Détection de terminaison silencieuse, anonyme, stabilisante instantanément

Lélia Blin, Colette Johnen, Gabriel Le Bouder, Franck Petit

Détection de terminaison silencieuse, anonyme, stabilisante instantanément

Lélia Blin[1,2] and Colette Johnen[4] and Gabriel Le Bouder[1,3] and Franck Petit [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. Bordeaux, CNRS, LaBRI UMR 5800, 351 cours de la libération, 33405 Talence, France.*

Nous abordons le problème de la détection de terminaison (TD) dans les réseaux asynchrones. Un algorithme qui résout ce problème détecte si un algorithme distribué observé A a convergé vers une configuration qui satisfait un prédicat particulier. Il est connu que TD ne peut pas être résolu dans le contexte de l'auto-stabilisation, sauf si l'algorithme est instantanément stabilisant (snap-stabilization), c'est-à-dire qu'il se comporte toujours conformément à sa spécification, indépendamment de la configuration initiale. Un algorithme déterministe instantanément stabilisant pour TD est donné dans [7] pour les réseaux enracinés, autrement dit les réseaux où tous les nœuds exécutent le même algorithme, sauf un. Dans cet article, nous proposons un algorithme générique, déterministe, silencieux, instantanément stabilisant, qui détecte la terminaison d'un algorithme observé $\mathcal{A}$, auto-stabilisant, silencieux et terminant. Notre algorithme ne nécessite pas de structure sous-jacente, ni de topologie spécifique et fonctionne dans des réseaux anonymes. Plus précisément notre algorithme n'utilise aucune hypothèse permettant de distinguer un ou plusieurs processus, de plus, il fonctionne sous les hypothèses de l'ordonnanceur le plus faible, l'ordonnanceur inéquitable. Construite au-dessus de n'importe quel unisson asynchrone auto-stabilisant sous-jacent $\mathcal{U}$, notre solution requiert comme espace mémoire supplémentaire uniquement $O(\log D)$ bits par nœud, où D est le diamètre du réseau. Puisqu'il n'existe aucun algorithme d'unisson avec une meilleure complexité spatiale, l'espace mémoire supplémentaire de notre solution est comparable à la complexité spatiale de l'algorithme d'unisson sous-jacent. Notre algorithme fournit une réponse positive en $O(\max(k,k',D))$ rounds, où k et k' sont les complexités respectives en temps de stabilisation de $\mathcal{A}$ et $\mathcal{U}$.

1 Introduction

La *Détection de terminaison* [13] est un problème fondamental des systèmes distribués. Il s'agit d'un problème d'observation globale du réseau, de même que la capture d'instantané ou la détection d'interblocage. Ces problèmes d'observation globale sont par nature difficiles à résoudre sur les systèmes distribués où chaque nœud n'a qu'une vue partielle du réseau. Pour le problème de la détection de terminaison (TD), il s'agit de fournir à chaque nœud la possibilité de détecter si le système a atteint une configuration désirée. Plus précisément, les nœuds émettent localement une requête qui va déclencher la procédure de TD, qui informera ensuite le nœud quand le système aura atteint une configuration désirée.

Nous nous intéressons dans cet article aux algorithmes *auto-stabilisants* [1, 10, 11]. Un tel algorithme garantit que, quelle que soit la configuration initiale du système, l'exécution de l'algorithme convergera en temps fini dans un ensemble de configurations qui satisfont la spécification du problème traité par l'algorithme. L'auto-stabilisation est adaptée aux systèmes susceptibles de subir des fautes transitoires, c'est-à-dire des fautes imprévisibles et rares qui modifient le contenu des variables des nœuds. Contrairement à d'autres méthodes pour développer la robustesse, l'auto-stabilisation ne vise pas à empêcher les erreurs de se produire, mais les répare. Cela signifie que l'auto-stabilisation correspond aux systèmes qui tolèrent des écarts temporaires à leur spécification. Une fois que les erreurs transitoires cessent, il faut un certain temps, appelé le *temps de stabilisation* avant que le système atteigne une configuration correcte.

Puisque l'auto-stabilisation n'empêche pas les fautes de se produire, il est impossible pour un nœud de décider si le système a globalement convergé ou non, à l'exception de certains problèmes triviaux. En effet, un nœud peut avoir exactement la même vue locale dans deux situations où le système a globalement convergé, et où le système n'a pas globalement convergé. Cependant, un algorithme auto-stablisant garantit qu'en répétant un tel mécanisme de détection, le système donnera ultimement des réponses correctes, mais au bout d'un temps non déterminé qui dépend notamment du temps de stabilisation de l'algorithme observé.

