Synchroniseurs

Master 2 Informatique - UFR S.A.T

Pr. Ousmane THIARE

ousmane.thiare@ugb.edu.sn
http://www.ousmanethiare.com/

16 avril 2020

Chapitre 5 : Synchroniseurs

Chapitre 5 : Synchroniseurs

Synchroniseurs
Introduction

La conception d'un algorithme distribué pour un système asynchrone est bien plus difficile que pour un système synchrone. L'analyse de sa validité doit prendre en compte tous les scenarii possibles, alors qu'un synchrone il n'y a, le plus souvent, qu'un seul scenario possibles. Pour s'en convaincre, il suffit de considérer le problème de la construire d'un arbre BFS (vu au chapitre 4 précédent).

Synchroniseurs
Introduction

L'idée, introduite en 1982 par Gallager [Gal82] puis généralisée par Awerbuch en 1985 [Awe85], est de transformer automatiquement un algorithme synchrone pour qu'il puisse fonctionner sur un système asynchrone. Le synchroniseur simule donc des envoies de messages synchrones, au top d'horloge t, sur un système asynchrone, sans horloge. Ceci se fait en payant des surcoûts en nombre de messages dits de synchronisation et sur le temps. Il faut également prendre en compte de temps de construction éventuelle du synchroniseur. Plusieurs synchroniseurs sont possibles, chaque fois avec des compromis différents sur ces surcouts.

Soit S un algorithme synchrone écrit pour un système synchrone et σ un synchroniseur. On souhaite construire un nouvel algorithme $A = \sigma(S)$, obtenu en combinant σ et S, qui puisse être exécuté sur un système asynchrone. Pour que la simulation soit correcte, il faut que l'exécution de A sur un système asynchrone soit « similaire » à l'exécution de S sur un système synchrone (on va donner plus tard un sens plus précis à « exécutions similaires »).

Synchroniseurs
Méthodologie

Dans chacun des sommets, l'algorithme A a deux composantes :

- La composante originale représentant l'algorithme S que l'on veut simuler, c'est-à-dire toutes les variables locales, les routines et les types de message utilisés par S.
- La composante de synchronisation, c'est-à-dire le synchroniseur σ, qui peut comprendre ces propres variables locales, routines et types de message utilisés.

C'est deux composantes doivent être bien distinctes puisque pendant l'exécution de A, par exemple, des messages des deux composantes vont être amener à coexister. Les messages « ack » originaux de S ne doivent pas être confondus avec ceux ajoutés par le synchroniseur.

L'idée de base est que chaque processeur u possède un générateur de pulse (=top horloge), Pulse(u)=0, 1, 2, . . . variable qui intuitivement simule les tops horloges d'une horloge globale. Dans l'algorithme A on va exécuter la phase p de l'algorithme S (grâce à la composante originale) uniquement lorsque Pulse(u) = p.

Malheureusement, après avoir exécuté la phase p de S, u ne peut pas se contenter d'incrémenter Pulse(u) et de recommencer avec la phase p+1. Il faut attendre ses voisins puisque sinon un voisin v pourrait recevoir des messages de u de la phase p+1 alors que v est toujours dans la phase p. La situation peut être encore moins favorable, avec un décalage pire encore. Il pourrait par exemple se produire qu'aux 5 phases suivantes, p+1,...,p+6 de S, il ne soit pas prévu que u envoye et reçoive des messages. Si bien que u pourrait effectuer tous ses calculs locaux de la phase p+1 à la phase p+6 et passer directement Pulse(u) = p+7. Si à la phase phase p+7 il envoie un message à v, v qui est dans la phase p aurait légitimement l'impression de recevoir des messages du futur.

Synchroniseurs
Méthodologie

Pour résoudre le problème, on pourrait penser à première vue qu'il suffit de marquer chaque message par le numéro de phase de l'émetteur, et ainsi retarder les messages venant du futur. La question qui se pose est alors de savoir combien de temps il faut les retarder. Malheureusement, un processeur v ne peut pas savoir a priori combien de messages de la phase p il est censé recevoir. Sans cette information v pourrait attendre de recevoir un message de la phase p, alors dans S à la phase p il est peut être prévu qu'aucun voisin ne communique avec v.

En asynchrone, lorsqu'un sommet attend de recevoir un message il attend potentiellement pour toujours. La réception étant bloquante, il n'y a pas de *time out* car pas d'horloge. En synchrone, les réceptions se débloquent à la fin de chaque phase.

Definition (Compatibilité de pulse)

Si un processeur u envoie à un voisin v un message M de la composante originale de la phase p lorsque Pulse(u) = p, alors M est reçu en v lorsque Pulse(v) = p.

