Systèmes et Algorithmiques Répartis

Master 1
Informatique des Organisations - MIDO

Joyce EL HADDAD
elhaddad@lamsade.dauphine.fr

Chapitre 3 : Le Temps

- Introduction
- Environnement synchrone
- Horloge logique

Introduction

- Le temps est un concept fondamental des applications et des systèmes informatiques

- Dans un contexte réparti, la multiplicité des sites crée autant de référentiels possibles

- Les différentes notions associées au temps et leurs applications :
  - Le temps interne *(horloge, quatum)*
  - Le temps de l'environnement *(temps réel, synchrone)*
  - Le temps universel *(http://www.bipm.org/)*
  - Le temps logique *(ordre causal, horloge logique)*

Environnement synchrone

- Comparaison entre environnement asynchrone et synchrone
  - Afin d'illustrer le gain en complexité que fournit un environnement synchrone, nous allons étudier la construction d'un arbre de plus courts chemins sur un graphe de communication quelconque
  - La construction est initiée par un site qui deviendra la racine de l'arbre
  - Le chemin qui conduit de l'initiateur à un site quelconque doit être l'un des plus courts chemins possibles parmi ceux qui les relient

Environnement synchrone

- Arbre de plus courts chemins : environnement asynchrone
  - Principe : lorsqu'un site est rattaché à l'arbre, il propose à ses autres voisins de devenir ses fils. Il adjoint à sa proposition, la nouvelle distance à la racine qu'obtiendra le nœud sollicité
  - Dans un environnement asynchrone, la première proposition de rattachement n'est pas nécessairement la meilleure : à chaque rattachement de meilleure qualité, on réitère sa proposition aux voisins

Environnement synchrone

- Arbre de plus courts chemins : environnement asynchrone
  - Les variables d'un site S_i
    - $voisins_i$: sous-ensemble des sites voisins de S_i. Cette constante définit le graphe de communication
    - $père_i$: identité du site "père" dans l'arbre, initialisée pour l'initiateur à sa propre identité, et elle n'est pas initialisée pour les autres sites
    - $distance_i$: variable contenant la longueur du chemin allant du site jusqu'à la racine. Elle est initialisée à 0 pour l'initiateur et à l'infini ou à une constante supérieure ou égale au nombre de sites pour les autres sites

Environnement synchrone

- Arbre de plus courts chemins : environnement asynchrone
  - Algorithme d'un site S_i

```
Construire( )
    Début
        Pour tout j ∈ voisins_i \{père_i}
                Envoyer_à  ( j, ( cons, distance_i+1) );
        Fin pour
    Fin
```

```
Sur_réception_de ( j, (cons, distance) )
    Début
        Si (distance_i  >  distance)  Alors
            père_i = j;
            distance_i = distance;
            Construire( ) ;
        Fsi
    Fin
```

Environnement synchrone

- Arbre de plus courts chemins : environnement asynchrone
  - Complexité en nombre de messages dans le pire des cas
    - Bornons supérieurement Nbmess, le nombre de messages échangés (n étant le nombre de sites)
      - Le site initiateur appelle Construire exactement une fois; d'où un envoi d'au plus (n-1) messages,
      - A chaque appel à Construire, un site envoie au plus (n-2) messages (son père est exclu de l'envoi),
      - A chaque fois qu'un site appelle Construire sa distance décroît. La valeur maximale (différente de l'infini) que celle-ci peut prendre est (n-1), donc un site peut changer au plus (n-1) fois de valeurs Par conséquent, il appellera au plus (n-1) fois Construire.

$$Nbmess \leq (n-1) + (n-1)x(n-1)x(n-2) = \theta(n^3)$$

Environnement synchrone

- Arbre de plus courts chemins : environnement asynchrone
  - Complexité en nombre de messages dans le pire des cas
    - Vérifions qu'il existe une exécution dont le nombre de messages échangés est de l'ordre de n^3
      - Notre scénario est basé sur un graphe de communication totalement maillé. Dans une clique, le résultat de l'algorithme est nécessairement un arbre de hauteur 1.
      - Le site initiateur, S_1, envoie (n-1) messages de construction à ses voisins
      - Supposons qu'à l'exception du message au site S_2, tous les autres messages sont retardés. A la réception, S_2 prend comme père S_1 et envoie (n-2) messages à ses voisins.

