

University of Ottawa

CSI5308 course

Course Project

Election in Ring

Emad Badawi

Ebada090@uottawa.ca

https://github.com/Ebadawi/CSI5308-Project

**Idea**

In this project, the C++ and Message Passing Interface were exploits to implementing MinMax, MinMax+, all the way and as far election algorithm in unidirectional rings, and capture the number of exchanged messages in each one of the algorithms, then compare the acquired messages number with the worst case complexity for each of the algorithms.

**System Configuration and Platform Details**

In the conducted experiments, a cluster of 5 computers were used. Each of the cluster nodes has different HW/SW specifications. Table 1 contains the used cluster HW specifications.

Table 1System Configuration & Platform Details

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Computer name** | **Kingpenguin** | **Dune-970** | **Dune-770** | **Dune-Frg** | **Setup-T3600** |
| **CPU** | Intel(R) Xeon(R) CPU E5-2640 @ 2.50GHz | Intel(R) Core(TM) i7-5820K CPU @ 3.30GHz | Intel(R) Core(TM) i7-4820K CPU @ 3.70GHz | Intel(R) Core(TM) i7-5820K CPU @ 3.30GHz | Intel(R) Xeon(R) CPU E5-1620 0 @ 3.60GHz |
| **CPU Cores** | 12 | 12 | 4 | 6 | 4 |
| **Threads / Core** | 2 | 2 | 2 | 2 | 2 |
| **RAM** | 64 GB | 32 GB | 32 GB | 32 GB | 16 GB |
| **GPU** | - | GeForce GTX 970 | GeForce GTX 770 | - | Quadro 2000 |
| **GPU Cores** | - | 1664 | 1536 | - | 192 |
| **GPU RAM** | - | 4GB | 2GB | - | 1GB |
| **Compute Capability** | - | 5 | 3 | - | 2 |
| **Number of GPUs** | - | 2 | 2 | - | 1 |

The software environment is the same for all the test beds in **Table** 1 and is mentioned below:

* Operating System: Ubuntu 16.04.1 LTS (64 bit)
* The GNU C++ 4.9.3 compiler.

Election algorithm was executed using MPI parallel implementation on a NoW formed by the machines shown in Table 1.

**Implemented Algorithms**

In all of the implemented algorithms, the number of lunched processes represents the number of nodes in the rings. The connections between the nodes is identified by the fact that MPI processes have distinct IDs (process rank), in which these IDs can be viewed as a ring. Figure 1 represents 8 lunched processes connected with each other in such a way to create ring. The processes IDs can be used as a distinct IDs for the corresponding nodes, but this will limit the simulation with one case, which is the ascending order or descending order. Another way to assign distinct IDs is to use a master process to generate random distinct IDs and distribute them between the other processes. The later way of assigning IDs was used.

1. **All the way**

This is the easiest algorithm, here each node sends two values to the neighbor node, the first is the node ID and the second as a counter to count how many nodes the ring consists from. Each node has a local counter to count how many nodes passed by, and has a minimum value equals to its ID initially, this minimum value will be updated if a node that has smaller value passed by. When a node receives its value back, it will know how many nodes in the ring using the counter value traversed with the ID. When the number of passed nodes equals the number of nodes in the ring, I’ll be the leader if my min value equals my ID and I’ll be follower otherwise.

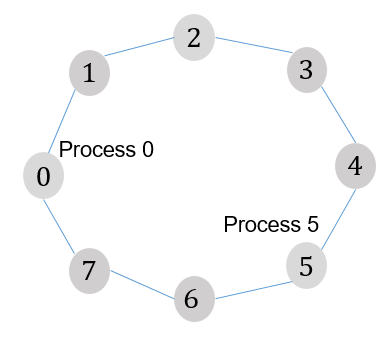


Figure 1: Ring constructed using MPI processes

1. **As Far**

This is the a variation of All the Way algorithm, in which each node sends its value to its neighbor, but the main difference that the message will be stopped if its counter a node with smaller ID in the way. The termination in this algorithm is achieved when a node receives its value back, which means that I did not counter any node with smaller ID than mine, thus I am the leader. Here a termination message is required to be send to the other nodes telling them that the algorithm terminated and I am the leader.

