Le premier algorithme stabilisant de construction d'arbre totalement polynomial

Alain Cournier, Stephane Rovedakis, Vincent Villain

Le premier algorithme stabilisant de construction d'arbre totalement polynomial †

Alain Cournier[1] and Stéphane Rovedakis[2] and Vincent Villain[1]

[1]*Laboratoire MIS, Université de Picardie, 33 Rue St Leu, 800 39 Amiens Cedex 1, France.*
[2]*Laboratoire CEDRIC, CNAM, 292 Rue St Martin, 75141 Paris Cedex 03, France.*

Un algorithme stabilisant A est dit *totalement polynomial* s'il existe deux constantes a et b tel que pour tout réseau $\mathcal{R}$ (de diamètre d comportant n noeuds), la complexité de notre algorithme A appartient à $O(d^a)$ rondes et $O(n^b)$ pas de calcul. La construction d'un arbre couvrant est un problème fondamental des systèmes distribués. L'utilisation d'un arbre couvrant permet de résoudre plus efficacement d'autres problèmes comme le routage, l'exclusion mutuelle ou la réinitialisation d'un réseau. La stabilisation est une technique générale et flexible permettant de traiter les fautes transitoires qui peuvent survenir dans un réseau. Or les solutions existantes résolvent ce problème soit avec un nombre de pas de calcul exponentiel soit avec un nombre de rondes fonction de la taille du réseau. Ces caractéristiques les rendent inappropriées dans le cadre du passage à l'échelle. Nous proposons le premier algorithme stabilisant totalement polynomial permettant de construire un arbre couvrant en largeur ayant une complexité en rondes de $O(d^2)$ et en pas de calcul de $O(n^6)$. Le diamètre est généralement inférieur à la taille du réseau ($\log(n)$ en moyenne). Ainsi, cet algorithme atteint le meilleur compromis entre les complexités en rondes et en pas de calcul.

Keywords: Algorithme distribué, Tolérance aux fautes, Stabilisation, Construction d'arbre couvrant.

1 Introduction

La construction d'un arbre couvrant est un problème fondamental dans les systèmes distribués. Un arbre couvrant est une structure virtuelle sans cycles, connectant tous les noeuds du réseau. En systèmes distribués, cette structure est habituellement utilisée comme un prérequis par certains algorithmes de routage, de circulation de jeton ou de diffusion de messages dans un réseau. L'utilisation d'arbres couvrants permet aussi de résoudre des problèmes distribués de façon plus efficace, c'est-à-dire avec une meilleure complexité en temps. Il est donc crucial de concevoir des solutions performantes pour la construction d'arbres couvrants. Dans cet article nous allons étudier un type particulier d'arbres couvrants largement utilisé dans les réseaux afin de diffuser rapidement des informations à partir d'une source. C'est l'arbre couvrant en largeur, minimisant le nombre de noeuds intermédiaires (sauts) entre tout noeud et la source.

L'auto-stabilisation est une technique introduite par Dijkstra [Dij74], pour tolérer les fautes transitoires dans les systèmes distribués. Un algorithme distribué est dit *auto-stabilisant* si partant d'un état global arbitraire du système, le système retrouve en un temps fini un comportement (état global) correct sans intervention extérieure. Comme aucune hypothèse n'est faite sur la nature des fautes, cette technique tolère aussi les changements dynamiques de topologie du réseau car ces modifications peuvent être vues comme des fautes par le système. Un nouveau concept de stabilisation a été introduit par Bui *et al* [BDPV99], appelé *stabilisation instantanée*. Les algorithmes instantanément stabilisants ont la capacité de toujours garantir un comportement du système en conformité avec les spécifications du problème à résoudre.

Pour mesurer l'efficacité des algorithmes distribués diverses mesures de complexité en temps ont été introduites, dans cet article, nous regarderons les complexités en nombre de rondes et en nombre de pas de calcul. Bien que ces deux mesures soient complémentaires, il est très rare qu'un algorithme soit évalué selon ces deux critères.

†Les résultats présentés dans cet article sont issu de [CRV11], ce travail a été supporté par le projet ANR SPADES.

