Algorithme de k-partitionnement auto-stabilisant et compétitif

Ajoy K. Datta, Stéphane Devismes, Karel Heurtefeux, Lawrence L. Larmore, Yvan Rivierre

Algorithme de k-partitionnement auto-stabilisant et compétitif[†]

A. K. Datta[1], S. Devismes[2], L. L. Larmore[1], K. Heurtefeux[2] et Y. Rivierre[2]

[1] *School of Computer Science, UNLV, Las Vegas, USA*
[2] *VERIMAG, UMR 5104, Université Joseph Fourier, Grenoble*

Nous proposons un algorithme auto-stabilisant qui calcule un k-partitionnement de taille $O(\frac{n}{k})$ dans un réseau quelconque de n processus. Lorsque le réseau est un graphe de disques unitaires, le k-partitionnement calculé est au plus $7.2552k+O(1)$ fois plus grand que le minimum possible. Lorsque le réseau est un graphe de disques quasi-unitaires de paramètre λ, le k-partitionnement calculé est au plus $7.2552k\lambda^2+O(\lambda)$ fois plus grand que le minimum possible.

Keywords: systèmes distribués, auto-stabilisation, k-partitionnement, compétitivité

1 Introduction

L'*auto-stabilisation* [Dij74] 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 auto-stabilisant retrouve de lui-même, et en temps fini, un comportement normal dès lors que les fautes transitoires ont cessé.

Dans cet article, nous proposons un algorithme distribué auto-stabilisant calculant un *k-partitionnement* du réseau. Le *k-partitionnement* d'un graphe (ou d'un réseau) consiste à diviser ses nœuds en sous-ensembles disjoints $C_1,\ldots,C_x$, appelés *grappes*, et à identifier un nœud $Tête_i$, appelé *tête*, dans chaque grappe C_i tel que tout nœud de C_i est au plus à distance k de $Tête_i$.

Le k-partitionnement est généralement utilisé dans les réseaux pour réaliser du routage hiérarchique. Supposons qu'un processus p souhaite communiquer avec un processus q situé dans une grappe différente. Alors, p envoie son message à sa tête de grappe, qui le transmet ensuite à la tête de la grappe de q ; enfin, cette dernière envoie le message à q.

Généralement, on souhaite limiter la taille d'un k-partitionnement, *i.e.*, son nombre de grappes. Cependant, trouver un k-partitionnement de taille minimum est $\mathcal{NP}$-difficile [GJ79]. Nous proposons de calculer un k-partitionnement dont la taille est une approximation [‡] du minimum possible dans le cas où le réseau est un *UDG* (graphe de disques unitaires) ou un *quasi-UDG* (les UDG et quasi-UDG modélisent les topologies des réseaux de capteurs sans fils). Dans les topologies UDG ou quasi-UDG, chaque nœud est placé dans le plan. Dans un UDG, deux nœuds sont voisins si et seulement s'ils sont distants d'au plus 1. Dans un quasi-UDG de paramètre $\lambda \geq 1$, (1) si deux nœuds sont distants d'au plus 1, alors ils sont voisins, et (2) pour chaque paire de voisins (x,y), la distance entre x et y est au plus λ.

Un algorithme de k-partitionnement est dit *X-compétitif* s'il calcule un k-partitionnement de taille au plus X fois la taille du plus petit possible. L'algorithme auto-stabilisant présenté dans cet article calcule un k-partitionnement de taille $O(\frac{n}{k})$ dans un réseau quelconque de n processus. Il stabilise en $O(n)$ rondes [§] et a un encombrement mémoire en $O(\log n)$ bits par processus. Notre algorithme est $7.2552k+O(1)$-*compétitif* lorsque le réseau est un *UDG* et $7.2552k\lambda^2+O(\lambda)$-*compétitif* lorsque le réseau est un *quasi-UDG* de paramètre λ.

