Our project is an implementation of leader election algorithm using sense of direction. The algorithm is a work of Gurdip Singh. First let me introduce the leader election problem in this paper. As we learned in class, leader election is the problem that, a group of processors must choose one among them to be the leader. In the paper, the leader election algorithm runs on an asynchronous complete graph. And there is bidirectional link between every two nodes. The algorithm is based on sense of direction provided by a Hamiltonian cycle in the network. By sense of direction, I mean first, every processor can distinguish between different channels of itself; second, each processor knows the distance from it to its neighbors across the Hamiltonian cycle. For example, the distance from a to c on Hamiltonian cycle is 2, then a labeled this channel as 2; the distance from a to d is 3, then a labeled this channel as 3. Later when we talk about distance, we always mean distance across the Hamiltonian cycle. However, processor does not know the id of its neighbors, they distinguish their neighbors by the distance labeled on the channel. The time complexity of the algorithm is O(logn), and the message complexity is O(n), which are both better than the two leader election algorithms we previous learnt.

Initially, the algorithm assume number of processors is power of 2, later we will discuss the situation when n is not power of 2. And each node has 3 states: candidate, captured, elected. Candidate nodes may be a leader in the feature, captured nodes is out of game and elected nodes is the leader. These figure shows the network with n = 64. For the sake of simplicity, We just draw the Hamiltonian cycle of the graph here. Every processor runs the algorithm in 2 phases. In phase one, the cycle is separated into k subrings interleafed. And k is given by this formula. Every n/k nodes form a subring. For example, here is n is 64 and k is 8, so n/k is 8. Node number 0, 8, 16, 24, 32, 40, 48, 56 forms a ring. Initially some processors wake up spontaneously and become a candidate, some wakes up by receiving a message and becomes captured. Captures messages are sent by one node to another to compare two nodes and determine whose is the winner. In phase 1, a leader is elected within every subring. In phase two, every phase 1 leader sends message to nodes in other rings, and get information about who’s other ring’s leader. Then try to capture other leaders. The last winner among k leaders is the final leader. To reduce message and time complexity, the send message in special ways. In phase 1, candidates send message in the order of distance, from near to far. And Candidate captures other nodes by a method called inherit. For example, this is within a subring, j already captures node as far as y hoops away. And now i wants to capture j, j informs i of y, and i will automatically captures the node j already captures without sending them extra messages. This is where sense of direction works, i will know which channel leads to nodes within the distance of x+y. In phase 2, phase 1 leader sends capture messages to other nodes in other rings step by step. The number of messages send per step increases exponentially. For example, in step 1 leader of ring 0 sends message to ring 4, in step 2, it sends to ring 2, 6; in step 3, it sends messages to ring 1,3,5,7. By this way, the message and time complexity is reduced to O(n) and O(logn) respectively.

Here are some local variables in the algorithm, state and phase are as we mentioned above. Level are used in phase 1 and step are used in phase 2 counts how many nodes are captured respectively in phase 1 and 2. Owner stores the winner id of phase 1. And here are messages been send. CAPTURE is send by candidate and receiver compare the tuple by lexicographical order. If loses it send ACCEPT to candidate. When phase 1 ends, owner is send by winner in its subring to inform its id. ACK is send to respond to OWNER message. In phase 2, INFORM is used to inform candidate in other rings about phase 1 leader. ELECTED is send in the end to inform everyone final leader id.

Now we go ahead to talk about our implementation. During the implementation, we fixed some errors in the pseudocode. For example, some indents are wrong, leading to some misunderstanding of the logic of algorithm; some messages content are wrong and leading to infinite loops or ties in execution. We also clarified ambiguous description such as unmentioned state changes. Last but not least, we extended the algorithm to be suit for n that is not power of 2, although the paper only mentioned one paragraph about it in the end.

There are some troubles when n is not power of 2. Here Rj means the ring that node j belongs to. When n is power of 2, if i belongs to R\_j, then j belongs to R\_i. When n is not power of 2, it’s not the case. For example, if n is 10, k is 4. When construct rings in phase 1, R\_0 consists node of {0, 4, 8}, but R\_4 dose not includes node 0. So here in phase 1, we extend the subring by counting k nodes backwards. For example, R\_0 originally consists node of {0, 4, 8}, and we then counting k nodes backwards from R\_0, then the ring is extended to node of 2 and 6. In phase 2, when phase 1 winner sends capture messages to others, it also extend its sender in this way but skip those who is already send a message in phase 1.

Now let’s move on to the experiment. We verifies the time and message complexity of the algorithm. We run the code on n nodes, which ranges from 2 to 100. We repeat the test for 10 times to reduce random error. As DistAlgo do not support access to nodes attributes and global variable, we print out time and message and analyze them. Experiment on message complexity is shown here. As the fitting function shows, number message grows linearly to number of processors, so we verified the O(n) message complexity. And for time complexity, we set all message delays to be 0.1 seconds The time-processor relation is shown in this figure. As the fitting function shows, time grows proportional to log(n). And the R-square value for fitting is 0.8157. However when we add a term of n into fitting function, the R-square value increases to 0.8214, which means this is more proper for describing time growth. We think this is because theoretically, computational time is ignored. But in practice, computational time cannot be ignored and grows linearly to number of processors. The experiment data contains some errors in it coming from random variable in the code, unstable computer performance. Also if we increase number of processor and number of repeats it would be less uncertainties in the data.

And our assessment of DistAlgo, we think enables us to simulate multiple processors on a single machine and send messages between each other. However, it does not allow us to access attributes or functions of nodes, nor compile global variables, which is not convenient for doing experiments. We find that during experiments, many executions seem to be the same, so maybe DistAlgo offers low-quality of randomness. And we hope it can have API to directly choose from asynchronous and synchronous message passing.

As a conclusion, in the project, we found that Gurdip Singh’s algorithm solves leader election using a sense of direction on a complete network. We implemented the algorithm in DistAlgo and

We verified the time complexity to be O(log(n)), message complexity to be O(n). In the future, we think we can Simplify our code, test on a bigger network with more processors, and with arbitrary message delay within a certain interval, which is more similar to the delay in real world is like.

Dear Professor Welch,

We are team with member of Yiran Huang, Guanda Li, Peixin Liu. Attached is our presentation slides. Thank you!

Best regards,

Peixin Liu