Programmation répartie

Des algorithmes répartis

Hinde Bouziane

bouziane@lirmm.fr

Introduction (rappel)

Algorithmique répartie

La discipline visant à développer des algorithmes dédiés aux systèmes distribués et prenant en compte les spécificités de ces systèmes.

Quelques spécificités des systèmes répartis

- Répartition géographique des entités de calcul.
- Pas de mémoire commune ni d'horloge commune, donc communications et coordination via des échanges de messages.
- Connaissance partielle / limitée du système.
- Synchronie (mode synchrone ou asynchrone).
- Présence d'erreurs, de pannes ou de défaillances.
- Le non déterminisme (chaque exécution produit éventuellement un nouveau scénario même si les entrées sont les mêmes)

Des problèmes fondamentaux de l'algorithmique répartie

- Diffusion : permettre à un processus d'envoyer un même message à tous les autres processus en respectant des propriétés d'ordre (FIFO, causal, total) dans la transmission des messages.
- Élection : partir d'une configuration où les processus sont dans un même état, pour arriver à une configuration où un seul processus est dans un état "gagnant" et tous les autres dans un état "perdant".
- Exclusion mutuelle : assurer que l'exécution d'une portion de code manipulant une ressource partagée (section critique) ne soit autorisée qu'à un seul processus à la fois.
- Prévention / détection d'interblocage : l'interblocage se produit lorsqu'un ensemble de processus est tel que chacun d'eux tient au moins une ressource, et pour progresser, est en attente d'une ressource prise par l'un des autres.
- Détection de la terminaison : détecter si un algorithme se termine de façon implicite (sans que les processus n'appellent la fonction exit par exemple).
- ...

Problématique

Assurer qu'à tout un instant, seul un processus puisse exécuter une partie d'un code concurrent, appelée section critique.
Autrement dit, à tout instant, il peut y avoir 0 ou 1 processus qui accède à une ressource / section critique.

Exemples d'utilisation

- Mécanismes de réservations de billets, etc.
- Opérations bancaires.
- Prise de rendez-vous
- Partage de ressources dans le Cloud.
- etc.

Propriétés à assurer

Correction / sûreté (safety)

A tout instant, il ne peut y avoir au plus qu'un seul processus en section critique.

Vivacité / pas de famine (liveness)

Tout processus souhaitant entrer en section critique y arrivera en un temps fini.

Dans la littérature : deux familles d'algorithmes

Solutions à base de permissions

Il est possible d'entrer en section critique après la réception de la permission d'un ensemble de processus.
Exemples :

- algo. de Lamport , algo. de Ricart et Agrawala, algo. de Maekawa, algo. de Carvalho et Roucairol, algo. de Sanders, algo. de Singhal.

Solutions à base d'un jeton

Un jeton unique circule entre les processus. La possession du jeton donne le droit exclusif d'entrer en section critique.
Exemples :

- algo. de Le Lann, algo. de Martin, algo. Nielsen-Mizuno, algo. de Raymond, algo. de Naimi-Tréhel, algo. de Naimi-Tréhel-Arnold, algo de Susuki-Kasami, algo de Chang, Singhal et Liu.

Utilisation d'un algorithme d'exclusion mutuelle

Schéma global d'une section critique dans un Processus P_i

... // état de P_i : "dehors" (en dehors de la section critique).

...

demander_accès(...) // état de P_i : "demandeur".

...

... // section critique. Etat de P_i : "dedans".

...

libérer_accès(...) // retour à l'état : "dehors".

...

...

Algo. Ricart et Agrawala - permission individuelle (1/8)

Hypothèses

- Soit $\Pi = \{P_1, P_2, ..., P_N\}$ l'ensemble des processus d'un système réparti S.
- Réseau de communication quelconque (mais complètement maillé).
- Communications fiables.
- Processus corrects.

Principe

- Un processus qui souhaite accéder à une section critique, envoie une demande à tous les autres processus et attend de recevoir $N - 1$ permissions. Une demande contient au moins la date de la demande (HL).
- Un processus donne sa permission à un site ayant envoyé une demande, s'il n'est pas en section critique ou s'il attend la section critique mais que sa requête est moins prioritaire. Un site peut alors donner plusieurs permissions à différents sites de manière simultanée.