[†] Cet article est un résumé du rapport technique en ligne : `http://www-verimag.imag.fr/TR/TR-2011-16.pdf`.
‡. Cette approximation sera formalisée par la notion de X-compétitivité.
§. La notion de *ronde* mesure le temps d'exécution rapporté au processus le plus lent.

2 Modèle

Nous considérons des réseaux bidirectionnels connexes 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. L'exécution d'un algorithme est une suite d'étapes atomiques : à chaque étape, s'il existe des processus — dits *activables* — souhaitant modifier leurs états, 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 en supposant une *équité faible*, *i.e.*, tout processus *continûment* activable est activé en temps fini.

3 L'algorithme

Notre algorithme est une *composition parallèle* de deux algorithmes. Le premier calcule un arbre couvrant particulier, le second calcule un k-partitionnement de l'arbre couvrant calculé par le premier algorithme. Les propriétés de cet arbre couvrant nous permettent d'obtenir la compétitivité de notre algorithme.

Dans une composition parallèle de deux algorithmes auto-stabilisants : les deux algorithmes s'exécutent de manière concurrente, mais le second algorithme utilise les variables de sortie du premier algorithme dans ses calculs. Ainsi, le second algorithme ne stabilise qu'une fois que le premier a stabilisé.

3.1 Arbre couvrant EIM

La première partie de l'algorithme consiste à calculer un arbre couvrant EIM (pour Ensemble Indépendant Maximal). Un arbre couvrant EIM est un arbre couvrant dont les nœuds de hauteur paire forment un ensemble indépendant maximal (*n.b.* la racine de l'arbre est de hauteur 0).

Un ensemble indépendant est un ensemble de nœuds E tel qu'aucun nœud de E n'est voisin d'un autre nœud de E. Un ensemble indépendant E est dit maximal si aucun de ses sur-ensembles propres n'est un ensemble indépendant (*i.e.*, l'ajout d'un nœud à E donne un ensemble qui n'est plus indépendant).

Notez que par définition, tout arbre couvrant EIM, T, a la propriété suivante : tout chemin de T de longueur ℓ contient au moins $\lceil \frac{\ell}{2} \rceil$ nœuds de I, où I est l'ensemble indépendant maximal formé par les nœuds de hauteur paire de T. La figure 1 présente un arbre couvrant EIM. Les arêtes sont représentées par des flèches et les nœuds de hauteur paire sont coloriés en noir.

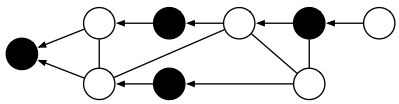

FIGURE 1: Arbre couvrant EIM.

Notre calcul auto-stabilisant d'arbre couvrant EIM utilise aussi la composition parallèle : un premier algorithme calcule un arbre couvrant (pour cela, nous utilisons l'un des nombreux algorithmes auto-stabilisants de la littérature), le second algorithme utilise les hauteurs des processus de l'arbre calculé par le premier algorithme pour calculer un arbre couvrant EIM. Cette hauteur est notée $p.hauteur$ pour tout processus p.

Tout processus p maintient deux variables supplémentaires dans le second algorithme : (1) un booléen $p.dominant$ et (2) un pointeur père $p.père_{EIM}$ qui pointe soit vers p soit vers un de ses voisins.

Lorsque la variable $p.dominant$ est vraie, p est dit *dominant*, dans le cas contraire, p est dit *dominé*. Une fois stabilisée, $p.dominant$ est vraie si et seulement si p est à une hauteur paire ; et l'ensemble constitué des processus q tel que $q.dominant = vrai$ est un ensemble indépendant maximal.

La variable $p.père_{EIM}$ désigne le père de p dans l'arbre couvrant EIM : une fois stabilisée, si p est la racine de l'arbre, alors $p.père_{EIM} = p$ sinon $p.père_{EIM} = q$ où q est le père de p dans l'arbre EIM.

