Hindawi Publishing Corporation
EURASIP Journal on Wireless Communications and Networking
Volume 2008, Article ID 720852, 10 pages
doi:10.1155/2008/720852

Research Article

A Stabilizing Algorithm for Clustering of Line Networks

Mehmet Hakan Karaata

Department of Computer Engineering, Kuwait University, P.O. Box 5969, Safat 13060, Kuwait

Correspondence should be addressed to Mehmet Hakan Karaata, karaata@eng.kuniv.edu.kw

Received 27 March 2007; Accepted 25 October 2007

Recommended by Bhaskar Krishnamachari

We present a *stabilizing* algorithm for finding *clustering* of path (line) networks on a distributed model of computation. Clustering is defined as covering of nodes of a network by subpaths (sublines) such that the intersection of any two subpaths (sublines) is at most a single node and the difference between the sizes of the largest and the smallest clusters is minimal. The proposed algorithm evenly partitions the network into nearly the same size clusters and places resources and services for each cluster at its center to minimize the cost of sharing resources and using the services within the cluster. Due to being stabilizing, the algorithm can withstand transient faults and does not require initialization. We expect that this stabilizing algorithm will shed light on stabilizing solutions to the problem for other topologies such as grids, hypercubes, and so on.

1. INTRODUCTION

Path clustering is defined as covering of nodes of a path (line) network by subpaths such that the difference between the sizes of the largest subpath and the smallest subpath is minimal, and the intersection of any two subpaths is at most a single node. Partitioning of computer networks into *clusters* is a fundamental problem with many applications where each cluster is a disjoint subset of the nodes and the links of the network that share the usage of a set of resources and/or services. The resources and the services distributed to clusters may include replicas, databases, tables such as routing tables, and name, mail, web servers, and so on. The distribution of the resources and/or services to clusters reduces the access time, the communication costs, allows the customization of the services provided within each cluster, and eliminates bottlenecks in a distributed system. For instance, the clustering problem is used to model the placement of emergency facilities such as fire stations or hospitals where the aim is to have a minimum guaranteed response time between a client and its facility center.

The problem of clustering is closely related to the graph theoretic problem of p-center and p-median problems, also known as the Min-Max multicenter and Min-Sum multicenter problems, respectively. The problems of finding a p-center of a graph was originated by Hakimi [1–3] and are discussed in a number of papers [4–10]. Although the problem is known to be np-complete for general graphs [1], there are a number of sequential algorithms [11–15] for clustering of trees which are generalizations of paths. In [16], Wang proposes a parallel algorithm for p-centers and r-dominating sets of tree networks.

Stabilizing clustering algorithms for ring and tree networks are available in the literature [17, 18]. In [17], Karaata presents a simple self stabilizing algorithm for the p-centers and r-dominating sets of ring networks. The self stabilizing algorithm of Karaata is capable of withstanding transient faults and changes in the size of the ring network. The problem of ring clustering is less challenging than the problem of path clustering. This is due to the fact that a ring is a regular topology where each process has a right and a left neighbor allowing a relatively simple scheme to be employed, whereas the two endpoints of a path require a special treatment. A stabilizing tree clustering algorithm is presented in [18]. Although this algorithm can be used to obtain clustering of paths, the relative simplicity of the proposed algorithm due to being specially tailored for path networks makes it more extensible to other topologies such as grid, torus, and hypercube networks. Clustering of grid, torus, and hypercube networks is highly desirable for sensor and mobile ad hoc networks.

To the best of our knowledge, no distributed algorithm for path clustering is available in the literature. Although the sequential solution to the problem is relatively easy for a static topology where faults do not exist, the challenge lies in achieving it through local actions without global knowledge in a distributed environment, and in making it resilient to both *transient failures* and topology changes in the form of addition and/or removal of edges and vertices. We view a fault that perturbs the state of the system but not the program as a transient failure.

In this paper, we present a simple *stabilizing* distributed algorithm for path clustering. The proposed algorithm ensures that cluster sizes are ascending from left to right, and the difference between the smallest cluster size and the largest cluster size is at most one in the path network. The proposed solution to the clustering problem for paths also constitutes a solution to the p-center and p-median problems for paths. A stabilizing system guarantees that regardless of the current configuration, the system reaches a legal state in a bounded number of steps and the system state remains legal thereafter. Due to being stabilized, the proposed algorithm can withstand transient failures and can deal with topology changes in a transparent manner.

The paper is organized as follows. Section 2 contains additional motivations, required notations, and the computational model. In Section 3, we present the basis of the proposed algorithm. Section 4 presents the stabilizing algorithm. In Section 5, we provide a correctness proof of the proposed algorithm, a proof of the time complexity bound, and a proof of stabilization of the algorithm. We conclude the paper in Section 6 with some final remarks.

2. PRELIMINARIES

2.1. Motivation

Ad hoc mobile wireless networks consist of a set of identical nodes that move freely and independently, and communicate with other nodes via wireless links. Such networks may be logically represented as a set of *clusters* by grouping together nodes that are in close proximity with one another. Using a distributed clustering algorithm, specific nodes are elected to be *clusterheads*. Consecutively, all nodes within the transmission range of a clusterhead are assigned to the same cluster allowing all nodes in a cluster to communicate with the clusterhead and (possibly) with each other [19, 20]. Clusterheads form a virtual backbone and may be used to route packets for nodes in their clusters [21]. In such networks, the aggregation of nodes into clusters each of which is controlled by a clusterhead provides a convenient framework for the development of important features such as code separation (among clusters), channel access, routing, and bandwidth allocation [19, 22].

In sensor networks, scalability is a major issue since they are expected to operate with up to millions of nodes. This has implications particularly with energy which ideally should not be wasted on sending data to base stations that are potentially far away. Energy waste can be prevented by separating the sensor networks into clusters and nominating nodes that carry out aggregation and forward the data to the base station [23].

