Borne inférieure optimale pour la complexité spatiale des algorithmes déterministes auto-stabilisants d'élection

Lélia Blin, Laurent Feuilloley, Gabriel Le Bouder

Borne inférieure optimale pour la complexité spatiale des algorithmes déterministes auto-stabilisants d'élection.

Lélia Blin[1,2] and Laurent Feuilloley[4] and Gabriel Le Bouder[1,3]

[1] *Sorbonne Université, CNRS, LIP6 UMR 7606, 4 place Jussieu, 75005 Paris, France.* †
[2] *Université d'Evry-Val-d'Essonne.*
[3] *INRIA*
[4] *Univ. Lyon, Université Lyon 1, LIRIS UMR CNRS 5205, F-69621, Lyon, France.*

Dans cet article nous considérons les réseaux non anonymes et des problématiques dont la solution peut être capturée par un prédicat booléen Π, problématiques comme la coloration des noeuds, l'élection... Un algorithme auto-stabilisant pour Π est un algorithme distribué qui, quelle que soit la configuration initiale du réseau (c'est-à-dire une configuration où chaque nœud peut se voir attribuer une valeur arbitraire en chacune de ses variables), converge vers une configuration qui satisfait Π. Il est connu que le problème de l'élection n'admet pas d'algorithme déterministe auto-stabilisant utilisant une mémoire constante en bits à chaque nœud, autrement dit, pour certains réseaux, la taille de la mémoire par noeud doit croître en même temps que la taille du réseau. D'autre part, il existe un algorithme déterministe auto-stabilisant pour le problème de l'élection dans un anneau dont la taille mémoire par noeud est $O(\log\log n)$ bits, où n est la taille du réseau. Nous montrons dans cette article que cette dernière complexité spatiale est optimale. Plus précisément, nous prouvons que tout algorithme déterministe auto-stabilisant résolvant le problème de l'élection doit utiliser une taille mémoire de $\Omega(\log\log n)$ bits par nœud. Nous montrons également que notre borne inférieure s'applique au-delà du problème de l'élection à tous les problèmes qui ne peuvent être résolus par des algorithmes anonymes.

1 Introduction

L'auto-stabilisation est un paradigme adapté aux réseaux distribués asynchrones sujets aux erreurs transitoires. De telles erreurs peuvent placer le système dans une configuration arbitraire. Un algorithme est auto-stabilisant s'il garantit que, partant d'une configuration non-conforme à la spécification du problème le système atteindra en temps fini une configuration conforme, et restera dans des configurations conformes en l'absence de faute.

Durant l'exécution d'un algorithme auto-stabilisant, les nœuds voisins échangent de l'information entre eux, cette information est stockée localement dans chaque nœud. Plus précisément, les nœuds ont deux types de mémoire : la mémoire persistante et la mémoire modifiable. La mémoire persistante est utilisée pour stocker l'identifiant des nœuds (l'adresse MAC par exemple), ses numéros de port, ou encore le code de l'algorithme. Cette mémoire n'est pas modifiable lors de l'exécution de l'algorithme, ce qui la rend moins sensible à la corruption. La plupart des travaux relatifs à l'auto-stabilisation font ainsi l'hypothèse que la mémoire persistante n'est pas corruptible. La mémoire modifiable est utilisée pour stocker les variables utilisées par l'algorithme, et est susceptible d'être corrompue. La complexité spatiale d'un algorithme auto-stabilisant est la taille de la mémoire modifiable. Une des questions fondamentales est la détermination de bornes pour la complexité spatiale des problèmes classiques de la littérature, et notamment le problème de l'élection.

La minimisation de la complexité spatiale présente de nombreux intérêts. Premièrement, il est souhaitable que les algorithmes auto-stabilisants offrent une certaine forme d'universalité, autrement dit qu'ils

† Support by ANR ESTATE (ANR-16-CE25-0009-03) and ANR GrR (ANR-18-CE40-0032)

soient exécutables quelles que soient les contraintes réseau. Par exemple, les réseaux de capteurs de l'IoT, ainsi que les essaims de robots, reposent sur des entités de calcul à capacité de mémoire limitée, ce qui exclut l'utilisation d'algorithmes distribués nécessitant une mémoire importante. Deuxièmement, limiter la complexité spatiale revient à réduire la bande passante, réduisant ainsi l'overhead dû à la congestion des liens [1]. Pour détecter des fautes, les nœuds d'un système doivent échanger de l'information en permanence. Ainsi en réduisant la taille de l'information échangée on augmente l'autonomie des systèmes à faible capacité énergétique. Enfin, la réplication des données [10] est une solution pour rendre les systèmes robustes. Cette réplication est d'autant plus réalisable quand la taille de la mémoire est faible.

