AN OPTIMAL SHOUT-ECHO ALGORITHM FOR SELECTION IN DISTRIBUTED SETS

John M. Marberg
Eli Gafni

April 1985
CSD-850015

An Optimal Shout-Echo Algorithm for Selection in Distributed Sets †

John M. Marberg
Eli Gafni

Computer Science Department
University of California
Los Angeles, CA 90024

Abstract

In a Shout-Echo network, processors communicate by means of broadcast messages (shout) and replies to received messages (echo). Performance is measured in terms of communication cycles, where each cycle consists of one broadcast message and the replies to it from all other processors. In this paper we consider the problem of selection by rank in a set distributed among the processors of a Shout-Echo network. We show a tight lower and upper bound of size $\Theta(\log \min\{n_{max2}, k\})$ on the number of cycles required for selection, where n_{max2} is the size of the second-largest subset of elements in any processor, and k is the rank of the element to be selected. We first show the lower bound, then give an efficient algorithm which achieves this bound in a wide range of cases. The algorithm improves the previous best upper bound [Rote83] by a factor of $O(\log p)$, where p is the number of processors in the network.

1. Introduction

A *Shout-Echo* network [Sant83a] is a distributed computation model in which a collection of independent processors communicate by means of broadcast messages (shout) and replies to received broadcast messages (echo). The performance of algorithms in this model is measured in terms of the total number of *communication cycles* required by the computation, where a communication cycle consists of one broadcast message and the replies to it from all processors.

In this paper we consider the problem of selecting the k'th largest element in a set of n elements distributed arbitrarily among the processors of a Shout-Echo network. The problem was previously investigated in the context of this model by Santoro and Sidney [Sant82, Sant83a, Sant83b], who presented two different selection algorithms which run in $O(\min\{k, p\log\frac{k}{p}\})$ cycles, where p is the number of processors in the network. They also showed a lower bound of $\Omega(\log k)$ cycles for the case where the set is evenly distributed (i.e., all

† This research was supported by an IBM Faculty Development Award.

processors have the same number of elements) and $k \leq \frac{n}{p}$. The lower bound is a generalization of the $\Omega(\log n)$ bound on finding the median when $p=2$, shown by Rodeh [Rode82]. Rotem, Santoro and Sidney [Rote83] gave an $O(\log p \log \frac{n}{p})$ algorithm for finding the median of an evenly distributed set. This algorithm can be easily generalized for arbitrary (uneven) distributions and any rank k, thereby reducing the upper bound of the general problem to $O(\log p \log \min\{n_{max2}, k\})$, where n_{max2} is the size of the second-largest subset of elements in any processor.

Our work improves on the above results in the following way. First, we generalize the lower bound by showing that $\Omega(\log \min\{n_{max2}, k\})$ cycles are required for any rank k and any distribution of the elements. We then present a selection algorithm which runs in $O(\log k)$ cycles when $k \leq n_{max2}$, and otherwise in $O(\log n)$ cycles. By showing that in a wide range of cases $O(\log n) = O(\log n_{max2})$, we prove that the upper and lower bounds on selection are tight. Our algorithm thus improves the previous best upper bound by a factor of $\log p$.

The remainder of this paper is organized as follows. Section 2 defines our notation. The lower bound is shown in Section 3. Sections 4 presents the selection algorithm, and Section 5 analyzes its complexity. Concluding remarks are given in Section 6.

2. Definitions and Notation

We denote the p processors of a Shout-Echo network by $P_1, P_2, \ldots, P_p$, where each P_i is a unique identifier taken from a totally ordered set of identifiers.

Let N be a collection of n elements distributed arbitrarily among the processors. Without loss of generality (w.l.g.) we may assume that N is a set, i.e., that all elements in N are distinct. If not, we can replace each element ξ in P_i with the triple (ξ, P_i, j_ξ) where j_ξ is a unique index within P_i, and use lexicographic order among the triples. We denote by $N[j]$ the j'th largest element in N.

The subset of N at each processor P_i is denoted N_i, with cardinality $|N_i| = n_i$. We assume that $n_i > 0$. The terms n_{max} and n_{max2} denote, respectively, the largest and the second largest n_i. $N_i[j]$ denotes the j'th largest element in N_i. $N_i[\lceil \frac{n_i}{2} \rceil]$ is called the median of N_i.

Selection is defined as the task of identifying $N[k]$, the k'th largest element in N, for a given rank k. W.l.g. we may assume $1 \leq k \leq \lceil \frac{n}{2} \rceil$ (if not, reverse the sorting order and select the element of rank $n-k+1$).

Throughout the paper we use log to denote logarithm of base 2. We assume that each message is of size at most $O(\log\beta)$ bits, where β is the value of the largest parameter or datum involved in the computation.

3. Lower Bounds

To show the lower bounds we use an adversary argument adapted from Frederickson [Fred83]. The discussion is limited to comparison-based algorithms.

Theorem 1. Let $n_{max2} \leq k \leq \lceil \frac{n}{2} \rceil$. The number of communication cycles required to select the k'th largest element in a Shout-Echo network is $\Omega(\log n_{max2})$.

