Parallel Processing Letters,

SELF-STABILIZING ALGORITHMS FOR TREE METRICS

AJOY K. DATTA
Department of Computer Science, University of Nevada
Las Vegas, Nevada 89154, USA
E-mail: datta@cs.unlv.edu

TEOFILO F. GONZALEZ
Department of Computer Science, University of California
Santa Barbara, CA 93106, USA
E-mail: teo@cs.ucsb.edu

VISALAKSHI THIAGARAJAN
Department of Computer Science, University of Nevada
Las Vegas, Nevada 89154, USA
E-mail: visa@cs.unlv.edu

Received August 1995
Revised November 1997
Communicated by Sajal K. Das

ABSTRACT

We present algorithms for finding the diameter, centroid(s), and median(s) for tree structured networks subject to transient faults. In our solutions, the system reaches its final correct configuration in a finite time after the faults cease. The fault-tolerance is achieved using Dijkstra's paradigm of self-stabilization. A self-stabilizing algorithm, regardless of the initial system configuration, converges, in finite time, to a set of *legitimate* configurations.

Keywords: Distributed algorithms, diameter, centroid, median, self-stabilization.

1. Introduction

Topological information, such as location of centroid and median, plays an important role in distributed systems. This information is used for dynamic routing of messages among nodes. But it cannot be taken into account once and for all at design time since several unpredictable factors make it time varying. The problem of dynamically finding the diameter and locating centroids and medians of a tree structured network therefore assumes importance. This paper presents protocols for finding the diameter and locating centroids and medians of a dynamic tree network. The solutions presented require only local topological knowledge at each node, and are self-stabilizing [1,2]. The self-stabilizing algorithm terminates after it computes the metrics, but any unexpected perturbation reactivates it, and possibly new values for the metrics are computed if there are changes in the network topology. The

model assumes that there are n nodes $1, ..., n$ arranged in a tree configuration, 1 being the root. The tree is maintained by a self-stabilizing spanning tree protocol over a graph, thus making the model more general. Work has been done by Karaata et al [3] in this area. They require that each action has a very large atomicity whereas we have no such requirement. Also, every node in the network knows the identity of the centroid and median of the network when our protocol terminates, thus making it an ideal underlying protocol for routing purposes. In [3], only the medians and the centroids themselves know who they are.

The rest of the paper is organized as follows. Section 2 contains a description of the protocols while Section 3 provides proofs of correctness. Section 4 states some conclusions. In the Appendix, we give some properties of centroids and medians of a tree which we used in developing the protocols.

2. Tree Metrics Protocols

Each node maintains a read/write register r containing several *fields*. The *state* of the system is defined by a value for every field of the registers maintained by the nodes. Each node in the system executes a *protocol* which has the form: {Phase name} $<$ *phase* $>$ {Phase name} ... {Phase name} $<$ *phase* $>$ Each *phase* is of the form: $<$ *rule* $>$ $\|$ ... $\|$ $<$ *rule* $>$

Each *rule* has the form: $<$ *guard* $>$ $\longrightarrow$ $<$ *assignment statement* $>$

A *guard* is a Boolean expression over the state of a node and its neighbors. An *assignment statement* updates the state of a node. A rule whose guard is true at some state of the system is said to be *enabled* at that state. A node i *depends* on a node j if a change in the state of j enables some rule of i.

We define a phase to be *convergent* if its rules are so constructed as to make the dependency relation between the nodes of the system a *partial order*, and upon execution of these rules, the state of the system eventually satisfies a global state predicate. Intuitively, the dependency relation is antisymmetric so that thrashing cannot occur. A phase is defined to be *closed* if no rules in it are enabled once the state of the system satisfies a global state predicate. A phase is said to be *stabilizing* if it is convergent and closed [4].

The *write set* of a phase is the set of register fields that are updated in the phase. If the write sets of the phases constituting the protocol are mutually disjoint and each of the phases is individually stabilizing, then the protocol is stabilizing.

