Vol. 41 No. 2 Transactions of Information Processing Society of Japan Feb. 2000

Regular Paper

Distributed Algorithms for Leader Election on Partially Ordered Keys

Zixue Cheng† and Qian-Ping Gu†

The leader election problem is a fundamental problem in distributed computing. The classical leader election problem can be considered as finding the processor with the maximum key in a distributed network in which each processor has one key and a total order is defined on the keys. In this paper, we define a generalized leader election problem that finds all the processors with the maximal keys on the basis of a partial order on the keys. We propose two distributed algorithms for the generalized leader election problem. The first algorithm solves the problem on a network by using a spanning tree of the network. The message complexity of the algorithm is $O(mn)$, where m is the number of different keys and n is the number of processors. The time complexity of the algorithm is $O(n)$. The second algorithm solves the problem using a coterie of the n processors. The number of messages exchanged on the coterie is $O(\max\{rn, n^{1.5}\})$, where r is the number of the maximal keys. When the physical network for connecting the n processors is considered, the message and time complexities of the second algorithm are $O(\max\{drn, dn^{1.5}\})$ and $O(d)$, respectively, where d is the diameter of the network.

1. Introduction

The leader election problem is a fundamental problem in distributed computing[1),6),11),13)]. The problem is to find a processor in a distinguishable computational state among a set of initial processors in the same computational state in a distributed network. It can be simplified as finding the maximum key among the n keys held by n processors in the network, where each processor has one key and a total order is defined on the keys. This problem has numerous applications in many distributed control problems such as those that may occur when token-based algorithms are used: when the token is lost or the owner has failed, the remaining processors elect a leader to issue a new token.

In this paper, we consider a generalized leader election problem. Given n processors in a network, assume that each processor has one key and that a partial order is defined on the n keys. The generalized leader election problem is to find all the maximal keys defined by the partial order. Note that the previous algorithms for the classical leader election problem do not work for the generalized problem defined by a partial order that is not linear.

The generalization is motivated by some distributed applications in computer-supported cooperative work and groupware that introduce new distributed problems[3),7),15),18)]. Those applications are realized through the cooperation of persons/processors interconnected by a computer network. Each of the persons/processors has a key characterized by multiple parameters. The value of a parameter of one key can be compared with that of the same parameter of another key, but it cannot be compared with the value of a different parameter. Considering the parameters of a key as the character vector of the key, the linear order on each parameter then defines a partial order of the character vectors of the keys. More specifically, let us consider a group of persons working in a network environment who are requested to make proposals on a subject. Each person makes one proposal and those proposals are evaluated and selected in a distributed manner on the basis of independent multi-parameters. The goal is to select every proposal such that no other proposal is superior to it in all parameters. Finding the maximal character vectors is an example of the generalized leader election problem, and can be applied to distributed problems such as group decision support systems and consensus with partially ordered domain[7),15),18)]. Leader election based on partially ordered keys is also a natural generalization of the classical leader election problem.

A straightforward algorithm for the generalized leader election problem is to have every processor send its key to all the other proces-

† Department of Computer Software, University of Aizu

sors by flooding and then find the maximal received keys. For a network with diameter d and communication link set E, this algorithm has the message complexity $O(n|E|)$ and the time complexity $O(d)$ (see Lynch[11] for example). We propose two distributed algorithms for the generalized leader election problem on asynchronous networks. The first algorithm solves the problem for a network containing n processors by using a spanning tree of the network. The message complexity of the algorithm is $O(mn)$, where m is the number of different keys. The time complexity of the algorithm is $O(n)$. Notice that a spanning tree of $G(V,E)$ can be found by using, for example, the algorithm in Gallager, et al.[5] with message complexity $O(n \log n + |E|)$ and time complexity $O(n \log n)$. The algorithm based on the spanning tree takes fewer messages but more time to find the maximal keys than the straightforward one. The second algorithm solves the problem for n processors by using a coterie of the processors. The number of messages exchanged on the coterie is $O(\max\{rn, n^{1.5}\})$, where r is the number of maximal keys. If the n processors are physically connected by a network of diameter d, the message and time complexities of the second algorithm are $O(\max\{drn, dn^{1.5}\})$ and $O(d)$, respectively. In particular, when the network is a complete graph, the message and time complexities of the second algorithm become $O(\max\{rn, n^{1.5}\})$ and $O(1)$, respectively.

All the above algorithms can be initiated by an arbitrary set of processors, and on the termination of the algorithms, every processor knows all the maximal keys. These properties are important in many applications. For example, in group decision support systems, each processor should know which leader represents it.

In the rest of the paper, Section 2 gives the preliminaries. The algorithms based on a spanning tree and a coterie are given in Sections 3 and 4, respectively. Some further research problems are discussed in the final section.