Proof. We devise an adversary that, given the cardinalities n_i and a selection algorithm, provides input sets N_i such that the algorithm requires $\Omega(\log n_{max2})$ cycles when executed on that input. The adversary is free to make each element arbitrary large or small, as long as the relative order in each N_i is maintained consistently. Initially, none of the elements has a fixed magnitude. The adversary follows the execution of the algorithm, fixing the magnitude of elements as the algorithm proceeds. Elements not yet fixed are candidates for selection. Fixed elements are made either "very small" or "very large", in the sense that they are smaller or larger than all the remaining candidates in the network. By keeping fewer than k very large elements and fewer than $n-k$ very small elements at all times, the adversary excludes such elements from being selected. Clearly, the algorithm cannot terminate before the number of candidates is reduced to one. Total order is maintained among the fixed elements of all processors by making each new very small element (very large element, resp.) larger (smaller) than all the existing very small (very large) elements.

Let P_{max} and P_{max2} denote two processors which have, respectively, n_{max} and n_{max2} elements. The adversary initializes the elements of N as follows. n_{max2} elements in each of P_{max} and P_{max2} are made candidates. Among the remaining $n-2n_{max2}$ elements, an arbitrary $k-n_{max2}$ elements are made very large and the rest very small. With this setup, the problem of selecting $N[k]$ reduces to finding the median of the $2n_{max2}$ candidates.

Whenever a message is sent which contains a candidate of P_{max} that is smaller or equal (larger, resp.) than the median of the candidates in P_{max}, the adversary fixes this candidate and all those smaller (larger) than it in P_{max} to be very small (very large), and an equal number of candidates in P_{max2} to be very large (very small). From now on, these elements are no longer candidates. When the message contains a candidate of P_{max2}, the same action with the roles of P_{max} and P_{max2} reversed is taken. No action is taken by the adversary if the message does not contain a candidate. Concurrent messages are handled in some arbitrary order.

Let $2m$ be the number of remaining candidates immediately before a message containing one of the candidates is sent. It can be seen that no more than $m+1$ candidates are fixed by the adversary as a result of that message. Since there are initially $2n_{max2}$ candidates, all of them located in P_{max} and P_{max2}, $\Omega(\log 2n_{max2})$ messages originating from either P_{max} or P_{max2} are needed to reduce the number of candidates in the network to one. Clearly, this requires $\Omega(\log n_{max2})$ cycles. ■

We now show a lower bound for the case $k \leq n_{max2}$.

Theorem 2. Let $k \leq n_{max2}$. The number of communication cycles required to select the k'th largest element in a Shout-Echo network is $\Omega(\log k)$.

Proof. We use the same adversary argument as in Theorem 1. The only difference is in the initial choice of candidates and fixed elements. Here, only k elements in each of P_{max} and P_{max2} are made candidates, and all the remaining elements in the network are made very small. The result follows from a similar analysis to that of Theorem 1. ■

Corollary 1. The number of communication cycles required to select the k'th largest element in a Shout-Echo network is $\Omega(\log \min\{k, n_{max2}\})$. ■

4. The Selection Algorithm

A naive approach to selection is to sort all elements, then retrieve the selected element directly by rank. This, however, is inefficient because the extra information provided by sorting comes at a cost and is not really needed. A more promising approach is the following. Reduce the number of candidates for selection by repetitively applying some filtering mechanism. When the number of remaining candidates gets below a specified threshold value, sort the remaining candidates and retrieve the selected element by rank. In the sequel we present a selection algorithm which follows this approach. The algorithm works for arbitrary distribution of the input. We first give the algorithm and show its correctness, then describe the implementation in the Shout-Echo model. The complexity is discussed in Section 5.

4.1. Description of the Algorithm

Let us denote the number of remaining candidates for selection at each stage of the algorithm by m. The subset of remaining candidates in each P_i is denoted M_i, with cardinality $|M_i| = m_i$. The threshold value discussed above is denoted m^*, and k is the rank of the element to be selected. The algorithm consists of three stages: initialization (step 1); iterative filtering (steps 2-6); and termination (step 7).

Algorithm Select(k)

(1) $M_i := N_i$; $m_i := n_i$; $m := \sum_{i=1}^{p} m_i$.

(2) Find the median $M_i[\lceil \frac{m_i}{2} \rceil]$ of each M_i.† Denote this element med_i.
If M_i is empty, med_i is set to a dummy value.

(3) Sort the medians in descending order $med_{i_1} \geq med_{i_2} \geq \cdots \geq med_{i_p}$.

(4) Find the smallest index l with respect to the sorted medians such that $\sum_{j=1}^{l} m_{i_j} \geq \frac{m}{2}$. Denote med_{i_l} as $med_{1/2}$.‡

(5) Find the total number of candidates that are greater or equal to $med_{1/2}$. Denote this number $m_{1/2}$.

(6) Compare $m_{1/2}$ with k and choose the appropriate case below.

Case 1. $m_{1/2} = k$

The selected element is $med_{1/2}$; terminate the algorithm.

Case 2. $m_{1/2} > k$

(a) Purge from each M_i the elements smaller or equal to $med_{1/2}$.
(b) $m_i := |M_i|$; $m := m_{1/2} - 1$.
(c) If $m > m^*$ then go to step 2; otherwise go to step 7.

Case 3. $m_{1/2} < k$

(a) Purge from each M_i the elements greater or equal to $med_{1/2}$.
(b) $m_i := |M_i|$; $m := m - m_{1/2}$; $k := k - m_{1/2}$.
(c) If $m > m^*$ then go to step 2; otherwise go to step 7.

(7) (a) Collect all the remaining candidates into one set M.
(b) Find $M[k]$, which is the selected element; terminate the algorithm.

† This can be done using some efficient sequential selection algorithm (e.g., [Blum73]) or by sorting.

‡ Intuitively, $med_{1/2}$ is chosen so that sufficiently many candidates are larger than it and sufficiently many are smaller. This will be further explained in Section 5.

