Distributed Algorithms for Dynamic Networks

February 19, 2014

Welles Robinson

Senior Thesis

Department of Computer Science

Georgetown University

Abstract

Abstract

1 High-Level Description of Goal

The goal of this paper is to simplify the creation of distributed algorithms for dynamic networks by demonstrating that any algorithm that works for the broadcast variant of the synchronous model with a star topology can be made to work for the broadcast variant of the synchronous model with any topology. We will do so by describing a simulation algorithm that, if run on every node in the broadcast model, will match perfectly the output of the nodes of the centrally controlled model.

2 Models

Main Execution Model

1. We are considering the broadcast variant of the synchronous model, defined with respect to a connected network topology G=(V,E).
2. The broadcast variant is defined such that a given node has no knowledge of its neighbors but may send a single message per round that all of its neighbors will receive.
3. Nodes in the network have comparable unique identification numbers (UIDs) and have knowledge of their own UID.

Reference Execution Model

1. We consider the synchronous model with a star network topology.
2. A given node records its sent message and its received message for each round in a communication log.

3 Problem Definition

This paper considers the simulation problem, the problem of correctly simulating an algorithm designed for the reference model on the main model. We examine the simulation problem using three different assumptions regarding the arrival and departure of nodes in the network.

We define the main execution to be an instance of the main execution model with n processes. We define the reference execution to be an instance of the reference execution model with $n+1$ processes.

3.1 The Static Case

The static case assumes that all nodes arrive in the network at the same time and that no node may depart from the network during the execution. Because the simulation problem is defined in the synchronous model, this assumption means that round r begins and ends at the same time for each node in the network.

The simulation problem provides the algorithm the following input, algorithms a_1 and a_2, and expects it to produce a list of communication logs. We say a given algorithm solves the simulation problem if it outputs list Y where Y is equivalent to the list of communication logs generated by the reference execution given input of algorithm a_1 to the leader and algorithm a_2 to all other nodes.

3.2 The Arrival Only Case

The arrival only case assumes that a node may arrive in the network at the beginning of any round of the main execution and no node may depart from the network during the execution. A node that arrives in the beginning of a round r in the main execution will arrive in the reference execution at the beginning of the next round of the reference execution.

3.3 The Departure Only Case

3.4 The Arrival and Departure Case

4 Algorithm

1. Run unmodified Leader Election without Network Information

(a) Assume nodes have UIDs (which is also assumed for the distributed model)

(b) Each process runs Terminating Synchronous Breadth First Search and the node that manages to terminate elects itself leader and tells the other nodes to terminate

2. The Simulation

(a) Assume that each and every round of the given algorithm has a finite repetitions of a step, which is comprised of two parts

i. The star node sends a broadcast message to all the leaf nodes (Broadcast part)

ii. All leaf nodes send a receive message to the star node in response to the broadcast message (Receive part)

(b) Each step is simulated by the leader node running a modified instance of terminating synchronous BFS

i. The search message sent by a parent node to its children nodes is modified to be "search" plus the broadcast message, which is dictated by the output of the input algorithm A1

ii. Upon receiving a search message, a node runs the input algorithm A2 on the broadcast message portion and then creates the receive message portion using the output of A2

iii. The done message sent by a child node to its parent node is modified to be "done" plus both the UID and the receive message of the child node as well as any done messages received by the child node

iv. Upon receiving a search message, a node writes the broadcast message portion to its communication log

v. Upon sending a done message, a node writes the receive message portion to its communication log

vi. The leader simulates sending a search message to itself and then simulates sending a done message back to itself

vii. This algorithm terminates when the leader has received done messages from all of its children and a simulated done message from itself

Static Model - All the nodes turn on at the same time

Simulation Algorithm takes one input, algorithm A, the algorithm to be simulated. Algorithm A can be broken into two distinct algorithms, A1, the algorithm run by the star process, and A2, the algorithm run by the leaf processes.

Member Variables - maxID (UID); parent (UID); totalChildren (int); childCount (int); wait (int); Message has a root (a round, the UID); a id of the sender (UID); a type search, choose, done; a receiver (UID), defaults to NULL;

Dynamic Addition Model - Nodes turn on at various times but they don't turn off The leader elected will be the node with the highest UID out of all of the nodes that turned on at round 1

Variables - maxRoot - (a round, the UID); parent (UID); totalChildren (int); childCount (int); wait (int); Message has a root (a round, the UID); a id of the sender (UID); a type search, choose, done; a receiver (UID);

5 Analysis

Lemma 5.1. *For the given network, a node will eventually set leader to true and no more than one node will have leader equal to true at the beginning of any round r.*

Proof. One node will eventually set leader to true (Lemma 5.4). No more than one node will have leader set to true at any point (Lemma 5.2). □

Definition 1. *Let u_{max} be the ID of the process with the maximum UID in the network. Let BFS instance b_i refer to an instance of the terminating breadth-first search protocol initiated by process with ID i.*

Lemma 5.2. *For every round r, at most one node has leader = true at the beginning of round r.*

Proof. A node with ID i will only set leader = true if the BFS instance b_i terminates. A BFS tree b_i will terminate only if i equals u_{max} (Lemma 5.3). Only the process with ID u_{max}, will set leader = true.

□

Lemma 5.3. *A BFS instance b_i will only terminate if i equals u_{max}.*

Proof. Termination of a BFS instance b_j requires all other processes in the network to send a done message to b_j. Given BFS instance b_j where $j < u_{max}$, there is at least one process, the process with ID u_{max}, that will never reply done to b_j. Therefore, b_j will never terminate. □

Lemma 5.4. *One node will eventually set its variable leader to true.*

Proof. A BFS instance b_i will eventually terminate if every node in the network runs b_i. Every node in the network will eventually run $b_{u_{max}}$ so $b_{u_{max}}$ will eventually terminate and the process with ID u_{max} will set leader = true.