Pour décider s'il est dominant ou dominé, chaque processus p utilise une priorité définie par le couple $(p.hauteur, p)$ (le deuxième membre du couple désigne l'identité de p). Le processus p est prioritaire sur le processus q si et seulement si $(p.hauteur, p)$ est lexicographiquement plus petit que $(q.hauteur, q)$. En fonction des priorités, p décide qu'il est dominant si et seulement si tous ses voisins sont dominés ou moins prioritaires. En utilisant cette règle, la racine de l'arbre calculée par le premier algorithme, notée R, se désigne « dominante ». Tous ses voisins deviennent alors « dominés », *etc*. Ainsi les nœuds dominants finissent par définir un ensemble indépendant maximal.

Chaque processus doit aussi choisir un père dans l'arbre de telle manière que les nœuds de hauteur paire définissent un ensemble indépendant maximal. Pour cela, R se désigne comme racine de l'arbre EIM en assignant $R.père_{EIM}$ à R. Les autres processus choisissent comme père dans l'arbre EIM leur voisin le plus prioritaire parmi ceux ayant un statut dominant/dominé différent du leur. Cette règle force une alternance stricte dominant/dominé le long des chemins de l'arbre. Le fait que la racine soit de hauteur 0 et qu'il y ait une stricte alternance dominant/dominé induit que l'arbre couvrant calculé est un arbre EIM.

3.2 k-partitionnement de l'arbre

Nous calculons le k-partitionnement en deux phases. La première phase consiste à calculer un ensemble d'au plus $1+\lfloor\frac{n-1}{k+1}\rfloor$ têtes de grappes. La seconde phase consiste à calculer une forêt couvrante dont chaque arbre représente une grappe et a ainsi pour racine l'une des têtes calculées lors de la première phase.

Pour cela, chaque nœud p utilise trois variables : $p.\alpha$, $p.tête_{grappe}$ et $p.père_{grappe}$. La variable $p.\alpha$ contient un entier entre 0 et $2k$, $p.tête_{grappe}$ contient une identité et $p.père_{grappe}$ désigne soit p soit l'un de ses voisins.

La variable $p.\alpha$ est utilisée pour déterminer si p est une tête. Une fois que l'algorithme a stabilisé, p est une tête de grappe si et seulement si $p.\alpha = k$ ou $p.\alpha < k$ et $p = R$ (où R est la racine de l'arbre EIM). La variable $p.tête_{grappe}$ contient l'identité de la tête de grappe de p et $p.père_{grappe}$ désigne le père de p dans sa grappe (si p est une tête, alors $p.père_{grappe} = p$).

Calcul des têtes. Le calcul des têtes est uniquement basé sur le calcul de la variable α. Cette dernière est calculée de bas en haut dans l'arbre EIM. Une fois l'algorithme stabilisé, $p.\alpha$ est égale à la distance entre le nœud p et le nœud le plus éloigné qui est à la fois dans le sous-arbre de p et dans la même grappe que p.

(a) Si $p.\alpha < k$, alors p est dit *minus* et deux cas sont possibles : $p \neq R$ ou $p = R$. Dans le premier cas, la tête de la grappe contenant p est *en dehors du sous-arbre* de p, *i.e.*, le chemin menant à cette tête passe par le père de p dans l'arbre EIM, et la distance entre p et cette tête est au plus égale à $k - p.\alpha$. Dans le second cas ($p = R$), p doit être une tête de grappe car, par définition, il n'existe aucun nœud hors de son sous-arbre.

(b) Si $p.\alpha \geq k$, alors p est dit *balaise* et la tête de sa grappe est *dans son sous-arbre* à distance $p.\alpha - k$.

$p.\alpha$ est calculée à partir des deux fonctions suivantes :

- $MaxAlphaMinus(p)$ est égale à la valeur α maximale des fils « minus » de p (dans l'arbre EIM). Si p n'a pas de fils « minus », alors $MaxAlphaMinus(p) = -1$.
- $MinAlphaBalaise(p)$ est égale à la valeur α minimale des fils « balaise » de p (dans l'arbre EIM). Si p n'a pas de fils « balaise », alors $MinAlphaBalaise(p) = 2k+1$.

