Chloé DESDOUITS
Guillerme DUVILLIE
Swan ROCHER

M1 Informatique MOCA

TP d'algorithmes distribués

2012

UM2

Table des matières

1 Introduction **2**

2 Algorithme d'exclusion mutuelle de Naimi-Trehel **3**

2.1 Description 3

2.2 Exemple 3

2.3 Propriétés 5

2.4 Complexité 6

3 Extension tolérante aux pannes **7**

3.1 Description 7

4 Implémentation **8**

5 Tests et résulats **9**

Chapitre 1

Introduction

Afin de résoudre des problèmes complexes sur des données de grandes tailles, il existe plusieurs possibilités : utiliser un algorithme avec une bonne complexité, augmenter la puissance de l'ordinateur utilisé ou utiliser plusieurs ordinateurs interconnectés. Lorsque les deux premières possibilités ont été poussées à leur maximum, il est nécessaire d'utiliser des algorithmes dits distribués afin de répartir les calculs sur plusieurs machines.

Les algorithmes distribués sont divisés en plusieurs catégories. Il existe tout d'abord des algorithmes d'élection qui permettent de donner la priorité temporaire à un site par rapport aux autres. Il y a également des algorithmes d'exclusion mutuelle qui permettent à tous les sites de travailler sur la même ressource (section critique) et qui garantissent le fait que deux sites ne peuvent pas accéder en même temps à cette ressource. Enfin les algorithmes de terminaison permettent de détecter la fin du calcul distribué pour tous les sites et ainsi de terminer l'application.

Nous allons nous intéresser aux algorithmes distribués d'exclusion mutuelle. Ceux-ci sont jugés sur les critères suivants :

- la vivacité : le fait qu'un site qui demande à entrer en section critique puisse y entrer au bout d'un temps fini,
- la sureté : la capacité de l'algorithme à faire en sorte qu'un seul site à la fois soit en section critique,
- le respect de l'ordre des demandes,
- la tolérance aux pannes,
- la complexité en nombre de messages échangés.

Nous allons tout d'abord étudier l'algorithme de Naïmi-Trehel [1] qui fait partie de la catégorie des algorithmes distribués d'exclusion mutuelle. Nous allons ensuite nous intéresser à une extension tolérante aux pannes de cet algorithme [3]. Nous expliquerons alors les choix effectués lors de l'implémentation de ces deux algorithmes. Enfin, nous exposerons les résultats de nos phases de test.

Chapitre 2

Algorithme d'exclusion mutuelle de Naimi-Trehel

2.1 Description

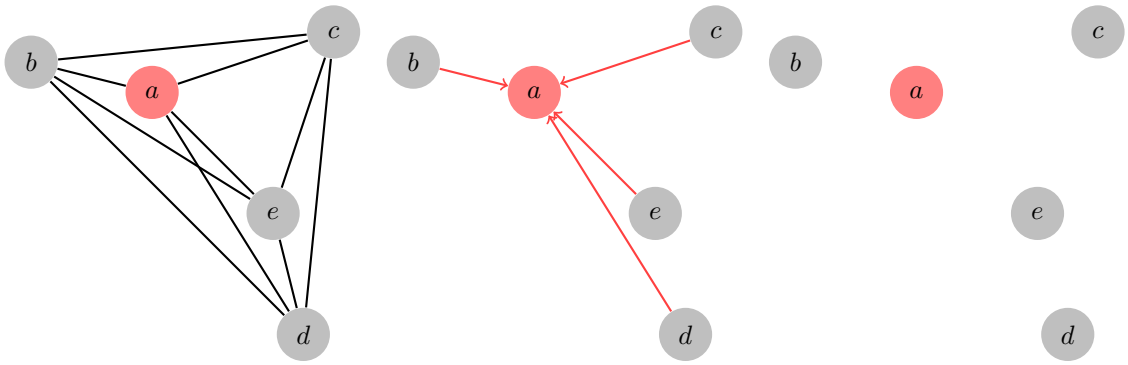
L'algorithme de Naimi-Trehel [1] est basé sur le fait de conserver deux structures de données distribuées et un jeton. Le jeton est présent sur un des sites à la fois et matérialise la permission d'entrer en section critique. Les structures de données distribuées sont un arbre logique dynamique et une file d'attente.