Travaux précédents. De nombreux travaux traitent du problème de la construction stabilisante d'arbre couvrant. L'approche stabilisante avec la complexité en temps la plus faible en terme de rondes suit l'approche classique en algorithmique distribuée basée sur les distances à la racine et utilisée pour construire un arbre couvrant en largeur. Ainsi, la source (ou racine) possède une distance nulle et les autres processeurs du réseau cherchent à obtenir la valeur la plus faible en cohérence avec les valeurs présentes dans leur voisinage. Cette approche a été utilisée par exemple dans [HC92] pour construire un arbre couvrant en largeur de manière stabilisante en $O(d)$ rondes. Bien que la complexité en rondes capture le taux d'exécution du processeur le plus lent dans toute exécution pour un algorithme distribué, un autre critère est important pour établir la complexité en temps nécessaire au calcul de la solution, le nombre de *pas de calcul*. En effet, ce critère reflète la quantité d'informations échangées par un algorithme, spécialement dans le cas des algorithmes stabilisants pour lesquels chaque processeur doit informer ses voisins lorsque son état a changé. Quelques travaux analysent la complexité en pas de calcul, par exemple [KK05, CDV09, Cou09], mais la majorité des solutions proposées étudient uniquement la complexité en rondes. Cependant, nous avons observé que les solutions existantes avec une complexité en rondes proportionnelle au diamètre ont un nombre de pas de calcul exponentiel. Réciproquement, les solutions donnant de bons résultats en nombre de pas de calcul ont une complexité en nombre de rondes liée au nombre de noeuds (au lieu du diamètre). Or, lors du passage à l'échelle, les réseaux, constitués d'un grand nombre de processeurs, conservent en moyenne un diamètre très faible. Il est donc irréaliste de considérer l'utilisation d'algorithmes ayant une complexité en rondes fonction de la taille du réseau, et encore moins avec une complexité en pas de calcul exponentielle.

Une question légitime est donc la suivante : Est il possible de concevoir un algorithme stabilisant totalement polynomial pour le problème de la construction d'un arbre couvrant ?

Nos contributions. Nous introduisons la notion d'algorithme stabilisant *totalement polynomial* et montrons que la construction d'arbre couvrant admet de tels algorithmes. Nous proposons le premier algorithme construisant un arbre couvrant en largeur en $O(d^2)$ rondes et $O(n^6)$ pas de calcul, sous un démon inéquitable, où d et n sont le diamètre et la taille du réseau. Cet algorithme atteint le meilleur compromis entre complexités en rondes et en pas de calcul, démontrant ainsi qu'il existe des algorithmes stabilisants permettant de résoudre efficacement des problèmes sur des réseaux (indépendamment de leur topologie).

2 Modèle et notations

Notations. Le réseau est modélisé par un graphe $G = (V,E)$, où V est l'ensemble des noeuds ou processeurs ($n = |V|$) et E est l'ensemble des liens de communication *bidirectionnels* et *asynchrones* ($m = |E|$). Nous supposons qu'un processeur particulier est désigné comme *racine* dans le réseau (notée r). Les processeurs voisins du processeur p dans G sont notés $Neig_p$ et localement ordonnés par $\prec_p$. On note Δ le degré maximum d'un processeur dans le réseau, et d le diamètre. Un arbre couvrant $T = (V_T, E_T)$ de G est un sous-graphe acyclique et connexe de G, tel que $V_T = V$ et $E_T \subseteq E$.

Modèle. Nous considérons le modèle à états, où tout processeur $p \in V$ peut lire/écrire dans ses propres variables et uniquement lire la valeur des variables de ses voisins. Chaque processeur (sauf la racine) exécute le même programme composé de règles. Un processeur est dit *activable* si au moins une de ses règles est exécutable. L'*état local* d'un processeur est défini par la valeur de ses variables. Une *configuration* du système est le produit des états locaux de tous les processeurs du système. La transition (ou pas de calcul) d'une configuration à une autre est réalisée par l'exécution d'une action sur un ou plusieurs processeurs. Une *exécution* du système est une séquence de configurations, $e = (\gamma_0,\gamma_1,...,\gamma_i,\gamma_{i+1},...)$, tel que $\forall i \geq 0$, $\gamma_i \mapsto \gamma_{i+1}$ est un *pas de calcul* possible si γ_{i+1} existe, sinon γ_i est une configuration terminale. Un *démon* indique à chaque pas de calcul l'ensemble des processeurs exécutant leur programme. Nous faisons l'hypothèse d'un démon *distribué* et *inéquitable*, ainsi le démon peut empêcher un processeur d'exécuter son programme excepté si c'est le seul processeur activable. Le calcul de la complexité en temps est réalisé en considérant le nombre de pas de calcul et le nombre de *rondes*. Dans une exécution e, la première ronde est le préfixe minimal de e contenant une action de chaque processeur activable dans la configuration initiale. La ronde suivante commence sur le dernier état de la première ronde. S'agissant d'algorithmes stabilisants, nous considérons que la configuration initiale est quelconque.