À partir de ces fonctions, $p.\alpha$ est calculée comme suit :

- Si $MaxAlphaMinus(p) + MinAlphaBalaise(p) > 2k-2$, alors $p.\alpha = MaxAlphaMinus(p) + 1$.
- Si $MaxAlphaMinus(p) + MinAlphaBalaise(p) \leq 2k-2$, alors $p.\alpha = MinAlphaBalaise(p) + 1$.

Les valeurs des variables α sont calculées inductivement à partir des feuilles de l'arbre couvrant EIM.

Considérons une feuille f. Par définition, $MaxAlphaMinus(f) + MinAlphaBalaise(f) = -1 + 2k + 1 > 2k-2$. Ainsi, $f.\alpha = -1+1 = 0$. En effet, la distance maximale entre f et un nœud de son sous-arbre qui est dans la même grappe est, par définition, 0.

Considérons maintenant un nœud interne p et supposons que les valeurs des variables α de tous ses fils sont correctes. p doit choisir si sa tête de grappe est soit (1) dans son sous-arbre (dans ce cas, p sera *balaise*), soit (2) hors de son sous-arbre (dans ce cas, p sera *minus*). Nous privilégions, autant que possible, le choix (1) pour limiter le nombre de têtes.

Soit q un nœud qui a la même tête de grappe que p et qui appartient au sous-arbre d'un fils *minus* de p. D'après (a), le chemin menant de q à sa tête de grappe passe par p. Ainsi, pour ne pas créer de cycle, (*) p ne doit *jamais* choisir comme tête de grappe un nœud appartenant au sous-arbre d'un de ses fils *minus*. Soit x le plus éloigné des sommets q. Toujours d'après (a), x est à distance $MaxAlphaMinus(p)+1$ de p. Deux cas sont alors possibles :

$MaxAlphaMinus(p) + MinAlphaBalaise(p) > 2k-2$**.** Si p choisit un nœud y de son sous-arbre comme tête de sa grappe (choix (1)), alors d'après (*), le chemin vers y passe par un fils *balaise* de p et p sera au moins à distance $MinAlphaBalaise(p) - k + 1$ de sa tête de grappe, d'après (b). Or, dans ce cas, x sera au moins à distance $MaxAlphaMinus(p) + MinAlphaBalaise(p) - k + 2 > 2k - 2 - k + 2 = k$ de

la tête y, ce qui viole la définition du k-partitionnement. Ainsi, p doit nécessairement choisir une tête de grappe hors des sous-arbres de ses fils (choix (2)). D'après (a) et (b), le nœud x est alors le nœud le plus éloigné appartenant à la fois au sous-arbre de p et à la même grappe que p, ce qui implique que $p.\alpha = MaxAlphaMinus(p) + 1$.

$MaxAlphaMinus(p) + MinAlphaBalaise(p) \leq 2k-2$. Soit z un fils *balaise* de p tel que $z.\alpha = MinAlphaBalaise(p)$. Contrairement au cas précédent, p peut choisir un nœud y du sous-arbre de z comme tête de grappe (choix (1)). En effet, ici, x sera à distance $MaxAlphaMinus(p) + MinAlphaBalaise(p) - k + 2 \leq 2k - 2 - k + 2 = k$ de y. D'où, les nœuds (autres que p) appartenant à la fois au sous-arbre de p et à la même grappe que p sont soit des nœuds des sous-arbres de fils *minus* de p, soit des nœuds du sous-arbre de z. Puisque par définition $MinAlphaBalaise(p) > MaxAlphaMinus(p)$, le nœud le plus éloigné appartenant à la fois au sous-arbre de p et à la même grappe que p sera à la distance $MinAlphaBalaise(p) + 1$ de p, *i.e.* $p.\alpha = MinAlphaBalaise(p) + 1$.

Construction des grappes. Lorsqu'un nœud p vérifie $p.\alpha = k$ ou $p.\alpha < k$ et $p = R$, il se désigne comme tête en assignant $p.père_{grappe}$ et $p.tête_{grappe}$ à p.

