RDA

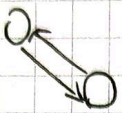
第一章：

Communication links / channels.

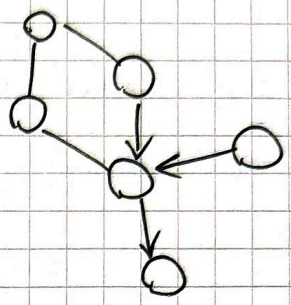
Δ. Communication network graph $|V| = n$ → $E = \{\langle i,j\rangle \mid i,j \in V\}$.

$G(V,E)$ directed graph 有向图. (V nodes, E directed edges).

~~bidirected~~ bidirectional link 双向链接



unidirectional link 单向链接



n = 6

○ Each directed edge in E:

at a time:

- a single message of M.
- an empty message named NULL

○ for each Process i, we denote by:

- out_nbrs_i the set of successor nodes of p_i 出邻节点 (后继节点).
- in_nbrs_i the set of predecessor nodes of p_i. 入邻接节点 (前驱节点).

In a non-directed graph G. 无向图中.

↳ $nbrs_i = out\_nbrs_i = in\_nbrs_i$.

(邻接节点集) = 出 = 入.

Δ. A process p_i consists of:

$states_i$: 状态集. set of all possible states of p_i.

(can be infinite) 不一定为有限集.

$start_i \subseteq states_i$: 初始状态. set of initial states.

$msgs_i$: 发消息. message generation function 消息生成函数.

$states_i \times out\_nbrs_i \mapsto M \cup \{NULL\}$.

$trans_i$: 变状态. transition function 转换函数

状态×接收到的消息↦新状态 ① $states_i \times [M \cup \{NULL\}]_{in\_nbrs_i} \mapsto states_i$

△. Synchronous execution. 同步执行规则.

start with:

1. all processes in the initial state. 所有进程处于初始状态.
2. all processes wake up at the same time. 同时启动.
3. the communication links are empty 通信链路为空(无消息)

In each round, each process p_i: (a round is assumed to execute without interruption).

1. Apply msg_i.
2. Apply $trans_i$. ~~瞬时~~ 瞬时完成

△. Why a perfectly synchronous model?

1. Easy to design and analysis
2. there are synchronizer. 同步器.
3. useful for impossibility proofs. 同步网络中无法解决.
   异步网络更无法解决.

△ Algorithm correctness (formal definitions)

① configuration (系统状态): vector of states of all processes.
(系统在某一时刻全局状态, 快照) $\{s_1, s_2, s_3 \cdots s_n\}$.

② Exaution (执行序列): a infinite sequence $C_0, M_1, C_1, M_2, \cdots$
(系统从初始状态一直运行下去的完整历史.)
↓状态 ↓消息传递.
刻画算法运行的所有可能

③ Terminal configuration. (终止配置)
状态不再变化.

1. All processes does not change state (apply $trans_i$)
2. msg_i generates only "null" message. 不再发送任何消息.

④ Problem P. 谓词 (predicate)

Δ. Complexity measures. O(n)

- Time complexity: the maximum number of rounds to termination 终止所需最大轮数.
- Message complexity: the max number of non-null messages sent (generated)
- Messages' bit complexity: int number of generated bits.
  ~~bits for an~~ bits/message × number of msg.

第二章: Leader Election

Δ. LE problem specification.

① At termination, have a P_i has state i = leader
② During any execution, at most one process has ~~state~~ status = leader

Variants: 变体

At termination, for any other process:

① it learns that it will never become a leader (~~state~~ status = non-leader)
② it learns id of leader (know who is leader)
③ it determines when the leader is already elected.

Δ why? ① Simplifies coordination 简化协调工作
② symmetry breaking problem 对称性破缺问题 (做决策)
③ Helps to tolerate failures 容错 (eg. 备份)
④ saving resources 节省资源.

△ LCR.

每个进程都线环发送自己的UID。当一个进程收到一个UID时，和自己的UID比较，如果比自己大，则继续传递；如果比自己小，则丢弃；如果和自己一样大，则宣布自己是leader.

