

# Designing Efficient Distributed Algorithms Using Sampling Techniques

S. Rajasekaran and D.S.L. Wei

University of Florida and University of Aizu

## Abstract

In this paper we show the power of sampling techniques in designing efficient distributed algorithms. In particular, we show that using sampling techniques, on some networks, selection can be done in such a way that the message complexity is independent of the cardinality of the set (file), provided the file size is polynomial in the network size. For example, given a file  $F$  of size  $n$  and an integer  $k(1 \leq k \leq n)$ , on a  $p$ -processor de Bruijn network, our deterministic selection algorithm can find the  $k$ th smallest key from  $F$  using  $O(p \log^3 p)$  messages and with a communication delay of  $O(\log^3 p)$ , and that our randomized selection algorithm can finish the same task using only  $O(p)$  messages and a communication delay of  $O(\log p)$  with high probability, provided the file size is polynomial in network size. 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 the communication delay are independent of the size of the file makes our distributed selection schemes extremely attractive in such domains as very large database systems. Making use of our selection algorithms to select pivot element(s), we also develop a near optimal quicksort-based sorting scheme and a nearly optimal enumeration sorting scheme for sorting large distributed files on the hypercube and de Bruijn networks. Our algorithms are fully distributed without any *a priori* central control.

## 1 Introduction

Numerous computing applications call for sorting very large files. Often these large files are distributed over a number of computing sites of an interconnection network (ICN) to gain speed of data retrieval or data manipulation. Due to the fact that the cost for passing a message from one site to an adjacent one is much higher than a singular local computation, distributed algorithms are always designed in such a way that the number of messages needed for synchronizing or encooperating the computations is minimized, while keeping the communication delay as small as possible [16]. In this paper, we consider the problems of selection and sorting in a distributed system. The computational model of the distributed system we adopt

is a set of computing sites connected together via a sparse network of communication links. Each computing site is an autonomous processor (We will use the term “processor” in the rest of the paper). These processors operate asynchronously and communicate with each other completely by passing messages. The interconnection networks we consider are the de Bruijn network and the hypercube. We assume that the network size and the diameter are known to each processor<sup>1</sup> of the network. This assumption has been justified in numerous previous works (see e.g. [1] [16]). 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 processor or link will become faulty. As such, each message is guaranteed to pass through a link within an arbitrary but finite delay. We let  $\tau$  be the maximum transmission delay on a link. We leave the problems involving fault-tolerant computing for further research. Throughout the paper a message means a key or a record or a number with a constant number of bytes.

Both selection and sorting problems have attracted considerable attention within the distributed computing community. In [15], Shriram *et al.* present a selection algorithm based on a CSP-like synchronous message passing model. Their algorithm can find  $k(1 \leq k \leq n)$ th key from a file of size  $n$  in  $O(pn^{0.91})$  messages using  $p$  processors. Using sampling techniques, Frederickson [4] designed three selection algorithms for networks of asynchronous message passing models. On a ring of  $p$  processors, his selection algorithm can find  $k$ th 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  $\sqrt{p} \times \sqrt{p}$ , his algorithm can select  $k$ th key from a file of size  $n$  in  $O(p^{1+\frac{\epsilon}{2}} \frac{\log n}{\log p})$  messages with  $O(\tau p^{1/2} \frac{\log n}{\log p})$  communication delay, or in  $O(p \log^{1/2} p \log n)$  messages with  $O(\tau p^{1/2} \log n)$  communication delay. And on a  $p$ -processor binary tree network, his algorithm can perform a selection using  $O(p \log n)$  messages, with  $O(\tau \log^2 p \log n)$  communication delay. The sampling technique that [4] adopts is a variant of [7]. In this paper we use a different version which is a variant of [2]. On both de Bruijn network and hypercube of size  $p$ , our selection algorithm can find the  $k$ th key from a file of size  $n$  in  $O(p \log^2 p \log n)$

<sup>1</sup>We also assume that each processor knows its own unique identity.

