Algorithme autostabilisant avec convergence sûre construisant une (f, g)-alliance

Fabienne Carrier, Ajoy Datta, Stéphane Devismes, Lawrence Larmore, Yvan Rivierre

Algorithme autostabilisant avec convergence sûre construisant une (f,g)-alliance

Fabienne Carrier[1], Ajoy K. Datta[2], Stéphane Devismes[1], Lawrence L. Larmore[2] et Yvan Rivierre[1]

[1] *VERIMAG, Université Grenoble 1, France*
[2] *School of Computer Science, UNLV, Las Vegas, USA*

Nous proposons un algorithme distribué, autostabilisant et silencieux avec convergence sûre qui calcule une (f,g)-alliance minimale dans un réseau asynchrone identifié, où f et g sont deux fonctions associant à chaque nœud un entier positif ou nul qui vérifient : pour tout nœud p, $f(p) \geq g(p)$ et $\delta_p \geq g(p)$, où δ_p est le degré du nœud p dans le réseau.

Keywords: système distribué, autostabilisation, convergence sûre, (f,g)-alliance.

1 Introduction

L'*autostabilisation* est un paradigme général permettant de concevoir des algorithmes distribués tolérant les fautes transitoires. Une faute transitoire est une panne non-définitive qui altère le contenu du 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 autostabilisant retrouve de lui-même, et en temps fini, un comportement normal dès lors que les fautes transitoires ont cessé. L'autostabilisation ne fait aucune hypothèse sur la nature ou l'ampleur des fautes transitoires qui frappent le système. Ainsi, l'autostabilisation permet à un système de récupérer de l'effet de ces fautes d'une manière unifiée. Cependant, l'autostabilisation a quelques inconvénients ; le plus important étant sûrement la perte de sûreté temporaire, *i.e.*, suite à des fautes transitoires, il y a une période temporaire — appelée *phase de stabilisation* — au cours de laquelle le système n'offre aucune garantie de sûreté.

Plusieurs approches dérivées de l'autostabilisation ont été introduites pour offrir des propriétés supplémentaires lors de la phase de stabilisation, *e.g.*, la superstabilisation, la contention de fautes ou encore la convergence sûre. Nous considérons ici la notion de *convergence sûre*.

Pour la plupart des problèmes, il est souvent difficile de concevoir des algorithmes autostabilisants assurant un temps de stabilisation réduit, même lorsque peu de fautes touchent le réseau. L'importance du temps de stabilisation est souvent dûe à la complexité de la spécification que l'algorithme doit satisfaire. Ainsi, le but d'un algorithme *autostabilisant avec convergence sûre* est en premier lieu de converger rapidement d'une configuration quelconque vers une configuration dite « légitime réalisable », où une qualité de service minimum est garantie ; lorsqu'une telle configuration est atteinte, l'algorithme doit, tout en restant dans l'ensemble des configurations légitimes réalisables, continuer à converger vers un sous-ensemble particulier de ces configurations, appelé sous-ensemble des configurations « légitimes optimales », où une spécification plus complexe est garantie. L'autostabilisation avec convergence sûre est particulièrement intéressante pour résoudre les problèmes de construction de structure. Par exemple, des solutions autostabilisantes avec convergence sûre ont été proposées pour le calcul d'un ensemble dominant minimal [2], d'un ensemble dominant faiblement connecté approximant l'optimum [3], ou encore d'un ensemble dominant connecté approximant l'optimum [4].

Dans cet article, nous nous intéressons au calcul d'une (f,g)-alliance. Soit $G = (V,E)$ un graphe non-orienté. Soit f et g deux fonctions qui associent un entier positif ou nul à chaque nœud de V. Un sous-ensemble de nœuds A est une (f,g)-*alliance* de G *ssi* :

$$(\forall p \in V \setminus A, |\mathcal{N}_p \cap A| \geq f(p)) \wedge (\forall p \in A, |\mathcal{N}_p \cap A| \geq g(p))$$

où $\mathcal{N}_p$ désigne l'ensemble des voisins de p dans G. De plus, A est une (f,g)-alliance *minimale* si aucun de ses sous-ensembles propres n'est une (f,g)-alliance. Les (f,g)-alliances généralisent plusieurs notions issues de la théorie des graphes, qui sont notamment utilisées dans le domaine des systèmes distribués. Considérons un sous-ensemble de nœuds S et notons δ_p le degré d'un nœud p :

1. S est un ensemble dominant (minimal) *ssi* S est une $(1,0)$-alliance (minimale) ;
2. S est un ensemble k-dominant [†] (minimal) *ssi* S est une $(k,0)$-alliance (minimale) ;
3. S est un ensemble k-tuple dominant (minimal) *ssi* S est une $(k,k-1)$-alliance (minimale) ;
4. S est une alliance défensive (minimale) *ssi* S est une $(f,0)$-alliance (minimale), telle que $\forall p \in V$, $f(p) = \lceil \delta_p/2 \rceil$;
5. S est une alliance offensive (minimale) *ssi* S est une $(1,g)$-alliance (minimale), telle que $\forall p \in V$, $g(p) = \lceil \delta_p/2 \rceil$.

