of y_i. In Chapter 3, we have seen a number of summaries of size $O(1/\varepsilon)$ or $O(1/\varepsilon \log(1/\delta))$ that can answer point queries with additive error εn. Here, δ is the probability that any y_i is estimated with an error greater than εn. Constructing such a summary for each piece and then merging them together will lead to a communication cost of $O(k/\varepsilon)$ or $O(k/\varepsilon \log(1/\delta))$. Here we present an improved algorithm with communication cost $O(\sqrt{k}/\varepsilon \log(1/\delta))$ bits.

Operations on the summary. The algorithms described in this section all return an unbiased estimator of y_i for any given i, with variance $O((\varepsilon n)^2)$. By the Chebyshev inequality, this gives us an estimate with additive εn error with a constant probability. If a higher success probability $1 - \delta$ is desired, one can use the standard technique of running

$O(\log(1/\delta))$ independent instances of the algorithm, and returning the median estimate. We now describe a sequence of three increasingly complicated sampling algorithms for the distributed point estimation problem.

Uniform coin-flip sampling. A very simple algorithm is for each node to sample each of its items with probability $p = 1/(\varepsilon^2 n)$. If an item i has local count $x_{i,j}$ at node j, it is treated as $x_{i,j}$ copies of i and each copy is sampled by node j independently. All the sampled items are sent to the coordinator. Then for a queried item i, we use the estimator $Y_i = X_i/p$, where X_i is the number of copies of item i received by the coordinator. It follows that $\mathsf{E}[Y_i] = y_i$ and $\mathsf{Var}[Y_i] = O((\varepsilon n)^2)$ (details presented below). The expected communication cost of this simple algorithm is $pn = O(1/\varepsilon^2)$. This method works well if $k > 1/\varepsilon^2$. Otherwise, the following algorithm works better.

Importance sampling. The idea is still to use random sampling, but bias the sampling probability so as to favor heavy items, i.e., those with large local counts. The basic version of the algorithm is very simple. Let $g(x) = \min\{x\sqrt{k}/\varepsilon n, 1\}$ be the sampling function. We assume that the common parameters n, k, ε are known to all nodes; otherwise we can spend an extra $O(k)$ communication cost to broadcast them. If an item i has local count $x_{i,j}$ at node j, then the node with probability $g(x_{i,j})$ samples this item and sends the item together with its local count $x_{i,j}$ to the coordinator. Set $Y_{i,j} = x_{i,j}$ if the coordinator receives the item-count pair $(i, x_{i,j})$ from node j, and $Y_{i,j} = 0$ otherwise. Then for any given i, the coordinator can estimate y_i as (define $\frac{0}{0} = 0$)

$$Y_i = \frac{Y_{i,1}}{g(Y_{i,1})} + \cdots + \frac{Y_{i,k}}{g(Y_{i,k})}. \tag{8.1}$$

The full analysis shows that $\mathsf{E}[Y_i] = y_i$ and $\mathsf{Var}[Y_i] = O((\varepsilon n)^2)$, and this algorithm transmits a total of $O(\sqrt{k}/\varepsilon)$ item-count pairs.

An advanced version of the importance sampling algorithm. One can be more careful with the exact number of bits communicated. Let u be the size of the universe where the items are drawn from. For example, if the items are IPv6 addresses, then $u = 2^{128}$. Note that an item thus needs $O(\log u)$ bits to represent, and a count needs $O(\log n)$ bits, so the basic version of the algorithm communicates $O(\sqrt{k}/\varepsilon(\log u + \log n))$ bits in total. A more advanced version of the algorithm can reduce this cost to just $O(\sqrt{k}/\varepsilon)$ bits. The idea is to encode the sampled items into a BloomFilter. Recall from Section 2.7 that a BloomFilter is a space-efficient encoding scheme that compactly stores a set of items S. We recall its properties that are needed for our purpose here. Given any item, the BloomFilter can tell us whether this item is in S or not. It does not have false negatives, but may have a false positive probability q for any queried item. More precisely, if the queried item is in S, the answer is always "yes"; if it is not in S, then with probability q it returns "yes" and with probability $1-q$ returns "no". The false positive probability q can be made arbitrarily small by using $O(\log(1/q))$ bits per item, and the value of q can be computed as (2.6). Thus the BloomFilter uses $O(n\log(1/q))$ bits to store a total of n items, regardless of the size of the universe.