messages, with  $O(\tau \log^2 p \log n)$  communication delay. When  $n = f(p)$ , e.g.  $n = O(p^k)$  or  $n = 2^{O(p)^2}$ , both message complexity and communication delay of our algorithms and those of [4] are independent of the file size. This demonstrates that sampling techniques (see e.g. [12][13]) are very useful tools for designing efficient algorithms for processing very large distributed files. We also use a randomized sampling technique which is a variant of [3, 10] to design a much more efficient randomized selection algorithm. Given a file of size  $n$  and a  $p$ -processor de Bruijn network or hypercube,  $n$  is polynomial in  $p$ , our algorithm can perform a selection using only  $O(p)$  messages and delay  $O(\tau \log p)$  with high probability. Our randomized algorithm beats a deterministic lower bound given in [4]. Furthermore, we also use our selection algorithms to develop efficient sorting schemes for sorting large files. Sorting is of vital importance in very large database systems in that a sorted distributed file can provide much faster data retrieval or other data manipulation operations. In [17], Wegner presents a distributed sorting algorithm which sorts a file of size  $n$  in  $O(np)$  messages using  $p$  processors, in the worst case, and uses an expected  $O(n \log n)$  messages on an average. Making use of our selection algorithms to select pivot element(s), we develop a quicksort-based sorting scheme which in the worst case can sort a distributed file of size  $n$  on a  $p$ -processor hypercube in  $O(n \log^2 p)$  messages. Using our selection algorithms, we also develop both deterministic and randomized enumeration sorting algorithms which can sort a distributed file of size  $n$  on a  $p$ -processor hypercube or de Bruijn network in  $O(n \log p)$  messages, which is optimal in the sense of the message complexity. Our distributed sorting scheme is fully distributed whereas [17]'s is with central control.

The rest of this paper is organized as follows. Section 2 defines the computation models and gives some preliminary facts that will be helpful throughout. Section 3 presents deterministic selection algorithms. The more efficient randomized selection is presented in Section 4. Section 5 presents several sorting algorithms.

## 2 Preliminaries

### 2.1 Models Definition

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

---

<sup>2</sup>This is a practical assumption since for a moderate size network, the size of a very large file which might exist in the world can always be expressed as a polynomial function of  $p$  with a small degree, or be expressed as a function of  $p$ , which grows slower than  $2^{O(p)}$ .

### De Bruijn Networks

A  $d$ -ary directed de Bruijn network  $DB(d, n)$  has  $p = d^n$  processors. A processor  $v$  can be labelled as  $d_n d_{n-1} \dots d_1$  where each  $d_i$  is a  $d$ -ary digit. Processor  $v = d_n d_{n-1} \dots d_1$  is connected to the processors labelled  $d_{n-1} \dots d_2 d_1 r$ , denoted by  $SH(v, r)$ , where  $r$  is an arbitrary  $d$ -ary digit. One can easily see that any processor in the network is reachable from any other processor in exactly  $n$  steps although there might exist a shorter path.

### Hypercube

A hypercube of dimension  $n$ , denoted by  $H_n$ , consists of  $p = 2^n$  processors. Each processor is labelled with an  $n$ -bit binary string,  $b_n b_{n-1} \dots b_2 b_1$ . For any processor  $u$ , there is a bidirectional link connecting  $u$  to processor  $v$  if and only if addresses of  $u$  and  $v$  differ in exactly one bit. It is easy to see that the diameter of the network is  $n$ . A  $H_n$  can be recursively partitioned into  $2^k$  subhypercubes of  $H_{n-k}$ ,  $1 \leq k \leq n$ .

Bermond and Konig [1] have developed a decentralized general consensus protocol on the de Bruijn network. This algorithm can be used to perform data broadcasting, prefix computation, etc., which will be repeatedly invoked by our selection and sorting algorithms. This algorithm can achieve a consensus or finish a computation on a binary de Bruijn network using  $O(p \log p)$  messages with  $O(\tau \log p)$  communication delay. We show that by introducing randomization and modifying the algorithm slightly, the number of messages needed can be reduced to be  $O(p)$ . We also apply the algorithm to the hypercube:

**Lemma 2.1** *Prefix computation can be done decentralizedly on a  $p$ -processor de Bruijn or hypercube network using  $O(p \log p)$  messages with  $O(\tau \log p)$  communication delay.*

## 3 The Deterministic Selection Algorithm

Our selection algorithm makes use of the following Lemma:

**Lemma 3.1** *The sorting of  $p$  keys on a  $p$ -processor de Bruijn network, one key per processor, can be performed distributedly in  $O(p \log^2 p)$  messages with  $O(\tau \log^2 p)$  communication delay.*