Contributions. Nous proposons un algorithme distribué, autostabilisant et silencieux [‡] qui calcule une (f,g)-alliance minimale dans un réseau asynchrone identifié, où f et g sont deux fonctions de V dans $\mathbb{N}$ vérifiant les deux contraintes suivantes : (1) $f \geq g$, *i.e.*, $\forall p \in V, f(p) \geq g(p)$; et (2) $\forall p \in V, \delta_p \geq g(p)$ [§]. La classe des (f,g)-alliances minimales vérifiant $f \geq g$ englobe les ensembles dominants, k-dominants et k-tuple dominants minimaux, ainsi que les alliances défensives minimales. Cependant, les alliances offensives minimales ne font pas partie de cette classe.

Notre algorithme garantit une convergence sûre dans le sens où à partir d'une configuration quelconque, il calcule tout d'abord une (f,g)-alliance (pas nécessairement minimale) en au plus 4 rondes, puis il supprime des éléments de l'alliance jusqu'à obtenir une (f,g)-alliance minimale. Le temps de stabilisation global étant d'au plus $8n+4$ rondes, où n est le nombre de nœuds.

Notre algorithme est écrit dans le modèle à états. Il est prouvé en supposant un démon distribué inéquitable (l'hypothèse la plus faible du modèle). Il nécessite $O(\log n)$ bits de mémoire par processus et son temps de stabilisation en termes d'étapes de calcul est $O(\Delta^3 n)$, où Δ est le degré du réseau.

État de l'art. Les (f,g)-alliances ont été introduites dans [1]. Dans le même article, les auteurs proposent plusieurs algorithmes distribués non-autostabilisants pour ce problème et ses variantes. À notre connaissance, il n'y avait pas jusqu'alors de travaux proposant des algorithmes autostabilisants pour les (f,g)-alliances. Cependant, il existe des algorithmes autostabilisants pour des instances particulières de (f,g)-alliances. L'article le plus proche de nos travaux est certainement celui de Kakugawa *et al* [2]. Dans cet article, les auteurs proposent un algorithme autostabilisant avec convergence sûre pour calculer un ensemble dominant minimal d'un réseau synchrone en $O(\mathcal{D})$ rondes, où $\mathcal{D}$ est le diamètre du réseau.

Plan. Dans la section 2, nous décrivons brièvement le modèle dans lequel notre algorithme est écrit. Dans la section 3, nous rappelons une propriété sur laquelle notre algorithme se fonde. Nous présentons les idées principales de notre algorithme dans la section 4 [¶]. Nous concluons avec des perspectives dans la section 5.

2 Modèle à états

Nous considérons des réseaux bidirectionnels de n processus (ou nœuds) où chaque processus peut communiquer directement avec un ensemble restreint d'autres processus appelés *voisins*. Les processus sont identifiés de manière unique. Dans la suite, nous ne distinguerons pas un processus de son identité. Ainsi, selon le contexte, p désignera soit un processus, soit son identité.

Les processus communiquent par le biais de *variables de communication* localement partagées : chaque processus détient un nombre fini de variables, chacune de domaine fini, dans lesquelles il peut lire et écrire. De plus, il peut lire les variables de ses voisins.

Les variables d'un processus définissent son état. Une configuration est un vecteur contenant un état de chaque processus. L'exécution d'un algorithme est une suite d'*étapes atomiques* : à chaque étape, s'il existe

†. Nous considérons la définition qui dit que S est k-dominant *ssi* tout nœud hors de S a au moins k voisins dans S.

‡. Un algorithme silencieux converge vers une configuration dite « terminale » où les valeurs des variables de communication de tous les processus ne changent plus.

§. Cette contrainte assure l'existence d'une solution.

¶. Pour plus de détails, voir le rapport technique en ligne (`www-verimag.imag.fr/TR/TR-2012-19.pdf`).

des processus — dits *activables* — souhaitant modifier leur état, alors un sous-ensemble non-vide de ces processus est activé. En une étape atomique, chaque processus activé lit ses propres variables, ainsi que celles de ses voisins, puis modifie son état. Nous supposons ici que les processus sont activés de manière asynchrone sans hypothèse d'équité particulière (*i.e.*, le démon est inéquitable). Toute configuration où aucun processus n'est activable est dite *terminale*.