2. Preliminaries

A distributed asynchronous network is a set of processors connected by bidirectional communication channels. We consider a network with an arbitrary interconnection topology. The network is denoted by an undirected graph $G(V,E)$, where $V = \{p_1, p_2, \ldots, p_n\}$ is the set of processors (called nodes) in the network and E is the set of communication channels (called edges) between the processors. Each node $p_i \in V$ has one key denoted as k_i. There is no centralized controller, shared memory, or global clock in the network. Each processor communicates with others by exchanging messages through the communication channels. Messages can be transmitted independently in both directions on a communication channel and arrive after an unpredictable but finite time delay, without error, and in the FIFO order.

The complexity measures for evaluating the algorithms are adopted from Barbosa[4]. The message complexity of an algorithm is the maximal number of messages sent between neighbors during the computation on all possible topologies of $G(V,E)$ and all possible executions of the algorithm. We assume that the size of a message is $O(\log n)$ bits. To define the time complexity, we assume that the local computation (within each node) takes no time and that communicating one message to one adjacent node takes $O(1)$ time. The time complexity of an algorithm is the number of messages in the longest causal chain of the form "receive a message and send a message as a consequence" occurring in all executions of the algorithm over all possible topologies of $G(V,E)$. Obviously, the time complexity is always bounded by the message complexity. For more details of the complexity measures, readers may refer to Barbora[4] (e.g., pp.81–83).

A partial order $\leq$ on the set $S = \{k_1, \ldots, k_n\}$ is defined so that (1) $k_i \leq k_i$, (2) $k_i \leq k_j$ and $k_j \leq k_i$ imply $k_i = k_j$, and (3) $k_i \leq k_j \leq k_k$ implies $k_i \leq k_k$. For $k_i, k_j \in S$, if $k_i \leq k_j$ or $k_j \leq k_i$ then we say k_i and k_j are *comparable*; otherwise, we say they are *uncomparable* (denoted as $k_i <> k_j$). A key $k_i \in S$ is called *maximal* if $\forall k_j \in S$, $k_i \leq k_j$ implies $k_i = k_j$. For $k_i, k_j \in S$, if $k_i \leq k_j$ and $k_i \neq k_j$ then $k_i < k_j$. In what follows, we also say that k_i is covered by k_j or k_i is smaller than k_j if $k_i < k_j$. For $k_i, k_j \in S$ with $k_i = k_j$ and $i \neq j$, k_i and k_j are considered as the same key. We use m to denote the number of different keys of S ($m \leq n$).

The generalized leader election problem considered in this paper is to find all the maximal keys defined by the partial order $\leq$ on S. We propose two algorithms for this problem. The first one uses a spanning tree of the network to exchange messages. The spanning tree has been used for constructing efficient algorithms for many problems in distributed systems[14),16),19]. We assume that a spanning tree of $G(V, E)$ has been established and that each node knows its neighbors in the spanning tree.

The second algorithm solves the problem on a coterie of the processors in the network. A *coterie* is a class $\mathcal{C} = \{Q_i | Q_i \subseteq V\}$ of subsets of nodes that satisfies the following properties: For any Q_i and Q_j with $i \neq j$, $Q_i \cap Q_j \neq \emptyset$ and $Q_i \not\subseteq Q_j$. The subsets Q_i are called *quorums*. Coteries are logical structures for achieving coordination among processors and have been used in many distributed problems such as mutual exclusion, replica control, and distributed consensus (including leader election)[9),10),12)]. Descriptions of how to construct a coterie can be found, for example, in Agrawal and Jalote[2)], and Maekawa[12)]. We assume that a coterie of V has been established and that each node knows the other nodes in the same quorums.

In both algorithms, initially each node knows only its own key. A set of arbitrary nodes start executing the algorithms. On the termination of the algorithms, each node knows all the maximal keys. The algorithms are described by the template introduced in Barbosa[4)].

3. The Algorithm on the Spanning Tree

Assume that a spanning tree of $G(V, E)$ has been established and that each node knows its neighbors in the spanning tree. The algorithm follows a broadcasting strategy to solve the problem: Every node p_i broadcasts its key k_i over the spanning tree. Node p_i finds the maximal keys among the keys it has received. To reduce the message complexity, when a key k_i is known to be covered at some node of the tree, that node stops broadcasting k_i to its descendants in the tree.

