Test case report

Afek and Gafni for election in asynchronous complete networks

Jeffrey Goderie 4006232

Alexander Simes 4415299

Test case 1:

For this test case we used only one candidate process, and an arbitrary amount of ordinary processes. The expected pattern is that the candidate process will go over all the ordinary processes and captures them along the way.

Log transcript:

```
Process 500 sent message: Level 0, ID 500 to Process 204 (Ordinary)
Process 500 Level 0 has received message: Level 0, ID500
Process 500 has received acknowledgement for its own message
Process 500 sent message: Level 1, ID 500 to Process 206 (Ordinary)
Process 500 Level 1 has received message: Level 1, ID500
Process 500 has received acknowledgement for its own message
Process 500 sent message: Level 2, ID 500 to Process 362 (Ordinary)
Process 500 Level 2 has received message: Level 2, ID500
Process 500 has received acknowledgement for its own message
...
Process 500 sent message: Level 497, ID 500 to Process 254 (Ordinary)
Process 500 Level 497 has received message: Level 497, ID500
Process 500 has received acknowledgement for its own message
Process 500 sent message: Level 498, ID 500 to Process 221 (Ordinary)
Process 500 Level 498 has received message: Level 498, ID500
Process 500 has received acknowledgement for its own message
Process 500 sent message: Level 499, ID 500 to Process 133 (Ordinary)
Process 500 Level 499 has received message: Level 499, ID500
Process 500 has received acknowledgement for its own message
Process 500 Level 500 has been elected.
```

Total Messages *(N = amount of ordinary processes)*:
N Capture Requests Sent/Received
N Acknowledgements Sent/Received

Maximum Level *(N = amount of ordinary processes)*: N

Number of captures: Each process is only captured once, as the candidate process only sends a request to the processes still in its request set, and after an acknowledgement it is removed from this set. There is only one candidate process, so each ordinary process receives a maximum of 1 request.

Test Case 2:

For this test case we will look into the situation where there is only are only 2 candidate processes, both starting at the same time.

Log Transcript:

```
Process 1 sent message: Level 0, ID 1 to Process 0
Process 0 sent message: Level 0, ID 0 to Process 1
Process 1 Level 0 has received message: Level 0, ID0
Process 0 Level 0 has received message: Level 0, ID1
Process 0 is now stuck
Process 0 is now killed
Process 1 Level 0 has received message: Level 0, ID1
Process 1 has received acknowledgement for its own message
Process 1 Level 1 has been elected.
```

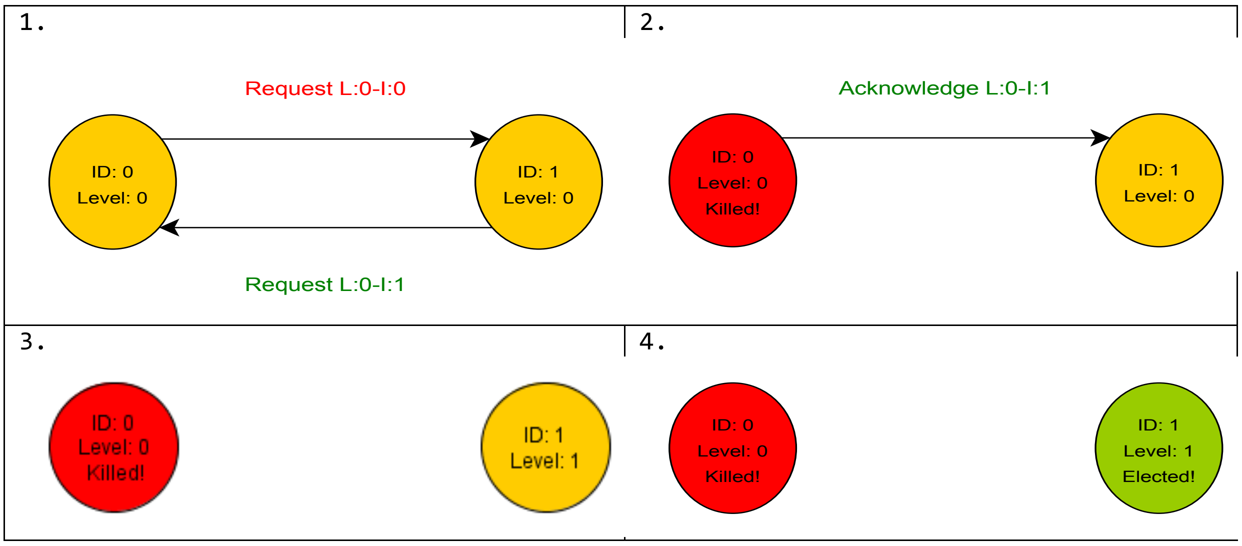
Process 1 sent a message to process 0, and process 0 sent a request message to process 1. Upon receiving the message from process 0, process 1 does nothing as the ID is lower and the levels are equal, and decides to ignore it (causing P0 to get stuck). Process 0, receiving process 1's original message, is killed, and acknowledges this to process 1. Process 1 then receives this acknowledgement, and as its request set is empty now, it is elected.