M = UID (Set of possible message).

statesi + startsi (possible states + starts).

my_id : UID ; my_id := id of i ;

msg : M ∪ {NULL}. ; msg := my_id ;

status : {leader, unknow}. ; status := unknow ;

msgsi: (message generation function)

msgsi (<state of pi>, pi+1) := msg. 生成一条msg，发送给 pi+1

transi (transition function):

msg := null ~~和msg相同~~.

if (m msg in UID) then

case ~~msg~~

1. m msg > my_id: msg := m msg

2. m msg = my_id: status := leader

End case

End if

⊙proof

complexity (LRC)

① in rounds : n

② in messages: $O(n^2)$

- order by from n to 1. → (n发送n次, n-1发送n-1次 ----)

$n+(n-1)+\cdots+1 = \frac{n(n+1)}{2}$

- order by $1 \to n$ (除了n发送n次, 其它发送1次) → n-1个节点.

$n+(n-1) = 2n-1$

③ in bits. suppose size of id is $O(\log n)$ bits.

$O(n^2 \log n)$.

LCR variants :

① $M = UID \cup \{terminated\}$.

② $states_i$ + $starts_i$:

my-id : UID my-id := id of i

msg : $M \cup \{null\}$ msg := my-id

status : {leader, unknown, non-leader} status := unknown

leader-id : UID leader-id = my-id.

terminated : boolean terminated := false.

③ $msgs_i$:

if (status = unknown) then

  $msgs_i(\langle state\ of\ p_i \rangle, p_{i+1}) := msg$

else if (terminated = false) then → 再all others set terminated.

  $msgs_i(\langle state\ of\ p_i \rangle, p_{i+1}) :=$ "terminated"

  terminated = true

else

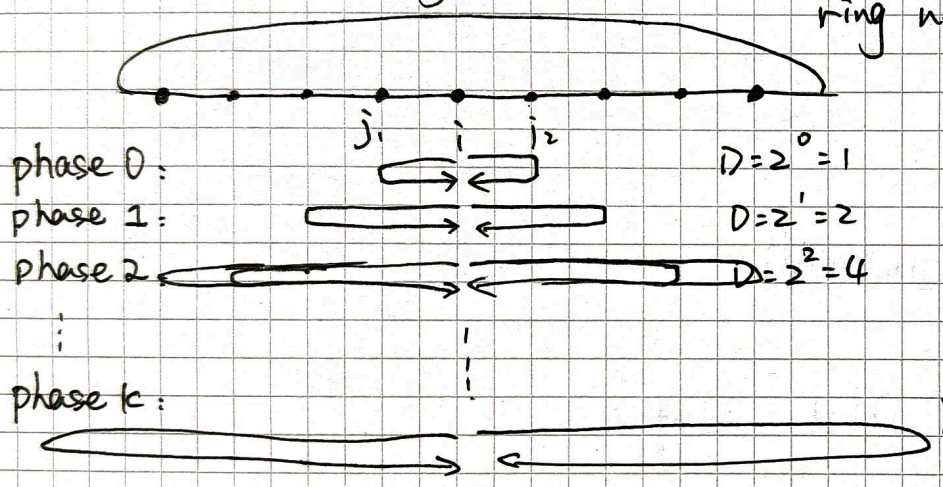
  $msgs_i(\langle state\ of\ p_i \rangle, p_{i+1}) :=$ null. → u < uid 不转发

End if

④ transi :

```
msg := null ;
if (m in UID) then
    case.  1. m > my-id :  msg := m
                           leader-id = max (leader-id , m)
           2. m = my-id :  status = leader
    Endcase
elseif ( m = "terminated" and status ≠ leader) then
    status = non-leader
Endif
```

A HS. (Hirschberg and Sinclair)



Each Process P_i in phase 0, 1, 2 ... k.