We now give a more detailed outline of the algorithm. Each node $p_i \in V$ broadcasts its key k_i over the spanning tree. Each node that receives k_i sends a message to p_i to acknowledge the receipt. More specifically, each node sends its key to its neighbors to start the broadcasting. For each $p_i \in V$, when p_i receives a key k_l from its neighbor p_j, it compares k_l with the other keys that it has received. If k_l is not covered by any other received key and p_i is not a leaf of the spanning tree, then p_i records (k_l, p_j) and sends k_l to all its neighbors except p_j, from which k_l was received. Otherwise (if either k_l is covered by some received key or p_i is a leaf), p_i stops broadcasting k_l to its descendants and sends a message (k_l, ack) to p_j. For each recorded (k_l, p_j), when p_i receives (k_l, ack) from all its neighbors except p_j, it forwards (k_l, ack) to p_j. When p_i receives (k_i, ack) from all its neighbors, p_i knows that the broadcasting of its key has been completed. We assume that the information identifying the index i of key k_i is sent with k_i during the broadcasting.

Each leaf node p_i of the spanning tree, when its broadcasting is completed, starts to check whether the broadcasting for every node of V has been completed. If so, then each node of V finds the maximal keys among the received keys and terminates its computation.

The algorithm for each node $p_i \in V$ is given in **Fig. 1**. An arbitrary subset V_0 of nodes initiate the computation.

We assume that each node p_i has the following states, and use a variable $state_i$ to represent p_i's current state.

- *idle*: the node has not started the computation.
- *active*: the key of the node is being broadcasted over the spanning tree.
- *wait-terminate*: the broadcasting of the key has been completed and the node is waiting for the global terminate message.
- *terminated*: the whole computation has been completed.

For each node p_i, let N_i be the set of neighbors of p_i in the spanning tree and S_i be the set of keys that p_i has received. Initially, each node p_i is in *idle* state and $S_i = \{k_i\}$. In addition, every p_i employs the following variables:

$term_i$: an integer representing the number of *wait-terminate* messages received from p_i's adjacent nodes ($0 \le term_i \le |N_i|$).

$parent_i^l$: an adjacent node from which p_i received key k_l.

ack_i^l: an integer representing the number of (l, ack) messages received from p_i's adjacent nodes ($0 \le ack_i^l \le |N_i|$).

The following types of messages are used by the algorithm:

- (k_l, l) is a pair of key k_l and node p_l.
- (l, ack) is an acknowledge message to p_l, indicating that broadcasting of k_l has been finished.
- *check-terminate* is a message to check whether every node has entered the *wait-terminate* state.
- *terminate* is a message to announce that every node has entered the *wait-terminate* state.

Theorem 1 The algorithm given in Fig. 1

Algorithm *Leader_Election_on_Tree*:
▷ **Variables:** $state_i = idle$;
$term_i = 0$; $S_i = \{k_i\}$;
for $1 \leq j \leq n$, $parent_i^l = nil$, $ack_i^l = 0$.
▷ **Input:** $msg_i = nil$.
Action if $p_i \in V_0$:
$state_i :=active$;
send (k_i, i) to all $u \in N_i$.
▷ **Input:** $msg_i = (k_l, l)$ from $p_j \in N_i$.
Action:
if $state_i = idle$ **then**
$\{state_i := active$;
send (k_i, i) to all $u \in N_i$;$\}$;
if $\forall k_m \in S_i$, $k_m < k_l$
or $k_m <> k_l$ **then**
$\{$**if** $k_m < k_l$ **then**
$S_i := (S_i \cup \{k_l\}) \setminus \{k_m\}$
else $S_i := S_i \cup \{k_l\}$;
if $|N_i| = 1$ **then**
send (l, ack) to p_j
else $\{$send (k_l, l)
to all $u \in (N_i \setminus \{p_j\})$;
$parent_i^l := p_j$;$\}\}$;
else send (l, ack) to p_j.
▷ **Input:** $msg_i = (l, ack)$ from $p_j \in N_i$.
Action:
$ack_i^l := ack_i^l + 1$;
if $l = i$ **and** $ack_i^i = |N_i|$ **then**
$state_i :=wait\text{-}terminate$;
if $l \neq i$ **and** $ack_i^l = |N_i| - 1$ **then**
send (l, ack) to $parent_i^l$.
▷ **Input:** $msg_i = nil$.
Action when $state_i =wait\text{-}terminate$
and $|N_i| = 1$:
send *check-terminate* to $u \in N_i$.
▷ **Input:** $msg_i =check\text{-}terminate$
from $p_j \in N_i$.
Action:
$term_i := term_i + 1$;
if $term_i = |N_i| - 1$
and $state_i =wait\text{-}terminate$ **then**
send *check-terminate*
to the node of N_i from which
check-terminate is not received;
if $term_i = |N_i|$
and $state_i =wait\text{-}terminate$ **then**
$\{$send *terminate* to all $u \in N_i$;
find all the maximal keys
from those of S_i based on $\leq$;
$state_i := terminated$;$\}$.
▷ **Input:** $msg_i = terminate$ from $p_j \in N_i$.
Action when $state_i \neq terminated$:
send *terminate* to all $u \in N_i \setminus \{p_j\}$;
find all the maximal keys
from those of S_i based on $\leq$;
$state_i := terminated$.