We now show that the algorithm works correctly. Assume inductively that at the beginning of the current iteration of the filtering stage (steps 2-6) the element to be selected has not been purged, and that k is the correct rank of this element among the remaining candidates. This is clearly true at the beginning of the algorithm.

In case 1 of step 6, the number of candidates greater or equal to $med_{1/2}$ is found to be k. Since w.l.g. we assume that all elements are distinct, the decision to select $med_{1/2}$ is correct. In case 2, since $m_{1/2}>k$, the element we are looking for is greater than $med_{1/2}$. Thus, all candidates smaller or equal to $med_{1/2}$ can be purged. In case 3 a similar argument applies, except that since the $m_{1/2}$ candidates being purged are greater than the selected element, the rank k has to be reduced by the same quantity. Since at least one candidate is purged in each occurrence of case 2 or 3, m becomes smaller and smaller in each iteration, and the algorithm will eventually either terminate in case 1 of step 6, or reach step 7 where the correct element is selected by collecting all the remaining candidates into one set M and finding $M[k]$.

4.2. Implementation in the Shout-Echo Model

The implementation of the algorithm in the Shout-Echo model is straightforward. An arbitrary processor is designated "supervisor". A computation that requires local data of more than one processor is implemented as follows. The supervisor requests the data by means of a broadcast message, and each processor replies by sending its local data. The supervisor then performs the computation and broadcasts the result to all processors.

In the following we give the implementation details of each step. W.l.g. let us assume that the supervisor is P_1. In step 1, P_1 requests m_i from all processors, then computes m and broadcasts it to all processors. Step 2 is performed locally at each processor. In step 3, processors sends med_i to P_1, who then sorts the medians. In step 4, the values m_i are sent to P_1, who finds $med_{1/2}$ and broadcasts it to all processors. In step 5, each P_i finds the number of candidates in M_i that are greater or equal to $med_{1/2}$. These numbers are then sent to P_1, who adds them up and broadcasts the sum $m_{1/2}$ to all the processors. Step 6 is performed locally at each processor. In step 7, all the remaining candidates are collected into one set at P_1. This is done in multiple cycles. In each cycle each processor sends one element. When all m candidates have been collected, P_1 finds the selected element and notifies all the processors.

5. Complexity Analysis

In analyzing the cycle complexity of the algorithm, we must calculate the cost of each step and determine the number of iterations performed in the filtering stage.

The analysis of the number of filtering iterations is illustrated by Figure 1, which captures the situation at the beginning of a typical iteration. The sets of candidates M_{i_j} are ordered in descending order of the medians from left to right. The elements in each set are shown in descending order from top to bottom. Since the medians are ordered, it can be seen that for any given set M_{i_j}, half the candidates in M_{i_j} and at least half the candidates in each set to the right of M_{i_j} are smaller or equal to med_{i_j}. Similarly, half the set M_{i_j} and at least half of each set to the left of M_{i_j} are greater or equal to med_{i_j}. This is shown by the encircled areas in Figure 1.

In particular, since $med_{1/2}(=med_{i_l})$ was chosen such that $\sum_{j=1}^{l} m_{i_j}$ is the smallest partial sum of candidates which is greater or equal to $\frac{m}{2}$, it can be seen that at least $\frac{m}{4}$ candidates are smaller or equal to $med_{1/2}$, and at least $\frac{m}{4}$ candidates are greater or equal to $med_{1/2}$. Consequently, in each of cases 2 and 3 of step 6, at least one fourth of the remaining candidates are purged. Thus, $O(\log\frac{n}{m^*})$ iterations suffice in order to reduce the number of candidates below m^*.

From the implementation in Section 4 it can be seen that each of steps 1-6 requires a constant number of cycles. The number of cycles in step 7 is bounded by $O(m^*)$. The total complexity of the selection algorithm is therefore $O(m^*+\log\frac{n}{m^*})$ cycles. Choosing $m^*\leq\log n$, the complexity of the algorithm is $O(\log n)$ cycles. The following corollary shows that this is optimal in a wide range of cases.

Corollary 2. Let $k\geq n_{max2}$. Given a constant $0<\epsilon\leq 1$ such that $n_{max2}\geq\epsilon p$ and $n_{max}\leq\frac{n_{max2}^2}{\epsilon}$, the complexity of selecting the k'th largest element in a Shout-Echo network is $\Theta(\log n_{max2})$ cycles.

Proof. $n_{max2}\geq\epsilon p$ and $n_{max}\leq\frac{n_{max2}^2}{\epsilon}$ imply $n\leq(p-1)n_{max2}+n_{max}\leq\frac{2n_{max2}^2}{\epsilon}$, and hence $O(\log n)=O(\log n_{max2})$. The result follows from the discussion above and from the lower bound of Theorem 1. ■

We now give simple modifications to step 1 which will improve the performance of the algorithm for ranks $k\leq n_{max2}$. We first discuss the case $p\leq k\leq n_{max2}$. Clearly, only the k largest elements in each set N_i have the potential of being selected. Therefore, instead of initializing each M_i to the entire set N_i, we take only the k largest elements of N_i. This can be done by

sorting N_i locally at P_i. The initial number of candidates is therefore at most kp. If we choose $m^* \leq \log k$, the complexity of the algorithm is $O(m^* + \log \frac{kp}{m^*}) = O(\log k)$. Given the lower bound of Theorem 2, this is optimal.