Dans [7], il est montré que la détection de terminaison peut être résolue en n'utilisant qu'une seule exécution de l'algorithme TD. Autrement dit, dès la première requête, la réponse de l'algorithme de détection de terminaison est correcte, même si cette requête a été émise pendant la phase de stabilisation de l'algorithme observé. Un tel algorithme de détection de terminaison est dit *instantanément stabilisant* [6]. La stabilisation instantanée est une propriété plus forte que celle d'auto-stabilisation, puisqu'après des fautes transitoires, un système instantanément stabilisant produit immédiatement une exécution correcte, sans besoin d'intervention extérieure. Il s'agit en fait des algorithmes auto-stabilisants dont le temps de stabilisation est nul. Il est important de remarquer que les algorithmes instantanément stablisants, tout comme les algorithmes auto-stabilisants, n'empêchent pas les erreurs transitoires de se produire ni d'avoir des effets qui se propagent dans le réseau. Ils garantissent cependant que le comportement de l'algorithme respectera la spécification du problème immédiatement après la dernière faute.

Remarquons que les solutions instantanément stabilisantes de [7] font l'hypothèse de systèmes distribués *avec identifiants* uniques. Bien que la plupart des systèmes distribués mettent en jeu des entités avec identifiants, développer des algorithmes qui n'utilisent pas les identifiants des nœuds présente plusieurs intérêts. De tels algorithmes sont dits *anonymes*.

L'anonymat du réseau rend souvent plus complexe, voire impossible, la résolution d'un problème, ce qui rend cette approche intéressante du point de vue de vue de l'étude de la calculabilité. L'élection est un cas emblématique de cet effet de seuil. En effet, ce problème est relativement simple à résoudre de façon déterministe en utilisant l'ordre total que fournissent les identifiants. À contratio, si l'on ne dispose pas de moyens tels que les identifiants pour briser les symétrie du réseau, alors le problème de l'élection ne peut être résolu. De plus, les algorithmes anonymes requierent à priori moins de mémoire. Ils n'ont notamment pas besoin de stocker ni d'échanger les $O(\log n)$ bits de mémoire des identifiants des nœuds. L'approche par algorithmes anonymes est également intéressante d'un point de vue pratique. En effet, elle propose des solutions qui préservent l'anonymat des utilisateurs, fonctionne pour les systèmes dans lesquels existent des homonymes, ou pour les systèmes dans lesquels les identifiants des nœuds peuvent changer dans le temps.

2 État de l'art

Le problème de la détection de la stabilité des algorithmes auto-stabilisants dans des réseaux anonymes a été traité pour la première fois dans [14]. Dans cet article, les auteurs introduisent la notion d'observateur, c'est-à-dire un nœud qui peut détecter la correction d'un algorithme donné, mais ne peut pas l'influencer. Si l'on considère un ordonnanceur central, où les nœuds sont activés un par un, et un anneau uniforme à n nœuds, avec n premier, alors les auteurs proposent un algorithme déterministe pour l'observateur qui détecte la stabilité en $\Theta(n^2)$ pas de calcul après la stabilisation. Cependant, dans cet article, l'algorithme de l'observateur n'est pas sujet aux fautes transitoires, il n'est donc pas auto-stabilisant. Dans [2], les auteurs proposent également un algorithme non auto-stabilisant pour l'observateur dans le cadre des réseaux synchrones avec racine, et en introduisant de l'aléatoire, ils retirent la contrainte d'avoir un nœud distingué [3].

Le premier algorithme déterministe auto-stabilisant qui résout ce problème est proposé dans [7]. Les auteurs montrent que l'auto-stabilisation n'est pas la propriété adaptée pour résoudre ce problème. Par contre la stabilisation instantanée peut exprimer la propriété qui nous intéresse qui est de garantir que notre algorithme ne renvoit pas de réponse erronée y compris durant la phase de stabilisation. Remarquons que [7] fonctionne dans le cadre des réseaux avec identifiants.

