Université Jopseph Fourier – UFR-IMA

MST ESI2 : Examen d'algorithmique distribuée

Durée 2h

Tous documents autorisés – Le barême est indicatif

11 Avril 2000

Election par extinction

Il existe une forme particulière d'exclusion mutuelle qui consiste à ne faire exécuter une action donnée qu'une seule fois même si plusieurs processus le demandent. Une façon de garantir qu'à un instant donné de l'exécution d'un système distribué, seul un site effectue l'action est d'utiliser un algorithme d'élection qui décide du site devant exécuter l'action. Le principe de l'algorithme distribué d'élection par extinction est le suivant :

1. dès qu'un site désire effectuer l'action critique, il lance une élection. Pour cela, il initie une vague pour envoyer aux autres sites un message de candidature portant le nombre de *voix* dont il dispose,
2. si un site candidat reçoit un message de candidature d'un autre site, il **doit** participer à l'élection s'il ne l'a pas déja fait. Il ignore la candidature reçue si le nombre de voix proposé est inférieur aux siennes et il initie sa propre vague de candidature éventuellement. Sinon il accepte la candidature de l'autre site et propage la vague correspondante. [1].
3. Le site gagnant effectue l'action critique. Le gagnant est celui dont la vague n'est pas éliminée.

I - Election unique

Dans un premier temps, on considère qu'une **seule** exécution de l'action critique est à faire dans le système distribué. Naturellement, la façon de conduire l'élection dépend fortement de la géométrie du réseau que l'on considèrera fiable par la suite.

Election sur anneau (2 pts) Compléter le squelette de l'automate de contrôle exécuté sur chaque site du réseau pour faire une élection sur anneau monodirectionnel.

```
boolean élection = false;
int mesvoix ; /* voix obtenue à l'élection courante */
repeter alt
   | besoin_d'élection  && !élection  ==>élection=true;
   |                 mesvoix=Voix();
   |                 lout!<CAND,ICI,mesvoix>
   | lin?<CAND,src,vx> && src!=ICI && élection  ==>
   |                  if(vx,src)>(mesvoix,ICI)
   |                       lout!<CAND,src,vx>
   |                  /* else extinction*/
   | lin?<CAND,src,vx> && src!=ICI && !élection ==> élection=true;
```

[1] On utilise les No des sites en cas de ballottage

```
   |                  mesvoix=Voix();
   |                  if(vx,src)>(mesvoix,ICI)
   |                       lout!<CAND,src,vx>
   |                  else
   |                       lout!<CAND,ICI,mesvoix>
   | lin?<CAND,src, vx> && src==ICI  ==> ACTION();
fin_alt
```

La procédure `int Voix()` permet à un site d'obtenir son quota de voix. L'action critique est appelée `ACTION()`. La variable `ICI` donne l'identité du site l'exécutant. Le booléen `besoin_d'élection` local à chaque site ne devient vrai qu'une seule fois par site mais éventuellement sur plusieurs sites du système.

Election sur clique (2 pts) Dans ce cas, le réseau est complètement maillé. Proposer un algorithme d'élection basé sur la vague dans une clique. Le message de retour de vague est marqué `ACK`. Compléter le squelette suivant:

```
boolean élection = false;
int mesvoix ; /* voix obtenue à l'élection courante */
W /* ensemble des liaisons dont on attend un ACK */
repeter alt  Quelque soit l de L
   |  besoin_d'élection && !élection  ==>élection=true;
   |                mesvoix=Voix(); W=L;
   |                Quelque soit l de W l!<CAND,ICI,mesvoix>
   | l in L  l<CAND,src,vx> && src!=ICI && élection  ==>
   |                  if(vx,src)>(mesvoix,ICI)
   |                       l!<ACK,src,vx>
   |                  /* else extinction*/
   | l in L  l?<CAND,src,vx> && src!=ICI && !élection ==> élection=true;
   |                  mesvoix=Voix(); W=L;
   |                  if(vx,src)>(mesvoix,ICI)
   |                       l!<ACK,src,vx>
   |                  else
   |                       Quelque soit l de W  l'!<CAND,ICI,mesvoix>
   | l in L  l?<ACK,src,vx>  ==> W=W sans l; if(vide(W)) ACTION();               .........
fin_alt
```