2 Contributions

Nous nous concentrons dans cet article sur l'un des problèmes fondamentaux du calcul distribué, le problème de l'*élection*, c'est-à-dire le maintien d'un unique nœud distingué dans le réseau. Un algorithme auto-stabilisant qui résout le problème de l'élection doit converger vers une configuration avec un unique nœud distingué et maintenir l'unicité du nœud distingué dans le temps.

Il est remarquable de noter que la spécification de l'élection peut s'encoder sur un unique bit de mémoire, en implémentant une variable booléenne en chaque nœud, qui vaut 1 si le nœud est distingué, 0 sinon. Ainsi, la complexité d'un algorithme d'élection auto-stabilisant est entièrement due à la complexité de résolution du problème et non à sa spécification.

Nous établissons une borne inférieure en complexité de $\Omega(\log\log n)$ bits par nœud pour le problème de l'élection. Ceci améliore l'unique borne inférieure connue jusqu'à present (voir [2]), selon laquelle l'élection ne peut être résolue en espace mémoire constant. De plus, sur les réseaux de degré borné notre borne inférieure coïncide avec la meilleure complexité connue de $O(\log\log n)$ bits par nœud [6] .

Nous obtenons notre borne inférieure en établissant une équivalence entre les algorithmes auto-stabilisants à faible mémoire et les algorithmes auto-stabilisants pour réseaux anonymes, c'est-à-dire les réseaux dans lesquels les nœuds n'ont pas d'identifiant. Il est connu que de nombreux problèmes d'auto-stabilisation, comme la coloration, l'élection, la construction d'un arbre couvrant... ne sont pas résolubles dans le cadre anonyme, car les identifiants sont nécessaires pour briser les symétries. Nous montrons que n'importe quel algorithme pour réseau avec identifiant, s'il utilise trop peu de mémoire, n'a pas plus de puissance que les algorithmes pour réseau anonyme. Plus précisément, soit A un algorithme pour réseau avec identifiant, et supposons que A a une complexité en espace en $o(\log\log n)$ bits par nœud. Une telle complexité permet aux nœuds d'échanger leur identifiants par morceaux, mais ne leur permet pas de les stocker en entier dans leur mémoire modifiable. Nous montrons que, avec cette complexité spatiale en $o(\log\log n)$ bits par nœud, il existe des graphes et des affectations d'identifiants aux nœuds du graphe tels que, pour ces réseaux, A se comporte exactement comme le ferait un algorithme A' sur les mêmes graphes privés d'identifiant sur les nœuds. Puisque l'élection ne peut pas être résolue dans les réseaux sans identifiant, nous pouvons en conclure que A ne résout pas l'élection dès lors que sa complexité spatiale est en $o(\log\log n)$ bits par nœud. Une version longue de cet article a été publiée [3].

3 Etat de l'art

La complexité spatiale des algorithmes auto-stabilisants a été largement étudiée pour les algorithmes silencieux, c'est-à-dire les algorithmes dans lesquels plus aucun nœud ne modifie sa mémoire après convergence vers une configuration légitime. Dolev et al. [8] montrent que trouver les centres d'un graphe, la construction d'arbre, l'élection, nécessitent des registres en $\Omega(\log n)$ bits par nœud pour les algorithmes silencieux. Le concept des *proof-labeling scheme* (PLS) [12], développé indépendamment, a été utilisé pour étudier la complexité des algorithmes silencieux [4].

Un exemple typique est la borne inférieure de $\Omega(\log^2 n)$ bits par nœud pour la construction d'un arbre couvrant de poids minimal, qui provient d'une borne identique dans la théorie des PLS [11], [4]. Le lien étroit entre algorithmes auto-stabilisants silencieux et PLS a permis de connaître précisément la complexité d'un grand nombre de problèmes (voir [9] pour plus de détails sur les PLS). D'autre part, il n'existe à notre connaissance qu'une seule borne inférieure pour la complexité spatiale dans le cadre général des

algorithmes auto-stabilisants (plus forcément silencieux), qui établit que l'élection ne peut être résolue avec des registres de taille constante [2].

