The Round Complexity of Distributed Sorting

[Extended Abstract]

Boaz Patt-Shamir[*]
School of Electrical Engineering
Tel Aviv University
boaz@eng.tau.ac.il

Marat Teplitsky
School of Electrical Engineering
Tel Aviv University
marattep@tau.ac.il

ABSTRACT

We consider the model of fully connected networks, where in each round each node can send an $O(\log n)$-bit message to each other node (this is the CONGEST model with diameter 1). It is known that in this model, min-weight spanning trees can be found in $O(\log\log n)$ rounds. In this paper we show that distributed sorting, where each node has at most n items, can be done in time $O(\log\log n)$ as well. It is also shown that selection can be done in $O(1)$ time. (Using a concurrent result by Lenzen and Wattenhofer, the complexity of sorting is further reduced to constant.) Our algorithms are randomized, and the stated complexity bounds hold with high probability.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

General Terms

Theory, Algorithms

Keywords

network algorithms, communication complexity, distributed sorting, CONGEST model

1. INTRODUCTION

The round complexity of network algorithms can measure either the *locality* of information required for decision making (when the critical restriction is that messages progress at the maximal rate of one hop per round), or it can also measure the *quantity of information* required to solve the given problem—when the throughput of communication link is also restricted. The former model is called the LOCAL model, and the latter is called the CONGEST model [18]. In the LOCAL model, all problems (functions) can be solved in D rounds, where D is the diameter of the network. This is because in D rounds, all input values can be made available in all processors, allowing for local computation of all outputs. In the CONGEST model, on the other hand, it is known that there are problems (such as minimum-weight spanning tree [8, 15, 17] and stable marriage [10]) with classes of instances whose graphs have small diameter but whose worst-case round complexity is high. For example, finding the minimum-weight spanning tree (abbreviated MST) of graphs of diameter 3 graphs takes roughly $\Omega(n^{1/3})$ communication rounds in the worst case, where n is the number of nodes in the network.[1]

One special case of the CONGEST model is particularly intriguing: The fully connected graph (a.k.a. clique), where the diameter is 1. The point is that any piece of information is within easy reach (at most one hop away), so locality is not an issue. However, getting *all* input into a single node (which allows any question to be answered at that node) takes $\Omega(N/n)$ rounds, where N denotes the number input items. It therefore seems that the fully-connected CONGEST model allows one to investigate the *complexity of congestion*, in a model almost completely divorced from locality. Unfortunately, the exact power of the fully-connected CONGEST model is unknown, except the fact that it admits an ultra-fast algorithms in a few cases. In fact, the only pertinent result in this model is by Lotker et al. [14], where it was shown that MST can be computed in this case in $O(\log\log n)$ rounds.

In this paper we do not resolve the mystery of the communication complexity of the fully-connected CONGEST model. Rather, we continue the exploration of this curious model, and show that the basic task of *sorting* can also be accomplished by an ultra-fast algorithm, whose time complexity is $O(\log\log n)$ rounds (with high probability). To achieve this time complexity, we need to carefully balance the work

[*]Supported in part by the Israel Science Foundation (grant 1372/09) and by Israel Ministry of Science and Technology. Research done in part while visiting MIT CSAIL.

[1]More precisely, the lower bound is $\Omega((n/B)^{1/3})$ rounds, where B is the maximal number of bits in a message. Following the convention, we assume that $B = O(\log n)$ bits.

load over the nodes (and communication load over links). We hope that the techniques developed in this paper will be helpful in the next steps of understanding this model.

Related Work. Sorting is a classical computational problem, possibly one of the best studied ones, with many algorithms and analyzes. Knuth [11] covers classical algorithms in depth. Below we give a very brief summary of a few highlights in parallel sorting.

Multi-processor sorting was also the target of much research, starting with Batcher's odd-even sorting network [3]. There is an $O(\log n)$ time algorithms for sorting in an $O(n)$-processor PRAM due to Cole [5], and the celebrated AKS $O(\log n)$-depth sorting network [2] (extended in a non-uniform way by Chvátal [4]). Valiant proved that finding the maximum of n items using n parallel comparisons in a round requires $\Theta(\log\log n)$ rounds [19].