Algo. Ricart et Agrawala - permission individuelle (2/8)

Initialisation :

$etat_i$: état $\in$ {dehors, demandeur, dedans}, initialisé à dehors ;

h_i : horloge logique initialisée à 0. Rappel : $(h, i) < (k, j) \Leftrightarrow (h < k$ ou $(h = k$ et $i < j))$;

$last_i$: date de la dernière demande de P_i, initialisée à 0 ;

$prioritaire_i$: booléen ;

$attendus_i$: ensemble des sites devant fournir une permission, initialisé à $\emptyset$;

$differes_i$: ensemble des sites mis en attente par P_i, initialisé à $\emptyset$;

Algo. Ricart et Agrawala - permission individuelle (3/8)

Procédure demander_accès() :

$etat_i \leftarrow demandeur$;

$h_i \leftarrow h_i + 1$;

$last_i \leftarrow h_i$;

$attendus_i \leftarrow \Pi - \{P_i\}$;

pour $P_j \in \Pi - \{P_i\}$ **faire**

  envoyer $< REQUEST, last_i, i >$ à P_j;

attendre($attendus_i = \emptyset$) ;

$etat_i \leftarrow dedans$;

Algo. Ricart et Agrawala - permission individuelle (4/8)

Procédure libérer_accès() :

$etat_i \leftarrow dehors$;

pour $P_j \in differes_i$ **faire**

  envoyer $< PERMISSION, i >$ à P_j;

$differes_i \leftarrow \emptyset$;

Algo. Ricart et Agrawala - permission individuelle (5/8)

Lors de la réception d'un message $< REQUEST, date_j, j >$ **:**

$h_i \leftarrow max(h_i, date_j)$;

$prioritaire_i \leftarrow (etat_i \neq dehors)$ et $(last_i, i) < (date_j, j)$;

si $prioritaire_i$ **alors**

  $differes_i \leftarrow differes_i \cup \{P_j\}$;

sinon

  envoyer $< PERMISSION, i >$ à P_j;

Lors de la réception d'un message $< PERMISSION, j >$ **:**

$attendus_i \leftarrow attendus_i - \{P_j\}$;

Algo. Ricart et Agrawala - permission individuelle (6/8)

Sûreté ?

- Deux processus P_i et P_j peuvent-ils être en même temps en section critique ?
- Deux messages $< REQUEST, date_i, i >$ et $< REQUEST, date_j, j >$ ont été envoyés et deux messages $< PERMISSION, i >$ et $< PERMISSION, j >$ ont été reçus.

Preuve par l'absurde : deux cas

1. P_j envoie $< PERMISSION, j >$ à P_i suivi de $< REQUEST, date_j, j >$.
   - Exercice.
2. Envois simultanés de $< REQUEST, date_i, i >$ et $< REQUEST, date_j, j >$.
   - Exercice.

Algo. Ricart et Agrawala - permission individuelle (7/8)

Vivacité ?

Est ce que tout processus souhaitant entrer en section critique y arrivera en un temps fini ?

Preuve

Exercice.

Algo. Ricart et Agrawala - permission individuelle (8/8)

Complexité en nombre de messages

Combien de messages nécessite une utilisation de la section critique ?

- Réponse : ?

Complexité en temps

- Le but est d'utiliser au maximum la section critique.
- On suppose un temps de transit d'un message borné par T.
- Quelle est la durée durant laquelle la section critique n'est pas utilisée bien qu'elle soit demandée ?
  - Le pire des cas ? Réponse : ?
  - Le meilleur des cas ? Réponse : ?

Algo. Carvalho et Roucairol - permission individuelle (1/8)

Principe

- Peu être vu comme une amélioration de l'algorithme précédent avec une diminution du nombre de messages par demande d'entrée en section critique.
- Les processus maintiennent de l'information sur la liste des permissions. Un processus P_i lorsqu'il obtient la permission d'un autre processus P_j, il peut considérer qu'il a cette permission tant que P_j ne lui a pas envoyé une nouvelle demande.
- Ainsi, un processus n'enverra pas de demande aux sites non-demandeurs et considère qu'il a déjà leur permission pour éventuellement entrer plusieurs fois en section critique.

Algo. Carvalho et Roucairol - permission individuelle (2/8)

Une solution de mise en oeuvre

- A partir de l'algo. de Ricart et Agrawala.
- Associer à tout couple de processus distincts P_i et P_j un et un seul message *permission*(i, j) (ou son synonyme *permission*(j, i)).
- Si *permission*(i, j) est chez P_i, alors P_i a la permission de P_j et inversement.
- Initialement *permission*(i, j) est chez P_i ou P_j (pas les deux !).
- Lorsque le processus P_i invoque l'opération *demander_accès()*, il doit demander les permissions manquantes, ie. envoyer une demande aux processus qui détiennent ces permissions.
- L'ensemble $attendus_i$ contient les processus à qui P_i doit envoyer une demande.

