Sur le Coloriage Auto-stabilisant dans les Réseaux Unidirectionnels Anonymes †

Samuel Bernard[1], Stéphane Devismes[2], Katy Paroux[3], Maria Potop-Butucaru[1] et Sébastien Tixeuil[1]

[1] *Université Pierre et Marie Curie LIP6/CNRS, UMR 7606, Paris, France*
[2] *Université Grenoble I VERIMAG/CNRS, UMR 5104, Grenoble, France*
[3] *INRIA Bretagne Atlantique, France*

Nous considérons des réseaux unidirectionnels anonymes. Nous démontrons que contrairement aux réseaux bidirectionnels, l'auto-stabilisation de tâches locales peut y être aussi difficile que l'auto-stabilisation de tâches globales. Pour ce faire, nous prenons comme exemple le problème du *coloriage* des nœuds d'un réseau. Plus précisément, nous démontrons que le coloriage auto-stabilisant est intrinsèquement aussi difficile à résoudre de manière déterministe qu'une tâche globale. Nous proposons ensuite une approche probabiliste pour retrouver une complexité locale.

Keywords: coloriage, réseaux unidirectionnels, auto-stabilisant, anonyme, algorithme distribué

1 Introduction

Un algorithme distribué est *auto-stabilisant* [Dol00] si, à partir d'un état global quelconque du système (obtenu suite à des fautes ou des attaques transitoires, par exemple), l'algorithme retrouve un comportement correct en un temps fini et sans aucune intervention extérieure (et en particulier humaine).

La plupart des solutions auto-stabilisantes de la littérature [Dol00] considère des réseaux bidirectionnels, *i.e.*, si un processus p est capable d'envoyer des informations à un processus q, alors p peut aussi recevoir des informations en provenance de q. Cette hypothèse est valide dans la plupart des cas. Cependant, les communications peuvent être asymétriques dans certains cas : par exemple, dans un réseau de capteurs communiquant sans fil, les portées des antennes peuvent être hétérogènes et un processus q peut être à portée d'un processus p, alors que p n'est pas à portée de q. Dans ce cas, q peut recevoir des informations venant de p, alors que p ne peut rien recevoir de q.

Dans les réseaux bidirectionnels, les tâches *globales*‡ (l'élection de leader, l'exclusion mutuelle [DGT04], par exemple) sont souvent considérées comme plus complexes à résoudre [Dol00] que les tâches *locales*§ (le coloriage, par exemple [GT00]). En effet, dans la plupart des cas, l'espace mémoire et le nombre d'actions par processus nécessaires pour stabiliser dépendent de paramètres globaux (le diamètre, par exemple) pour les tâches globales et de paramètres locaux (le degré des processus, par exemple) pour les tâches locales.

Dans cet article, nous considérons des réseaux unidirectionnels anonymes. Nous démontrons que contrairement aux réseaux bidirectionnels, les tâches locales peuvent être aussi difficiles à résoudre que les tâches globales. Pour ce faire, nous considérons le problème du *coloriage* où chaque processus doit afficher une couleur qui diffère de celles de ses voisins. Si n est le nombre de processus, nous démontrons que $\Omega(n)$ états et $n(n-1)/2$ pas de calculs sont nécessaires pour résoudre ce problème de manière *déterministe*. Ces bornes démontrant une difficulté intrinsèque équivalente à celle d'une tâche globale. Nous proposons ensuite un algorithme *déterministe* optimal pour le coloriage, contraint toutefois par une hypothèse de synchronisation spatiale. Nous montrons ensuite qu'une approche probabiliste permet de retrouver une complexité locale et de lever la contrainte spatiale. La complexité de ce dernier algorithme montre un compromis entre la quantité de mémoire nécessaire et le temps de convergence moyen.

† Ce travail a bénéficié du support des projets ANR SHAMAN et SOGEA

‡ *i.e.*, des tâches dont les spécifications interdisent certaines combinaisons d'états entre processus arbitrairement éloignés.

§ *i.e.*, des tâches dont les spécifications interdisent certaines combinaisons d'états entre processus voisins.

processus i
constantes
  k : nombre de couleurs
  $P.i$: ensemble des prédécesseurs de i
paramètre
  p : processus de $P.i$
variable
  $c.i$: couleur d'un processus i
action
  $p \in P.i, c.i = c.p \rightarrow$
    faire $p \in P.i, c.i = c.p \rightarrow$
      $c.i := c.i + 1 \mod \mathtt{k}$
    fin faire

FIG. 1: Algorithme déterministe

processus i
constantes
  k : nombre de couleurs
  $P.i$: ensemble des prédécesseurs de i
  $C.i$: ensemble des couleurs des processus de $P.i$
paramètre
  p : processus de $P.i$
variable
  $c.i$: couleur d'un processus i
fonction
  $\mathtt{random}(S$: ensemble des couleurs$)$: couleur
  // renvoie une couleur de S équiprobablement
action
  $p \in P.i, c.i = c.p \rightarrow c.i := \mathtt{random}((\{0,\ldots,\mathtt{k}-1\} \setminus C.i) \cup \{c.i\})$

FIG. 2: Algorithme probabiliste

2 Modèle

Dans cet article, nous considérons un réseau anonyme et unidirectionnel, *i.e.*, un graphe orienté connexe sans boucle où les sommets représentent des processus et où chaque arc (p,q) représente la possibilité pour q de recevoir des informations de p (mais pas l'inverse). Dans ce cas, on dit que p est prédécesseur de q et que q est successeur de p. Notons que les arcs (p,q) et (q,p) peuvent exister simultanément.

