Silence dans la forêt ![†]

Karine Altisen[1], Stéphane Devismes[1] et Anaïs Durand[2]

[1] *Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, 38000 Grenoble*
[2] *Sorbonne Université, CNRS, Inria, LIP6, F-75005 Paris, France*

Nous formalisons des schémas d'algorithmes distribués, classiquement utilisés en autostabilisation, afin d'obtenir des résultats généraux relatifs à leur correction et leur complexité. Précisément, nous étudions une classe d'algorithmes dédiés aux réseaux munis d'un sens de direction décrivant une forêt couvrante. La définition de cette classe est simple au sens où elle est quasi-syntaxique. Tous les algorithmes de cette classe sont (1) autostabilisants et silencieux, et (2) ont un temps de stabilisation à la fois polynomial en mouvements et asymptotiquement optimal en rondes. Pour illustrer la polyvalence de notre méthode, nous passons en revue plusieurs travaux où nos résultats s'appliquent.

Mots-clefs : autostabilisation, silence, arbres, forêts

1 Introduction

L'*autostabilisation* est une propriété caractérisant une aptitude d'un système distribué à tolérer des *fautes transitoires*. Les fautes transitoires sont rares, de durée finie et affectent l'état du composant du réseau (processus ou lien) où elles surviennent. La corruption de la mémoire locale d'un processus ou du contenu d'un message en transit sont deux exemples de fautes transitoires. Après un nombre fini de fautes transitoires et en supposant que ces fautes n'aient pas affecté le code de l'algorithme, un système autostabilisant retrouve de lui-même et en temps fini un comportement correct, c'est-à-dire conforme à sa spécification.

Contexte. Les forêts et arbres couvrants sont omniprésents en autostabilisation. Par exemple, on retrouve souvent ces structures de données dans la *composition* d'algorithmes. La composition [2] est une méthode modulaire, usuelle en autostabilisation, permettant de simplifier à la fois la conception et les preuves d'algorithmes. De nombreux algorithmes autostabilisants sont conçus comme une composition entre un algorithme de construction de forêt ou d'arbre couvrant et un algorithme dédié aux topologies arborescentes (arbres ou forêts). De multiples algorithmes autostabilisants efficaces de construction d'arbre ont déjà été proposés, *cf.* l'état de l'art proposé par Gärtner [6]. La majorité d'entre-eux sont aussi silencieux. Un algorithme *silencieux* converge vers une configuration dite *terminale* à partir de laquelle les valeurs des variables de communication des processus ne changent plus. Nous nous intéressons ici aux algorithmes autostabilisants utilisant une structure arborescente, par exemple, calculée par l'un de ces algorithmes dans le cadre d'une composition. De manière générale, les algorithmes dédiés aux structures arborescentes sont fondés (quasiment uniquement) sur des approches ascendantes (*bottom-up*) et/ou descendantes (*top-down*).

Contribution. Nous formalisons ici ces pratiques usuelles pour en faire un résultat systématique. Nous définissons dans le modèle à états (le modèle de calcul communément utilisé en autostabilisation), une classe d'algorithmes, dédiés aux réseaux munis d'un sens de direction décrivant une forêt couvrante, et utilisant des approches ascendantes et descendantes. Évaluer l'appartenance d'un algorithme à cette classe est quasi-syntaxique. Ses algorithmes sont autostabilisants et silencieux sous l'hypothèse d'un démon distribué inéquitable, l'hypothèse d'ordonnancement la plus générale du modèle. De plus, ils sont efficaces en temps de stabilisation, *i.e.*, la durée maximale avant que l'algorithme retrouve un comportement correct, puisque celui-ci est polynomial en mouvements et asymptotiquement optimal en rondes. Le challenge a été d'obtenir un compromis entre efficacité et généralité. En effet, plus la classe considérée est générale, plus il est difficile de prouver des bornes de complexité fines pour tous les algorithmes de la classe.

[†] Cette étude a été financée par les projets ANR DESCARTES (ANR-16-CE40-0023) et ESTATE (ANR-16-CE25-0009).

