Alistarh et al. (2015) — "How to Elect a Leader Faster than a Tournament"

November 21, 2017; rev. November 30, 2017

Lily Li

1 Initial Board Configuration

Problem Definition. An algorithm which solves leader election among n processes must satisfy: (1) termination - every non-faulty process p_i must eventually output win or $lose$ and (2) unique winner - only one process may output win. Further, no process may lose before the eventual winner starts its execution.

Model. A complete asynchronous message passing system with at most $f < \lceil n/2 \rceil$ faulty processes. The randomized algorithm we will presents works against a strong(adaptive) adversary.

2 Introduction

Today I will present "How to Elect a Leader Faster than a Tournament" by Alistarh et al. (*Gelashvili, and Vladu*) which appeared in the proceeding of PODC 2015.

Move to the left board with the prepared material.

The problem is define as follows: *Read material on "Problem Definition".*

To ensure that operations are linearizable, the first operation is win and all subsequent operations are $lose$, and the order of the non-overlapping operations is respected. Observe that this problem is equivalent to implementing a *Test-and-Set* object. The *winner* is analogous to the process which receives 0 from the $Test\&Set$ call and the *losers* are analogous to the processes which receives 1 afterwards. Hopefully everyone remembers our discussion of $Test\&Set$ objects earlier in the semester.

Our model is: *Read material on "Model".*

To appreciate the results obtained in this paper, we first survey related work. *Change Slide.* The best known previous algorithm, by Afek et al. in '92, makes use of a *tournament tree*. In-class, we have a seen *tournament trees* in the context of bounded mutual exclusion. The n processes are the leaves of the tree. Processes compete at each internal nodes and the survivor proceeds up the tree. The *winner* (resp. process which gets to enter the critical section) is the survivor at the root node. This algorithm has $O(\log n)$ step complexity and uses $O(n \log n)$ registers. In the context of a message passing system, we can use the ABD algorithm to simulate registers with $O(n)$ overhead, so the tournament tree has $O(n^2 \log n)$ message complexity.

Change Slide. The randomized algorithm presented in this paper makes significant improvements to the expected step and message complexities. Each process is expected to take $O(\log^* n)$ step and expected to send $O(n^2)$ point-to-point messages. *Change Slide.* The authors also proved a lower bound of $\Omega(n^2)$ required messages for this problem, so the algorithm also has optimal message complexity.

3 Leader Election Algorithm

3.1 Intuition

Before we look at the code for the algorithm, lets try to get a idea of what it wants to do at a high-level. *Change Slide.* The algorithm works in phases. At each phase we must ensure that *there is at least one survivor*. During a phase, a process p_i flips a biased bit and decides whether to $survive$ (continue to the next phase) or

$lose$ (drop out of contention). If p_i gets 1, then it survives. If p_i gets 0 then it must make sure that some other process will survive before $losing$. Thus if p_i observes that some p_j flipped a 1 then it is safe in the knowledge that someone will $survive$ to the next phase and can $lose$. Otherwise, if p_i only observes processes which flipped 0, it is not sure if they will choose to $lose$ or $survive$. To ensure that some process survives to the next phase, p_i must survive even though it flipped 0.

Ideally we want many processes to drop out of contention at every phase, but a strong adversary can tamper with a naive implementation of the above idea. In particular, the adversary watch all the bit flips and schedule those processes which flipped 0 first. Then all the processes must survive (those which flipped 0 will only see other processes which flipped 0 and will $survive$ and those which flipped 1 will always survive). One of the main ideas in this algorithm, called the Poison Pill Technique, will ensure that this does not occur.

3.2 Local Variables and Communication Primitives

Change Slide. Move to board bottom left. A process p_i has two local variables: a vector S_i of length n which records the states of all observed processes and an $n \times n$ matrix V_i storing the view as seen by other processes. States can be any of w (waiting), c (committed), 0 (lose), and 1 (survive). All entries of S_i are initialized to w and all entries of V_i are empty. A process p_j is **active** from the perspective of p_i if column j in V_i contains at least one entry of status 1 or c and no entries of status 0.