Dans les réseaux anonymes de taille quelconque, l'*unisson* [5, 8, 9, 12] fournit un support pratique pour construire des solutions déterministes [4]. L'unisson asynchrone consiste en le maintient d'une horloge logique en chaque nœud, telle que : (i) chaque horloge diffère d'au plus 1 avec les horloges de ses voisins, et (ii) l'horloge de chaque nœud est infiniment souvent incrémentée de 1.

3 Contribution

Cet article se place dans le cadre des réseaux anonymes asynchrones. Nous nous intéressons à des algorithmes auto-stabilisants *terminant et silencieux*, c'est-à-dire des algorithmes auto-stabilisants qui convergent en temps fini vers une configuration désirée, configuration depuis laquelle plus aucune variable n'est ensuite modifiée. Nous introduisons le premier algorithme déterministe silencieux, générique, qui détecte si un algorithme auto-stabilisant terminant et silencieux observé a convergé vers une configuration désirée. Notre solution utilise des outils similaires à ceux de [4] pour obtenir la propriété de stabilisation instantanée, notamment l'utilisation d'un algorithme d'unisson sous-jacent. Notre contribution peut utiliser n'importe quel algorithme d'unisson asynchrone trouvable dans la littérature, comme [5, 8, 9, 12]. Contrairement à la solution de [7], nous ne faisons pas l'hypothèse qu'il existe un nœud distingué ni d'identifiants. Notre algorithme est effectif sous l'hypothèse de l'ordonnanceur le plus fort, l'ordonnanceur inéquitable. Le coût en mémoire des variables propres de notre algorithme est en $O(\log D)$ bits par nœud, où D est le diamètre du réseau. À ce coût il convient d'ajouter la complexité en mémoire de l'algorithme d'unisson utilisé. À ce jour, à notre connaissance, la meilleure complexité obtenue pour l'unisson asynchrone est due à [12] et est $O(\log D)$ bits par nœuds, ce qui signifie que notre solution n'ajoute pas de coût supplémentaire par rapport à celle de l'algorithme d'unisson que nous utilisons. Le temps de réponse est le nombre de rondes entre le moment où une requête est initiée en un nœud, et le moment où la réponse est rendue en ce même nœud. Le temps de réponse de notre solution est en $O(\max(k,k',D))$ rondes, où k et k' sont respectivement les temps de stabilisation de l'algorithme observé, et de l'algorithme d'unisson utilisé. Autrement dit, une fois que ces deux algorithmes ont stabilisé, notre algorithme fournit des réponses aux requêtes en temps optimal, $O(D)$ rondes.

4 Algorithme

Nous considérons un algorithme auto-stabilisant silencieux A qui résout un problème P. Nous introduisons un mécanisme générique qui permet de contruire un algorithme T, algorithme qui résout la détection instantanément stabilisante de la terminaison de A, en utilisant un algorithme d'unisson U.

Notre algorithme T réalise une simulation parallèle et indépendante des algorithmes A et U en chaque nœud du réseau. Les variables de T incluent donc les variables de A et celles de U. À chaque fois qu'un nœud v est activé pour T, il exécute la règle de A pour laquelle il est activable, si elle existe, et la règle de U pour laquelle il est activable, si elle existe. En parallèle, l'algorithme T va mettre à jour les variables qui lui sont spécifiques, à savoir deux compteurs doi et cnt, prenant des valeurs entre 0 et $2D+2$ pour doi, et entre 0 et $2D+1$ pour cnt.

La variable doi_v est mise à jour à sa valeur maximale $2D+2$ à chaque fois que le nœud v effectue une action de convergence pour A ou U (l'algorithme d'unisson est auto-stabilisant, il peut donc être toujours en train de converger pendant la première partie de l'exécution de T). Ainsi, les nœuds pour qui l'algorithme n'a pas localement convergé voient leur variable doi maintenue à son maximum. Lorsqu'un nœud est activé, il peut ne pas effectuer d'action de convergence, s'il effectue uniquement une action d'incrémentation de sa variable d'horloge. Dans ce cas là, il modifie également sa variable doi en lui faisant prendre comme valeur 1 de moins que la plus grande valeur de doi des nœuds dans son voisinage, lui compris. Ainsi, lorsque plus aucun nœud ne fait de pas de calcul pour A, les variables doi de tous les nœuds du système vont décroître jusqu'à atteindre 0. De plus, et c'est la propriété essentielle de notre algorithme, si un nœud effectue une règle de convergence, à la première activation de ses voisins, ces voisins prendront une valeur pour doi proche du maximum, puis lorsque les voisins des voisins seront activés, ils prendront également une valeur proche de $2D$, et ainsi de suite jusqu'à atteindre tout le réseau.

