4-24-1994

Tree-based Algorithm to Find the k-th Value in Distributed Systems

Yoonsik Cheon  
Iowa State University

Johnny S. Wong  
Iowa State University, wong@iastate.edu

Follow this and additional works at: http://lib.dr.iastate.edu/cs_techreports

Part of the Theory and Algorithms Commons

Recommended Citation
http://lib.dr.iastate.edu/cs_techreports/33

This Article is brought to you for free and open access by the Computer Science at Digital Repository @ Iowa State University. It has been accepted for inclusion in Computer Science Technical Reports by an authorized administrator of Digital Repository @ Iowa State University. For more information, please contact digirep@iastate.edu.
Tree-based algorithm to Find the k-th Value in Distributed Systems

TR94-07
Yoonsik Cheon and Johnny Wong

April 27, 1994

Iowa State University of Science and Technology
Department of Computer Science
226 Atanasoff
Ames, IA 50011
Tree-based Algorithm to Find the $k$-th Value in Distributed Systems

Yoonsik Cheon and Johnny Wong

TR #94-07
April, 1994

Keywords: searching $k$-th value, extrema finding, distributed algorithms, message passing, tree structures.


Department of Computer Science
226 Atanasoff Hall
Iowa State University
Ames, Iowa 50011-1040, USA
Tree-based Algorithm to Find the $k$-th Value in Distributed Systems

Yoonsik Cheon and Johnny Wong
Department of Computer Science, 226 Atanasoff Hall
Iowa State University, Ames, Iowa 50011-1040 USA
cheon@cs.iastate.edu and wong@cs.iastate.edu

Abstract

In this paper, we study distributed algorithms for finding the $k$-th value in the decentralized systems. First we consider the case of circular configuration of processors where no processor knows the total number of participants. Later a network of arbitrary configuration is examined and a tree-based algorithm is proposed. The proposed algorithm requires $O(N)$ messages and $O(\log N)$ rounds of message passing, where $N$ is the number of nodes in the network.

Keywords: searching $k$-th value, extrema finding, distributed algorithms, message passing, tree.

1 Introduction

Given $N$ processors or computer systems that communicate only with message passing, the distributed extrema-finding problem is to select the processor with the maximum (or minimum) value. Each processor is assumed to have a unique value in a set with a total order. The extrema-finding problem has been studied for quite some time now and several algorithms have appeared in the literature. We extend this problem to find the processor with the $k$-th largest (or smallest) value.

Distributed extrema-finding in circular configuration of processors and where no processor knows the total number of participants has been extensively studied. The original solution is due to LeLann [1] who presented an algorithm that requires $O(N^2)$ messages. Chang and Roberts [2] proposed an improved algorithm that requires only $O(N\log N)$ messages on the average but, in the worst case, still requires $O(N^2)$ messages. Both algorithms assumed the capability of each processor to pass a message "to the left" in a global sense, i.e., the communication is unidirectional. This condition was relaxed in [3] in that a processor can pass messages in either or both directions (i.e., bidirectional communication), giving an algorithm requiring $O(N\log N)$ messages in the worst case. Later, Peterson [4] showed that one still can achieve an $O(N\log N)$ algorithm with unidirectional communications, which is a simulation of a bidirectional algorithm.

The original problem was slightly modified in [5] in that they assumed knowledge of the number of participating processes and of the communication scheme, did not use circular configuration, and required the computation to be done within two rounds of message exchange. Using finite projective planes, they developed an algorithm of $O(N\sqrt{N})$ message passing.

We propose two algorithms for finding the $k$-th value for a computer network of $N$ nodes, communicating only by messages. The messages are guaranteed to be delivered by the communication network. Initially we assume that all the nodes are completely reliable, and node failures will be considered in the later section. The first algorithm is an extension to Peterson’s extrema-finding algorithm [4]. It requires $O(\log N)$ steps and each step requires $O(N)$ message passing, thus requiring $O(N\log N)$ messages in total. In the second algorithm we do not require the processors to be configured in a circle. They can be in an arbitrary network topology. The algorithm uses a spanning tree of the computer network, and is based on the Raymond’s tree algorithm for distributed mutual exclusion problem [6]. The performance of the algorithm depends on the topology of the network spanning tree used, but the average number of steps (round of message passing) is $O(\log N)$ and the total number of messages is $O(N)$.

The main contributions of our work is: (1) This seems to be the first work focused on finding the $k$-
th values in the distributed system. (2) It is shown that an existing extrema-finding algorithm can be easily extended to finding the \( k \)-th value. Though we showed only an extension to the Peterson’s algorithm, we expect other extrema-finding algorithms can also be generalized in the similar manner. (3) It is demonstrated that our tree-based algorithm can be easily and practically implemented using either BSD sockets or RPC.