In traditional networks, when a data object is accessed from multiple locations in a network, it is often advantageous to replicate the object and disperse the replicas throughout the network [24, 25]. Potential benefits of data replication include increased availability of data, decreased access time, and decreased communications traffic cost. As a result, distributed algorithms for finding clustering of networks are extremely useful. These potential benefits can be realized when the clustering parameters, such as the number and the relative sizes of the clusters, and the placement of the resources and/or services within each cluster are carefully determined.

The key parameter influencing the benefits of the clustering is the relative sizes of the clusters. Observe that if some clusters are relatively small whereas some are relatively large implying that the resources and/or services concentrate in a section of the network, they cannot be utilized as desired and the utilization of the resources and/or services in a fair manner is reduced. Therefore, the cluster sizes should be approximately the same. In addition, in each cluster the resources and/or services should be located at some particular locations (nodes) such as the *center* of the cluster, a location that minimizes the maximum distance to this location in the cluster. Clearly, these choices decrease the access time and the communications traffic cost to the services and resources by minimizing the maximum distance from a node to its closest facility center.

This work is primarily concerned with the placement of resources and/or services in a distributed system. The other aspects of management of resources such as maintaining the consistency of replicas, and the distribution of databases, tables, and services are outside the scope of this work.

In traditional networks, a nonfault tolerant path clustering can be achieved in linear time by collecting the required topology information at a predefined process and computing the path clustering at this process. To make such a protocol fault tolerant, this process has to be repeated at certain time intervals. Unlike traditional networks, clustering cannot readily be performed in this manner in sensor networks due to power and memory limitations. This establishes the viability of the less efficient stabilizing solutions.

Path clustering is an interesting problem since it establishes the basis of solutions to clustering of other topologies such as mesh, torus, hypercube, and star networks. A discussion on how the proposed solution can be used to solve clustering problems for other topologies is out of the scope of this paper.

2.2. Notation

In a distributed system, the ideal location for the placement of resources and/or services within each cluster is often *a center* (or a *median*) of the cluster [2, 3]. For a simple connected graph representing a cluster, the *eccentricity* of a vertex is defined as the largest distance from the vertex to any vertex in the graph, then a vertex with minimum eccentricity is called a center of the cluster. We call a center or a median of each cluster where the resources and/or services are placed as a *clusterhead*. Similar to a token or a mobile agent, clusterhead is a property of nodes such that this property can be transferred from a node to another and a node may possess at most one clusterhead. Each node in G containing a clusterhead is called as a *clusterhead node*, or simply a *clusterhead*. Similarly, a node without a clusterhead will be referred to as a *nonclusterhead node*. The property of being a clusterhead can be transferred by a *clusterhead move* from a node that possesses it to a neighboring node that does not possess the property. A clusterhead move by clusterhead i corresponds to the transfer of the clusterhead contained in process i to a neighboring process.

Consider a connected graph $G = (V, E)$ representing the network of computation, where V is the set of n nodes, and E is the set of e edges. Let the shortest path between nodes $i, j \in V$ be denoted by $d(i, j)$, referred to as the *distance* between nodes $i, j \in V$. We assume that an arbitrary set of nodes in G contain p clusterheads. Let $C \subseteq V$ be a set of *clusterheads* such that $0 \leq |C| \leq |V|$. Two clusterheads $i, j \in C$ connected by a path not containing another clusterhead are referred to as *adjacent* or *neighboring clusterhead nodes*. Define *cluster* C_i, where $i \in C$, as the set of connected nodes that are closer to clusterhead $i \in C$ than any other clusterhead in C, that is, for node $j \in V$, $j \in C_i$ if and only if i is a clusterhead at a minimal distance from node j. (Observe that based on the above definition, if a node is at the same distance from two distinct clusterheads, then it is in two clusters.) The *size* of cluster C_i is the maximum distance between processes $j, k \in C_i$. It is easy to see that set of clusterheads C defines a clustering of G. The two clusters C_i and C_j are referred to as the *neighboring clusters* if and only if i and j are neighboring clusterheads. Observe that p clusterheads in the system, where $0 \leq p \leq n$, partition the graph into p clusters of varying sizes such that each cluster has a clusterhead at its centroid.

The *p-clustering problem*, or simply the *clustering problem*, is defined for path networks as starting from an arbitrary initial clustering defined by p clusterheads, or after a change in the number of clusters or in the topology of the path, covering of nodes in G by subpaths such that the intersection of any two subpaths is at most a single node and the difference between the size of the largest and the smallest cluster is minimal. Therefore, the clear objective of clustering is to ensure that all the clusters are nearly the same size. Observe that when a network is partitioned into p clusters of nearly equal sizes, clusterheads are evenly distributed in the network and vice versa.

The *p-eccentricity* of node $i \in V$ is defined as the distance of i to the nearest clusterhead. Note that the term eccentricity is related to center finding problem, whereas the term p-eccentricity is related to p-center finding problem. Let the *radius*, or *p-radius*, of cluster C_i, $i \in C$, be the largest p-eccentricity of nodes in cluster C_i.

We illustrate the above ideas using the following illustrative example. A path on 9 nodes is given in Figure 1. In the figure, the *p-eccentricity* values, that is, the distance of node i to the closest clusterhead, are given by the nodes. Although the selection is not unique, since the maximum *p-eccentricity* is minimal, nodes $\{2, 5, 8\}$ are identified as the 3-clustering of the path.