Fig. 1 Algorithm for leader election on a spanning tree.

solves the generalized leader election problem on a network with an arbitrary interconnection topology in $O(mn)$ message complexity and $O(n)$ time complexity, where m is the number of different keys and n is the number of processors.

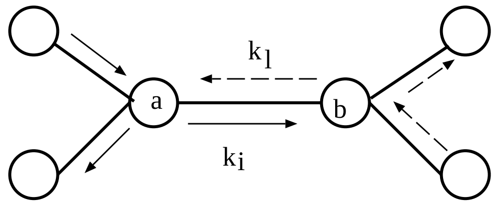

Fig. 2 When $k_i = k_l$, nodes b and a stop broadcasting k_i and k_l to their descendants, respectively.

Proof: First, note that the broadcasting for every node is completed in a finite time. Assume that p_i is a node with key k_i maximal. Then for any key k_l with $k_l \neq k_i$, either $k_l < k_i$ or $k_l <> k_i$. If $k_l \neq k_i$ for all $l \neq i$ then the broadcasting for k_i is completed (the state of p_i becomes *wait-terminate*) only after p_i has received (k_i, ack) from all the leaf nodes of the spanning tree, which implies that every node in the tree has received k_i. Assume that $k_l = k_i$ for some $l \neq i$. Let $V_i = \{p_l | k_l = k_i\}$. The states of all the nodes of V_i then become *wait-terminate* only after every node in the tree has received k_i. It is also easy to check that message *terminate* is broadcast only after the states of all the nodes in the tree have become *wait-terminate*. Therefore, when a node receives the message *terminate*, it has received all the maximal keys. Thus, the algorithm solves the problem correctly.

The message complexity for broadcasting a key k_i that is different from any other key is $O(n)$. For the keys $k_i = k_l$ $(i \neq l)$, let E_i and E_l be the sets of edges of the tree on which k_i and k_l have traveled during the broadcasting, respectively. Then $|E_i \cap E_l| \leq 1$ (see **Fig. 2**). From this, the message complexity for broadcasting the key of the nodes in $V_i = \{p_l | k_l = k_i\}$ is $O(n)$. The message complexity for broadcasting *check-terminate* and *terminate* is $O(n)$. Thus, the message complexity of the algorithm is $O(mn)$, where m is the number of different keys.

Obviously, the number of messages that cause all the nodes in the network to start broadcasting their own keys is at most $n - 1$. That is, the time complexity for all nodes to enter *active* state is at most $n - 1$. For any nodes p_i and p_j, the messages for broadcasting the key of p_i and the messages for broadcasting the key of p_j are sent concurrently. Therefore, it takes at most $2(n - 1)$ time steps from the *active* state to the *wait-terminate* state for all nodes. It is easy to see that to check the global termination of the algorithm also takes $O(n)$ time. Thus, the time complexity of the algorithm is $O(n)$. □

If multiple nodes hold the same maximal key, the algorithm find that key as a leader. More specifically, if p_i and p_l hold the same key (i.e., $k_i = k_l$), and that key is maximal, then all nodes know this maximal key. However, some nodes know that the maximal key comes from node p_i while others know that it comes from p_l. In some applications, we may further need to identify a particular node among those nodes that hold the maximal key. Let k_i be a maximal key and let $V_i = \{p_l | k_l = k_i\}$. We can use the following approach to identify a unique node of V_i. A total order $<$ is defined on V_i such that $p_i < p_j$ if and only if $i < j$. On the basis of the total order, the classical leader election problem on V_i can be defined. On the termination of the algorithm in Fig. 1, the nodes of V_i with k_i as their maximal key identify a unique node by using an algorithm for the classical leader election problem on the set V_i. The message complexity of identifying a unique node of V_i is $O(n)$ on the spanning tree. Therefore, the message complexity of identifying a unique node for every maximal key is $O(rn)$, where r is the number of maximal keys. Notice that $r \leq m$. The time complexity of the above process is $O(n)$.

4. The Algorithm on the Coterie

Several coteries on n processors have been proposed. In this paper, we use the coterie introduced by Agrawal and Jalote [2], which is constructed as follows. Assume that $n = m(m-1)/2$ for some integer m. Create a complete graph K_m with vertices $\{1, 2, \ldots, m\}$ and n edges $\{(i,j) | i, j \in \{1, 2, \ldots, m\}, i \neq j\}$. Perform a one-to-one mapping from the set of n nodes of V to the n edges of K_m. For each vertex i of K_m, let E_i be the set of edges incident to i. A quorum Q_i is defined as the set of nodes that are mapped to the edges in E_i. The coterie is defined as $\mathcal{C} = \{Q_i | i \text{ is a vertex of } K_m\}$. For example, let $V = \{p_1, p_2, \ldots, p_6\}$ and let the one-to-one mapping between V and the set of edges of K_4 be

