Algorithmes Répartis - Jean-Michel Couvreur

Alexandre Masson

14 Janvier 2013

Sujets traités
- Modele synchrone
-

Évaluation
- Controle continu
  - controle sur table
  - mini-projet
- Controle terminal
  - un controle sur table
  - la nature du sujet sera développée en cours

1 Modèle Synchrone

Processus sont attachés à des noeuds d'un graphe orienté et communique par des messages.

Graphe G = (V,E, n = |V|
- outnbr, innbr
- distance(i,j) taille du plus court chemin de i à j
- diametre

M : alphabet des messages plus false pour rien.

pour tout i dans V, un processus est donnée par
- States : ensemble d'états
- start : états initiaux
- msgs : states x out-nbrs -> M U false
- trans : ensemble des transitions

Exécution d'un tour ;
- appliquer msgs pour déterminer les messages à envoyer
- envoyer et recevoir les messages,
- Appliquer trans , pour déterminer l'état suivant.

Remarques
- pas de restriction sur la durée des calculs locaux,
- Déterministes,
- on peux définir des "états d'arrêt",
- Plus tard nous examinerons quelques problèmes :
  - temps de démarrage variable
  - défaillance
  - choix aléatoires

Exécution
- une exécution est un objet mathématique servant à décrire comment un algorithme fonctionne.

- Définition :
  - un état global
  - Messages
  - exécution : C0,M1,N1,C1,M2,N2,...
    - C* : sont des états globaux
    - M* : sont les messages envoyés
    - N* : sont les messages reçus
    - Séquence infinie (mais on peut considérer des préfixes finis)

2 Problème de l'élection d'un leader

- Réseau de processus
- Vous voulez distinguer un processus , le leader.
- Finalement, exactement un processus sera désigné leader
- Motivation : le leader peux prendre en charge :
  - les communications
  - coordination des traitements des données (par exemple ; dans les protocoles de validation)
  - Allocation de ressources
  - etc...

Cas simple l'anneau

- Variantes :
  - bidirectionnel ou unidirectionnel
  - taille de l'anneau n connu ou inconnu

Nous avons besoin de qqc de plus

- besoin de distinguer les processus
- supposons un UID , s'il connaît
- chaque processus démarre en stockant son pid
- il sont comparable et pouvoir faire de l'arithmétique simple dessus est un plus
- dans le réseau tous les identifiants sont différents

Un algorithme

- autors : LeLann , Chang,Roberts
- hypothèse
  - comm unidirectionnelle
  - les proc ne connaissent pas n
  - comparaison d'UID seulement
- Idée :
  - chaque processus envoi son uid dans un message, à relayer étape par étape, autour de l'anneau.
  - le proc compare l'id reçu avec le sien , et envoi a son voisin le plus grand des deux entre le sien et celui reçu.

Preuve , complexité, terminaison

- M, l'alphabet de messages : UID
- état : valeur des variables :
  - u : a pour valeur son pid
  - send, son pid ou false *status : ? ou leader, initialement ?*
- start : défini par l'init des variables
- msgs : transmet la valeur send à son voisin
- trans : défini par le pseudo-code :

- if incomming = v, a UID, then
- ĉase :
- v > u : send := v ;
- v = u : status := leader
- v<u : no-op
- ên̂dcase

Réduire le nombre de communication
- autor : Hirschberg,Sinclair
- hypothèse :
  - bidirectionnel
  - les processus ne connaissent pas n
  - comparaisons d'id seulement
- Idée :
  - Stratégie du doublement
  - Chaque processus envoie son UID dans les deux sens, à des distances de plus en plus grandes (successivement deux fois plus grande à chaque fois)
  - phase aller : un jeton est ignoré si il atteint un noeud dont l'uid est plus grand
  - phase retour : tout le monde passe le jeton *un processus débute la phase suivante que si ses deux jetons reviennent. le processus qui reçoit son propre jeton dans la phase aller est l'élu.*

Élection d'un leader
- Hypothèse
  - UID avec comparaisons.
  - pas d'hypothèse sur la répartition des UID
  - les processus connaissent un majorant du diamètre

3 14 Février 2013

défaillance de processus défaillance byzantine : chaque processus envoie sa valeur, le cas de l'arrêt fonctionne, nb rang = nb panne +1. byz : un menteur au moment du choix, va envoyer au suivant une mauvaise valeur .

collecte d'information exponentielle valeur transmise le long d'un réseau en forme d'arbre. la première couche correspond au résultats reçu à l'étape numéro 1., le seconde correspond à la 2eme étape. le dernier chiffre du numéro marqué au dessus de la "feuille" c'est le dernier processus à avoir envoyer le message reçu. on peut ainsi remonter le chemin parcouru par le message.