2.3. Computational model

Let $G = (V, E)$ be an arbitrary path with node set V and edge set E, where $|V| = n$. We assume that each node i of G with a unique id i is a process. The computational model used is an asynchronous network of processes where each process maintains a set of local variables whose values can be updated only by the process. Moreover, corresponding to each edge of the graph, one bidirectional, noninterfering communication link is assumed. A communication link between a pair of processes i and j consists of two FIFO channels: one for transmitting messages form i to j and one for transmitting messages from j to i. We assume that the proposed algorithm always starts with the processes at the beginning of their programs (i.e., each program starts executing the first line of its program when the system is started) and communication links are empty. As a result, the proposed algorithm is not self stabilizing in the traditional sense of being able to tolerate an arbitrary transient failure and only supports a weaker property of self stabilization. We later relax these assumptions and present a mechanism to make the algorithm self stabilizing in the traditional sense. No common or global memory is shared by the nodal processors for interprocessor communication. Instead the interprocess communication is done by exchanging messages. It is assumed that the network is sufficiently reliable such that there is no process or channel failure during the transmission. We consider a very simple protocol for message communication, where if process A sends a message to a neighbor process B, then the message gets appended at the end of the input buffer of B and this process takes a finite but arbitrary time (due to the transmission delay). Two or more messages arriving simultaneously at an input buffer are ordered arbitrarily and appended to the buffer. A process receives a message by removing it from the corresponding buffer and waits for a message if the buffer is empty. Therefore, the receive primitives are of blocking type, whereas the sent primitives are of nonblocking type.

We assume a simple message format for interprocess communication. A message is a triple and is expressed as

$$(\text{type, id, parameter(s)}), \tag{1}$$

where type may be *DIST*, *BACK*, or *CHEAD* and *CCHEAD* (the functions of which are explained later), id is the process id of the sender or the recipient of the message; the meaning of parameter fields is self explanatory. The parameter field of a message depends on its type field.

We say that a clusterhead is *enabled* if the conditions are satisfied for it to move to a neighboring process, *disabled*, otherwise. We assume the *weak fairness* of processes, that is, if an enabled clusterhead remains enabled, it eventually makes a move. We also assume that each clusterhead takes the decision to move mutually exclusively among its neighbors, that is, while a clusterhead is in the process of deciding whether to move or not, no neighboring clusterhead can take such a decision but can be involved in other activities. In addition, we assume that each local computation is atomic and no transient fault will take place while a local computation is taking place.

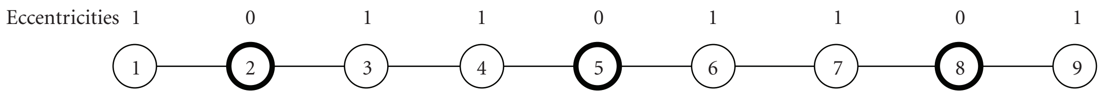

FIGURE 1: The 3-centers of a path on 9 nodes.

The *state* of a process is composed of the set of variables at the process. The *system state* is the Cartesian product of the states of the processes in the system. A *state* of the system is an element of the state space. The system state before the system is started is referred to as the *initial state*.

3. BASIS OF THE ALGORITHM

Prior to formally describing the SPC algorithm, we first present the basis of the algorithm.

The stabilizing path clustering algorithm is based on the following important observation. This is presented in the form of the following lemma, whose proof is straightforward and hence omitted.

Lemma 1. *For any directed path G on n nodes, and any $p \leq n$, there exists a p-clustering such that on each path from the leftmost to the rightmost process, cluster sizes are ascending, and the difference between the smallest cluster size and the largest cluster size is at most one.*

In the path, the arbitrary initial placement of p clusterheads in G decides the initial formation of the clusters. Starting in such an initial state, each clusterhead moves so as to reduce the size of the largest cluster in the network. Prior to making a move, each clusterhead i finds its distance from its left and right neighbors and the difference between the size of the largest cluster and that of cluster i. Consecutively, the clusterhead moves towards a neighboring clusterhead so as to the cluster sizes are ascending from left to right and the difference between the maximum and the minimum cluster sizes is minimal.

These moves ensure that clusterheads move towards the largest cluster to reduce its size without forming new clusters with size larger than that of the largest cluster in the network. In addition, variations between moves towards the right and the left are introduced to ensure that the clusters are sorted in the aforementioned manner.

The above concepts are illustrated with the help of an example in Figure 1. In the example, a path on nine processes is shown. In addition, clusterhead and nonclusterhead processes after entering a stable state are shown. The path contains three clusterhead processes, namely, 2, 5, 8.

If two neighboring clusterheads are allowed to make moves simultaneously, then these moves may undo the effect of each other. Therefore, we assume that each clusterhead moves mutually exclusively in its neighborhood. That is, while a clusterhead moves, no neighbor of the clusterhead can move simultaneously. Though it appears that this is a strong assumption, the mutual exclusion requirement is local with respect to the entire network and can be implemented locally reducing the overhead. The mutual exclusion algorithm ensuring mutually exclusive moves of neighboring clusterheads can readily be adapted from [26], hence omitted. Also observe that the problem is significantly harder (if solvable) without this assumption. In addition, we ensure that after a neighbor moves, the clusterhead has to recompute its distance from its neighbors and then may move again.

To facilitate the description of the algorithm, we introduce several internal functions (variables) that return the current state of the process.

D_R, D_L: $V \rightarrow \{0, \dots, n-1\}$ denote distances of process i from the right and left neighboring clusterheads, and referred to as the D_R-value and D_L-value of process i, respectively.

clusterhead: $V \rightarrow \{\text{true, false}\}$ denotes whether or not process i contains a clusterhead referred to as the clusterhead-value of i.

inc: $V \rightarrow \{0, \dots, n\}$ denotes the difference between D_R-value of i and the D_R-value of the largest cluster in size to the right of clusterhead i. If $inc = 0$ for clusterhead i, then no clusterhead with a larger size exists to the right of clusterhead i. Whereas if $inc = k$ for clusterhead i, then the difference between D_R-value of i and the D_R-value of the largest cluster in size to the right of clusterhead i is k. The *inc*-value of process i is meaningful only when cluster sizes are ascending from clusterhead i to its right. Note that the *inc*-value of each process is calculated through local interactions of processes.

