Homework of RDA

JIN Zhuoyuan, DIAO Daokui

October 22, 2025

Contents

0.1 Ex1 Proof of Termination and Correctness for the Basic LCR Algorithm

Consider the basic LCR solution to leader election seen in class (see Leader Election slides 5 - 7). Prove that this algorithm terminates and that it is also correct at termination (the definitions of these terms can be found in the Model slides of the course).

0.1.1 Proof of Correctness of the LCR Algorithm.

Let there be n processes arranged in a ring.

Denote:

$i_{\max}$ is the index of the process with maximum id.

$u_{\max}$ is this maximum id.

u_i is an id of process P_i.

1. The process with the maximum id will become leader.

According to the algorithm, for any process $Pi \neq i_{\max}$, $u_{\max}$ will be sent to the next process. And for all $0 \leq r \leq n-1$, after round r, $\text{msg}_{(i_{\max}+r) \bmod n} = u_{\max}$, thus, after n rounds, the process $Pi_{\max}$ will receive its own id $u_{\max}$, and its status will be set to leader.

2. No other process becomes the leader.

For any process Pi, if $i \neq i_{\max}$, since $u_i < u_{\max}$ when the msg arrive $Pi_{\max}$ will be discarded thus the $status_i$ always = unknown, so no other process becomes the leader.

0.1.2 Proof of Termination of the LCR Algorithm.

Since any $U_i \neq U_{\max}$ satisfies $U_i < U_{\max}$, each process $i \neq i_{\max}$ will forward $U_{\max}$ to the next process. Thus, the process $i_{\max}$ will eventually receive its own UID ($U_{\max}$) and become the leader. After that, no further messages are sent, and the algorithm terminates.

0.2 Ex2

Assume an arbitrary communication topology and synchronous model. Propose a solution to the leader election problem; satisfying also the three variants appeared in slide 3 (Leader Election slides). Give a short textual explanation of the algorithm and provide also the complete pseudo-code (states, starts, msgs(), trans()). Assume that an upper bound on the network diameter Diam is given to the processes (diameter is the longest among all the shortest paths in the given network). Explain why the latter assumption is needed.

Three variants:

At termination (or from some configuration), for any other process: 1. it learns that it will never become a leader (e.g., by status := non_leader). 2. it learns the id of the leader. 3. it determines when the leader is already elected.

0.2.1 FloodMax Leader Election Algorithm

Each process has a unique identifier (UID) and keeps track of the largest UID it has seen. The algorithm proceeds in synchronous rounds: in each round, every process sends its current largest UID to all neighbors and updates it to the maximum of the received values. After at most Diam rounds, the largest UID has reached every process, the process with the largest UID becomes the leader, and all other processes know they are not the leader and record the leader's UID. At this point, the algorithm terminates, and every process knows the election result.

0.2.2 Pseudo-code

```
States:
  status   {unknown, leader, non_leader}  // current status
  uid             // unique identifier of the process
  view            // the largest UID seen so far (initially uid)
  leader_id       // the final leader's UID (initially NULL)
  round   Z 0    // current round number (initially 0)
  Diam            // known upper bound on the network diameter


Starts:
  status := unknown
  view := uid
  leader_id := NULL
  round := 0


msgs():    // messages sent in each round
  // continue sending view if round < Diam; stop sending after round == Diam
  if round < Diam:
    send_to_all_neighbors(view)
  else:
    send_nothing()


trans(received_msgs):
  // 1. update view with the maximum value received
  for each v in received_msgs:
    if v > view: view := v

  // 2. increment the round counter
  round := round + 1

  // 3. after reaching Diam rounds, determine the leader and status, and stop
  if round == Diam:
    leader_id := view
```

```
if leader_id == uid:
  status := leader
else:
  status := non_leader
```

0.2.3 Why need Diameter

Processes need to know an upper bound on the network diameter, Diam, in order to ensure that:

- 1. The largest UID can propagate to the entire network, allowing a unique leader to be elected.
- 2. Each process can safely stop after a finite number of rounds, guaranteeing algorithm termination.
- 3. Each process can determine that the leader election has been completed, thereby satisfying the variant requirements.

0.3 Ex3

Consider the Time-Slice algorithm (see Leader Election slides 21 - 24). Propose a slight change to the algorithm to improve its bit complexity (in terms of O). Explain why the changed algorithm is correct and analyze the new bit complexity.