At each phase, send two tokens with id.

$$\begin{cases} u_i < u_j & \text{discard.} \\ u_i > u_j & \text{continue.} \\ u_i = u_j & \text{declare leader} \end{cases}$$

distance.

① $M = UID \cup \{in, out\} \cup N^+$

② states$_i$ + starts$_i$:

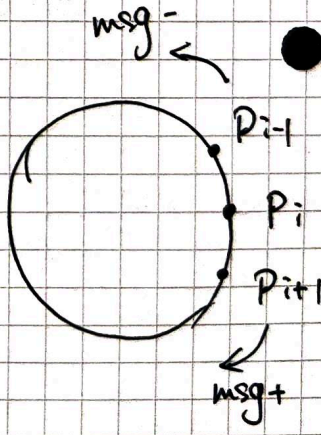
my_id : UID , myid = id of i

msg+ : $M \cup \{null\}$ msg+ = (my_id, out, 1)

msg- : $M \cup \{null\}$ msg- = (my_id, out, 1)

status : {leader, unknow}, status = unknown

phase : N° phase = 0



③ msgs$_i$:

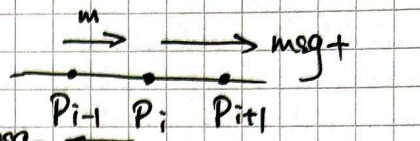
~~for msg+ :~~ ~~msgs~~ msgs$_i$(<state of P_i>, P_{i+1}) = msg+

msgs$_i$(<state of P_i>, P_{i-1}) = msg-

④ transi:

msg+ = msg- = null;

if (m from P_{i-1} = (id, out, h)) then ← hop count，初始为 2^k，当 h=1 时说明要返回。

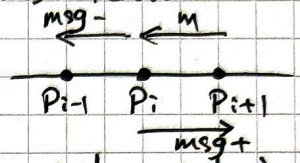


case:

1. id > my_id and h>1 : msg+ = (id, out, h-1)
2. id > my_id and h=1 : msg- = (id, in, 1)
3. id = my_id : status = leader

Endcase

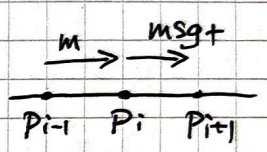
if (m from P_{i+1} = (id, out, h)) then



case:

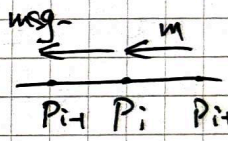
1. id > my_id and h>1 : msg- = (id, out, h-1)
2. id > my_id and h=1 : msg+ = (id, in, 1)
3. id = my_id : status = leader

Endcase



if (m from P_{i-1} = (id, in, 1) and id > my_id) then 收到比自己大的 in 消息，继续转发

msg+ = (id, in, 1)



if (m from P_{i+1} = (id, in, 1) and id > my_id) then

msg- = (id, in, 1)

if (two message receive from P_{i-1} and P_{i+1}, each is (my_id, in, 1)) then

phase += 1

msg- = (my_id, out, 2^{phase})

msg+ = (my_id, out, 2^{phase})

Δ. complexity:

① Message: for each active P_i, there are at most $4 \cdot 2^k$ tokens.

- k=0, there are n active ~~token~~ processes.
- $k \geq 1$, there are at most $\frac{n}{2^{k-1}+1}$ active processes.

so at most $4 \cdot 2^k \cdot \frac{n}{2^{k-1}+1} \leq 8n$ messages sent in phase $k \geq 1$.

and 4n for k=0.

~~O(nlogn)~~ the max phase of n processes is logn+1 (起始0). 最多 logn+1 轮.

thus. At most $8n \cdot (\lceil \log n \rceil + 1) + 4n$ messages. O(nlogn).

number of

② Time complexity.

最后一阶段 the last phase = $\lceil \log n \rceil$，需要 n rounds.

other phase = k, at most $2 \cdot 2^k$ rounds.