For each clusterhead i, bound$_R$ and bound$_L$ denote whether or not clusterhead i is the rightmost and the leftmost clusterhead, respectively. In addition, $r\_enabled_R$ denotes whether or not the right neighbor of clusterhead i is enabled to make a right move. Clusterhead $i \in C$, where $C \subseteq V$ is the set of clusterheads, makes a right move when $(D_R - D_L > 1) \vee (D_R - D_L = 1 \wedge inc > 1 \wedge \neg\text{bound}_R \wedge \neg\text{bound}_L)$ holds. In addition, clusterhead $i \in C$ makes a left move when $(D_L - D_R > 1) \vee (D_L - D_R = 1 \wedge r\_enabled_R \wedge \neg\text{bound}_R \wedge \neg\text{bound}_L)$ holds. Eventually, it is guaranteed that the set of clusterheads yields a clustering of G.

To facilitate the description of the predicate that holds after the clusterhead moves terminate, we need the following notation and definitions.

Let $C = c_1, c_2, \dots, c_p$, where $p > 1$, be an ordered set $C \subseteq V$ of clusterheads in the system from left to right, where c_{i-1} is the left neighbor c_i for $1 < i \leq p$. Let $D = d_0, d_1, \dots, d_p$ be the sequence of distances between clusterheads such that d_0 denotes the distance of c_1 from the leftmost process, d_p denotes the distance of c_p from the rightmost process, and d_i, where $0 < i < p$, denotes the distance between clusterheads c_i and c_{i+1}, that is, $d_i = d(c_i, c_{i+1})$.

Now, we define predicate P as follows:

$$\begin{aligned}(d_1 = 2d_0 \vee d_1 = 2d_0 + 1 \vee d_1 = 2d_0 - 1)\\ \wedge (d_{p-1} = 2d_p \vee d_{p-1} = 2d_p + 1 \vee d_{p-1} = 2d_p - 1)\\ \wedge \exists_{1<j<p}((d_{j-1} = d_j \vee d_{j-1} = d_j - 1)\\ \wedge \forall_{1\le i<j}(d_1 = d_i) \wedge \forall_{j\le i<p}(d_{p-1} = d_i)).\end{aligned} \quad (2)$$

The first conjunction of predicate P describes the relationship between the distance of the left endpoint of the line from the leftmost clusterhead and distance d_1 between the leftmost clusterhead and its right neighboring clusterhead. Similarly, the second conjunction describes the relationship between distance d_p of the right endpoint of the line from the rightmost clusterhead and the distance d_{p-1} between the rightmost clusterhead and its left neighboring clusterhead. The third conjunction describes the relationships of distances between neighboring clusterheads such that these distances between any two clusterheads (or a clusterhead and an endpoint) differ at most by one and distances are ascending from left to right.

Notice that P is a predicate description of the system state in terms of the distance between two neighboring clusterheads or a clusterhead and an endpoint of the line satisfying the conditions mentioned in the statement of Lemma 1 that a solution to the p-clustering problem satisfies. Recall that this condition states that cluster sizes of processes are ascending from left to right and cluster sizes between any two processes differ by at most one.

When the above predicate is satisfied, the path clustering is obtained by the algorithm. This is presented in the form of the following lemma whose proof is omitted.

Lemma 2. *If predicate P holds, C is a path clustering.*

4. ALGORITHM

This section presents the stabilizing algorithm for clustering of a path implementing the strategy described above.

We use the following notation: R, L denote right and left neighbors of process i, respectively. j_O denotes the other neighbor of i. That is, if j is the left neighbor of i, j_O denotes the right; otherwise, it denoted the left neighbor of i. $update$, $update_R$, and $update_L$ denote atomic operations updating the variables of clusterheads i, its right and left neighboring clusterheads, respectively. In addition, $r\_enabled_R$ denote the right enabledness status of the right neighbor of clusterhead i, that is, whether or not the right neighbor of clusterhead i is enabled to make a right move. The implementations of these are not given for the sake of brevity. However, it is easy to see that these functions and predicates can be implemented through a message exchange between clusterhead i and its neighbors in mutual exclusion implemented by primitives **begin**(*mutex*) and **end**(*mutex*) in the proposed algorithm.

The messages of the algorithm are described as follows.

DIST: two DIST messages are sent by process i to its right and left neighbors to find process i's distance from the processes with clusterhead on its right and on its left.

BACK: when a clusterhead process receives a DIST message, it sends a BACK message with a zero distance value in the reverse direction. Until the BACK message reaches a clusterhead process, each time it encounters a nonclusterhead process, it increments its distance value. Therefore, when the BACK message reaches a clusterhead process, its distance value denotes the distance from the clusterhead on the direction from where the BACK message is received.

CHEAD: the CHEAD message is sent by a clusterhead process i to a neighboring nonclusterhead process j to transfer the clusterhead from process i to process j.

The stabilizing algorithm, called SPC algorithm, for finding the clustering in a path is given in Algorithm 1. Notice that the algorithm is uniform, that is, each process executes the same algorithm.

5. CORRECTNESS

Now, we show that algorithm SPC is stabilizing.

Let PRE be a predicate defined over SYS, the set of global states of the system. An algorithm ALG running on SYS is said to be *stabilizing* with respect to PRE if it satisfies the following.

Closure: if a global state q satisfies PRE, then any global state that is reachable from q using algorithm ALG also satisfies PRE.

Convergence: starting from an arbitrary global state, the distributed system SYS is guaranteed to reach a global state satisfying PRE in a finite number of steps of ALG.

Global states satisfying PRE are said to be *stable*. Similarly, a global state that does not satisfy PRE is referred to as an *instable state*. To show that an algorithm is stabilizing with respect to PRE, we need to show the satisfiability of both closure and convergence conditions. In addition, to show that an algorithm solves a certain problem, we need to either prove *partial correctness* or show that through transitions made by the algorithm among stable states the problem is solved.

We now show that algorithm SPC is stabilizing by establishing the convergence and the closure properties.

Following lemmas establish the convergence of the algorithm.