$(p_1,(1,2)), (p_2,(2,3)), (p_3,(3,4)),$
$(p_4,(4,1)), (p_5,(1,3)), (p_6,(2,4)).$

Then we get the coterie

$$\mathcal{C} = \{\{p_1,p_4,p_5\},\{p_1,p_2,p_6\},\{p_2,p_3,p_5\},\{p_3,p_4,p_6\}\}.$$

Given a coterie $\mathcal{C} = \{Q_1, \ldots, Q_l\}$ of n nodes, for each node p_i, let $I_i = \{j | p_i \in Q_j\}$ and $C_i = \cup_{j \in I_i} Q_j$. For the coterie $\mathcal{C}$ on $V = \{p_1, p_2, p_3, p_4, p_5, p_6\}$ given above,

$$C_1 = \{p_1, p_4, p_6, p_2, p_5\},$$
$$C_2 = \{p_2, p_6, p_3, p_1, p_5\},$$
$$C_3 = \{p_3, p_6, p_2, p_5, p_4\},$$
$$C_4 = \{p_4, p_5, p_3, p_1, p_6\},$$
$$C_5 = \{p_5, p_1, p_2, p_3, p_4\},$$
$$C_6 = \{p_6, p_2, p_3, p_1, p_4\}.$$

We Now show that the coterie employed in our paper indeed satisfies the intersection property. The coterie is created on the basis of a complete graph. For every vertex, a quorum that is a set of nodes mapped to the edges incident to the vertex is created. Since every pair of vertices are incident to a common edge in a complete graph, every pair of quorums have a common node, i.e., $\forall Q_i, Q_j \in C,\ Q_i \cap Q_j \neq \phi$.

Moreover, since a communication set C_i is the union set of quorums to which process p_i belongs, every pair of communication sets C_i and C_j have at least one common node.

Assume that each node p_i knows the nodes of C_i. An outline of the algorithm is as follows. Each node $p_i \in V$ sends its key k_i to the nodes of C_i. (We assume that the information identifying the index i of key k_i is sent with k_i to the nodes of C_i.) For each node p_i, when p_i receives the keys from all the nodes of C_i, p_i finds the maximal keys from the received keys. For each received key k_j, if k_j is maximal then p_i sends the message *uncovered* to p_j; otherwise, p_i sends the message *covered* to p_j. When p_i receives *covered* from some node of C_i, p_i knows that its key k_i is not maximal and p_i enters the *wait-terminate* state. When p_i receives *uncovered* from all the nodes of C_i, p_i knows that its key k_i is maximal and broadcasts k_i to all nodes of V via the coterie. Each node of V sends p_i a message via the coterie to acknowledge receipt of k_i. When p_i has received acknowledgments from all the nodes of V, it enters the *wait-terminate* state.

If there are $k_j, k_l \in S$ with $k_j = k_l$ maximal and $j \neq l$, then the maximal key k_j may be broadcast by nodes p_j and p_l. This is not efficient in the sense of message complexity. We use the following approach to reduce the number of messages. For each node $p_i \in V$ and each key k_j received by p_i, define $V_{ij} = \{p_l | p_l \in C_i, k_l = k_j\}$. If node p_i finds that k_j is maximal among the received keys, it selects only one node of V_{ij} for broadcasting k_j. More precisely, p_i sends *uncovered* to the node of V_{ij} with the largest index and *covered* to all the other nodes of V_{ij}.

For each node $p_i \in V$, when p_i enters the *wait-terminate* state, it starts to check whether all the nodes of V are in the *wait-terminate* state. If so, p_i terminates its computation.

Figure 3 gives the algorithm for each node p_i. To simplify the description of the algorithm, when we say node p_i sends a message to all nodes of C_i, we mean that the message is sent to all the nodes, including p_i itself, of C_i. An arbitrary subset V_0 of nodes initiate the computation. We assume that each node p_i has the following states:

- *idle*: the node has not started the computation.
- *active*: the node is finding the maximal keys.
- *wait-terminate*: the node is waiting for the global terminate message.
- *terminated*: the whole computation has been completed.

The following types of messages are employed in the algorithm. A *covered* message is used to inform p_i that k_i has been covered by some other key. An *uncovered* message is used to inform p_i that k_i has not been covered by any key it has been compared with. In order to broadcast the maximal key k_i, (k_i, max) and (k_i, max, f) are used. (k_i, max) is sent by p_i to all $p_j \in C_i$. The maximal key k_i is forwarded by p_j, using (k_i, max, f), to all the nodes that are not in C_i. In addition, (k_i, ack) is the acknowledgment of (k_i, max, f), and *ack* is the acknowledgment of (k_i, max). In order to detect termination, *check-termination* is used to check whether every process in C_i has entered the *wait-terminate* state, and *termination* is used to announce the termination of the algorithm.

