1988

Distributed Algorithms for Selection in Sets

Greg N. Frederickson
Purdue University, gnf@cs.purdue.edu

Report Number:
84-471

DISTRIBUTED ALGORITHMS FOR SELECTION IN SETS*

(CSD-TR-471 revised January 1988)

Greg N. Frederickson

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

* This research was supported in part by the National Science Foundation under Grants MCS-8201083 and CCR-86202271 and by the Army Research Office under Contract DAAG03-86-K-0106.

Abstract. Algorithms are presented for selecting an element of given rank from a set of elements distributed among the nodes of a network. Network topologies considered are a ring, a mesh, and a complete binary tree. For the ring and the mesh, algorithms are presented whose performance exhibits a tradeoff between the number of messages transmitted and the total delay due to message transmission. For the mesh and the tree, algorithms are presented that use an asymptotically optimal number of messages. The algorithms are based on a sampling approach that also gives rise to a new linear-time selection algorithm for a single processor.

Key words and phrases. binary tree, distributed computation, mesh, message complexity, networks, ring, selection problem, tradeoffs.

1. Introduction

Two resource measures appear to be relevant to a computation on a distributed network: 1) the number of messages transmitted, and 2) the total delay due to message transmission. A fundamental question is how these two measures interact for any basic computation activity. We explore this question by examining the problem of selecting an element of given rank from a set of elements distributed among the nodes of the network. Selection in a set is a basic problem in computation, which admits a linear-time algorithm on a single processor [BFPRT, SPP].

Selection in a network of two processors is studied in [R], where it is shown that the number of messages, and also the delay, involved in selecting in a set of size n is $\Theta(\log n)$. A somewhat different model of communication is considered in [SS], but by using techniques in [FJ], selection algorithms can be generated for a star network of m processors that use $O(m \log(2n/m))$ messages and $O(\log(n/m))$ delay. In this paper we investigate selection in networks in which the topology plays a more crucial role.

For networks in the topology of a ring or mesh, we present algorithms that realize tradeoffs between our two resource measures. We also present an efficient algorithm for a network in the form of a complete binary tree. In addition, by virtue of the techniques we employ, we give a single processor linear-time algorithm that is completely different from that in [BFPRT].

We summarize below our results for the selection problem in networks, giving both extremes for the algorithms' performance. in the case in which the number of elements n equals the number of processors m. We present a unidirectional algorithm for

the ring, which realizes $O(m(\log m)^3)$ messages with $O(m \log m)$ delay at one extreme, and $O(m^2)$ messages with $O(m)$ delay at the other extreme. This compares with other results as follows. An algorithm for a unidirectional ring appears in [M] and uses $O(m^{1+\varepsilon})$ messages, for $\varepsilon > 0$. A bidirectional algorithm with $O(m(\log m)^2)$ messages and $O(m \log m)$ delay has recently been proposed in [S].

Our algorithms for the mesh realize $O(m)$ messages with $O(m^{1/2}\text{loglog } m)$ delay at one extreme, and $O(m^{5/4}/(\log m)^{1/2})$ messages with $O(m^{1/2})$ delay at the other extreme. Our algorithm for the tree uses $O(m)$ messages with $O((\log m)^3)$ delay. The delay for the tree algorithm has recently been improved by a loglog m factor in [S], at the expense of a log m factor in the number of messages.

For $n > m$, our upper bounds increase by factors of either $(\log n)/(\log m)$ or $\log(2n/m)$. In particular, the algorithms for the mesh and the binary tree use $O(m \log(2n/m))$ messages. Using an adversary argument, we establish that the number of messages used by these two algorithms is asymptotically optimal.

We make the following assumptions in our model. A message will carry a constant number of "words" along one link of the network. For simplicity we shall assume that a message will contain one set element and/or one count, where the count is no larger than cardinality of the set. The transmission time, or delay, along each communication link will be assumed to be equal to some fixed value. Computation time at a processor will be assumed to be small in comparison with message transmission time, and thus will be ignored. Each processor will have a sufficiently large memory so that message buffering will not cause problems. Each processor will have as many ports as there

are communication lines incident on it, and input or output may be carried on simultaneously at these ports. Computation will originate at a distinguished processor, and each processor will have a unique name, so that we are not interested in issues related to electing a leader. (See [DKR] for a list of references.)

A preliminary version of this paper appeared in [F].

2. Samples and filters

One interesting feature of our work is the adaptation and generalization of a technique by Munro and Paterson [MP] that was designed for an entirely different model. They consider the selection problem on a single processor with a limited number of workspace registers and a read-only tape, on which the elements of the set reside. Their algorithm uses a pair of elements, called *filters*, between which the element to be chosen must fall. On each sequential pass through the input, information is gathered that allows for the choice of more refined filters at the end of the pass. Initially, all elements are between the filters. Passes over the input tape are made until the number of elements remaining between the filters is reduced to a number that can be held simultaneously in registers. The desired element may then be selected directly.