Chaque processus utilise la même structure d'arbre.
les noeuds sont étiquetés par des valeurs

initialement les têtes sont étiquetées par la valeur du noeuds
round r> 1
- envoyer le niveau r-1 à tous nœuds (soi même aussi)
- utiliser les messages reçu pour étiqueter le niveau r
- si pas de message utiliser $\perp$

règle de choix pour le cas de la panne d'arret :
- triviale
- on prend toutes les étiquettes de l'arbre
- si un seul élément , on le prend
- sinon on prend par défaut v_0

preuve et complexité
- preuve : il y aura toujours un moment ou il n'y a pas de panne
- complexité : nombre de panne +1 rounds, il envoi a tous ces voisins
- complexité en nombre de message : n-1 * nombre de pannes
- défaut : nombre de feuilles peut croître exponentiellement

3.1 Algorithmes : pannes byzantines

- conditions :
  - accord : pas deux processus sans panne décide différemment
  - si tous les processus sans panne une valeur identique alors elle est choisie par tous les processus sans panne
  - Terminaison : tous les processus sans panne font un choix
- EIG algorithme utilise :
  - un abre
  - f+1 rounds
  - n > 3f

mauvais exemple 3 processus, 1 panne supposons n > 3F :
nouvelle règle de décision :
- notons val(x) , la valeur du nœud x de l'arbre.
- redécorer l'arbre
  - parcours de bas en haut pour trouver la valeur la plus fréquente.

18 Février 2013

4 Modèles asynchrones

4.1 réseaux asynchrones

send/recv de message Graphe G = (e,v), processus associé à chaque noeud et, un canal associé à chaque arc.

processus interface utilisateur : inv, resp.
problème spécifié en terme de trace autorisées au niveau utilisateur.

Canaux différnets types de canaux : FIFO fiable, garantie fiable : perte, duplication, pas d'ordre.
Peut etre définie comme une fonction sur un message d'entrée vers les messages de sortie.
intégrité : préserve le message
pas de perte
pas de duplication
respecte l'ordre.

broadcast et multicast FIFO fiables enttre chaque noeud
différents processus peuvent recevoir de différents émetteurs , mais sans contrainte d'ordre.

4.2 Algorithmes sur des réseau asynchrones

Supposons que des FIFO fiables.
- Élection sur un anneau
- idem sur un réseau
- arbre couvrant minimum

hypothèse : G est un anneau , unidirectionnel ou bidirectionnel, nom pour les voisins , uids

Il stocke ses messages dans sa fifo locale et attend que le canal soit libre pour envoyer.

Algorithme de Peterson Baser sur la notion de phases .
Un processus est actif ou passif, les processus ne fond que relayer les messages
Dans chaque phase la moitié des processus actif devient inactif, ainsi l'élection ne prend au plus n log n phases.

Phase 1 : Envoyer UID de deux prédécesseurs, reste actif si l'uid du milieu est max.

v autrement dit, chaque processus attend de recevoir les uids de ses deux prédécesseurs, ainsi , il envoi son uid au suivant, et soit les trois uids reçus, si celui du milieu n'est pas le max des trois, alors la processus passe en inactif(relay) et ne fera que passer des messages. Sinon, il prend l'uid du milieu et reste actif.

5 4 mars 2013

5.1 Synchroniseurs

on prend un algo sync et on veux le faire tourner dans un env async. simuler les algo sync en async , mise en place de synchroniseurs. On part d'un graphe non orient, on suppose qu'on a l'algo sync qui résout le pb. Un noeud va effectuer l'étape k quand tous les autre sauront finis l'étape k-1.
Le principe : on introduit un synchroniseur, il attend la réponse des états dans les étapes, pour dire , c'est bon passez à la suite. le processus synchroniser , répartis dans tous les processus doit s'occuper de l'envoi/recv des messages. propriétés attendues : well formed : envoi bon messg au bon moment, cycle de vie, après avoir reçu mesg round r, passer au round r+1. Les messages sont envoyés au serveur et renvoyé par le serveur.

