International Journal of Foundations of Computer Science,
Vol. 14, No. 6 (2003) 1129–1146
© World Scientific Publishing Company

World Scientific
www.worldscientific.com

EFFICIENT ALGORITHMS FOR SELECTION AND SORTING OF LARGE DISTRIBUTED FILES ON DE BRUIJN AND HYPERCUBE STRUCTURES

DAVID S. L. WEI
Dept. of Comp. and Info. Sc., Fordham University, Bronx, NY 10458
wei@dsm.fordham.edu

SANGUTHEVAR RAJASEKARAN
Dept. of Comp. and Info. Sc., University of Florida, Gainesville, FL 32611
raj@cise.ufl.edu

KSHIRASAGAR NAIK
Dept. of Elect. and Comp. Eng., University of Waterloo, Waterloo, Ontario, N2L 3G1
knaik@swen.uwaterloo.ca

SY-YEN KUO
Dept. of Elect. Eng., National Taiwan University, Taipei, Taiwan
sykuo@cc.ee.ntu.edu.tw

Received 21 January 2002
Accepted 19 March 2003
Communicated by H.-C. Yen

ABSTRACT

In this paper we show the power of sampling techniques in designing efficient distributed algorithms. In particular, we apply sampling techniques in the design of selection algorithms on the hypercube and de Bruijn networks, and show that the message complexity of selecting an item from a set (file) is less sensitive to the cardinality of the set (file). Given a file with n keys, our algorithm performs a selection on a p-node de Bruijn network or hypercube using only $O(p \log\log n)$ messages and suffering a delay of $O(\tau \log p \log\log n)$, with high probability. Our selection scheme outperforms the existing approaches in terms of both message complexity and communication delay. Because of the lesser sensitivity of message complexity and communication delay of our algorithms to the file size, our distributed selection schemes are very attractive in applications where very large database systems are involved. Using our selection algorithms, we also show that both quicksort-based sorting scheme and enumeration sorting scheme can be developed for sorting large distributed files on the hypercube and de Bruijn networks. Both of our sorting algorithms outperform the existing distributed sorting schemes in terms of both message complexity and communication delay.

Keywords: Distributed selection; distributed sorting; hypercube; de Bruijn network; large distributed files.

1. Introduction

Large, distributed files must be processed in some applications, such as national census, personnel information system of large companies, etc. In this paper, we develop efficient schemes for selection of and sorting the keys of large files distributed over a number of sites linked by an interconnection network. In this context, selection and sorting are explained as follows. The n keys of a given file F are evenly distributed over a network of p nodes, *i.e.* each node contains approximately $\frac{n}{p}$ keys. Selection of the kth key means finding the value of the key whose rank is k in F. Sorting F means relocating the n keys among the p nodes such that all the keys at the ith node are smaller (greater) than all the keys at the jth node for $i < j$ $(i > j)$. Also, after sorting a file, each node contains the same number of keys as before sorting[a]. In the development of our schemes, we logically organize the nodes (sites) into *hypercube* and *de Bruijn* structures. Both structures have been widely used as logical structures for developing distributed algorithms for classical problems [3, 7, 9]. Both the structures can be emulated (realized) on a *wavelength division multiplexed* (WDM) network or an *ATM* network using wavelength assignment or virtual path layout, respectively [24].

Due to the fact that the cost of passing a message from one node to an adjacent one is much higher than a singular, local computation, distributed algorithms are designed in such a way that the number of messages needed for the desired computation is minimized, while keeping the communication delay as small as possible [22]. The efficiencies of distributed algorithms are thus commonly expressed in terms of message count and communication delay.

The selection problem has attracted considerable attention within the distributed computing community. In [20], Shrira *et al.* present a selection algorithm based on a Communicating Sequential Processes-like synchronous message passing model. Their algorithm finds the kth key from a file of size n using $O(pn^{0.91})$ messages, where p is the number of nodes. Using sampling techniques, Frederickson [6] designed three selection algorithms for a network of nodes with asynchronous message passing. On a ring of p nodes, his selection algorithm finds the kth key of a file of size n in $O(p^{1+\epsilon}\log n)$ messages with $O(\tau\frac{p\log n}{\log p})$ communication delay, or in $O(p\log^2 p\log n)$ messages with $O(\tau p\log n)$ communication delay. On a mesh of size $\sqrt{p}\times\sqrt{p}$, Frederickson's algorithm selects the kth key from a file of size n using $O(p^{1+\frac{\epsilon}{2}}\frac{\log n}{\log p})$ messages with $O(\tau p^{1/2}\frac{\log n}{\log p})$ communication delay, or using $O(p\log^{1/2} p\log n)$ messages with $O(\tau p^{1/2}\log n)$ communication delay. On a p-node binary tree network, his algorithm performs selection using $O(p\log n)$ messages, with $O(\tau\log^2 p\log n)$ communication delay. The sampling technique used by Frederickson [6] is a variant of [11]. In [21], Santoro et al. presented a distributed selection algorithm, which uses $O(p\log\log n)$ messages on an average and uses $O(p\log n)$ messages in the worst case for a point-to-point communication model. The

[a]The sorting problem we defined is referred to as *static sorting problem* in [18].

performance analysis in terms of communication delay was not given. A lower bound given in [6] shows that selection of the median from a file of n keys evenly distributed in a p-node complete interconnection network, in the worst case, requires at least $\Omega(p \log n)$ messages. Due to this deterministic lower bound, in order to obtain a selection algorithm in which the message complexity is asymptotically lower than the lower bound, we use a *randomized* sampling technique which is a variant of the sampling technique reported in [5]. Given a file with n keys and a p-node de Bruijn network or hypercube, our algorithm performs selection using only $O(p \log \log n)$ messages and suffering a delay of $O(\tau \log p \log \log n)$ *with high probability.* By "with high probability", we mean in the probability of at least $(1 - n^{-\alpha})$ for any fixed α, where n is the input size of the problem being solved, the algorithm will correctly solve the problem in the given message complexity and communication delay. Our randomized selection algorithm beats the said deterministic lower bound in terms of message complexity and outperforms those existing algorithms in terms of both the message complexity and the communication delay.