The read/write register r_i of node i contains the following fields:

$r_i.parent$	The id of the parent of i; zero for the root.
$r_i.ht$	The height of i.
$r_i.dt.up$	Used to convergecast the diameter information.
$r_i.dt.down$	The diameter of the tree.
$r_i.center.up$	Used to convergecast the centroid information.
$r_i.center.down$	The centroid of the tree.

$r_i.count$	Number of nodes in the subtree rooted at i.
$r_i.nodes$	Number of nodes in the tree.
$r_i.median.up$	Used to convergecast the median information.
$r_i.median.down$	The median of the tree.

Node i can perform *read/write* operations on its local register r_i, but it can only *read* from registers r_j of its neighbors (*i.e.*, its parent and children). We assume that an underlying spanning tree protocol [5,6,7] maintains the consistency of the *parent* field in the registers. We do not make any assumption about a fair scheduler. Our protocols also work under a distributed scheduler [8,9]. Although we do not use the read/write atomic model [7], our protocols will also work in this model.

The protocols for diameter, centroid and median computation work in two phases. In the up phase, the value of the metric is computed in each node's *up* variable using the *up* variables of its children, so that the *up* of the root stabilizes to the correct value of the metric. The root then copies its *up* variable to its *down* variable. In the down phase, each node copies the *down* variable of its parent into its *down* variable, so that *down* contains the correct value for the metric.

2.1. Functions Used in the Protocols

In order to simplify the presentation of the rules in Sections 2.2 and 2.3, we use the following functions. The function *NOTRHS* is used to avoid the following type of situations:

$r_i.ht \neq MAX\_CHILD\_HT(i) + 1 \longrightarrow r_i.ht := MAX\_CHILD\_HT(i) + 1;$

This check is done so that the protocol will work correctly even with an unfair scheduler. Rules whose guards are false will not be scheduled and are "blocked out" of execution.

```
(* Returns the set of registers of the children of i, and the null set if i is a leaf. *)
CHILD(i) { {r_j} := {};      (* {r_j} is a local variable. *)
           for each j | (r_j.parent = i)
               add*({r_j}, r_j);
           return {r_j}; }

(* Returns the largest ht value of the children of i, and 0 if i is a leaf. *)
MAX_CHILD_HT(i) { MAX(CHILD(i).ht); }

(* Returns the second largest ht value of the children of i, and 0 if i is a leaf. *)
MAX2_CHILD_HT(i) { return MAX2(CHILD(i).ht); }

(* Returns the largest dt.up value of the children of i, and 0 if i is a leaf. *)
MAX_CHILD_DT(i) { return MAX(CHILD(i).dt.up); }

CENTROID(i)†{ return r_i.ht == ⌈(r_i.dt.down/2)⌉ + 1); }
```

*The add function adds an element to a set if it is not already a member of the set.

†For the other centroid, replace $\lceil\ \rceil$ with $\lfloor\ \rfloor$

$ROOT(i)$ { return $r_i.parent == 0$; }

(* Returns the register of the parent of i *)
$PARENT(i)$ { return $r_j \mid r_i.parent = j$; }

$MEDIAN(i)$ { return $2 * MAX(CHILD(i).count) > r_i.nodes$; }

NOTRHS { Let X be the boolean expression obtained from the right hand side
of the rule, i.e., the 'assignment statement',
after replacing $:=$ with $\neq$.
Return the boolean value resulting from the evaluation of X. }

2.2. Diameter and Centroid Protocol

The protocol consists of eight rules—$R0$ ... $R7$, $R0$ ... $R3$ for diameter calculation and $R4$ ... $R7$ for centroid identification. The function MAX in $R1$ and $R3$, and the function $MAX2$ in $R3$ calculate the greatest and second greatest values of their parameters, respectively. These functions return zero when applied to the null set and the singleton set, respectively.

