Communications Efficaces et Auto-Stabilisation

Stéphane Devismes, Toshimitsu Masuzawa, Sébastien Tixeuil

Communications Efficaces et Auto-Stabilisation†

Stéphane Devismes[1], Toshimitsu Masuzawa[2] et Sébastien Tixeuil[3]

[1]*VERIMAG UMR 5104, Université Joseph Fourier, Grenoble I*
[2]*Graduate School of Information Science and Technology, Osaka University, Japan*
[3]*LIP6 UMR 7606, Université Pierre et Marie Curie, Paris VI*

Dans cet article, nous introduisons le concept d'efficacité des communications dans les algorithmes auto-stabilisants. Nous définissons plusieurs mesures permettant de juger de cette efficacité. Nous en étudions les limites. Enfin, nous illustrons notre approche en proposant deux algorithmes auto-stabilisants ayant de bonnes propriétés d'efficacité des communications.

Keywords: systèmes distribués, auto-stabilisation, algorithme silencieux, communications efficaces

1 Introduction

L'*auto-stabilisation* [Dij74] est un paradigme général permettant de concevoir des algorithmes distribués tolérants aux fautes transitoires. Une faute transitoire est une panne non-définitive qui altère le contenu des composant du réseau (processus ou canal de communication) où elle se produit. En supposant que les fautes transitoires n'altèrent pas le code de l'algorithme, un algorithme auto-stabilisant retrouve de lui-même et en un temps fini un comportement normal après que les fautes transitoires ont cessé. Le critère principal permettant de juger de l'efficacité d'un algorithme auto-stabilisant est le *temps de stabilisation*, c'est-à-dire, le temps maximal nécessaire à l'algorithme pour retrouver un comportement correct après que les fautes ont cessé. Dans cet article, nous soutenons que ce critère n'est pas suffisant. En effet, il ne permet pas, par exemple, d'évaluer le *surcoût* occasionné par la propriété d'auto-stabilisation : en l'absence de faute, un algorithme auto-stabilisant a un coût généralement supérieur à son équivalent non auto-stabilisant. Ce surcoût est observable à la fois au niveau du temps d'exécution, de l'occupation mémoire, du nombre et de la taille des messages échangés. Par exemple, la plupart des algorithmes auto-stabilisants nécessitent que tous les processus continuent à envoyer régulièrement des informations à tous leurs voisins (*heartbeat*) après que le système a stabilisé.

Dans cet article, nous proposons d'étudier comment réduire le surcoût en communication des algorithmes auto-stabilisants. Afin de détecter d'éventuelles nouvelles fautes, tout algorithme auto-stabilisant nécessite que les processus continuent à s'échanger des informations après que le système a convergé. Nous proposons ici de réduire le nombre de voisins avec lesquels un processus continue à échanger des informations une fois que l'algorithme auto-stabilisant a retrouvé un comportement normal, on parlera alors d'*efficacité des communications*. Une notion d'efficacité en communication a déjà été introduite pour des algorithmes tolérants aux pannes définitives : dans [LFA00], les auteurs proposent des algorithmes qui garantissent qu'un nombre global de liens de communication finissent par ne plus jamais être utilisés. Ici, notre approche est différente. Nous souhaitons réduire le nombre de voisins avec lesquels chaque processus communique. Ainsi, (1) le nombre de liens de communication globalement utilisé est réduit et (2) la charge des communication est *répartie* entre les différents processus. Notre notion d'efficacité des communications est particulièrement pertinente pour des algorithmes silencieux. De tels algorithmes convergent en un temps fini vers une configuration à partir de laquelle les informations échangées entre voisins sont toujours les

†Les résultats présentés ici sont issus de [DMT09]. Ce travail est soutenu par les projets ANR SHAMAN et EGIDE SAKURA.

mêmes. De tels algorithmes permettent de structurer un réseau, par exemple avec arbre couvrant ou un ensemble dominant.

Dans la section suivante, nous présentons brièvement le modèle considéré. Dans la section 3, nous définissons des mesures permettant de juger de l'efficacité des communications. Dans la même section, nous en étudions les limites. Dans la section 4, nous illustrons notre approche avec deux algorithmes auto-stabilisants ayant de bonnes propriétés d'efficacité des communications. Nous concluons dans la section 5.