Distributed sorting of a large file is of vital importance in very large database system in that a sorted distributed file can provide much faster data retrieval or other data manipulation operations. In [23], Wegner presents a distributed sorting algorithm which sorts a file of size n over p nodes by sending $O(np)$ messages, in the worst case, and uses an expected $O(n \log n)$ messages on an average. Rotem et al. studied the distributed sorting problem in [18]. Though the algorithm was not given in detail, according to their study, a sorting of a file of size n can be done in $O(n)$ messages using a completely connected network. In [8], Hofstee et al. presented a distributed sorting algorithm in Dijkstra's command language. The message complexity of their algorithm is $O(n \log \frac{n}{p})$ where n is the number of elements and p is the number of process. All of the above research works on distributed sorting ignore the communication structure of a network on which a distributed algorithm runs, and thus the performance analysis in terms of communication delay was not given. Besides, the communication structure of a network affects the message complexity of the algorithm designed for the network. Using our selection algorithms to select pivot element(s) we develop a quicksort-based sorting scheme which, in the worst case, sorts a distributed file of size n on a p-node hypercube using $O(n \log^2 p)$ messages. We also show a lower bound of $\Omega(n \log p)$ for sorting a file of size n on a p-node network with a diameter $\log p$. We then develop a sorting algorithm which sorts a distributed file of size n on a p-node hypercube or de Bruijn network in $O(n \log p)$ messages, which is optimal in terms of the message complexity. Our distributed sorting scheme is fully distributed without a central control.

Development of our selection and sorting schemes is carried out as follows. First, we develop a consensus protocol (a distributed version of a prefix computation). Second, we develop a distributed sparse enumeration sorting scheme, where each node of the network contains at most one key and there are at most $\sqrt{p}$ keys in the network to begin with. This sorting leads to the ith node ($i \leq \sqrt{p}$) holding the ith key of the sorted list. Third, using the consensus protocol and the distributed sparse enumeration sort, a randomized selection scheme is developed. Finally, using the idea of selection, we sort the n keys distributed among the p nodes. The rest of the paper is thus organized as follows. Section 2 defines the computation models and gives some facts that will be helpful throughout. Section 3 presents a consensus protocol and the distributed sparse enumeration sorting scheme. The randomized technique for selecting the keys of a large file is presented in Section 4. Section 5 presents several sorting algorithms. Finally, some concluding remarks are given in Section 6.

2. Preliminaries

2.1. *Definition of Models*

Though our selection algorithms are applicable on a variety of networks, we only employ the de Bruijn network and the hypercube as examples.

De Bruijn Networks

A d-ary directed de Bruijn network $DB(d,m)$ has $p = d^m$ nodes [3, 13, 24]. A node v is labelled as $d_m d_{m-1} ... d_1$ where each d_i is a d-ary digit. Node $v = d_m d_{m-1} ... d_1$ is connected to the nodes labelled $d_{m-1} ... d_2 d_1 r$, denoted by $SH(v,r)$, where r is an arbitrary d-ary digit. One can view the link connecting v to $SH(v,r)$ as an output edge of v and an input edge of node $SH(v,r)$ though we assume that a link can be used to carry data in both directions. We emphasize on the sense of direction for the convenience of algorithm description. We also use $l_{SH(v,r)}$ to denote the outgoing link connecting to node $SH(v,r)$. One can easily see that any node in the network is reachable from any other node in exactly m steps although there might exist a shorter path. Also, it's not hard to see that for a p-node binary de Bruijn network, i.e. the one with $d = 2$, the diameter of the network is $\log p$, i.e. $m = \log p$.

Hypercube

A hypercube of dimension m, denoted by H_m, consists of $p = 2^m$ nodes [10]. Each node is labelled with an m-bit binary string, $b_m b_{m-1} \cdots b_2 b_1$. For any node u, there is a bidirectional link connecting u to node v if and only if labels (addresses) of u and v differ in exactly one bit position. It is easy to see that the diameter of the network is m since the maximum number of different bits between any two nodes is m. Also, the hypercube H_m can be constructed from two identical H_{m-1} by connecting the ith node of one H_{m-1} to the ith node of the other for $0 \le i < 2^{m-1}$. In other words, an H_m can be recursively partitioned into 2^k subhypercubes $H_{m-k}, 1 \le k \le m$. In the design of our quicksort based sorting scheme for the hypercube, we use these properties of hypercube. Also, we define $H_m(0)$ to be the subhypercube of H_m composed of nodes of the form $\overbrace{* \cdots * 0}^{m \text{ bits}}$, and define $H_m(1)$ to be the subhypercube H_m composed of nodes of the form $\overbrace{* \cdots * 1}^{m \text{ bits}}$.

Computational Model

We assume that the network size and the diameter of the network are known to each node of the network. We also assume that each node knows its own unique identity. Both assumptions have been justified in numerous previous works (see e.g. [3, 22]). We also assume that each link in the network can be used to carry data in both directions. The network is supposed to be fault-free, *i.e.* during the entire course of computation no node or link will become faulty. A message sent by a node is eventually delivered to the destination node after an arbitrary, but finite delay. The readers interested in fault-tolerant computing are referred to [1], which addresses fault-tolerant distributed sorting problem.

The message complexity is measured in terms of the total number of messages produced by all of the nodes together during the entire course of the running of the algorithm. In message count, if a message travels for h hops during the entire course of the algorithm, h messages, rather than only one message, will be counted. Thus a message complexity of $O(p \log n)$ could mean that each of p node has individully produced $O(\log n)$ messages[b], or that all of p nodes together produce $O(p \log n)$ messages during the entire course of the algorithm. Throughout the paper a message means a key, a record, or a number with a constant number of bytes. Since a message takes up a constant number of bytes, the space complexity of the algorithm is the same as its message complexity. Communication delay is measured in terms of the number of hops needed for a packet (message) to travel from its source to its destination and the maximum transmission delay on a link. We let τ be the maximum transmission delay on a link.

2.2. *Some Facts from Probability Theory*

We need the following lemmas in the analysis of our randomized algorithms.

Lemma 2.1. *Let X be the number of heads in r independent flips of a coin, and q be the probability of a head in a single flip. X is also known to have a binomial distribution $B(r,q)$. The following three facts are known as Chernoff bounds [2]:*

$$\mathbf{Pr}[X \geq w] \leq \left(\frac{rq}{w}\right)^{w} e^{w-rq},$$

$$\mathbf{Pr}[X \geq (1+\epsilon)rq] \leq exp(-\epsilon^2 rq/2), \text{and}$$

$$\mathbf{Pr}[X \leq (1-\epsilon)rq] \leq exp(-\epsilon^2 rq/3),$$

for any $0 < \epsilon < 1$, and $w > rq$.