Bien sûr, les exécutions de A et de S ne peuvent pas être identiques, car d'une part sur un système asynchrone, les exécutions ne sont pas déterministes (deux exécutions sur les mêmes entrées ne donne pas forcément le même résultat !), et que d'autre part les messages de σ dans l'exécution de A n'existent évidemment pas dans l'exécution de S.
On donne maintenant un sens plus précis à la notion de similarité entre les exécutions de S et de A.

Definition (Exécutions similaires)

Les exécutions de S sur un système synchrone et de $A = \sigma(S)$ sur un système asynchrone sont similaires si pour tout graphe G (l'instance) sur lesquels s'exécutent S et A, toute variable X, toute phase p et toute arête uv de G :

- Les messages de la composante originale envoyés (ou reçus) par u sur l'arête uv lorsque Pulse(u)=p sont les mêmes que ceux de S à la phase p envoyés (ou reçus) par u sur l'arête uv.

Definition (Exécutions similaires)

- La valeur de X en u au début de l'exécution de A lorsque Pulse(u)=p est la même que celle de X en u dans l'exécution du début de la phase p de S. La valeur de X en u lorsque A est terminé est la même que celle de X en u lorsque S est terminé.

Pour comprendre la définition précédente supposons que S=SpanTree, qui calcule un arbre couvrant BFS (en largeur d'abord) sur un système synchrone. Si $A = \sigma(S)$ et S ont des exécutions similaires, alors en particulier les variables Père(u) et Fils(u) seront identiques à la fin des exécutions de A et de S. Elles définiront alors les mêmes arbres. Il en résulte que l'algorithme A calcule un arbre couvrant BFS sur un système asynchrone.

Proposition

Si le synchronisateur σ garantit la compatibilité de pulse pour tout algorithme synchrone S, toute phase, tout graphe G et tout sommet, alors les exécutions de S et de $\sigma(S)$ sur G sont similaires.

La question principale est donc : « Comment implémenter la compatibilité de pulse ? c'est-à-dire quand est-ce que u doit incrémenter Pulse(u) ? »

Pour cela nous avons encore besoin de deux notions. Le sommet u est Safe pour la phase p, et on note Safe(u, p) le prédicat correspondant, si tous les messages originaux de la phase p envoyés de u à ses voisins ont été reçus.
Le sommet u est Ready pour la phase p+1, et on note Ready(u, p+1) le prédicat correspondant, si tous ses voisins sont Safe pour la phase p.
La proposition suivante nous indique lorsque Pulse(u) peut être incrémenté.

Proposition

Si p=Pulse(u) et si Safe(u, p) et Ready(u, p + 1) sont vraies, alors la compabilité de pulse est garantie.

Notons qu'une seule des deux conditions ne suffit pas à garantir la compatibilité de pulse.
Preuve : Exercice.

Grâce à la proposition précédente il suffit d'implémenter les propriétés Safe(u,p) et Ready(u, p+1). On a donc deux étapes à réaliser.

- **Étape Safe.** À chaque message de la composante originale reçu on renvoie un « ack ». Ainsi l'émetteur u peut savoir lorsque tous ses messages de la phase p=Pulse(u) ont été reçus.
- **Étape Ready.** Lorsqu'un sommet v devient Safe pour la phase p=Pulse(v) il diffuse cette information à ses voisins via un message « ready », mais pas forcément sur l'arête uv. Il indique ainsi à son voisin u la possibilité d'incrémenter Pulse(u).

L'étape Safe n'est pas très couteuse. Le temps et le nombre de messages sont au plus doublés. L'étape Ready est généralement plus coûteuse puisque u peut, à une phase p donnée, recevoir potentiellement beaucoup plus de messages « ready » qu'il n'a émis de messages originaux. Par exemple, u pourrait ne pas avoir émis de messages originaux à la phase p, et pourtant u devra recevoir un message « ready » de ces voisins pour pouvoir incrémenter Pulse(u) (cf. proposition précédente).

Notons cependant qu'il n'y a aucune raison pour que v envoie son message « ready » à u directement sur l'arête uv. Certes c'est la solution la plus rapide de communiquer l'information. Mais si le temps n'est pas le critère principal on peut faire différemment et économiser des messages. Ce qui importe à u c'est de savoir quand tous ses voisins sont Safe. Un autre sommet pourrait jouer le rôle de collecteur de plusieurs messages « ready ». Et, dans un deuxième temps, ce sommet pourrait informer u par un message différent, disons « pulse », que tous ses voisins sont Safe pour la phase p. Globalement, à un moment donnée, il pourrait avoir m messages « ready » émis si tous les sommets sont Safe en même temps, alors qu'il y a que seulement n sommets qui sont intéressés de recevoir un message « pulse ». Il serait donc envisageable de n'utiliser que n messages « pulse ».