For the case $k < p$ we use the following observation. Consider the maximum element in each set N_i. Since there are more than k sets, only the k largest among the maximum elements have the potential of being selected. We may thus purge from candidacy in its entirety any set N_i whose maximum element is smaller than the k'th largest maximum element. To do this, we modify step 1 as follows. The maximum element of N_i is determined locally at P_i and sent to P_1, who then finds the k'th largest among the maximum elements and broadcasts it to all processors. Processors whose maximum element is smaller than the element broadcast by P_1 initialize M_i to the empty set, whereas other processors take as initial candidates the k largest elements of N_i. The computation then proceeds as before. Since now only k processors have candidates, the initial number of candidates is at most k^2. Choosing $m^* \leq \log k$, the complexity is $O(m^* + \log \frac{k^2}{m^*}) = O(\log k)$ cycles, which is optimal.

Corollary 3. Let $k \leq n_{max2}$. The complexity of selecting the k'th largest element in a Shout-Echo network is $\Theta(\log k)$ cycles.

Proof. The result follows from the above discussion and from Theorem 2. ■

6. Conclusions

In this paper we have considered the complexity of selection by rank in Shout-Echo networks. We have first shown a lower bound, then given an algorithm which achieves this bound in a wide range of cases, thus proving that the bound is tight. The algorithm improves by a factor of $\log p$ the previous best upper bound, derived from the work in [Rote83].

In [Sant83c], Santoro, Scheutzow and Sidney presented a probabilistic shout-echo selection algorithm and showed that the expected complexity of the algorithm is $O(\log n)$ cycles. The worst-case behavior of our algorithm compares favorably with the average-case behavior of that algorithm.

Santoro and Sidney [Sant83b] showed that for a threshold value of $m^* = O(p)$, the expected number of filtering iterations in the selection algorithm of [Sant83b] is $O(\log \log k)$. Also, in [Sant83c] it is shown that by combining the algorithm of [Sant83b] with the probabilistic algorithm of [Sant83c], an expected complexity of $O(\log p + \log \log k)$ cycles is achieved. It is interesting to see how an average-case analysis of our algorithm would compare with these results.

References

[Blum73] Blum, M., R.W. Floyd, V. Pratt, R.L. Rivest, and R.E. Tarjan, "Time Bounds for Selection," *JCSS* **7**, 4 (Aug. 1973), pp. 448-461.

[Fred83] Frederickson, G.N, "Tradeoffs for Selection in Distributed Systems," in *Proceedings 2nd ACM Symp. on Principles of Distributed Computing,* 1983, pp. 154-160.

[Rode82] Rodeh, M., "Finding the Median Distributively," *JCSS* **24**, 2 (April 1982), pp. 162-166.

[Rote83] Rotem, D., N. Santoro, and J.B. Sidney, "A Shout-Echo Algorithm for Finding the Median of a Distributed Set," in *Proceedings 14th S.E. Conf. on Combinatorics, Graph Theory and Computing,* Boca Raton, FL, 1983, pp. 311-318.

[Sant82] Santoro, N. and J. Sidney, "Order Statistics on Distributed Sets," in *Proceedings 20th Allerton Conf. on Communication, Control and Computing,* Univ. of Illinois at Urbana Champaign, 1982, pp. 251-256.

[Sant83a] Santoro, N. and J.B. Sidney, "Communication Bounds for Selection in Distributed Sets," Working Paper 83-39, Faculty of Administration, Univ. of Ottawa, Ottawa, Canada, 1983.

[Sant83b] Santoro, N. and J. B. Sidney, "A Reduction Technique for Distributed Selection: I," Tech. Rep. SCS-TR-23, School of Computer Science, Carleton Univ., Ottawa, Canada, April 1983.

[Sant83c] Santoro, N. and J. B. Sidney, "A Reduction Technique for Distributed Selection: II," Tech. Rep. SCS-TR-35, School of Computer Science, Carleton Univ., Ottawa, Canada, Dec. 1983.