We now describe the advanced version of the algorithm. First write each $x_{i,j}$ in a canonical form as

$$x_{i,j} = a_{i,j}\frac{\varepsilon n}{\sqrt{k}} + b_{i,j}, \tag{8.2}$$

where $a_{i,j}$ and $b_{i,j}$ are both non-negative integers such that $a_{i,j} \le \frac{\sqrt{k}}{\varepsilon}$ and $b_{i,j} < \frac{\varepsilon n}{\sqrt{k}}$. We assume $\frac{\varepsilon n}{\sqrt{k}}$ is an integer. Note that given these constraints, there is a unique way of expressing $x_{i,j}$ in the form above. Then we have

$$y_i = \frac{\varepsilon n}{\sqrt{k}}\sum_{j=1}^{k} a_{i,j} + \sum_{j=1}^{k} b_{i,j}. \tag{8.3}$$

Given a point query i, we will estimate the two terms of (8.3) separately.

The second term is easier to deal with. As in the basic version of the algorithm, the nodes sample each $b_{i,j}$ with probability $g(b_{i,j})$, and then estimate the second term of (8.3) similarly as before

$$B_i = \frac{B_{i,1}}{g(B_{i,1})} + \cdots + \frac{B_{i,k}}{g(B_{i,k})},$$

where $B_{i,j} = b_{i,j}$ if $b_{i,j}$ is sampled by node j, and 0 otherwise. The observation is that, since $g(x)$ is a linear function when $x \le \varepsilon n/\sqrt{k}$, $\frac{B_{i,j}}{g(B_{i,j})}$ is either 0 or $\varepsilon n/\sqrt{k}$. Thus, the nodes do not need to send out the values of $b_{i,j}$'s at all, they only need to inform the coordinator which items have had their $b_{i,j}$'s sampled. So all these items can be encoded in a Bloom filter. But as the Bloom filter has a false positive rate q, this has to be accounted for. More precisely, for a point query i, suppose among the k Bloom filters that the coordinator has received, Z_i of them says "yes", then we use the estimator

$$B_i = \frac{\varepsilon n}{\sqrt{k}} \cdot \left(\frac{Z_i - kq}{1-q}\right). \tag{8.4}$$

Algorithm 8.2: Point queries over distributed multiset: Node j

```
1 Initialize empty Bloom filter $F_j$ with any constant false positive
    rate $q$;
2 for $r \leftarrow 0$ to $\log(\sqrt{k}/\varepsilon)$ do
3     Initialize empty Bloom filter $F_j[r]$ with false positive rate
        $q_r \le 1/2^{3r+1}$;
4 foreach $i$ do
5     Let $x_{i,j} = a_{i,j}\frac{\varepsilon n}{\sqrt{k}} + b_{i,j}$, where $a_{i,j}$ and $b_{i,j}$ are non-negative
        integers and $a_{i,j} \le \frac{\sqrt{k}}{\varepsilon}, b_{i,j} < \frac{\varepsilon n}{\sqrt{k}}$ ;
6     for $r \leftarrow 0$ to $\log(\sqrt{k}/\varepsilon)$ do
7         if $a_{i,j}[r] = 1$ then insert $i$ into $F_j[r]$;
8     With probability $g(b_{i,j})$, insert $i$ into $F_j$;
9 Send $F_j, F_j[0], F_j[1], \dots, F_j[\log(\sqrt{k}/\varepsilon)]$ to Coordinator;
```

It can be shown that (8.4) is an unbiased estimator for the second term of (8.3) with variance $O((\varepsilon n)^2)$, for any constant q.