*Proof*: The algorithm can be viewed as a distributed version of bitonic sort. There are  $\log p$  stages, each of which has  $\log p$  phases. The  $i$  ( $1 \leq i \leq \log p$ )th stage is to sort each of  $2^{\log p - i}$  bitonic sequences of length  $2^i$  into a sorted one such that each odd-even

pair of sorted sequences forms a bitonic sequence of length  $2^{i+1}$ . Since bitonic sorting guarantees that the right data will be compared at the right processors, we only need to prove that the right data are compared at the right time (phases)<sup>3</sup>. To do so, we want to prove that each phase does the right job by induction on the timing (phases). More details can be found in [14]. The algorithm can also be tailored for the hypercube with the same performance and we thus have the following lemma.

**Lemma 3.2** *The sorting of  $p$  keys on a  $p$ -processor hypercube, one key per processor, can be performed distributedly in  $O(p \log^2 p)$  messages with  $O(\tau \log^2 p)$  communication delay.*

We employ deterministic sampling in our selection algorithm. The sampling technique we employ is a variant of [2, 11]. A summary follows: (1) To group the numbers into groups with  $l$  elements in each group (for an appropriate  $l$ ); (2) Sort each group independently; (3) Collect each  $q$ th element from each group (for some  $q$ ). This collection serves as a “sample” for the original input. For example, the median of this sample can be shown to be an approximate median for the input. However, we adopt a different approach. Initially, there are  $\frac{n}{p}$  keys (or records) at each processor. As the algorithm proceeds, keys get dropped from future consideration. We don’t perform any load balancing. The remaining keys from each processor will form the groups. Instead of picking the median of these medians as the element for partition, we choose a weighted median of these medians. Each group median is weighted with the number of remaining keys in that processor.

We now give a high-level description of our algorithm. Each processor  $P_i$  individually performs the algorithm shown below, assuming that the algorithm is for finding  $k$ th key from a file of size  $n$ . The algorithm can be implemented on both de Bruijn network and hypercube, as the building blocks, namely consensus protocol (distributed prefix computation) and distributed sorting, have been developed for both models.

### Deterministic Selection

**0.** Contribute the size of local file to the prefix computation and trigger 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.** Find the median of alive keys in local memory. Let  $M_i$  be the median and  $N_i$  be the number of remaining alive keys.

**2.** Contribute  $M_i$  to the sorting and trigger the distributed permutation sorting.  $M_i$  is carrying with  $N_i$ . Let  $M'_i$  be the key received after sorting.

**3.** Contribute  $N'_i$  to the prefix computation and trigger a prefix sum computation. The value  $\omega$  obtained by  $P_i$  will be  $\sum_{m=1}^i N'_m$ . If  $\omega$  is  $< \frac{N}{2}$ , then notify  $P_{i+1}$ ; Otherwise, notify  $P_{i-1}$ .

**4.** If the obtained value  $\omega$  is  $\geq \frac{N}{2}$  and  $P_i$  receives a notification from  $P_{i-1}$ , then  $M'_i$  is weighted median  $M$  and a prefix computation is triggered to broadcast  $M$ .

**5.** Count the number of alive keys (in local memory) which are smaller than  $M$  and contribute the number to the prefix computation. Trigger a prefix computation to compute the total number of alive keys which are smaller than  $M$ . This is to count the rank of  $M$  out of all the remaining alive keys.

**6. if  $k \leq r_M$  then** mark those alive keys (in the local memory) that are  $> M$  as dead **else** mark those alive keys that are  $\leq M$  as dead.

**7.** Let  $E_i$  be the number of dead keys. Contribute  $E_i$  and trigger a prefix computation to compute  $E$ , the total number of dead keys (keys eliminated).

**8. if  $k > r_M$  then**  $k = k - E$ ;  $N = N - E$ .

**until**  $N \leq c$ ,  $c$  being a constant.

**9.** Trigger a prefix computation to elect a leader, e.g. the processor with maximum identity, and route the surviving keys to the leader.

**10.** If  $P_i$  is elected, perform a local sorting on the surviving keys. Report the  $k$ th key out of the surviving keys.