3 Construction d'un arbre couvrant

Nous nous intéressons à la construction d'un arbre couvrant en largeur, c'est-à-dire un arbre $T = (V_T, E_T)$ de G tel que : (i) $V_T = V$ et $E_T \subseteq E$, (ii) T est connexe, et (iii) pour tout processeur p il n'existe pas de plus court chemin (en sauts) entre p et r dans G que celui dans T. Nous décrivons ici l'idée générale de l'algorithme construisant un arbre couvrant en largeur de manière stabilisante. Notre algorithme doit satisfaire les conditions suivantes : (a) l'algorithme doit atteindre une configuration terminale en un temps fini, et (b) toute configuration terminale vérifie la définition d'un arbre couvrant en largeur.

Notre algorithme $\mathcal{BFS}$ est en deux parties, la première réalise la connexion des processeurs constituant l'arbre, et la seconde autorise ou non les connexions de sorte à obtenir l'arbre désiré. La seconde partie est un mécanisme de questionnement vue comme un oracle par la première.

3.1 Algorithme de connexion

L'algorithme est basé sur le fait que chaque processeur doit se connecter à son voisin le plus proche de la racine r. Comme il n'y a qu'une seule racine dans le réseau, cette approche générale permet de rattacher tous les processeurs à l'arbre enraciné en r de façon distribuée et stabilisante en $O(d)$ rondes, mais pas forcément avec un nombre polynomial de pas de calcul. En effet, si aucune précaution n'est prise les processeurs peuvent se rattacher un nombre non polynomial de fois à des arbres *non valides* (non enraciné en r) présents dans la configuration initiale du système. Afin de borner le nombre de pas de calcul, les travaux suivants ont gelé les arbres non valides afin de limiter les reconnexions sur un même arbre non valide [KK05, Cou09]. Cependant, ces approches ont une complexité en rondes fonction de la taille du réseau. Notre algorithme $\mathcal{BFS}$ utilise donc, en plus d'un mécanisme de détection d'erreur destiné à détruire les arbres non valides, un mécanisme de questionnement (l'oracle) autorisant l'expansion de l'arbre enraciné en r, tout en limitant les mauvaises connexions.

Erreur. Une configuration quelconque contient toujours un arbre valide (arbre enraciné en r) et éventuellement des arbres non valides (arbres enracinés en une racine non valide, i.e., un processeur $p \neq r$). Tout processeur connecté dans un arbre vérifie que sa hauteur est strictement supérieure à celle de son père. Une racine non valide est donc un processeur p qui ne vérifie pas cette propriété. Dans ce cas, p prend la valeur erreur (E) qui est propagée à tous les processeurs de son arbre.

Raccordement. Le mécanisme effectif de raccordement d'un processeur p à un arbre est le suivant : un processeur q s'apercevant qu'un voisin p est susceptible de se connecter à lui (p est en erreur ou p est déjà connecté à un arbre mais sa hauteur est supérieure à celle de $q+1$) lance une requête à l'oracle. Si q obtient une réponse de l'oracle (cette réponse est forcément positive car l'oracle ne délivre pas de réponse négative, par contre, il peut ne pas en délivrer) alors p peut se raccorder à q si la hauteur de q est minimale pour p.

Oracle. Le mécanisme de l'oracle consiste à faire savoir à q s'il est bien dans l'arbre valide. Pour cela q génère une question destinée à la racine de l'arbre auquel il est connecté. Cette question est relayée successivement par les processeurs sur la branche de l'arbre reliant q à la racine. Si cette question venant de q parvient à la racine r, alors q obtient une réponse (positive) de r, sinon aucune réponse n'est obtenue par q. Afin d'assurer les complexités annoncées, l'oracle gère une notion de priorité liée à la distance dans l'arbre du demandeur à r. Ainsi toute demande effectuée par un processeur q' plus proche que q de r effacera toute trace de la demande de q sur la portion de chemin commune au parcours des deux demandes, y compris si r a déjà commencé à répondre à q. Si q n'a pas obtenu de réponse, sa demande sera à nouveau relayée à partir de la portion commune lorsque la réponse à q' aura été fournie.

3.2 Complexité en temps

L'oracle traite en priorité les demandes des processeurs les plus proches de r dans l'arbre valide, en $O(n)$ pas de calcul par demande. Une synchronisation est réalisée pour les demandes parallèles faites par les processeurs de même hauteur, nécessitant au plus un nombre polynomial de retransmissions. Les caractéristiques de ce mécanisme sont résumées par le lemme ci-dessous.