2 An Extension to Peterson’s Algorithm

Given \( N \) processors in a ring communicating only with message passing to its neighbors, the problem is to find the processor with the \( k \)-th largest (or smallest) value. All processors are identical except for their values. However, a processor can determine whether a message is received from its left neighbor or from its right neighbor. The number of processors, \( N \), is not initially known, a property known as *uniformity* of the algorithms. The algorithm is shown in Figure 1. It is an extension to the Peterson’s extrema finding algorithm [4]. All messages are assumed to be passed only to the right. Processors are divided into two categories: *active* and *relay*. Active processors operate in phases. Relay processors just pass on any messages they receive. All processors are initially active. Each processor starts with its own value as its temporary identifier (\( tid \)). During a phase, each active processor receives the values from its nearest active neighbor to the left (\( ntid \)) and that neighbors nearest left active neighbor (\( nntid \)). Each message consists of a sequence of \( k \) values, i.e., the variables \( tid \), \( ntid \), and \( nntid \) store sequences of values. When an active processor receives two messages \( tid \) and \( nntid \), it computes (by merging) the largest \( k \) values among its own list of values and the \( 2k \) values just received, and forwards the newly computed \( k \) values to its right neighbor. If any value of \( nntid \) is included in the newly computed \( k \) largest values, the processor stays in the active stage; otherwise it jumps to the relay step. Thus, at each phase, out of \( n \) processors about \( n/k \) of active processors move to the relay stage.

The algorithm requires \( k - 1 \) initialization steps (the first \( k - 1 \) phases) during which no active processor jumps to the relay stage. These steps require \((k - 1)N \) messages passes. Thus, in the worst case (i.e., when \( k \) is equal to \( N \)) the algorithm requires \( O(N^2) \) message passes. However, when \( k \) is small enough compared to \( N \), the average case complexity is \( O(N \log N) \). Since at each phase, \( n/k \) of active processors move to the relay stage, there can be at most \([\log N]\) phases to reach one active processor. During each phase every processor sends (and receives) two messages, and at the last phase one active processor sends one message which is relayed around the ring, thus, giving total \( 2N[\log N] + N \) messages. Therefore, the message complexity is \( O(N \log N) \). If \( k = 1 \), the algorithm become the Peterson’s original algorithm.

3 Tree-Based Algorithm

In this section we consider a network of arbitrary topology. The proposed algorithm is based on the Raymond’s tree-based distributed mutual exclusion algorithm [6]. The nodes are assumed to be arranged in an unrooted tree structure. The tree may be either a minimal spanning tree of the actual network topology, or merely a logical structure imposed on the underlying physical network. All messages travel along the (undirected) edges of this tree. There is no need for each node to be aware of the tree as a whole. It is sufficient for each node to know of the existence of its neighbors in the tree. When a node initiates a search of the \( k \)-th largest value (we call this node the *initiating node*), it sends a request message to all its neighbors, which temporarily become the children of the requesting node. As the request message propagates through the tree, the tree becomes directed,
rooted by the initiating node. To capture this information, each node has a variable \( parent \), which indicates the location of the initiating node relative to the node itself. Either a node is the initiating node or else it is located in a particular subtree of the node.

Suppose that the node A initiates a search for the second largest value (see Figure 2). In the Figure a node's value is shown beside it's label. The node A sends a request message to its immediate neighbors: nodes B, C, and D. When the nodes B, C, and D receive this request message, they set their \( parent \) variables to A, and the node D propagates the request message to the nodes E and F. Both the nodes E and F set their \( parent \) variables to D. The tree temporarily becomes directed, rooted by the initiating node A, as shown in the figure. On receiving the request message for the second largest value from the node B, both the nodes E and F send their values (1 and 6 respectively) to the node B. When the node B receives the values 1 and 6 from its children, it merges them with its own value (4) to select the two largest values (i.e., 6 and 4) and forward them to the node A, its current parent node. Similarly both the nodes C and D send their values (2 and 5 respectively) to the node A. When the node A receives all the replies from its children, it merges them to find the second largest value (5) of the whole tree. If necessary, the node A can broadcast the found value to all the other nodes in the network.

![Figure 2: Tree with parent-directed edges (The node A initiated the search).](image)