Let the elements falling between the current filters be called the *current population.* On each pass, a sample of the current population is constructed, and the new filters are chosen from this sample. For some fixed s, an *s-sample at level i* is a sorted set of s elements chosen from a *subpopulation* of $s2^i$ elements of the current population. An s-sample at level 0 consists of all elements of a subpopulation of size $s2^0$ in sorted

order. An s-sample at level $i+1$ is formed by taking a subpopulation of size $s2^{i+1}$ of the current population, splitting it into two subpopulations of size $s2^i$ consisting of the first and second halves, finding the level i s-samples of these subpopulations, "thinning" each sample by retaining every second element, and then merging the two thinned samples.

Let k be the integer such that the desired element is the kth largest in the current population at the beginning of a pass. After the pass, the new filters are chosen in the following way. Consider the jth largest element in an s-sample at level i. Let L_{ij} and M_{ij} be respectively the least and most number of elements from the corresponding subpopulation that may appear strictly above it in the total order. Let the size of the current population be $n' = s2^r$. (This size can be determined when the current population is scanned to form the samples.) In the sample for the entire current population, the new filters will be the uth and vth elements of the sample, where u is the greatest integer such that $M_{ru} < k$ and v is the least integer such that $L_{rv} > k$. It is shown in [MP] that appropriate choices are $v = \lceil k/2^r \rceil$ and $u = v - r$. With these choices it is shown that $O((n'/s)\log(n'/s))$ elements will be between the new filters. In order for the population to be reduced in size from one iteration to the next, the restriction that $s \geq c \log (n'/s)$ must hold for some constant c.

We use the Munro and Paterson algorithm in the following way. Instead of a pass through a read-only input tape, we have a *sweep* through the network. Instead of reading elements into workspace registers, we send elements via messages from one node to another in the network. Thus some node in the network will be designated the *leader*,

and the sample will be routed toward the leader as it is constructed. Also, the number of elements in the current population can be counted during this sweep. The leader then selects the new filters and broadcasts them to all nodes. The process repeats until a sufficiently limited number of elements remain between the filters. These are then all routed to the leader, and the desired element is selected directly.

While the use of a simple s-sample will yield good results for the ring network, it is not sufficient to achieve the performance bounds we claim for the mesh and the tree. In section 4 we shall introduce an improved sampling technique, making two changes in the way samples of small subpopulations are formed. The sampling technique is a generalization of s-samples, and allows us to also generate a linear-time selection algorithm for a single processor that is quite different from that in [BFPRT]. A referee has pointed out that our technique is related to a method given in [CY], which postdates our paper [F] by two years.

3. Selection in a circular network

We present unidirectional algorithms for the *ring* topology, in which m processors are arranged in a circle. We first handle the case in which there is one element at each processor. As before, let the number of elements currently between the filters be n'. We shall assume that n'*s* is a power of 2. If this is not the case, consider additional "virtual elements" of value $-\infty$ at virtual processors to make this so. The s-samples will be accumulated in a clockwise direction.

Let an element be called *active* if and only if it is in the current population. Sam-

ples at level 0 will be accumulated at the processors P_{k_l} containing the lsth active element, $l = 1, 2, \cdots, n'/s$. When the s values have arrived at the processor, they are sorted and thinned. If l is odd, the thinned sample is transmitted one element after the other to processor $P_{k_{l+1}}$. A sample at level $i > 0$ will be constructed at processor P_{k_l}, where l is a multiple of 2^i. The two samples used will be thinned samples at level $i-1$ from processors P_{k_l} and $P_{k_{l-2^{i-1}}}$. The last sweep will have $n' \leq s$, and thus all elements still active will be routed to the leader.

Lemma 1. Let each processor of an m-node unidirectional ring contain one element of a set. Selection can be performed in the set with $O(ms(\log(m/s))(\log m)/(\log(s/\log m)))$ messages and delay $O(m(\log m)/(\log(s/\log m)))$, where $s \geq c \log m$ for some constant $c \geq 2$.

Proof. Accumulating samples at level i, given samples at level $i-1$, will require no more than $ms/2$ messages. This follows since no more than $s/2$ elements are transmitted along any one edge during the construction of all samples at level i. Thus the total number of messages used to generate a sample of the whole population is $O(ms \log(n'/s))$. The message delay in generating this sample is no more than $m + O(s \log(n'/s))$. This follows since no sample element traverses more than $m-1$ edges, and no element is delayed more than $s/2$ time units at $O(\log(n'/s))$ processors P_{k_l}. By the results in [MP], the number of elements is reduced from n' to $O((n'/s)\log(n'/s))$ on a sweep. Thus $O((\log m)/(\log(s/\log m)))$ sampling sweeps are used. □

With $s = \Theta(\log m)$, there are $O(m(\log m)^3)$ messages and $O(m \log m)$ delay. Taking $s = \Theta(m)$, there are $O(m^2)$ messages and $O(m)$ delay.

We now consider the case in which there are $n > m$ elements distributed among the processors in some fashion. Before a sampling sweep, assume that there are n' active elements, with n_k' at processor P_k, k=1, 2, $\cdots$, m. To keep the amount of message passing low, we handle the elements in groups of $\lceil n'/(ms) \rceil$, all at the same processor. Thus we temporarily ignore $n_k' \bmod \lceil n'/(ms) \rceil$ active elements at processor P_k.