  $attendus_i = \{P_j$ tel que P_i ne possède pas *permission*$(i, j)\}$

Algo. Carvalho et Roucairol - permission individuelle (3/8)

Exercice

Donner un algorithme en modifiant celui de Ricart et Agrawala.

Algo. Carvalho et Roucairol- permission individuelle (8/8)

Sûreté et vivacité (exercice)

- Donner une preuve pour ces deux propriétés.

Complexité en nombre de messages

Combien de messages nécessite une utilisation de la section critique ?

- Réponse : ?

Complexité en temps

- Quelle est la durée durant laquelle la section critique n'est pas utilisée bien qu'elle soit demandée ?
  - Réponse : ?

Algo. Raymond - circulation d'un jeton (1/8)

Hypothèses

- Soit $\Pi = \{P_1, P_2, ..., P_N\}$ l'ensemble des processus d'un système réparti S.
- Réseau de communication quelconque mais avec une topologie logique sous forme d'un arbre.
- Communications fiables.
- Processus corrects.
- Aucun besoin d'horloge logique.

Algo. Raymond - circulation d'un jeton (1/7)

Principe

- Un jeton circule entre processus dans un arbre statique orienté où seule la direction des liens change de sens pendant l'exécution de l'algorithme.
- La racine de l'arbre est le processus possesseur du jeton (le seul à pouvoir entrer en section critique) et chaque noeud de l'arbre fait office de mandataire pour ses fils directs.
- Lorsqu'un processus souhaite entrer en section critique ou lorsqu'il reçoit une demande, il stocke cette demande dans une file locale FIFO, passe à l'état *demandeur* et retransmet (si ce n'est pas déjà fait) la demande à son père.
- Quand le processus à la racine sort de la section critique, il envoie le jeton au premier élément de sa file locale et prend ce dernier comme père.
- Quand un processus reçoit le jeton, il dépile le 1^{er} élément de sa file. S'il s'agit de lui même, il entre en SC, sinon il retransmet le jeton au processus correspondant, choisit ce dernier comme nouveau père et si la file n'est toujours pas vide, il envoie une nouvelle demande à ce père.

Algo. Raymond - circulation d'un jeton (2/7)

Initialisation :

$etat_i$: état $\in$ {dehors, demandeur, dedans}, initialisé à dehors ;

$pere_i$: processus père de P_i, initialisé en fonction de la configuration initiale du système;

$file_i$: file FIFO de processus, initialisée à $\emptyset$;

Algo. Raymond - circulation d'un jeton (3/7)

Procédure demander_accès() :

si $pere_i \neq nil$ **alors**

- ajouter P_i dans $file_i$;
- **si** $etat_i = dehors$ **alors**
  - $etat_i \leftarrow demandeur$;
  - envoyer $< REQUEST, i >$ à $pere_i$;
- attendre($pere_i = nil$) ;

$etat_i \leftarrow dedans$;

Algo. Raymond - circulation d'un jeton (4/7)

Procédure libérer_accès() :

$etat_i \leftarrow dehors$;
si $file_i \neq \emptyset$ **alors**
    $pere_i \leftarrow depiler(file_i)$;
    envoyer $< JETON, i >$ à $pere_i$;
    si $file_i \neq \emptyset$ **alors**
        $etat_i \leftarrow demandeur$;
        envoyer $< REQUEST, i >$ à $pere_i$;

Algo. Raymond - circulation d'un jeton (5/7)

Lors de la réception d'un message $< REQUEST, j >$ **:**

si $pere_i = nil$ *et* $etat_i = dehors$ **alors**

- $pere_i \leftarrow P_j$;
- envoyer $< JETON, i >$ à $pere_i$;

sinon

- **si** $pere_i \neq P_j$ **alors**
  - ajouter P_j dans $file_i$;
  - **si** $etat_i = dehors$ **alors**
    - $etat_i \leftarrow demandeur$;
    - envoyer $< REQUEST, i >$ à $pere_i$;

Algo. Raymond - circulation d'un jeton (6/7)