Plan. Dans la section suivante, nous décrivons rapidement le modèle à états ainsi que quelques notions et notations utilisées dans le reste de l'article. Dans la section 3, nous présentons notre classe d'algorithmes ainsi que les résultats de complexité obtenus pour cette classe. Nous illustrons notre définition avec un exemple jouet montrant que la condition d'appartenance à cette classe est simple à évaluer. Par manque de place, toutes les preuves de nos résultats sont omises (elles sont disponibles dans l'article [1]). Nous concluons dans la section 4 en passant en revue une partie des travaux existants où nos résultats s'appliquent.

2 Modèle

Nous considérons des réseaux bidirectionnels asynchrones connexes de n processus où chaque processus peut communiquer directement avec un sous-ensemble d'autres processus appelés *voisins*. On note V l'ensemble des processus. Ces réseaux sont munis d'un sens de direction : chaque processus p possède deux entrées constantes $p.père$ et $p.enfs$. Lorsque $p.père = \perp$, p est appelé *racine*. Dans le cas contraire, $p.père$ désigne un voisin appelé le père de p. La constante $p.enfs$ est l'ensemble des enfants de p, c'est-à-dire les voisins le désignant comme père. Si $p.enfs = \emptyset$, alors p est une *feuille*. Nous supposons que le sous-graphe orienté induit par les pointeurs pères est une forêt couvrante du réseau.

Le modèle à états est une abstraction du modèle à passage de messages dans lequel les échanges d'informations sont réalisés *via* des *variables localement partagées* : chaque processus détient un nombre fini de variables dans lesquelles il peut lire et écrire ; de plus, il peut lire les variables de ses voisins dans le réseau. L'*état* d'un processus est défini par la valeur de ses variables. La *configuration* du système est définie par l'état de chacun de ses processus. Un algorithme distribué est un ensemble de n algorithmes locaux, un par processus, où chaque algorithme local consiste en un ensemble fini de règles. Chaque règle est de la forme $Etiquette :: Garde \rightarrow Action$ où l'étiquette identifie la règle, sa garde est un prédicat booléen sur les variables du processus et de ses voisins, et l'action est un ensemble d'affectations modifiant l'état du processus. L'exécution d'un algorithme distribué est composée d'une succession d'*étapes atomiques* de calcul. À chaque étape de calcul, chaque processus détermine en fonction de son état et de celui de ses voisins s'il est *activable*, c'est-à-dire si la garde d'au moins une de ses règles est vraie. Un adversaire, le *démon*, choisit ensuite un sous-ensemble non vide de processus activables. Les processus choisis sont alors *activés* : ils effectuent un *mouvement* où ils exécutent atomiquement la partie action d'une de leur règles activables. Nous supposons ici que les processus sont activés de manière asynchrone sans hypothèse particulière sur l'équité (le démon est *distribué inéquitable*). S'il n'y a pas de processus activables, l'exécution se termine et la dernière configuration est dite *terminale*.

Le temps de stabilisation d'un algorithme est généralement exprimé selon deux types de mesures : le nombre de *mouvements* et de *rondes*. La première ronde d'une exécution termine dès lors que tous les processus qui étaient continûment activables depuis le début de l'exécution ont exécuté au moins une règle. La seconde ronde commence à la fin de la première, *etc.*

3 Notre résultat

Nous considérons maintenant un algorithme distribué quelconque $\mathcal{A}$ et sa spécification sous la forme d'un prédicat SP sur les configurations. Notre principal résultat est le théorème 1. Les notions et notations utilisées dans ce théorème seront définies et illustrées avec un exemple jouet dans la suite de la section.

Théorème 1 *Si $\mathcal{A}$* suit une stratégie acyclique *et toute configuration terminale de $\mathcal{A}$ satisfait SP alors*

- *$\mathcal{A}$ est autostabilisant et silencieux pour SP ;*
- *son temps de stabilisation en mouvements est borné par $\left(1+\mathbf{d}\cdot(1+\Delta)\right)^{\mathfrak{H}}\cdot k\cdot n^{\mathfrak{H}+2}$; et*
- *si $\mathcal{A}$ est (en plus) localement mutuellement exclusif, alors son temps de stabilisation en rondes est au plus $(\mathfrak{H}+1)\cdot(H+1)$;*

où Δ est le degré du réseau, H est la hauteur de la forêt couvrant le réseau, $\mathfrak{H}$ et $\mathbf{d}$ sont respectivement la hauteur et le degré entrant du graphe dit « de causalité » $\mathbf{GC}$, et k est le nombre de familles de $\mathcal{A}$.

