

Distributed MST Construction

Arvind Krishnamurthy
Fall 2003

Distributed MST

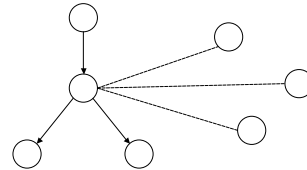
- Last week:
 - Started studying synchronous version of distributed MST algorithm
- Today:
 - Finish discussing synchronous version
 - Study algorithm in asynchronous model
- Both algorithms are based on MST construction algorithm by Gallager, Humblet, and Spira

Basic Node Execution

- Each node maintains state regarding its edges
 - Knows their weights
 - Classifies them into one of three bins:
 - Unexplored, tree branch, rejected
 - Initially all edges are unexplored
 - When an edge is discovered to be an intra-fragment edge, it is considered "rejected"
 - It knows which edge leads to parent
 - All other "branch" edges lead to children
- Beginning:
 - Consider all edges to be unexplored
 - Sort the edges according to increasing weight

Basic Node Execution (contd.)

- When a node receives a search for MWOE from the leader:
 - Makes note of the leader and parent
 - Propagates search query along all other branch edges
 - Starts exploring each of its unexplored edges in sorted order
 - If other endpoint in the same fragment, edge is "rejected"
 - Finds MWOE from its node
 - Waits for MWOE results from children and reports to parent



Overall Algorithm (synchronous version)

- In each phase of the algorithm
 - Leader of each fragment does:
 - 1) Broadcasts "search" for MWOE
 - 1) Nodes propagate broadcast
 - 2) Synchronize before beginning test for outgoingness
 - 2) Waits for results to be convergecasted
 - 3) If no MWOE, terminate
 - 4) Leader sends a "connect" message down the tree across MWOE
 - 1) Marks the MWOE edge as a "branch"
 - 2) If "connect" has gone across the same edge in both directions, elect one of the two endpoints as leader
 - Synchronize, repeat

Analysis of a single phase

- How to synchronize?
 - Use counters, wait for counter to reach a certain value
 - Implies algorithm is synchronous and non-uniform
- Potential points of synchronization:
 - Broadcast is received by everyone after "n" ticks
 - Nodes might explore a maximum of "n" neighbors
 - Node finds MWOE in "2*n" ticks
 - Convergecast can take at most "n" ticks
 - New leader is elected within another "n" ticks

Analysis of Algorithm

- How many phases does the algorithm go through?
- What is the message complexity?

Announcements

- Sample C-TCP code:
 - Does not transmit the "\0" at the end of the buffer
 - Multithreaded server:
 - Can still use "select" mechanism to identify when a connection dies
- Original GHS paper:
 - <http://lambda.cs.yale.edu/cs425/doc/ghs.pdf>

Asynchronous Algorithm

- Cannot use local clocks (counters) to synchronize
- How do we fashion an asynchronous algorithm?

Asynchronous Algorithm

- Fragment identifier now contains the following:
 - Leader of the fragment
 - Level of the fragment
- Fragments combine in two ways:
 - Two fragments at the same level "merge" to form a fragment of the next level
 - A lower-numbered fragment can be "absorbed" by a higher level fragment
 - Does not change the level of the higher-numbered fragment
 - Does not change the leader of the higher-numbered fragment
 - Lower-numbered fragment inherits level, leader of the other fragment

Asynchronous Algorithm: Balanced growth

- When a node n_1 queries n_2
 - If $\text{level}(n_1) < \text{level}(n_2)$
 - n_1 and n_2 should be in different fragments
 - n_2 "accepts" the outgoingness "test"
 - If $\text{level}(n_1) == \text{level}(n_2)$
 - Check the leader identities
 - If leaders are different, "test" is accepted
 - If leaders are same, "test" is rejected
 - If $\text{level}(n_1) > \text{level}(n_2)$
 - Possible that n_1 and n_2 belong to the same fragment
 - Even if they are determined to be on different fragments, we allow only "merges" and "absorbs"
 - Wait for $\text{level}(n_2) \geq \text{level}(n_1)$



Asynchronous Algorithm

- Leader of each fragment does:
 - 1) Broadcasts "search" for MWOE
 - 1) Nodes propagate broadcast
 - 2) Nodes begin test for outgoingness immediately
 - 2) Waits for results to be convergecasted
 - 3) If no MWOE, terminate
 - 4) Leader sends a "connect" message down the tree across MWOE
 - 1) If "connect" goes from lower level to higher level:
 - Mark edge as "branch" for both endpoints
 - 2) When "connect" goes across MWOE for the first time between two fragments at the same level
 - Mark edge as "branch" for both endpoints
 - 3) When "connect" goes across MWOE in both directions:
 - Elect a leader, increment level of the new fragment

Proving Correctness

- Safety property:
 - No cycles
 - Relies on the correctness of the "outgoingness" test
- Liveness properties:
 - System is making progress
 - We have introduced "wait" as part of the "test" procedure
 - Fortunately, there are no cyclic-wait dependencies
 - Therefore, no deadlocks
 - Eventually, lowest-numbered fragment does make progress

Analysis of Algorithm

- Initially "n" fragments
- Number of fragments as level increases by one:
 - Decreases by at least a factor of two
- Maximum level number: $\log(n)$
- Time complexity:
 - At time 0, all fragments are at level 0
 - By time n, none of the level-0 fragments exist
 - At time t, let lowest-numbered fragment be at level k
 - By time t+n, all level k fragments have been promoted
 - Total time: $O(n \log(n))$

Message Complexity

- Remains the same as synchronous algorithm
- Number of "test" and number of "accept/reject" messages:
 - m each
- At each level, number of broadcast, convergecast messages: $O(n)$
- Total number of messages: $O(n \log(n) + m)$

Algorithm Wrapup

- Powerful distributed algorithm
 - Elects a leader
 - Computes a spanning tree
 - Finds the minimum weight spanning tree
 - All at a message complexity of $O(n \log n + m)$
 - Recall that $n \log n$ was a lower bound on messages
 - Cannot do better than $O(m)$: need to explore all edges
- Time complexity:
 - Improved later to $\min(n, (D + d) \log n)$
 - D: diameter of the MST
 - d: maximum node-degree of input graph