Lemma 3. *If the clusterheads eventually stop moving, after the clusterheads stop, predicate P (defined in Section 3) is satisfied.*

Proof (contradiction). Assume the contrary, that is, no right or left moves are enabled; however, P is false. Then, we know that one of $(d_1 = 2\times d_0 \vee d_1 = 2\times d_0 + 1)$, $(d_{p-1} = 2\times d_p \vee d_{p-1} = 2\times d_p - 1)$, or $\exists_{1<j<p} j((d_{j-1} = d_j \vee d_{j-1} = d_j + 1) \wedge \text{for all}_{1\le i<j} i(d_1 = d_i) \wedge \text{for all}_{j\le i<p} i(d_{p-1} = d_i))$ is false. Now, we consider each one of the above cases.

Case 1. $(d_1 = 2\times d_0 \vee d_1 = 2\times d_0 + 1)$ is false. Then, clearly, $d_1 \ne 2\times d_0$ and $d_1 \ne 2\times d_0 + 1$ hold. It is easy to see that c_1 is enabled to make a move. This is a contradiction.

Case 2. $(d_{p-1} = 2\times d_p \vee d_{p-1} = 2\times d_p - 1)$ is false. Then, clearly, $(d_{p-1} \ne 2\times d_p)$ and $(d_{p-1} \ne 2\times d_p - 1)$ hold. We know that c_p is enabled to make a move. This a contradiction.

```
[Program for each clusterhead $i \in V$]
while (clusterhead($i$))
do {
    begin(mutex)
    send(DIST, $R$);
    send(DIST, $L$);
    receive(BACK, $R$, $D_R$, $inc_R$, bound$_R$);
    receive(BACK, $L$, $D_L$, $inc_L$, bound$_L$);
    if (bound$_R$)
      then { $D_R := D_R * 2$;
             $inc := 0$;
      else { if ($D_R > D_L$)
               then {$inc := inc_R + 1$;
               else { if ($D_R = D_L \wedge inc_R$)
                        then {$inc := inc_R$;
                        else {$inc := 0$;
    if (bound$_L$)
      then {$D_L := D_L * 2$;
    if $(D_R - D_L > 1) \vee (D_R - D_L = 1 \wedge inc > 1 \wedge \neg$bound$_R \wedge \neg$bound$_L)$
      then { send(CHEAD, $R$);
             clusterhead($i$) := false;
      else { if $(D_L - D_R > 1) \vee (D_L - D_R = 1 \wedge \neg r\_enabled_R \wedge \neg$bound$_R \wedge \neg$bound$_L)$
               then { send(CHEAD, $L$);
                      clusterhead($i$) := false;
    $update$;
    $update_R$;
    $update_L$;
    end(mutex)

Upon receipt of a DIST message from j
receive(DIST, $j$);
send(BACK, $j$, 0, $inc$, false);
[Program for each nonclusterhead process $i \in V$]
Upon receipt of a DIST message from j
receive(DIST, $j$);
if ($j$ is not a boundary process)
  then {send(DIST,$j_O$);
  else {send(BACK, $j$, 0, false, true);
Upon receipt of an BACK message from j
receive(BACK, $j$, $d$, $inc$, bound);
send(BACK, $j_O$, $d + 1$, $inc$, bound);
Upon receipt of a CHEAD message from j
receive(CHEAD, $j$);
clusterhead($i$) := true;
```

Algorithm 1

Case 3. $\exists_{1<j<p} j((d_{j-1} = d_j \vee d_{j-1} = d_j + 1) \wedge$ for all$_{1\le i<j} i(d_1 = d_i) \wedge$ for all$_{j\le i<p} i(d_{p-1} = d_i))$ is false. Then, we know that either $\exists_{1<j<p} j \exists_{j<k<p} k(d_{j-1} = d_j - 1 \wedge d_{k-1} = d_k - 1)$ or $\exists_{1<j<p}(d_{j-1} \ne d_j \wedge (d_{j-1} < d_j - 1 \vee d_{j-1} > d_j))$ holds. If $\exists_{1<j<p} j \exists_{j<k<p} k(d_{j-1} = d_j - 1 \wedge d_{k-1} = d_k - 1)$ holds, since no move (right or left) is enabled and $\exists_{1<j<p} j \exists_{j<k<p} k(d_{j-1} = d_j - 1 \wedge d_{k-1} = d_k - 1)$ holds, we know that for clusterhead c_{j-1}, $D_R - D_L = 1 \wedge$ inc$_R$ holds and c_{j-1} is enabled to make

a right move. This is a contradiction. If $\exists_{1<j<p}(d_{j-1} \neq d_j \wedge (d_{j-1} < d_j - 1 \vee d_{j-1} > d_j))$ holds, since no action is enabled, $\exists_{1<j<p}(d_{j-1} \neq d_j \wedge (d_{j-1} < d_j - 1 \vee d_{j-1} > d_j))$ holds, either c_{j-1} is enabled to make a move. This is a contradiction. Hence, the proof follows. □

We now show the partial correctness of the proposed algorithm.

Lemma 4 (partial correctness). *If the clusterheads eventually stop moving, after the termination of the clusterhead moves, the set of clusterhead processes C is a clustering of G, where p is the number of clusterheads.*

Proof. The proof immediately follows from Lemmas 2 and 3. □

Now, we present the worst case time complexity or the upper bound of the SPC algorithm. We first classify the moves of the algorithm into two categories called *initial moves* and *noninitial moves*. The initial moves are the ones that are caused by arbitrary initialization and the noninitial moves are the ones that are caused by other moves in the system. We categorize each move by a clusterhead as an initial move or as a noninitial move as follows.

Clusterhead move M_x by clusterhead i is a noninitial move if there exits a move M_y by clusterhead j adjacent to clusterhead i such that move M_y happens before move M_x, moves M_y and M_x are in the same direction, and move M_x is not *enabled* to make a move in this direction prior to move M_y or $|D_R.i - D_L.i|$ is increased. Otherwise, a move is referred to as an initial move. We say that a move is enabled if the conditions are satisfied for the move to take place. Let M_y be the last such move. Move M_y is referred to as the cause of move M_x.