Puisque les calculs de T sont synchronisés avec l'algorithme d'unisson, et grâce aux propriétés des algorithmes de cette famille, nous pouvons garantir qu'un nœud (ou qu'un sous-ensemble de nœud du réseau) ne peut pas effectuer des actions de convergences et faire ensuite décroître sa variable doi sans que ses voisins soient également activés. Ainsi, si un nœud met sa variable doi à son maximum, ceci aura pour effet que tous les nœuds du réseau (et notamment les éventuels nœuds qui ont émis une requête de détection de terminaison) auront, dans la suite de l'exécution, leur variable doi à une valeur supérieure à 0.

Losqu'un nœud v émet une requête de détection de terminaison, il modifie sa variable cnt_v à sa valeur maximale $2D+1$. Puis, à chaque fois que v est activé, il fait décroître de 1 la valeur de sa variable cnt_v, indépendamment de ce qu'il voit dans son voisinage. Cependant, si la variable doi_v vaut autre chose que 0, alors v réinitialise sa variable cnt_v à sa valeur maximale $2D+1$. Ainsi, la variable cnt_v ne peut atteindre 0 qu'après $2D+1$ pas de calculs de v durant lesquels il ne détecte pas d'activité dans le réseau via doi. Les propriétés de l'unisson nous permettent de garantir qu'alors aucun nœud n'a été activé, et donc également que l'algorithme A a terminé. Ceci signifie que l'algorithme T est un algorithme instantanément stabilisant pour la détection de terminaison de A.

Références

[1] K. Altisen, S. Devismes, S. Dubois, and F. Petit, editors. *Introduction to Distributed Self-Stabilizing Algorithms*. Synthesis Lectures on Distributed Computing. Morgan & Claypool Publishers, 2019.

[2] J. Beauquier, L. Pilard, and B. Rozoy. Observing locally self-stabilization. *J. High Speed Networks*, 14(1) :3–19, 2005.

[3] J. Beauquier, L. Pilard, and B. Rozoy. Observing locally self-stabilization in a probabilistic way. *J. Aerosp. Comput. Inf. Commun.*, 3(10) :516–537, 2006.

[4] C. Boulinier, M. Levert, and F. Petit. Snap-stabilizing waves in anonymous networks. In *Distributed Computing and Networking, 9th International Conference, ICDCN 2008*, volume 4904 of *Lecture Notes in Computer Science*, pages 191–202. Springer, 2008.

[5] C. Boulinier, F. Petit, and V. Villain. When graph theory helps self-stabilization. In *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing, PODC 2004*, pages 150–159. ACM, 2004.

[6] A. Bui, A. K. Datta, F. Petit, and V. Villain. Snap-stabilization and PIF in tree networks. *Distributed Computing*, 20(1) :3–19, 2007.

[7] A. Cournier, A. K. Datta, S. Devismes, F. Petit, and V. Villain. The expressive power of snap-stabilization. *Theor. Comput. Sci.*, 626 :40–66, 2016.

[8] J. Couvreur, N. Francez, and M. Gouda. Asynchronous unison. In *Proceedings of the 12th IEEE International Conference on Distributed Computing Systems (ICDCS'92)*, pages 486–493, 1992.

[9] S. Devismes and C. Johnen. Self-stabilizing distributed cooperative reset. In *39th IEEE International Conference on Distributed Computing Systems, ICDCS 2019*, pages 379–389. IEEE, 2019.

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

[11] S. Dolev. *Self-Stabilization*. MIT Press, 2000.

[12] Y. Emek and E. Keren. A thin self-stabilizing asynchronous unison algorithm with applications to fault tolerant biological networks. In *PODC '21 : ACM Symposium on Principles of Distributed Computing*, pages 93–102. ACM, 2021.

[13] N. Francez. Distributed termination. *ACM Trans. Program. Lang. Syst.*, 2(1) :42–55, 1980.

[14] C. Lin and J. Simon. Observing self-stabilization. In N. C. Hutchinson, editor, *Proceedings of the Eleventh Annual ACM Symposium on Principles of Distributed Computing, Vancouver, British Columbia, Canada, August 10-12, 1992*, pages 113–123. ACM, 1992.