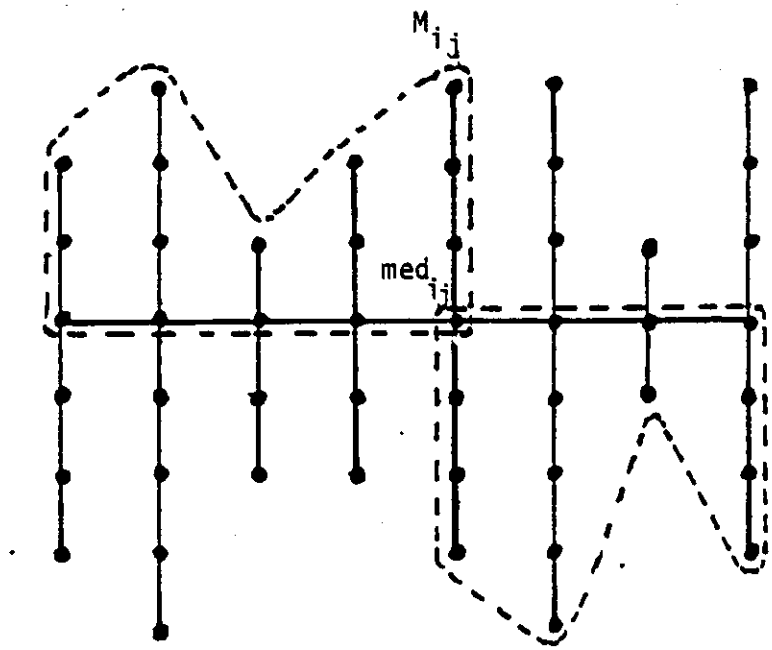

Figure 1. The Filtering Stage

AN OPTIMAL SHOUT-ECHO ALGORITHM FOR SELECTION IN DISTRIBUTED SETS

John M. Marberg
Eli Gafni

April 1985
Report No. CSD-850015

An Optimal Shout-Echo Algorithm for Selection in Distributed Sets †

John M. Marberg
Eli Gafni

Computer Science Department
University of California
Los Angeles, CA 90024

Abstract

In a Shout-Echo network, processors communicate by means of broadcast messages (shout) and replies to received messages (echo). Performance is measured in terms of communication cycles, where each cycle consists of one broadcast message and the replies to it from all other processors. In this paper we consider the problem of selection by rank in a set distributed among the processors of a Shout-Echo network. We show a tight lower and upper bound of size $\Theta(\log \min\{n_{max2}, k\})$ on the number of cycles required for selection, where n_{max2} is the size of the second-largest subset of elements in any processor, and k is the rank of the element to be selected. We first show the lower bound, then give an efficient algorithm which achieves this bound in a wide range of cases. The algorithm improves the previous best upper bound [Rote83] by a factor of $O(\log p)$, where p is the number of processors in the network.

1. Introduction

A *Shout-Echo* network [Sant83a] is a distributed computation model in which a collection of independent processors communicate by means of broadcast messages (shout) and replies to received broadcast messages (echo). The performance of algorithms in this model is measured in terms of the total number of *communication cycles* required by the computation, where a communication cycle consists of one broadcast message and the replies to it from all processors.

In this paper we consider the problem of selecting the k'th largest element in a set of n elements distributed arbitrarily among the processors of a Shout-Echo network. The problem was previously investigated in the context of this model by Santoro and Sidney [Sant82, Sant83a, Sant83b], who presented two different selection algorithms which run in $O(\min\{k, p\log\frac{k}{p}\})$ cycles, where p is the number of processors in the network. They also showed a lower bound of $\Omega(\log k)$ cycles for the case where the set is evenly distributed (i.e., all

† This research was supported by an IBM Faculty Development Award.

processors have the same number of elements) and $k \leq \frac{n}{p}$. The lower bound is a generalization of the $\Omega(\log n)$ bound on finding the median when $p=2$, shown by Rodeh [Rode82]. Rotem, Santoro and Sidney [Rote83] gave an $O(\log p \log \frac{n}{p})$ algorithm for finding the median of an evenly distributed set. This algorithm can be easily generalized for arbitrary (uneven) distributions and any rank k, thereby reducing the upper bound of the general problem to $O(\log p \log \min\{n_{\max 2}, k\})$, where $n_{\max 2}$ is the size of the second-largest subset of elements in any processor.

Our work improves on the above results in the following way. First, we generalize the lower bound by showing that $\Omega(\log \min\{n_{\max 2}, k\})$ cycles are required for any rank k and any distribution of the elements. We then present a selection algorithm which runs in $O(\log k)$ cycles when $k \leq n_{\max 2}$, and otherwise in $O(\log n)$ cycles. By showing that in a wide range of cases $O(\log n) = O(\log n_{\max 2})$, we prove that the upper and lower bounds on selection are tight. Our algorithm thus improves the previous best upper bound by a factor of $\log p$.

The remainder of this paper is organized as follows. Section 2 defines our notation. The lower bound is shown in Section 3. Sections 4 presents the selection algorithm, and Section 5 analyzes its complexity. Concluding remarks are given in Section 6.

2. Definitions and Notation

We denote the p processors of a Shout-Echo network by $P_1, P_2, \ldots, P_p$, where each P_i is a unique identifier taken from a totally ordered set of identifiers.

Let N be a collection of n elements distributed arbitrarily among the processors. Without loss of generality (w.l.g.) we may assume that N is a set, i.e., that all elements in N are distinct. If not, we can replace each element ξ in P_i with the triple (ξ, P_i, j_ξ) where j_ξ is a unique index within P_i, and use lexicographic order among the triples. We denote by $N[j]$ the j'th largest element in N.

The subset of N at each processor P_i is denoted N_i, with cardinality $|N_i| = n_i$. We assume that $n_i > 0$. The terms $n_{\max}$ and $n_{\max 2}$ denote, respectively, the largest and the second largest n_i. $N_i[j]$ denotes the j'th largest element in N_i. $N_i[\lceil \frac{n_i}{2} \rceil]$ is called the median of N_i.

Selection is defined as the task of identifying $N[k]$, the k'th largest element in N, for a given rank k. W.l.g. we may assume $1 \leq k \leq \lceil \frac{n}{2} \rceil$ (if not, reverse the sorting order and select the element of rank $n-k+1$).

Throughout the paper we use log to denote logarithm of base 2. We assume that each message is of size at most $O(\log\beta)$ bits, where β is the value of the largest parameter or datum involved in the computation.

3. Lower Bounds

To show the lower bounds we use an adversary argument adapted from Frederickson [Fred83]. The discussion is limited to comparison-based algorithms.

Theorem 1. Let $n_{max2} \leq k \leq \lceil \frac{n}{2} \rceil$. The number of communication cycles required to select the k'th largest element in a Shout-Echo network is $\Omega(\log n_{max2})$.