$2 \cdot (2^0 + 2^1 + 2^2 + \cdots + 2^{\lceil \log n \rceil - 1}) + n = 2(2^{\lceil \log n \rceil} - 1) + n$

if n is power of 2. at most $3n-2$ rounds.

if not, at most $5n-2$ rounds.

so $O(n)$

→ 如果 $n = 2^m \rightarrow \log n = m$

$2(2^m - 1) + 2^m = 3 \cdot 2^m - 2$

$= 3n - 2$

如果 $n \neq 2^m \rightarrow \lceil \log n \rceil = m+1$

$2 \cdot (2^{m+1} - 1) + 2^{m+1} = 5 \cdot 2^m - 2$

$= 5n - 2$

Δ TS (Time Slice) 非基于比较.

① n个进程. 每个进程知道 n，阶段 1：1～n 轮

② 划分成 n 个阶段，每个阶段长度 = n 轮. 阶段 v. $(v-1)n+1$ ～ vn 轮.

③ 第 v 阶段只允许 ID = v 进程发送 token，token 转一圈，如果没有比自己更小的 id，则 declare leader.

④ 判断是否有机会成为 leader，若阶段 v 开始前，v 未收到非空消息 → 之前没有 id 小于它，则有希望.

⑤ 若阶段 v 没有进程成为 leader，则 $v = v+1$

① M = UID

② states$_i$ + starts$_i$:

my_id : UID	my_id = id of i
round : N^+	round = 1
leader_id : UID	leader_id = my_id
msg : $M \cup \{null\}$	if (my_id = 1) then msg = my_id. (在阶段 v，只有 v 可以发 msg.)
	else msg = null.
status : {leader, non-leader, unknown}	if (my_id = 1) then status = leader.
	else status = unknown.

③ msgsi:

msgsi(<State of Pi>, Pi+1) = msg.

④ transi:

```
msg = null.
if (status = unknown) then.        // 处理未确定身份的
    if (m ≠ null, in UID) then.    // 如果m是token, 则说明收到非应消息. 自己不能成为leader.
        status = non_leader
        leader_id = msg = m
    Elseif (round = (my_id - 1)·n) then
        status = leader
        leader_id = msg = my_id.
    Endif
Endif
round = round + 1
```

A. 第三章：一致性问题 Fundamental problem.

(The processes must agree on a common data value).

⊙. Assumptions:

① complete undirected communication graph. 完全无向图.

② n processes (1, 2, ⋯ n) completely know the network.

③. each processes has an initial value ∈ U.

④. synchronous communication model. 同步.

⑤ msg be ~~reliest~~ reliable and is received.

⑥ any process can crash. 故障.

⑦ $f<n$ maximum number of the crashed / faulty processes.

complexity: ① $T = f+1$ (轮).

② $M = (f+1)[n(n-1)] = O(f \cdot n^2)$

每一轮，n个节点要向n-1个节点发消息 $n\times(n-1)$

A. Flood-Set algo.

① 每个进程维护一个集合 W，$W \subseteq UID$. 初始时，进程 P_i 的 W 仅包含 p_i.

② 在 f+1 个回合中，每个回合，每个进程广播自己的 W，并将接收到的加入 W.

③ 在 f+1 回合结束时，每个进程有相同的 W，并做出统一决策.

|W|=1 时，则为 v (W={v})，或一个预先定义的 v。

① $M = P(U) = 2^U$ → power set of U，U的幂集(所有U的子集组成的集合)

eg. $U = \{a, b\}$

$2^U = P(U) = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$.

② $state_i$ + $start_i$:

rounds : $\mathbb{N}^+$ rounds = 1

decision: $U \cup \{unknown\}$ decision = unknown (written only once)

W: P(U) ~~[illegible]~~ $W = \{v_i\}$ (v_i is the initial value of P_i)

③ $msgs_i$:

if (rounds ≤ f+1) then.

send W to every neighbouring process

④ transi:

Let X_j ($\in P(u)$) be the message recieved from neighbour p_j.