1. **Min Max**

This is the a variation of electoral stages, in which each node sends its value to its neighbor, but the main difference that the nodes elimination depends on two factors, the nodes ID’s as well as the current stage, thus in the message we send the stage in addition to the ID. In particular, in odd stages I’ll be eliminated if I receive a message from node with ID smaller than mine, while in even stages I’ll be eliminated if I receive a message from node with ID bigger than mine while. In this algorithm, it’s not necessary for the leader to be the maximum or minimum value, any value might survive until the last stage and become the leader. Here a termination message is required to be send to the other nodes telling them that the algorithm terminated and I am the leader.

1. **Min Max +**

This is the a variation of MinMax, in which we permit the passive nodes to give decisions. In particular, in even stages the nodes travers a specific range of passive nodes, and if the maximum number of possible nodes to travers is reached before arriving to the next candidate, the passive node that I am in will be activated and start the next stage. Also, if I got defeated in an even stage, I will save my current stage and the value I have send for later use by odd stages. In the other hand, in odd stages, the nodes that have been defeated in even stages will be activated and start the next stage if they receive a message with smaller value than the one they have sent before they got defeated. In both cases, even and odd stages, if a node receives a message from higher stage than the one it was waited, it will be defeated and forward the message to the next node. In this algorithm, it’s not necessary for the leader to be the maximum or minimum value, any value might survive until the last stage and become the leader. Here a termination message is required to be send to the other nodes telling them that the algorithm terminated and I am the leader.

**Algorithms Execution**

To build the algorithm:

mpic++ ./code\_file.cpp -o executable\_file

For Example, to build AS Far code: mpic++ ./AsFar.cpp -o asfar

To run the algorithm:

mpirun -np N --hostfile hosts ./ executable\_file

For Example, to run AS Far executable file: mpirun -hostfile hosts -np 30 ./asfar

Where:

N: represent the number of lunched processes (the number of nodes)

Hosts: a file that contains the IPs of the used cluster (1 IP per line)

**Experimental Results**

In the conducted experiments, I have used different ring with different sizes in the range {50,250} with a step of 50. The final result was derived as the average of 5 different execution for each algorithm in each possible value of the ring size. Figure 1 represents the number of exchange messages verses the ring size, while the execution time for the algorithm is discussed in Figure2. As presented in Figure 1, all the way protocol has the highest number of exchanged messages, while the other three protocols has a comparable values. As far protocol has the lowest number of exchanged messages up to 150 nodes, after this point, it became a little higher than MinMax+ protocol.

Figure 1: Exchanged Messages versus Number of Nodes

Figure 1: Execution Time versus Number of Nodes

As for the exaction time, All the way protocol has the least execution time followed by as far and MinMax+ protocols, while MinMax protocol have the highest execution time.

**Conclusion**

In this project, a comparison between four election algorithms was conducted. In this comparison, we choose the worst known algorithm (all the way), the best known algorithm (MinMax+) and another two algorithms which are as far and MinMax. The conducted experiments showed proved that all the way algorithm exchange the the maximum number of messages to elect the leader in the ring, in the other hand it was the easiest to implement and the fastest algorithm as well. For MinMax+ algorithm, we found that it is very comparable to as far in terms of messages exchanged, for small numbe rof nodes its exchanged more messages than as far, but with increasing the number of nodes in the ring, the number of exchanged message to elect the leader were less than as far. In terms of implementation, MinMax+ was the hardest and most complicated algorithm, also it has the second maximum execution time after MinMax. The worst execution time was for MinMax algorithm and it has the second highest exchanged messages.

In conclusion, choosing which algorithm to implement depends closely on the needed requirement, if we want the smallest number of exchanged messages, MinMax+ and as far will be our choice. If we want the fastest algorithm all the way will be the easiest and best to do. While if we want a tradeoff between the number of exchanged messages and the execution time, as far is our best choice. In my opinion, as far is the best algorithm to go with, due to its performance and simplicity.