il y a aussi l'aspect local, chaque processus attend les message de rand R de tous ces voisins avant de passer à l'étape r+1, et par relation de parallélisme, on peux dans un comportement décaler tous les messages de rang r+1 derrière les message de rang r, afin de retrouver le même comportement que dans le global, car tous le messages de rang r++1 ne seront utilisées qu'après que tous les processus soient arrivés au rang r+1. En réalité, on a une interface de synchro locale qui est créée par utilisateur, l'utilisateur, envoie tous les mesg qu'il doit envoyer à cette interface. Cet interface est connectée aux interfaces des utilisateurs voisins. Chaque interface envoi les mesg de son utilisateur, et attend la réponses de TOUS ses voisins pour rendre les messages reçu à l'utilisateur.

Safe Mode : on envoi un message, et on attend ceux de nos voisins, mais même si on a tout reçu, on attend de recevoir des ack pour tous nos mesg envoyés pour passer en "safe" et passer au round suivant.

6 18 Mars 2013

6.1 Horloge

Qu'est ce qu'une horloge

Applications

Temps logiques faibles et horloges vectorielles Savoir qu'un événement est après un autre, rapport de cause-effet , ceux qui ne sont pas dépendants, sont parallèle.
Trois aspect : il simplifie la programmation dans des réseaux asynchrones, Réaliser un ordre total sur tous les événements, chacun des processus connait sont ordre par rapport aux autres.Objectif : associé à chaque event une valeur d'horloge, e.g global snapshot(photo sdu système).

localtime : On considère un système send/recv . quelles sont les propriété quelle doit conservés.

- deux events distincts doivent avoir des valeurs d'horloge distinctes, indépendance des events
- les events doivent respecter le temps
- ltime(send) <ltime(recv) pour ordre logique
- si on se donne une date, le nombre d'event avant la date doit etre fini. quand le système évolue, le temsp avance, il ne converge pas vers un date , il avance vers l'infini et au-delà

chaque processus va maintenir une valeur d'horloge unsigned int, init à 0, on indique ensuite comment on associe a chaque élément une valeur, à chaque fois qu'un processus execute un process local, il incrémente son horloge, quand il send un mesg, il increment sa valuer et il envoie dans le mesg sa valeur d'horloge, et quand le process recoit un mesg, il a sa propre valeur d'horloge, et reçoit celle du voisin qui lui a envoyer le message, il prend la plus grande des deux +1 , pour nouvelle horloge.

pour mettre en place un ordre total , on tiens comptes d'un couple(pid,valeur d'horloge), on le definit de manière lexico-graphique, d'abord le time, ensuite le pid.
evenemtn ltime, uniques
les evenements incrementent le temps, non-zero.

Welch's algorithm Question : et si il y a déjà des horloges ?, on veux utiliser ces horloges pour faire du temps logique.
Chaque process a sa propre horloge, quand il y a un veent local, on regarde sa propre horloge et c'ets celle la que l'on donne. Quand on reçoit un mesg , on reçoit un mesg avec un date, le process attend que son horloge locale soit plus grande que la date recue. Il recoit le mesg dans sa fifo et ne l'accepte que si son horloge est arrivée et à dépassée la date envoyée avec le message

Banque dans le système , chaque banquier possède les dates à laquelle il doit indiquer à un serveur combien de pognon il possède, quand arrive la date, il collecte l'argent qu'il a localement, et il a besoin de calculer l'argent qui à été envoyé, il v attendre des autres processus des mesg avec les dates à laquelle le pognon à été envoyé, il va calculer l'argent qu'il a dans ses compte et calculer

combien il a reçu à des dates antérieurs à la date d'envoi. le processus envoi un infinité de messages à d'autres, même avec 0 comme valeur de pognon.

Global snapshot généralisation du banking system, on va avoir une vision globale du système
simuler machine a états, il reçoit des événements, il les mets dans une file, et les executent dans l'ordre, avant d'exécuter un évent r, il faut être sur de ne pas recevoir un event n< r, pour cela, on attend encore une fois que l'horloge locale passe la date r.

6.2 horloge vectorielle

on a envie de def un ordre sur des évents, mais pas forcement total, on peut comparer des events sur le même processus, et ceux appartenant au même envoie de message, mais les autres on ne peux pas. au lieu d'avoir une valeur d'horloge par processus on garde un vecteur d'horloge , soit mon horloge plus ma vision locale des horloges des autres processus. quand il envoie un message, il incrémente sa valeur, quand il reçoit , il incrémente sa valeur a lui de la valeur maximum entre le max du vecteur reçu +1 et la sienne +1, puis met a jours ses valeurs connues des autres processus , avec a chaque fois le maximum. Avec ce système là , on obtient l'ordre obtenu avec l'ordre total et l'ordre à l'exécution.