[b]A message is normally produced by executing a "send" instruction in the algorithm.

Lemma 2.2. *(Chebyshev's Inequality) Let X be a random variable with expectation $\mathbf{E}(X) = \mu_X$ and standard deviation σ_X. Then for any real number c, we have*

$$\mathbf{Pr}[|X - \mu_X| \geq c\sigma_X] \leq \frac{1}{c^2}$$

Lemma 2.3. *Let $X_1, X_2, \cdots, X_m$ be independent random variables and let $X = \sum_{i=1}^{m} X_i$. Then $\sigma_X^2 = \sum_{i=1}^{m} \sigma_{X_i}^2$.*

3. Consensus Protocol and Distributed Sparse Enumeration Sort

In this section we present a consensus protocol and a distributed sparse enumeration sort which are basic operations in the development of schemes for selecting and sorting the keys of large distributed files.

3.1. *The Consensus Protocol*

Bermond and Konig [3] have developed a generic consensus protocol on the de Bruijn network. A consensus protocol can be viewed as a distributed version of prefix computation [4] which will be repeatedly invoked by our selection and sorting algorithms. As such, to help understand our algorithms, their algorithm is verbatim presented in Figure 1. Bermond's algorithm has been well designed with termination detection, which can provide useful synchronization when it is invoked as a subroutine by an invoking algorithm such as our selection or sorting algorithm. Be reminded that a distributed algorithm is designed in such a way that each node of the network individually runs the identical copy of the algorithm. All nodes of the network run asynchronously, i.e. the speed of the nodes are independent of each other. Synchronization is achieved by designing the algorithm in three-phase computations, *i.e.* (i) receives messages from adjacent nodes, (ii) does computation based on the received messages, and (iii) sends the computed values to its adjacent nodes. By repeatedly performing this three-phase computation, the desired function (e.g. consensus) can be eventually achieved. When this algorithm is invoked as a subroutine by another algorithm, it will be treated as a computation step by the invoking algorithm and is thus represented by a single invoking statement in the invoking algorithm. Since this algorithm (which performs consensus or a distributed prefix computation) is with termination detection, when the invoking algorithm is running (Be reminded that each node of the network runs an identical copy of the invoking algorithm), no node can proceed to the next step of the invoking algorithm until the invoked algorithm has terminated at every node. The algorithm can be invoked spontaneously or upon receiving a message.

Bermond's algorithm can be used to compute various prefix computation functions by using different computations in Step 3.4.c of the protocol. For example, if we replace the statement "Compute R" of Step 3.4.c by a statement of summing up all the data items in New, the function of this protocol will be computing a sum of those data items contributed from each node in the network. It's not hard to see that this algorithm can achieve a consensus or finish a prefix computation on a p-node binary de Bruijn network using $O(p \log p)$ messages with $O(\tau \log p)$ communication delay. We can also apply the algorithm to the hypercube by just tailoring the algorithm in such a way that in phase i the set IN of incoming links contains only those links along the $(i-1)$th dimension and only send computed data to those outgoing links along the ith dimension.

In fact, the number of messages needed to achieve a consensus or finish a distributed prefix computation can be reduced to $O(p)$ by slightly modifying Bermond's algorithm. This modified algorithm for distributed prefix computation, stated in what follows, will be used as a building block in our distributed randomized selection algorithm. We give a high level description of this modified algorithm in the proof of the following lemma.

Lemma 3.1. *A prefix computation can be realized in a decentralized manner on a p-node de Bruijn network or hypercube using $O(p)$ messages with $O(\tau \log p)$ communication delay.*

Proof. A basic difference between the algorithm of Bermond and the modified one is explained as follows. On the one hand, the entire course of the computation of Bermond's algorithm can be thought of as a superposition of p prefix computation trees. On the other hand, in our modified distributed prefix computation, we allow exactly one prefix computation tree to participate in computation, thereby reducing the total number of messages needed.

The modified distributed prefix computation algorithm works as follows. The node in which the prefix computation was issued marks itself. It then sends a special message along both incoming links. The nodes which received the special message from the outgoing link(s)[c] pass the received special message to the neighboring nodes via the incoming links. After $\log p$ rounds of receiving and passing of the special message, each node in the network will receive the message. Thus the receiving and passing of the special message will be repeated for exactly $\log p$ times. In the ith iteration, the nodes which receive the special message update their variable OUT_i to include the outgoing link on which the special message was received. The variable OUT_i will be used later to control the behavior of the consensus algorithm such that exactly one prefix computation tree is involved in the computation. We then execute Bermond's algorithm with a modification. We replace Step 3.2, namely "**for** each outgoing link **do** send $\langle Phase, New \rangle$" by "**for** each outgoing link in OUT_{Phase} **do** send $\langle Phase, New \rangle$". The effect of this computation is that the computed results will be obtained only in those nodes which are marked. Then depending on the applications, there may be a need to broadcast the computed result to all nodes by simply sending the result from the marked nodes along the links which had

[c]Note that if there is only one marked node, there will be only one outgoing link receiving the message.

Variables used in each node P:
Inf: The global information known by P (To begin with, Inf consists of the identity of the node P plus its initial data).
New: The new information obtained by P since the beginning of the current phase.
$Phase$: The number of phase at which P is executing.
R: The result obtained by P. It depends on the application.
l: Represents an incoming link of P.
IN: The set of incoming links on which P can receive messages (when P receives a message "end" on some incoming link l it deletes this link from IN).

```
INITIATION
1.          $New = Inf$
2.          $Phase = 0$
PHASES
3.          while $New \neq \emptyset$ do
            begin
3.1               $Phase = Phase + 1$
3.2               for each outgoing link do send $\langle Phase, New \rangle$
3.3               $New = \emptyset$
3.4               for each incoming link $l$ in $IN$ do
                  begin
3.4.a                   receive $\langle Phase, I \rangle$ on $l$
3.4.b                   if $I =$"end"
                        then $IN = IN - \{l\}$
                        else begin
                             $New = New \cup (I - Inf)$
                             $Inf = Inf \cup I$
                             end
3.4.c                   Compute $R$
                  end
            end
TERMINATION
4.          $Phase = Phase + 1$
5.          for each outgoing link do send $\langle Phase, \text{"end"} \rangle$
6.          while $IN \neq \emptyset$ do
            begin
6.1               for each incoming link $l$ in $IN$ do
                  begin
6.1.a                   receive $\langle Phase, I \rangle$ on $l$
6.1.b                   if $I =$"end" then $IN = IN - \{l\}$
                  end
6.2               $Phase = Phase + 1$
            end
```

