

第二部分 分布式算法

第五次课

中国科学技术大学计算机系

国家高性能计算中心（合肥）

第三章 环上选举算法

本章提纲

- Leader选举问题
- 匿名环
- 异步环
- 同步环

■ 问题

在一组处理器中选出一个特殊结点作为 leader

■ 用途

① 简化处理器之间的协作；

有助于达到容错和节省资源。

例如，有了一个 leader，就易于实现广播算法

② 代表了一类破对称问题。

例如，当死锁是由于处理器相互环形等待形成时，可使用选举算法，找到一个 leader 并使之从环上删去，即可打破死锁。

§ 3.1 leader选举问题

Leader选举问题：

问题从具有同一状态的进程配置开始，最终达到一种配置状态。每个处理器最终确定自己是否是一个leader，但只有一个处理器确定自己是leader，而其他处理器确定自己是non-leader。

算法的作用：

如果要执行一个分布式算法，且没有一个优先的优选人做为算法的初始进程，就要进行进程选举。(例如指定根的DFS树的生成问题)

§ 3.1 leader选举问题

选举算法的定义：

- (1) 每个处理器具有相同的局部算法；
- (2) 算法是分布式的，处理器的任意非空子集都能开始一次计算；
- (3) 每次计算中，算法达到终止配置。在每一可达的终止配置中，只有一个处理器处于领导人状态，其余均处于失败状态。(此项有时可以弱化)

一个算法解决了leader选举问题需满足(根据形式化模型)：

- ① 终止状态被划分为两类：选中和未选中状态。一旦一个处理器进入选中(或未选中)状态，则该处理器上的转换函数将只会将其变为相同的状态；
- ② 在每个容许执行里，只有一个处理器进入选中状态，其余处理器进入非选中(non-selected)状态。

本章只讨论系统的拓扑结构是环的情况。

§ 3.1 leader选举问题

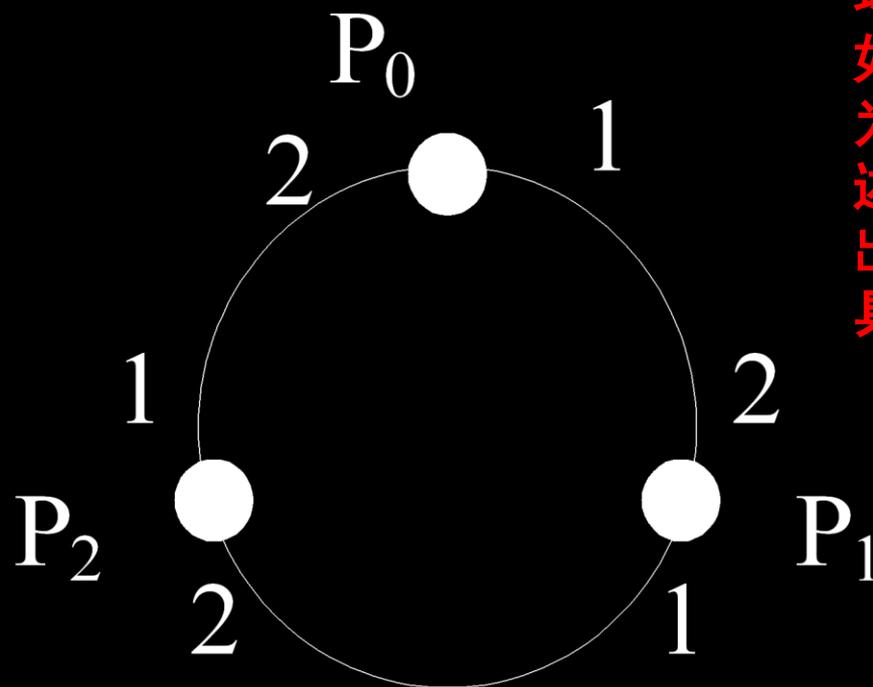
■ 环的形式化模型

对每个 i , $0 \leq i \leq n-1$,

P_i 到 P_{i+1} 的边标号为1, 称为左(顺时针)