Les synchroniseurs diffèrent sur l'étape Ready, l'étape Safe étant commune à tous les synchroniseurs que l'on va présenter.
On note respectivement $T_{init}(\sigma)$ et $M_{init}(\sigma)$ le temps et le nombre de messages pour initialiser la composante de synchronisation. Notons toute de suite que l'initialisation de σ est toujours au moins aussi couteuse qu'une diffusion car il faut au minimum que chaque sommet u exécute Pulse(u) :=0.

On note $M_{pulse}(\sigma)$ le nombre maximum de messages de la composante de synchronisation générés dans tout le graphe entre deux changements de pulse consécutifs, hormis les messages « ack » de l'étape Safe. Si on ne compte pas ces messages dans $M_{pulse}(\sigma)$ c'est parce qu'ils correspondent toujours au nombre de messages envoyés pendant deux changements de pulse et que ce nombre peut être très variable d'une phase à une autre. Il dépend de S, pas vraiment de σ.

Enfin, on note $T_{pulse}(\sigma)$ est la durée (en unité de temps) entre le moment où le dernier sommet passe à la phase p et le moment où le dernier sommet passe à la phase p+1 (maximisé sur toutes les phases p). Attention ! Cela ne veut pas dire que pour un sommet u, la durée entre le moment où Pulse(u)=p et où Pulse(u)=p+1 est borné par $T_{pulse}(\sigma)$. Cela pourrait être plus, car il est possible que le pulse de u soit passer de p-1 à p un peu avant le dernier.

Proposition

Soit σ un synchroniseur implémentant les étapes Safe et Ready, et S un algorithme synchrone S sur G. Alors,

- $Temps(\sigma(S), G) \leq T_{init}(\sigma) + T_{pulse}(\sigma) * Temps(S, G)$.

Preuve : A faire. Le facteur 2 est pour les messages « ack » du synchroniseur.

Proposition

Soit σ un synchroniseur implémentant les étapes Safe et Ready, et S un algorithme synchrone S sur G. Alors,

- $Temps(\sigma(S), G) \leq T_{init}(\sigma) + T_{pulse}(\sigma) * Temps(S, G)$.
- $Message(\sigma(S), G) \leq M_{init}(\sigma) + 2 * Message(S, G) + M_{pulse}(\sigma) * Temps(S, G)$.

Preuve : A faire. Le facteur 2 est pour les messages « ack » du synchroniseur.

Soit σ un synchroniseur implémentant les étapes Safe et Ready, et S un algorithme synchrone S sur G. Alors,
Synchroniseur σ
Le synchroniseur ? est efficace pour les graphes avec $m = O(n)$, comme les graphes de degré bornés, les graphes planaires, etc.
Synchroniseur β
...

Synchroniseur γ : Les complexité en Temps et Message sont chacune multipliée par un facteur $O(log^3 n)$. D'où le résultat du calcul d'un BFS, car en synchrone il est possible de faire $Temps = O(D)$ et $Message = O(m)$. Comme il n'existe pas de borne sur le temps mis par un message pour traverser un lien (et surtout les processeurs n'ont pas d'horloge capable de mesurer ce temps de manière absolu), il n'est pas possible pour un processeur d'attendre suffisamment longtemps.

On peut se demander s'il n'existe pas de meilleur synchroniseur. En fait, on a :

Proposition

Pour chaque entier k>0, il existe des graphes G_k à n sommets où tout synchroniseur σ tel que $Temps_{pulse}(\sigma) < k - 1$ doit vérifier $Message_{pulse}(\sigma) > \frac{1}{4} n^{1+\frac{1}{k}}$.

Preuve :

La preuve est basé sur le fait que pour tout entier k il existe un graphe G_k avec n sommets et $\frac{1}{4} n^{1+\frac{1}{k}}$ arêtes où tout cycle est de longueur au moins k.

La proposition précédente laisse l'espoir de trouver un synchroniseur σ avec $Temps_{pulse}(\sigma) = O(logn/loglogn)$ et $Message_{pulse}(\sigma) = O(logn/loglogn)$ en choisissant k=log n / log log n. Même s'il existe, encore faudrait-il savoir le construire efficacement.
On verra comment construire rapidement ces graphes à la base de la construction de ces synchroniseurs.