Assignment 1: Coordination and Leader Election Simulating and Evaluating Distributed Protocols in Java

The goal of this assignment is to write two algorithms for leader election and check their consistency and correctness. The network model defined throughout this coursework is a bidirectional ring; in other words, each processor can move in both directions, clockwise and counterclockwise. The first leader election algorithm is named “LCR algorithm”. It moves clockwise and finds the processor with the highest ID to be the leader. The second leader election algorithm or “HS algorithm” has the same goal as the LCR algorithm except that it sends the message in both directions. This report will explain the implementation of both algorithms, explain the main functionalities of the code, and discuss their correctness and performance.

1. Implementation of the LCR algorithm

The code is divided into four parts with two shared classes for both LCR and HS algorithms. The message class defines the structure of a message in the processor. It has the number of rounds (initially set to 1) the message has been through, the total number of messages, the ID of the message, the direction of the message (either IN or OUT) which is mostly used in HS algorithm, the hopcount which is also only used in the HS algorithm. The processor class defines the different components of each processor of the network. It has an ID, a sendID (the ID that will be sent every round), a status (either “unknown” or “leader”), the previous and the next processor since it needs to know those information to send data, the previous and next message which are the messages received counterclockwise or clockwise, the phase for the HS algorithm, the message to send clockwise and the message to send counterclockwise.

For the LCR implementation, there are two methods in the processor class: sendMessageClockwise() and LCRprocess(). As the name suggests, sendMessageClcokwise() is in charge of sending the processor. The way I did was by taking the previous processor and checking if it has a message to send in sendClock. If it does, then update my previous message to the sendClock message of the previous processor. Then, it is important to update that previous value to null so it does not send two times. Finally, increase the number of messages every time the message is sent. LCRprocess() defines the process to figure out if that processor is the leader. It follows the pseudocode defined in the assignment paper.

The class LCRAlgorithm illustrates the leader election but for all the processors. The first step is to create a LinkedList of processors. Then, for all the processors, go through the LCRprocess() and if the leader is found, break the loop to avoid sending too many messages. One round is defined by checking if the leader is there after sending messages.

2. Implementation of the HS algorithm

The ring used for the HS algorithm is bidirectional which means that it sends the message counterclockwise and clockwise. Therefore, in the processor class, there will be an additional method called sendBidirMessage() which will send the message to both directions. The way it works is as follows. First, do the same as in sendMessageClockwise().

Then go over the same process again but this time, with the next processor in the list, sendCouterClock and nextMessage.

The HSprocess() method is to check if the current processor is the leader. This part will follows the pseudocode given in the assignment sheet.

The class HSSimulator contains the method that will reiterate the process in a loop so that it checks for every processor and sends the message if the leader has not been found yet. It will also call the createprocessorlist() function defined in the processor class.

3. Experimental evaluation, comparison, and report

3.1. Random list

Size of ring	number of messages LCR	number of messages HS	rounds LCR	rounds HS
10	25	132	11	41
20	65	322	21	83
100	516	2030	101	355
200	1093	5068	201	711
300	2201	8746	301	1323
400	2884	11336	401	1423
500	3311	13790	501	1523
600	3735	19228	601	2647
700	4764	21364	701	2747
800	5096	25759	801	2847
900	6480	26499	901	2947
1000	6488	31935	1001	3047