2 Modèle

Nous considérons des réseaux bidirectionnels quelconques de n processus où chaque processus peut communiquer directement avec un ensemble restreint d'autres processus appelés *voisins*. Les processus communiquent par le biais de *variables de communication* localement partagées : chaque processus dispose de variables de communication qui peuvent être lues par lui-même et par ses voisins mais où il est le seul à pouvoir écrire. Les processus disposent en outre de *variables internes* qu'ils sont les seuls à pouvoir lire ou écrire. L'*état* d'un processus correspond aux contenus de ses variables internes et de ses variables de communication. Nous appelons *état de communication* l'état d'un processus restreint aux variables de communication. L'ensemble des états des processus à un instant donné forme la *configuration* du système. La *configuration de communication* correspond à la restriction aux états de communication de la configuration du système.

Les exécutions sont des suites d'étapes atomiques : à chaque étape, un sous-ensemble des processus est activé. En une étape atomique, chaque processus activé lit ses propres variables (internes et de communication) ainsi que les variables de communication de ses voisins, puis éventuellement modifie son état. Les processus sont activés de manière asynchrone mais équitable. Pour mesurer le temps de stabilisation, nous utilisons la notion de *ronde*, qui permet de mesurer le temps d'exécution rapporté au processus le plus lent. Ainsi, la première ronde d'une exécution termine dès lors que tous les processus ont été activés au moins une fois, la seconde ronde commence, etc.

3 Mesures et Limites de l'Efficacité des Communications

Nous proposons une nouvelle mesure, appelée *k-efficacité*, permettant de distinguer les algorithmes où les processus activés lisent les variables de communication de tout ou partie de leurs voisins : un algorithme est dit *k-efficace* si à chaque étape de calcul, les processus activés lisent les variables de communication d'au plus k de leurs voisins. La k-efficacité permet de réduire le nombre et la taille des informations échangées à chaque étape. Aussi pour comparer le gain des algorithmes k-efficaces aux autres algorithmes, nous avons introduit la notion de *volume de communication* qui correspond au volume maximum d'informations lu (c'est-à-dire la somme des tailles des variables de communication lu) par chaque processus à chaque étape de calcul. Ce critère est plus pertinent que la notion classique d'encombrement mémoire des algorithmes auto-stabilisants qui correspond à la somme des tailles des variables internes plus les tailles des variables de communication.

Un autre critère intéressant concerne la stabilité des communications, c'est-à-dire, le fait qu'un processus finisse par ne plus lire que les variables de communication d'un sous-ensemble particulier de ses voisins. Un algorithme est dit *k-stable* si durant toutes ses exécutions, tout processus lit les variables de communication d'au plus k voisins différents au moins une fois. Une telle propriété serait très intéressante pour un algorithme auto-stabilisant. Cependant, nous avons démontré dans [DMT09] que la plupart des problèmes n'admettent pas de solution auto-stabilisante *k-stable* où k est inférieur au degré du réseau même lorsque le réseau est enraciné[‡] et comporte une orientation des arêtes définissant un graphe orienté acyclique ayant la racine comme source. Plus précisément, nous avons identifié une classe de problèmes, appelée *problèmes à voisinage complet*, qui n'admet pas de telles solutions. Informellement, un problème à voisinage complet est un problème dans lequel pour chaque paire de voisins, il existe des états qui sont légitimes séparément mais pas simultanément. Par exemple, dans le problème du couplage maximal, si un processus n'est pas dans un couplage, alors son état est correct à condition que tous ses voisins soient dans des couplages.

‡. C'est-à-dire, il contient un processus distingué.

Cette classe de problèmes comporte bien d'autres problèmes comme l'ensemble indépendant maximal et la construction d'arbre couvrant.

Nous avons alors introduit une notion de stabilité moins contraignante : un algorithme est dit *finalement k-stable* si dans toutes ses exécutions, il existe un suffixe au cours duquel chaque processus lit les variables de communication d'au plus k voisins différents au moins une fois. Cette propriété admet des solutions auto-stabilisantes : dans [MIKW09], Masuzawa *et al* présentent un algorithme auto-stabilisant de construction d'arbre couvrant *finalement* 1-*stable* pour des réseaux enracinés de topologie quelconque. Cependant, une fois encore, tous les problèmes n'admettent pas de solution auto-stabilisante *finalement k-stable*. En effet, nous avons démontré dans [DMT09] que les problèmes à voisinage complet n'admettent pas de solutions auto-stabilisantes *finalement k-stable* où k est inférieur au degré dans les réseaux *anonymes*.