Total Messages:
2 Capture Requests Sent/Received
1 Acknowledgements Sent/Received

Maximum Level: 1, only one acknowledgement has been received.

Number of captures: As there are no ordinary processes there are no captures. There is a kill though. The amount of kills is equal to N-1, where N is the amount of candidate processes.

Algorithm steps:



Test Case 3:

This test case involves 2 candidate processes and 2 ordinary processes, where a total of 4 claim attempts are done, of which 3 succeed.

Log Transcript:

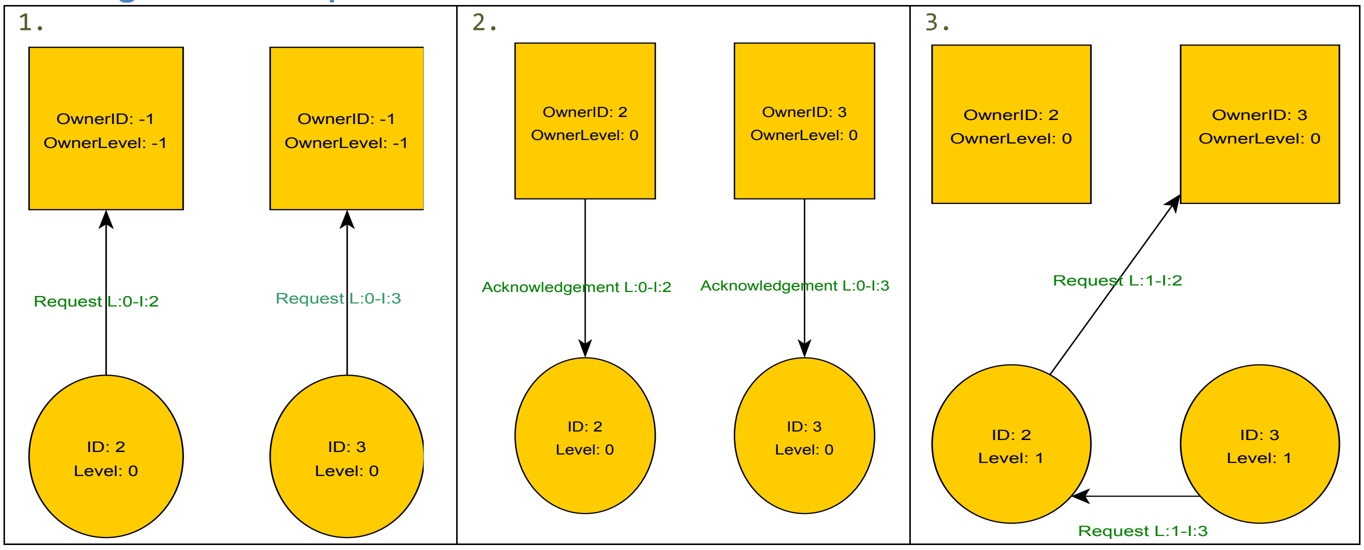
```
Process 2 sent message: Level 0, ID 2 to Process 1 (Ordinary)
Process 3 sent message: Level 0, ID 3 to Process 0 (Ordinary)
Process 3 Level 0 has received message: Level 0, ID3
Process 3 has received acknowledgement for its own message
Process 2 Level 0 has received message: Level 0, ID2
Process 2 has received acknowledgement for its own message
Process 3 sent message: Level 1, ID 3 to Process 2
Process 2 Level 1 has received message: Level 1, ID3
Process 2 is now killed
Process 2 sent message: Level 1, ID 2 to Process 0 (Ordinary)
Process 3 Level 1 has received message: Level 1, ID3
Process 3 has received acknowledgement for its own message
Process 3 Level 1 has received message: Level 1, ID2
Process 3 sent message: Level 2, ID 3 to Process 1 (Ordinary)
Process 2 is now stuck
Process 2 Level 1 has received message: Level 2, ID3
Process 2 was already killed
Process 3 Level 2 has received message: Level 2, ID3
Process 3 has received acknowledgement for its own message
3 has been elected.
```