P_i 到 P_{i-1} 的边标号为2, 称为右(逆时针)

这里的标号加减均是 $\text{mod } n$ 的



环网络之所以吸引了如此多的研究, 是因为它们的行为易于描述; 且从环网络推导出的下界, 可应用于具有任意拓扑结构的网络算法设计

§ 3.2 匿名环 (anonymous)

- **匿名算法**：若环中处理器没有唯一的标识符，则环选举算法是匿名的。**更形式化的描述**：每个处理器在系统中具有相同的状态机，在这种算法里，msg接收者只能根据信道标号来区别。
- **(一致性的) uniform算法**：若算法不知道处理器数目，则算法称之为uniform，因为该算法对每个n值看上去是相同的。
- **non-uniform算法**：算法已知处理器数目n
- **形式化描述**：在一个匿名、一致性的算法中，所有处理器只有一个状态机；在一个匿名、非一致性的算法中，对每个n值（处理器数目）都有单个状态机，但对不同规模有不同状态机，也就是说n可以在代码中显式表达。

§ 3.2 匿名环 (anonymous)

- 对于环系统，不存在匿名的选举算法。

为简单起见，我们只证明

非均匀（非一致性）算法：非均匀算法（ n 已知）的不可能性 \Rightarrow 均匀（ n 未知）算法的不可能性。

Ex3.1 证明同步环系统中不存在匿名的、一致性的领导者选举算法。

同步算法：同步算法的不可能性 \Rightarrow 异步算法的不可能性。（同步是异步的一种特例）

Ex3.2 证明异步环系统中不存在匿名的领导者选举算法。

§ 3.2 匿名环

■ 同步算法的不可能性

在同步系统中，一个算法以轮的形式进行。每轮里所有待发送msg被传递，随后每个处理器进行一步计算。

一个处理器的初始状态包括在outbuf里的任何msg。这些消息在第一轮里被传递到某处理器的左和右邻居。

不可能性：

① 在一个匿名环中，处理器间始终保持对称，若无某种初始的非对称(如，标识符唯一)，则不可能打破对称。在匿名环算法里，所有处理器开始于相同状态。

② 因为他们执行同样的程序(即他们的状态机相同)，在每轮里各处理器均发送同样的msg，所以在每一轮里各处理器均接收同样的msg，改变状态亦相同。

因此，若选中一个处理器，则其他所有处理器亦被选中。因此，不可能有一个算法在环中选中单个处理器为leader。

§ 3.2 匿名环

假设 R 是大小为 $n > 1$ 的环（非均匀）， A 是其上的一个匿名算法，它选中某处理器为leader。因为环是同步的且只有一种初始配置，故在 R 上 A 只有唯一的合法执行。

- **Lemma 3.1** 在环 R 上算法 A 的容许执行里，对于每一轮 k ，所有处理器的状态在第 k 轮结束时是相同的。

Pf. 对 k 用归纳法

$K=0$ (第一轮之前)，因为处理器在开始时都处在相同的初始状态，故结论是显然的。

设引理对 $k-1$ 轮成立。因为在该轮里各处理器处在相同状态，他们都发送相同的消息 m_r 到右边，同样的消息 m_l 到左边，所以在第 k 轮里，每处理器均接收右边的 m_l ，左边的 m_r 。因此，所有处理器在第 k 轮里接收同样的消息，又因为它们均执行同样的程序，故第 k 轮它们均处于同样的状态。

§ 3.2 匿名环

上述引理蕴含着：若在某轮结束时，一个处理器宣布自己是leader(进入选中状态)，则其它处理器亦同样如此，这和A是一个leader选举算法的假定矛盾！因此证明：

- **Th3.2** 对于同步环上的leader选举，不存在非均匀的匿名算法。

+

+

同步环 → 异步环

非一致性 → 一致性算法

↓↓

对于环系统，不存在匿名的选举算法

§ 3.3 异步环

本节将讨论异步环上leader选举问题的msg复杂性上下界。

由Th3.2知，对环而言没有匿名的leader选举算法存在。因此以下均假定处理器均有唯一标识符。

当一个状态机(局部程序)和处理器 P_i 联系在一起时，其状态成分变量 id_i 被初始化为 P_i 的标识符的值，故各处理器的状态是有区别的。

- **环系统：**通过指派一个处理器列表按顺时针(从最小标识符起)指定环。注意是通过 id 排列，不是通过 P_i 的下标 i 来排列($0 \leq i \leq n-1$)，假定 id_i 是 P_i 的标识符。(因为下标 i 通常是不可获得的)