Ensuite, chaque nœud non-tête, q, doit choisir un père dans sa grappe. Si q est *minus*, il choisit son père dans l'arbre couvrant EIM comme père dans sa grappe. Sinon, q est *balaise* et choisit un fils *balaise* c tel que $c.\alpha = q.\alpha - 1$ comme père de grappe.

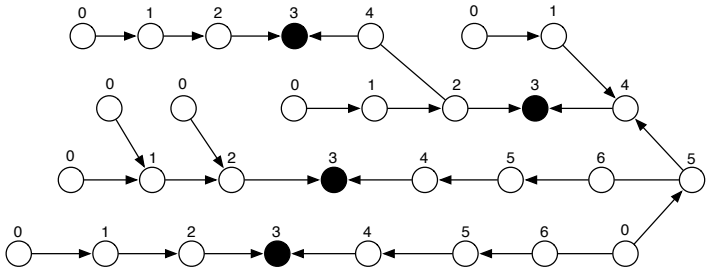

FIGURE 2: Exemple de 3-partitionnement.

De cette manière, les grappes sont structurées en arbres. La dernière étape consiste à propager, via les liens « père » ($père_{grappe}$), la valeur $tête_{grappe}$ de chaque tête dans les variables $tête_{grappe}$ des autres nœuds de son arbre.

La figure 2 montre un 3-partitionnement calculé par notre algorithme : la racine est à droite, les entiers sont les valeurs de α, les têtes sont en noir et les flèches représentent les liens « père » des « grappes ».

4 Taille du k-partitionnement

Tout d'abord, toutes les têtes t, sauf éventuellement une, vérifient $t.\alpha = k$ (si R est *minus*, R est une tête). Donc, toutes les grappes sauf éventuellement une, contiennent au moins k nœuds. Ainsi, il y a au plus $1 + \lfloor \frac{n-1}{k} \rfloor = O(\frac{n}{k})$ grappes, quelle que soit la topologie du réseau.

Nous avons étudié la compétitivité de notre algorithme dans les cas où la topologie du réseau est soit un UDG, soit un quasi-UDG en utilisant un résultat précédent de Folkman et Graham [FG69]. À partir de ce résultat, nous avons montré que, dans un UDG (connexe), (a) la taille de tout ensemble indépendant est au plus $\left(\frac{2\pi k^2}{\sqrt{3}} + \pi k + 1\right)$ fois plus grande que la taille Opt du plus petit k-partitionnement. Or, nous savons d'après la propriété de notre arbre EIM, que toutes les grappes, sauf éventuellement une, contiennent au moins $\lceil \frac{k}{2} \rceil$ nœuds de l'ensemble indépendant maximal I défini par les nœuds de hauteur paire de notre arbre EIM. Soient $nb_I = |I|$ et nb_T le nombre de têtes de grappes. Nous avons donc (b) $(nb_T - 1) \times \lceil \frac{k}{2} \rceil \leq nb_I - 1$. D'après (a) et (b), nous déduisons alors que $nb_T \leq 1 + \left(\frac{4\pi k}{\sqrt{3}} + 2\pi\right) \times Opt$. Comme $\frac{4\pi}{\sqrt{3}} \approx 7.2552$, nous déduisons que notre algorithme est $7.2552k + O(1)$-*compétitif* lorsque le réseau est un *UDG*. Par une approche similaire, nous montrons que notre algorithme est $7.2552k\lambda^2 + O(\lambda)$-*compétitif* lorsque le réseau est un *quasi-UDG* de paramètre λ.

Références

[Dij74] E. W. Dijkstra. Self-Stabilizing Systems in Spite of Distributed Control. *Commun. ACM*, 17 :643–644, 1974.

[FG69] J. H. Folkman and R. L. Graham. An Inequality in the Geometry of Numbers. *Canad. Math. Bull.*, 12 :745–752, 1969.

[GJ79] M. R. Garey and David S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.