~~$W = W \cup X_j$ (所有 X_j)~~ $W = W \cup \bigcup_{\text{over all } p_j} X_j$

if (rounds = f+1) then

  if (|W| = 1) then → W长度为1，只有一个元素。

    decision = v such that W = {v}.

  Else

    decision = v_0 (pre-difined value)

  Endif

~~or desired~~

Endif

可替换 e.g. decision = min(W).

if(rounds ≤ f+1) then

  rounds = rounds + 1

A. proof scheme 证明框架. 证明 Flood-set 正确性. (2点)

① At the end of a round with no faults, each non-faulty (correct) process up to this round obtains the same set of value W.

And after this round, W of any correct process does not change anymore.

在一个无故障回合结束后，所有进程有相同 W，且之后任何正确进程的 W 都不会变了。

如果某一轮，$W_i = W_j$.

② Among the f+1 rounds, at least one is without faults.

在 f+1 回合中，至少有一回合无故障。

证①: Lemma 1. if no process crashes during a round r ($1 \le r \le f+1$), then $\forall i,j \in A(r)$, $W_i(r) = W_j(r)$.

正确性 correctness

$W_i(r)$: W of process P_i at the end of round r

$A(r)$: set of active process at the end of round r

proof:

Because no process crash during round r, thus.

$\forall P_i \in A(r)$, $W_i(r) = \bigcup_{x \in A(r-1)} W_x(r-1)$

thus all active processes have same $W_i(r)$ at the end of round r.

Lemma 2. if $W_i(r) = W_j(r) = W$ after r rounds, then for any round r' ($0 \le r \le r' \le f+1$), $W_i(r') = W_j(r) = W$. $W_i(r)=W \rightarrow W_i(r')=W$, $r' \ge r$

proof: we prove by induction on r'. 一致性.

① when $r' = r$, is trivially true.

② Assume r' is true, that is $W_i(r') = W$

③ on $r'+1$, each active process recieves only empty messages (crash) or the same set W_i, thus, $W_i(r'+1) = W \cup W = W$

Therefore, the statement is correct for any $r' \le f+1$. 最终一致性

Lemma 3. if processes i, j are active after $f+1$ rounds, then $W_i(f+1) = W_j(f+1)$.

proof: ① if there is a crash in every one of the $f+1$ rounds, then at least $f+1$ processes crashed, which is a contradiction. (with at most f processes may crash). 矛盾.

② ~~by lemma, all active processes~~

therefore at least one round without any crash.

by Lemma 1, all active processes will have same W after that round.

And by Lemma 2, W will not change until the end.

正: ② Lemma 4. if all processes have the same initial value v, 有效性 validity v is the only possible decision value. 所有进程都有初始值 v, 则最终决定值也只能是 v. (不会凭空出现别的值).

proof: if all processes have the same initial value v. then

$\forall p_i \in A(0): W_i(0) = \{v\} \xrightarrow{\text{by Lemma 2}} W_i(f+1) = \{v\}$. for all $p_i \in A(f+1)$

$\longrightarrow |W_i(f+1)| = 1$. so decision is v.

Brief: Theorem: The Flood-set algo solves consensus with crashes (with parameter f). Flood-set 解决存在 crash 的一致性问题.

- termination: every active P decides in round $f+1$.
- Validity: Lemma 4
- Agreement (一致性): By Lemma 3, $\forall i, j \in A(f+1)$, $W_i(f+1) = W_j(f+1)$ thus they decide the same value.

Δ. 一致性问题不可解性 impossibility.

在 2 进程系统中，如果消息可以无限丢失，则不存在算法同时满足 termination, validity and agreement.

只要消息可能丢失，则两进程问题无解；

Δ. Byzantine faults.

○ Assumptions:

1. complete undirected communication graph.
2. processes know the network.
3. initial value $\in U$.
4. perfectly synchronous communication model.
5. communication links reliable.
6. a process can be faulty (Byzantine).
7. $f < n$ (maximum number of faulty processes).
8. $n > 3f$ $(n \geq 3f+1)$.