§ 3.3 异步环

- 在非匿名算法中，均匀（一致性）和非均匀（非一致性）的概念稍有不同

- ① **均匀算法**：每个标识符id，均有一个唯一的状态机，但与环大小n无关。而在匿名算法中，均匀则指所有处理器只有同一个状态。（不管环的规模如何，只要处理器分配了对应其标识符的唯一状态机，算法就是正确的。）
- ② **非均匀算法**：每个n和每个id均对应一个状态机，而在匿名非均匀算法中，每个n值对应一个状态机。（对每一个n和给定规模n的任意一个环，当算法中每个处理器具有对应其标识符的环规模的状态机时，算法是正确的。）

下面将讨论msg复杂性： $O(n^2) \rightarrow O(n \log n) \rightarrow \Omega(n \log n)$

§ 3.3.1 一个 $O(n^2)$ 算法

→ 下界

Le Lann、Chang和Roberts给出，LCR算法

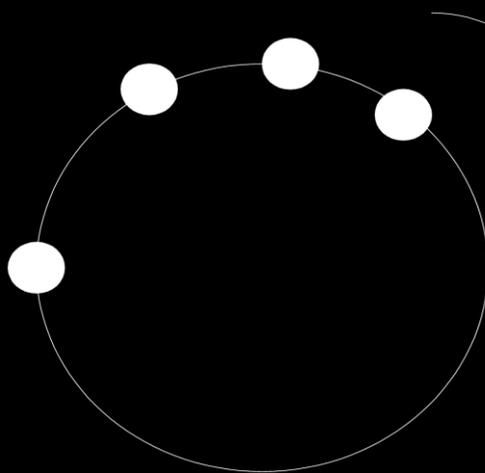
- **基本思想**

- ① 每个处理器 P_i 发送一个msg(自己的标识符)到左邻居，然后等其右邻居的msg
- ② 当它接收一个msg时，检验收到的 id_j ，若 $id_j > id_i$ ，则 P_i 转发 id_j 给左邻，否则没收 id_j (不转发)。

§ 3.3.1 一个 $O(n^2)$ 算法

- ③ 若某处理器收到一个含有自己标识符的msg，则它宣布自己是leader，并发送一个终止msg给左邻，然后终止。
- ④ 当一处理器收到一个终止msg时，向左邻转发此消息，然后作为non-leader终止。

因为算法不依赖于 n ，故它是均匀的。



i —表示id 单向

§ 3.3.1 一个 $O(n^2)$ 算法

Code for P_i

init var: asleep \leftarrow true, id \leftarrow I

Begin

While (receiving no message) do

(1) if asleep do

(1.1) asleep \leftarrow false

(1.2) send <id> to left-neighbor

end if

End while

While (receiving <i> from right-neighbor) do

(1) if id<<i> then send <i> to left-neighbor

end if

(2) if id=<i> then

(2.1) send <Leader,i> to left-neighbor

(2.2) terminates as Leader

end if

End while

While (receiving <Leader,j> from right-neighbor) do

(1) send <Leader,j> to left-neighbor

(2) terminates as non-Leader

End while

end

§ 3.3.1 一个 $O(n^2)$ 算法

■ 分析

① 正确性

在任何容许执行里，只有最大标识符 id_{\max} 不被没收，故只有具有 id_{\max} 的处理器接受自己的标识符并宣布是leader，其他处理器不会被选中，故算法正确。