A sample at level 0 will be a set of s elements representing a population of $\max\{s, \lceil n'/(ms) \rceil\}$ elements that are not ignored. If $s \leq \lceil n'/(ms) \rceil$, then samples at level 0 will correspond to groups of $\lceil n'/(ms) \rceil$ elements. If $s > \lceil n'/(ms) \rceil$, then samples at level 0 will be accumulated at the processors P_{k_l} containing the ls th non-ignored active element. Note that for samples at level 0 residing in processors P_{k_l}, the indices $k_1, k_2, \cdots$ may not all be distinct. Samples at level $i > 0$ are defined as previously. The new filters must be chosen slightly differently, since some elements have been ignored. There will be no more than $m\lfloor n'/(ms) \rfloor \leq n'/s$ elements ignored in a sweep. Thus M_{ij} is larger by at most n'/s. Thus the value of u should be smaller by 1.

Theorem 1. Let $n \geq m$ elements be distributed among the processors of an m-node unidirectional ring. Selection can be performed in the set with $O(ms(\log(n/s)\ (\log m)\ /(\log(s/\log m)))$ messages and delay $O(m(\log n)\ /(\log(s/\log m)))$, where s is $O(m)$ and $s \geq c \log m$ for some constant $c \geq 2$.

Proof. As before the number of messages required to generate all samples at level i,

given samples at level $i-1$, will be no more than $ms/2$. The number of levels will be $O(\log(n'/\lceil (n'/(ms)) \rceil)) = O(\log m + \log s) = O(\log m)$. The total number of messages generated in each sweep with $n' > m$ will be $O(ms \log m)$. Since a sweep will reduce n' elements to $O((n'/s)\log(n'/s))$ elements, $O((\log(n/m))/(\log(s/\log m)))$ such sampling sweeps with $n' > m$ are sufficient. Thus reducing n elements to at most m elements will use $O(ms(\log m)(\log(n/m))/(\log(s/\log m)))$ messages and $O(m(\log(n/m))/(\log(s/\log m)))$ delay. Adding these quantities to the the corresponding values in Lemma 1, representing the number of messages and delay for handling m elements, will give the claimed bounds. □

With $s = \Theta(\log m)$, there are $O(m(\log m)^2(\log n))$ messages, and $O(m \log n)$ delay. With $s = \Theta(m)$, there are $O(m^2\log(2n/m))$ messages and $O(m(\log n)/(\log m))$ delay.

4. An improved sampling strategy

We introduce a generalization of the Munro and Paterson sampling technique, which will reduce the number of messages sent on lower levels of sampling. This new sampling technique will thus realize asymptotically fewer messages than if only the s-sample is used. Let d be a factor of s, and s/d a power of 2. Let c be a positive integer. We define a (c, s, d)-*sample at level* i as a sorted set of $\min\{s, d2^{\lceil i/c \rceil}\}$ elements representing a population of $d2^i$ elements. A sample at level i, where i is one more than a multiple of c and less than $c \log(s/d)$, is formed by merging two samples at level $i-1$. A sample at any other level $i > 0$ is formed as in the Munro and Paterson

algorithm by thinning two samples at level $i-1$, and then merging. If $c = 1$ and $d = s$, then we have precisely the Munro and Paterson technique. We give a generalization of Lemma 2 in [MP] below.

Lemma 2. Let L_{ij} and M_{ij} be respectively the least and most elements in a corresponding population that may lie above the jth largest element in an (c, s, d)-sample at level i. For $i \le c \log(s/d)$,

$$L_{ij} = j2^{\lfloor i(1-1/c) \rfloor} - 1 \text{ and}$$
$$M_{ij} = ((c-1)2^{\lceil i/c \rceil} + j - c + i - c\lceil i/c \rceil)2^{\lfloor i(1-1/c) \rfloor}$$

For $i > c \log(s/d)$,

$$L_{ij} = j(d/s)2^i - 1 \text{ and}$$
$$M_{ij} = (i - c \log(s/d) + j + (c-1)s/d - c)(d/s)2^i$$

Proof. The proof is by induction. For $i = 0, L_{ij} = M_{ij} = j-1$. For i one more than a multiple of c and no more than $c \log(s/d)$, we have

$$L_{ij} = \min_{p+q=j}\{L_{i-1,p} + L_{i-1,q} + 1\}$$
$$M_{ij} = \max_{p+q=j}\{M_{i-1,p} + M_{i-1,q+1}\}$$

For other $i > 0$, we have the recurrence equations from [MP]

$$L_{ij} = \min_{p+q=j}\{L_{i-1,2p} + L_{i-1,2q} + 1\}$$
$$M_{ij} = \max_{p+q=j}\{M_{i-1,2p} + M_{i-1,2q+2}\}$$

The first pair of results can be proved inductively from these. Then, using the fact that $2^{i/c} = s/d$ if $i = c \log(s/d)$, the second pair of results can also be proved inductively.

□

It follows from the requirements on u and v that appropriate choices are $v = \lceil k(s/d)/2^r \rceil$ and $u = v - r + c\log(s/d) - (c-1)s/d + (c-1)$. We next give a generalization of Lemma 1 in [MP].