La file d'attente stocke les sites qui ont demandé à entrer en section critique. La file d'attente étant distribuée, chaque site stocke uniquement son suivant (s'il y en a un) dans la file d'attente. La tête de la file est le site actuellement en section critique et la queue de la file est le dernier site à avoir demandé à y entrer. Quand un site demande à entrer en section critique, il est ajouté à la fin de la file. Quand un site quitte la section critique, il envoie le jeton à son suivant dans la file (s'il y en a un).

La seconde structure de données est l'arbre logique dynamique. Cet arbre a pour but d'indiquer aux sites lequel d'entre eux est le dernier dans la file d'attente. La structure étant distribuée, chaque site stocke uniquement son père dans l'arbre. Lorsqu'un site veut entrer en section critique, il transmet la requête à son père qui (s'il n'est pas racine) transmet la requête à son père jusqu'à ce que la requête atteigne la racine de l'arbre. Le site racine dans l'arbre étant également le dernier dans la file d'attente, il fixe son suivant au site qui a émis la requête. Puis tous les sites qui ont transmis la requête fixent leur père au site qui a émis la requête (désormais le dernier dans la file d'attente).

2.2 Exemple

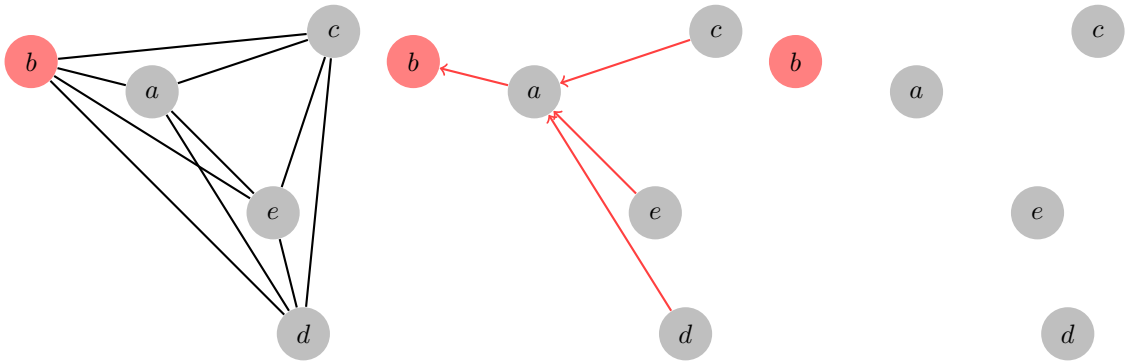
Dans cet exemple, initialement, le nœud a possède le jeton (figure 2.1).



(a) Réseau physique (b) Arbre logique (c) File d'attente

FIGURE 2.1 –

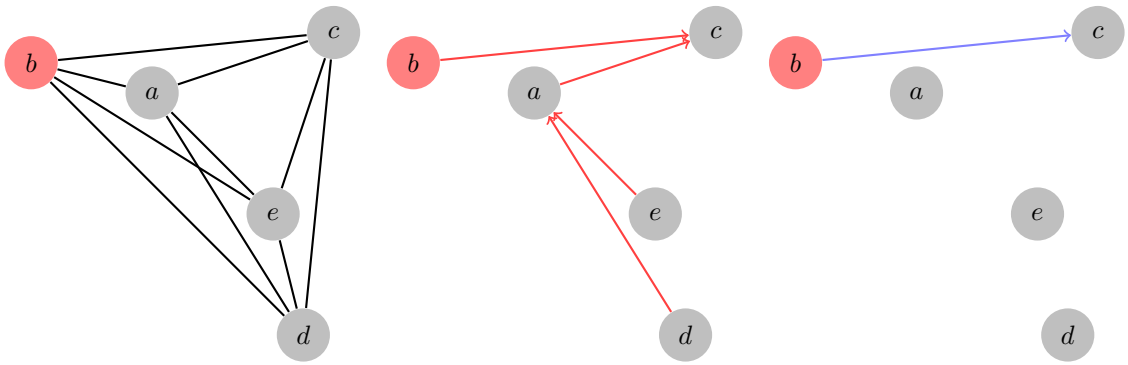
Puis le nœud b fait une requête pour obtenir la section critique. Il envoie cette requête au nœud a et deviens la nouvelle racine. Le site a reçoit la requête venant de b et lui envoie le jeton. Le père de a dans l'arbre devient b. Le site b reçoit le jeton et entre en section critique. (figure 2.2)