An execution in a distributed system can be described as a sequence of moves $M_1, M_2 \ldots$, where M_j for $j > 0$ is a move made by a process in the system. Consider a clusterhead move M_x by an arbitrary process i. We identify a unique clusterhead i of process i and a unique move M_y where $l < k$, by process j to be the "cause" of move M_x defined as follows.

Define cause() for initial moves:

(i) cause$(M_x) = M_x$ if M_x is an initial clusterhead move.

Define cause() for noninitial moves:

(ii) cause$(M_x) = M_y$ if M_y is a move such that move M_y happens before move M_x, moves M_y and M_x are in the same direction, and move M_x is not *enabled* prior to move M_y.

We now state several useful properties related to the function cause(). The first property is that distinct moves by a process have distinct causes.

Proposition 1. *If M_p and M_q are distinct moves by process i, then*

$$\text{cause}(M_p) \neq \text{cause}(M_q). \tag{3}$$

Proposition 2. *Let M_x be a clusterhead move by clusterhead i. If M_x is not an initial move by clusterhead i, then the clusterhead move cause(M_x) is not made by clusterhead i.*

The next property that we establish is that the cause relationship is "acyclic." The following proposition follows from the definition of cause.

Proposition 3. *Let M_x be a clusterhead move by clusterhead i. If M_x is not an initial move by clusterhead i, then the move cause(cause(M_x)) is not made by clusterhead i.*

Proposition 4. *Each clusterhead $i \in C$ can make initial moves only in one direction (right or left).*

Proof. The proof immediately follows from the definition of initial moves. □

We now show the upper bound on the number of initial moves.

Lemma 5. *The total number of initial moves in the system is at most n.*

Proof. By Proposition 4, we know that initial moves by a clusterhead can be in one direction. We also know that clusterhead i can make at most $|d(i,j) - d(i,k)|$ initial moves, where i_R and i_L denote the right and the left neighboring clusterheads of clusterhead i, respectively. It is easy to see that $\sum_{i\in T}|d(i,i_R) - d(i,i_L)| \leq n$. Hence, the proof follows. □

We know that each move is either an initial move or it has a source which is an initial move such that this initial move causes the move through a causal chain of noninitial moves. The following lemmas show the upper bound on the number of noninitial moves possible with the same initial source move.

We need the following definitions to facilitate the following proofs.

If a clusterhead move by clusterhead i is caused by a neighboring clusterhead move, such a clusterhead move by clusterhead i is referred to as a *type I* clusterhead move. Otherwise, a clusterhead move is referred to as a *type II* clusterhead move.

Observe that type I moves are caused by neighboring clusterhead moves changing D_R and/or D_L of i, whereas type II moves are caused by the right neighbor changing its *inc* variable.

Lemma 6. *An initial right clusterhead move can be the source of at most $p - 1$ right clusterhead moves.*

Proof. We first show that a right clusterhead move by clusterhead $i \in C$ can only be caused by a right clusterhead move of type I by its right or left neighbor, or a right clusterhead move of type II by a clusterhead to the right of clusterhead i.

We know that a right clusterhead is caused by another move by changing either the distance variables D_L and D_R, or the *inc* variable for clusterhead i.

We first consider those clusterhead moves that change D_R and D_L of i. Observe that if $D_R = D_L \wedge inc > 1$ hold for clusterhead i, after a right clusterhead move by the left neighbor of i, we have $D_R - D_L = 1 \wedge inc > 1$ for i. Also observe that if $D_R = D_L$ holds for i and $D_R > D_L + 1$ holds for the right neighbor j of i, after the right move by j, i is enabled to make a right move. Therefore, i 's right or left neighbor's right clusterhead move may cause a right move of type I by clusterhead i.

Also observe that a left move by the right or the left neighbor of i cannot cause a right move by a clusterhead by changing $D_R - D_L$ since $D_R - D_L$ is decreased by each such move.

Now, we consider those clusterhead moves that change *inc* value of i possibly through other moves changing the *inc* variables of clusterheads.

It is easy to see that a right move by clusterhead i can be caused only when the *inc* variable of its right neighbor increases when $D_R - D_L = 1 \wedge inc \leq 2$ hold for clusterhead i.

The cause of such a move can be an initial move by a clusterhead updating its *inc* variable to the right of clusterhead i. Otherwise, it is caused by a clusterhead move by a clusterhead to the right of clusterhead i. Clearly, a left move by a clusterhead to the right of clusterhead i cannot be the cause of increasing the *inc* value of i. Then, it can be caused by a right clusterhead move by a clusterhead to the right of clusterhead i. It is easy to see that if clusterhead i_1 is the right neighbor of i and clusterhead i_2 is the right neighbor of i_1, and so on, the D_R values for the sequence of clusterhead $i, i_1, i_2, \ldots, i_k, i_{k+1}, i_{k+2}$ should be such that $D_R - 1, D_R, D_R, \ldots, D_R - 1, D_R + c$, where $c > 1$ and the *inc* values for clusterheads i_1 through i_{k+1}, has to be 0. Clearly, a right clusterhead move by clusterhead i_{k+2} can be the cause of a right clusterhead move by clusterhead i and there is no other right move that can be the cause of such a move by clusterhead i. We showed that a right clusterhead move by clusterhead i can cause a right clusterhead move of type II by a clusterhead to the left of clusterhead i (not the left neighbor of clusterhead i).

It is easy to see that a type I right clusterhead move cannot cause a type II clusterhead move through other type I moves. In addition, from Proposition 3 we know that a clusterhead move by clusterhead i caused by a clusterhead move by a neighboring clusterhead j cannot in turn cause a clusterhead move by clusterhead j. Furthermore, we know that a clusterhead move cannot be the cause of another clusterhead move by the same clusterhead by Proposition 2. Therefore, a right clusterhead move can be the source of at most $p - 1$ right clusterhead moves. Hence, the proof follows. □

Lemma 7. *An initial left clusterhead move can be the source of at most $p - 1$ left clusterhead moves.*

Proof. We first show that a left clusterhead move by clusterhead $i \in C$ can only be caused by a left clusterhead move of type I by the left or a right neighbor of i, or a left clusterhead move of type II to the right of clusterhead i. We now consider these two cases.