Environnement synchrone

- Arbre de plus courts chemins : environnement asynchrone
  - Complexité en nombre de messages dans le pire des cas
    - Vérifions qu'il existe une exécution dont le nombre de messages échangés est de l'ordre de n^3
      - Supposons de même que tous les messages sont retardés excepté celui envoyé au site S_3
      - En itérant ce procédé, on construit un arbre (non stabilisé) qui n'est autre qu'un chemin qui parcourt les sites de S_1 à S_n
      - Intéressons-nous au site S_i : il est actuellement à une distance (i-1) de la racine. Supposons qu'il reçoive successivement les messages de construction i-2,i-3,...,1. Sa distance diminuera par pas de 1 jusqu'à la valeur finale 1

Environnement synchrone

- Arbre de plus courts chemins : environnement asynchrone
  - Complexité en nombre de messages dans le pire des cas
    - Vérifions qu'il existe une exécution dont le nombre de messages échangés est de l'ordre de n^3 :
      - Supposons un tel scénario pour chacun des sites S_i. L'arbre sera stabilisé.
      - L'initiateur envoie exactement (n-1) messages
      - Un site S_i différent de l'initiateur appelle exactement (i-1) fois Construire provoquant l'envoi de (i-1)x(n-2) messages

Nbmess = (n-1) + (n-2)+2x(n-2)+3x(n-2)+...+(n-2)x(n-2)+(n-1)x(n-2)

= (n-1) + (n-2).(1+2+3+...+(n-1))

= (n-1) + (n-2)x ½ x n x (n-1)

= θ (n^3)

Environnement synchrone

- Définissons un environnement synchrone pour une application répartie
  - Hypothèse n°1 : l'application de chaque site travaille sous forme de cycles numérotés. Chaque cycle est initié par le battement d'une pulsation
  - Hypothèse n°2 : la primitive exécutée lors du battement de la pulsation est notée Sur_pulsation(numéro). Son unique paramètre correspond au numéro de la pulsation courante. L'exécution de ce code est non bloquant *(pas d'appel à Attendre)* et doit se terminer avant le battement de la pulsation suivante

Environnement synchrone

- Définissons un environnement synchrone pour une application répartie
  - Hypothèse n°3 : durant l'exécution de Sur_pulsation, un site émet au plus un seul message vers chaque autre site
  - Hypothèse n°4 : un message émis lors de l'appel à Sur_pulsation(p) est reçu et traité par le site récepteur après l'exécution de Sur_pulsation(p) et avant le battement de la pulsation p+1
  - Hypothèse n°5 : il n'y a pas d'émission de message dans les primitives Sur_réception_de()

Environnement synchrone

- Définissons un environnement synchrone pour une application répartie
  - L'application travaille de manière synchrone :
    - Au début d'un cycle, tous les sites exécutent la primitive Sur_pulsation conduisant à des émissions de message. Ces messages sont ensuite reçus et traités par les différents sites. Puis une nouvelle pulsation est battue
    - Notons que si l'application se comporte comme indiqué, il est inutile que la même pulsation soit simultanément battue sur deux sites différents

Environnement synchrone

- Arbre de plus courts chemins : environnement synchrone
  - Principe : reprendre l'algorithme précédent en remarquant que, puisque l'environnement est synchrone, la première proposition de rattachement est nécessairement la meilleure :
    - Le site racine est rattaché à la pulsation 0, les sites à distance 1 le sont à la pulsation 1,…
    - Il suffit donc de comparer la distance du site à la valeur de la pulsation pour savoir quand proposer à ses voisins le rattachement à l'arbre

Environnement synchrone

- Arbre de plus courts chemins : environnement synchrone
  - Les variables d'un site S_i
    - $voisins_i$: sous-ensemble des sites voisins de S_i. Cette constante définit le graphe de communication.
    - $père_i$: identité du site "père" dans l'arbre, initialisée pour l'initiateur à sa propre identité, et elle n'est pas initialisée pour les autres sites.
    - $distance_i$ **:** variable contenant la longueur du chemin allant du site jusqu'à la racine. Initialisée à 0 pour l'initiateur et à l'infini ou à une constante supérieure ou égale au nombre de sites pour les autres sites.
    - $pulsation_i$: indice de la pulsation courante, initialisée à 0.