Notez que $\mathfrak{H}$, $\mathbf{d}$ et k sont quasiment toujours des constantes indépendantes de la taille du réseau.

3.1 Un exemple illustratif

Dans la suite, nous proposons d'étudier un algorithme simple, noté $\mathcal{TE}$, fonctionnant dans un réseau en arbre orienté enraciné en r. Le but de cet algorithme est de calculer la somme des entrées de chaque

processus et de diffuser ce résultat à l'ensemble des processus du réseau. Ainsi, chaque processus p détient une entrée constante entière $p.E \in \mathbb{N}$. Ensuite, p maintient deux variables : $p.sub \in \mathbb{N}$ — dans laquelle p calcule la somme des entrées de ses descendants dans l'arbre — et $p.res \in \mathbb{N}$ — la sortie de l'algorithme qui devra finalement contenir la somme de toutes les entrées. Le but de $\mathcal{TE}$ est donc de converger vers une configuration terminale vérifiant le prédicat $\texttt{Somme} \equiv \left(\forall p \in V, p.res = \sum_{q \in V} q.E\right)$. Pour cela, $\mathcal{TE}$ est défini par les règles suivantes.

Pour tout processus p :	$S(p) \;::\; p.sub \neq (\sum_{q \in p.enfs} q.sub) + p.E \;\rightarrow\; p.sub := (\sum_{q \in p.enfs} q.sub) + p.E$
Pour la racine r :	$R(r) \;::\; r.res \neq r.sub \;\rightarrow\; r.res := r.sub$
Pour tout processus $p \neq r$:	$R(p) \;::\; p.res \neq \max(p.père.res, p.sub) \;\rightarrow\; p.res := \max(p.père.res, p.sub)$

3.2 Stratégie acyclique

Définition : $\mathcal{A}$ suit une *stratégie acyclique* si (1) $\mathcal{A}$ est *bien formé*, (2) son *graphe de causalité* **GC** est sans circuit, et (3) pour toute *famille de règles* A_i de sa *partition familiale*, A_i est *ponctuelle* et soit *descendante*, soit *ascendante*. Nous explicitons maintenant, les notions employées dans cette définition.

Famille de règles bien formée. Tout d'abord, nous appelons *famille de règles* A_i un ensemble de n règles où chaque règle appartient à un processus différent. On notera $A_i(p)$ la règle du processus p appartenant à la famille A_i.

Un algorithme distribué est *bien formé* si ses règles peuvent être partitionnées en familles $A_1, \ldots, A_k$, appelées *partition familiale*, telles que $\forall i \in \{1, \ldots, k\}$, les variables écrites (donc non-constantes) par les règles de A_i ont le même nom. Dans $\mathcal{TE}$ nous avons deux familles : $S = \{S(p) \mid p \in V\}$ (qui écrit dans les variables *sub*) et $R = \{R(p) \mid p \in V\}$ (qui écrit dans les variables *res*). Il faut plutôt comprendre la propriété « bien formé » comme une norme d'écriture permettant de simplifier l'analyse de l'algorithme.

Graphe de causalité. Nous supposons maintenant que $\mathcal{A}$ est bien formé. Soient $A_1, \ldots, A_k$ la partition familiale de $\mathcal{A}$. On définit la relation binaire $\prec_{\mathcal{A}}$ sur les familles de $\mathcal{A}$ comme suit : $A_j \prec_{\mathcal{A}} A_i$ si et seulement si $i \neq j$ et il existe au moins deux processus p et q tels que la règle $A_j(p)$ écrit dans des variables lues par $A_i(q)$, *i.e.*, apparaissant dans sa garde. Pour $\mathcal{TE}$, on a $S \prec_{\mathcal{TE}} R$. Nous représentons ensuite cette relation sous la forme d'un graphe **GC** appelé graphe de *causalité* : $\mathbf{GC} = (\{A_1, ..., A_k\}, \{(A_j, A_i) \mid A_j \prec_{\mathcal{A}} A_i\})$. Pour $\mathcal{TE}$, **GC** consiste simplement en l'unique arc $S \longrightarrow R$, en particulier ici **GC** est sans circuit.