For the first term of (8.3), the idea is to consider each $a_{i,j}$ in its binary form and dealing with each bit individually. Let $a_{i,j}[r]$ be the r-th rightmost bit of $a_{i,j}$ (counting from 0). For each r, node j encodes all the items i where $a_{i,j}[r] = 1$ in a Bloom filter with false positive probability $q_r \le 1/2^{3r+1}$. For any item i, suppose $Z_{i,r}$ is the number of Bloom filters that assert $a_{i,j}[r] = 1$. Then we use the following estimator for the first term of (8.3):

$$A_i = \frac{\varepsilon n}{\sqrt{k}} \sum_{r=0}^{\log(\sqrt{k}/\varepsilon)} 2^r \frac{Z_{i,r} - kq_r}{1 - q_r}. \tag{8.5}$$

This is an unbiased estimator of the first term of (8.3) with variance $O((\varepsilon n)^2)$ (details in the further details section below).

The pseudocode for the nodes and the coordinator is given in Algorithm 8.2 and 8.3.

All the Bloom filters received by the coordinator constitute the summary for the entire multiset, on which any point query can be posed. The total size needed for all the BloomFilter instances can be shown to be $O(\sqrt{k}/\varepsilon)$ bits.

Algorithm 8.3: Point queries over distributed multiset: Coordinator with query i

```
1 $Z_i \leftarrow$ the number of $F_j$'s that assert containing $i$;
2 for $r \leftarrow 0$ to $\log(\sqrt{k}/\varepsilon)$ do
3     $Z_{i,r} \leftarrow$ the number of $F_j[r]$'s that assert containing $i$;
4 return $\frac{\varepsilon n}{\sqrt{k}} \cdot \frac{Z_i - kq}{1-q} + \frac{\varepsilon n}{\sqrt{k}} \sum_{r=0}^{\log(\sqrt{k}/\varepsilon)} 2^r \frac{Z_{i,r} - kq_r}{1-q_r}$;
```

Further Discussion. *The uniform sampling algorithm.* The analysis of the uniform sampling algorithm is quite standard. Consider any given i, which has y_i copies in the entire multiset. For $\ell = 1, \ldots, y_i$, let $X_{i,\ell} = 1$ if the ℓ-th copy is sampled, and 0 otherwise. We have $\mathsf{E}[X_{i,\ell}] = p$, and $\mathsf{Var}[X_{i,\ell}] = p(1-p)$. Setting $p = 1/\varepsilon^2 n$, the estimator has expectation

$$\mathsf{E}[Y_i] = \mathsf{E}[X_i/p] = 1/p \cdot \mathsf{E}\left[\sum_{\ell=1}^{y_i} X_{i,\ell}\right] = 1/p \cdot py_i = y_i,$$

and variance

$$\mathsf{Var}[Y_i] = \mathsf{Var}[X_i/p] = 1/p^2 \cdot \mathsf{Var}\left[\sum_{i=1}^{y_i} X_{i,\ell}\right] = 1/p^2 \cdot y_i p(1-p) \leq y_i/p \leq (\varepsilon n)^2.$$

The basic version of the importance algorithm. We now analyze the basic version of the biased sampling algorithm. Recall that the algorithm samples each local count $x_{i,j}$ with probability $g(x_{i,j})$ where $g(x) = \min\{x\sqrt{k}/\varepsilon n, 1\}$. The estimator given in (8.1) has expectation

$$\mathsf{E}[Y_i] = \sum_{j=1}^{k} \frac{\mathsf{E}[Y_{i,j}]}{g(Y_{i,j})} = \sum_{j=1}^{k} g(x_{i,j}) \frac{x_{i,j}}{g(x_{i,j})} = \sum_{j=1}^{k} x_{i,j} = y_i.$$

Now we consider the variance of Y_i. Since we sample an item with probability one (i.e., zero variance) when the local count $x_{i,j} > \varepsilon n/\sqrt{k}$, it is sufficient to consider the worst case when all $x_{i,j} \leq$