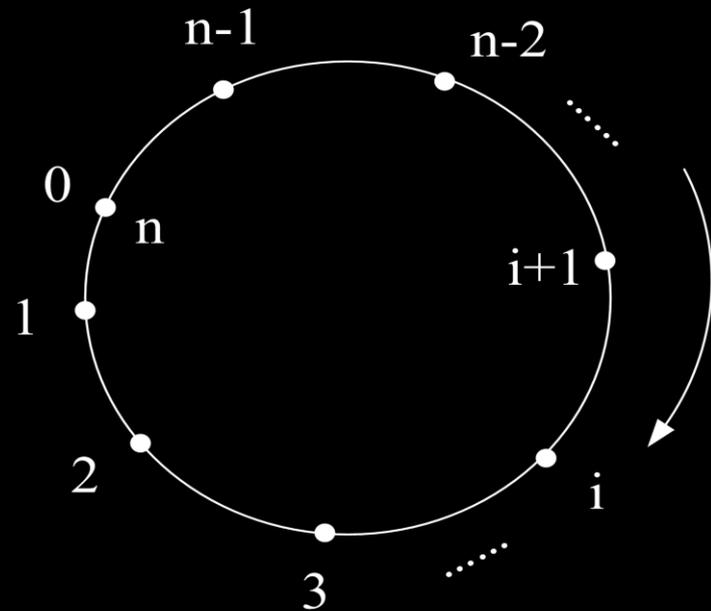
② msg复杂性

在任何容许执行里，算法绝不会发送多于 $O(n^2)$ 个msgs，更进一步，该算法有一个容许执行发送 $O(n^2)$ 个msgs：

§ 3.3.1 一个 $O(n^2)$ 算法

考虑处理器标识符为 0, 1, ..., $n-1$ 构成的环, 其次序如右图:

在这种配置里, $id=i$ 的处理器 msg 恰好被发送 $i+1$ 次, 即发送到 $i-1, i-2, \dots, 1, 0$, 直到 $n-1$ 时没收。因此, msg 总数为:



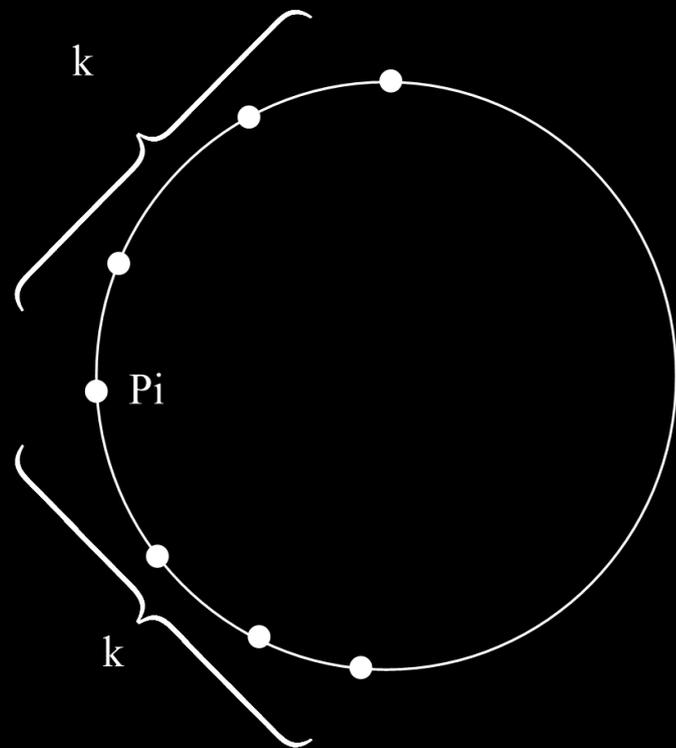
$$n + \sum_{i=0}^{n-1} (i+1) = \theta(n^2) \quad // \text{前一项为终止msg}$$

§ 3.3.2 一个 $O(n \lg n)$ 算法

仍然是绕环发送id，
但使用更聪明的方法。
保证最大id在环上周游
且返回。

■ k 邻居

一个处理器 P_i 的 k 邻居
是一个处理器集合：该
集合中的任一处理器与
 P_i 在环上的距离至多是 k ，
一个处理器的 k -邻居集
合中恰好有 $2k+1$ 个处理
器。