Famille de règles ponctuelle. Une famille de règles A_i est *ponctuelle* si pour tout processus p, la règle $A_i(p)$ devient inactivable à chaque fois que $A_i(p)$ est exécutée et qu'à part les variables écrites par $A_i(p)$, aucune autre variable lue par $A_i(p)$ n'est modifiée. Il s'agit du seul critère non syntaxique pour évaluer si l'algorithme suit une stratégie acyclique. Les deux familles de $\mathcal{TE}$, S et R, sont ponctuelles car les actions de leur règles rendent leurs gardes fausses.

Famille de règles descendante/ascendante. Intuitivement, une famille de règles A_i est *descendante* si ses règles sont uniquement propagées vers le bas dans la forêt, *i.e.*, quand la règle $A_i(q)$ est exécutée, elle ne peut rendre activable des règles de sa famille A_i qu'au niveau de ses enfants. Dans ce cas, $A_i(q)$ écrit dans certaines variables lues par des règles $A_i(p)$ de ses enfants, ces variables pouvant être comparées aux variables écrites par $A_i(p)$ elle-même. Ainsi, une famille de règles A_i est *descendante* si pour tout processus p, pour toute variable $q.v$ lue par $A_i(p)$, $q.v$ est écrite par $A_i(q)$ seulement si $q = p$ ou $q = p.père$. Similairement, une famille de règles A_i est *ascendante* si pour tout processus p, toute variable $q.v$ lue par $A_i(p)$, $q.v$ est écrite par $A_i(q)$ seulement si $q = p$ or $q \in p.enfs$. Dans $\mathcal{TE}$, S est ascendante et R est descendante.

3.3 Complexité en mouvement de $\mathcal{TE}$

Nous avons vu que l'algorithme distribué $\mathcal{TE}$ suit une stratégie acyclique. De plus, par induction, nous pouvons facilement démontrer que toute configuration terminale de $\mathcal{TE}$ satisfait le prédicat $\texttt{Somme}$. Ainsi, d'après le théorème 1, $\mathcal{TE}$ est autostabilisant et silencieux sous l'hypothèse d'un démon distribué inéquitable. De plus, puisque $\mathfrak{H} = 1$, $\mathbf{d} = 1$ et $k = 2$, le temps de stabilisation en mouvements de $\mathcal{TE}$ est au plus $(4 + 2\Delta) \cdot n^3$. En utilisant un lemme intermédiaire, aussi proposé dans notre article [1], cette borne

peut être affinée pour obtenir un temps de stabilisation d'au plus $(3+2H)\cdot n^2$ mouvements. Cette dernière complexité est précise car nous avons aussi prouvé qu'elle était atteignable sur des exemples.

3.4 Complexité en rondes de $\mathcal{TE}$

Malheureusement suivre une *stratégie acyclique* n'est pas suffisant pour garantir des bornes intéressantes en nombre de rondes. Dans notre article [1], nous montrons l'existence d'un pire des cas en $\Omega(n)$ rondes pour $\mathcal{TE}$ lorsque le réseau est un arbre de hauteur 1 ! Comme le suggère le théorème 1 cela est dû au fait que les familles S et R de $\mathcal{TE}$ ne sont pas localement mutuellement exclusives (*n.b.*, ce critère n'est pas syntaxique). En effet, deux familles A_i et A_j sont localement mutuellement exclusives si pour tout processus p, on ne peut jamais avoir $A_i(p)$ et $A_j(p)$ activables simultanément. Or, dans $\mathcal{TE}$, les règles $R(p)$ et $S(p)$ d'un même processus p peuvent être activables dans la même configuration.

