Election

- Objectif
  - ✓ Choisir un processus/site parmi N
    - ➡ Reprise sur panne
    - ➡ Algorithme avec initiateur / collecteur
    - ➡ ...
  - ✓ Selon le problème, les capacités des sites diffèrent
    - ➡ Notion de capacité par site : c_i
    - ➡ Relation d'ordre strict sur ces capacités
  - ✓ On parle d'*élection* lorsque l'application décide par elle même le site le plus apte

- Election sur un anneau unidirectionnel
- Election sur un arbre couvrant
- Election dans un graphe générique

Sur un anneau unidirectionnel

- Algorithme de Chang & Roberts

✓ Le site initiateur P_0 transmet le message $El(c_0, 0, 0)$

✓ Un site $P_{j\neq0}$ qui reçoit le message $El(c_i, i, 0)$:
If $(c_i < c_j)$ transmission $El(c_j, j, 0)$ à succ(P_j)
Else transmission $El(c_i, i, 0)$ à succ(P_j)

✓ Lorsque P_0 reçoit $El(c_i, i, 0)$, il élit le processus P_i
=> *phase de proclamation*
=> il fait circuler le message $Pl(i)$

Operating Systems — R. Stockton Gaines Editor

An Improved Algorithm for Decentralized Extrema-Finding in Circular Configurations of Processes

Ernest Chang
University of Toronto

Rosemary Roberts
University of Waterloo

This note presents an improvement to LeLann's algorithm for finding the largest (or smallest) of a set of uniquely numbered processes arranged in a circle, in which no central controller exists and the number of processes is not known a priori. This decentralized algorithm uses a technique of selective message extinction in order to achieve an average number of message passes of order (n log n) rather than $O(n^2)$.

Key Words and Phrases: decentralized algorithms, distributed systems, operating systems
CR Categories: 4.32, 4.35, 5.25, 5.32

Sur un arbre couvrant

- Si l'initiateur (P_0) est la racine de l'arbre

Phase 1 - Diffusion de la demande des capacités *Demande()*

✓ Initialisation : $(c_{max}, i_{max}) \leftarrow (c_0, 0)$; N_{fils} = |succ()|;
If (N_{fils} > 0), P_0 émet le message *Demande()* → succ();
Else Proclamation(c_{max}, i_{max})
✓ Lorsqu'un site P_i reçoit le message *Demande()* : N_{fils} = |succ()|;
(c_{max}, i_{max}) = (c_i, i) ;
If (N_{fils} > 0), P_i transmet ce message *Demande()* → succ();
Else transmission *REP*(c_{max}, i_{max}) → daddy();

Phase 2 - Remontée des informations *REP*(c_j, j)

✓ Lorsqu'un site P_i, reçoit le message *REP*(c_j, j) d'un de ses fils : N_{fils} = N_{fils} - 1;
(c_{max}, i_{max}) = *max*((c_j, j), (c_{max}, i_{max}));
If (N_{fils} == 0)
If (|daddy()|==0) alors Proclamation(c_{max}, i_{max}) /* Il s'agit de P_0*/
Else transmission *REP*(c_{max}, i_{max}) → daddy();

Phase 3 - Proclamation (P_0) Proclamation(c_{max}, i_{max}) => *ELU*(c_{max}, i_{max})

If(N_{fils} > 0) P_0 émet le message *ELU*(c_{max}, i_{max}) → succ(); Else **FIN**
✓ Lorsqu'un site P_i, reçoit le message *ELU*(c_{max}, i_{max}): N_{fils} = |succ()|; elu ← i_{max} ;
If (N_{fils} > 0) P_i transmet le message *ELU*(c_{max}, i_{max}) → succ();
Else *OK!* → daddy();
✓ Lorsqu'un site P_i, reçoit le message *OK!* d'un de ses fils : N_{fils} = N_{fils} - 1;
If (N_{fils} == 0)
If P_i n'a pas de père alors **FIN**
Else *OK!* → daddy();

Sur un arbre couvrant

- Si l'initiateur P_k n'est pas la racine P_0 de l'arbre
  - ✓ Soit on ajoute une phase d'*InitDemande*() du noeud jusqu'à la racine => algo précédent
  - ✓ Soit on applique un algo plus général :

Phase 1 - Initialisation : P_k émet le message *Demande*() → succ() + daddy();
Lorsqu'un site non feuille reçoit une *Demande*() :
  si *Demande*() ← daddy : état=*descendant* & transmission *Demande*() → succ();
  si *Demande*() ← fils (P_{fils}) : état=*ascendant* & *Demande*() → succ()\fils + daddy();
Lorsqu'un noeud feuille P_i reçoit le message *Demande*(): *REP*(c_i, *i*) → daddy();