Election sur graphe bidirectionnel quelconque (4 pts) Proposer un algorithme d'élection basé sur des vagues de type *echo* explorant de façon récursive le graphe. On notera que, sur un site, on a seulement besoin du souvenir de la vague portant la dernière candidature gagnante.

```
boolean élection = false;
int mesvoix ; /* voix obtenue à l'élection courante */
W /* ensemble des liaisons dont on attend un ACK */
up /* chemin de retour à l'initiateur */
win /* gagnant temporaire */
repeter alt  Quelque soit l de L
   |  besoin_d'élection && !élection  ==>élection=true;
   |                mesvoix=Voix(); W=L;win=ICI;
   |                Quelque soit l de W lout!<CAND,ICI,mesvoix>
   | l in L  l<CAND,src,vx> && src!=ICI && élection && !dejavu[src]  ==>
   |                dejavu[src];
   |                if(vx,src)>(mesvoix,ICI)
   |                        W=L sans l; up= l;win=src;
   |                        Quelque soit l' de W  l'!<CAND,src,vx>
```

```
   |                /* else extinction*/
   | l in L  l?<CAND,src,vx> && src!=ICI && !élection && !dejavu[src] ==>
   |                élection=true;
   |                mesvoix=Voix();
   |                if(vx,src)>(mesvoix,ICI)
   |                        W=L sans l; up= l;win=src;
   |                        Quelque soit l' de W  l'!<CAND,src,vx>
   |                else
   |                       W=L; win=src;
   |                       Quelque soit l de W l!<CAND,ICI,mesvoix>
   | l in L  l?<CAND,src,vx> && dejavu[src] ==>  l!<ACK,src,vx>
   | l in L  l?<ACK,src,vx> && win==src && src==ICI  ==> W=W sans l; if(vide(W)) ACTION();
   | l in L  l?<ACK,src,vx> && win==src && src!=ICI  ==> W=W sans l; up!<ACK,src,vx>;
fin_alt
```

II - Elections multiples

Le régime de fonctionnement du système est tel que des exécutions successives de l'action critique sont nécessaires. Le booléen `besoin_d'élection` local à chaque site passe de vrai à faux (et réciproquement) un nombre quelconque de fois dans le système. Il faut donc être capable de lancer des élections successives.

Séquentialisation implicite Dans un premier temps, on suppose que `besoin_d'élection` ne redevient vrai que postérieurement à l'exécution de l'action résultant de l'élection précédente. Il s'agit donc pour un site perdant de distinguer les candidatures d'élections différentes. Un mécanisme d'horloge logique d'élection parait approprié à ce type de problème : **Comme dans une période donnée pour un site donné, `besoin_d'élection` ne devient pas nécessairement vrai, un site perdant ne peut savoir si la bascule à vrai de cette variable est pour la période d'élection en cours ou la suivante. Il n'est pas possible de savoir à la réception d'un paquet s'il y eu déclenchement ou non de l'élection dans cette phase là puisqu'un site perdant ne sait pas si l'élection précédente est terminée. Pour dater, il faut faire l'hypothèse du déclenchement synchrone de `besoin_d'élection` sur la partie des sites concernées Cette hypoyhèse synchrone implique que si `besoin_d'élection` bascule sur un site, il devient avant réception de tout message d'élection d'un autre site.**

1. Peut on borner cette horloge modulo X ? Que vaut X au moins ? **Puisque une élection suit la terminaison de l'action de l'élection précédente, un site perdant a juste besoin de savoir si l'élection en cours est terminée ou si on débute la suite. On peut donc utiliser la méthode du bit alterné soit modulo 2 pour les messages. (2 pts)**