Lemma 3. Let (c, s, d)-samples be used to generate filters. If at most n' elements lie between the filters at the beginning of a pass, then the number of elements between the new filters will be less than

$$2(n'/s)\log(n'/d) + 2(c-1)n'/d \quad .$$

Proof. Values of u and v were chosen such that $u = v - r + c\log(s/d) - (c-1)s/d + (c-1)$. The number of elements between the uth and vth elements of the sample is at most

$$M_{rv} - L_{ru} - 1 = ((r - c\log(s/d) + (c-1)s/d - c) + (v - u))(d/s)2^r$$

$$= (2r - 2c\log(s/d) + 2(c-1)s/d - (2c-1))(d/s)2^r$$

$$< (2\log(n'/d) + 2(c-1)s/d)n'/s \quad . \ \square$$

In addition to being useful in the design of distributed algorithms, our (c, s, d)-sampling strategy is notable for the following reason. With c, s and d chosen appropriately, our sampling strategy leads to a linear-time algorithm for selection that is distinctly different from the linear-time algorithm in [BFPRT]. While the multiplicative factor we achieve on the number of comparisons is not as good as that in [SPP], the approach is conceptually different, and thus interesting.

Theorem 2. Using a $(2, 5\log n, 5)$-sampling procedure, selection can be performed in a set of n elements in $O(n)$ time.

Proof. Our procedure forms a sample of the current population in time proportional to its size. With $s = d \log n$, the number n' of elements will be reduced to $(2c/d)n'$ on one sweep. For the choice of values above, at least 1/5 of the elements will be discarded on each sweep. Thus the whole procedure is linear. □

The above choice of parameters is not optimal, but the values of c and d are small and thus the algorithm is not unnecessarily complicated. Let $S(d)$ be the number of comparisons needed to sort d numbers by binary insertion sort. (See [K] for a partial list of $S(d)$ values.) Then for sufficiently large n the number of comparisons is no larger than $n(S(d)/d + (3-2^{-c+2})/(1-2^{-c+1}))/(1-2c/d))$. For the choices of c and d, and $S(d) = 8$, the number of comparisons is no larger than $28n$. Choosing $c = 2$ and $d = 13$ yields a number of comparisons less than $10n$.

By way of comparison, note that use of the Munro and Paterson samples will require $O(n \log s)$ comparisons just to sort the samples for level 0 of the first iteration. Since $s \geq c \log(n/s)$ for some constant c, a first iteration using the straight Munro and Paterson samples will take $O(n \text{ loglog } n)$ time.

5. Selection in a mesh-shaped network

We now consider a *mesh* topology, with m processors arranged in a $\sqrt{m} \times \sqrt{m}$ grid. As a notable consequence of our modified sampling procedure, there is an algorithm that uses just $O(m)$ messages for a mesh when $n = m$ elements. It is natural to use a spanning tree for communication when generating the sample of the current population. However, not all spanning trees are equally good, in terms of minimizing the

number of messages. We have found that bushier trees are better in this respect.

We use the following spanning tree of the mesh for our communications. Assume $m = 2^{2a}$ for some integer a. For $a > 0$, the mesh is composed of four submeshes of size $2^{2(a-1)}$. The spanning tree will contain the edges of the spanning trees of the four submeshes, along with the topmost edge that connects the top two submeshes, the leftmost edge that connects the leftmost two, and the leftmost edge that connects the rightmost two. The root of the resulting spanning tree will be the upper leftmost node in the mesh. An example of such a spanning tree is shown in Figure 1.

We first consider the case in which there is initially a single value at each processor. We use (c, s, d)-sampling with $c = 4$, and $d = \Theta(s / \log m)$. Samples at level 0 will be accumulated in the following way. For each node v in the spanning tree, compute σ_v, the number of descendants (including v itself) that have active elements. The samples at level 0 are accumulated by forwarding upward toward the root from v the first $\sigma_v \bmod d$ elements that reach it (including any element that starts at v).

The number of samples at level 0 accumulated at descendants of node v will thus be $\sigma_v' = \lfloor \sigma_v / d \rfloor$. The routing and merging of samples is accomplished in the following way. If the ith bit in the binary representation of σ_v' is 1, then the first sample at level i reaching v will be forwarded to the parent of v in the spanning tree. Any other sample at level i arriving at v will be thinned (if appropriate) and merged with a matching sample at the same level. The result will be forwarded according to the same rules.

Lemma 4. Let each processor of an m-node mesh contain one element of a set.

Selection can be performed in the set using $O(m\sqrt{s/\log m})$ messages and delay $O(\sqrt{m}\,(\log m)/(\log(s/\log m)))$, $c\log m \le s = O(\sqrt{m}/(\log m))$ for some constant c.