### Analysis

Steps 1, 6, and 8 need only local computations. Step 0 uses  $O(p \log p)$  messages and needs  $O(\tau \log p)$  communication delay according to Lemma 2.1. Likewise, Steps 3, 4, 5, and 7 can be done in  $O(p \log p)$  messages with  $O(\tau \log p)$  communication delay. In Step 2, we sort the medians and thereby compute the weighted median. Let  $M'_1, M'_2, \dots, M'_p$  be the sorted order of the medians, we identify  $j$  such that  $\sum_{m=1}^j N'_m \geq \frac{N}{2}$  and  $\sum_{m=1}^{j-1} N'_m < \frac{N}{2}$ . Such a  $j$  can be computed with an additional prefix computation. Thus the weighted

<sup>3</sup>There are totally  $\log p \times \log p = \log^2 p$  phases.

median,  $M$ , can be identified in  $O(p \log^2 p)$  messages with  $O(\tau \log^2 p)$  communication delay according to Lemma 3.1 or Lemma 3.2. Therefore, each run of the **repeat** loop uses  $O(p \log^2 p)$  messages and suffers  $O(\tau \log^2 p)$  communication delay.

The way we identify the weighted median guarantees that at least  $\frac{N}{4}$  keys are dropped out in each run of the **repeat** loop. Assume that  $k > r_M$  in a given run. (The other case can be proved similarly.) The number of keys dropped out is at least  $\sum_{m=1}^j \lceil \frac{N'_m}{2} \rceil$  which is  $\geq \frac{N}{4}$ . Consequently, the repeat loop will be executed for  $O(\log n)$  times. Besides, Step 9 also needs  $O(p \log p)$  messages and  $O(\tau \log p)$  communication delay. And Step 10 needs only local computations. In summary, the algorithm uses  $O(p \log^2 p \log n)$  messages to finish the selection with  $O(\tau \log^2 p \log n)$  communication delay. We thus get the following theorem.

**Theorem 3.1** *Selection on a file of size  $n$  can be decentralizedly performed on a  $p$ -processor de Bruijn network or hypercube in  $O(p \log^2 p \log n)$  messages with communication delay  $O(\tau \log^2 p \log n)$ .*

This theorem also leads to the following corollaries.

**Corollary 3.1** *Selection on a file of size  $n$  can be decentralizedly performed on a  $p$ -processor de Bruijn network or hypercube in  $O(p \log^3 p)$  messages with communication delay  $O(\tau \log^3 p)$  provided file size is polynomial in network size.*

**Corollary 3.2** *Selection on a file of size  $n$  can be decentralizedly performed on a  $p$ -processor de Bruijn network or hypercube in  $O(p^2 \log^2 p)$  messages with communication delay  $O(\tau p \log^2 p)$  provided file size is exponential in network size.*

## 4 The Randomized Selection Algorithm

By introducing randomization, the number of messages needed to achieve a consensus or finish a distributed prefix computation can be reduced to be only  $O(p)$  with high probability. This randomized consensus protocol (distributed prefix computation), will be used as a building block of our randomized selection algorithm. Proof of the following Lemma can be found in [14].

**Lemma 4.1** *A prefix computation can be realized decentralizedly in a  $p$ -processor de Bruijn network or hypercube using  $O(p)$  messages with  $O(\tau \log p)$  communication delay with probability  $\geq (1 - p^{-\alpha})$ , for some constant  $\alpha$ .*

Sorting will be still used as a building block of our randomized selection. Our randomized selection algorithm uses a distributed sparse enumeration sort.

**Lemma 4.2** *For any fixed  $\epsilon \leq \frac{1}{2}$ , a set of  $p^\epsilon$  keys distributed in a  $p$ -node hypercube  $H_n$  or a  $p$ -node de Bruijn network ( $n = \log p$ ) with no more than one keys per node can be sorted in  $O(p)$  messages with  $O(\tau \log p)$  communication delay.*

The selection algorithm is designed based on random sampling. The basic idea of the 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 all the input keys in between  $a$  and  $b$ , then  $|S'|$  should not be very large so that we can directly process  $S'$ .