Case 1. The move is caused by a type I move by a neighbor.

We know that such a clusterhead move by i can be triggered by another clusterhead move by changing the distance variables of i. Observe that if $D_L = D_R \wedge \neg r\_enabled_R$ holds for clusterhead i, after a left clusterhead move by the left neighbor of i, we have $D_L - D_R = 1 \wedge \neg r\_enabled_R$ holds for i. Similarly, if $D_L = D_R$ holds for clusterhead i, after a left clusterhead move by the right neighbor of i, we may have $D_L - D_R = 1 \wedge \neg r\_enabled_R$ for i. Therefore, i 's left or right neighbor's left move can cause a left move by clusterhead i.

Since a right clusterhead move by a neighbor (right or left) of i decreases $D_L - D_R$, such a move does not cause a left move of type I by clusterhead i.

In addition, by Proposition 2, we know that a move by clusterhead i cannot cause a clusterhead move by i. Therefore, a left move of type I by i can be caused by only a left (but not right) move of only the right or left neighbors of i

Case 2. The move of type II by i can be caused by a clusterhead move by a clusterhead to the right of clusterhead i.

We know that clusterhead i makes a type II left move only when predicate $r\_enabled_R$ changes from true to false. We also know that this can only take place if the *inc* value of the right neighbor j of i decreases.

Now, we consider those clusterhead moves that change *inc* value of j possibly through other moves changing the *inc* values of clusterheads.

It is easy to see that a right move by clusterhead j can be caused only when the *inc* variable of its right neighbor decreases when $D_L - D_R = 1 \wedge inc = 2$ holds for clusterhead j. Observe that a change of *inc* value of clusterhead moves and the changes of *inc* source of the clusterhead move by i can be an initial move by a clusterhead updating its *inc* value to the right of clusterhead j. This *inc* value change may trigger *inc* value changes in the left direction eventually reducing *inc* of j and triggering a left move by clusterhead i.

If the *inc* values are up-to-date to the right of clusterhead i, then it can be shown that a left clusterhead move may be the source of *inc* value changes propagating to the left and causing a left clusterhead move.

It can readily be shown that a left move by clusterhead i can cause a type II clusterhead move by its left neighbor k, however, the move by k cannot in turn cause a left move of type I by a clusterhead to its left. From the above discussion and Propositions 2 and 3, we know that the causal relationship is acyclic.

Since a left clusterhead move by i can be caused by a left clusterhead move by one of the neighbors or a type II left clusterhead move to the right of clusterhead i and the causal relationship is acyclic, the proof follows. □

The following lemma establishes the termination of the algorithm.

Lemma 8 (termination). *Algorithm SPC terminates after $O(np)$ clusterhead moves.*

Proof. It is easy to see that the proof follows from Lemmas 5, 6, and 7. □

As a consequence of Lemma 4 and Lemma 8, we now establish the total correctness of our algorithm.

Lemma 9 (total correctness). *Algorithm SPC identifies clustering of G after $O(np)$ moves.*

Lemma 10 (alternate proof of termination). *Algorithm SPC eventually terminates.*

Proof. We know that no clusterhead can move to the left infinitely many times without making a right move. Therefore, if the algorithm does not terminate, then we have at least one clusterhead that makes infinitely many moves in both directions (right and left). Let c_i be a clusterhead that moves in both directions infinitely many times. By Propositions 1 and 2, we know that a neighbor c_j of c_i also moves in both directions infinitely many times. Without loss of generality, let c_j be the right neighbor of clusterhead c_i, that is, $c_j = c_{(i+1)\mathrm{mod}_p}$. By Proposition 3, we have that in order for clusterhead $c_{(i+1)\mathrm{mod}_p}$ to make infinitely many moves in both directions, clusterhead $c_{(i+2)\mathrm{mod}_p}$ has to make infinitely many moves in both directions. Then, it can be shown inductively that c_p also makes infinitely many moves in both directions. This is a contradiction. Hence, the proof follows. □

By Lemmas 4 and 10, we know that predicate P is eventually satisfied establishing the convergence property. In addition, we know that eventually no taken is enabled by Lemma 10. Therefore, the closure property is trivially satisfied. Hence, algorithm SPC is stabilizing. From the above discussion, Lemmas 4 and 9, we have the following lemma.

Lemma 11. *Algorithm SPC is stabilizing and it identifies a clustering of G after $O(np)$ moves.*

6. CONCLUSIONS

On a distributed or network model of computation, we have presented a self stabilizing algorithm that identifies the p-clusterings of a path. We expect that this distributed and self stabilizing algorithm will shed light on distributed and self stabilizing solutions to the problem for other topologies such as grids, hypercubes, and so on. Solutions to the problem for these topologies have a wide range of applications in parallel, mobile, and distributed computing applications requiring location management. In addition, clustering of sensor networks yields a number of desirable properties.

We assumed that the proposed algorithm always starts with the processes at the beginning of their programs, and communication links are empty. This implicitly brings that the only self stabilization provided by the algorithm is with respect to the initial placement of the clusterheads. In addition, this might have some effect on the claims about dynamicity of the protocol, since dynamic changes need to be only allowed at certain safe states of the algorithm.

The proposed algorithm is unable to cope with arbitrary transient faults and topology changes primarily since a clusterhead may not receive the message it expects as a result of a transient fault or a topology change and cannot proceed with the appropriate actions. This problem can be alleviated by employing a timeout mechanism for the received primitive executed by the clusterhead nodes under appropriate synchronization assumptions. That is, when a clusterhead node does not receive all expected messages, it issues a timeout and executes its program from its beginning. In addition, this approach eliminates the need to have background processes maintaining p clusterheads in the system and allows the number of clusterheads to increase and decrease.