Lors de la réception d'un message $< JETON, j >$ **:**

$pere_i \leftarrow depiler(file_i)$;

si $pere_i = P_i$ **alors**

  $pere_i \leftarrow nil$;

sinon

  envoyer $< JETON, i >$ à $pere_i$;

  si $file_i \neq \emptyset$ **alors**

    $etat_i \leftarrow demandeur$;

    envoyer $< REQUEST, i >$ à $pere_i$;

  sinon

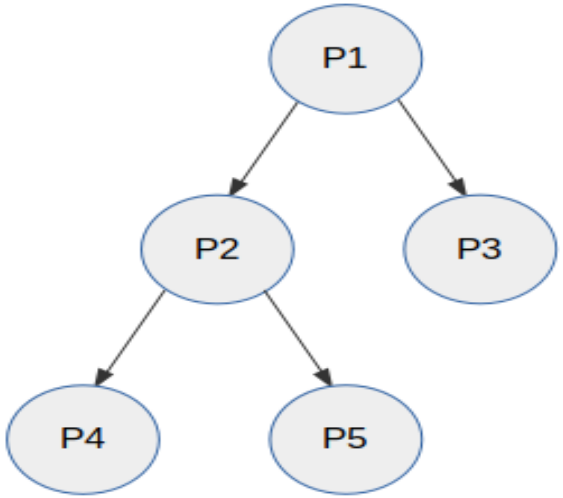
    $etat_i \leftarrow dehors$;

Algo. Raymond - circulation d'un jeton (7/7)

Exercice

Soit un système réparti illustré dans la figure

- Qui a le jeton ?
- Expliquer ce qui se produit à l'exécution si P_3 et P_5 font une demande d'entrée en section critique, puis le processus P_4. Nous supposons que les trois demandes parcourent l'arbre avant la transmission du jeton.



Complexité

Quelle est la complexité de l'algorithme en terme de nombre de messages ?

- le pire cas : ?
- sinon : ?

Problématique

Parmi un ensemble de processus, choisir un et un seul processus (et le faire connaître de tous). Ce processus élu est appelé "leader".

Exemples d'utilisation

Le plus commun est de choisir un leader pour coordonner une activité et qu'une élection se produit quand le leader actuel tombe en panne.

- Systèmes utilisant une exclusion mutuelle centralisée.
- Algorithmes de consensus (exemple : Paxos (L. Lamport) dans les infrastructures comme Amazon, Google, Microsoft, etc).
- Algorithmes de réplication "Primary-Backup"
- Synchronisation de Berkeley
- Choix d'un master pour Bigtable
- Choix d'un serveur répondant à une requête (Paypal, etc.)
- etc.

Propriétés à assurer

Correction / sûreté

Un seul processus est élu "leader".

Vivacité

Un processus doit être élu en un temps fini.

Dans la littérature

Quelques hypothèses majeures

- L'élection peut être déclenchée par n'importe quel processus, éventuellement par plusieurs.
- En général, les processus on des identifiants différents et comparables. On peut par exemple choisir le processus ayant l'identifiant le plus grand ou le plus petit, etc.
- En fonction de l'algorithme, des processus peuvent ou ne doivent pas tomber en panne pendant l'élection.

Exemples

- algo. de Le Lann, algo. de Chang et Roberts, algo. de Peterson, algo. de Segall, algo. de Franklin, algo. de Dolev, Klawe et Rodeh, l'algorithme du plus fort (bully algorithm) par Garcia-Molina, l'algorithme Yo-Yo par N. Santoro, etc.

Utilisation d'un algorithme d'élection

Il n'y a pas de schéma type, mais l'idée est que n'importe quel processus, suite à un évènement, peut déclencher une élection. C'est le cas par exemple lorsque ce processus détecte la panne de l'actuel coordinateur.

Algo. de Chang et Roberts (1/7)

Hypothèses

- Soit $\Pi = \{P_1, P_2, ..., P_N\}$ l'ensemble des processus d'un système réparti S.
- Réseau en anneau virtuel unidirectionnel et plusieurs candidats possibles.
- Canaux FIFO et communications fiables.
- Processus corrects et durant l'élection, aucune panne ne se produit.

Principe

- Élection du processus ayant l'identifiant le plus grand.
- Un processus qui décide d'être candidat, envoie son identifiant à sont voisin.
- Un processus P_i qui reçoit un identifiant j, compare ce dernier avec i. Soit $i > j$, dans ce cas, la candidature de P_j est éliminée et P_i décide, si ce n'est pas déjà fait, de candidater. Soit $i < j$, dans ce cas, P_i ne fait que retransmettre la candidature de P_j à son voisin. Sinon, la candidature de P_i a fait le tour, P_i est élu et le résultat est annoncé aux autres.

Algo. de Chang et Roberts (2/7)

Initialisation :

$etat_i$: état $\in$ {dormant, candidat, gagnant, perdant}, initialisé à dormant ;

$suivant_i$: Processus suivant P_i dans l'anneau, initialisé en fonction de la configuration du moment du système ;

leader : processus élu à la fin de l'algorithme ;

Dans la suite de l'algorithme, il y a deux types de messages : *ELECT* et *ELECTED*

Algo. de Chang et Roberts (3/7)

Procédure candidature() :

$etat_i \leftarrow candidat$;

envoyer $< ELECT, i >$ à $suivant_i$;

- Quand cette procédure est-elle appelée ?