2. Modifier les trois algorithmes précédents pour permettre de multiples élections **(1+1+2 pts)**

```
int HE=0;
int mesvoix ; /* voix obtenue à l'élection courante */

repeter alt
   | besoin_d'élection  ==>
   |             HE=+1%2;
   |             mesvoix=Voix();
   |             lout!<CAND,ICI,HE,mesvoix>
   | lin?<CAND,src,h,vx> && src!=ICI && h==HE   ==>
   |             if(vx ,src)>(mesvoix,h,ICI)
   |                   lout!<CAND,src,h,vx>
   |             /* else extinction*/
   | lin?<CAND,src,h,vx> && src!=ICI && h!=HE==>
   |             HE=h;
```

```
   |             mesvoix=Voix();
   |             if(vx,src)>(mesvoix,ICI)
   |                  lout!<CAND,src,h,vx>
   |             else
   |                  lout!<CAND,ICI,h,mesvoix>
   | lin?<CAND,src, vx> && src==ICI  ==> ACTION();
fin_alt

int mesvoix ; /* voix obtenue à l'élection courante */
HE =0; /* horloge d'élection */
W /* ensemble des liaisons dont on attend un ACK */
repeter alt  Quelque soit l de L
   |  besoin_d'élection &&  ==>HE+1%2
   |                mesvoix=Voix(); W=L;
   |                Quelque soit l de W l!<CAND,ICI,HE,mesvoix>
   | l in L  l<CAND,src,h,vx> && src!=ICI && h==HE  ==>
   |                  if(vx,src)>(mesvoix,ICI)
   |                       l!<ACK,src,h,vx>
   |                  /* else extinction*/
   | l in L  l?<CAND,src,vx> && src!=ICI && h!=HE ==> h=HE;
   |                  mesvoix=Voix(); W=L;
   |                  if(vx,src)>(mesvoix,ICI)
   |                       l!<ACK,src,h,vx>
   |                  else
   |                       Quelque soit l de W  l'!<CAND,ICI,h,mesvoix>
   | l in L  l?<ACK,src,h,vx>  ==>
   |            W=W sans l; if(vide(W)) ACTION();              .........
fin_alt

boolean élection = false;
int mesvoix ; /* voix obtenue à l'élection courante */
W /* ensemble des liaisons dont on attend un ACK */
up /* chemin de retour à l'initiateur */
win /* gagnant temporaire */
int HE=0;
repeter alt  Quelque soit l de L
   |  besoin_d'élection   ==>HE+=1%2;
   |                mesvoix=Voix(); W=L;win=ICI;
   |                Quelque soit l de W lout!<CAND,ICI,HE,mesvoix>
   | l in L  l<CAND,src,h,vx> && src!=ICI && H==HE && !dejavu[src]  ==>
   |                dejavu[src];
   |                if(vx,src)>(mesvoix,ICI)
   |                        W=L sans l; up= l;win=src;
   |                        Quelque soit l' de W  l'!<CAND,src,h,vx>
   |                /* else extinction*/
   | l in L  l?<CAND,src,vx> && src!=ICI && h!=HE&& !dejavu[src] ==> HE=h;
   |                mesvoix=Voix();
   |                if(vx,src)>(mesvoix,ICI)
   |                        W=L sans l; up= l;win=src;
   |                        Quelque soit l' de W  l'!<CAND,src,h,vx>
   |                else
   |                       W=L; win=src;
   |                       Quelque soit l de W l!<CAND,ICI,h,mesvoix>
   | l in L  l?<CAND,src,h,vx> && h==HE &&  dejavu[src] ==>  l!<ACK,src,h,vx>
```

```
   | l in L  l?<ACK,src,h,vx> && win==src && src==ICI  && h==HE ==> W=W sans l; if(vide(W)) ACTION();
   | l in L  l?<ACK,src,h,vx> && win==src && src!=ICI && h==HE  ==> W=W sans l; up!<ACK,src,h,vx>;
fin_alt
```

Séquentialisation explicite Ici, on suppose que `besoin_d'élection` peut redevenir vrai n'importe quand. Il faut donc éviter de lancer une nouvelle élection avant la fin de l'exécution de l'action de l'élection précédente.