PRAM abstract data access completely (every read and every write may use the shared memory in a single computational cycle). This critical drawback is addressed by later models, such as the BSP model [20] and the LogP model [6]. In the BSP model, each processor can send and receive at most h messages in each communication round (this model abstracts away a major difficulty of our model, in that it allows a processor to send h distinct messages to another processor in a single round). It is known [9] that sorting can be done in the BSP model in $O(\frac{\log N}{\log(h+1)})$ communication rounds for the case of N keys, n processors and $h = \Theta\left(\frac{N}{n}\right)$. Recently, Lenzen and Wattenhofer [12] considered a similar model, where in each communication round, each node can send and receive $O(n^\epsilon)$ bits, for a constant $\epsilon > 0$. They show that any algorithm that runs in T rounds, using less than n^ϵ bits of memory, over networks of polylogarithmic maximal degree in the LOCAL model, can be emulated in the globally restricted model in time $O(\log T + T(\log\log n/\log n))$, which means that any problem with a log-time algorithm in the polylog-bounded-degree CONGEST model has an $O(\log\log n)$ time algorithm in the globally-restricted model. (We note that our a part of algorithm can be used to emulate this model: see Section 5).

Our Results. In this paper we give a randomized algorithm that sorts n^2 items in $O(\log\log n)$ rounds over a fully-connected n-node graph, where each link can carry $O(\log n)$ bits in each round. We also show how to solve the selection (and median finding) problem in this model, in constant number of rounds.

Paper Organization. The remainder of this paper is organized as follows. In Section 2 we formalize the computational model and the problem statement. In Section 3 we describe our algorithm, and we analyze it in Section 4. Some extensions are outlined in Section 5, and we conclude in Section 7.

Remark: Later developments. After the initial submission of this work, we have learned that concurrently and independently, Lenzen and Wattenhofer [13] have showed that in the same model, N messages can be routed to their destinations in time $O(\frac{s+r}{n})$ w.h.p., where s is the maximal number of messages originating at any node, and r is the maximal number of messages destined to any node. A direct consequence of the result of [13] is that sorting n^2 items can be performed in $O(1)$ rounds in this model, as we explain in Section 6.

2. MODEL AND PROBLEM STATEMENT

In this section we define the main problem we study. In a nutshell, we use the fully-connected network topology under the CONGEST model [18], and we assume that in each processor there are n input values, called keys, that need to be globally sorted. A full specification is given below.

We are given a set of n *processors*, where each processor is a randomized state machine. We refer to processors also as *nodes* and denote them as $V = \{v_1, \ldots, v_n\}$. Each processor has some *input* registers and corresponding *output* registers. The value of the input registers may be an integer or a special symbol signifying "no input." The values of the input registers are initially set by the environment (users). Upon termination of the computation, the output registers in each processor indicate the *ranks* of the input values, i.e., the output register corresponding to the largest value should be "1," the output register corresponding to the second largest value should be "2" and so on.

To compute this input, the processors execute a *protocol* that proceeds in synchronous *rounds*: in each round each processor may send $O(\log n)$ bits to each other processor.[2] Formally, each round consists of three steps:
(1) Receive messages from the previous round. There may be at most $n-1$ such messages: one from every processor.
(2) Perform local computation. This step may read from a tape (sequence) of random bits.
(3) Send messages to other processors. Each other processor may be sent an $O(\log n)$ bits message.

Execution starts when input is injected to the processors: We assume that n or less input values called *keys* are placed in the input registers of each processor (this may be viewed as receiving messages from round 0). We assume that all input values are encoded using $O(\log n)$ bits. The execution terminates when all processors enter a special *halting* state. The *round complexity* of the protocol is the number of rounds until the protocol terminates. We say that the protocol solves the problem *with high probability* in time T if with probability $1 - n^{-\Omega(1)}$, for any input values, after T rounds the protocol has terminated with the correct values in its output registers (probability is taken over the space of random bit tapes).