0.3.1 Improvement

Only transmit the candidate UID (i.e., the current largest UID in the ring)

- 1. In each round, only the current candidate UID is sent.
- 2. Processes with UID smaller than the candidate UID are eliminated and no longer send messages.
- 3. The process with the maximum UID continues sending until its UID completes a full round in the ring $\rightarrow$ it becomes the leader.

0.3.2 Correctness

Any process with UID $<$ current candidate UID is eliminated upon receiving a larger UID and stops sending. The process with the maximum UID is never eliminated; its UID can propagate completely around the ring and return to itself, and become the leader.

0.3.3 Complexity

Original Time-Slice Algorithm:

- Each process sends its own UID, which is forwarded around the ring n times $\Rightarrow$ the total number of messages in the worst case is $O(n^2)$.

- Total bit complexity = (number of messages) × (message length) = $O(n^2 \log U)$.

Optimized Time-Slice Algorithm:

- Eliminated processes stop sending; only the candidate UID is propagated.
- The process with the maximum UID sends at most n messages $\Rightarrow$ total number of messages $\leq n$. Each message length = $O(\log U)$.
- Total bit complexity = $O(n \log U)$.

After the optimization, the bit complexity decreases from $O(n^2 \log U)$ to $O(n \log U)$, significantly reducing the bit complexity.

0.4 Ex4

Consider the Flood-Set algorithm and assumptions in slide 3 of Consensus slides. Assume that n=4 and f=2. Present an execution scenario of the algorithm where every two (still) alive (not yet crashed) processes have different values in variable W at least in round 1 and 2. Whether these values become equal in round 3? Explain why.

0.4.1 Execution Scenario

Let the four processes have distinct initial values:

$$P_1 : a, \quad P_2 : b, \quad P_3 : c, \quad P_4 : d.$$

Round 1. All processes broadcast their values. Suppose that process P_4 crashes after sending only to P_2.

As a result, after round 1 we have:

$$\begin{aligned} W_1 &= \{a, b, c\}, \\ W_2 &= \{a, b, c, d\}, \\ W_3 &= \{a, b, c\}, \\ W_4 &= \{d\} \text{ (crashed)}. \end{aligned}$$

Thus, P_1 , P_2, P_3 hold different sets.

Round 2. Now only P_1, P_2,P_3 remain alive. They attempt to broadcast their sets: P_1 sends $\{a, b, c\}$,P_2 sends $\{a, b, c, d\}$ and P_3 sends $\{a, b, c\}$. Assume that P_2 crashes after sending only to P_1.

Then, after round 2:

$$\begin{aligned} W_1 &= \{a, b, c, d\}, \\ W_2 &= \{a, b, c, d\} \text{ (crashed)}. \\ W_3 &= \{a, b, c\}. \end{aligned}$$

Their sets remain different.

Round 3. Only P_1 and P_3 remain alive. They attempt to broadcast their sets: P_1 sends $\{a, b, c, d\}$ and P_3 sends $\{a, b, c\}$.

$$W_1 = \{a, b, c, d\} = W_1 = \{a, b, c, d\}.$$

0.4.2 After f+1 rounds

For the case $n = 4$ and $f = 2$, the Flood-Set algorithm guarantees that after round 3 ($f + 1$ rounds), the sets W_i of all surviving processes will be identical, because the initial value of each correct process can be propagated to all other correct processes within at most $f + 1$ rounds of broadcasting.

0.5 Ex5

1. Show that at the end of the first round, the majority values that could have been calculated/assigned (from the set of the received values) by correct processes may be different (so such a solution with the decision at the end of the 1st round, may not be correct).

Setup. Let the processes be $C = \{p_x, p_y, p_z\}$ (correct) and b (Byzantine). Initial values $v_{p_x} = x,\ v_{p_y} = y,\ v_{p_z} = z$ with x, y, z pairwise distinct.

Round 1 sends.

$$\forall c \in C:\ c \text{ broadcasts } (c, v_c), \qquad \forall c \in C:\ b \to c:\ (b, v_c).$$

(i.e., b equivocates and sends to each receiver c the pair carrying value v_c.)