Table 1: Collected values for randomized ring

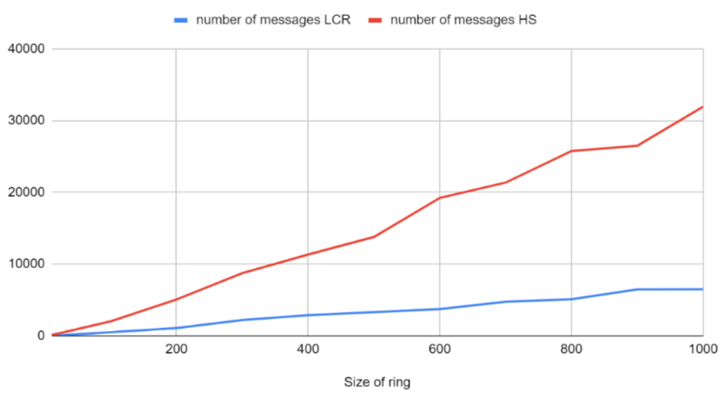

Figure 1: Number of messages in LCR and HS in function of the ring size

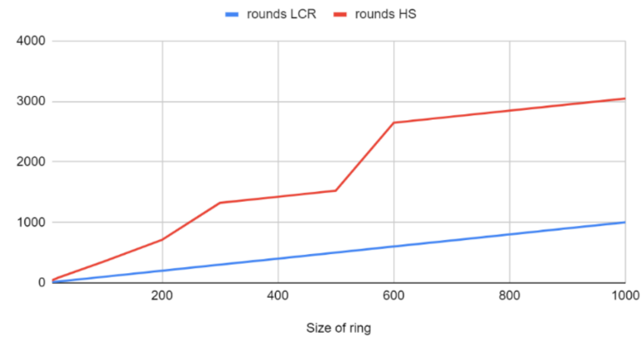

Figure 2: Number of rounds in LCR and HS in function of the ring size

3.2. Ascending

Size of ring	number of messages LCR	number of messages HS	rounds LCR	rounds HS	phase HS
10	19	106	11	41	4
20	39	220	21	83	5
100	199	1004	101	355	7
200	399	2016	201	711	8
300	599	3540	301	1323	9
400	799	4040	401	1423	9
500	999	4540	501	1523	9
600	1199	7088	601	2647	10
700	1399	7588	701	2747	10
800	1599	8088	801	2847	10
900	1799	8588	901	2947	10
1000	1999	9088	1001	3047	10

Table 2: Collected values for an ascending ring

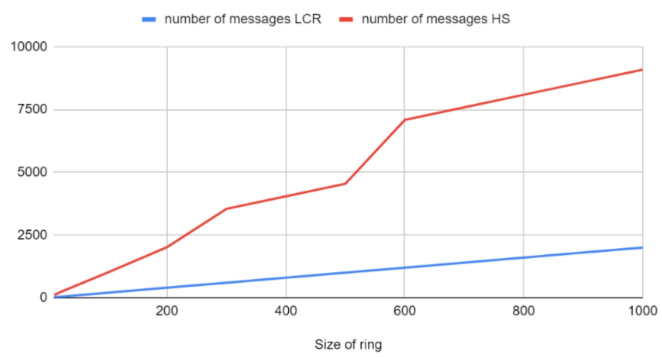

Figure 3: Number of messages in LCR and HS in function of the ring size

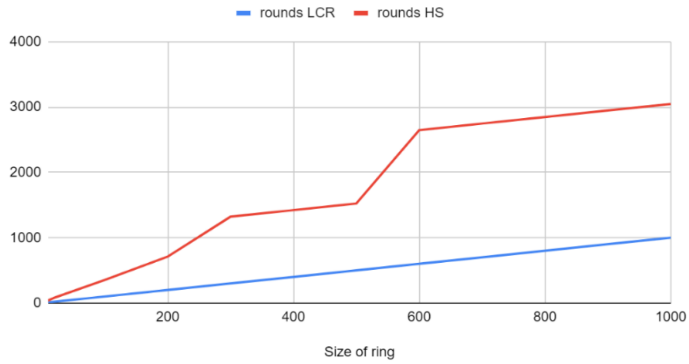

Figure 4: Number of rounds in LCR and HS in function of the ring size

3.3. Descending

Size of ring	number of messages LCR	number of messages HS	rounds LCR	rounds HS	phase HS
10	55	106	11	41	4
20	210	220	21	83	5
100	5050	1004	101	355	7
200	20100	2016	201	711	8
300	45150	3540	301	1323	9
400	80200	4040	401	1423	9
500	125250	4540	501	1523	9
600	180300	7088	601	2647	10
700	245350	7588	701	2747	10
800	320400	8088	801	2847	10
900	405450	8588	901	2947	10
1000	500500	9088	1001	3047	10

