Theoretical Computer Science 602 (2015) 60–79

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Boosting distinct random sampling for basic counting on the union of distributed streams [☆]

Bojian Xu

Department of Computer Science, Eastern Washington University, Cheney, WA 99004, USA

A R T I C L E   I N F O

Article history:
Received 10 May 2014
Received in revised form 27 April 2015
Accepted 16 August 2015
Available online 20 August 2015
Communicated by R. Gavaldà

Keywords:
Basic counting
Data stream
Distributed streams
Coordinated adaptive sampling
Distinct sampling
Direct sampling

A B S T R A C T

We revisit the classic basic counting problem in the distributed streaming model. In the solution for maintaining an (ϵ, δ)-estimate, we make the following new contributions: (1) For a bit stream of size n, where each bit has a probability at least γ to be 1, we exponentially reduced the average total processing time from the best prior work's $\Theta(n \log(1/\delta))$ to $O((1/(\gamma\epsilon^2))(\log^2 n)\log(1/\delta))$, thus providing the first sublinear-time streaming algorithm for this problem. (2) In addition to an overall much faster processing speed, our method provides a new tradeoff that a lower accuracy demand (a larger value for ϵ) promises a faster processing speed, whereas the best prior work's processing speed is $\Theta(n \log(1/\delta))$ in any case and for any ϵ. (3) The worst-case total time cost of our method matches the best prior work's $\Theta(n \log(1/\delta))$, which is necessary but rarely occurs in our method. (4) The space usage overhead in our method is a lower order term compared with the best prior work's space usage and occurs only $O(\log n)$ times during the stream processing and is too negligible to be detected by the OS in practice. We further validate these theoretical results with experiments on both real-world and synthetic data, showing that our method is faster than the best prior work by a factor of several to several hundreds depending on the stream size and accuracy demands, without any detectable space usage overhead. Our method is based on a faster sampling technique that we design for boosting the sampling procedure in the best prior work and we believe this technique can be of other independent interest.

1. Introduction

Advances in modern science and technology have given rise to massive data (or so-called big data). Some of the data naturally arrives as *streams*. Examples include network data packets passing through a router, environmental data collected by sensor networks, and search requests received by search engines. In many cases, such massive streaming data need to be monitored in a real-time fashion. Such data process requirements make conventional methods such as storing them in a relational database and issuing SQL queries thereafter infeasible, and thus brings up the phenomenon of *data stream processing* [2,21]. In data stream processing, the workspace is often orders of magnitude smaller than the stream size, requiring the data be processed in one pass.

However, most streaming algorithms need to look at every data element at least once [1,4,7–9,11,16] (see [2,21] for many other example references). In some cases where extremely fast paced streaming data is involved, even a single glance

☆ A preliminary version of this article appeared in [27]. Supported in part by EWU's Faculty Grants for Research and Creative Works.
E-mail address: bojianxu@ewu.edu.

http://dx.doi.org/10.1016/j.tcs.2015.08.014
0304-3975/© 2015 Elsevier B.V. All rights reserved.

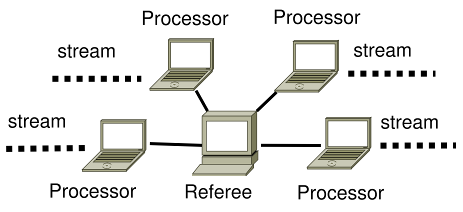

Fig. 1. The generalized system setting in distributed basic counting.

at every stream element can be unaffordable. For example, a typical OC48 link transfers 2.5 Gbits per second and AT&T backbone networks carry over 15 petabytes of data traffic on an average business day. Deploying a streaming algorithm for monitoring purpose to process every data element in such massive data streams is very computationally expensive and can greatly hurt the performance of the system. The goal of sublinear-time algorithms is to solve computational problems without having to look at every input data element. However, in sublinear time algorithm design, the input data is often stored statically [12,17,23,24], meaning we can visit any part of the input data at any time if needed.

In this paper, we demonstrate that designing a sublinear-time algorithm for streaming data processing is also possible, without losing accuracy guarantee compared with linear-time streaming algorithms. In particular, we propose the first streaming algorithm for the distributed basic counting problem using time sublinear of the stream size in the average case. To our best knowledge, the best prior result [14] for basic counting over distributed streams has to visit every stream element at least once and thus needs a time cost at least linear of the stream size in any case.

1.1. Distributed basic counting

Alice and Bob (called processors) are processing in parallel two geographically distributed bit streams $A = \{a_1, \dots, a_n\}$ and $B = \{b_1, \dots, b_n\}$, respectively. Each stream is synchronous, i.e., the ith bit is received before the jth bit, if $i < j$. Between Alice and Bob, there is no direct network connection. There is also a referee, who is located on another remote site and has direct network connection to both Alice and Bob. At any point of time when the referee receives a query for the number of 1-bits in the union of the two streams A and B, the referee wants to know

$$U(A, B) = \sum_{i=1}^{n} (a_i \vee b_i),$$

where $\vee$ is the bit-wise logical OR operator, and n is the number of bits that Alice and Bob both have received and processed so far when the query arrives at the referee. Note that both streams evolve over time and thus the stream size n and the value of U monotonically increase over time. The constraints and challenges in the computation of U are: (1) No direct communication between Alice and Bob (processors) is allowed, since there are no direct connection between Alice and Bob, (2) Use small workspace on the processors as well as on the referee, and (3) Use small communication cost on the links connecting the processors and the referee.

The problem can be generalized to k streams $R_1, R_2, \dots, R_k$, processed by k processors respectively, for some constant $k \geq 2$. Fig. 1 shows the generalized system setting that is assumed in the distributed basic counting. For $j = 1, 2, \dots, k$, we write the stream R_j in the form of $\{r_{j,1}, r_{j,2}, \dots, r_{j,n}\}$. Upon receiving a query, the referee wants to know the number of 1-bits in the union of the k streams

$$U(R_1, R_2, \dots, R_k) = \sum_{i=1}^{n} (r_{1,i} \vee r_{2,i} \vee \dots \vee r_{k,i}). \tag{1}$$

The same constraints and challenges for the 2-stream case hold in this general setting. Because our method for the 2-stream case can be easily extended for the general setting, we will focus on the 2-stream case in our presentation. The extension for the general setting will be presented in the end.

We refer readers to [14] for a detailed discussion on the extensive applications of the distributed basic counting in large-scale data aggregation and monitoring.

1.2. Prior work

A naive method for the referee to maintain the knowledge of U is to get Alice and Bob to continuously forward their stream elements to the referee. The referee will then simply do a straightforward calculation of U in one pass of the two streams, using $O(\log n)$ bits of workspace at both the processors and the referee. However, this approach introduces a high communication cost between the processors and the referee, which is prohibited in many applications such as network monitoring. To reduce the communication cost, Gibbons and Tirthapura (GT) proposed a communication-efficient distributed computing scheme [14], where Alice and Bob each maintains a small-space data structure (a.k.a. sketch) without communicating neither to each other nor to the referee over the course of stream processing. When the query arrives, the referee