A slightly modified version of the proposed algorithm can work for ring networks but not vice versa. Therefore, the ring clustering algorithm can be viewed as a special case of the path clustering algorithm. Furthermore, although the ring clustering algorithm allows the ring size to change, it does not allow a link to be broken. Whereas the proposed algorithm allows the path to be split into subpaths and finds the clustering of the subpaths.

ACKNOWLEDGMENTS

The author would like to thank the anonymous referees for their suggestions and constructive comments on an earlier version of the paper. Their suggestions have greatly enhanced the readability of the paper.

REFERENCES

[1] O. Kariv and S. L. Hakimi, “Algorithmic approach to network location problems I: the p-centers,” *SIAM Journal on Applied Mathematics*, vol. 37, no. 3, pp. 513–538, 1979.

[2] S. L. Hakimi, “Optimal distribution of switching centers in a communication network and some related problems,” *Operations Research*, vol. 13, pp. 462–475, 1965.

[3] S. L. Hakimi, “Optimum locations of switching centers and the absulate centers and medians of a graph,” *Operation Research*, vol. 12, pp. 450–459, 1964.

[4] E. Korach, D. Rotem, and N. Santoro, “Distributed algorithms for finding centers and medians in networks,” *ACM Transactions on Programming Languages and Systems*, vol. 6, no. 3, pp. 380–401, 1984.

[5] P. M. Dearing and R. L. Francis, “A minimax location problem on a network,” *Transportation Science*, vol. 8, no. 4, pp. 333–343, 1974.

[6] G. Y. Handler, “Minimax network location theory and algorithms,” Tech. Rep. 107, Flight Transportation Laboratory, Massachussetts Institute of Technology, Cambridge, Mass, USA, 1974.

[7] G. Y. Handler, “Minimax location of a facility in an undirected tree graph,” *Transportation Science*, vol. 7, pp. 287–293, 1973.

[8] S. Halfin, “On fnding the absolute and vertex centers of a tree with distances,” *India Business Insight Database*, vol. 8, pp. 75–77, 1974.

[9] S. L. Hakimi, E. F. Schmeichel, and J. G. Pierce, “On p-Centers in networks,” *Transportation Science*, vol. 12, no. 1, pp. 1–15, 1978.

[10] A. J. Goldman, “Minimax location of a facility in a network,” *Transportation Science*, vol. 6, no. 4, pp. 407–418, 1972.

[11] M. Nesterenko and A. Arora, “Stabilization-preserving atomicity refinement,” in *Proceedings of the 13th International Symposium on Distributed Computing (DISC ’99)*, pp. 254–268, Springer, Bratislava, Slovak Republic, September 1999.

[12] G. N. Frederickson, “Parametric search and locating supply centers in trees,” in *Algorithms and Data Structures, 2nd Workshop (WADS ’91)*, F. Dehne, J.-R. Sack, and N. Santoro, Eds., vol. 519 of *Lecture Notes in Computer Science*, pp. 299–319, Springer, Ottawa, Canada, August 1991.

[13] T. Sheltami and H. T. Mouftah, “Clusterhead controlled token for virtual base station on-demand in manets,” in *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS ’03) Workshop in Mobile and Wireless Networks (MWN)*, pp. 716–721, Phoenix, Ariz, USA, April 2003.

[14] E. Erkut, R. Francis, and A. Tamir, "Distance-constrained multifacility minimax location problems on tree networks," *Networks: An International Journal*, vol. 22, pp. 37–54, 1992.

[15] A. Tamir, "Improved complexity bounds for center location problems on networks by using dynamic data structures," *SIAM Journal on Discrete Mathematics*, vol. 1, no. 3, pp. 377–396, 1988.

[16] A. Tamir, D. Pérez-Brito, and J. Moreno-Pérez, "A polynomial algorithm for the p-centdian problem on a tree," *Networks*, vol. 32, no. 4, pp. 255–262, 1998.

[17] M. H. Karaata, "Stabilizing ring clustering," *Journal of Systems Architecture*, vol. 50, no. 10, pp. 623–634, 2004.

[18] M. H. Karaata, "Self-stabilizing clustering of tree networks," *IEEE Transactions on Computers*, vol. 55, no. 4, pp. 416–427, 2006.

[19] C. Chiang, H. Wu, W. Liu, and M. Gerla, "Routing in clustered multihop, mobile wireless networks," in *Proceedings of the IEEE Singapore International Conference on Networks (SICON '97)*, pp. 197–211, April 1997.

[20] T. R. L. Francis, "Convex location problems on tree networks," *Operations Research*, vol. 37, 1976.

[21] A. D. Amis, R. Prakash, D. Huynh, and T. Vuong, "Max-min d-cluster formation in wireless ad hoc networks," in *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '00)*, vol. 1, pp. 32–41, Tel Aviv, Israel, March 2000.

[22] E. J. Coyle and S. Bandyopadhyay, "An energy efficient hierarchical clustering algorithm for wireless sensor networks," in *Proceedings of the IEEE INFOCOM*, vol. 3, pp. 1713–1723, 2003.

[23] E. Royer and C. Toh, "A review of current routing protocols for ad hoc mobile wireless networks," *IEEE Personal Communications*, vol. 6, no. 2, pp. 46–55, 1999.

[24] S. A. Cook, J. Pachl, and I. S. Pressman, "The optimal location of replicas in a network using a read-one-write-all policy," *Distributed Computing*, vol. 15, no. 1, pp. 57–66, 2002.

[25] H. Garcia-Molina, "The future of data replication," in *Proceedings of the IEEE Symposium on Reliability in Distributed Software and Database Systems*, pp. 13–19, Los Angeles, Calif, USA, January 1986.

[26] E. Minieka, "The m-center problem," *SIAM Review*, vol. 12, no. 1, pp. 138–139, 1970.