CS 553 DISTRIBUTED SYSTEMS

REPORT ON

HIRSCHBERG-SINCLAIR ALGORITHM FOR LEADER ELECTION ON A LOGICAL RING

Submitted to: Prof. Ajay Kshemkalyani

Submitted by: Surya Selvaraj [UIN: 679003533]

INTRODUCTION: The main aim of this project was to implement the Hirschberg Sinclair Algorithm on a logical ring using Java RMI/Sockets. Every node acts as a server as well as a client. That is, it both sends and receives messages to and from the neighbors.

ENGINEERING DECISIONS ON THE IMPLEMENTATION: These are some of the engineering decisions employed in this implementation:

- Parameter n: n represents the number of nodes. This input is taken as a comman line argument. However, if the command line argument is not specified, a message is displayed informing the user the syntax for running the program.
- Process IDs: Process ID are randomly generated using Random class object i Java.
- Process chosen for activation: A process is randomly chosen for activation, i.e: To initiate the leader election process.

EXPLANATION:

This is an implementation of the Hirschberg-Sinclair Leader election algorithm. It' written in Java and uses Java RMI for communication between Nodes.

This was developed on a Mac running Mac OS X 10.7.3

The build and run script uses osascript to open a terminal window for the `Ringer process, and `n` terminal windows for `n Node` processes.

You can choose to not have any terminal windows for the Node processes by editing the `run-hs` file. Inside the file is documentation to help you do that.

The build script also generates a new java security policy file from `java.policy.default` that gives file writing, reading and executing permission for the current working directory.

`run.sh` automatically places the current working directory in the appropriate place in the policy file and outputs the new file as `java-security.policy`.

Since for this project I was required to keep track of average number of messages sent by nodes, I have some reporting functionality that writes a file named `stat.txt`.

This file contains a record of runs of this algorithm with the node count and average messages sent and average messages received.

The source codes are all well-documented to explain the approach clearly.

TO RUN THE ALGORITHM:

>`$ ./run-hs <number of nodes>`

NEED FOR HIRSCHBERG-SINCLAIR:

The LeLang, Chang, Roberts (LCR) Algorithm is the predecessor to Hirschberg-Sinclair Algorithm in an asynchronous distributed system for ring topology. In LCR a node initiates the algorithm, it sends its process ID to its right neighbor and awaits the same from the left neighbor. Suppose its process ID is i and it receives process ID k.

- If i < k, it passes the process ID k to its right neighbor. Because it cannot be the leader.
- If i > k, it ignores the message as it is dominant.
- If i = k, the message it passed has traversed around the ring and reached itself. Hence it declares itself as Leader.
  This is the working of LCR Algorithm. As we can see, the time complexity is O(n) and the message complexity is n(n-1)/2 messages. Asymptotically, $O(n^2)$ messages.

  Hirschberg-Sinclair Algorithm was proposed as a modification of LCR. The time complexity is linear but the message complexity is quadratic and hence

they came up with an idea to reduce the number of messages. The basic ide is to take a Binary Search Approach.
Instead of passing the process ID only to the right neighbors, each node passes its process ID to 2^k neighbors on both sides (left and right), k goes from 0 to log n. Thus in each phase, each node competes to be a leader. On nodes that become leaders (a node which is dominant among 2^k neighbors on left and right side) goes to the next phase.

Though the time complexity is the same, O(n), the message complexity gets reduced to O(n log n) due to Binary Search Approach.

WORKING OF HIRSCHBERG-SINCLAIR:

- This algorithm works in phases *0, 1, 2, . . . , O(log n)* in a bidirectional ring.
- Let u_i be the id of node *i*.
- In phase *l*, each node *i* sends its id in a token out to a distance 2^l that then returns to node *i*.
- If node *i* does not receive both its tokens back in phase *l*, it does not send out its id any more.
- In phase *l*, if a node *j* receives node *i*'s token and $u_i < u_j$, node *j* discards the token and does not pass it along any more.
- If $u_i = u_j$, then node *j* elects itself leader and sends a final round of messages informing every other node of its leader status.

MESSAGE COMPLEXITY OF HIRSCHBERG-SINCLAIR ALGORITHM:

- Consider any winner *w* that survives to phase *l*. At least $2^{l-1} + 1$ nodes on, say, its counterclockwise side do not survive to phase *l* and can be accounted against *w*. Therefore, the number of winners in phase *l* is at most $\lfloor n/(2^{l-1} + 1) \rfloor$.
- Each winner's tokens account for $4 \cdot 2^l$ messages.

- Therefore, total number of messages in phase *l* is $4 \cdot 2^{l} \cdot \lfloor n / (2^{l-1} + 1) \rfloor \leq 8n$.
- Since the total number of phases is *O(log n)*, the message complexity i *O(n log n)*.

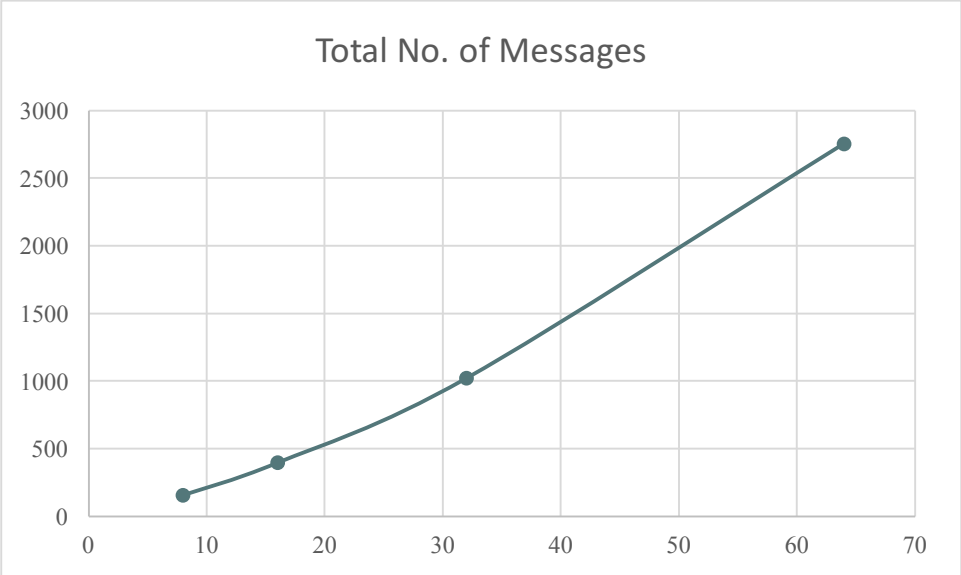
TIME COMPLEXITY OF HIRSCHBERG-SINCLAIR ALGORITHM:

- Each phase *l* requires $2 \cdot 2^{l} = 2^{l+1}$ rounds.
- The last but one phase requires $2^{O(\log n)} = O(n)$ rounds and the last round requires *n* rounds.
- Therefore, total number of rounds is $2 + 4 + 8 + \cdots + O(n) + n = O(n)$

ANALYSIS:

All values are averaged over 2 runs.

No. of Nodes (n)	No. of Phases (log n)	Average No. of Messages in each Phase (Must be <= 8n)	Total No. of Messages (Must be approximately constant order of n log n)
8	3	57 (<= 64)	158 (≈ 7.125 x 24)
16	4	106 (<= 128)	397 (≈ 6.625 x 64)
32	5	235 (<= 256)	1021 (≈ 7.344 x 160)
64	6	489 (<= 512)	2754 (≈ 7.641 x 384)



LEGEND:

x – axis: No. of Nodes

y – axis: Total No. of Messages