Environnement synchrone

- Arbre de plus courts chemins : environnement synchrone
  - Algorithme d'un site S_i

Sur_ pulsation(pulsation$_i$)

Début
  Si (pulsation$_i$ = = distance$_i$) Alors
    Pour tout j ∈ voisins$_i$\{père$_i$}
      Envoyer_à (j, (cons, distance$_i$+1));
    Finpour
  Fsi
Fin

Sur_réception_de (j, (cons, distance))

Début
  Si (distance$_i$ = = +∞) Alors
    père$_i$ = j;
    distance$_i$ = distance;
  Fsi
Fin

Environnement synchrone

- Arbre de plus courts chemins : environnement synchrone
  - Complexité en nombre de messages dans le pire des cas
    - Le calcul du nombre de messages échangés est simple : un site n'envoie qu'une proposition de construction à tous ses voisins excepté son père. Donc le pire des cas est atteint sur une clique où l'initiateur envoie (n-1) messages et chaque autre site envoie (n-2) messages
    - Ce qui nous donne :

      $Nbmess = (n-1) + (n-1)x(n-2) = (n-1)^2$ messages

    - Un gain d'un ordre de grandeur par rapport à l'algorithme asynchrone.

Horloge logique

- Considérons le cas d'un serveur traitant une requête à la fois et mettant en attente les requêtes arrivées en cours de traitement

- Choix de la prochaine requête à traiter
  - Choix 1 : traiter les requêtes dans l'ordre de réception. L'inconvénient est de favoriser les sites les plus "proches" du serveur
  - Choix 2 : traiter les requêtes selon l'ordre d'émission. L'inconvénient est l'écart éventuel entre les horloges physiques

Horloge logique

- Choix de la prochaine requête à traiter
  - Choix 3 : traiter les requêtes de la façon suivante
    - si la requête r_1 précède causalement r_2, alors traiter d'abord r_1
    - sinon si la requête r_2 précède causalement r_1, alors traiter d'abord r_2
    - sinon les traiter dans un ordre arbitraire (*l'ordre causal est un ordre partiel)*
    - Contrainte : transformation du réseau en un réseau FIFO
  - Définir un mécanisme qui permette d'adopter une telle politique sans contraindre le réseau à être FIFO
  - Les horloges logiques [Lamport 1978]

Horloge logique

- Les horloges logiques [Lamport 1978]
  - Chaque site gère une horloge logique, un compteur croissant initialisé à 0.
  - Chaque message est estampillé avec la valeur courante de l'horloge logique et on désire que :

$$m <_c m' \Rightarrow h_m < h_{m'} \quad (h_m \text{ étant l'estampille de } m)$$

  - La réciproque n'est pas nécessairement vérifiée puisque l'ordre causal est partiel et l'ordre sur les entiers est total
  - Au vu de la transitivité de l'ordre, il suffit de garantir la propriété précédente pour l'ordre immédiat

Horloge logique

- Les horloges logiques [Lamport 1978]
  - Ce qui conduit aux deux règles de mise à jour suivantes :
    - Règle d'émission : après chaque émission de messages, $h_i = h_i+1$
    - Règle de réception : à la réception d'un message (m,h_m), $h_i = \max(h_i, h_m+1)$
  - Pour appliquer le principe des horloges logiques au serveur traitant une requête à la fois, il faut disposer d'un ordre total entre les messages
    - Comparaison des paires *(horloge, identité du site)*

$$(h,i) < (h',i') \Leftrightarrow (h < h') \text{ ou } (h = h' \text{ et } i < i')$$

Références

- P. Ramanathan, K.G. Shin, R.W. Butler, "Fault-tolerant Clock synchronization in Distributed Systems", IEEE Transactions on Computers vol C-39, pp. 514-524, 1990
- S. Toueg, "An all-pairs shortest-path distributed algorithm" Technical Report RC 8327, IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, 1980
- B. Awerbuch, "Complexity of network synchronization", JACM 32, pp. 804-823, 1985
- L. Lamport, "Time, clocks, and the ordering of events in a distributed system", Communications of the ACM 21, pp. 558-564, 1978
- BIPM, "Bureau International des Poids et Mesures" http://www.bipm.org/