Proof. First consider the forming of all samples at level 0. Each element will traverse no more than $2\sqrt{md/n'}$ edges to reach the root of some submesh of size md/n' (*i.e.*, $\sqrt{md/n'} \times \sqrt{md/n'}$). Thus the total number of messages used to forward all elements to roots of submeshes is $O(\sqrt{mdn'})$. Since the first $\sigma_v \bmod d$ elements reaching node v are forwarded upward, at most $d-1$ elements may be transmitted upward from each submesh of size md/n' during the formation of all level 0 samples. The number of submeshes of size $(md/n')2^{2l}$ is $(n'/d)2^{-2l}$. The total transmission distance from submeshes of size $(md/n')2^{2l}$ to those of size $(md/n')2^{2l+2}$ will be $\sqrt{md/n'}\,2^l$. Thus the total number of messages from the roots of submeshes of size $(md/n')2^{2l}$ to the roots of submeshes of size $(md/n')2^{2l+2}$ will be less than $3n'\sqrt{md/n'}\,2^{-l}$. Thus the total number of messages needed to accumulate all samples at level 0 will be $O(\sqrt{mdn'})$.

The formation of all samples at levels $i > 0$ in a sweep will also use $O(\sqrt{mdn'})$ messages. Each element will participate in no more than $O(\sqrt{md/n'})$ messages up to a root of a submesh of size md/n'. The number of messages from roots of submeshes of size md/n' or larger is maximized if all samples arriving at roots of submeshes of size md/n' are samples at level 0. This follows since all of the n' elements will thus arrive at these roots. In this case, at most one sample at each level i=0, 1, $\cdots$, $2l$+1 will be transmitted upward from a root of a submesh of size at least $(md/n')2^{2l}$. Thus the number of messages transmitted from the root of a submesh of size $(md/n')2^{2l}$ will be no more than $\sum_{i=0}^{2l+1} d2^{\lceil i/4 \rceil} = O(d2^{l/2})$. The total number of messages from all roots of

submeshes of size $(md/n')2^{2l}$ will be $O(d2^{l/2}(n'/d)2^{-2l}\sqrt{mdn'}\,2^{l}) = O(\sqrt{mdn'}\,2^{-l/2})$. Thus $O(\sqrt{mdn'})$ messages are used in the formation of all samples in a sweep.

Let n_q be the number of active elements on sweep q. With d chosen as $s/\log m$, the number of active elements will be at least halved by each sweep, so that $n_q \le m/2^q$. Thus the total number of messages for all sweeps is $O(\sum_q \sqrt{mdn_q}) = O(\sum_q (m\sqrt{d}/2^{q/2}))$, which is $O(m\sqrt{d})$. The message delay for generating the sweep will be no more than $2\sqrt{m} + O(s \log(n'/s))$, which is $O(\sqrt{m})$. As before, the number of sweeps necessary is $O((\log m)/(\log(s/\log m)))$. □

For $s = \Theta(m^{1/2}/(\log m))$, the number of messages is $O(m^{5/4}/\log m)$ and the delay is $O(m^{1/2})$. For $s = \Theta(\log m)$, the lemma gives $O(m)$ messages and delay $O(m^{1/2}\log m)$. It is possible to improve the delay while maintaining a linear number of messages if the value of s is allowed to change from one sweep to the next. A two-level approach would be to have $s = \Theta(\log m)$ for $\Theta(\sqrt{\log m})$ sweeps, and then have $s = 2^{\sqrt{\log m}}$ for the remaining iterations, of which there would be $O(\sqrt{\log m})$. The number of messages for the second group of sweeps would be $O(m)$, giving $O(m)$ messages total. The delay would be $O(\sqrt{m}\sqrt{\log m})$.

The above idea can be extended as follows. Again we have $s = d \log m$. After every sweep we increase the value of d so as to get a larger decrease in the number of remaining elements on the next sweep. Specifically, for sweep q, let n_q, s_q and d_q be the number of active elements, the value of s, and the value of d, respectively. We choose successive values of d_q so that $\sqrt{2md_{q+1}n_{q+1}} \le \sqrt{md_q n_q}$. Taking d_1 a constant, this will guarantee that the number of messages $O(\sum_q \sqrt{md_q n_q})$ for all sweeps will be

$O(m)$.

Theorem 3. Let each processor of an m-node mesh contain one element of a set. Selection can be performed in the set using $O(m)$ messages and delay $O(\sqrt{m}\ \text{loglog}\ m)$.

Proof. With $s_q = d_q \log m$, from Lemma 3 we have $n_{q+1} \le 2cn_q/d_q$. Choosing d_{q+1} such that $2d_{q+1}(2cn_q/d_q) = d_q n_q$ will guarantee that $\sqrt{2md_{q+1}n_{q+1}} \le \sqrt{md_q n_q}$. Thus $d_{q+1} = d_q^2/(4c)$. It follows that $d_q = d_1^{2^{q-1}}/(4c)^{2^{q-1}-1}$. Thus for $d_1 > 4c$, there will be $O(\text{loglog}\ m)$ sweeps. Since the delay on each sweep will be $O(\sqrt{m})$, the bound on the delay follows. By an argument similar to that in the proof of Lemma 4, the number of messages used on the qth sweep will be $O(\sqrt{md_q n_q})$, which is $O(m/2^{q/2})$ by choice of d_q. Thus over all sweeps there will be $O(m)$ messages. □

We move on to the case in which there are $n > m$ elements. Let n_k' be the number of active elements at processor P_k. Ignore $n_k' \bmod \lceil n'/(md^{1/3}) \rceil$ elements at processor P_k. Let $d = s/\log m$. For each sweep with $n' > m$ we use $(c, s, d^{1/3})$-samples. For the later sweeps, we use (c, s, d)-samples. The new filters must again be chosen differently, since some elements have been ignored. For each sweep with $n' > m$, there will be no more than $m\lfloor n'/(md^{1/3}) \rfloor \le n'/d^{1/3}$ elements ignored. Thus M_{ij} is larger by at most $n'/d^{1/3} = (n'/s)(s/d^{1/3})$ elements. Thus the value of u should be smaller by $s/d^{1/3}$.