Table 3: Collected values for a descending ring

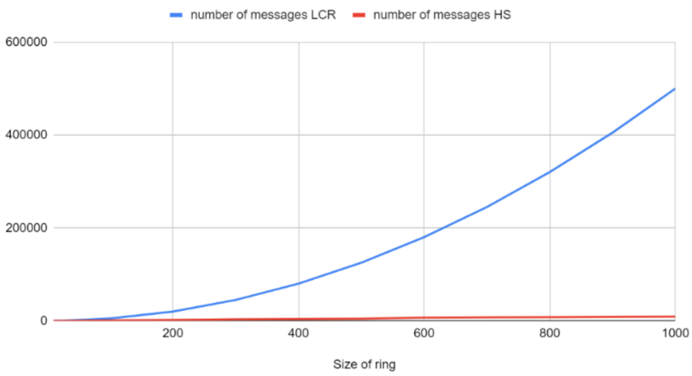

Figure 5: Number of messages in LCR and HS in function of the ring size

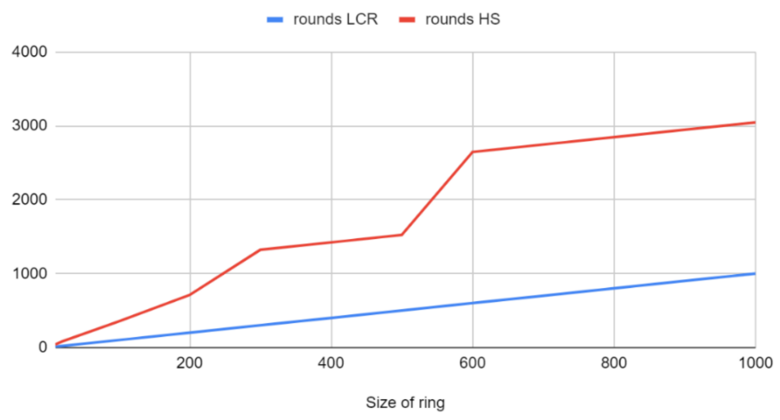

Figure 6: Number of rounds in LCR and HS in function of the ring size

3.4. Results and discussions

In theory, the number of rounds in LCR for a ring should be O(n) or O(n+1). While simulating the leader election algorithm for all three ring types, the number of rounds for LCR is always n+1. Therefore, the simulation for the number of rounds is accurate for LCR.

The number of rounds in HS should be smaller than the number of rounds in LCR because the message is sent both clockwise and counterclockwise which means that the message should be found quicker. However, according to figures 2, 4, and 6, the red function which represents HS is always above the blue function representing LCR. Since LCR is correct, the problem could be in the round count of HS.

When it comes to the number of messages sent, the situation is slightly different. In a randomized ring, the number of messages in LCR is smaller than in HS. That can be explained by the fact that an HS algorithm sends messages in both directions. For an ascending ring, the situation is the same except that the function for HS has more disruptions as it can be seen in figure 3. For a descending ring, the number of messages in HS is smaller than the number of messages in LCR. In a descending ring, as seen in figure 5, the number of messages in LCR is higher than in HS. The message complexity in LCR is increasing at a high rate. That can be explained by the fact that the message needs to be sent many times before arriving at the end of the ring which contains the highest value and ten do a full circle again.

Conclusion:

To conclude, both simulations find a unique leader. However, there may be a problem in the counting of rounds in the HS algorithm and a problem for counting messages in both simulations. This assignment was challenging because the bigger the ring size is, the harder it becomes to check if the values are right.