Nous prenons comme modèle de calcul le modèle à « états » : les variables d'un processus définissent son état. L'ensemble des états des processus à un instant donné forme la configuration du système. Les variables des processus sont partagées : chaque processus a un accès direct en lecture aux variables de ses prédécesseurs. De plus, en une seule étape atomique, un processus peut lire son état et ceux de ses prédécesseurs et mettre à jour son propre état.

Les exécutions sont gérées par un ordonnanceur : à chaque instant, s'il existe des processus souhaitant modifier leurs états, l'ordonnanceur autorise au moins un de ces processus à exécuter une étape de son algorithme local. On distingue deux types d'ordonnanceur : *distribué* ou *localement central*. Un ordonnanceur distribué peut activer indépendamment chaque processus à tout moment. À l'inverse, un ordonnanceur localement central ne peut pas activer deux processus voisins en même temps.

Impossibilité Notons qu'il n'existe pas d'algorithme auto-stabilisant déterministe uniforme résolvant le problème du coloriage dans un réseau anonyme unidirectionnel si l'ordonnanceur est distribué. Supposons un état initial où tous les processus sont dans le même état (nombre de voisins et mémoire). Activons tous les processus au même moment, ce qui est possible si l'ordonnanceur est distribué. Puisque l'algorithme est uniforme, ils vont tous exécuter les mêmes instructions. Comme ils sont dans le même état initialement et que l'algorithme est déterministe, après leur activation ils atteignent le même nouvel état. Nous nous retrouvons dans le cas initial. Il est donc possible d'exécuter un nombre infini d'opérations sans arriver à un état correct.

3 Algorithme Déterministe

Dans cette section, nous supposons que l'ordonnanceur est localement central, c'est-à-dire que deux processus voisins ne peuvent pas être activés au même moment.

Bornes inférieures Les bornes inférieures en temps et en espace pour résoudre le coloriage de manière déterministe dans un réseau unidirectionnel anonyme sont différentes de celles que l'on obtient avec des réseaux bidirectionnels ou avec des algorithmes probabilistes. En effet, un algorithme auto-stabilisant uniforme résolvant le problème du coloriage dans un réseau unidirectionnel a besoin d'au moins n états mémoire différents, où n est le nombre de processus dans le réseau. Par manque de place, la preuve est uniquement présentée dans la version longue de notre article [BDPBT08].

Concernant la complexité en temps, un algorithme auto-stabilisant uniforme résolvant le problème du

coloriage dans un réseau unidirectionnel a besoin d'effectuer au moins $\frac{n(n-1)}{2}$ étapes dans le pire cas. Pour le prouver, nous observons qu'un processus qui a le même état que son prédécesseur doit nécessairement agir, et nous prenons comme réseau une chaîne unidirectionnelle avec un puits noté 1 et une source notée n, les autres étant notés naturellement entre 1 et n dans l'ordre. On note E_{i-1} l'état atteint après une activation d'un processus ayant un seul prédécesseur et étant à l'origine dans l'état E_i. On suppose que dans l'état initial, tous les processus sont dans l'état E_n. En activant dans l'ordre les processus 1 à $n-1$ (ordonnancement valide), ces processus vont se retrouver dans l'état E_{n-1}. Dans le meilleur des cas, le processus $n-1$ ne va plus changer d'état puisque son unique prédécesseur n est dans un état différent E_n de lui E_{n-1}. En considérant les processus 1 à $n-1$ dans l'état E_{n-1}, on se retrouve avec une instance du problème initial de taille $n-1$. Par récurrence et parce qu'une chaîne de taille 1 nécessite 0 étape, le nombre d'étapes nécessaires est alors de $\frac{n(n-1)}{2}$.

Algorithme optimal en temps et en espace L'algorithme 1 est un algorithme de coloriage dont les bornes supérieures en temps et en espace correspondent exactement aux bornes inférieures prouvées précédemment, donc il est optimal. Cet algorithme peut être vu comme un algorithme « glouton ». Chaque processus ayant sa couleur en commun avec un de ses prédécesseurs change de couleur. La nouvelle couleur est choisie de manière cyclique : c'est la première couleur disponible dans l'ordre des couleurs, c'est-à-dire qui n'est prise par aucun de ses prédécesseurs.