We assume that processors are numbered 1 through n. This assumption does not restrict generality when speaking of computations that succeed with high probability, because

[2]The choice of $\log n$ as the number of bits in a message is motivated similarly to the choice of $\log n$ to be the number of bits in a computer word: the idea is that values polynomial in the input length can be stored in a single unit. In the context of network algorithms, the essential assumption is that a message may contain a constant number of node and link IDs, counters, and input items.

in our model each processor can choose an ID of $O(\log n)$ bits and broadcast it to all its neighbors in the first round, and then take the rank of its ID to be their number. This little pre-processing procedure adds just one round to the time complexity, and a term of $n^{-\Omega(1)}$ to the probability of failure.

3. THE ALGORITHM

In this section we describe the algorithm for the sorting problem described in Section 2. Analysis is provided in Section 4.

Overview. The basic idea underlying the algorithm is to partition the input keys into n disjoint ranges, gather all keys of each range in a distinct node, and sort them locally. It turns out that the bottleneck in this approach is the "gathering" step: it may be the case that a node needs to send multiple keys to another node, something that cannot be implemented by the primitive send operation in a single round. We therefore develop a simple "scatter-gather" procedure that allows any single sender to send n items to any set of receivers, using intermediate nodes. However, since there are n parallel senders, the link between an intermediate node and a destination node might also get congested. To overcome this difficulty we employ a simple but subtle control mechanism, that first takes care of the less loaded destinations, and then of the heavily loaded ones.

Pseudo-code of the high-level algorithm is given in Algorithm 1. We now elaborate on the way it is implemented.

Algorithm 1 SORT

```
(1) Choose $n-1$ keys $k_{r_2},\dots,k_{r_n}$ uniformly at random.
    // delimiter keys
(2) Define, for $1 \le i \le n$, range $i$ $R_i \overset{\text{def}}{=} [k_{r_i}, k_{r_{i+1}}-1]$,
    where $k_{r_1} \overset{\text{def}}{=} -\infty$ and $k_{r_{n+1}} \overset{\text{def}}{=} \infty$.
    Define, for each key $k$, $d(k)=i$ if $k_{r_i} \in R_i$.
    // key $k$ needs to get to node $d(k)$
(3) Let $c(i) = |\{k : d(k)=i\}|$. A node $i$ is called active
    destination only if $c(i) < 2n \ln\ln n$. A key $k$ is active
    only if $d(k)$ is an active destination.
(4) repeat
    (4a) Each node picks a random intermediate destination
         $m(k)$ for each active key $k$ it has.
    (4b) At each node $i$: For $j=1,\dots,n$, let $P_i(j) =$
         $\{k \mid k$ is active and $m(k)=j\}$; If $P_i(j) \ne \emptyset$, pick
         a random $k \in P_i(j)$ and send $k$ to $j$.    // source
         to intermediate
    (4c) At each node $j$: For $l=1,\dots,n$, let $Q_j(l) =$
         $\{k \mid m(k)=j$ and $d(k)=l\}$. If $Q_j(l) \ne \emptyset$, pick
         a random $k \in Q_j(l)$ and send it to $l$. Mark $k$
         inactive. All other keys are sent back to their
         sources.              // intermediate to destination
    until all active keys have reached their destination.
(5) Cleanup stage: Repeat Steps 1–4 only with keys that
    did not reach their destinations.
(6) Each node sorts all keys associated with it, and determines
    their global rank.
(7) Ranks are routed back by reversing the routes taken
    by the keys.
```

Assigning keys to nodes. The idea is as follows. Let a_i be the number of keys at node v_i, and let $N \overset{\text{def}}{=} \sum_{i=1}^{n} a_i$ be the total number of keys. We first order all keys in an arbitrary order (not necessarily sorted) as follows. Locally, each node v_i orders its a_i keys arbitrarily $k_{i_1},\dots,k_{i_{a_i}}$. This induces a global order, obtained by concatenating the local node orders in the natural way: the key k_{i_j} indexed locally at node i is the key whose global index is $\sum_{l=1}^{i-1} a_l + k_{i_j}$. From this global indexation, $n-1$ keys $k_{r_2},\dots,k_{r_n}$ are chosen independently and uniformly at random from $\{1,\dots,N\}$ (Step 1). These keys serve as "range delimiters," in the sense that all keys in the range $[k_{r_i}, k_{r_{i+1}}-1]$ are associated with node v_i for $1 \le i \le n$, where $k_{r_1} = -\infty$ and $k_{r_{n+1}} = \infty$ by convention (Step 2).