In addition to $state_i$ and S_i, as used in the algorithm in Fig. 1, the following variables are employed. Max_i is a set of maximals known to p_i. Variable max_i is the number of *uncovered* messages received by p_i, $(0 \leq max_i \leq |C_i|)$. If $max_i = |C_i|$, p_i knows that k_i is maximal. The variable $term1_i$ shows the number of *check-terminate* messages received by p_i. $(0 \leq term1_i \leq |C_i|)$. The condition $term1_i = |C_i|$ indicates that every process in C_i has entered the *wait-terminate* state. The variable $term2_i$ shows the number of *terminate* messages received by p_i, $(0 \leq term2_i \leq |C_i|)$. The condition $term2_i = |C_i|$ means that every process in the coterie has entered the *wait-*

Algorithm *Leader_Election_on_Coterie*:
▷ **Variables:** $state_i = idle$; $S_i = \{k_i\}$;
  $Max_i = \emptyset$; $max_i = 0$;
  $term1_i = 0$; $term2_i = 0$; $ack1_i = 0$;
  for $1 \leq j \leq |C_i|$, $ack2_i^j = 0$.
▷ **Input:** $msg_i = nil$.
  Action if $p_i \in V_0$:
    $state_i := active$;
    send k_i to all $u \in C_i$.
▷ **Input:** $msg_i = k_j$ from $p_j \in C_i$.
  Action:
    $S_i := S_i \cup \{k_j\}$;
    if $state_i = idle$ **then**
      $\{state_i := active$;
      send k_i to all $u \in C_i$;$\}$;
    if $|S_i| = |C_i|$ **then**
      {find the maximal keys from S_i;
      $\forall k_j \in S_i$, **if** k_j is maximal
        and $j = \max\{l | p_l \in V_{ij}\}$ **then**
          send p_j *uncovered*
        else send p_j *covered*;}.
▷ **Input:** $msg_i = covered$ from $p_j \in C_i$.
  Action when $state_i = active$:
    $state_i$:=wait-terminate;
    send *check-terminate* to all $u \in C_i$;
▷ **Input:** $msg_i = uncovered$ from $p_j \in C_i$.
  Action when $state_i = active$:
    $max_i := max_i + 1$;
    if $max_i = |C_i|$ **then**
      send (k_i, max) to all $u \in C_i$.
▷ **Input:** $msg_i = (k_j, max)$ from $p_j \in C_i$.
  Action:
    $Max_i := Max_i \cup \{k_j\}$;
    send (k_j, max, f) to all $u \in (C_i \setminus C_j)$.
▷ **Input:** $msg_i = (k_l, max, f)$ from $p_j \in C_i$.
  Action:
    $Max_i := Max_i \cup \{k_l\}$;
    send (k_l, ack) to p_j.
▷ **Input:** $msg_i = (k_l, ack)$ from $p_j \in C_i$.
  Action:
    $ack2_i^l := ack2_i^l + 1$;
    if $ack2_i^l = |C_i \setminus C_l|$ **then**
      send *ack* to p_l.
▷ **Input:** $msg_i = ack$ from $p_j \in C_i$.
  Action:
    $ack1_i := ack1_i + 1$;
    if $ack1_i = |C_i|$ **then**
      $\{state_i := wait\text{-}terminate$;
      send *check-terminate* to
      all $u \in C_i$;$\}$.
▷ **Input:** $msg_i =$*check-terminate*
    from $p_j \in C_i$.
  Action:
    $term1_i := term1_i + 1$;
    if $term1_i = |C_i|$ **then**
      send *terminate* to all $u \in C_i$;
▷ **Input:** $msg_i = terminate$ from $p_j \in C_i$.
  Action:
    $term2_i := term2_i + 1$;
    if $term2_i = |C_i|$ **then**
      $\{state_i := terminated$;
      terminates the computation;}.

Fig. 3 Algorithm for leader election on a coterie.

terminate state. The variable $ack1_i$ shows the number of *ack* messages received by p_i, $(0 \leq ack1_i \leq |C_i|)$. The variable $ack2_i^j$ shows the number of (k_j, ack) messages received by p_i, $(0 \leq ack2_i \leq |C_i \backslash C_j|)$.

Theorem 2 The algorithm in Fig. 3 solves the generalized leader election problem. The number of messages exchanged on the coterie is $O(\max\{rn, n^{1.5}\})$, where r is the number of maximal keys. If the n processors are physically connected by a network of diameter d, the message and time complexities of the algorithm are $O(\max\{drn, dn^{1.5}\})$ and $O(d)$, respectively.