Algo. de Chang et Roberts (4/7)

Exercice

- Faire une proposition pour compléter l'algorithme.

Algo. de Chang et Roberts (6/7)

Sûreté

- Exercice : Donner une preuve.

Vivacité

- Exercice : Donner une preuve.

Algo. de Chang et Roberts (7/7)

Complexité en nombre de messages

Combien de messages nécessite une élection ? Pour répondre, donner une configuration et la complexité dans les cas suivants :

- Pire cas : ?
- Meilleur des cas : ?
- En moyenne : ?

Algo. du plus fort (Bully) (1/9)

Hypothèses

- Soit $\Pi = \{P_1, P_2, ..., P_N\}$ l'ensemble des processus d'un système réparti S.
- Réseau quelconque.
- Chaque processus connaît les identifiants de tous les autres et peut échanger directement avec eux.
- Communications fiables et synchrones (borne connue sur le temps de communication).
- Pannes possibles des processus durant l'élection.

Algo. du plus fort (Bully) (2/9)

Principe

- Élection du processus (vivant) ayant l'identifiant le plus grand.
- Un processus qui suspecte la panne de l'actuel coordinateur demande une élection à tous les processus ayant un identifiant plus grand - principe de propagation par inondation.
- Un processus qui reçoit une demande d'élection, répond à l'expéditeur si ce dernier a un identifiant inférieur au sien - principe d'intimidation.
- Un processus qui ne reçoit aucune réponse au bout d'un certain temps, constate qu'il est élu -principe du silence.

Algo. du plus fort (Bully) (3/9)

Initialisation :

$etat_i$: état $\in$ {dormant, initiateur, gagnant, perdant}, initialisé à dormant ;

leader : processus élu à la fin de l'algorithme, initialisé à la valeur max des identifiants des processus dans S ;

Dans la suite de l'algorithme, il y a trois types de messages : *ELECT*, *ACK* et *ELECTED*.

Algo. du plus fort (Bully) (4/9)

Procédure *demande_elect*() :

$etat_i \leftarrow initiateur$;

pour $P_j \in \Pi$ *et* $j > i$ **faire**

  envoyer $< ELECT >$ à P_j;

armer délai de garde T ;

Algo. du plus fort (Bully) (5/9)

Lors de la réception d'un message $< ELECT >$ **de** P_j **:**

si $etat_i = dormant$ **alors**

  $demande\_elect()$;

envoyer $< ACK >$ à P_j;

Lors de la réception d'un message $< ELECTED >$ **de** P_j **:**

$etat_i \leftarrow perdant$;

$leader \leftarrow P_j$;

Algo. du plus fort (Bully) (6/9)

Déclenchement du délai de garde T :

si *aucun message ACK reçu* **alors**

- $etat_i \leftarrow gagnant$;
- $leader \leftarrow P_i$;
- **pour** $P_j \in \Pi - \{P_i\}$ **faire**
  - envoyer $< ELECTED >$ à P_j;

sinon

- // attendre la fin de l'élection
- armer délai de garde T' ;

Déclenchement du délai de garde T' :

si *aucun message ELECTED reçu* **alors**

- *demande_elect*();

Algo. du plus fort (Bully) (7/9)

Exercice

- dérouler l'algorithme sur des exemples en l'absence et en présence de pannes.

Discussion

- Que se passe t-il en cas de redémarrage après panne d'un processus ?

Algo. du plus fort (Bully) (8/9)

Sûreté

- Exercice : Donner une preuve.

Vivacité

- Exercice : Donner une preuve.

Algo. du plus fort (Bully) (9/9)

Complexité en nombre de messages

Combien de messages nécessite une élection ? Pour répondre, donner une configuration et la complexité dans les cas suivants :

- Pire cas : ?
- Meilleur des cas : ?

Bully VS Chang et Roberts ?

Quels sont les avantages de l'un par rapport à l'autre ?

Algo. **mystère**

Hypothèses

- Soit $\Pi = \{P_1, P_2, ..., P_N\}$ l'ensemble des processus d'un système réparti S.
- Réseau quelconque.
- Un processus ne communique qu'avec ses voisins. Initialement il ne connaît pas les identifiants de ces derniers.
- Communications fiables.

Exercice

- En entrée : un algorithme et un système réparti.
- Objectif : déduire le principe de l'algorithme. Pour cela, dérouler l'algorithme sur le système en entrée.