The implementation is as follows. Each node v_i broadcasts to all other nodes the number a_i of keys it has. This allows each node to compute the global index of each of its keys as explained above. Node v_1, which will be the designated "leader" for the remainder of the algorithm, chooses $n-1$ indices $r_2,,\dots,r_n$ uniformly at random from $\{1,\dots,N\}$, and for each $2 \le i \le n$, v_1 sends r_i to node v_i. Node v_i then broadcasts r_i to all other nodes. This way, after 3 rounds, all nodes know all r_i values. Using the known global ranking, each node that holds k_{r_i} sends it to node i (a node may send more than a single value, but to different destinations). Finally, each node i broadcasts to all the value of k_{r_i}, and, as a result, all nodes know, for each key k, what is $d(k)$, i.e., which node is the destination of each key.

Next, we route each key k to its destination $d(k)$ in two stages as follows. In the first stage, only keys whose destination is the destination of at most $2n \ln\ln n$ keys are routed (Step 3). To implement this distinction, each node v_i sends to each other node v_j the number of keys v_i whose destination is v_j. Each node v_j thus finds the total number of keys destined to it, and v_j broadcast this number to all. (We note that the size of all ranges is recorded by all nodes, so that local ranking of the keys in a range can be translated to the global ranking later in Step 6.)

Routing the keys. In Step 4, we solve the following problem. We have n nodes, each with n or less keys, where each key has a destination in $\{v_1,\dots,v_n\}$ so that no node is the destination of more than $2n \log\log n$ keys. The difficulty stems from the fact that each link can carry at most one key in each round, while the number of keys that initially reside at the same node and share the same destination node may be $\Omega(n)$. We solve this problem using the Valiant an Brebner paradigm of random intermediate destination [21] as follows. We run a sequence of phases (phases are iterations of the loop in Step 4). In each phase, each node first selects a random intermediate destination for each of its remaining undelivered keys. Then the node selects a random single key for each possible intermediate destination, i.e., congestion conflicts in source-to-intermediate links are resolved at the source nodes. After resolving the conflicts, the keys are sent to their intermediate destination in a single round. Next, each node (now playing as an intermediate destination) picks one key for each final destination and sends it over, i.e., congestion conflicts on the intermediate-to-destination links are resolved at the intermediate node. In the following round, keys that were not sent to their final destination (due to congestion in the intermediate-to-destination link) are sent back to their origin, where they are ready for the next phase. We shall show that with high probability, all keys arrive at their destination within $O(\log\log n)$ phases.

We are then left with the keys associated with large ranges (and were therefore not routed in the first routing stage). This case is in fact easier to handle than the original instance because, as we show, only about a $\frac{1}{\log n}$ fraction of the keys remain, and applying Steps 1–4 again will deliver all of them to their destinations in $O(\log\log n)$ additional rounds, because now, w.h.p., all ranges have size only $O(n)$.

Finally, when we know the rank of each key in its range, the global ranks can be computed as the size (and ordering) of all ranges is known from Step 3. To comply with the problem requirement, these computed ranks are routed back to the source nodes, using the reverse of routing schedule that brought the keys in.

4. ANALYSIS

In this section we analyze the algorithm specified in Section 3. We start by analyzing the partition into ranges, and then analyze the routing procedure.

4.1 Partition Into Ranges

Partition into ranges (Steps 1–2) is done twice during the execution of the algorithm. We analyze them in order.

4.1.1 The First Stage

We first analyze the partition of the key set into ranges in the first stage. We start with a technical lemma, that analyzes the following scenario. Order the N keys by value, and partition them into $\lceil n/\ln\ln n\rceil$ *segments* of about the same size $(N/n)\ln\ln n$ (rounded up and down as necessary; we'll ignore rounding for simplicity of exposition). Now, call a segment *selected* if one of the keys it contains was selected in Step 1 (of the first stage) as a delimiter. Then the following holds true.[3]

LEMMA 4.1. *With high probability, the number of non-selected segments is at most* $\frac{2n}{\ln n \ln\ln n}$.