Proof. We devise an adversary that, given the cardinalities n_i and a selection algorithm, provides input sets N_i such that the algorithm requires $\Omega(\log n_{max2})$ cycles when executed on that input. The adversary is free to make each element arbitrary large or small, as long as the relative order in each N_i is maintained consistently. Initially, none of the elements has a fixed magnitude. The adversary follows the execution of the algorithm, fixing the magnitude of elements as the algorithm proceeds. Elements not yet fixed are candidates for selection. Fixed elements are made either "very small" or "very large", in the sense that they are smaller or larger than all the remaining candidates in the network. By keeping fewer than k very large elements and fewer than $n-k$ very small elements at all times, the adversary excludes such elements from being selected. Clearly, the algorithm cannot terminate before the number of candidates is reduced to one. Total order is maintained among the fixed elements of all processors by making each new very small element (very large element, resp.) larger (smaller) than all the existing very small (very large) elements.

Let P_{max} and P_{max2} denote two processors which have, respectively, n_{max} and n_{max2} elements. The adversary initializes the elements of N as follows. n_{max2} elements in each of P_{max} and P_{max2} are made candidates. Among the remaining $n-2n_{max2}$ elements, an arbitrary $k-n_{max2}$ elements are made very large and the rest very small. With this setup, the problem of selecting $N[k]$ reduces to finding the median of the $2n_{max2}$ candidates.

Whenever a message is sent which contains a candidate of P_{max} that is smaller or equal (larger, resp.) than the median of the candidates in P_{max}, the adversary fixes this candidate and all those smaller (larger) than it in P_{max} to be very small (very large), and an equal number of candidates in P_{max2} to be very large (very small). From now on, these elements are no longer candidates. When the message contains a candidate of P_{max2}, the same action with the roles of P_{max} and P_{max2} reversed is taken. No action is taken by the adversary if the message does not contain a candidate. Concurrent messages are handled in some arbitrary order.

Let $2m$ be the number of remaining candidates immediately before a message containing one of the candidates is sent. It can be seen that no more than $m+1$ candidates are fixed by the adversary as a result of that message. Since there are initially $2n_{max2}$ candidates, all of them located in P_{max} and P_{max2}, $\Omega(\log 2n_{max2})$ messages originating from either P_{max} or P_{max2} are needed to reduce the number of candidates in the network to one. Clearly, this requires $\Omega(\log n_{max2})$ cycles. ■

We now show a lower bound for the case $k \leq n_{max2}$.

Theorem 2. Let $k \leq n_{max2}$. The number of communication cycles required to select the k'th largest element in a Shout-Echo network is $\Omega(\log k)$.

Proof. We use the same adversary argument as in Theorem 1. The only difference is in the initial choice of candidates and fixed elements. Here, only k elements in each of P_{max} and P_{max2} are made candidates, and all the remaining elements in the network are made very small. The result follows from a similar analysis to that of Theorem 1. ■

Corollary 1. The number of communication cycles required to select the k'th largest element in a Shout-Echo network is $\Omega(\log \min\{k, n_{max2}\})$. ■

4. The Selection Algorithm

A naive approach to selection is to sort all elements, then retrieve the selected element directly by rank. This, however, is inefficient because the extra information provided by sorting comes at a cost and is not really needed. A more promising approach is the following. Reduce the number of candidates for selection by repetitively applying some filtering mechanism. When the number of remaining candidates gets below a specified threshold value, sort the remaining candidates and retrieve the selected element by rank. In the sequel we present a selection algorithm which follows this approach. The algorithm works for arbitrary distribution of the input. We first give the algorithm and show its correctness, then describe the implementation in the Shout-Echo model. The complexity is discussed in Section 5.

4.1. Description of the Algorithm

Let us denote the number of remaining candidates for selection at each stage of the algorithm by m. The subset of remaining candidates in each P_i is denoted M_i, with cardinality $|M_i| = m_i$. The threshold value discussed above is denoted m^*, and k is the rank of the element to be selected. The algorithm consists of three stages: initialization (step 1); iterative filtering (steps 2-6); and termination (step 7).

Algorithm Select(k)

(1) $M_i := N_i$; $m_i := n_i$; $m := \sum_{i=1}^{p} m_i$.

(2) Find the median $M_i[\lceil \frac{m_i}{2} \rceil]$ of each M_i.† Denote this element med_i.
If M_i is empty, med_i is set to a dummy value.

(3) Sort the medians in descending order $med_{i_1} \geq med_{i_2} \geq \cdots \geq med_{i_p}$.

(4) Find the smallest index l with respect to the sorted medians such that $\sum_{j=1}^{l} m_{i_j} \geq \frac{m}{2}$. Denote med_{i_l} as $med_{1/2}$.‡

(5) Find the total number of candidates that are greater or equal to $med_{1/2}$. Denote this number $m_{1/2}$.

(6) Compare $m_{1/2}$ with k and choose the appropriate case below.

Case 1. $m_{1/2} = k$

The selected element is $med_{1/2}$; terminate the algorithm.

Case 2. $m_{1/2} > k$

(a) Purge from each M_i the elements smaller or equal to $med_{1/2}$.
(b) $m_i := |M_i|$; $m := m_{1/2} - 1$.
(c) If $m > m^*$ then go to step 2; otherwise go to step 7.

Case 3. $m_{1/2} < k$

(a) Purge from each M_i the elements greater or equal to $med_{1/2}$.
(b) $m_i := |M_i|$; $m := m - m_{1/2}$; $k := k - m_{1/2}$.
(c) If $m > m^*$ then go to step 2; otherwise go to step 7.

(7) (a) Collect all the remaining candidates into one set M.
(b) Find $M[k]$, which is the selected element; terminate the algorithm.

† This can be done using some efficient sequential selection algorithm (e.g., [Blum73]) or by sorting.