Phase 2 - Remontée des informations *REP*(c_1, *1*)
- Lorsqu'un site *descendant* $P_{j\neq k}$ a reçu tous les messages *REP*(c_i, *i*) de ses fils:
il transmet message *REP*(c_1, *1*) → daddy() | c_1 = max ({c_i},c_j)
- Lorsqu'un site *ascendant* $P_{j\neq k}$ a reçu le message *REP*(c_p, *p*) de son père *p* et *REP*(c_i, *i*) de tous ses fils excepté P_{fils} : il transmet *REP*(c_1, *1*) → P_{fils} | c_1 = max({c_i}, c_p, c_j)
- Lorsque P_0 *(racine)* a reçu le message *REP*(c_i, *i*) de tous ses fils excepté P_{fils} :
il transmet *REP*(c_1, *1*) → P_{fils} | *c1* = max ({c_i}, c_0)

Phase 3 - Élection
Lorsque le site P_k a reçu le message *REP*(c_p, *p*) de son père *p* et *REP*(c_i, *i*) de tous ses fils : il élit P_1 | c_1 = max({c_i}, c_p, c_k)

Phase 4 - Proclamation *ELU*(*1*)
P_k proclame le résultat par l'émission du message *ELU*(*1*) → succ() + daddy();
Lorsqu'un site reçoit le message *ELU*(*1*) : elu ← 1 ; If (site_non_feuille) :
  si *ELU*(*1*) ← daddy : il transmet ce message *ELU*(*1*) → succ();
  si *ELU*(*1*) ← fils : il transmet ce message *ELU*(*1*) → succ()\fils + daddy();

Sur un graphe générique

- Avec diffusion, the ***Bully algorithm*** de Garcia Molina
  - ✓ Compléxité message au pire : 3N(N-1)/2

Le site initiateur P_0 diffuse le message *El*(c_0, 0) à tous les processus et passe à l'état *Enquête*

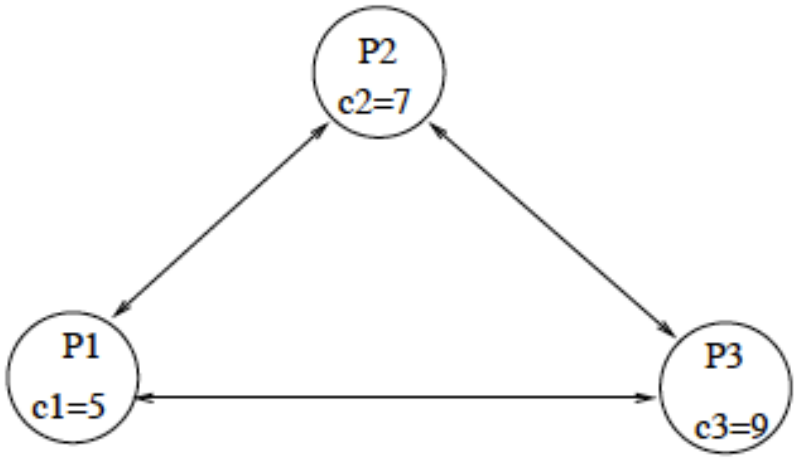
A la réception d'un *El*(c_k, *k*), un processus P_i :

– s'il est dans l'état *Peut-être* :
If ($c_i \leq c_k$) : il répond *Ack*(*i*) à P_k
Else : il diffuse *El*(c_i, *i*) et passe dans l'état *Enquête*

– s'il est dans l'état *Enquête* :
If ($c_i > c_k$) : il ignore ce message
Else : il répond *Ack*(*i*) à P_k et passe dans l'état *Perdu*

– s'il est dans l'état *Perdu* :
If ($c_i \leq c_k$) : il répond *Ack*(*i*) à P_k

Sur un graphe générique



- Déroulez sur cet exemple avec P_1 comme racine...(choisissez un scénario défavorable - P_1 commence, P_2 et P_3 répondent...)
- Comparez avec et sans l'introduction de l'état «Enquete»

Le site initiateur P_0 diffuse le message $El(c_0, 0)$ à tous les processus et passe à l'état *Enquête*

A la réception d'un $El(c_k, k)$, un processus P_i :

– s'il est dans l'état *Peut-être* :
If ($c_i \leq c_k$) : il répond $Ack(i)$ à P_k
Else : il diffuse $El(c_i, i)$ et passe dans l'état *Enquête*

– s'il est dans l'état *Enquête* :
If ($c_i > c_k$) : il ignore ce message
Else : il répond $Ack(i)$ à P_k et passe dans l'état *Perdu*

– s'il est dans l'état *Perdu* :
If ($c_i \leq c_k$) : il répond $Ack(i)$ à P_k

Sur un graphe générique

- Sans diffusion (fiable), le principe général
  - ✓ On ne communique qu'avec ses voisins directs
  - ✓ Compléxité message $M + N - 1$ (M : nombre de liens orientés)

- Pourquoi une telle compléxité ?

Phase 1 - Diffusion adjacente de la demande des capacités

On demande à tous ses voisins leur capacités.
Si premier demandeur : on répond c_i
Sinon : on répond NOK.

Phase 2 - Remontée des informations

Lorsqu'un site aura reçu une réponse (soit un c_i soit un NOK) de tous ses voisins : il répond c_{max} au premier demandeur

Phase 3 - Proclamation

Le site initiateur enverra le proclamation à tous les sites qu'il connait.
Les autres sites ne relaient cette proclamation qu'aux sites !NOK