$\varepsilon n/\sqrt{k}$. We have

$$\begin{aligned}
\mathsf{Var}[Y_i] &= \sum_{j=1}^{k} \frac{x_{i,j}^2(1 - x_{i,j}\sqrt{k}/\varepsilon n)}{x_{i,j}\sqrt{k}/\varepsilon n} \\
&= \frac{\varepsilon n}{\sqrt{k}} \sum_{j=1}^{k} x_{i,j} - \sum_{j=1}^{k} x_{i,j}^2 \\
&\le \frac{\varepsilon n}{\sqrt{k}} y_i - \frac{1}{k} y_i^2 \qquad \text{(Cauchy-Schwartz inequality)} \\
&= -\left(\frac{y_i}{\sqrt{k}} - \frac{\varepsilon n}{2}\right)^2 + \frac{(\varepsilon n)^2}{4} \le \frac{1}{4}(\varepsilon n)^2.
\end{aligned}$$

For the communication cost, we can easily derive that this algorithm samples and transmits a total of $\sum_{i,j} g(x_{i,j}) \le \sum_{i,j} x_{i,j}\sqrt{k}/\varepsilon n = n \cdot \sqrt{k}/\varepsilon n = \sqrt{n}/\varepsilon$ item-count pairs (in expectation).

The advanced version of the biased sampling algorithm. Recall that in the advanced version of the algorithm, we decompose the global count y_i into two terms as in (8.3) and estimate each of them separately. The second term is easier to analyze, as it is similar to the basic version, except that we need to take into account the false positive rate q of the Bloom filters.

We define $Z_{i,j}$ to be the indicator random variable set to 1 if the Bloom Filter from node j asserts that it contains the item i, and 0 otherwise. It is easy to see that $\Pr[Z_{i,j} = 1] = g(b_{i,j}) + (1 - g(b_{i,j}))q$, and thus $\mathsf{E}[Z_{i,j}] = g(b_{i,j}) + (1 - g(b_{i,j}))q$. Then we have

$$\begin{aligned}
\mathsf{E}[B_i] &= \frac{\varepsilon n}{\sqrt{k}} \cdot \frac{\mathsf{E}[Z_i] - kq}{1 - q} \\
&= \frac{\varepsilon n}{\sqrt{k}} \cdot \frac{\sum_{j=1}^{k} \mathsf{E}[Z_{i,j}] - kq}{1 - q} \\
&= \frac{\varepsilon n}{\sqrt{k}} \cdot \frac{(1 - q)\sum_{j=1}^{k} g(b_{i,j}) + kq - kq}{1 - q} \\
&= \frac{\varepsilon n}{\sqrt{k}} \sum_{j=1}^{k} g(b_{i,j}) = \sum_{j=1}^{k} b_{i,j}.
\end{aligned}$$

The variance of the estimator is

$$\mathsf{Var}[B_i] = \frac{(\varepsilon n)^2}{k(1 - q)^2} \mathsf{Var}[Z_i]$$

$$= \frac{(\varepsilon n)^2}{k(1-q)^2}\sum_{j=1}^{k}\mathsf{Var}[Z_{i,j}]$$

$$= \frac{(\varepsilon n)^2}{k(1-q)^2}\sum_{j=1}^{k}((g(b_{i,j})+(1-g(b_{i,j}))q)(1-g(b_{i,j})-(1-g(b_{i,j}))q))$$

$$= \frac{(\varepsilon n)^2}{k(1-q)^2}\left(\frac{k}{4}-\left(\frac{(1-q)\sum_{j=1}^{k}b_{i,j}}{\varepsilon n}-\frac{(1-2q)\sqrt{k}}{2}\right)^2\right)$$

$$\le \frac{(\varepsilon n)^2}{4(1-q)^2}.$$

Thus, it is sufficient to set a constant q so that $\mathsf{Var}[Y_i] = O((\varepsilon n)^2)$. The communication cost for this part is the same as in the basic version, except that since now each sampled item-count pair only consumes $O(\log(1/q)) = O(1)$ bits, the total cost is $O(\sqrt{n}/\varepsilon)$ bits.