Round 1 receipts (value multisets).

$$\begin{aligned} V_x &= \{x, x, y, z\}, \quad \text{maj}(V_x) = x, \\ V_y &= \{x, y, y, z\}, \quad \text{maj}(V_y) = y, \\ V_z &= \{x, y, z, z\}, \quad \text{maj}(V_z) = z. \end{aligned}$$

Conclusion. Since $\text{maj}(V_x) = x$, $\text{maj}(V_y) = y$, and $\text{maj}(V_z) = z$ (all different), there exists an execution where correct processes compute different majorities after Round 1. Therefore any protocol that decides at the end of Round 1 may be incorrect (Agreement can fail).

2. Show that at the end of the second round, a correct process cannot receive, for the same j, 2 pairs (j, v_j) and 2 pairs (j, v'_j) with $v_j \neq v'_j$.

Setup. There are three correct processes p_x, p_y, p_z and one Byzantine process b. Their initial values are x, y, z, respectively. Fix p_x as the observer. Messages have the form (j, value), where $j \in \{p_x, p_y, p_z, b\}$ denotes "about whom" the message is.

Round 1. p_x accepts only messages "about j" that are sent by p_j itself, and discards any "about j" message forged by b. If $j = p_x$, then p_x does not send to itself; hence in this round it can receive at most one such message.

Round 2. Each process forwards only those messages about j that it received in Round 1. Therefore the valid messages about j that p_x can accept now come only from other processes' forwards. If $j \neq p_x$, there are only two such processes, so in this round p_x can receive at most two. If $j = p_x$, the three other processes can each forward what they received in Round 1; thus by the end of this round p_x can receive at most three in total.

Conclusion. Four messages cannot occur. Hence, by the end of Round 2, for the same j, a correct process cannot simultaneously receive two copies of (j, v_j) and two copies of (j, v'_j) with $v_j \neq v'_j$.

3. Show that, at the end of the second round, given a process j, all correct processes assign to j the same value.

Proof . By Q2, each correct process holds *at most three* about-j messages.

(1) If p_j is correct: In Round 1 every correct process receives the same v_j from p_j. In Round 2 each correct process gets two forwards (from the two other processes), of which at least one is correct and thus forwards v_j. Hence every correct process has at least two copies of v_j among at most three messages, so v_j is a strict majority and is assigned by all.

(2) If p_j is Byzantine: Let the three correct processes receive (in Round 1) values w_x, w_y, w_z (possibly with repetitions). In Round 2 each correct process forwards exactly what it received in Round 1, so by the end of Round 2 every correct process holds the same multiset $\{w_x, w_y, w_z\}$. Applying the same assignment rule (strict majority if any; otherwise the smallest among the most frequent), they all choose the same value.

Thus, for any j, all correct processes assign the same value about j.

4. Deduce that the executions of this protocol verify the agreement condition. **Prove.**
For any two correct processes p_i and p_k. For each identifier $j \in \{p_x, p_y, p_z, b\}$, by Q3 we know that at the end of Round 2 all correct processes have the *same* value about j. Hence p_i and p_k hold the same four values (one about each $j \in \{p_x, p_y, p_z, b\}$).

The decision rule is: take the majority among these four values (with the tie broken by choosing the smallest value). This rule is deterministic and depends only on that four-value collection. Since p_i and p_k have identical inputs to the rule, they produce the same decided value. Therefore Agreement holds.

5. Do they verify the other conditions of the Byzantine consensus? **Prove.**

Agreement. Agreement holds by Q4; we now prove Termination and Validity.

Termination. The protocol has a fixed finite number of rounds (two). In a synchronous setting every correct process completes Round 1 and Round 2 and then immediately computes its assignments and decision; no additional waiting is required. Therefore every correct process decides by the end of Round 2.

Validity (all correct initial values equal). Assume the three correct processes have the same initial value v.

Correct identifiers $j \in \{p_x, p_y, p_z\}$. In Round 1 every correct process receives (j, v) directly from p_j. In Round 2, for any correct receiver, at least one of the two other senders is correct and forwards (j, v). Thus each correct process has at least two copies of v among at most three about-j messages, so the assigned value for j is v.

Byzantine identifier $j = b$. The assignment for b may be arbitrary, but the final decision is the majority among the four assigned values (for p_x, p_y, p_z, b). Since the three assignments for p_x, p_y, p_z are all v, v wins 3-to-1, so every correct process decides v.

Therefore, if all correct initial values are v, all correct processes decide v. Validity holds.