3.1 Algorithm

For simplicity, we assume that at one time only one node initiates the search. It is a simple exercise to extend the following algorithm to allow multiple searches by several initiating nodes concurrently. Assume that we want to find the \( k \)-th largest value, where \( k \) is less than or equal to the number of nodes in the tree. the node initiating the search becomes a temporary root of the tree. Each node computes

\[
\text{parent} := \text{sender}; /\ast\text{sender can be the node itself}*/ \\
n := 0; \\
\text{if} (\text{num}_{\text{neighbor}} = 1) /\ast\text{leaf?} */ \text{then send the message \text{reply}(my_{\text{val}})} \text{to parent} \\
\text{parent} := \text{nil}; \\
\text{else } \text{val}[1] = my_{\text{val}}; \\
n := 1; \\
\text{for each child} \text{send the message \text{find}(k)} \text{to it}; \\
\text{endfor} \\
\text{endif} \\
\]

\[
\text{val}[1..\min(k, n+m)] := \text{merge}(\text{val}[1..n], \ldots, \text{val}[m]); \\
n = \min(k, n+m) \\
\text{if} (\text{num} \text{of received messages} = \text{num} \text{of children}) \text{then if} (\text{parent} = \text{myself}) \\
\text{then found the k-th largest value, val}[k]; \\
\text{else send \text{reply}(val[1], \ldots, val[n]) to parent; } \\
\text{parent} := \text{nil}; \\
\text{endif} \\
\text{endif} \\
\]

![Figure 3: Processing of the message \text{find}(k).](image)

![Figure 4: Processing of the message \text{reply}(n_1, \ldots, n_m)\text{.}]  

the \( k \) largest values in the subtree rooted by itself and passes them to its temporary \( parent \). The the initiating node, after receiving all the replies from its children, can compute the \( k \)-th largest value of the whole tree.

An identical algorithm runs on each node of the tree. The search is initiated when some node sends a request message to itself, which becomes the temporary root. Two kinds of messages are used; \text{find}(k) message and \text{reply}(n_1, n_2, \ldots, n_m) message. The \text{find} message is a request for search and is sent by the parent to its children, and the \text{reply} message is sent the other way around. Thus, we need to consider two cases: (1) when a node receives a \text{find} message and (2) when it receives a \text{reply} message. When a node receives a \text{find}(k) message from itself or one of its immediate neighbors, the algorithm in Figure 3 is executed. If the node is a leaf, then it simply sends its value to the sender; otherwise, it forwards the request message to its children.

When a node receives a \text{reply}(n_1, \ldots, n_m) (m \leq k) message from itself or one of its children, the algo-
rithm in Figure 4 is executed. The node merges the received values and the current values of \(v[i][n]\) and stores them back to \(v[i]\). If the number of messages received so far is equal to the number of children (i.e., received all replies from its children), the node checks whether it is the initiating node. If it is, then the \(k\)-th largest value is found in \(v[i][k]\); otherwise, it sends the new \(v[i][n]\) to \(parent\). The new \(v[i][n]\) is the largest \(n\) values in the subtree rooted by the node. When the \(k\)-th value is found, if necessary, the initiating node can broadcast it to all other nodes in the network.

### 3.2 Complexity

It is clear that only two messages are passed through each edge of the tree. One is the \(find\) message and the other is the \(reply\) message. Thus, the total number of messages required is twice the number of edges in the tree, i.e., \(2(N - 1)\), leading to the complexity of \(O(N)\). This is the lower bound in the sense that for each node at least two messages are required: one for requesting, the other for replying. Therefore, our algorithm is optimal when considering the total number of messages required. How many steps (rounds of message passing) does our algorithm require? It depends on the structure of the tree. If the height of the tree is \(h\), then it needs \(h - 1\) steps to reach the out-most nodes. Thus, on the average the algorithm requires \(O(\log N)\) steps. The best topology for our algorithm is a radiating star formation and the worst topology is a straight line arrangement. In any case, if the tree structure is logically imposed upon the underlying network, then pathological cases (e.g., the straight line formation) can be avoided in favor of trees which approximates a radiating star formation.

### 4 Discussion

We considered the problem of finding \(k\)-th value in the distributed systems. We first extended an existing extrema-finding algorithm [4] to finding the \(k\)-th value in a circular configuration. Later, a tree-based algorithm that requires \(O(\log N)\) rounds and \(O(N)\) messages was devised. The tree-based algorithm has a couple of interesting features. First, when \(k\) is equal to \(N\), they become distributed (merge) sorting algorithms. This is because we have to find \(k\) largest values to find the \(k\)-th value. The extended Peterson algorithm also has this property. Second, the algorithm does not require the uniqueness of node values; even if several processors have the same value, the algorithm still works.

In addition to the above features, the tree-based algorithm has a nice property that even if some node fails the algorithm still works. The parent of the failed node can detect by time-out that a node (or communication link) failure occurs. Once detected, the parent can either ignore the failed node or wait for its recovery. If it is ignored, the final value would be the \(k\)-th value of a partial tree, i.e., the tree excluding the subtree rooted by the failure node. Multiple node failures can be treated in the same way. In the case of recovering, it is possible for the failed node to reconstruct the lost information from node’s neighbors when the node restarts. The recovering node needs to interact with only its immediate neighbors. The procedure will similar to that of Raymond’s [6].

The tree-based algorithm was implemented using both the Berkeley Software Distribution (BSD) Socket Programming Interface and Remote Procedure Call (RPC).

### References


Tech Report: TR94-07
Submission Date: April 27, 1994