For the first term, we need to show that $\mathsf{E}[A_i] = \dfrac{\varepsilon n}{\sqrt{k}}\sum_{j=1}^{n} a_{i,j}$, and bound $\mathsf{Var}[A_i]$. Let $Z_{i,r}$ be the number of Bloom filters that asserts $a_{i,j}[r] = 1$. Let $c_{i,r} = \sum_{j=1}^{k} a_{i,j}[r]$. Since there are $k - c_{i,r}$ Bloom filters which may, with probability q_r, assert $a_{i,j}[r] = 1$ despite $a_{i,j}[r] = 0$, it is easy to see that $\mathsf{E}[Z_{i,r}] = c_{i,r} + (k - c_{i,r})q_r$, and $\mathsf{Var}[Z_{i,r}] = (k - c_{i,r})q_r(1-q_r) \le kq_r(1-q_r)$. Thus we have

$$\mathsf{E}[A_i] = \frac{\varepsilon n}{\sqrt{k}}\sum_{r=0}^{\log(\sqrt{k}/\varepsilon)} 2^r\frac{\mathsf{E}[Z_{i,r}]-kq_r}{1-q_r}$$

$$= \frac{\varepsilon n}{\sqrt{k}}\sum_{r=0}^{\log(\sqrt{k}/\varepsilon)} 2^r c_{i,r} = \frac{\varepsilon n}{\sqrt{k}}\sum_{j=1}^{k} a_{i,j},$$

and

$$\mathsf{Var}[A_i] = \frac{(\varepsilon n)^2}{k}\sum_{r=0}^{\log(\sqrt{k}/\varepsilon)}\frac{2^{2r}}{(1-q_r)^2}\mathsf{Var}[Z_{i,r}]$$

$$\le (\varepsilon n)^2\sum_{r=0}^{\log(\sqrt{k}/\varepsilon)} 2^{2r}\frac{q_r}{1-q_r}.$$

So as long as we set $q_r \le 1/2^{3r+1}$, we can bound $\mathsf{Var}[A_i]$ by $O((\varepsilon n)^2)$, as desired. The cost for each $a_{i,j}[r] = 1$ is thus $O(\log(1/q_r)) = O(r)$ bits. Since each $a_{i,j}[r] = 1$ represents $2^r\frac{\varepsilon n}{\sqrt{k}}$ copies of an item, the amortized cost for every $\frac{\varepsilon n}{\sqrt{k}}$ copies is $O(r/2^r) = O(1)$ bits. Therefore, the total communication cost is $O(\sqrt{k}/\varepsilon)$ bits.

Implementation Issues. For the above analysis to go through, the nodes need to use independent random sources, including the randomness in the sampling and the random hash functions in their Bloom filters.

History and Background. The sampling framework and the idea to combine with Bloom filters were introduced by Zhao *et al.* [235]. Their algorithm was later simplified and improved by Huang *et al.* [130] to the version presented here. The $O(\sqrt{k}/\varepsilon)$-bit communication cost was later shown to be optimal for $k \leq 1/\varepsilon^2$ [229], while the $O(1/\varepsilon^2)$-bit communication cost achieved by the simple random sampling algorithm is optimal for $k > 1/\varepsilon^2$.

8.3 Distributed Ordered Data

Brief Summary. In this section, we revisit the problem considered in Chapter 4, i.e., rank and quantile queries on a set A of n elements drawn from an ordered universe, except that here the set A is partitioned into k pieces, each held by a different node. In this section we assume that there are no duplicates in the set A. Recall that for an element x, the rank of x in A (x may or may not be in A) is $\text{rank}(x) = |\{y < x : y \in A\}|$. An ε-approximate rank query for element x returns an estimated rank $\tilde{r}$ such that

$$\text{rank}(x) - \varepsilon n \leq \tilde{r} \leq \text{rank}(x) + \varepsilon n,$$

while an ε-approximate quantile query for a rank r returns an element x such that

$$r - \varepsilon n \leq \text{rank}(x) \leq r + \varepsilon n.$$

Using the summaries described in Chapter 4 and merging them requires a communication cost of at least $\Omega(k/\varepsilon)$. Below we present an algorithm with communication cost $\tilde{O}(\sqrt{k}/\varepsilon)$.