Proof: Let Y_i to be binary indicator variable, taking the value 1 if segment i is *not* selected and 0 otherwise. Then

$$\Pr[Y_i = 1] \;=\; \left(1 - \frac{\ln\ln n}{n}\right)^n \le e^{-\ln\ln n} = \frac{1}{\ln n}\ ,$$

and therefore, $E[\sum Y_i] \le \frac{n}{\ln n \ln\ln n}$. Now, it is easy to see that the vector of Y_i random variables is *negatively associated* ([7], and see also [16]). Therefore we may apply the Chernoff-Hoeffding bound and conclude that

$$\Pr\left[\sum Y_i > \frac{2n}{\ln n \ln\ln n}\right] \;<\; e^{\frac{-n}{3\ln n \ln\ln n}} \;<\; n^{-\Omega(1)}$$

for all $n > 1$. ▮

[3]We note that the constants in Lemma 4.2 can be improved; we make no attempt to optimize them here.

We can now deduce the following.

LEMMA 4.2. *Suppose that the number of keys is at most n^2. Then, with high probability, the number of ranges with more than $2n\ln\ln n$ keys is at most $2n/(\ln n\cdot\ln\ln n)$.*

Proof: First, note that since $N \le n^2$ by assumption, segment size is at most $n\ln\ln n$ keys. Therefore, a range with $2n\ln\ln n$ keys or more must contain a complete segment that is not *selected*. It follows that the number of non-selected segments is an upper bound on the number of ranges of size at least $2n\ln\ln n$, and by Lemma 4.1 this number is, with high probability, bounded by $\frac{2n}{\ln n \ln\ln n}$. ▮

4.1.2 Cleanup Stage

We need to show that one more iteration of steps 1–3 is enough to sort all remaining keys. To this end, we apply Lemma 4.1 once again as follows.

LEMMA 4.3. *W.h.p, the number of keys remaining to the cleanup at most* $\frac{4n^2}{\ln n}$.

Proof: Since each range that is deferred to the cleanup stage contains at least $2n\ln\ln n$ keys, each such range must contain a run of non-selected segments. It follows that the total number of keys in ranges that are deferred to the cleanup stage is at most $2n\ln\ln n$ times the number of non-selected ranges. Applying Lemma 4.1 once again, we conclude that w.h.p., the number of keys in the cleanup stage is at most

$$2n\ln\ln n\cdot\frac{2n}{\ln n\cdot\ln\ln n} = \frac{4n^2}{\ln n}\ . \qquad ▮$$

Lemma 4.3 implies the following.

LEMMA 4.4. *In the cleanup stage, with high probability, all ranges are of size $O(n)$.*

Proof: By Lemma 4.3, the number of remaining keys is at most $4n^2/\ln n$. Similarly to the analysis of the range selection in the first stage, we partition the keys in segments of length $12n$ (say). The probability that a particular segment is unselected is at most $(1 - 3\ln n/n)^n \le n^{-3}$, and hence the probability that all segments are selected (implying that the size of all ranges is bounded by $24n$) is at least $1 - \frac{2\ln n}{n^2}$. ▮

4.2 Analysis of Routing

Routing keys to their destination nodes is done in both stages by Step 4. It is immediate from the code that each iteration takes $O(1)$ rounds; it remains to analyze the number of iterations taken in Step 4.

We distinguish between two cases: "heavily loaded" destinations and "lightly loaded" ones. For the first case, we have the following lemma.

LEMMA 4.5. *Suppose that there are $m \geq n$ active keys with destination v_i at the beginning of an iteration of Step 4. Then with high probability, at least $n/9$ keys will be delivered at v_i in that iteration.*

Proof: It may be helpful to note that we have a classical balls-and-bins type of situation here, where keys are balls and intermediate destinations are bins. So, keeping this intuition in mind, consider first the choice of intermediate destinations at the source nodes.

The probability that an intermediate destination is unique at a source node (i.e., was chosen for exactly one key at that source) is at least $1/e$, because there are at most n keys at the source, and there are exactly n intermediate random choices for each. It therefore follows from standard balls-and bins results (see [7]) that with high probability, at least $n/3$ (say) nodes will receive, as intermediate destinations, keys whose final destination is v_i.