Theorem 4. Let $n \ge m$ elements be distributed among the processors of an m-node mesh. Selection can be performed in the set using $O(m\sqrt{s/\log m}\ (\log(2n/m))/(\log(s/\log m)))$ messages and $O(m^{1/2}(\log n)/(\log(s/\log m)))$ delay, $c \log m$

$\leq s = O(\sqrt{m}/(\log m))$, for some constant c.

Proof. For $n' > m$, the number of elements contained in all samples will be $O(md^{1/3})$. Adapting the argument of Lemma 4, this means that on one sweep with $n' > m$, there will be $O((md^{1/3})(d^{1/3})^{1/2}) = O(md^{1/2})$ messages. There will be $O((\log(n/m))/(\log d))$ sweeps of this type. The delay for each sweep will be $O(\sqrt{m})$. Thus the number of messages consumed by sweeps in which $n' > m$ is $O(m\sqrt{d}(\log(n/m))/\log d)$ $= O(m\sqrt{s/\log m}(\log(n/m))/\log(s/\log m))$. The delay will be $O(\sqrt{m}(\log(n/m))$ $/\log(s/\log m))$. Adding to these the bounds from Lemma 4 for the messages and delay when $n' \leq m$ gives the claimed results. □

If $s = \Theta(\log m)$, then $O(m \log(2n/m))$ messages and $O(\sqrt{m} \log n)$ delay will suffice.

6. Selection in a tree-shaped network

In this section we handle a *tree* topology, with m processors arranged in the configuration of a complete binary tree. Our algorithm will use (c, s, d)-samples with $c = 2$, $s = \Theta(\log n)$, and d set equal to an appropriate constant. Let σ_v be as before. Accumulate the samples at level 0 by forwarding up the first $\sigma_v \bmod d$ elements that reach v. Let $\sigma_v' = \lfloor \sigma_v/d \rfloor$. As before, if the ith bit in the binary representation of σ_v' is 1, then the first thinned sample at level i reaching v will be forwarded to the parent of v.

Lemma 5. Let each processor of an m-node complete binary tree network contain one element of a set. Selection can be performed in the set using $O(m)$ messages and delay

$O((\log m)^3)$.

Proof. The proof is similar to that of Lemma 4. The formation of all samples at level 0 will require at most $O(n' \log(m/n'))$ messages, by the following reasoning. Each element will traverse $O(\log(m/n'))$ edges to reach a vertex at depth $\log n'$ in the tree. At most $d-1$ elements can be forwarded up from each of the $O(n')$ nodes at depths no greater than $\log n'$. Thus $O(n')$ messages suffice to complete the samples at level 0.

The formation of all samples at levels $i > 0$ in a sweep will also use $O(n' \log(m/n'))$ messages. Each element will participate in no more than $O(\log(m/n'))$ messages up to a vertex at depth $\log(n'/d)$ in the tree. The number of messages from vertices at this depth and above is maximized if all samples arriving at vertices at depth $\log(n'/d)$ are samples at level 0. The number of messages transmitted from a vertex at depth $\log(n'/d) - l$ will be no more than $\sum_{i=0}^{l} d2^{\lceil i/2 \rceil} = O(d2^{l/2})$. The total number of messages from all vertices at depth $\log(n'/d) - l$ will be $O(d2^{l/2}(n'/d)2^{-l}) = O(n'2^{-l/2})$. Thus $O(n')$ messages are used at depth $\log(n'/d)$ and above in the tree.

Let n_q be the number of active elements on sweep q. As before, $n_q \leq m/2^q$. Thus the total number of messages for all sweeps is $O(\sum_q (n_q \log(m/n_q)) = O(\sum_q (mq/2^q))$, which is $O(m)$. The message delay for generating the sweep will be no more than $O(\log m + s \log(n'/d))$. This yields a total delay of $O(s(\log m)^2)$ for all sweeps. □

The case in which there are $n > m$ elements is similar to that in the previous section. Group elements into groups of size $\lceil n'/(md) \rceil$. No more than n'/d elements are

ignored in a sweep. Thus M_{ij} may be larger by the quantity n'/d, and u is adjusted accordingly.

Theorem 5. Let $n \geq m$ elements be distributed among the processors of an m-node binary tree network. Selection can be performed in the set using $O(m \log(2n/m))$ messages and $O((\log n)(\log m)^2)$ delay.