□

Step 1: relate rounds of real execution to a single round in the reference execution Simulation of a round of the reference execution for leader starts when the leader broadcasts its message according to the algorithm and ends when the leader has received messages from all of its children Simulation of a round of the reference execution for a child node is between when the node broadcasts its message and when the node receives the message from the leader

Step 2: Define what it means for the real execution to correctly implement a round of the reference Successful simulation of the reference execution means that the leader receives the messages from all of the nodes and the nodes receive the message from the leader node

Step 3: Assume up until Round R in the reference execution, everything in the main execution has matched everything in the reference execution; prove Round R in the reference execution is correctly simulated So what happens in round R-1? The CNs all receive their message from the leader and the leader receives a message from every node in the network In the first round of the rounds of the main execution that simulate round R, the leader broadcasts his message to his children and all of the leaf nodes broadcast their message, which is received by their parent In the second round, the children of the leader broadcast his message, which is received by their children; the nodes who received messages from all of their children sends their messages to their parents

```
initVariables() ;
for round 1 ... r do
    for each message m in Inbox do
        if m.maxID > maxID then
            updateMaxRoot() ;
        end
        if m.maxID == maxID then
            if m.type == choose AND receiver == myUID then
                childCount++ ;
                totalChildren++ ;
            end
            if m.type == done AND receiver == myUID then
                childCount– ;
                if childCount == 0 then
                    sendDoneMsg( ) ;
                end
            end
        end
        if m.maxID < maxID then
            msg = (type=search, sender=myUID, maxID=maxID) ) ;
            Outbox.enqueue(msg) ;
        end
    end
    if wait != 0 AND childCount == 0 then
        wait– ;
        if wait == 0 then
            sendDoneMsg( ) ;
        end
    end
    for each message m in Outbox do
        broadcast(m);
    end
    myRound++ ;
end
```

Algorithm 1: Simulation Algorithm for Static Model

```
myRound == 0 ;
maxID = myUID ;
message m = (type=search, sender=myUID, maxID=maxID) ;
```

Algorithm 2: initVariables method

```
maxID = m.maxID;
parent = m.sender;
childCount = totalChildren = 0; msg1 = (type=choose, sender=myUID,
maxID=maxID, receiver=m.sender) ;
Outbox.enqueue(msg1) ;
msg2 = (type=search, sender=myUID, maxID=maxID) ;
Outbox.enqueue(msg2) ;
wait = 3 ;
```

Algorithm 3: updateMaxRoot method

```
msg = ( type=done, sender=myUID, maxID=maxID ) ;
Outbox.enqueue( msg ) ;
```

Algorithm 4: sendDoneMsg method

```
for each action in A1 do
    if leader == true then
        message = (type=r.action, sender=myUID) ;
        broadcast ( message ) ;
        msg = ( response to message ) ;
        commLog.write (response to message) ;
    end
    for each round r do
        for each message m in Inbox do
            if m.sender == parent then
                forwardMsgToChildren( ) ;
            end
            if m.receiver == myUID then
                add m to msg ;
                childCount– ;
                if childCount == 0 then
                    Outbox.enqueue(msg) ;
                end
            end
        end
        for each message m in Outbox do
            broadcast(m) ;
        end
    end
end
```

Algorithm 5: Static Simulation Algorithm

```
msg = (response to m, receiver = parent) ;
commLog.write(response to m) ;
if totalChildren == 0 then
    Outbox.enqueue(msg) ;
else
    forwardMsg = (type = m.type, sender=myUID) ;
    Outbox.enqueue(forwardMsg) ;
end
```

Algorithm 6: forwardMsgToChildren

```
initVariables() ;
for round 1...r do
    for each message m in Inbox do
        if m.root > maxRoot then
            updateMaxRoot() ;
        end
        if m.root == maxRoot then
            if m.type == choose AND receiver == myUID then
                childCount++ ;
                totalChildren++ ;
            end
            if m.type == done AND receiver == myUID then
                childCount– ;
                if childCount == 0 then
                    sendDoneMsg() ;
                end
            end
        end
        if m.root < maxRoot then
            msg = (type=search, sender=myUID, root=(r=maxRoot.r+1,
            id=maxRoot.id) ) ;
            Outbox.enqueue(msg) ;
        end
    end
    if wait != 0 AND childCount == 0 then
        wait– ;
        if wait == 0 then
            sendDoneMsg() ;
        end
    end
    for each message m in Outbox do
        broadcast(m);
    end
    myRound++ ;
    maxRoot = (r=maxRoot.r+1, id=maxRoot.id) ;
end
```

Algorithm 7: Simulation Algorithm for the Dynamic Addition Model

```
myRound == 0 ;
maxRoot = (r=myRound, sender=myUID) ;
message m = (type=search, id=myUID, root=maxRoot) ;
```

Algorithm 8: initVariables method for Dynamic Addition Model

maxRoot = m.root;
parent = m.sender;
childCount = totalChildren = 0; msg1 = (type=choose, sender=myUID,
root=(r=maxRoot.r+1, id=maxRoot.id), receiver=m.id) ;
Outbox.enqueue(msg1) ;
msg2 = (type=search, sender=myUID, root=(r=maxRoot.r+1,
id=maxRoot.id)) ;
Outbox.enqueue(msg2) ;
wait = 3 ;

Algorithm 9: updateMaxRoot method for Dynamic Addition Model

msg = (type=done, sender=myUID, root=(r=maxRoot.r+1,
id=maxRoot.id), receiver=parent) ;
Outbox.enqueue(msg) ;

Algorithm 10: sendDoneMsg method for Dynamic Addition Model