1. Comment synchroniser le lancement d'une élection avec la fin de l'action de l'élection précédente. Donner le principe d'une solution s'appliquant aux réseaux précédents.**(1 pt) On utilise 2 horloges HE et HF. Une pour les vagues lançées et une pour les terminées. On ne peut initier une nouvelle vague que si HE==HF. Le gagnant débloque tout le monde en lançant une vague de déblocage.**

2. Proposer une solution pour graphe bidirectionnel quelconque.**(3 pts)**

```
boolean élection = false;
int mesvoix ; /* voix obtenue à l'élection courante */
W /* ensemble des liaisons dont on attend un ACK */
up /* chemin de retour à l'initiateur */
win /* gagnant temporaire */
int HE=0, HF=0;
repeter alt  Quelque soit l de L
   |  besoin_d'élection  && HE==HF ==>HE+=1%2; /* synchronisation */
   |              mesvoix=Voix(); W=L;win=ICI;
   |              Quelque soit l de W lout!<CAND,ICI,HE,mesvoix>
   | l in L  l<CAND,src,h,vx> && src!=ICI && H==HE && !dejavu[src]  ==>
   |              dejavu[src];
   |              if(vx,src)>(mesvoix,ICI)
   |                      W=L sans l; up= l;win=src;
   |                      Quelque soit l' de W  l'!<CAND,src,h,vx>
   |              /* else extinction*/
   | l in L  l?<CAND,src,vx> && src!=ICI && h!=HE&& !dejavu[src] ==> HE=h;
   |              mesvoix=Voix();
   |              if(vx,src)>(mesvoix,ICI)
   |                      W=L sans l; up= l;win=src;
   |                      Quelque soit l' de W  l'!<CAND,src,h,vx>
   |              else
   |                     W=L; win=src;
   |                     Quelque soit l de W l!<CAND,ICI,h,mesvoix>
   | l in L  l?<CAND,src,h,vx> && h==HE &&  dejavu[src] ==>  l!<ACK,src,h,vx>
   | l in L  l?<ACK,src,h,vx> && win==src && src!=ICI && h==HE  ==> W=W sans l; up!<ACK,src,h,vx>;
   | l in L  l?<ACK,src,h,vx> && win==src && src==ICI  && h==HE ==>
   |                    W=W sans l; if(vide(W)) ACTION();
   |                    W=L;
   |                    Quelque soit l' de L  l'!<FIN,src>
/* vague de fin */
   | l in L  l?<FIN,src> && fin  ==>  l!<OKFIN,src> /* boucle */
   | l in L  l?<FIN,src> && !fin  ==> fin=true;
   |                    W=W sans l; if(vide(W)) ACTION();
   |                    W=L sans l; up=l;
   |                    Quelque soit l' de L  l'!<FIN,src>
   | l in L  l?<OKFIN,src> && src!=ICI  ==> W=W sans l;
   |                     if(vide(W))up!<OKFIN,src>;HF+=1%2
```

```
   | l in L  l?<OKFIN,src> && src==ICI  ==> W=W sans l;
   |                       if(vide(W)) HF+=1%2
fin_alt
```

III - Liaison non FIFO

- La sérialisation des élections est elle toujours possible ? pourquoi ? **(1 pt) Oui. Ceci provient de la sérialisation explicite qui ne lance une vague qu'après la fin de la précédente donc quand les liaisons FOFO ont été vidées**
- Les algorithmes précédents sont ils corrects ? pourquoi ?**(1 pt)Oui car il implante cette synchronisation sans faire d'hypothèse sur le caractère FIFO des liaisons**