Definition 1 *The height of a non-leaf node is one plus the maximum height of its children; the height of a leaf being one.*

Definition 2 *The diameter of a tree is the number of edges in a longest simple path in the tree.*

The diameter protocol ensures that the register field $r_i.dt.down$ in each node stabilizes to the value of the diameter of the tree. This occurs in three phases. In Phase I, rule $R0$ calculates the height of the node in $r_i.ht$. Rule $R1$ performs a convergecast so that the variable $dt.up$ at the root stabilizes to the value of the diameter of the tree. This is Phase 2. The variable $dt.up$ at each node is the sum of the two greatest ht values of its children, or the greatest $dt.up$ value of its children, which is the diameter of the subtree rooted at the node. $dt.up$ of a leaf is zero. Rules $R2$ and $R3$ constituting Phase 3 broadcast the diameter, so that the value of $dt.down$ at each node equals the diameter of the tree. Each node except the root copies $dt.down$ from $dt.down$ of its parent ($R3$). The root copies it from its own $dt.up$ instead ($R2$).

Definition 3 *A node in a tree is called a centroid if it is a middle node in a longest simple path in the tree.*

The centroid protocol has two phases. In Phase 1, a convergecast of the index of the centroid occurs ($R4$ and $R5$). One of the two centroids of the tree (or the only one: refer to Lemma A.3 in the Appendix) is the node whose ht equals $\lceil \frac{dt.down}{2} \rceil + 1$ ($R4$). In Phase 2 ($R6$ and $R7$), the index of the centroid is broadcast to all nodes. Each node copies $center.down$ from $center.down$ of its parent ($R7$). The root copies $center.down$ from its own $center.up$ ($R6$).

{Compute *ht* values}
R0 :: $NOTRHS \longrightarrow r_i.ht := MAX\_CHILD\_HT(i) + 1$

{Convergecast the diameter}
|| R1 :: $NOTRHS \longrightarrow r_i.dt.up := MAX(MAX\_CHILD\_HT(i) + MAX2\_CHILD\_HT(i), MAX\_CHILD\_DT(i))$

{Broadcast the diameter}
|| R2 :: $ROOT(i) \wedge NOTRHS \longrightarrow r_i.dt.down := r_i.dt.up$

|| R3 :: $\sim ROOT(i) \wedge NOTRHS \longrightarrow r_i.dt.down := PARENT(i).dt.down$

{Convergecast the centroid }
|| R4 :: $CENTROID(i) \wedge NOTRHS \longrightarrow r_i.center.up := i$

|| R5 :: $\sim CENTROID(i) \wedge NOTRHS \longrightarrow r_i.center.up := MAX(CHILD(i).center.up)$

{Broadcast the centroid}
|| R6 :: $ROOT(i) \wedge NOTRHS \longrightarrow r_i.center.down := r_i.center.up$

|| R7 :: $\sim ROOT(i) \wedge NOTRHS \longrightarrow r_i.center.down := PARENT(i).center.down$

2.3. Median Protocol

The protocol consists of seven rules, $R8$ ... $R14$. The function MAX in $R11$ and $R12$ calculates the greatest value of its parameters, and the function SUM in $R8$ calculates the sum of its parameters. Both of these functions return 0 when applied to the null set.

Definition 4 *A node in a tree is called a median if the sum of the distances from this node to all other nodes in the tree is the least possible.*

The protocol ensures that the register field $r_i.median.down$ in each node stabilizes to the index of one of the medians of the tree. This occurs in four phases. In Phase I, rule $R8$ calculates the *count* at each node i, which is the number of nodes in the subtree rooted at i. At the end of Phase I, the value of *count* at the root is the count of nodes in the tree. In Phase II, the value of *nodes* at each node i stabilizes to the value of the number of nodes in the tree. The rules for Phase II involve the root copying its *nodes* from its *count* ($R9$) and each node copying *nodes* from the variable *nodes* of its parent ($R10$). In Phase III, the median is computed using the rules $R11$ and $R12$. These rules perform a convergecast so that the value of *median.up* at the root stabilizes to the node index of one of the medians of the tree. A node i checks if twice the greatest $r_j.count$ of all its children is less than *nodes*, and if so, it declares itself the median by setting $r_i.median.up$ to its own index ($R11$). Otherwise, it copies the greatest *median.up* from its children into $r_i.median.up$ ($R12$). The value of *median.up* at the root stabilizes to the index of the median of the tree.

In Phase IV, a broadcast of the index of the median is done. The root copies its *median.up* variable into its *median.down* variable ($R13$). Each non-root node copies *median.down* from its parent's *median.down* ($R14$). Thus the value of *median.down* at each node stabilizes to the index of the median of the tree.