Fig. 1. A general consensus protocol.

ever been traveled by the special message. Clearly, the one and only one prefix computation tree needs no more than $2\sum_{i=1}^{\log p} 2^i = O(p)$ messages. Thus the revised version of distributed prefix computation uses only $O(p)$ messages while it never asymptotically increases the communication delay. □

3.2. *A Distributed Sparse Enumeration Sorting Algorithm*

To guarantee an $O(p)$ message complexity, in addition to using the distributed prefix computation presented in the previous subsection, our randomized selection algorithm also uses a distributed sparse enumeration sort as one of the building blocks. In this subsection, we present this sparse enumeration sorting scheme. We give a high level description of this sparse enumeration sort in the proof of the following lemma.

Lemma 3.2. *For any fixed $\epsilon \leq \frac{1}{2}$, a set of p^ϵ keys distributed in a p-node hypercube H_m ($m = \log p$), with no more than one key per node, can be sorted in $O(p)$ messages with $O(\tau \log p)$ communication delay.*

Proof. The proof is done by giving a four-step sorting scheme. **1)** Using the modified algorithm, perform a distributed prefix computation to assign a unique label to each key from the range $[1, p^\epsilon]$. **2)** Route these keys to a sub-hypercube of size $p^{\frac{1}{2}}$—a packet whose label is q can be routed to a node indexed q in the subhypercube. Note that the routing is a monotone routing and, thus, a greedy routing can be performed requiring only $O(\sqrt{p} \cdot \log p) = O(p)$ messages and with $O(\log p)$ communication delay. With this prefix computation and routing step we basically concentrate the keys to be sorted in a subhypercube whose size is $p^{1/2}$. If $\epsilon < \frac{1}{2}$, some tail nodes (in lexicographic order) may not receive a key and thus use ∞ as a pseudo-key. Let the subhypercube in which the keys are concentrated be an $H_{\frac{1}{2}\log p}$.

3) Next we make a copy of these keys in every $H_{\frac{1}{2}\log p}$ in the hypercube. The number of such copies made will be $\sqrt{p}$. These copies can be made in $O(p)$ messages with $\frac{1}{2}\log p = O(\log p)$ communication delay, because in the ith step we can double the copies by sending the keys via the links along the $(i + \frac{1}{2}\log p)$th dimension, which requires $\sqrt{p} \cdot 2^{i-1}$ messages. Thus, the entire copying process requires $\sqrt{p}\sum_{i=1}^{\frac{1}{2}\log p} 2^i = O(p)$ messages. If $H^1_{\frac{1}{2}\log p}, H^2_{\frac{1}{2}\log p}, \ldots, H^t_{\frac{1}{2}\log p}$ is the sequence of $H_{\frac{1}{2}\log p}$'s in H_m, we make use of the copy in $H^r_{\frac{1}{2}\log p}$ to compute the rank of the rth key, i.e., the key whose label is r (as computed in step 1). Rank computation is done using the modified distributed prefix algorithm. **4)** Finally we route the key whose rank is j to the node indexed j in $H^1_{\frac{1}{2}\log p}$. Once again the routing is a monotone one.

According to Lemma 3.1, distributed prefix computation uses $O(p)$ messages and suffers $O(\tau \log p)$ delay. Routing uses $O(p)$ messages and suffers $O(\tau \log p)$ delay. Copying of keys also uses $O(p)$ messages and suffers $O(\tau \log p)$ delay. Thus, in summary, the sparse enumeration sort can be done in $O(p)$ messages with $O(\tau \log p)$ communication delay. □

Using a similar scheme and the sparse enumeration sorting of Nassimi and Sahni [12] for perfect shuffle networks, we can also prove the following lemma.

Lemma 3.3. *For any fixed $\epsilon \leq \frac{1}{2}$, a set of p^ϵ keys distributed in a p-node de Bruijn network, with no more than one keys per node can be sorted in $O(p)$ messages with $O(\tau \log p)$ communication delay.*

4. The Randomized Distributed Selection Algorithm

We are now ready to present the details of our randomized selection algorithm. The algorithm is designed based on random sampling [16, 17]. The basic idea in random sampling is as follows: (1) Sample a set S of $o(n)$ keys at random from the collection N of surviving keys (To begin with, N is the given file). (2) Identify two keys a and b in S such that, with high probability, the key to be selected is in between a and b. Also, if S' is the set of all input keys in between a and b, then $|S'|$ should be small enough so that we can directly process S'. The techniques of recursive randomized selection of Floyd and Rivest [5] can not be directly used in the design of our distributed randomized selection algorithm. This is because in a recursive version of random sampling the size of sample set S, which contains information for which we are seeking the kth key, is shrinking as recursion proceeds. It can be shown that in a typical stage of recursion the probability of failure at that stage is $O(|S|^{-\alpha})$. But $|S|$ is diminishing such that the probability of failure is rising as the algorithm proceeds. Therefore, the failure probability is very high when the size of sample set is very small, say a constant. This is a fundamental barrier with recursive random sampling and is also a challenging problem to handle in developing distributed algorithms using random sampling. A feasible solution is to stop the algorithm when the problem size (or the size of sample set) is reduced to a certain size[d] and switch to a different technique, say sparse enumeration sorting.

Our selection algorithm is given in Figure 2. Be reminded that each node of the network has an identical copy of the algorithm and these nodes work together (Noticed that each node individually execute its own copy of the algorithm.)to achieve the task of distributed selection. Throughout our selection algorithm, the algorithm of Lemma 3.1 is employed several times for distributed prefix computations. We assume that the algorithm selects the kth key from a file of size n. To begin with, each key of the file is alive. To obtain a small set S' of keys from the given large file such that the key to be selected is extremely likely to be in S', we have to repeatedly execute Step 1 through Step 5 for several times (the number of times needed for the **repeat** block is analyzed in the proof of the theorem). If kth key is the one originally to be selected, since the value of k is updated accordingly in Step 5 of the **repeat** block, kth key in S' remains the one originally desired. Step 6 sorts the keys in S' such that the jth key of S' moves to jth node for all $j, 1 < j < |S'|$. Step 7 is to report the desired key, the kth key of the originally large file. Be advised that only node k (Noticed that the current value of k may be different from the original value of k.) needs to report this key.