(a) Réseau physique (b) Arbre logique (c) File d'attente

FIGURE 2.2 –

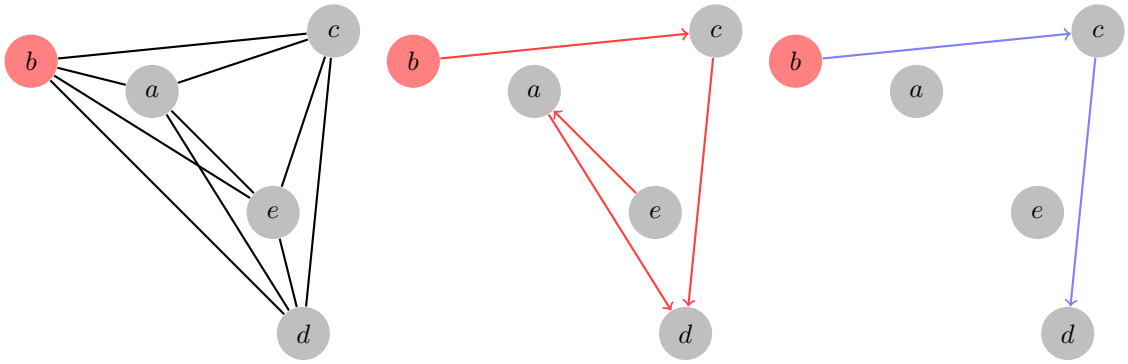
Le site c demande à entrer en section critique. Il envoie une requête au nœud a qui la transmet au nœud b. Le nœud b reçoit la requête et considère c comme son suivant dans la file d'attente. a et b fixent c comme leur père dans l'arbre. (figure 2.3)



(a) Réseau physique (b) Arbre logique (c) File d'attente

FIGURE 2.3 –

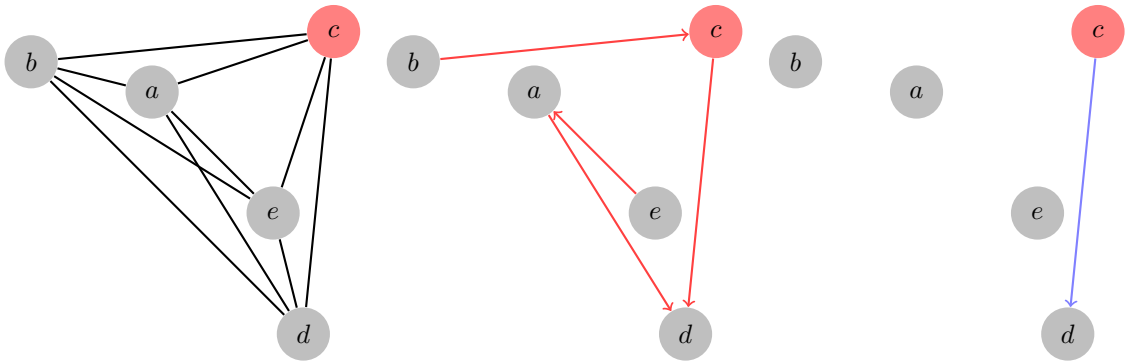
Le site d demande à entrer en section critique. Il envoie une requête au site a qui transmet au site c. Le site c reçoit la requête relayée par le site a, son suivant dans la file devient d, son père également. Le père de a devient d. (figure 2.5)



(a) Réseau physique (b) Arbre logique (c) File d'attente

FIGURE 2.4 –

Le site b relâche la section critique et envoie le jeton à son suivant : c.