‡ Intuitively, $med_{1/2}$ is chosen so that sufficiently many candidates are larger than it and sufficiently many are smaller. This will be further explained in Section 5.

We now show that the algorithm works correctly. Assume inductively that at the beginning of the current iteration of the filtering stage (steps 2-6) the element to be selected has not been purged, and that k is the correct rank of this element among the remaining candidates. This is clearly true at the beginning of the algorithm.

In case 1 of step 6, the number of candidates greater or equal to $med_{1/2}$ is found to be k. Since w.l.g. we assume that all elements are distinct, the decision to select $med_{1/2}$ is correct. In case 2, since $m_{1/2}>k$, the element we are looking for is greater than $med_{1/2}$. Thus, all candidates smaller or equal to $med_{1/2}$ can be purged. In case 3 a similar argument applies, except that since the $m_{1/2}$ candidates being purged are greater than the selected element, the rank k has to be reduced by the same quantity. Since at least one candidate is purged in each occurrence of case 2 or 3, m becomes smaller and smaller in each iteration, and the algorithm will eventually either terminate in case 1 of step 6, or reach step 7 where the correct element is selected by collecting all the remaining candidates into one set M and finding $M[k]$.

4.2. Implementation in the Shout-Echo Model

The implementation of the algorithm in the Shout-Echo model is straightforward. An arbitrary processor is designated "supervisor". A computation that requires local data of more than one processor is implemented as follows. The supervisor requests the data by means of a broadcast message, and each processor replies by sending its local data. The supervisor then performs the computation and broadcasts the result to all processors.

In the following we give the implementation details of each step. W.l.g. let us assume that the supervisor is P_1. In step 1, P_1 requests m_i from all processors, then computes m and broadcasts it to all processors. Step 2 is performed locally at each processor. In step 3, processors sends med_i to P_1, who then sorts the medians. In step 4, the values m_i are sent to P_1, who finds $med_{1/2}$ and broadcasts it to all processors. In step 5, each P_i finds the number of candidates in M_i that are greater or equal to $med_{1/2}$. These numbers are then sent to P_1, who adds them up and broadcasts the sum $m_{1/2}$ to all the processors. Step 6 is performed locally at each processor. In step 7, all the remaining candidates are collected into one set at P_1. This is done in multiple cycles. In each cycle each processor sends one element. When all m candidates have been collected, P_1 finds the selected element and notifies all the processors.

5. Complexity Analysis

In analyzing the cycle complexity of the algorithm, we must calculate the cost of each step and determine the number of iterations performed in the filtering stage.

The analysis of the number of filtering iterations is illustrated by Figure 1, which captures the situation at the beginning of a typical iteration. The sets of candidates M_{i_j} are ordered in descending order of the medians from left to right. The elements in each set are shown in descending order from top to bottom. Since the medians are ordered, it can be seen that for any given set M_{i_j}, half the candidates in M_{i_j} and at least half the candidates in each set to the right of M_{i_j} are smaller or equal to med_{i_j}. Similarly, half the set M_{i_j} and at least half of each set to the left of M_{i_j} are greater or equal to med_{i_j}. This is shown by the encircled areas in Figure 1.

In particular, since $med_{1/2}(=med_{i_l})$ was chosen such that $\sum_{j=1}^{l} m_{i_j}$ is the smallest partial sum of candidates which is greater or equal to $\frac{m}{2}$, it can be seen that at least $\frac{m}{4}$ candidates are smaller or equal to $med_{1/2}$, and at least $\frac{m}{4}$ candidates are greater or equal to $med_{1/2}$. Consequently, in each of cases 2 and 3 of step 6, at least one fourth of the remaining candidates are purged. Thus, $O(\log\frac{n}{m^*})$ iterations suffice in order to reduce the number of candidates below m^*.

From the implementation in Section 4 it can be seen that each of steps 1-6 requires a constant number of cycles. The number of cycles in step 7 is bounded by $O(m^*)$. The total complexity of the selection algorithm is therefore $O(m^*+\log\frac{n}{m^*})$ cycles. Choosing $m^*\leq\log n$, the complexity of the algorithm is $O(\log n)$ cycles. The following corollary shows that this is optimal in a wide range of cases.

Corollary 2. Let $k\geq n_{\max 2}$. Given a constant $0<\epsilon\leq 1$ such that $n_{\max 2}\geq\epsilon p$ and $n_{\max}\leq\frac{n_{\max 2}^2}{\epsilon}$, the complexity of selecting the k'th largest element in a Shout-Echo network is $\Theta(\log n_{\max 2})$ cycles.

Proof. $n_{\max 2}\geq\epsilon p$ and $n_{\max}\leq\frac{n_{\max 2}^2}{\epsilon}$ imply $n\leq(p-1)n_{\max 2}+n_{\max}\leq\frac{2n_{\max 2}^2}{\epsilon}$, and hence $O(\log n)=O(\log n_{\max 2})$. The result follows from the discussion above and from the lower bound of Theorem 1. ■

We now give simple modifications to step 1 which will improve the performance of the algorithm for ranks $k\leq n_{\max 2}$. We first discuss the case $p\leq k\leq n_{\max 2}$. Clearly, only the k largest elements in each set N_i have the potential of being selected. Therefore, instead of initializing each M_i to the entire set N_i, we take only the k largest elements of N_i. This can be done by

sorting N_i locally at P_i. The initial number of candidates is therefore at most kp. If we choose $m^* \leq \log k$, the complexity of the algorithm is $O(m^* + \log \frac{kp}{m^*}) = O(\log k)$. Given the lower bound of Theorem 2, this is optimal.