Proof. From the above discussion, while $n' > m$, there will be $O(md)$ elements in samples at level 0. Since d is a constant, and for all levels of samples, this implies $O(m)$ messages and delay $O((\log m)^2)$ per sweep. Since n' is reduced by a constant factor on each sweep, there will be $O(\log(n/m))$ such sweeps with $n' > m$. Thus for $n' > m$ there will be $O(m \log(n/m))$ messages and $O((\log m)^2(\log(n/m)))$ delay. Combining this with the previous lemma will yield the claimed result. □

7. A lower bound on message transmission

We have shown that selection can be performed on a mesh or a tree network using $O(m \log(2n/m))$ messages. In this section we show that this is asymptotically optimal. Our argument can be viewed as a generalization of the lower bound argument for two processors in [R] that is attributed to Nick Pippenger. We establish a lower bound in a network model in which every processor is connected to every other processor. A query message will consist of two phases. The forward phase will supply an element, along with its rank in the subset of the originating processor. The return phase will supply the rank of the element in the subset at the destination processor. Precisely stated,

the rank at the destination will be with respect to the subset with the element inserted into it. Certainly, the basic technique employed in our algorithms can be adapted to yield an algorithm with $O(m \log(2n/m))$ messages in this model.

For our lower bound we assume that each element is distinct, and the element to be selected is the median. Given an odd number n of elements, let the number of elements in the subset of each processor be either $\lfloor n/m \rfloor$ or $\lceil n/m \rceil$. Let the number of elements at processor P_k equal the number at processor $P_{k \oplus 1}$ for all such processor pairs that do not include the last processor. If m is even the last processor will contain one more element than the next to the last.

The lower bound argument uses an adversary. When a message is sent, the adversary will provide the requested rank information, which will always be consistent with previous answers and will be contrived to give the adversary a large degree of freedom. For its own benefit, the adversary will view all messages as serialized. The adversary will maintain the following information, also for its own benefit. For the subset at each processor, it will maintain a partition of the elements into three subsets:

1. those elements that are *candidates* for being the median,

2. those elements deemed smaller than the eventual median, called *small elements,*

3. those elements deemed larger than the eventual median, called *large elements.*

For small elements, the adversary will maintain a total order. Similarly for large elements, a total order will be maintained. The adversary will consider processors as paired together, and will maintain the invariant that the number of small (large) elements

at processor P_k will equal the number of large (small) elements at processor $P_{k\oplus 1}$. (The invariant will be modified slightly to deal with the last processor.) Initialization for the adversary is as follows. If m is odd then no elements at the last processor will be deemed candidates. In this case the median of the subset, and all smaller elements, will be deemed small, and the remainder large. If m is even, then one element at the last processor will be deemed small. The rest of the elements will be candidates.

When a query message is transmitted, the adversary must respond with appropriate rank information. If the message's element is small, then the adversary consults the total order for small elements, and finds the corresponding rank within the destination processor's subset. In this case no candidates will have their status changed. If the message's element is large, then the adversary will handle this case similarly. If the message's element is a candidate, the adversary will do the following. Suppose the element is from processor P_k. If the element is no larger than the (lower) median of the candidates at P_k, then the adversary will determine the rank between the small and candidate elements at the destination processor, and respond with this rank. It will make the element, and all candidates smaller than it at P_k small elements. It will extend the total order on small elements by making these new small elements larger than any other elements currently in the total order. The adversary will also make an equal number of candidates at processor $P_{k\oplus 1}$ large, and extend the total order on large elements similarly. If the element is above the (lower) median, then a similar procedure is carried out, with the appropriate candidate elements at P_k made large.

The rank information given is always consistent, since whenever an element's

rank is determined, it is immediately classified as large or small, and entered into the appropriate total order in a fashion consistent with previous query responses. When there are exactly two candidate elements remaining, the adversary deems one of them the median, and the other large. The remainder of the queries may be resolved by using the resulting total order for appropriate answers to queries.

As a result of the adversary handling a message, the number of candidate elements at the originating processor and its paired processor are at worst halved. Hence we have the following result.

Theorem 6. Let $n \geq m$ elements be distributed among the processors of an m-node complete interconnection network. The number of messages required to compute the median is $\Omega(m \log(2n/m))$. □

Corollary 1. The message complexity of selecting the median in a set distributed among the processors of a tree or mesh network is $\Theta(m \log(2n/m))$. □

8. Conclusion

We have investigated the problem of selecting an element of given rank in a set of elements distributed among the nodes of a network. Our goal has been to determine how the topology of the network affects the complexity of solving this problem. For the ring and mesh network topologies, interesting upper bound tradeoffs were identified. The tradeoffs were such that a reasonably mild increase in the delay allowed due to message transmission will give a meaningful decrease in the number of messages used.

Algorithms were presented for the mesh and tree networks whose total number of messages is asymptotically optimal. The number of messages required is no worse than the number of messages required by the same computation on a network with complete interconnection. It remains an open question as to how to take the ring topology into account in deriving lower bounds on the number of messages required on these networks. The issue is complicated in that recent results in [S] suggest that the message complexity may be different for unidirectional and bidirectional rings.

Acknowledgment. The author would like to thank Quentin Stout and a referee for their careful reading of the paper.

References

[BFPRT] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest and R. E. Tarjan, Time bounds for selection, *J. Comput. Sys. Sci. 7*, 4 (1973) 448-461.

[CY] R. Cole and C. K. Yap, A parallel median algorithm, *Inf. Proc. Lett. 20* (1985) 137-139.