(a) Réseau physique (b) Arbre logique (c) File d'attente

FIGURE 2.5 –

2.3 Propriétés

La première des propriétés que respecte cet algorithme est la sureté. En effet, il est nécessaire qu'au plus un processus à la fois entre en section critique. Ceci est garanti par la présence du jeton. Ce jeton est initialement présent chez un seul site. Lors du déroulement de l'algorithme, il y a deux possibilités pour qu'un autre site reçoive le jeton :

- soit le site qui possède le jeton sort de la section critique avec un suivant non-nul. Il lui envoie alors le jeton et le perd pour lui-même.
- soit le site qui possède le jeton reçoit une demande et n'est pas lui-même en section critique. Il envoie alors le jeton et le perd pour lui-même.

Dans les deux cas, l'unicité du jeton est préservée et l'exclusion mutuelle est garantie.

La seconde propriété est la vivacité c'est à dire le fait que tout site demandant à entrer en section critique le puisse dans un temps fini. Cette propriété est respectée grâce à l'arbre

logique et à la file d'attente. En effet, chaque site voulant entrer en section critique fait la requête à son père dans l'arbre. Le fait que la structure d'arbre distribué soit toujours valide garantie que les requêtes parviennent à la racine de l'arbre. Or la racine de l'arbre étant le dernier nœud de la file d'attente, le site ayant fait la requête se retrouve le dernier dans la file d'attente. Par conséquent, quand son prédécesseur dans la file sortira de la section critique, il pourra y entrer. Dans l'hypothèse où un site ne reste pas éternellement en section critique, la propriété de vivacité est respectée par l'algorithme de Naimi-Trehel.

Cependant cet algorithme n'étant pas parfait, il ne respecte pas l'ordre des demandes et n'est pas nativement tolérant aux pannes (bien qu'une extension de tolérance aux pannes [2] puisse être ajoutée).

2.4 Complexité

Soit M_n le nombre moyen de messages nécessaires pour que la demande d'un site atteigne la racine dans un réseau à n nœuds. Naimi et Trehel prouvent dans leur article [1] que $M_n \simeq H_{n-1} \simeq log(n-1) + 0.577$. C'est à dire que la complexité en nombre de message pour une requête est $\bigcirc(log(n))$.

Chapitre 3

Extension tolérante aux pannes

Dans leur article [1], Naimi et Trehel propose d'utiliser une extension [2] afin de rendre leur algorithme tolérant aux pannes. L'extension qu'ils proposent admet (en cas de panne) une complexité en nombre de messages de 4 broadcasts.

De plus, l'extension nécessite d'avoir une borne supérieure sur le temps de réception d'un message et sur le temps d'exécution de la section critique. Enfin, la file d'attente doit être entièrement reconstruite en cas de panne.

L'extension de tolérance aux pannes que nous allons présenter [3] tente d'améliorer ces performances.

3.1 Description

Le principe de cet algorithme est de reconstituer la file d'attente à partir des portions restantes. Cependant, si la reconstitution n'est pas possible, une nouvelle file d'attente ainsi qu'un nouvel arbre logique seront reconstruits.

Dans l'algorithme de Naimi-Trehel, chaque site connait sont successeur dans la file d'attente mais pas son prédécesseur.

Chapitre 4

Implémentation

Chapitre 5

Tests et résulats

Bibliographie

[1] M. Naimi, M. Trehel, and A. Arnold. A log (n) distributed mutual exclusion algorithm based on path reversal. *Journal of Parallel and Distributed Computing*, 34(1) :1–13, 1996.

[2] Mohamed Naimi and Michel Trehel. How to detect a failure and regenerate the token in the log (n) distributed algorithm for mutual exclusion. *Distributed Algorithms*, pages 155–166, 1988.

[3] J. Sopena, L. Arantes, M. Bertier, and P. Sens. A fault-tolerant token-based mutual exclusion algorithm using a dynamic tree. *Euro-Par 2005 Parallel Processing*, pages 644–644, 2005.

Réalisé avec LaTeX