{Compute *count* values}
R8 :: $NOTRHS \longrightarrow r_i.count := SUM(CHILD(i).count) + 1$

{Broadcast value of *nodes*}
|| R9 :: $ROOT(i) \wedge NOTRHS \longrightarrow r_i.nodes := r_i.count$

|| R10 :: $\sim ROOT(i) \wedge NOTRHS \longrightarrow r_i.nodes := PARENT(i).nodes$

{Convergecast the median}
|| R11 :: $MEDIAN(i) \wedge NOTRHS \longrightarrow r_i.median.up := i$

|| R12 :: $\sim MEDIAN(i) \wedge$
$NOTRHS \longrightarrow r_i.median.up := MAX(CHILD(i).median.up)$

{Broadcast the median}
|| R13 :: $ROOT(i) \wedge NOTRHS \longrightarrow r_i.median.down := r_i.median.up$

|| R14 :: $\sim ROOT(i) \wedge$
$NOTRHS \longrightarrow r_i.median.down := PARENT(i).median.down$

3. Proof of Correctness

To prove that a protocol is correct, we prove that each phase constituting the protocol is *convergent* and *closed*. *Closure* is proved by defining a *global state predicate* for each phase and proving that once this state is reached, no rule in the phase is enabled for any node. In each phase, we prove the *convergence* by induction. This is acceptable since every phase is either *up convergent* or *down convergent*. An up convergent phase maintains a linear order $\prec$ among the nodes of the system such that

$$(\forall\, i)(\forall\, j)\ (i \prec j) \quad \text{iff} \quad r_i.ht < r_j.ht$$

For a down convergent phase, the order $\prec$ is such that

$$(\forall\, i)(\forall\, j)\ (i \prec j) \quad \text{iff} \quad r_i.ht > r_j.ht$$

Intuitively, information flow is upwards towards the root for an up convergent phase, while it is towards the leaves for a down convergent phase. For an up convergent phase, the leaves are the minimal elements of the partial order while for a down convergent phase, the root is the minimal element. Hence, for an up convergent phase, induction is done with the leaves as the bases, while for a down convergent phase, the root forms the basis of the induction.

Convergence is guaranteed even with an unfair scheduler because the nodes form a partial order, and thus, the scheduler is constrained to schedule those nodes which have not stabilized yet. Therefore, convergence is achieved in finite time.

The distributed scheduling permits simultaneous actions by different nodes. Our

protocols work with a such a scheduler because the dependency graph of the nodes is acyclic. Thus one node executing actions concurrently with another cannot interfere with, and undo the actions of, another.

3.1. Diameter and Centroid Protocols

The following global state predicates are defined for different phases in our protocols:

$$
\begin{aligned}
G_h &:: \forall i,\ r_i.ht = MAX\_CHILD\_HT(i) + 1 \\
G_{d1} &:: G_h \wedge (\forall i,\ r_i.dt.up = MAX(MAX\_CHILD\_HT(i) \\
&\qquad + MAX2\_CHILD\_HT(i),\ MAX\_CHILD\_DT(i))) \\
G_{d2} &:: G_{d1} \wedge (\forall i,\ (ROOT(i) \wedge (r_i.dt.down = r_i.dt.up)) \vee \\
&\qquad (\sim ROOT(i) \wedge (r_i.dt.down = PARENT(i).dt.down))) \\
G_{c1} &:: G_{d2} \wedge (\forall i,\ (CENTROID(i) \wedge (r_i.center.up = i)) \vee \\
&\quad (\sim CENTROID(i) \wedge (r_i.center.up = MAX(CHILD(i).center.up)))) \\
G_{c2} &:: G_{c1} \wedge (\forall i,\ (ROOT(i) \wedge (r_i.center.down = r_i.center.up)) \vee \\
&\qquad (\sim ROOT(i) \wedge (ri.center.down = PARENT(i).center.down)))
\end{aligned}
$$

Lemma 1 *The phase {Compute ht values} is stabilizing.*

Proof: It is evident that the only rule for this phase, $R0$, is not enabled in the state G_h. So, the phase is closed. This phase is up convergent by inspection. This can be proved inductively using the definition of *ht* of a node. □

Lemma 2 *The value of $r_i.dt.up$ in each node i stabilizes to the diameter of the subtree rooted at i after a finite number of executions of Rule $R1$.*

Proof: $R1$ implements an up convergent phase since the guard of $R1$ for i is an expression over registers r_j of the children j of i. The guard of $R1$ is not true in state G_{d1}. So, this phase is closed.