Theorem 4.1. Selection on a file F can be distributedly performed on a p-node de Bruijn network or hypercube in $O(p \log\log n)$ messages with communication delay $O(\tau \log p \log\log n)$ with probability $1 - p^{-\alpha}$.

[d]We choose square root of the network size as the certain size to obtain the desired performance.

```
0.  Contribute the size of local file to the prefix computation and trigger or participate in
    a prefix computation to obtain the size $n$ of the entire file.
    (*To begin with, each key in each local memory is alive.*)
    $N = n$.
    repeat
1.      Flip an $N^\epsilon$-sided coin for each alive key in the local memory.
        An alive key gets included in the random sample, $S$, with probability $N^{-\epsilon}$.
        $\epsilon$ is chosen in such a way that $|S| \le p^{\frac{1}{2}}$. (*The value of $\epsilon$ is given in the analysis.*)
2.      Let $\eta_{P_i}$ be the number of alive keys included in $S$.
        Contribute $\eta_{P_i}$ and trigger or participate in a prefix computation to elect
        $\eta = \max_{P_i}\{\eta_{P_i}\}$.
        repeat for $\eta$ times
            If there still are more than 1 alive keys included in $S$, contribute 1;
            Otherwise contribute 0.
            Trigger or participate in a prefix computation to obtain the label $q$
            for the key to be contributed if there is one.
            If 1 was contributed, using the randomized routing scheme of [14]
            to route the key to node $q$.
        end repeat
        (*The function of Step 2 is to concentrate the sample keys.*)
3.      Perform the sparse enumeration sorting of Lemma 3.2.
4.      Let $x = \lceil k\frac{|S|}{N}\rceil = \lceil k \cdot N^{-\epsilon}\rceil$.
        If $i$ (the label of the node) is equal to $\max\{1, x - N^{\frac{3}{4}(1-\epsilon)}\}$,
        then mark the key received from sorting (Step 3) as $l$ and
        trigger a prefix computation to broadcast key $l$.
        If $i$ (the label of the node) is equal to $\min\{x + N^{\frac{3}{4}(1-\epsilon)}, |S|\}$,
        then mark the key received from sorting (Step 3) as $h$ and
        upon receiving key $l$, trigger a prefix computation to broadcast key $h$.
5.      Count the number, $\breve{a}_i$, of alive keys with a value in the range $[key_l, key_h]$.
        Contribute $\breve{a}_i$ and trigger a prefix sum to obtain $\breve{a} = \sum_i \breve{a}_i$.
        Also count the number, $\grave{a}_i$, of alive keys with a value $< key_l$.
        Contribute $\grave{a}_i$ and trigger a prefix sum to obtain $\grave{a} = \sum_i \grave{a}_i$.
        If $k$ is not in the interval $(\grave{a}, \grave{a} + \breve{a}]$ or if $\breve{a} \ne O(N^{\frac{3}{4}+\frac{1}{4}\epsilon})$ go to Step 1.
        Mark those alive keys (in the local memory) that are $< key_l$ or $> key_h$ as dead.
        Set $k = k - \grave{a}$.
    until $N \le p^{\frac{1}{2}}$
6.  Perform Step 2 to concentrate surviving keys and perform Step 3 to sort these keys.
7.  If the label of the node is $k$, report the unique alive key.
```

Fig. 2. The randomized selection algorithm.

Proof. The message bound and communication delay can be obtained by examining each step of the **repeat** loop followed by an estimation of the number of times the **repeat** loop will be executed.

Assuming that $N = O(p^t)$, we choose ϵ to be $1 - \frac{1}{2.1t}$ such that with high probability $|S| = O(N^{1-\epsilon}) \le p^{\frac{1}{2}}$. Though N is diminishing in each iteration, we can always perform an extra prefix computation so that each node knows the new N and thereby choose an appropriate ϵ. In each iteration, Step 1 needs only local computation. In Step 2, using Chernoff bounds, it can be shown that with probability $1 - N^{-\alpha}$, $\alpha > 0$, $\eta = O(1)$ and, thus, the **repeat** loop of Step 2 will be executed for only $O(1)$ times. And both the prefix computation (Lemma 3.1) and randomized routing [14] use $O(p)$ messages and suffer $O(\tau \log p)$ delay with probability $1 - p^{-\alpha}$ for some constant α. Thus, Step 2 requires only $O(p)$ messages with $O(\tau \log p)$ communication delay with probability $1 - p^{-\alpha}$. According to Lemma 3.2, Step 3 uses $O(p)$ messages with $O(\tau \log p)$ communication delay with probability $1 - p^{-\alpha}$. Steps 4 and 5 perform several prefix computations (in addition to some local computations) and thus can also be done in $O(p)$ messages with $O(\tau \log p)$ communication delay with probability $1 - p^{-\alpha}$.

We now consider the failure that the kth key we seek falls outside of S'. There are two possible cases for this mode of failure, namely $key_k < key_l$ and $key_k > key_h$. We shall only prove the case of $key_k < key_l$. (The other case can be proved similarly.) This case happens when fewer than l keys in sample set S are less than or equal to key_k. Let N be the set of surviving keys and let $X_i = 1$ if the ith key in S is at most key_k, and 0 otherwise. Also let $|\mathsf{N}| = N$. Then, $\mathbf{Pr}[X_i = 1] = \frac{k}{N}$ and $\mathbf{Pr}[X_i = 0] = 1 - \frac{k}{N}$. Let $X = \sum_{i=1}^{|S|} X_i$ be the number of keys of S that are $\leq key_k$. Because each X_i is a *Bernoulli trial*, X has a binomial distribution. We thus have

$$\mu_X = \mathbf{E}[X] = N^{1-\epsilon}\frac{k}{N},$$

and

$$\sigma_X^2 = \mathbf{var}[X] = N^{1-\epsilon}\left(\frac{k}{N}\right)\left(1 - \frac{k}{N}\right) \leq \frac{1}{4}N^{1-\epsilon}.$$