Nous calculons le temps de stabilisation suivant deux métriques : le nombre d'étapes atomiques et le nombre de *rondes*. Cette dernière métrique 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 continûment activables depuis le début de l'exécution ont été activés au moins une fois, la seconde ronde commence immédiatement après, *etc*.

3 Minimalité et 1-minimalité

Une (f,g)-alliance A est 1-*minimale ssi* $\forall p \in A, A \setminus \{p\}$ n'est pas une (f,g)-alliance, *i.e.*, la suppression d'un seul élément de A provoque la perte de la propriété. Étonnamment, une (f,g)-alliance 1-minimale n'est pas nécessairement une (f,g)-alliance minimale [1]. Cependant, nous avons la propriété suivante :

Propriété 1 ([1]) *Soit f et g deux fonctions associant un entier positif ou nul à chaque nœud.*

1. *Toute (f,g)-alliance minimale est aussi 1-minimale ;*
2. *Si $f \geq g$, alors toute (f,g)-alliance 1-minimale est aussi minimale.*

4 Principes de l'algorithme

Notre algorithme prend en entrée les deux fonctions f et g et calcule une seule variable de sortie par processus p : le booléen $p.inA$, qui est vrai *ssi* p est dans l'alliance. Soit γ une configuration. Nous notons A_γ l'ensemble $\{p \in V, p.inA = \text{vrai dans } \gamma\}$ (nous omettons l'indice γ lorsque le contexte est clair). Notre algorithme converge, à partir d'une configuration quelconque, vers une configuration terminale où A_γ est une (f,g)-alliance 1-minimale. Ainsi, puisque nous supposons que $f \geq g$, A_γ est aussi une (f,g)-alliance minimale.

La configuration initiale étant quelconque, un processus peut avoir à quitter ou rejoindre l'alliance en cours d'exécution. L'idée intuitive pour obtenir la convergence sûre est qu'il doit être plus difficile pour un processus de quitter l'alliance que de la rejoindre. Ainsi, cela permet de converger rapidement vers une configuration où A est une (f,g)-alliance, mais pas nécessairement une (f,g)-alliance minimale.

4.1 Quitter l'alliance

Pour obtenir la 1-minimalité, un processus p est autorisé à quitter l'alliance si les deux conditions suivantes sont vérifiées :

Condition 1 : p aura au moins $f(p)$ voisins dans l'alliance après l'avoir quittée ;

Condition 2 : chaque voisin q de p aura encore assez de voisins (*i.e.*, $f(q)$ ou $g(q)$, en fonction de la valeur de $q.inA$) dans l'alliance une fois p hors de l'alliance.

Vérifier la condition 1. Pour assurer la première condition, nous avons introduit un mécanisme qui impose que les suppressions dans A soient « localement » séquentielles, c'est-à-dire qu'à chaque étape de l'exécution, au plus un processus quitte l'alliance dans le voisinage de chaque processus. Ainsi, si un processus p souhaite quitter l'alliance, il a seulement besoin de vérifier que son nombre de voisins dans l'alliance, $\texttt{NbA}(p) = |\{q \in \mathcal{N}_p, q.inA\}|$, est supérieur ou égal à $f(p)$ avant de quitter l'alliance. Alors, s'il quitte effectivement l'alliance, il est le seul dans son voisinage à pouvoir le faire et ainsi la condition 1 reste vraie.

Pour réaliser les suppressions localement séquentielles, chaque processus q dispose d'un pointeur $q.choice$. q affecte $\perp$ à $q.choice$ s'il ne peut autoriser aucun voisin à quitter l'alliance, c'est-à-dire s'il est dans l'alliance (resp. hors de l'alliance) et $\texttt{NbA}(q) \leq g(q)$ (resp. $\texttt{NbA}(q) \leq f(q)$). Dans le cas contraire, q désigne l'un de ses voisins qui appartient à l'alliance pour l'autoriser à la quitter.

Ainsi, pour quitter l'alliance, tout processus p ne doit autoriser aucun voisin à quitter l'alliance ($p.choice = \perp$) et doit recevoir l'autorisation de tous ses voisins ($\forall q \in \mathcal{N}_p, q.choice = p$).