We prove the following theorem [14]:

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

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

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

## 5 Sorting

Sorting is important in the applications of very large database systems because a sorted distributed file provides much faster data retrieval or other data manipulation operations. We have shown the following results [14]:

**Theorem 5.1** *Sorting of a distributed file of size  $n$  can be distributedly performed on a  $p$ -processor hypercube in  $O(n \log^2 p)$  messages and with communication delay  $O(\tau \frac{n}{p} \log^2 p)$  provided  $n$  is polynomial in  $p$  and  $n = \Omega(p \log^2 p)$ .*

**Theorem 5.2** *Sorting of a distributed file of size  $n$  can be distributedly performed on a  $p$ -processor de Bruijn network in  $O(n \log p)$  messages and with communication delay  $O(\tau n)$  provided  $n$  is polynomial in  $p$  and  $n = \Omega(p^2 \log^2 p)$ , which is nearly optimal.*

**Theorem 5.3** *Sorting of a distributed file of size  $n$  can be distributedly performed on a  $p$ -processor de Bruijn network in  $O(n \log p)$  messages and with communication delay  $O(\tau n)$  provided  $n$  is polynomial in  $p$  and  $n = \Omega(\frac{p^2}{\log p})$  with high probability, which is nearly optimal.*

**Theorem 5.4** *Sorting of a distributed file of size  $n$  can be distributedly performed on a  $p$ -processor hypercube in  $O(n \log p)$  messages and with communication delay  $O(\tau n)$  provided  $n$  is polynomial in  $p$  and  $n = \Omega(p^2 \log^2 p)$ , which is nearly optimal.*

**Theorem 5.5** *Sorting of a distributed file of size  $n$  can be distributedly performed on a  $p$ -processor hypercube in  $O(n \log p)$  messages and with communication delay  $O(\tau n)$  provided  $n$  is polynomial in  $p$  and  $n = \Omega(\frac{p^2}{\log p})$  with high probability, which is nearly optimal.*

## Acknowledgements

This research was supported in part by NSF Grants CCR-9596065 and CCR-9503007 and Grants G-50 and R-3-2 from the University of Aizu.

## References

- [1] J.-C. Bermond and J.-C. König, "General and Efficient Decentralized Consensus Protocols II," *Parallel and Distributed Algorithms*, editors: M. Cosnard et al., North-Holland, 1989, pp. 199-210.
- [2] M. Blum, R. Floyd, V.R. Pratt, R. Rivest, and R. Tarjan, "Time Bounds for Selection," *Journal of Computer and System Science*, vol. 7, no. 4, 1972, pp. 448-461.
- [3] 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.
- [4] G.N. Frederickson, "Tradeoffs for Selection in Distributed Networks," in *Proceedings of 2nd ACM Symposium on Principles of Distributed Computing*, 1983, pp. 154-160.
- [5] E. Horowitz, S. Sahni, and S. Rajasekaran, *Computer Algorithms/C++*, W. H. Freeman Press, 1997.
- [6] D.F. Hsu and D.S.L. Wei, "Permutation Routing and Sorting on Directed de Bruijn Networks," *Proc. ICPP*, Oconomowoc, Wisconsin, August, 1995, pp. I96-I100.
- [7] J.I. Munro and M.S. Paterson, "Selection and Sorting with Limited Storage," *Theoretical Computer Science* 12, 1980, pp. 315-323.
- [8] D. Nassimi and S. Sahni, "Parallel Permutation and Sorting Algorithms and a New Generalized Connection Network," *JACM*, July 1982, pp. 642-667.
- [9] 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, pp. 145-157.
- [10] S. Rajasekaran, "Randomized Parallel Selection," *Proc. Tenth conference on Foundations of Software Technology and Theoretical Computer Science*, Dec. 1990, Bangalore, India. Springer-Verlag Lecture Notes in Computer Science 472, pp. 215-224.
- [11] S. Rajasekaran, W. Chen, and S. Yooseph., "Unifying Themes for Parallel Selection," *Proc. Fifth International Symposium on Algorithms and Computation*, August 1994.
- [12] 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.
- [13] 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.
- [14] S. Rajasekaran and D.S.L. Wei, "Designing Efficient Distributed Algorithms Using Sampling Techniques," Technical Report, Language Processing Systems Lab., The University of Aizu, Japan, 1996.
- [15] 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.
- [16] G. Tel, *Introduction to Distributed Algorithms*, Cambridge University Press, 1994.
- [17] L.M. Wegner, "Sorting a Distributed File in a Network," in *Proc. Princeton Conf. Inform. Sci. Syst.*, 1982, pp.505-509.