7 25 Mars 2013

7.1 Systèmes à mémoire partagées

on ne veux pas que deux processus accèdent à une même ressource, algorithme de Peterson, Dijkstra, Lamport, Burns.

qu'est ce qu'un système asynchrone ?
Ce qui est important c'est qu'on a un ensemble de processus, et un ensemble de variables les deux forment un graphe, chaque processus accède a certaines variables, elles sont donc partagées. trois modes d'accès contraints , faible(R/W), fort(R/M/W) et Queues/stack/others. Avec Java concurrency, on a des structures pour gérer la concurrence.

Nous allons étudier deux problèmes, dans un environnement ou il n'y a pas de faute, mutual exclusion et ressources allocation.
On suppose que le système peut être dans n'importe quel état et il doit converger vers un état stable.

On va rester sur un description des programmes à base de graphe, à base d'in-

vocation et réponse, pour l'accès aux variables. On va décrire les algorithmes sous forme d'automates.

Mutual exclusion Partager une ressource entre plusieurs Users.
User_i à quatre régions, la zone normale, l'entrée en zone critique, la zone critique et la sortie de zone critique, notre objectif est de coder le comportement de l'algo en entrée et sortie de section critique , c'est a dire comment il prend la main et comment il la rend. Le système doit respecter le modèle, il doit être safe, donc respecter l'exclusion mutuelle, il doit aussi être équitable, un processus ne doit pas attendre "trop longtemps" pour accéder à la donnée souhaitée.

algo de Dijkstra 1965, principe , on a une variable turn qui peux etre lues par tout le monde et peut prendre des valeurs $1..n$, multiread, multiwrite. Chaque process à une variable flag qui vaux entre $0, 1, 2$ seul le processus i peut modifier son flag et seul les autres processus peuvent le lire. Quand le processus met son flag à 1 , il essaye de rentrer en session critique, il va essayer de mettre son id en variable turn. Si il n'est pa sturn , alors il regarde si le processus gagnant(celui qui a su imposer son pid) est actif, (si le flag du process winner est à 1) si je réussi à accéder à la section critique mon flag est à deux. donc avant d'écrire je regarde si je suis le seul à deux.

```
try
L: flag[i]= 1;
while(turn != i)
if (flag[turn] = 0)
turn = i
end try
flag[i] = 2;
for j != i do
if(flag[j] = 2) then
 goto L;
exit
flag[i] = 0;
```

On passe a Peterson, algo pour seulement 2 processus

```
try
L: flag[i]= 1;
turn = i
end try
wait flag[1-i]=0 v turn != i
```

On peut l'étendre à n processus

Sequence of competition : les processus doivent monter un escalier, celui le plus haut à accès à la données.

Peterson : Tournament algorithm On suppose 2^n processus

8 8 Avril 2013

8.1 Examen

Deux ou trois parties, certainement , un exercice sur algorithmes synchrones (chaque processus à un id, faire la somme de toutes les valeurs, que ce passe t-il si un proc n'as pas d'id, si le graph est unidir, bidir, que ce passe t-il dans le cas asynchrone(pensez a mettre un synchroniseur et utiliser l'algo d'avant) v, on peut aussi s'intéresser à connaître tout le graphe, on envoi a tout le monde ces voisins, pb efficacité, on transmet des messages de plus en plus énorme, dans les algos synchrones , on essaie de ne pas transmettre tous le graphe , mais des infos utiles, on peut par exemple arrêter quand on ne reçoit aucune info nouvelles)

Sûrement un exo sur les algo asynchrones, souvent comment traiter de manière asynchrone l'exo davant, soit synchroniser, soit algo spécifique (algo du calcul de distance minimum).
Prenons le cas de l'algo de peterson, pour l'election, cet algo marche-t-il de manière asynchrone ? cette question est difficile, mais reste en suspend (en fait non la réponse dans la section sur les modèles asynchrones. Algorithme de terminaison, avoiir un état global du graphe.

troisième partie sur les variables partagées.

droit aux documents Yep Distributed Algorithm , Nancy A.Lynch