Next, consider the situation at an intermediate destination v^*: if there is any key at v^* whose final destination is v_i, then v_i will receive *some* key from v^* in the next round. Applying [7] once again, we may conclude that with high probability, if there are $m \geq n$ keys destined to v_i, then at least $n/9$ of them (say) will be delivered at v_i. ▮

To analyze the lightly-loaded destination case, we use a fundamental fact proven in [1]. Consider the following iterative balls-and-bins process. There are n balls at start, and in each round, all remaining balls are thrown (independently, randomly) into n bins. In each round, each bin accepts only one of the balls thrown into (if any), and all other balls are thrown again in the next round, until all balls are placed in bins.

LEMMA 4.6. *Suppose that there are at most n active keys with destination v_i. Then with high probability, all keys are delivered in $O(\log\log n)$ iterations.*

Proof: Follows from the fact that Step 4 is implemented like the algorithm THRESHOLD(1) in [1], wherein it is shown that in $O(\log\log n)$ rounds suffice with probability $1 - n^{-\Omega(1)}$. ▮

4.3 Summary

THEOREM 4.7. *With high probability, Algorithm* SORT *solves the sorting problem and terminates in $O(\log\log n)$ communication rounds.*

Proof: Correctness is immediate from the Union Bound: all statements hold with high probability, and there is only a polynomial number of times a failure may occur. Consider now the round complexity. Steps 1–3 take $O(1)$ rounds. Regarding Step 4, in the first stage, all destinations have at most $O(n \log\log n)$ keys routed to them. By Lemma 4.5, w.h.p., after $O(\log\log n)$ rounds, for each destination there are at most n keys that are still undelivered, and therefore, by Lemma 4.6, after $O(\log\log n)$ additional rounds, the first stage is over. In the cleanup stage, there are at most $O(n)$ keys to be routed to each destination, and therefore, by Lemmas 4.5 and 4.6, in $O(\log\log n)$ additional rounds, the cleanup stage is over as well. Step 6 is communication-free, and Step 7 just doubles the overall complexity. ▮

5. EXTENSIONS AND APPLICATIONS

In this section we outline two simple extensions of the algorithm. One is for a small number of input keys, and the other is for the median problem. But let us first point out an application to models with global restriction on the communication that can be delivered in a round.

Application: global restriction on communication. We note that the routing step of our algorithm can be used to emulate the model used by Lenzen and Wattenhofer in [12]. Specifically, the model considered in [12] allows for the global exchange of $O(n^\epsilon)$ bits in each round (for some constant parameter $0 < \epsilon \leq 1$), without any other restriction on communication. Let us call this model $G(\epsilon)$. In our fully-connected CONGEST model, we only restrict the number of bits exchanged by any pair of nodes in a round to be $O(\log n)$ (but the total number of bits moving in a round can be as high as $O(n^2 \log n)$). Clearly, algorithms for the $G(\epsilon)$ model cannot be run directly on the fully connected CONGEST model, because in the $G(\epsilon)$ model, it may be the case that n^ϵ bits may need to move from one node to another in a single round. However, our analysis implies that any algorithm running in time T on the $G(\epsilon)$ model can be emulated (w.h.p.) on our model in time $O(T)$ for any $\epsilon \leq 1$. This can be done in a round-by-round emulation as follows. First, we break the model $G(\epsilon)$ messages to packets of size $O(\log n)$ bits. Then we run Step 4 of the algorithm, using the actual destinations of the packets instead of ranges. Since by assumption all node are the destination of at most $O(n^\epsilon)$ bits, Lemma 5.2 ensures (w.h.p.) that all $O(n/\log n)$ packets will be delivered in $O(1)$ time.

A small number of input items. The algorithm specified in Section 3 works when the number of input keys is at most the square of the number of processors, i.e., $N \leq n^2$. It should be noted that if the number of keys is smaller than $n^2/\log^2 n$, then the time complexity drops to $O(1)$: this follows from the following facts.