The convergence can be proved by induction on the height of the subtree rooted at i as follows.

Basis: The minimal elements are the leaves. If i is a leaf, rule $R1$ stores in $r_i.dt.up$ the value zero which is the diameter of the tree rooted at i. Thus, the base case is true.

Induction Hypothesis: Assume that $R1$ converges $r_j.dt.up$ to the diameter of the subtree rooted at j where js are those nodes which have height $h \geq 1$.

Induction Step: We now establish that $R1$ converges $r_i.dt.up$ to the diameter of the subtree rooted at i when the subtree has height $h + 1 > 1$.

Let ρ be a largest simple path in the subtree rooted at i. Since the subtree rooted at i has height > 1, it must have at least one child. We deal with two cases: (1) i has exactly one child and (2) i has more than one children.

Case 1: Node i has exactly one child (node j).

In this case, either path ρ has i as an endpoint, or it does not include i. If i is an endpoint of ρ, the diameter of the tree rooted at i is $r_i.ht$ which, by definition, is greater than or equal to the diameter of the subtree rooted at j. By the induction hypothesis, $r_j.dt.up$ has converged, so that $r_i.dt.up$ also converges.

If ρ does not include i, the diameter of the subtree rooted at i equals the diameter of the subtree rooted at j which, by the hypothesis, has already converged. Since path ρ does not include node i, $r_j.ht$ must be less than or equal to the diameter of the subtree rooted at j. Thus, in either case, the variable $r_i.dt.up$ converges to the diameter of the subtree rooted at i.

Case 2: Node i has more than one children.

Again, either path ρ goes through node i or it does not include i. In the former case, the rule $R1$ computes $r_i.dt.up$ as the sum of the largest two heights of the children of i (the value of ht at all nodes has stabilized), which by definition, is greater than or equal to the diameter of any subtree rooted at a child of i. By the induction hypothesis, the value of $r_j.dt.up$ has converged to the value of the diameter of the subtree rooted at j for every child j of i.

In the latter case, the diameter of the subtree rooted at i is equal to the diameter of the subtree of a child j of i. By definition, this value is greater or equal to the sum of the largest two heights of the children of i (which have already stabilized). In either case, it is simple to verify that the value of $r_i.dt.up$ converges to the value of the diameter of the subtree rooted at i. □

Corollary 1 *The variable dt.up at the root stabilizes to the value of the diameter of the tree after a finite number of executions of the rules R0 and R1.*

Proof: Follows directly from Lemma 2. □

Lemma 3 *The variable dt.down in each node i stabilizes to the value of the diameter of the tree after a finite number of executions of R2 and R3.*

Proof: The phase is closed with respect to G_{d2} since rules $R2$ and $R3$ are not enabled when the system is in this state.

$R2$ and $R3$ implement a down convergent phase since the guards of $R2$ and $R3$ are expressions over registers r_j of the parent of i, if one exists. The proof of convergence by induction is as follows.

Basis: The root is the basis of the induction. By Corollary 1, the value of $r_1.dt.down$ eventually becomes equal to the diameter of the tree. By applying $R2$, the root sets $dt.down$ equal to $dt.up$. Hence the value of $r_1.dt.down$ equals the diameter of the tree.

Induction Hypothesis: Assume that all nodes at level l have $dt.down$ equal to the diameter of the tree.