Change Slide. The following are the communication primitives we will use in the algorithm. COMMUNICATE is a high-level protocol which takes a procedure p which broadcasts a message and waits for a response. COMMUNICATE$\langle p \rangle$ will execute p and wait for at least $\lceil n/2 \rceil + 1$ responses before proceeding. *This is reminiscent to what we did when we simulated single-reader single-writer registers in a message passing system*. Possible procedures p include: *read off* broadcast(v) *and* collect$()$ *from the slide. Change Slide.* Further we define the method random as *read off* random(r) *from the slide.*

We call a set of $\geq \lceil n/2 \rceil + 1$ processes a **quorum**. *Add the definition of quorum to the board.* They will arise frequently in our analysis.

3.3 Poison Pill (Simple)

Change Slide. Here is the algorithm for the poison pill algorithm (*see Algorithm 1*). This code is executed by each process at every phase to determine whether it should $survive$ or $lose$ the current phase. *Go through the code line by line.* To analyze the step complexity of the algorithm we need to prove two claims. *Change Slide.*

Algorithm 1 Poison Pill: code for process p_i during one phase.

1: Initialize $S_i[1...n] = [w, ..., w]$ and $V_i[1...n][1...n] = \emptyset$
2: $S_i[i] \leftarrow c$
3: COMMUNICATE$\langle$broadcast$(S_i[i])\rangle$
4: $S_i[i] \leftarrow$ random $\left(\frac{1}{\sqrt{n}}\right)$
5: COMMUNICATE$\langle$broadcast$(S_i[i])\rangle$
6: $V_i \leftarrow$ COMMUNICATE$\langle$collect$()\rangle$
7: **if** $S_i[i] = 0$ and $\exists k : p_k$ is active **then**
8: return 0
9: **end if**
10: return 1

Claim 1. *If all processes return then some process outputs* 1.

Proof. (c) Suppose for a contradiction that all processes output 0. If a process receives 1 from random $\left(\frac{1}{\sqrt{n}}\right)$, then it must output 1 so all processes received 0. Let process p_i be the last to complete the broadcast operation on Line 5. When p_i's broadcast operation finishes, the 0 of every process gets stored by a quorum. Consider V_i

at the completion of Line 6. Every column must have one 0-entry since the quorum that stored 0 must overlap with the quorum which successfully sent their status vector to p_i. Thus p_i must output 1. □

Lemma 2. *If a process received* 1 *from its execution of* random *at* t*. Then all process which received* 0 *from* random *at any time* $\geq t$ *must output* 0*.*

Proof. Suppose p_i received 1 from its execution of random at t and p_j received 0 at any time $\geq t$. Observe the COMMUNICATE of $S_i[i] = c$ on Line 3 has finished by t. Thus the quorum which received $S_i[i]$ and the quorum which sent their status to p_j upon its execution of collect() on Line 6 overlaps. p_j will see that p_i is active and will return 0. □

Claim 3. *The expected number of processes that return* 1 *is* $O(\sqrt{n})$*.*

Proof. The expected number of 1s is $\sqrt{n}$. All these processes will survive. The expected number of trails before the first 1 is flipped is also $\sqrt{n}$. By Lemma 2, at most $\sqrt{n}$ processes which flipped 0 will survive. □

By Claim 3, $O(\sqrt{n})$ processes are expected to survive each phase. Thus by a careful argument (which we can go through if there is time) there are $O(\log \log n)$ expected phases.

4 Improvements

Observe that this is not the $O(\log^* n)$ step complexity that we claimed for the algorithm at the beginning. The issue is with our choice of p (probability of flipping 1). In this simple version of the algorithm that we have presented, this value is fixed. It can be shown that if we must fix p then $O(\sqrt{n})$ is the best expected number of survivors we can achieve. Instead what we need is the probability to change dynamically throughout a phase. Intuitively, we want the p to be high at the beginning. When a process gets a 1 and broadcasts its value, all subsequent processes which flip a 0 can $lose$. So as more and more processes become active we want to decrease p so more processes receive 0.

By implementing this idea in the complete form of Poison Pill, the expected number of processes to survive at each phase drops to $O(\log n)$.

5 Conclusion and Recap

(c) To recap what we have discussed: *Talk about the main ideas of the Poison Pill Technique.* *(c)* Using these ideas, the authors developed a randomized algorithm for the Leader Election problem (or alternatively, $Test\&Set$ in a message passing model) with expected $O(\log^* n)$ step complexity and $O(n^2)$ message complexity. *(c)* We did not get an opportunity to talk about this today, but they also applied their algorithm to the renaming problem *read results obtained for the renaming problem.*

Reference

Alistarh, D., Gelashvili, R., and Vladu, A. (2015). How to elect a leader faster than a tournament. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, pages 365–374. ACM.