LEMMA 5.1. *Suppose we have $K \leq (n/\log n)^2$ ordered elements, and we choose uniformly at random n elements. Then with high probability, the longest gap between two consecutive chosen elements is $O(n/\log n)$.*

Proof: As in the proof of Lemma 4.1, we partition the elements into n segments of length $n/\log^2 n$ each (up to rounding). The probability that a segment remains unselected (i.e., no delimiter is contained in it) after Step 1 is $(1 - \frac{1}{n})^n < e^{-1}$, and thus the probability that $\log n$ segments or more are unselected is $n^{-\Omega(1)}$. The claim follows from the fact that a gap of $n/\log n$ is possible only if there are $\log n - 1$ unselected segments. ▮

LEMMA 5.2. *If $n/\log n$ balls are thrown into n bins, the maximal number of balls in a bin is $O(1)$ with high probability.*

Proof: Follows from the Chernoff Bound. ▮

COROLLARY 5.3. *If the number of keys is $N = O(n^2/\log^2 n)$ then sorting on the fully connected n-clique can be done in $O(1)$ rounds.*

Proof: Algorithm 1 works. Lemma 5.1 implies that w.h.p., all nodes are active, and Lemma 5.2 implies that the routing can be done in $O(1)$ rounds. ▮

Finding the median (and general selection). In the selection problem, we are asked to output the kth largest key, where k is part of the input. The median problem is a special case of selection, where $k = N/2$ (N is the number of input keys). It turns out that a minor modification in Algorithm 1 yields a simple $O(1)$ selection algorithm. First, we run Steps 1–3 to figure out what is exactly the range the contains the kth largest key. This is easy because after Step 3 each node knows what is the number of keys in each range. We then apply the sorting algorithm with input that consists of the keys of this range *only* (this is also easy because after Step 2 each key knows what is its range). When the algorithm terminates, we can identify exactly what is the kth largest requested key. To analyze the time complexity, we need the following straightforward fact.

LEMMA 5.4. *With high probability, no range defined by the delimiters chosen in Step 1 contains more than $O(n \log n)$ keys.*

Now, the following holds with high probability. First, Steps 1–3 take $O(1)$ rounds. Also, by Lemma 5.4, the invocation of Algorithm SORT has only $O(n \log n)$ keys as input, and therefore Corollary 5.3 ensures that SORT will terminate in $O(1)$ rounds as well. We therefore have the following.

COROLLARY 5.5. *The Selection problem can be solved in the fully-connected* CONGEST *model in $O(1)$ rounds, with high probability.*

6. POSTSCRIPT: SORTING IN $O(1)$ ROUNDS

Concurrently to our work, Lenzen and Wattenhofer proved the following result.

THEOREM 6.1. **[13]** *Suppose there are $O(n)$ messages in each node, and that the number of messages destined to each node is $O(n)$. Then routing all messages to their destinations can be done in $O(1)$ rounds with probability $1 - n^{-\Omega(1)}$.*

Using this result, we propose the following algorithm for sorting.

(1) Select each key with probability $\frac{1}{\log^2 n}$. Let S denote the set of selected keys.

(2) Apply Algorithm 1 to S. For an integer $1 \le i \le |S|$, let $k_S(i)$ denote the key ranked i in S.

(3) Define delimiters $l(0), \ldots, l(n)$ by
$$l(i) = \begin{cases} -\infty & \text{if } i = 0 \\ k_s(\lceil |S| \frac{i}{n} \rceil) & \text{if } 0 < i \le n \end{cases},$$
and set the destination of each key k to be $d(k) \overset{\text{def}}{=} i$ if $l(i-1) < k \le l(i)$.

(4) Send each key k to destination $d(k)$ using the routing implied by Theorem 6.1.

(5) Locally sort all received keys and send back their ranks.

To see why this procedure works, we first note that by the Chernoff Bound, the set S selected at Step 1 has size $O(n/\log^2 n)$ w.h.p., and therefore, by Corollary 5.3, Step 2 ends in $O(1)$ rounds. Finding the delimiters takes $O(1)$ time. The key to the success of the algorithm is that with high probability, for all $1 \le i \le n$, the number of keys in the range $[l_{i-1}+1, l_i]$ (which are all destined to node i) is bounded from above by $O(n)$. This fact can be shown to follow from the Chernoff bound as well. We can therefore conclude with the following theorem.

