Self-stabilizing minimum spanning tree

Group 43
Shivneshwar Velayutham and Madhumitha Venkatesan

Algorithm

The first step is to do leader election using the following algorithm from the book “Leader Election in a General Communication Network” (Page 34). This is possible since every processor has a unique global ID. We already know that this algorithm is self-stabilizing, so we can use this confidently.

After doing the leader election, we want our leader to know the topology of the graph so that the leader will be able to form the minimum spanning tree and teach it to all the other processors.

The topology of the graph can be retrieved using the following algorithm from the book “Self-Stabilizing Convergecast for Topology Update” (Page 96).
After h number of cycles (h being the height of the graph ie. Largest distance from root to leaf) we can be sure that the leader will have the accurate topology of the graph.

The leader will then start forming the minimum spanning tree. This can be done using the Kruskal algorithm which uses a greedy approach to solving this.

Kruskal algorithm pseudocode

1. Sort all the edges in ascending order.
2. Pick smallest edge and check if it forms a cycle with the tree formed so far.
3. If it forms a cycle, discard the edge. If not, include it in the tree.
4. Repeat until N-1 edges are found (N being number of processors) since a minimum spanning tree of N processors will have N-1 edges.

Cycle detection using Union Find Algorithm

1. Initially create subsets for each processor containing only itself.
2. As we add edges, check if the source and destination belong to the same subset.

3. If they do not belong to the same subset, then do a union over the two subsets. (Processors in the same subsets mean there is a route from each processor in the subset to the other.)
4. If they already belong to the same set, a cycle exists.

Using the above algorithm, the leader builds the minimum spanning tree. Now the leader must teach all the processors this topology. This can be done using the following algorithm in the book "Self-Stabilizing Broadcast for Topology Update"(Page 97). This algorithm will self-stabilize and teach the topology to all the processors after h cycles (height of the graph).

Proof for self-stabilization

The proof that this algorithm will be self-stabilizing is by using fair composition (Section 2.7 in the book) of self-stabilizing algorithms which are stacked upon each other one by one. The Kruskal algorithm is a deterministic algorithm and will give the right result as long the topology update stabilizes and returns the correct topology to the leader.

Message size

Let's take a look at the message size needed by each processor.

We can ignore the memory needed by each node for leader election since it's quite minimal (just contains the leader ID and it's distance to it).

For topology update and broadcast, the maximum would the leader who would have to store the entire graph's edges in the message or shared memory.

Communication rounds

The following are the number of communications rounded for each algorithm to stabilize.

1. Leader election takes N rounds. (N being number of processors.)
2. Both topology update and broadcast takes h rounds to stabilize. (h being height of the graph).

So in total the number of communication rounds needed is **N + 2h**.