For the case $k < p$ we use the following observation. Consider the maximum element in each set N_i. Since there are more than k sets, only the k largest among the maximum elements have the potential of being selected. We may thus purge from candidacy in its entirety any set N_i whose maximum element is smaller than the k'th largest maximum element. To do this, we modify step 1 as follows. The maximum element of N_i is determined locally at P_i and sent to P_1, who then finds the k'th largest among the maximum elements and broadcasts it to all processors. Processors whose maximum element is smaller than the element broadcast by P_1 initialize M_i to the empty set, whereas other processors take as initial candidates the k largest elements of N_i. The computation then proceeds as before. Since now only k processors have candidates, the initial number of candidates is at most k^2. Choosing $m^* \leq \log k$, the complexity is $O(m^* + \log \frac{k^2}{m^*}) = O(\log k)$ cycles, which is optimal.

Corollary 3. Let $k \leq n_{max2}$. The complexity of selecting the k'th largest element in a Shout-Echo network is $\Theta(\log k)$ cycles.

Proof. The result follows from the above discussion and from Theorem 2. ■

6. Conclusions

In this paper we have considered the complexity of selection by rank in Shout-Echo networks. We have first shown a lower bound, then given an algorithm which achieves this bound in a wide range of cases, thus proving that the bound is tight. The algorithm improves by a factor of $\log p$ the previous best upper bound, derived from the work in [Rote83].

In [Sant83c], Santoro, Scheutzow and Sidney presented a probabilistic shout-echo selection algorithm and showed that the expected complexity of the algorithm is $O(\log n)$ cycles. The worst-case behavior of our algorithm compares favorably with the average-case behavior of that algorithm.

Santoro and Sidney [Sant83b] showed that for a threshold value of $m^* = O(p)$, the expected number of filtering iterations in the selection algorithm of [Sant83b] is $O(\log \log k)$. Also, in [Sant83c] it is shown that by combining the algorithm of [Sant83b] with the probabilistic algorithm of [Sant83c], an expected complexity of $O(\log p + \log \log k)$ cycles is achieved. It is interesting to see how an average-case analysis of our algorithm would compare with these results.

References

[Blum73] Blum, M., R.W. Floyd, V. Pratt, R.L. Rivest, and R.E. Tarjan, "Time Bounds for Selection," *JCSS* **7**, 4 (Aug. 1973), pp. 448-461.

[Fred83] Frederickson, G.N, "Tradeoffs for Selection in Distributed Systems," in *Proceedings 2nd ACM Symp. on Principles of Distributed Computing,* 1983, pp. 154-160.

[Rode82] Rodeh, M., "Finding the Median Distributively," *JCSS* **24**, 2 (April 1982), pp. 162-166.

[Rote83] Rotem, D., N. Santoro, and J.B. Sidney, "A Shout-Echo Algorithm for Finding the Median of a Distributed Set," in *Proceedings 14th S.E. Conf. on Combinatorics, Graph Theory and Computing,* Boca Raton, FL, 1983, pp. 311-318.

[Sant82] Santoro, N. and J. Sidney, "Order Statistics on Distributed Sets," in *Proceedings 20th Allerton Conf. on Communication, Control and Computing,* Univ. of Illinois at Urbana Champaign, 1982, pp. 251-256.

[Sant83a] Santoro, N. and J.B. Sidney, "Communication Bounds for Selection in Distributed Sets," Working Paper 83-39, Faculty of Administration, Univ. of Ottawa, Ottawa, Canada, 1983.

[Sant83b] Santoro, N. and J. B. Sidney, "A Reduction Technique for Distributed Selection: I," Tech. Rep. SCS-TR-23, School of Computer Science, Carleton Univ., Ottawa, Canada, April 1983.

[Sant83c] Santoro, N. and J. B. Sidney, "A Reduction Technique for Distributed Selection: II," Tech. Rep. SCS-TR-35, School of Computer Science, Carleton Univ., Ottawa, Canada, Dec. 1983.

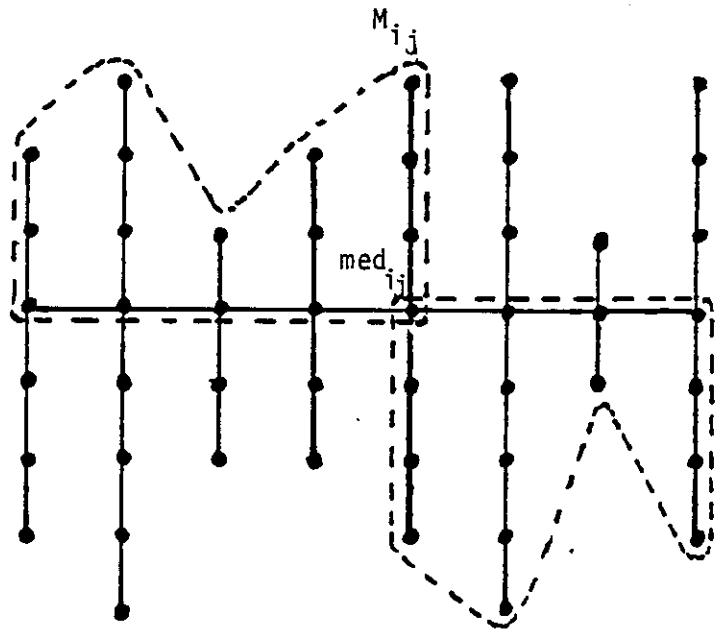

Figure 1. The Filtering Stage