Proof: We first show the correctness of the algorithm. Let p_i and p_j be any two nodes of V. The definition of the coterie guarantees that C_i and C_j have a common node p_k. Since both p_i and p_j send their keys k_i and k_j to p_k, these two keys are compared there. Therefore, the key k_i is compared with all the other keys. After this, if k_i is not covered by any key then it is maximal; otherwise it is not. Let k_j be a maximal key. If for all k_l with $l \neq j$, $k_l \neq k_j$ then node p_j receives only *uncovered* and k_j is broadcast to all nodes of V. If there are keys $k_{j_1}, \ldots, k_{j_l}$ with $k_{j_i} = k_j$ and $j_i \neq j$ $(1 \leq i \leq l)$, then one node (whichever has the largest index) of p_j and $p_{j_1}, \ldots, p_{j_l}$ receives only *uncovered* and k_j is broadcast to all nodes of V. Obviously, all the maximal keys are found and broadcast to every node of V in a finite time.

Each node p_i whose key k_i is maximal, enters the *wait-terminate* state only after all the nodes of V have received k_i. p_i sends a *terminate* message only after all the nodes of C_i have entered the *wait-terminate* state. Therefore, p_i terminates only after all nodes of V have received all the maximal keys. Obviously, the algorithm terminates in a finite time.

The number of messages for determining whether k_i is maximal for all $p_i \in V$ is $O(\sum_{i=1}^{n} |C_i|)$ on the coterie. The number of messages for broadcasting one maximal key k_i is $O(|C_i| + \sum_{p_j \in C_i} |C_j|)$. Assume that $k_1, \ldots, k_r$ are the maximal keys. Then the number of messages for broadcasting all the maximal keys is $O(\sum_{i=1}^{r}(|C_i| + \sum_{p_j \in C_i} |C_j|))$ and number of messages for termination is $O(\sum_{i=1}^{n} |C_i|)$. Using the coterie of Agrawal and Jalote [2], $|C_i| = O(\sqrt{n})$ for $1 \leq i \leq n$.

From this, the number of messages for termination and determining whether k_i is maximal for all $p_i \in V$ is

$$O\left(\sum_{i=1}^{n} |C_i|\right) = O(n^{1.5}).$$

The number of messages for broadcasting r maximal keys is

$$O\left(\sum_{i=1}^{r}\left(|C_i| + \sum_{p_j \in C_i} |C_j|\right)\right) = O(rn).$$

Thus, the number of messages on the coterie is $O(\max\{rn, n^{1.5}\})$, where r is the number of maximal keys.

If n processors are physically connected by a network of diameter d then the message complexity for exchanging one message on the coterie is $O(d)$ and the message complexity of the algorithm becomes $O(\max\{drn, dn^{1.5}\})$.

Since all nodes send their keys (also the *covered* or *uncovered* messages) to the nodes in the communication sets concurrently, there are $O(1)$ messages in the causal chain determining the maximal keys on the coterie. Similarly, there are $O(1)$ messages in the causal chain for broadcasting the maximal keys and detecting the global termination. It takes $O(d)$ time in a real network of diameter d to send one message on the coterie. Therefore, the time complexity of the algorithm is $O(d)$ on a network with diameter d. □

5. Conclusion

In this paper, we have proposed a generalized leader election problem based on partially ordered keys. We showed that this problem can be solved efficiently on a distributed network by using either a spanning tree of the network or a coterie of processors. For the classical leader election problem based on totally ordered keys, the problem can be solved within a message complexity of $O(n \log n)$ on a complete connected network [8] or a logical structure called k-dimensional arrays of the nodes [17]. The transitive property of the linear order is critical for achieving the message complexity bound of $O(n \log n)$. For the partial order $\leq$, two keys may be uncomparable; furthermore, the uncomparable relation $<>$ is not transitive (for example, we do not know the relation between k_i and k_l if we do not compare them, even though the information that $k_i <> k_j$ and $k_j <> k_l$ is known). Because of the property of the relation $<>$, the algorithms of Korach, et al. [8] and Yuan and Agrawala [17] do not work for the generalized leader election problem. Let

r be the number of the maximal keys and m be the number of different keys of the n processors in a network. If $r = m$ then the message complexity $O(mn)$ of our first algorithm is optimal, because it takes $\Omega(rn)$ messages to deliver the r maximal keys to the n nodes. However, whether $O(mn)$ can be reduced further is open for $r < m$. For solving the problem on a logical structure of n processors such as a coterie or k-dimensional array, when $r \geq n^{0.5}$, the number of messages $O(rn)$ of our second algorithm is optimal. Whether $O(n^{1.5})$ can be reduced further for $r < n^{0.5}$ is another open question.

Acknowledgments We would like to express our thanks to the anonymous referees for their critical comments which improve the paper.