Induction Step: We now establish that all nodes at level $l+1$ will eventually have $dt.down$ equal to the diameter of the tree. The down convergence of this phase implies that the nodes at level $l+1$ *depend* only on those at levels l and below, so that if those at level l have converged, then so do those at level $l+1$. □

Theorem 1 *The diameter protocol is correct.*

Proof: The *write sets* of the phases of this protocol are $\{ r_i.ht \}$, $\{ r_i.dt.up \}$ and $\{ r_i.dt.down \}$. These are mutually disjoint, by observation. Hence, the diameter protocol is correct since its individual phases have been proven correct by Lemma 1, Corollary 1, and Lemma 3. □

Lemma 4 *The value of $r_i.center.up$ at the root stabilizes to the index of one of the centroids of the tree after a finite number of executions of rules R4 and R5.*

Proof: It is evident that the guards of $R4$ and $R5$ are not enabled once the system reaches G_{c1}. Hence this phase is closed with respect to G_{c1}. The proof of convergence follows.

For at least one node P_c, the expression $(r_c.ht = \lceil \frac{r_c.dt.down}{2} \rceil + 1)$ is true. Refer to Lemma A.1 in the Appendix for a proof. This expression forms part of the guards of $R4$ and $R5$ and hence will be true for at least one node, namely, one of the centroids of the tree. This node sets its register field $r_c.center.up$ to its index. Since this phase is up convergent, it can be proved by induction using this node as the basis that the root eventually gets the centroid's index in its register field *center.up*. □

Lemma 5 *The variable center.down in each node i stabilizes to the index of the centroid of the tree after a finite number of executions of R6 and R7.*

Proof: It is evident that the guards of $R6$ and $R7$ are not enabled once the system reaches G_{c2}. Hence this phase is closed with respect to G_{c2}.

The phase $\{Broadcast\ the\ centroid\}$ being down convergent, an inductive proof can be constructed similar to that of Lemma 3. □

Theorem 2 *The centroid protocol is correct.*

Proof: Notice that the centroid protocol includes the three phases of the diameter protocol, apart from the two phases that find the centroid. By inspection, the *write sets* of the five phases are mutually disjoint. Thus, the centroid protocol is correct since its individual phases have been proven correct by Lemma 1, Corollary 1, Lemmas 3, 4 and 5. □

3.2. Median Protocol

The following global states are defined for the phases in this protocol and will be used in the lemmas that follow:

$$
\begin{aligned}
G_c &:: \forall i, r_i.count = SUM(CHILD(i).count) + 1 \\
G_n &:: G_c \wedge (\forall i, (ROOT(i) \wedge (r_i.nodes = r_i.count)) \vee \\
&\qquad (\sim ROOT(i) \wedge (r_i.nodes = PARENT(i).nodes))) \\
G_{m1} &:: G_n \wedge (\forall i, (MEDIAN(i) \wedge (r_i.median.up = i)) \vee \\
&\qquad (\sim MEDIAN(i) \wedge (r_i.median.up = MAX(CHILD(i).median.up)))) \\
G_{m2} &:: G_{m1} \wedge (\forall i, (ROOT(i) \wedge (r_i.median.down = r_i.median.up)) \vee \\
&\qquad (\sim ROOT(i) \wedge (r_i.median.down = PARENT(i).median.down))))
\end{aligned}
$$

Lemma 6 $\{Compute\ count\ values\}$ *stabilizes the value of $r_i.count$ at each node i to the count of the nodes in the subtree rooted at i.*

Proof: It is easy to verify that this phase is closed when the system reaches the state G_c. This phase is up convergent and a proof for this is inductive with the leaves as the bases. □

Lemma 7 $\{Broadcast\ value\ of\ nodes\}$ *stabilizes the value of $r_i.nodes$ in each node to the count of nodes in the tree.*

Proof: The proof is similar to that of Lemma 3. The system is closed when it reaches the state G_n. □

Lemma 8 *{Convergecast the median} stabilizes the value of $r_i.median.up$ at the root to the index of the median.*

Proof: Refer to Lemma A.5 in the Appendix for a proof that for at least one node P_m in the tree, the predicate $(\forall j)\ (r_j.parent = m)\ (2{*}max(r_j.count) < r_m.nodes)$ will be true, this node being one of the medians of the tree. This expression being part of the guards of $R11$ and $R12$, the node P_m sets $r_m.median.up$ to its node index. Using this node as the basis, we can prove that this phase is up convergent. The proof is similar to that of Lemma 4. The system is closed when it reaches the state G_{m1}. □