Notre théorème montre que si un algorithme suit une stratégie acyclique et est localement mutuellement exclusif (c'est-à-dire, les familles qui le constituent sont deux à deux localement mutuellement exclusives) alors *(a)* il est autostabilisant, *(b)* sa complexité en mouvements est polynomiale et *(c)* sa complexité en rondes est dans la majorité des cas asymptotiquement optimale (en effet, k est généralement une constante).

Nous avons aussi proposé une transformation simple permettant de rendre localement mutuellement exclusif tout algorithme suivant une stratégie acyclique. Cette méthode ne dégrade pas la complexité en mouvement de l'algorithme à transformer. Elle consiste simplement à ajouter des priorités aux règles de l'algorithme. Ces priorités suivent n'importe quel ordre total compatible avec $\prec_{\mathcal{A}}$. Pour réaliser ces priorités, il suffit d'ajouter à la garde de chaque règle de chaque processus p la négation de la disjonction des gardes des règles de p plus prioritaires. Ainsi, aucune règle de p ne peut plus être activable en même temps qu'une autre règle de p plus prioritaire. Par exemple, pour $\mathcal{TE}$, il suffit d'ajouter la négation de la garde de $S(p)$ à la garde de $R(p)$ pour tout processus p. Ainsi, nous obtenons un nouvel algorithme autostabilisant pour `Somme` qui stabilise en au plus $2H+2$ rondes et au plus $(4+2\Delta)\cdot n^3$ mouvements.

4 Conclusion

Nous avons étudié la correction et la complexité d'un schéma d'algorithmes distribués, classiquement utilisés en autostabilisation, dédié aux réseaux munis d'un sens de direction décrivant une forêt couvrante. Nos résultats permettent de déduire aisément (*i.e.*, quasi-syntaxiquement) des bornes fines sur le temps de stabilisation en mouvements et en rondes de ces algorithmes. Nous avons identifié plusieurs algorithmes de la littérature, *e.g.*, [8, 4, 3, 7], où s'applique notre schéma. Pour plusieurs de ces travaux (*e.g.*, [4, 7]), notre approche permet de déduire la stabilisation de l'algorithme sous des hypothèses plus générales, c'est-à-dire, sous l'hypothèse d'un démon distribué inéquitable. Dans certain cas (*e.g.*, [8, 3]), nous exhibons une borne de complexité (en mouvements ou en rondes) alors qu'elle n'avait pas été étudiée dans l'article original.

Dans la plupart des articles, l'hypothèse de l'existence d'une forêt couvrante est une hypothèse intermédiaire puisque cette dernière est construite par un algorithme sous-jacent. Il existe de nombreux algorithmes autostabilisants efficaces (en mouvements et en rondes), *e.g.* [5], permettant de construire de telles structures. La composition des instances de notre schéma avec ces derniers doit être attentivement abordée afin d'obtenir un algorithme combiné efficace.

Références

[1] K. Altisen, S. Devismes, and A. Durand. Acyclic strategy for silent self-stabilization in spanning forests. In *SSS'18*, pages 186–202, 2018.

[2] A. Arora, M. Gouda, and T. Herman. Composite routing protocols. In *SPDP'90*, pages 70–78, 1990.

[3] P. Chaudhuri. An $O(n^2)$ self-stabilizing algorithm for computing bridge-connected components. *Computing*, 62(1) :55–67, 1999.

[4] P. Chaudhuri and H. Thompson. Self-stabilizing tree ranking. *Int. J. Comput. Math.*, 82(5) :529–539, 2005.

[5] S. Devismes, D. Ilcinkas, and C. Johnen. Silent Self-Stabilizing Scheme for Spanning-Tree-like Constructions. In *ICDCN'19*, pages 158–167, 2019.

[6] F. C. Gärtner. A survey of self-stabilizing spanning-tree construction algorithms. Technical report, EPFL, 2003.

[7] M. H. Karaata. A self-stabilizing algorithm for finding articulation points. *Int. J. Found. Comput. Sci.*, 10(1) :33–46, 1999.

[8] V. Turau and S. Köhler. A distributed algorithm for minimum distance-k domination in trees. *J. Graph Algorithms Appl.*, 19(1) :223–242, 2015.