Lemme 1 *Soit T un arbre valide enraciné en r et h_{min} la hauteur des processeurs demandeurs les plus proches de r. En $O(h_{min})$ rondes et $O(n^2)$ pas de calcul, au moins un processeur demandeur obtient une autorisation de r à sa demande.*

L'oracle permet de construire l'arbre couvrant en largeur niveau après niveau. Il y a au plus d niveaux et la construction de chaque niveau nécessite l'obtention d'autorisations obtenues en $O(d)$ rondes (Lemme 1), conduisant à une complexité en $O(d^2)$ rondes pour construire la solution. N.B. : les processeurs issus d'un arbre non valide peuvent se reconnecter sans attendre la fin de la propagation du statut E.

L'oracle a aussi la propriété d'interdire la reconnexion d'un processeur à un arbre non valide par le même voisin. Ainsi, pour construire l'arbre en largeur final au plus $\Delta n + n^2$ connexions dans le réseau sont réalisées (au plus Δ connexions aux arbres non valides par processeur et n connexions dans l'arbre valide par processeur). La connexion d'un processeur est la conséquence d'une demande d'autorisation, faite par au plus chaque voisin (c'est-à-dire, au plus Δ demandes générées par connexion). Donc au total, il y aura au plus $\Delta m + mn$ demandes. De plus, seul des réponses positives sont délivrées. Chaque autorisation est transmise en $O(n^2)$ pas de calcul (Lemme 1). En $O(n^3)$ pas de calcul toutes les autorisations sont délivrées (il y a au plus n demandes à tout moment dans le réseau). Ainsi, le produit de la complexité en pas de calcul pour traiter une demande et le nombre maximum de demandes donne une borne supérieure sur la complexité en pas de calcul donnée dans le théorème 1 pour atteindre une configuration terminale.

Théorème 1 *Partant d'une configuration arbitraire, en $O(d^2)$ rondes et $O(\Delta mn^3 + mn^4) \leq O(n^6)$ pas de calcul l'algorithme instantanément stabilisant $\mathcal{BFS}$ construit un arbre couvrant en largeur enraciné en r.*

4 Conclusions et perspectives

Dans cet article nous avons proposé un algorithme permettant de construire un arbre couvrant avec une complexité en $O(d^2)$ rondes et $O(\Delta mn^3 + mn^4)$ pas de calcul. Il en résulte que le problème de la construction d'un arbre couvrant sur un réseau quelconque est un problème totalement polynomial. Pour autant quelques questions restent en suspens :

- Peut-on améliorer la complexité en pas de calcul en conservant une complexité en $O(d^2)$ rondes ?
- Est-il possible de construire un arbre couvrant en $O(d)$ rondes et un nombre polynomial de pas de calcul ?
- Quels sont les problèmes qui acceptent une solution totalement polynomiale ?
- Quels sont les problèmes qui n'acceptent pas de solution totalement polynomiale ?

Cette notion d'algorithme totalement polynomial permet d'autre part d'envisager le passage à l'échelle des algorithmes distribués stabilisants. Il nous semble donc important voire primordial de prolonger cette recherche pour des algorithmes « totalement » efficaces.

Références

[BDPV99] A. Bui, A.K. Datta, F. Petit, and V. Villain. State-optimal snap-stabilizing pif in tree networks. In *Workshop on Self-stabilizing Systems (WSS)*, pages 78–85. IEEE Computer Society, 1999.

[CDV09] A. Cournier, S. Devismes, and V. Villain. Light enabling snap-stabilization of fundamental protocols. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(1), 2009.

[Cou09] A. Cournier. A new polynomial silent stabilizing spanning-tree construction algorithm. In *SIROCCO*, volume 5869 of *LNCS*, pages 141–153. Springer, 2009.

[CRV11] A. Cournier, S. Rovedakis, and V. Villain. The first fully polynomial stabilizing algorithm for bfs tree construction. In *OPODIS*, pages 159–174, 2011.

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

[HC92] S.-T. Huang and N.-S. Chen. A self-stabilizing algorithm for constructing breadth-first trees. *Inf. Process. Lett.*, 41(2) :109–117, 1992.

[KK05] A. Kosowski and L. Kuszner. A self-stabilizing algorithm for finding a spanning tree in a polynomial number of moves. In *PPAM*, volume 3911 of *LNCS*, pages 75–82. Springer, 2005.