Vérifier la condition 2. Cette condition est aussi garantie par le fait qu'un processus p peut quitter l'alliance seulement s'il a une autorisation de chacun de ses voisins q. Un voisin q donne une telle autorisation à p seulement s'il a assez de voisins dans l'alliance sans tenir compte de p. Ainsi, q exécute $q.choice \leftarrow q'$ où q' est l'un de ses voisins seulement si q' est dans l'alliance et que, sans q', q a toujours assez de voisins dans l'alliance.

Il est possible qu'un voisin q' de q ne puisse quitter l'alliance — dans ce cas, q' est dit « occupé » — parce qu'au moins une des deux conditions suivantes est vraie :

(a) $\texttt{NbA}(q') < f(q')$. Dans ce cas, q' n'a pas assez de voisins dans l'alliance pour être hors de l'alliance.

(b) *au moins un voisin de q' a besoin que q' reste dans l'alliance*.

Si nous laissons les processus désigner avec leur pointeur *choice* de tels processus occupés, cela peut mener à un interblocage. Nous utilisons une variable booléenne supplémentaire $q'.busy$ pour informer les voisins de q' que celui-ci est occupé, c'est-à-dire qu'il vérifie la condition (a) ou la condition (b). Ainsi, un processus q n'a pas le droit de désigner un voisin q' tel que $q'.busy = \text{vrai}$.

La condition (a) est évaluée par q' en lisant sa variable *inA* et celles de ses voisins. En revanche, la condition (b) nécessite que q' ait accès au statut (*inA*) et au nombre de voisins dans l'alliance de chacun de ses voisins. Cette dernière information nécessite l'introduction d'une nouvelle variable, notée *nbA*, dans laquelle chaque processus maintient son nombre de voisins dans l'alliance.

Si q a plusieurs candidats, c'est-à-dire des voisins dans l'alliance qui ne sont pas occupés, alors il choisit celui avec l'identité la plus petite. Il reste un problème : q peut être le candidat de son propre candidat. Nous résolvons ce problème en ajoutant la condition suivante : si q est lui-même candidat, il désigne un candidat à condition que celui-ci ait une identité plus petite que la sienne.

Enfin, il est possible qu'un processus q change la valeur de son pointeur alors que, simultanément, l'un de ses voisins, q', quitte l'alliance. Dans ce cas, la condition 2 peut être violée. En effet, q choisit un candidat en supposant que q' reste dans l'alliance. Cette situation ne peut se présenter que si la valeur précédente de $q.choice$ était q'. Pour éviter une telle situation, nous forçons un processus à réinitialiser son pointeur à $\perp$, afin d'interdire à tout voisin de quitter l'alliance, avant d'envisager de désigner un nouveau candidat.

4.2 Rejoindre l'alliance

Un processus p hors de l'alliance doit rejoindre l'alliance si :

1. il n'a pas assez de voisins dans l'alliance, *i.e.*, $\texttt{NbA}(q) < f(q)$, ou
2. l'un de ses voisins a besoin que p joigne l'alliance.

Chaque processus peut évaluer la première condition en lisant sa variable *inA* et celles de ses voisins. Pour la seconde condition, un processus p doit lire dans l'état de chaque voisin q, (i) son statut $q.inA$ et (ii) son nombre de voisins dans l'alliance $q.nbA$.

Enfin, notez que pour empêcher un processus d'entrer et sortir indéfiniment de l'alliance, nous imposons qu'un processus ne puisse rentrer dans l'alliance qu'une fois que tous ses voisins arrêtent de le désigner (cela introduit un retard d'au plus une ronde).

5 Perspectives

La perspective immédiate de ce travail est d'étudier la possibilité de réduire le temps de stabilisation global à $O(\mathcal{D})$ rondes, où $\mathcal{D}$ est le diamètre du réseau. Ensuite, il serait intéressant de trouver un algorithme autostabilisant avec convergence sûre construisant des (f,g)-alliances sans l'hypothèse $f \geq g$.

Références

[1] M. C. Dourado, L. D. Penso, D. Rautenbach, and J. L. Szwarcfiter. The south zone : Distributed algorithms for alliances. In *SSS*, pages 178–192, 2011.

[2] H. Kakugawa and T. Masuzawa. A self-stabilizing minimal dominating set algorithm with safe convergence. In *IPDPS*, 2006.

[3] S. Kamei and H. Kakugawa. A self-stabilizing approximation algorithm for the minimum weakly connected dominating set with safe convergence. In *WRAS*, pages 57–67, 2007.

[4] S. Kamei and H. Kakugawa. A self-stabilizing 6-approximation for the minimum connected dominating set with safe convergence in unit disk graphs. *Theoretical Computer Science*, 428 :80–90, 2012.