Lemma 9 *{Broadcast the median} stabilizes the value of $r_i.median.down$ at each node to the index of one of the medians.*

Proof: The proof for this lemma is again identical to that of Lemma 3. When the system reaches state G_{m2}, it is closed since no rules are enabled. □

Theorem 3 *The median protocol is correct.*

Proof: Notice that the write sets of the phases constituting the median protocol are disjoint. Since we have proven that each phase is individually stabilizing, the median protocol is correct. □

3.3. Complexity

Lemma 10 *The time complexity of any phase is proportional to the length of the longest dependency chain of the partial order for the phase.*

Proof: The minimal elements of the partial order stabilize immediately. Each non-minimal element depends directly or indirectly on those elements that *precede* it in the partial order. Thus the time taken for a phase to stabilize increases with increasing length of the longest dependency chain. □

4. Conclusion

The protocols presented in this paper are self-stabilizing algorithms for computing the diameter and locating the centroids and medians of a distributed tree structured network. They provide fault-tolerant means of drawing topological information about a tree network. No assumptions are made about the fairness of the scheduler. A distributed scheduling model may also be assumed for the network and the protocol will still work correctly. The model assumed has very weak atomicity. The ideas behind these algorithms could conceivably be extended to finding the diameter, centroids, and medians of a general graph network; this would be a challenging problem.

References

[1] E. Dijkstra, "Self-Stabilizing Systems in Spite of Distributed Control," *Communications of the ACM*, Vol. 17, 1974, pp. 643-644.

[2] M. Schneider, "Self-Stabilization," *ACM Computing Surveys*, Vol. 25, No. 1, March 1993, pp. 45-67.

[3] M. H. Karaata et al, "Self-Stabilizing Algorithms for Finding Centers and Medians of Trees," *The 13th Annual ACM Symposium on Principles of Distributed Computing* , Los Angeles, CA, August 14-17, 1994, pp. 374.

[4] A. Arora and M. Gouda, "Closure and convergence: A foundation of fault-tolerant computing," *22nd International Symposium on Fault-Tolerant Computing*, pp. 396-403, 1992; also *IEEE Transactions on Software Engineering*, Vol. 19, No. 11, 1993, pp. 1015-1027.

[5] A. Arora and M. Gouda, "Distributed Reset," *10th Conference on Foundations of Software Technology and Theoretical Computer Science, Bangalore, India, pp.316-331, December 17-19, 1990, Lecture Notes in Computer Science 472, Springer-Verlag*; also *IEEE Transaction of Computers*, Vol. 19, No. 11, November 1993, pp. 1015-1027.

[6] N. Chen, H. Yu, and S. Huang, "A Self-Stabilizing Algorithm for Constructing Spanning Trees," *Information Processing Letters*, Vol. 39, pp. 147-151, 1991.

[7] S. Dolev, A. Israeli, and S. Moran, "Self-Stabilization of Dynamic Systems Assuming only Read/Write Atomicity," *9th Annual ACM Symposium on Principles of Distributed Computing*, Quebec City, Canada, pp. 103-117, 1990; also *Distributed Computing* Vol. 7, 1993, pp. 3-16.

[8] G. Brown, M. Gouda, and M. Wu, "Token Systems that Self-Stabilize," *IEEE Transactions on Computers*, Vol. 38, No. 6, pp. 845-852, 1989.

[9] J. Burns, M. Gouda, and R. Miller, "On Relaxing Interleaving Assumptions," *Proc. MCC Workshop on Self-Stabilization*, Austin, Texas, November 1989.

[10] 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, 1994.

[11] N. Deo, "Graph Theory with Applications to Engineering and Computer Science," *Englewood Cliffs, N.J., Prentice Hall*, 1974.