Therefore, $\sigma_X \leq \frac{1}{2}N^{\frac{1}{2}(1-\epsilon)}$. Using Chebyshev's Inequality, we have

$$\mathbf{Pr}[|X - \mu_X| \geq N^{\frac{3}{4}(1-\epsilon)}] = \mathbf{Pr}[|X - \mu_X| \geq 2N^{\frac{1}{4}(1-\epsilon)}\sigma_X] = O(N^{-\frac{1}{2}(1-\epsilon)}).$$

The second mode of failure is that S contains more than $O(N^{\frac{3}{4}+\frac{1}{4}\epsilon})$ keys. Let $x_r = rank(select(r, S), \mathsf{N})$. We can prove that

$$\mathbf{E}[x_r] = r\frac{N}{|S|}. \tag{1}$$

Therefore, if we select a key x from S with rank $\lceil k\frac{|S|}{N}\rceil$, the expected rank of x in N, $\mathbf{E}[rank(x, \mathsf{N})]$, is $k\lceil\frac{|S|}{N}\rceil \times \frac{N}{|S|} = k$. We can also prove that

$$\sigma_{x_r} \leq \frac{1}{2}\frac{N}{|S|^{1/2}}. \tag{2}$$

Let $\mathbf{E}[|S'|]$ be the expected number of keys lying in between key_l and key_h. Then we have

$$\mathbf{E}[|S'|] = \mathbf{E}[rank(key_l, \mathsf{N})] - \mathbf{E}[rank(key_h, \mathsf{N})]$$

$$\begin{aligned}
&= \frac{N}{|S|}(k\frac{|S|}{N} + N^{\frac{3}{4}(1-\epsilon)}) - \frac{N}{|S|}(k\frac{|S|}{N} - N^{\frac{3}{4}(1-\epsilon)}) \\
&= \frac{N}{|S|} \cdot 2N^{\frac{3}{4}(1-\epsilon)} \\
&= 2N^{\epsilon} \cdot N^{\frac{3}{4}(1-\epsilon)} \\
&= 2N^{\frac{3}{4}+\frac{1}{4}\epsilon}
\end{aligned}$$

Let $\mu_{X_k} = \mathbf{E}[rank(key_k, \mathsf{N})]$. By (2), we have $\sigma_{X_k} \leq \frac{1}{2}\frac{N}{|S|^{1/2}} = \frac{1}{2}N^{\frac{1}{2}+\frac{1}{2}\epsilon}$. Then using Chebyshev's Inequality, we have

$$\mathbf{Pr}[|rank(key_k, \mathsf{N}) - \mu_{X_k}| \geq N^{\frac{3}{4}+\frac{1}{4}\epsilon}] = \mathbf{Pr}[|rank(key_k, \mathsf{N}) - \mu_{X_k}| \geq 2N^{\frac{1}{4}-\frac{1}{4}\epsilon}\sigma_{X_k}] = O(N^{-\frac{1}{2}(1-\epsilon)}).$$

Consequently, each iteration of the **repeat** loop can be done in $O(p)$ messages with $O(\tau \log p)$ communication delay with probability $1 - N^{-\alpha}$, N being the number of alive keys at the beginning of this iteration. Because we assume that $N = O(p^t)$ and we choose ϵ to be $1 - \frac{1}{2.1t}$, the probability of failure of both modes is $O(p^{-\alpha})$ for some constant α.

We then show that the expected number of times that the **repeat** loop is executed is $O(\log \log n)$. If there are N alive keys at the beginning of any iteration of the loop, then the number of remaining alive keys at the end of this iteration is $O(N^{\frac{3}{4}+\frac{1}{4}\epsilon}) = O(N^{\epsilon'})$ (for any fixed $\epsilon' < 1$) with high probability. This implies that the expected number of times the **repeat** loop is executed is $O(\log t + 1) = O(\log \log n)$. Steps 0 and 7 can also be performed in $O(p)$ messages with $O(\tau \log p)$ communication delay with probability $1 - p^{-\alpha}$. □

We can also trade message complexity and running time for the lower failure probability. This leads to the following corollary.

Corollary 4.1. *Selection on a file of size n can be distributedly performed on a p-node de Bruijn network or hypercube in $t \cdot O(p) = O(\frac{p \log n}{\log p})$ messages with communication delay $t \cdot O(\tau \log p) = O(\tau \log n)$ with probability $1 - n^{-\alpha}, \alpha > 1$.*

5. Sorting Large Distributed Files on the Hypercube and de Bruijn Networks

In this section we present a quicksort-based sorting scheme which uses the selection of previous section for sorting a large distributed file on the hypercube. Using our randomized selection algorithms, we also develop an enumeration sorting scheme for sorting large distributed files in both de Bruijn networks and the hypercube. A common index scheme used for sorting a distributed file F of size n in a p-node network is that each key, key_i, will be residing in $\lceil rank(key_i, F) \cdot \frac{p}{n} \rceil$th node after sorting. We first present a lower bound for sorting a distributed file in the hypercube in Lemma 5.1.

Lemma 5.1. *Sorting a distributed file F of size n in a p-node hypercube in the worst case requires at least $\Omega(n \log p)$ messages and delay $\Omega(\tau \frac{n}{p})$.*

Proof: WLOG, assume that n divides p. For any key, key_i, if initially key_i resides in node $P_{b_n b_{n-1} \cdots b_2 b_1}$ and $\lceil rank(key_i, F) \cdot \frac{p}{n} \rceil = \sum_{i=1}^{n} \overline{b_i}$[e], the key needs $\log p$ hops to reach its destination (node). If each of n keys suffers the same situation, in total we need at least $n \log p$ messages for each key to reach its right position (node). Since we have $\Omega(n \log p)$ messages to be consumed and in a p-node hypercube we have only $p \log p$ links to evacuate these messages, it suffers at least $\frac{n}{p}$ delay. □

Using a similar argument, we can also prove the following lemma.