References

1) Afek, Y. and Gafni, E.: Time and message bounds for election in synchronous and asynchronous complete networks, *SIAM J. Computing*, Vol.20, No.2, pp.376–394 (1991).
2) Agrawal, G. and Jalote, P.: An efficient protocol for voting in distributed systems, *Proc. 12th IEEE International Conference on Distributed Computing Systems*, pp.640–647 (1992).
3) Barborak, M., Malek, M. and Dahbura, A.: The consensus problem in fault-tolerant computing, *ACM Computing Surveys*, Vol.25, No.2, pp.171–220 (1993).
4) Barbosa, V.C.: *An Introduction to Distributed Algorithms*, MIT Press (1996).
5) Gallager, R.G., Humblet, P.A. and Spira, P.M.: A distributed algorithm for minimum-weight spanning trees, *ACM Trans. Programming Language Systems*, Vol.5, No.1, pp.66–77 (1983).
6) Singh, G.: Leader election in the presence of link failures, *IEEE Trans. Parallel and Distributed Systems*, Vol.7, No.3, pp.231–236 (1996).
7) Ito, T. and Shintani, T.: On a persuasion mechanism among agents for group choice design support systems, *IEICE Trans.*, Vol.J80-D-II, No.9, pp.1–9 (1997).
8) Korach, E., Moran, S. and Zaks, S.: Tight lower and upper bounds for some distributed algorithms for a complete network of processors, *Proc.ACM-PODC '84*, pp.199–207 (1984).
9) Kumar, A.: Hierarchical quorum consensus: A new method for managing replicated data, *IEEE Trans. Computers*, Vol.40, No.9, pp.996–1104 (1991).
10) Lakshman, T.V. and Agrawala, A.K.: Efficient decentralized consensus protocols, *IEEE Trans.Softw.Eng.*, Vol.SE-12, No.5, pp.600–607 (1986).
11) Lynch, N.A.: *Distributed Algorithms*, Morgan Kaufmann (1996).
12) Maekawa, M.: A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems, *ACM Trans. Comput. Syst.*, Vol.3, No.2, pp.145–159 (1995).
13) Masuzawa, T., Nishikawa, N., Hagihara, K. and Tokura, N.: Optimal fault-tolerant distributed algorithms for election in complete networks with a global sense of direction, *Proc. Third Int'l Workshop on Distributed Algorithms* (1989).
14) Raymond, K.: A tree-based algorithm for distributed mutual exclusion, *ACM Trans. Computer Systems*, Vol.7, No.1, pp.61–77 (1989).
15) Shimojo, I., Tachikawa, T. and Takizawa, M.: Distributed consensus protocols with partially ordered domain, Technical Report of IEICE, 97-DPS-83-7, pp.37–42 (1997).
16) Wu, M.M. and Loui, M.C.: An efficient distributed algorithm for maximum matching in general graphs, *Algorithmica*, Vol.5, pp.383–406 (1990).
17) Yuan, S. and Agrawala, A.K.: A class of optimal decentralized commit protocols, *Proc. 8th International Conference on Distributed Computing Systems*, pp.234–241 (1988).
18) Yahata, C. and Takizawa, M.: General protocols for consensus in distributed systems, *Proc. DEXA*, *Lecture Notes in Computer Science*, Vol.978, pp.227–236, Springer-Verlag (1995).
19) Zaks, S.: Optimal distributed algorithms for sorting and ranking, *IEEE Trans. Computers*, Vol.C-34, No.4, pp.376–379 (1985).

(Received May 11, 1999)
(Accepted December 2, 1999)

Zixue Cheng received the M.E. and Ph.D. degrees from Tohoku University in 1990 and 1993, respectively. He was an assistant professor from 1993 to 1999 and has been an associate professor since Apr. 1999, at the Department of Computer Software, University of Aizu. Currently he is working on distributed algorithms, network agents, and distance education. Dr. Cheng is a member of IEEE, ACM, IEICE, and IPSJ.

Qian-Ping Gu received his B.S., M.S., and Ph.D. degrees in computer science from Shandong University, China, Ibaraki University, Japan, and Tohoku University, Japan, in 1982, 1985, and 1988, respectively. He is currently a professor at the Department of Computer Software, the University of Aizu, Japan. He was with the Institute of Software, Chinese Academy of Sciences, Beijing, China. He was a visiting researcher at the Department of Computer and Information Sciences at Ibaraki University in 1990 and the Department of Electrical and Computer Engineering at the University of Calgary, Canada, from 1991 to 1993. His research interests include parallel/distributed processing, network routing and communication algorithms, analysis of algorithms and computation, machine learning, and computational biology. He has published over 30 papers in archival journals in these areas. He is a member of ACM, IEEE, IEEE Computer Society, and IEICE of Japan.