Finalement, nous avons introduit une version affaiblie de la *finalement k-stabilité* : un algorithme est dit *finalement* (x,k)-*stable* si dans toutes ses exécutions, il existe un sous-ensemble de processus S de taille x et un suffixe d'exécution au cours duquel chaque processus de S lit les variables de communication d'au plus k voisins différents au moins une fois.

4 Algorithmes

Dans cette section, nous illustrons les notions de 1-*efficacité* et de *finalement* $(x,1)$-*stabilité* en auto-stabilisation avec deux algorithmes. Ces deux algorithmes sont écrits pour des réseaux *colorés* de topologie quelconque. *Coloré* signifie que chaque processus dispose d'une constante « couleur » C qui diffère de celle de ses voisins. Nous supposons en outre que les couleurs sont totalement ordonnées.

Ensemble Indépendant Maximal. Nous proposons tout d'abord un algorithme auto-stabilisant qui construit un *ensemble indépendant maximal* du réseau. Un *ensemble indépendant* est un sous-ensemble de processus tel que aucun processus de cet ensemble n'est voisin d'un autre processus de cet ensemble. Un ensemble indépendant est dit *maximal* si aucun de ses sur-ensembles n'est un ensemble indépendant.

Dans notre algorithme, chaque processus dispose d'une variable de communication S qui a deux états possibles : *dominant* et *dominé*. Le but de notre algorithme est de converger vers une configuration où l'ensemble des processus « dominants » forme un ensemble indépendant maximal. Ainsi, toute configuration légitime de notre algorithme vérifie les deux propriétés suivantes :

- Tous les voisins d'un processus dominant sont des processus dominés (*indépendance*).
- Chaque processus dominé a au moins un voisin dominant (*maximalité*).

En plus de la variable S, chaque processus utilise une variable interne *Cur*. Cette variable désigne un voisin du processus et permet d'obtenir la 1-*efficacité* : un processus ne lit que l'état de communication de son voisin pointé par *Cur*.

Ensuite, en fonction de la valeur de S, chaque processus p adopte la stratégie suivante :

- *Si p est dominant* (c'est-à-dire, $S_p = dominant$), alors p regarde l'un après l'autre les états de communication de ses voisins (à chaque action, p lit l'état de communication de son voisin pointé par *Cur* puis change la valeur de *Cur* pour son voisin suivant). Si p découvre que l'un de ses voisins q est aussi dominant, alors la propriété d'indépendance est violée. Dans ce cas, soit p soit q doit devenir dominé. Nous utilisons alors les couleurs pour faire un choix déterministe, c'est-à-dire, le processus ayant la plus grande couleur devient dominé. Noter que dans une configuration légitime, chaque dominant contrôle continuellement ses voisins.
- *Si p est dominé* (c'est-à-dire, $S_p = dominé$), alors p doit avoir la garantie qu'au moins l'un de ses voisins est dominant (maximalité). Si le voisin de p pointé par Cur_p n'est pas dominant alors p devient dominant. Aussi, pour accélérer le temps de convergence, si le voisin q de p pointé par Cur_p a une couleur plus grande que celle de p alors p devient dominant même si q est dominant.

Nous avons prouvé dans [DMT09] qu'en suivant cette stratégie notre algorithme était auto-stabilisant, 1-*efficace* et *finalement* $(\lfloor \frac{\mathcal{L}_{max}+1}{2} \rfloor, 1)$-*stable* où $\mathcal{L}_{max}$ est la longueur (en nombre d'arêtes) du plus long chemin élémentaire du réseau, c'est-à-dire, qu'une fois que l'algorithme a convergé, au moins $\lfloor \frac{\mathcal{L}_{max}+1}{2} \rfloor$ processus ne lisent plus que les variables de communication d'un seul voisin. Enfin, notre algorithme stabilise en $O(\Delta \times \sharp\mathcal{C})$ rondes où Δ est le degré du réseau et $\sharp\mathcal{C}$ est le nombre de couleurs différentes utilisées.

Couplage Maximal. Notre deuxième algorithme auto-stabilisant construit un *couplage maximal* du réseau. Un *couplage* est un ensemble d'arêtes tel qu'il n'y aucun sommet qui est incident à deux arêtes différentes de l'ensemble. Un couplage est dit *maximal* si aucun de ses sur-ensembles n'est un couplage.