$k=3$ ，共有7个结点

§ 3.3.2 一个 $O(n \lg n)$ 算法

■ 基本思想

算法按阶段执行，在第 l 阶段一个处理器试图成为其 2^l -邻居的临时leader。只有那些在 l -th阶段成为临时领袖的处理器才能继续进行到 $(l+1)$ th阶段。因此， l 越大，剩下的处理器越少。直至最后一个阶段，整个环上只有一个处理器被选为leader。

■ 具体实现

- ① **phase0**: 每个结点发送1个probe消息(其中包括自己的id)给两个1-邻居，若接收此msg的邻居的id大于消息中的id，则没收此msg；否则接收者发回一个reply msg。若一个结点从它的两个邻居收到回答msg reply，则该结点成为phase0里它的1-邻居的临时leader，此结点可继续进行phase1。

§ 3.3.2 一个 $O(n \lg n)$ 算法

② **phase l** : 在 $l-1$ 阶段中成为临时leader的处理器 P_i 发送带有自己id的probe消息至它的 2^l 邻居。若此msg中的id小于左右两个方向上的 $2 * 2^l$ 个处理器中任一处理器的id, 则此msg被没收。若probe消息到达最后一个邻居而未被没收, 则最后一个处理器发送reply消息给 P_i , 若 P_i 从两个方向均接收到reply消息, 则它称为该阶段中 2^l 邻居的临时leader, 继续进入下一阶段。

③ **终止**: 接收到自己的probe消息的结点终止算法而成为leader, 并发送一个终止msg到环上。



§ 3.3.2 一个 $O(n \lg n)$ 算法

④ 控制probe msg的转发和应答

probe消息中有三个域： $\langle \text{prob}, \text{id}, l, \text{hop} \rangle$

id-标识符

l -阶段数

hop-跳步计数器：初值为0，结点转发probe消息时加1.

若一结点收到的probe消息时，hop值为 2^l ，则它是 2^l 邻居中最后一个处理器。若此时msg未被没收也不能向前转发，而应该是向后发回reply消息。

§ 3.3.2 一个 $O(n \lg n)$ 算法

■ 算法: Alg3.1 异步leader选举

```
var asleep init true;
```

```
upon receiving no msg:
```

```
  if asleep then{
```

```
    asleep:=false; //每个结点唤醒后不再进入此代码
```

```
    send<probe, id, 0, 0> to left and right;
```

```
  }
```

```
upon receiving <probe, j, l, d> from left (resp, right):
```

```
  if(j=id) then //收到自己id终止, 省略发终止msg
```

```
    terminate as the leader;
```

```
  if(j>id) and (d<2l) then //向前转发probe msg
```

```
    send <probe, j, l, d+1> to right (resp, left)
```

§ 3.3.2 一个 $O(n \lg n)$ 算法

if($j > id$) and ($d \geq 2^l$) then // 到达最后一个邻居仍未没收
 send $\langle \text{reply}, j, l \rangle$ to left(resp, right) // 回答
// 若 $j < id$, 则没收probe消息

upon receiving $\langle \text{reply}, j, l \rangle$ from left (resp, right):

if $j \neq id$ then

 send $\langle \text{reply}, j, l \rangle$ to right (resp, left); // 转发reply

else // $j = id$ 时, P_i 已收到一个方向的回答msg

 if already received $\langle \text{reply}, j, l \rangle$ from right (resp, left)

 then // 也收到另一方向发回的reply

 send $\langle \text{probe}, id, l+1, 0 \rangle$ to left and right;

 // P_i 是 phase l 的临时leader, 继续下一阶段

§ 3.3.2 一个 $O(n \lg n)$ 算法

■ 分析

① **正确性**：因为具有最大id的探测消息不会被任何结点没收的，所以该处理器将作为leader终止算法；另一方面，没有其他探测消息能够周游整个环而不被吞没。因此，最大id的处理器是算法选中的唯一的leader。

② **msg复杂性（最坏情况下）**

在phase l 里：

- ◆ 一个处理器启动的msg数目至多为： $4 \cdot 2^l$
- ◆ 有多少个处理器是启动者呢？
 - $l=0$ ，有 n 个启动者（最多）
 - $l \geq 1$ ，在 $l-1$ 阶段结束时成为临时leader的节点均是启动者