THEOREM 6.2. *The sorting problem can be solved in $O(1)$ communication rounds with high probability.*

7. CONCLUSION

In this paper we have given more evidence that the fully-connected CONGEST model allows for ultra-fast algorithms for basic problems. Specifically, it turns out that sorting can be solved in constant time with high probability. Many very interesting problems remain open for this problem and this model. Regarding sorting, it may be the case that good deterministic algorithms exist. Regarding the model, we see two main directions that complement each other: One is to come up with a non-trivial lower bound for some natural problem in this model, and the other is to come up with ultra-fast algorithms for key problems in this model (e.g., weighted matching).

Acknowledgment

We thank the anonymous referees, and Christoph Lenzen and Roger Wattenhofer for providing us with a preprint and an explanation of [13].

8. REFERENCES

[1] Micah Adler, Soumen Chakrabarti, Michael Mitzenmacher, and Lars Rasmussen. Parallel randomized load balancing. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, STOC '95, pages 238–247, New York, NY, USA, 1995. ACM.

[2] M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. In *Proc. 15th Ann. ACM Symp. on Theory of Computing (STOC)*, pages 1–9. ACM Press, May 1983.

[3] K. E. Batcher. Sorting networks and their applications. In *Proc. 1968 Spring Joint Computer Conference*, pages 307–314, Reston, VA, 1968. AFIPS Press.

[4] V. Chvátal. Lecture notes on the new AKS sorting network. Technical Report DCS-TR-294, Computer Science Department, Rutgers University, 1992.

[5] Richard Cole. Parallel merge sort. In *Proc. 27th Ann. Symp. on Foundations of Computer Science*, pages 511–516, October 1986.

[6] David E. Culler, Richard M. Karp, David Patterson, Abhijit Sahay, Eunice E. Santos, Klaus Erik Schauser, Ramesh Subramonian, and Thorsten von Eicken. LogP: A practical model of parallel computation. *Comm. ACM*, 39:78–85, November 1996.

[7] Devdatt Dubhashi and Desh Ranjan. Balls and bins: a study in negative dependence. *Random Struct. Algorithms*, 13:99–124, September 1998.

[8] Micheal Elkin. An unconditional lower bound on the time-approximation tradeoff for the minimum spanning tree problem. *SIAM Journal on Computing*, 36(2):463–501, 2006.

[9] Michael T. Goodrich. Communication-efficient parallel sorting (preliminary version). In *Proc. 28th Ann. ACM Symp. on Theory of Computing (STOC)*, pages 247–256, New York, NY, USA, 1996. ACM.

[10] Alex Kipnis and Boaz Patt-Shamir. A note on distributed stable matching. In *Proc. 29th International Conf. on Distributed Computing Systems (ICDCS)*, June 2009.

[11] Donald E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, 2nd edition, 1998.

[12] Christoph Lenzen and Roger Wattenhofer. Brief announcement: Exponential speed-up of local algorithms using non-local communication. In *Proc. 29th ACM Symp. On Principles of Distributed Computing (PODC)*, pages 295–296, 2010.

[13] Christoph Lenzen and Roger Wattenhofer. Tight Bounds for Parallel Randomized Load Balancing. In *Proc. 43rd ACM Symposium on Theory of Computing (STOC)*, 2011. To appear.

[14] Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov, and David Peleg. Minimum-weight spanning tree construction in O(log log n) communication rounds. *SIAM Journal on Computing*, 35(1):120–131, 2005.

[15] Zvi Lotker, Boaz Patt-Shamir, and David Peleg. Distributed MST for constant diameter graphs. *Distributed Computation*, 18(6):453–460, 2006.

[16] Colin McDiarmid. On the method of bounded differences. In *Surveys in Combinatorics*, pages 148–188. Cambridge University Press, Cambridge, UK, 1989.

[17] D. Peleg and V. Rubinovich. Near-tight lower bound on the time complexity of distributed MST construction. *SIAM Journal on Computing*, 30:1427–1442, 2000.

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

[19] Leslie G. Valiant. Parallelism in comparison problems. *SIAM Journal on Computing*, 4(3):348–355, 1975.

[20] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33:103–111, August 1990.

[21] Leslie G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing (STOC)*, pages 263–277, Milwaukee, WI, May 1981.