Operations on the summary. Let $t = \lfloor \varepsilon n / \sqrt{k \log(2/\delta)/2} \rfloor$, where δ will be the probability of exceeding the ε-error guarantee. The algorithm is very simple. Each node first sorts its own set of elements. Then it chooses an offset b uniformly at random between 0 and $t - 1$, and sends the $(at+b)$-th element to the coordinator, for $a = 0, 1, 2, \ldots$. Since one out of every t elements is selected, the total communication cost is $O(n/t) = O(\sqrt{k}\log(1/\delta)/\varepsilon)$.

To answer a rank query for any given element x, the coordinator simply counts the number of elements received from all the nodes that are smaller than x, multiplied by t.

To answer a quantile query for a given rank r, the coordinator just sorts all the elements received, and returns the one at position $\lfloor r/t \rfloor$.

Further Discussion. We first analyze the error in the rank query for any given element x. Let $\mathrm{rank}(x, i)$ be the local rank of x at node i, i.e., the number of elements smaller than x stored at node i. It is clear that the global rank of x is $\mathrm{rank}(x) = \sum_i \mathrm{rank}(x, i)$. Effectively, the coordinator computes an estimate of $\mathrm{rank}(x, i)$, denoted $\widehat{\mathrm{rank}}(x, i)$, for each i and adds them up, whereas $\widehat{\mathrm{rank}}(x, i)$ is simply the number of selected elements at node i that are smaller than x, multiplied by t. Recall that the algorithm chooses the offset b uniformly at random between 0 and $t - 1$, so we have

$$\widehat{\mathrm{rank}}(x, i) = \begin{cases} \left\lfloor \frac{\mathrm{rank}(x,i)}{t} \right\rfloor \cdot t, & \text{w.p. } 1 - \frac{\mathrm{rank}(x,i) \bmod t}{t}; \\ \left(\left\lfloor \frac{\mathrm{rank}(x,i)}{t} \right\rfloor + 1\right) \cdot t, & \text{w.p. } \frac{\mathrm{rank}(x,i) \bmod t}{t}. \end{cases}$$

One can check that

$$\mathsf{E}[\widehat{\mathrm{rank}}(x, i)] = \left\lfloor \frac{\mathrm{rank}(x, i)}{t} \right\rfloor \cdot t + (\mathrm{rank}(x, i) \bmod t) = \mathrm{rank}(x, i).$$

Therefore, $\widehat{\mathrm{rank}}(x) = \sum_i \widehat{\mathrm{rank}}(x, i)$ is an unbiased estimator of $\mathrm{rank}(x)$. Since it is a sum of independent random variables, each of which has a bounded range of $t = \varepsilon n/(\sqrt{(k/2)\log(2/\delta)})$, we can invoke the Chernoff-Hoeffding inequality 1.4 to obtain the following concentration results:

$$\Pr\left[\left|\widehat{\mathrm{rank}}(x) - \mathrm{rank}(x)\right| > \varepsilon n\right] \le 2\exp\left(\frac{-2(\varepsilon n)^2}{kt^2}\right) = \delta,$$

i.e., the probability that the estimated rank deviates from the true rank by more than εn is at most δ.

To have a guarantee on quantile queries, we set $\delta = \varepsilon/3$ in the algorithm. This ensures that any rank query can be answered within error εn with probability at least $1 - \varepsilon/3$. Then, with at least constant probability, the estimated rank is accurate (within error of εn) for all the $1/\varepsilon - 1$ elements that rank at $\varepsilon n, 2\varepsilon n, \ldots, (1 - \varepsilon)n$, which is enough to answer all rank queries. When all rank queries can be answered within εn error, all the quantile queries can also be answered with the desired error guarantee.

History and Background. The algorithm presented above is a simplification of the algorithm described in [129] for the 1D case. The general algorithm works in higher dimensions and computes an ε-approximation (see Section 5.1) for any range space with bounded discrepancy. However, in higher dimensions, the algorithm relies on discrepancy minimization [17], which is not known to be practical.

If we directly use this algorithm on a general communication network, the communication cost will be $O(h\sqrt{k}/\varepsilon)$, where h is the diameter of the network. In [128], a better algorithm with communication cost $O(\sqrt{hk}/\varepsilon)$ was presented.