§ 3.3.2 一个 $O(n \lg n)$ 算法

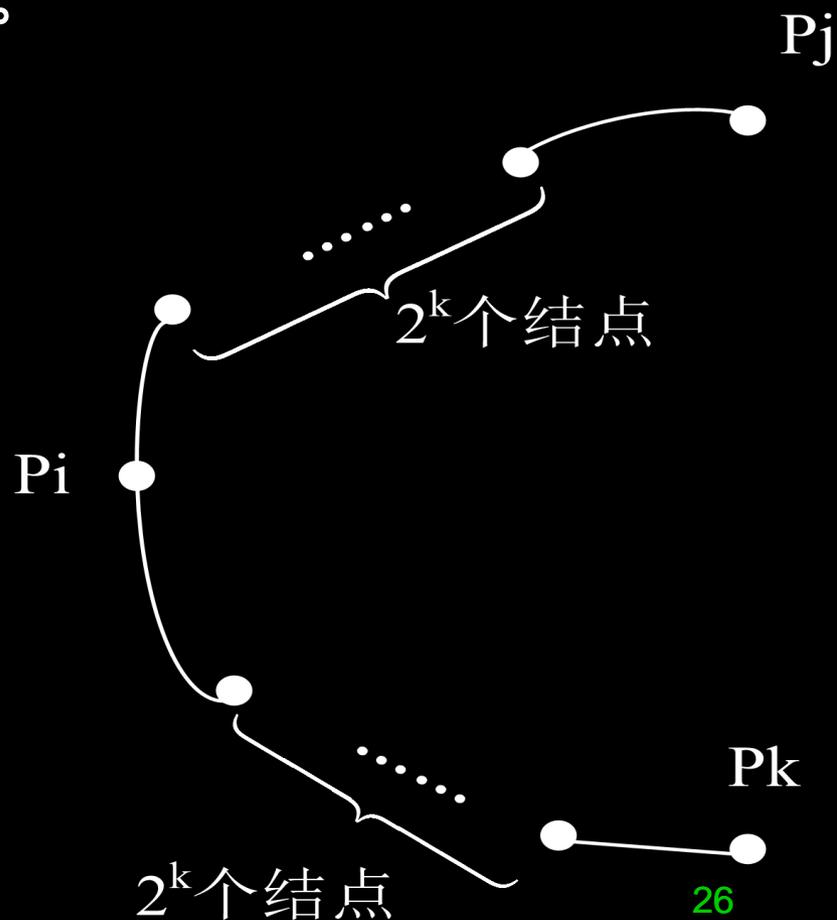
Lemma 3.3 对每个 $k \geq 1$, 在phase k 结束时, 临时leader数至多为 $n/(2^k+1)$.

pf: 若一结点 P_i 在 k 阶段结束时是一临时leader, 则在 P_i 的 2^k -邻居里每个结点的id均小于 P_i 的id。

在该阶段里, 距离最近的两个临时leader P_i 和 P_j 必满足:

P_i 的 2^k 邻居的左边恰好 P_j 的 2^k -邻居的右边, 即 P_i 和 P_j 之间有 2^k 个处理器。

因此, 在phase k 里临时leader的最大数目必是以上述方式分布的, 因为每 2^k+1 个结点至多有一个临时leader, 所以leader数至多是 $n/(2^k+1)$ 。



§ 3.3.2 一个 $O(n \lg n)$ 算法

Th3.4. 存在一个异步的leader选举算法，其msg复杂性为 $O(n \lg n)$.

Pf: 由lemma3.3知，知道phase $\lg(n-1)$ 时只剩下一个leader(最后的leader). msg总数:

$$4n + \sum_{l=1}^{\lg(n-1)} (4 \cdot 2^l) \frac{n}{2^{l-1} + 1} + n \leq 8n \lg n$$

i) phase 0: msg数为 $4n$.

ii) 终止msgs: n .

Note: 双向通信. 该msg复杂性的常数因子不是最优的.

下次继续！