Lemma 5.2. *Sorting a distributed file F of size n in a p-node de Bruijn network in the worst case requires at least $\Omega(n \log p)$ messages and delay $\Omega(\tau \frac{n}{p} \log p)$.*

Proof: The reason that it suffers more delay in the de Bruijn network is that de Bruijn network has only $O(p)$ links to evacuate $O(n \log p)$ messages. □

We then present our sorting algorithm which can sort a distributed file of size n in a p-node hypercube in $O(n \log^2 p)$ messages with $O(\frac{n}{p} \log^2 p)$ communication delay. This algorithm is near optimal in the sense of both message complexity and communication delay. The basic idea behind our algorithm is: (1) select a median from the keys in current (sub)hypercube, H_i (initially $i = n$), and broadcast the median to each node in current H_i; (2) each node in H_i compares each key in its local memory with the received median and mark a key as 0 if it is less than or equal to the median, and 1 otherwise; (3) each node in $H_{i-1}(0)$ sends those keys (in its own local memory) marked 1 via the link along the ith dimension to the node[f] in $H_{i-1}(1)$, and each node in $H_{i-1}(1)$ sends those keys marked 0 via the link along ith dimension to the node in $H_{i-1}(0)$; (4) perform prefix computation and packing (monotone routing) for at most $\log p$ times to balance the load such that each node has $\frac{n}{p}$ keys; (5) repeat (1)-(4) for $\log p$ times. This way, upon termination of the algorithm, each key will be in its right position (node), i.e. each key in node P_i is less than or equal to each key in node P_j, for all i, j, $1 \le i < j \le p$. To do so, each node in the network individually executes the following algorithm.

```
$i = 0$
**repeat**
1.   Perform selection algorithm of Figure 2 to select $\lfloor \frac{n}{p} \rfloor$th key.
     (*The node which holds the selected key will broadcast the key to every node
     by triggering a prefix computation. Label the selected key as median.*)
2.   Compare each key in the local memory with the received median.
     Mark a key as 0 if it is less than or equal to the median, and 1 otherwise.
     If $i$th bit of the address of the node is 0,
     then send those keys marked 1 to the adjacent node via the link along $i$th dimension;
     Otherwise, send those keys marked 0 to the adjacent node via the link along $i$th dimension;
3.   Let $\eta_{P_i}$ be the number of keys in the local memory.
     Contribute $\eta_{P_i}$ and trigger a prefix computation to elect $\eta = \max_{P_i}\{\eta_{P_i}\}$.
     **repeat** for $\eta$ times
             If there still are more than 1 old keys in local memory, contribute 1;
```

[e] $\overline{b_i}$ denotes the complement of b_i.
[f] Note that the node in $H_{i-1}(1)$ is the adjacent node of the one in $H_{i-1}(0)$ along the ith dimension.

```
            Otherwise contribute 0.
            Trigger a prefix computation to obtain the label $q$ for the key to be packed.
            If 1 was contributed, using a greedy routing scheme to route the key to node $q$.
            (*The routing is a monotone one.*)
        end repeat (*The node will receive exactly $\frac{n}{p}$ new keys.*)
4.      Close the link along the $i$th dimension.
        $n = \frac{n}{2}$; $p = \frac{p}{2}$.
5.      $i = i + 1$
until $i = \log p$
```

Analysis

The message bound and communication delay can be obtained by examining each step of the **repeat** loop. Assume that each node can only process one key at a time. According to Theorem 4.1, Step 1 can be performed in $O(p \log\log n)$ messages with $O(\tau \log p \log\log n)$ communication delay. Step 2 can be done in n messages and with $\tau\frac{n}{p}$ communication delay in the worst case. In Step 3, in the worst case, η will be $2\frac{n}{p}$. It thus needs $2\frac{n}{p} \cdot p + n\log p = O(n \log p)$ messages for balancing the keys among nodes. The communication delay in Step 3 is clearly $O(\tau\frac{n}{p}\log p)$. Steps 4 and 5 need only local computation. Thus each iteration of the **repeat** loop generates $O(n \log p)$ messages and contributes a communication delay of $O(\tau\frac{n}{p}\log p)$. The **repeat** loop is performed for exactly $\log p$ times. We thus have the following theorem.

Theorem 5.1. Sorting of a distributed file of size n can be distributedly performed on a p-node hypercube in $O(n\log^2 p)$ messages and with communication delay $O(\tau\frac{n}{p}\log^2 p)$, provided $n = \Omega(p\log^2 p)$, with probability $1 - n^{-\alpha}, \alpha > 1$.

This quicksort-based sorting scheme doesn't work on the de Bruijn network due to the fact that quick-sort based sorting is a recursive scheme whereas the de Bruijn network is not a recursively constructed network. Fortunately, using a different approach, the following sorting scheme performs efficient sorting on the de Bruijn network by repeatedly employing selection and routing algorithms.

```
0.      $i = 1$
1.      repeat
1.a             Perform a selection algorithm to select $i\lfloor\frac{n}{p}\rfloor$th key.
                (*The node which holds the selected key will broadcast the key to every node
                by triggering a prefix computation.*)
1.b             Compare each unmarked key in the local memory with the selected key.
                Label an unlabelled key as $i$ if it is less than or equal to the selected key.
1.c             $i = i + 1$
        until $i = p + 1$
        $i = 1$
2.      repeat
2.a             Route the key in memory cell $i$ to node $r$ (assuming that the label of the key is $r$)
                using a greedy routing scheme.
2.b             $i = i + 1$
        until $i = \frac{n}{p} + 1$
```

If we employ the selection algorithm of Figure 2 in the above sorting scheme, we obtain the following theorem.

Theorem 5.2. Sorting of a distributed file of size n can be distributedly performed on a p-node de Bruijn network in $O(n \log p)$ messages and with communication delay $O(\tau n)$ provided $\frac{n}{\log \log n} = \Omega(\frac{p}{\log p})$, with probability $1 - n^{-\alpha}, \alpha > 1$, which is nearly optimal.

Proof: Selection requires $O(p \log \log n)$ messages and $O(\tau \log p \log \log n)$ delay, and is executed for p times. Also, in the worst case, each of n keys needs $\log p$ hops to reach its destination. It, thus, requires $O(p \log \log n) + n \log p = O(n \log p)$ messages in total provided $\frac{n}{\log \log n} = \Omega(\frac{p}{\log p})$. Since each node has $\frac{n}{p}$ keys, the greedy routing algorithm will be executed for $\frac{n}{p}$ times for routing p keys each time. In the worst case, each time each of p keys may go for the same destination (node) and it thus takes p steps for the destination node to process the received keys. Therefore, the algorithm will totally suffers $\tau n + O(\tau p \log \log n) = O(\tau n)$ communication delay.□

This selection-based enumeration sorting scheme can also be applied on the hypercube to obtain the following theorem. Comparing the quicksort-based sorting with the selection-based enumeration sort, one can see a trade off between message complexity and communication delay. However, it is obvious that the performance of both sorting schemes (for sorting large distributed files) is highly dependent on the performance of the selection algorithm employed.