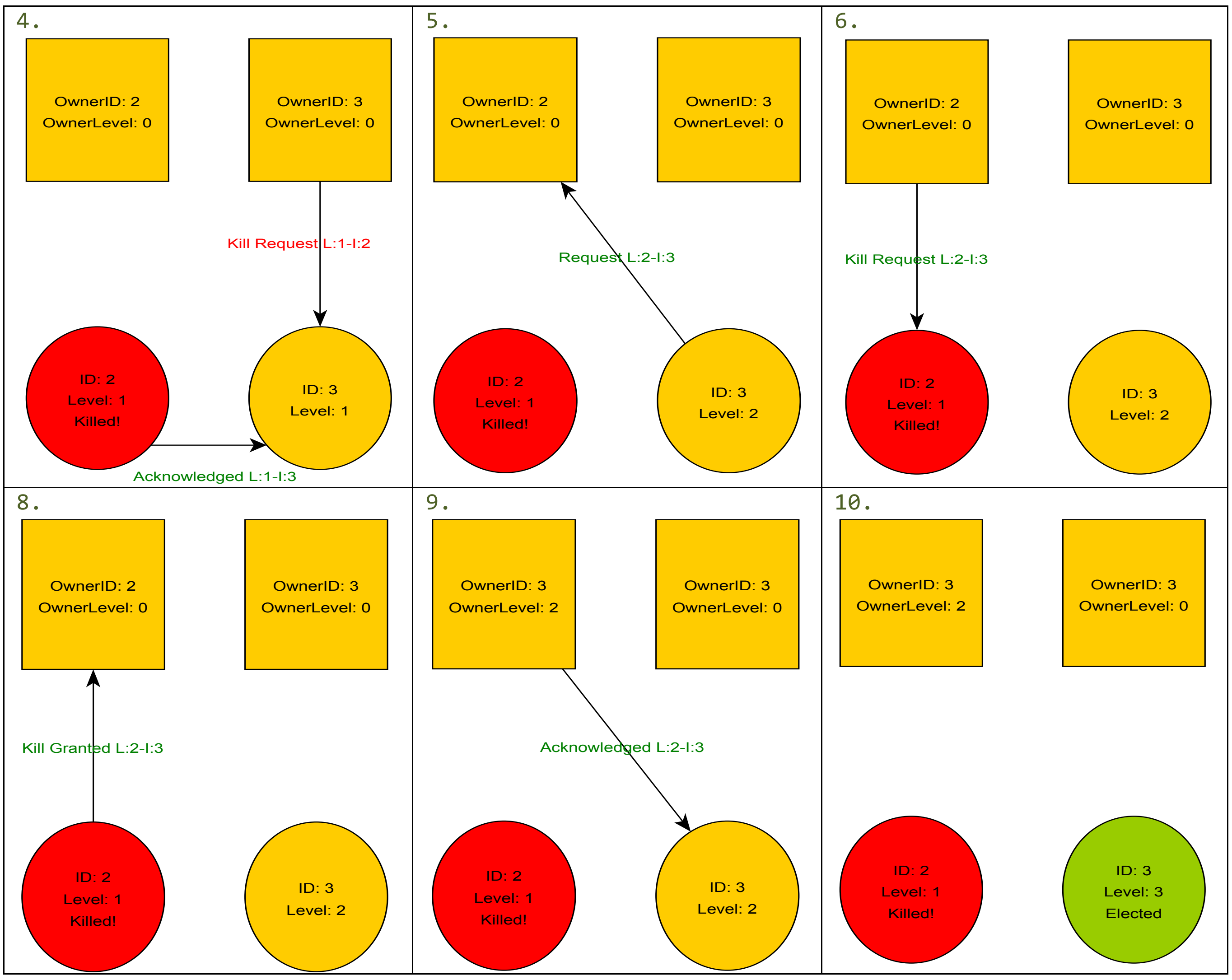
Total Messages:
5 Requests Sent/Received
3 Successful Captures
2 Kill Messages Received (1 was a re-kill)
4 Acknowledgements Sent/Received

Maximum Level: 3

Number of captures: One of the ordinary processes was captured 2 times, while the other one was being captured a second time, but this attempt failed. Both kill messages were received by process with ID 2, one directly from process ID 3, and one as a result of a capture + kill request.