Dans notre algorithme, les processus utilisent la variable interne *Cur* de la même manière que dans l'algorithme précédent afin d'obtenir la *k-efficacité*.

Ensuite, le principe de base de l'algorithme est de créer des paires de voisins « mariés ». Les liens entre ces paires constituent le couplage maximal. Pour créer ces paires, chaque processus utilise une variable de communication *PR* qui désigne un voisin ou est égale à 0. Deux voisins sont *mariés* si et seulement si ils se désignent l'un l'autre avec leurs variables *PR*.

Lorsque PR_p vaut 0, cela signifie que le processus p n'est pas marié et qu'il n'a pas trouvé de voisin à qui proposer le mariage. Dans ce cas, p est dit *libre*.

Lorsque PR_p désigne un voisin q, cela signifie que le processus p est soit marié avec q soit il a proposé un mariage à q.

Chaque processus maintient une variable de communication booléenne M afin d'informer tous ses voisins de son statut (*marié* ou *pas marié*).

En se basant sur ces 3 variables, chaque processus p applique la stratégie suivante :

- p n'est autorisé à se marier qu'avec le voisin pointé par Cur_p. Donc, si $PR_p \notin \{0, Cur_p\}$, alors PR_p est affecté à Cur_p (ce cas est toujours consécutif à une faute).
- p maintient à jour M_p de telle manière que M_p soit vrai lorsqu'il est marié.
- Si p est libre et que son voisin désigné par Cur_p lui propose un mariage, alors p accepte en affectant PR_p à q.
- p réinitialise PR_p à 0 lorsque p n'est par marié et que son voisin pointé par PR_p est soit (i) marié avec un autre processus ($PR_q \notin \{0, p\}$) soit (ii) a une couleur plus petite que la sienne. La condition (i) évite que p attende un voisin déjà marié. La condition (ii) est utilisé pour casser les cycles.
- Lorsque p est libre, il doit essayer de se marier. Donc, il regarde un à un les états de communication de ses voisins jusqu'à trouver un voisin libre avec une couleur plus grande que la sienne. Donc, p fait varier Cur_p jusqu'à ce qu'il trouve un voisin q vérifiant la condition, dans ce cas p propose le mariage à q en affectant PR_p à q.

Nous avons prouvé dans [DMT09] qu'en suivant cette stratégie notre algorithme était auto-stabilisant, 1-*efficace* et *finalement* $(\lceil \frac{2m}{2\Delta-1} \rceil, 1)$-*stable* où m est le nombre d'arêtes du réseau et Δ est le degré du réseau, c'est-à-dire, qu'une fois que l'algorithme a convergé, au moins $\lceil \frac{2m}{2\Delta-1} \rceil$ processus ne lisent plus que les variables de communications d'un seul voisin. Enfin, notre algorithme stabilise en au plus $(\Delta+1)n+2$ rondes (n est le nombre de processus).

5 Conclusion

Dans cet article, nous nous sommes intéressés à la minimisation du surcoût en communication des algorithmes auto-stabilisants calculant des points fixes (algorithmes silencieux). Nous avons tout d'abord proposé de nouvelles mesures permettant d'évaluer ce surcoût, particulièrement dans la phase stabilisée. Dans un second temps, nous avons proposé deux algorithmes où le surcoût en communication est amoindri. Dans nos travaux futurs, nous souhaiterions généraliser notre approche en proposant une méthode de transformation automatique d'un algorithme auto-stabilisant en un algorithme auto-stabilisant efficace.

Références

[Dij74] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11) :643–644, 1974.

[DMT09] Stéphane Devismes, Toshimitsu Masuzawa, and Sébastien Tixeuil. Communication efficiency in self-stabilizing silent protocols. In *ICDCS*, pages 474–481. IEEE Computer Society, 22-26 June 2009.

[LFA00] Mikel Larrea, Antonio Fernández, and Sergio Arévalo. Optimal implementation of the weakest failure detector for solving consensus. In *SRDS*, pages 52–59, 2000.

[MIKW09] Toshimitsu Masuzawa, Taisuke Izumi, Yoshiaki Katayama, and Koichi Wada. Brief announcement : Communication-efficient self-stabilizing protocols for spanning-tree construction. In *OPODIS*, volume 5923 of *Lecture Notes in Computer Science*, pages 219–224. Springer, 2009.