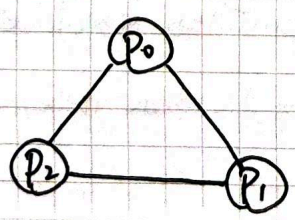
○. The difinition

1. Agreement. ~~two process~~ two non-faulty processes never decide on different value.
2. Validity. all non-faulty processes initial value v $(\in U)$. then. v is the only possible decision.
3. Termination. all non-faulty processes will decide on a value.

~~decid~~ decision is made only once. and irreversible.

Δ. $n=3$, $f=1$.

proof: by contradiction.



We have 3 processes, and their initial value are $P_0=0, P_1=0, P_2=1$ and P_2 is byzantine, P_2 send 0 to P_0, send 1 to P_1, then. $P_0: \{0,0\}$ $P_1: \{0,1\}$. may will decide different value.

A. Reliable Broadcast 可靠广播.

一个 correct 进程发送消息，保证即使存在 Byzantine，所有 non-faulty processes 最终都会一致地接收到这个消息。

- formal definition in 3 conditions: (m, i, r) — i: 发送进程编号; r: 轮次.

① 两步传播机制：(init + ~~echo~~ echo)

① 发送者在 r 轮广播消息. ("init", m, i, r)

② 所有正确进程收到后，在 r+1 轮再广播 ("echo", m, i, r).

③ 当一个进程从足够多的进程（至少 n-f 个）收到 "echo" 时，就接收 (accept) 该消息. (加到 W).

Cond. I. if a correct process disseminates (broadcasts) a message (m, i, r) in round r, the message is accepted by every correct process until round r+1. 如果一个P在r轮发送了消息，则所有正确进程都会在第 r+1 轮前接受该消息.

正确发送者的消息一定被所有正确进程接收.

proof: P_i sends ("init", m, i, r) to every process in round r

→ every correct process among at least n-f correct processes send ("echo", m, i, r) to every process in round r+1.

then, at the end of round r+1, every correct process recieves ("echo", m, i, r) from at least n-f processes and thus accepts the message (m, i, r).

condition II. if a P_i does not disseminate (broadcast) a msg (m, i, r) in round r, (m, i, r) is never accepted by a correct process. 如果 P_i 没发消息，则不会有任何正确进程接受该消息.

未发送的消息不会被接受.

proof: if p_i does not disseminate (m, i, r), p_i does not send a message ("init", m, i, r), thus, no correct process send ("echo", m, i, r). At most have f Byzantines send "echo". However $f < f+1 < n-f$ (Because $n > 3f$), thus never accepts (m, i, r).

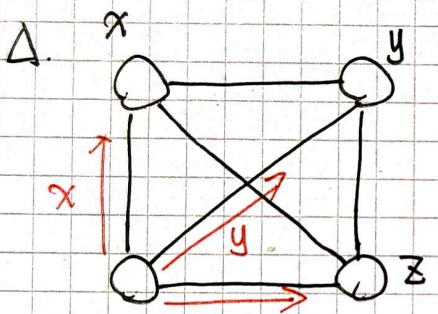
condition III. if a message (m, i, r) is accepted by a correct process P_j in round r', it is accepted by every other correct process (at least in round $r'+1$ or earlier)

一旦一个正确进程接受了消息，所有人都会接受

proof: if (m, i, r) is accepted by P_j in r', P_j has recieved atleast $n-f$ "echo" until round r'. Among these $n-f$ processes, there are at least $n-2f$ ($n-f-f$) correct processes. ~~send "echo"~~ They already sent "echo" before round r'. thus other correct processes will recieve at least $n-f$ "echo" until $r'+1$.

therefore, ~~every~~ any correct process ~~will recieves (echo m, i, r)~~ accepts (m, i, r) until $r'+1$.

Δ.



Byzantine

round 1.

$V_x = \{x, x, y, z\}$ $maj(V_x) = x$

$V_y = \{x, y, y, z\}$ $maj(V_y) = y$ (all different)

$V_z = \{x, y, z, z\}$. $maj(V_z) = z$