Appendix A. Properties of Centroids and Medians

Lemma A.1 *[Korach, Rotem, and Santoro [10]] The statement* $r_i.ht = \lceil \frac{r_i.dt.down}{2} \rceil + 1$ *holds for only one node of the tree* T*, that node being a centroid of the tree.*

Proof: It is simple to see that for at least one centroid P_c of the tree, the statement $r_c.ht = \lceil \frac{r_c.dt.down}{2} \rceil + 1$ holds.

We will prove Lemma A.1 by contradiction, by assuming that there is another node P_{c1} in the tree for which the statement $r_{c1}.ht = \lceil \frac{r_{c1}.dt.down}{2} \rceil + 1$ holds.

Since the nodes P_c and P_{c1} are not identical, but have the same height, it cannot be that one is a predecessor of the other in the tree. Let P_x be the node which is the closest ancestor of both P_c and P_{c1}. Then the path consisting of a longest path from P_c to a leaf, plus the path from P_x to P_c, plus the path from P_x to P_{c1}, plus a longest path form P_c1 to a leaf, is a simple path and has length greater than or equal to $2 * \lceil \frac{r_x.dt.down}{2} \rceil + 2 > r_x.dt.down$. Since this contradicts the definition of diameter of a tree, it cannot be that the statement $r_i.ht = \lceil \frac{r_i.dt.down}{2} \rceil + 1$ holds for more than one node in T. □

Lemma A.2 *The statement* $r_i.ht = \lfloor \frac{r_i.dt.down}{2} \rfloor + 1$ *holds for only one node of the tree, that node being a centroid of the tree.*

Proof: The proof is similar to that of Lemma A.1. □

Lemma A.3 *[Deo [11]] There are at most two centroids in a tree.*

Proof: Refer to [11] for the proof. □

Lemma A.4 *There may be more than one longest path in a tree, but all of them contain the centroid(s) in the tree.*

Proof: Refer to [11] for the proof. □

Lemma A.5 *A median of the tree,* P_m*, satisfies the condition,* $2ct(j) > n$ *where ct(j) is the maximum count of the children of* P_m *as defined below and* n *is the number of nodes in the tree.*

Proof: Before we can prove this lemma, we will need the following definitions and observations:

Definition A.1 *The count of a leaf is 1, and the count of a non-leaf node is one plus the sum of the counts of its children.*

Definition A.2 *The total distance from node i to all the nodes in a tree is the sum of the lengths of the path from node i to each node in the tree.*

Observation A.1 *[Korach, Rotem and Santoro [10]] If node i in a tree has total distance to all nodes in T equal to dis(i), then the corresponding value for its child j is*

$$dis(j) = dis(i) + n - 2ct(j),$$

where ct(j) is the count of j.

This formula follows from the fact that the length of the path from a node i to a node k, which is not j or a descendant of j, is one less than the length of the path from node j to node k. The length of the path from i to a node k, which is either node j or a descendent of j, is one more than the length of the path from j to node k. The number of nodes in the subtree rooted at i is $ct(i)$.

Observation A.2 *If node i in the tree T has* $2ct(i) > n$*, then for at most one of its children j,* $2ct(j) \geq n$*. When such a child exists,* $dis(j) \leq dis(i)$*.*

This follows from the fact that n is the total number of nodes in the tree and $ct(i) \leq n$.

The proof of Lemma A.5 can now be stated. Assume that we know that all medians of the tree are in the subtree rooted at i and that $2ct(i) > n$. Then by Observation A.2, one of the following three cases applies:

Case 1: There is one child j of i for which $2ct(j) > n$. Then by Observation A.1, $dis(i) < dis(j)$, and $dis(k) > dis(i)$ for all the other children k of i. Therefore, all medians are in the subtree rooted at j and $2ct(j) > n$.

Case 2: There is only one child j of i for which $2ct(j) = n$. By Observation A.1, $dis(i) = dis(j)$ and $dis(k) > dis(i)$ for all other children k of i. Therefore j and i are the medians of the tree.

Case 3: There is no child j for which $2ct(j) \geq n$. By Observation A.1, there cannot be a median in the subtrees rooted at any child of i. Therefore node i is the median of the tree. □

Lemma A.6 *[Korach, Rotem, and Santoro [10]] There are at most two medians in a tree.*

Proof: Refer to [10] for the proof. □