[DKR] D. Dolev, M. Klawe and M. Rodeh, An $O(n \log n)$ unidirectional distributed algorithm for extrema finding in a circle, *J. Algorithms 3*, 3 (Sept. 1982) 245-260.

[F] G. N. Frederickson, Tradeoffs for selection in distributed networks, *Proc. 2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Systems*, Montreal (August 1983) 154-160.

[FJ] G. N. Frederickson and D. B. Johnson, The complexity of selection and ranking in $X+Y$ and matrices with sorted columns, *J. Comput. Sci. Sys. 24*, 2 (April 1982) 197-208.

[K] D. E. Knuth, *The Art of Computer Programming, vol. 3: Sorting and Searching*, Addison Wesley (1973).

[M] T. A. Matsushita, Distributed algorithms for selection, University of Illinois Coordinated Science Laboratory Report T-127 (1983).

[MP] J. I. Munro and M. S. Paterson, Selection and sorting with limited storage, *Theor. Comput. Sci. 12* (1980) 315-323.

[R] M. Rodeh, Finding the median distributively, *J. Comput. Sys. Sci. 24* (1982) 162-166.

[SS] N. Santoro and J. B. Sidney, Order statistics on distributed sets, *Proc. 20th Allerton Conf. on Communication, Control and Computing*, University of Illinois (October 1982) 251-256.

[SPP] A. Schönhage, M. Paterson and N. Pippenger, Finding the median, *J. Comput. Sys. Sci. 13* (1976) 184-199.

[S] Q. Stout, private communication (1983).

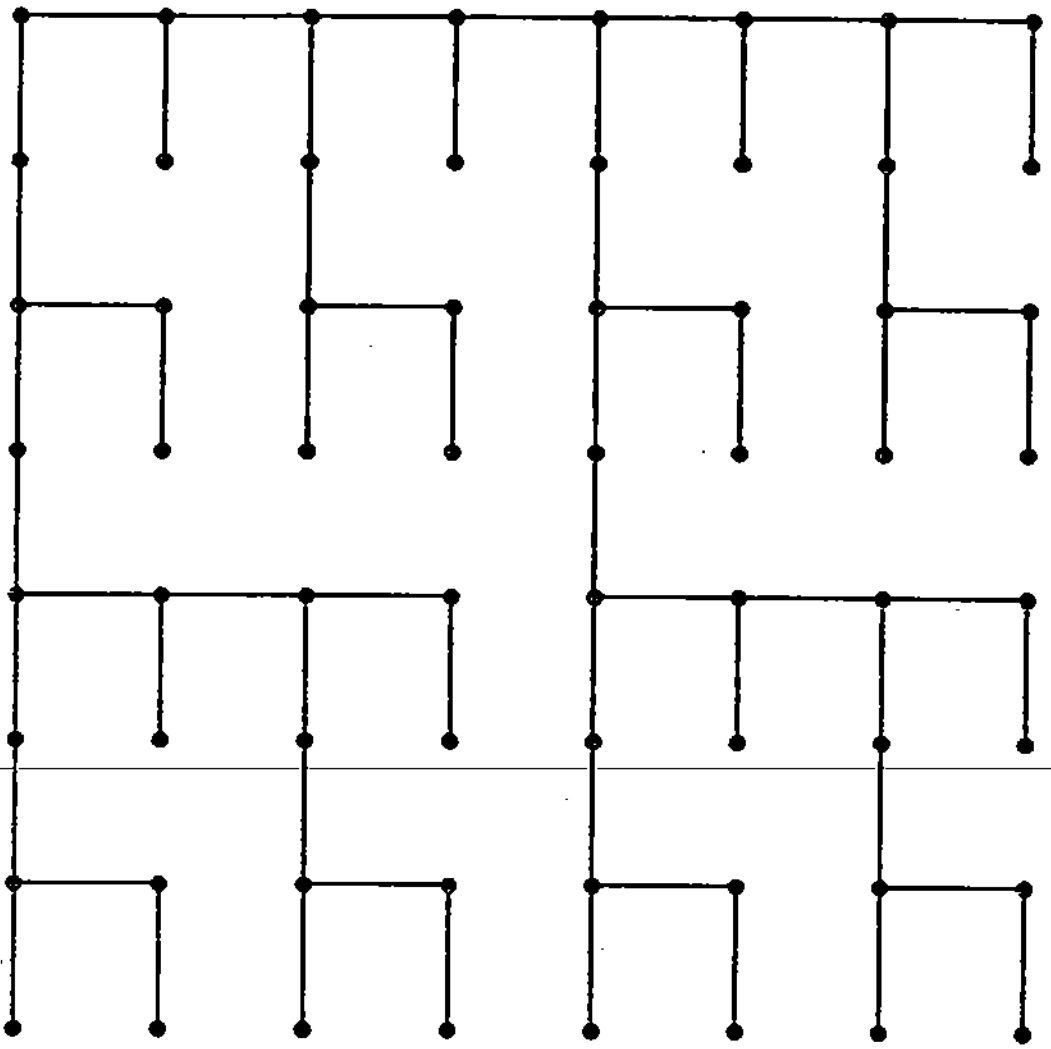

Figure 1. An appropriate spanning tree for a mesh.