En ce qui concerne les bornes supérieures la littérature est bien plus riche. En particulier, [5] propose un algorithme d'élection auto-stabilisant qui utilise une taille mémoire de $O(\log\log n)$ bits par nœud dans l'anneau à n nœuds. Cet algorithme a été ensuite généralisé aux réseaux de degré Δ, avec une mémoire de taille $O(\log\log n+\log\Delta)$ bits par nœud [6]. Il est intéressant de remarquer que la construction d'un arbre couvrant et la $(\Delta+1)$-coloration sont également résolues avec la même complexité $O(\log\log n+\log\Delta)$ bits par nœud dans [6]. Avant ces travaux, la meilleure borne supérieure connue était $O(\log n)$ bits par nœud [5], et il a été conjecturé que la technique utilisée pour passer de $O(\log n)$ bits par nœud à $O(\log\log n)$ bits par nœud pourrait être itérée, pour finalement atteindre $O(\log^* n)$ bits par nœud. Nous réfutons cette conjecture et prouvons que la complexité de l'élection est en $\Omega(\log\log n)$ bits par nœud.

4 Modèle

Nous utilisons dans cet article un modèle largement utilisé dans la littérature, le modèle à états [7]. Le réseau est modélisé par un graphe $G=(V,E)$ à n nœuds, chaque nœud possède des variables locales, lisibles par ses voisins, mais sur lesquelles il est le seul à pouvoir écrire. Les nœuds possèdent également un identifiant globalement unique, noté $ID(v)\in\{1,\ldots,n^c\}$ ($c>1$ est une constante), qui n'est pas modifiable ni corruptible, et non lisible par les voisins du nœud. Seule la mémoire lisible (et corruptible) est considérée dans le calcul de la complexité spatiale d'un algorithme. Un algorithme peut faire appel à l'identifiant des nœuds dans les règles qui le définissent, mais il ne maîtrise pas la définition des identifiants sur le réseau. Il doit donc fonctionner indépendamment de la distribution des identifiants. Si c'est le cas, alors l'algorithme est dit *à identifiants*, sinon il est dit *anonyme*.

5 Résultats

Nous établissons dans la suite un théorème principal, qui a pour corollaire la borne en mémoire de $\Omega(\log\log n)$ pour l'élection.

Théorème 1 (Équivalence faible mémoire-anonyme)**.** *Soit $c>1$, et soit $f_\Delta(n)\in o(\log n)$ une fonction. S'il existe un algorithme déterministe auto-stabilisant A qui résout un problème P en utilisant une mémoire de $o(\frac{\log\log n}{\Delta})$, alors il existe un algorithme déterministe auto-stabilisant anonyme A' qui résout P sur les réseaux suffisamment grands de degré maximal $f_{\Delta(n)}$.*

Corollaire 1 (Borne en mémoire pour l'élection)**.** *Soit $c>1$. Un algorithme déterministe auto-stabilisant pour l'élection sous un ordonnanceur équitable requiert $\Omega(\log\log n)$ bits par nœud pour les réseaux à n nœuds dont les identifiants sont éléments de $[1,n^c]$*

6 Intuition de la preuve

Comme nous l'avons vu, l'étude de bornes en auto-stabilisation s'est principalement concentrée sur les algorithmes silencieux. Dans le cadre général des algorithmes auto-stabilisants, il n'y a pas d'approche naturelle comme les PLS qui permet d'obtenir des bornes inférieures en mémoire. Pour obtenir notre borne, nous montrons que les algorithmes utilisant une mémoire en $o(\log\log n)$ bits par nœud ne sont pas plus puissants que les algorithmes anonymes, ce qui nous permet d'utiliser tous les résultats d'impossibilités connus dans le domaine anonyme.

Un nœud qui exécute un algorithme avec identifiant peut utiliser son identifiant dans son code. Par exemple, une des règles de l'algorithme peut être :

— Si mon voisin de gauche v_g respecte la propriété P_1 et si moi même je respecte la propriété P_2, alors, si mon identifiant est pair je modifie ma variable v_1 pour lui attribuer a, sinon je lui attribue b.

Pour un identifiant fixé, on peut réécrire cette règle en utilisant directement la valeur de l'identifiant. Si l'identifiant est par exemple 7, un nombre impair, la règle devient :

— Si mon voisin de gauche v_g respecte la propriété P_1 et si moi même je respecte la propriété P_2, alors je modifie ma variable v_1 pour lui attribuer b.