Algorithm Steps




4.
OwnerID: 2 OwnerLevel: 0
OwnerID: 3 OwnerLevel: 0
Kill Request L:1-I:2
ID: 2 Level: 1 Killed!
ID: 3 Level: 1
Acknowledged L:1-I:3
5.
OwnerID: 2 OwnerLevel: 0
OwnerID: 3 OwnerLevel: 0
Request L:2-I:3
ID: 2 Level: 1 Killed!
ID: 3 Level: 2
6.
OwnerID: 2 OwnerLevel: 0
OwnerID: 3 OwnerLevel: 0
Kill Request L:2-I:3
ID: 2 Level: 1 Killed!
ID: 3 Level: 2
8.
OwnerID: 2 OwnerLevel: 0
OwnerID: 3 OwnerLevel: 0
Kill Granted L:2-I:3
ID: 2 Level: 1 Killed!
ID: 3 Level: 2
9.
OwnerID: 3 OwnerLevel: 2
OwnerID: 3 OwnerLevel: 0
Acknowledged L:2-I:3
ID: 2 Level: 1 Killed!
ID: 3 Level: 2
10.
OwnerID: 3 OwnerLevel: 2
OwnerID: 3 OwnerLevel: 0
ID: 2 Level: 1 Killed!
ID: 3 Level: 3 Elected

Test Case 4:

One test case to consider is the one where the predicted winner is the only one actively participating in the algorithm. One would expect that this setting results in the same test case as Test Case 1, as the other candidate processes acts merely as relays when they are not active in the algorithm. One slight difference is that the processes that are candidates are getting killed.

Log Transcript:

```
Process 999 sent message: Level 0, ID 999 to Process 287 (Ordinary)
Process 999 Level 0 has received message: Level 0, ID999
Process 999 has received acknowledgement for its own message
Process 999 sent message: Level 1, ID 999 to Process 303 (Ordinary)
Process 999 Level 1 has received message: Level 1, ID999
Process 999 has received acknowledgement for its own message
Process 999 sent message: Level 2, ID 999 to Process 917
Process 917 Level 0 has received message: Level 2, ID999
Process 917 is now killed
Process 999 Level 2 has received message: Level 2, ID999
Process 999 has received acknowledgement for its own message
...
Process 999 sent message: Level 997, ID 999 to Process 953
Process 953 Level 0 has received message: Level 997, ID999
Process 953 is now killed
Process 999 Level 997 has received message: Level 997, ID999
Process 999 has received acknowledgement for its own message
Process 999 sent message: Level 998, ID 999 to Process 456 (Ordinary)
Process 999 Level 998 has received message: Level 998, ID999
Process 999 has received acknowledgement for its own message
999 has been elected.
```

Total Messages *(N=Candidate Processes, M=Ordinary Processes)*:
(N-1) + M Capture Requests Sent/Received, in this case 999
(N-1) + M Acknowledgements Sent/Received, in this case 999
(N-1) Candidate Processes Killed, in this case 499
M Ordinary Processes Captured, in this case 500

Maximum Level *(N=Candidate Processes, M=Ordinary Processes)*: (N-1) + M, in this case 999.

Number of captures: Each process is only captured once, as the candidate process only sends a request to the processes still in its request set, and after an acknowledgement it is removed from this set. There is only one candidate process, so each ordinary process receives a maximum of 1 request. Each of the other candidate processes is killed once, so there are 499 kills.

For this test case we made the assumption that if a process gets killed before it starts sending, it no longer is allowed to send its first message, as it can't be elected winner anymore anyways. If this assumption is not made, the amount of requests will become much higher.

Load Test Case:

For this test case we do a load test of our implementation. The bottleneck that we encountered was a StackOverflow error in the RMI. This could be due to physical memory running out, but it limits the load test.

We were able to increase the amount of candidate processes (with no ordinary processes) to a total of 4000 before running into problems.

When introducing ordinary processes the capacity is reduced, possible due to the larger chunk of memory an ordinary process takes. It would be possible that instead of storing the owner object storing the identifier of this object would allow for bigger sets. If the amount of ordinary processes nears 1000, the max capacity of candidate processes is around 1000 as well.

Overall Test Cases:

Candidate Processes	Ordinary Processes	Requests/Kills Received	Acknowledgements Received	Processes (Re-)Killed	Processes Captured	Max Level
10	**0**	26	17	19	0	9
100	**0**	385	315	331	0	99
1000	**0**	6600	6032	6147	0	999
3000	**0**	19425	18010	18363	0	1999
10	**10**	38	29	22	18	19
100	**10**	366	283	288	22	109
10	**100**	131	125	28	119	109
100	**100**	448	358	280	192	199
500	**500**	2023	1716	1268	908	999

If the algorithm was synchronous, one would expect to see the same amount of acknowledgements as (re-)killed messages, but due to the asynchronous structure it can occur that a process is killed between the moment where it sent the request, and receives the grant, which leads to the acknowledgement being discarded.

As can be seen from the table the amount of messages being sent increases very fast when the amount of processes increases. On average ordinary processes get captured twice, though this is probably more inverse-exponentially distributed.