Theorem 5.3. Sorting of a distributed file of size n can be distributedly performed on a p-node hypercube in $O(n \log p)$ messages and with communication delay $O(\tau n)$ provided $\frac{n}{\log \log n} = \Omega(\frac{p}{\log p})$, with probability $1 - n^{-\alpha}, \alpha > 1$, which is nearly optimal.

6. Conclusions

In this paper, we presented a randomized selection algorithm which performs selection of the kth key from a file of size n in a p-node de Bruijn network or hypercube using only $O(p \log \log n)$ messages and suffering $O(\log p \log \log n)$ communication delay. Our randomized selection outperforms the existing approaches in terms of both message complexity and communication delay. The property that the number of messages needed and communication delay are less sensitive to the size of the file makes our distributed selection schemes extremely attractive in the applications of very large database systems. Using our selection algorithms to select pivot element(s), we have also developed two near optimal sorting schemes for sorting large distributed files on hypercube and de Bruijn network. Our algorithms are fully distributed with no central control involved. According to the lower bound we have shown in this paper, the algorithm we have presented for sorting a large distributed file is still not optimal in the sense of communication delay. Searching a tighter lower bound or discovering an upper bound algorithm with matching time bounds (communication delay) is still open.

References

1. G. Alari, B. Bourgon, J. Chacko, and A.K. Datta, "Adaptive distributed sorting," Proceedings of the 1996 IEEE 15th Annual International Phoenix Conference on Computers and Communications, pp.1-7, 1996.
2. D. Angluin and L.G. Valiant, "Fast Probabilistic Algorithms for Hamiltonian Circuits and Matchings," Journal of Computer and System Sciences 18(2), 1979, pp. 155-193.
3. J.-C. Bermond and J.-C. Konig, "General and Efficient Decentralized Consensus Protocols II," Parallel and Distributed Algorithms, editors: M. Cosnard et al., North-Holland, 1989, pp. 199-210.
4. T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, McGraw Hill, 1991.
5. R.W. Floyd and R.L. Rivest, "Expected Time Bounds for Selection," *Comm. of the ACM*, vol.18, no. 3, March 1975, pp. 165-172.
6. G.N. Frederickson, "Tradeoffs for Selection in Distributed Networks," in *Proceedings of 2nd ACM Symposium on Principles of Distributed Computing*, 1983, pp. 154-160.
7. J.M. Helary and A. Mostefaoui, "A $O(log_2N)$ Fault-Tolerant Distributed Mutual Exclusion Algorithm Based on Open-Cube Structure," *IEEE ICDC'94*, June, 1994.
8. H.P. Hofsee, A.J. Martin, and J.L.A. Van De Snepscheut, "Distributed sorting," Science of Computer Programming, 15, 1990, pp. 119-133.
9. J.M. Helary, A. Mostefaoui, and M. Raynal, "A General Scheme for Token- and Tree- Based Distributed Mutual Exclusion Algorithms," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 5, No. 11, Nov. 1994, pp. 1185-1196.
10. F.T. Leighton, "*Introduction to Parallel Algorithms and Architectures: Arrays, Trees, and Hypercubes*," Morgan Kaufmann, 1992.
11. J.I. Munro and M.S. Paterson, "Selection and Sorting with Limited Storage," *Theoretical Computer Science* 12, 1980, pp. 315-323.
12. D. Nassimi and S. Sahni,"Parallel Permutation and Sorting Algorithms and a New Generalized Connection Network," *JACM*, July 1982, pp. 642-667.
13. R. K. Pankaj and R. G. Gallager, "Wavelength Requirements of All-Optical Networks," *IEEE/ACM Trans. on Networking*, Vol. 3, No. 3, June 1995, pp. 269-280.
14. M. Palis, S. Rajasekaran, and D.S.L. Wei, "Packet Routing and PRAM Emulation on Star Graphs and Leveled Networks," *Journal of Parallel and Distributed Computing*, vol. 20, no. 2, Feb. 1994,
15. S. Rajasekaran, W. Chen, and S. Yooseph., "Unifying Themes for Parallel Selection," Proc. *Fifth International Symposium on Algorithms and Computation*, August 1994.
16. S. Rajasekaran and J.H. Reif, "Derivation of Randomized Sorting and Selection Algorithms," in *Parallel Algorithm Derivation and Program Transformation*, Edited by R. Paige, J.H. Reif, and R. Wachter, Kluwer Academic Publishers, 1993, pp. 187-205.
17. S. Rajasekaran and S. Sen, "Random Sampling Techniques and Parallel Algorithms Design," in *Synthesis of Parallel Algorithms*, Editor: J.H. Reif, Morgan-Kaufman Publishers, 1993, pp. 411-451.
18. Rotem, D., Santoro, N., and S.J. Sidney, "Distributed Sorting," IEEE Trans. on Computers, 34 (4):372-376, 1985.
19. S. Rajasekaran and D. Wei, "Designing Efficient Distributed Algorithms Using Sampling Techniques," *11th International Parallel Processing Symposium, IEEE*, Geneva, Switzerland, April 1-5, 1997.
20. L. Shrira, N. Francez, and M. Rodeh, "Distributed K-Selection: From a Sequential to a Distributed Algorithm," in *Proceedings of 2nd ACM Symposium on Principles of Distributed Computing*, 1983, pp. 143-153.

21. Santoro, N., M., Sidney, J.B., and S.J. Sidney, "A distributed selection algorithm and its expected communication complexity," Theoretical Computer Science, 100(1992), pp. 185-204.
22. G. Tel, *Introduction to Distributed Algorithms*, Cambridge University Press, 1994.
23. L.M. Wegner, "Sorting a Distributed File in a Network," in *Proc. Princeton Conf. Inform. Sci. Syst.*, 1982, pp.505-509.
24. David S.L. Wei and K. Naik, "An Efficient Multicast Protocol Using de Bruijn Structures for Mobile Computing," *ACM Computer Communication Review*, vol. 27, no. 3, pp. 14-35, July 1997.