Cette transformation peut être faite pour toutes les règles de l'algorithme, et pour chaque identifiant. On peut ainsi construire un algorithme A_i spécifique à chaque identifiant i, de sorte que chaque algorithme A_i ne fait plus du tout référence à l'identifiant du nœud qui l'exécute.

L'observation clef est la suivante : si la taille de la mémoire qu'utilise un algorithme est très petite, alors la variété de comportement différents que peuvent avoir les nœuds du réseau est également très faible. Illustrons ceci avec un exemple, et considérons une topologie en anneau, sur laquelle l'algorithme considéré n'utilise qu'un bit en mémoire. Pour calculer son nouvel état partant d'une configuration donnée, chaque nœud v applique une fonction de transition qui prend en entrée les états de v et de ses deux voisins v^g et v^d. Dans notre cas, ces entrées sont l'ensemble des triplés (v_x, v_x^g, v_x^d) où $v_x, v_x^g, v_x^d \in \{0,1\}$. Il y a donc $2^3 = 8$ entrées différentes. À chacune de ses entrées, un algorithme associe une nouvelle valeur pour le bit du nœud qui exécute. Il existe $2^8 = 256$ différentes manières d'associer un bit à chaque triplé (v_x, v_x^g, v_x^d), donc autant de manières possibles pour un algorithme de se comporter sur de tels réseaux. Chaque algorithme A_i correspond à un de ces 256 comportements différents. Cela implique que si l'on considère un anneau de 257 nœuds, il existe nécessairement deux nœuds d'identifiants $i \neq j$ tels que les algorithmes A_i et A_j sont égaux.

Ceci n'est cependant pas suffisant pour valider notre théorème, puisque pour que l'algorithme soit anonyme, il faudrait que tous les nœuds exécutent le même algorithme A'. Il est possible de pousser plus loin le raisonnement pour obtenir le résultat souhaité. Si les identifiants sont choisis dans un espace polynomial en n, et si la mémoire reste suffisamment petite, c'est-à-dire en $o(\log\log n)$, alors pour n suffisamment grand, il existe toujours au moins n fois plus d'identifiants qu'il n'existe de comportements différents. Il est donc toujours possible de trouver n identifiants différents qui partagent le même algorithme A_i. Dans le réseau à n nœuds auquels on a affecté ces n identifiants, notre algorithme anonyme $A_i = A'$ se comporte exactement comme A.

Références

[1] J. Adamek, M. Nesterenko, and S. Tixeuil. Evaluating practical tolerance properties of stabilizing programs through simulation : The case of propagation of information with feedback. In *SSS 2012*, pages 126–132, 2012.

[2] J. Beauquier, M. Gradinariu, and C. Johnen. Memory space requirements for self-stabilizing leader election protocols. In *PODC 1999*, pages 199–207, 1999.

[3] L. Blin, L. Feuilloley, and G. L. Bouder. Optimal space lower bound for deterministic self-stabilizing leader election algorithms. In *OPODIS 2021*, volume 217, pages 24 :1–24 :12, 2021.

[4] L. Blin and P. Fraigniaud. Space-optimal time-efficient silent self-stabilizing constructions of constrained spanning trees. In *ICDCS 2015*, pages 589–598, 2015.

[5] L. Blin and S. Tixeuil. Compact deterministic self-stabilizing leader election on a ring : the exponential advantage of being talkative. *Distributed Computing*, 31(2) :139–166, 2018.

[6] L. Blin and S. Tixeuil. Compact self-stabilizing leader election for general networks. In *LATIN 2018*, pages 161–173, 2018.

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

[8] S. Dolev, M. G. Gouda, and M. Schneider. Memory requirements for silent stabilization. *Acta Inf.*, 36(6) :447–462, 1999.

[9] L. Feuilloley and P. Fraigniaud. Survey of distributed decision. *Bulletin of the EATCS*, 119, 2016. url : bulletin.eatcs.org link, arXiv : 1606.04434.

[10] T. Herman and S. V. Pemmaraju. Error-detecting codes and fault-containing self-stabilization. *Inf. Process. Lett.*, 73(1-2) :41–46, 2000.

[11] A. Korman and S. Kutten. Distributed verification of minimum spanning trees. *Distributed Computing*, 20(4) :253–266, 2007.

Borne inférieure optimale pour la complexité spatiale des algorithmes déterministes auto-stabilisants d'élection.

[12] A. Korman, S. Kutten, and D. Peleg. Proof labeling schemes. *Distributed Computing*, 22(4) :215–233, 2010.