Théorème 1 *L'algorithme présenté dans la figure 1 est un algorithme auto-stabilisant de coloriage dans les réseaux unidirectionnels anonymes, il nécessite n couleurs pour un réseau de taille n et a une complexité de* $\frac{n(n-1)}{2}$ *étapes. Il est optimal en temps et en espace.*

Preuve :

Considérons la table qui associe à chaque couleur l'ensemble des processus qui l'ont choisie. Par exemple, la table suivante nous indique que les processus 2 et 3 ont la couleur 0, que le processus 1 a la couleur 2, etc.

Processus	P_3, P_2		P_{n-1}, P_1		...	P_{n-2}
Couleurs	0	1	2	3	...	$n-1$

La configuration initiale est arbitraire mais l'évolution de la table suit deux règles principales, découlant de l'algorithme :

- Une case ne peut pas passer d'un état non vide à un état vide. En effet, si un processus a une couleur différente de tous ses voisins, il ne va pas en changer donc va rester dans la même case. De plus, l'ordonnanceur est localement central donc deux processus voisins de la même couleur ne pourront pas s'activer au même moment et changer tous les deux de couleurs.
- Un processus se déplace dans la table uniquement de gauche à droite de manière cyclique. De plus, il ne peut pas sauter une case vide. Ceci est dû au fait qu'un processus, quand il change de couleur, prend la première couleur disponible parmi ses voisins. Or une case vide indique une couleur choisie par aucun processus donc obligatoirement disponible.

Il y a n processus et n couleurs disponibles donc chaque processus peut choisir une couleur différente si besoin est. Remarquons qu'une configuration où chaque processus a choisi une couleur unique est toujours valide, quelle que soit la topologie du réseau mais que ce n'est pas une configuration nécessaire.

Comme un processus ne peut pas sauter une case vide, il est assuré au bout d'au plus $n-1$ déplacements de se retrouver dans une case vide. En effet, s'il a besoin de se déplacer, c'est qu'il y a un processus ayant choisi la même couleur que lui. Au plus $n-1$ couleurs sont prises (dont la sienne), et s'il parcourt toutes les autres, il arrivera nécessairement dans la cellule libre.

Le nombre maximum de mouvements pour trouver une case vide dépend du nombre de couleurs actuellement choisies. Si le réseau utilise k couleurs, un processus devra effectuer au plus k mouvements. Donc en partant d'une configuration où il n'y a qu'une couleur de prise, on aura un nombre d'étapes borné par $1+2+...+(n-1) = \frac{n(n-1)}{2}$.

Par les théorèmes précédents, on en déduit que cet algorithme est optimal à la fois en temps et en espace. □

4 Algorithme Probabiliste

L'algorithme présenté précédemment est à la fois déterministe et optimal en temps et en espace. Cependant, il nécessite un ordonnanceur localement central, n états et $O(n^2)$ étapes (où n est la taille du réseau) pour converger. Dans cette section, nous étudions le cas d'algorithmes probabilistes pouvant converger avec un ordonnanceur distribué.

Bornes inférieures Nous présentons deux bornes inférieures valides pour tout algorithme auto-stabilisant résolvant la coloration de sommets dans un réseau anonyme, qu'il soit unidirectionnel ou bidirectionnel.

1. *Le nombre minimal d'états nécessaire par processus est* $\Delta+1$ *où* Δ *est le degré du réseau*[¶]. Considérons un réseau contenant une clique de degré Δ donc ayant $\Delta+1$ processus. Alors avec moins de $\Delta+1$ états, il n'existe pas de coloriage valide du réseau.
2. *Le nombre minimal d'étapes pour arriver à une configuration valide est en* $\Omega(n)$. Supposons que le réseau est une clique de taille n et dont tous les processus ont initialement la même couleur. Alors au mieux, on aura besoin que $n-1$ processus s'activent et changent de couleur pour arriver à une configuration valide.

Algorithme et complexité L'algorithme présenté dans la figure 2 peut être décrit de la façon suivante. Si un processus a la même couleur que l'un de ses prédécesseurs alors il choisit une nouvelle couleur parmi les couleurs qu'il estime disponibles (*i.e.* qui ne sont pas utilisées par ses prédécesseurs) ou garde sa couleur. Le choix de la nouvelle couleur est équiprobable parmi les couleurs envisagées.

Théorème 2 *Si l'algorithme présenté dans la figure 2 utilise* $k \geq \Delta+1$ *états par processus, alors il est auto-stabilisant probabiliste pour la coloration des nœuds et converge en moyenne en* $\frac{nk}{k-\Delta}$ *étapes. En particulier si* $k = \Delta+1$*, l'espérance en nombre d'étapes est de* Δn*. Si k tend vers l'infini, alors l'espérance tend vers* $O(n)$.

La démonstration de ce théorème repose sur l'idée qu'il faut un nombre moyen fini d'étapes pour résoudre un conflit (*i.e.* un couple prédécesseur-successeur de même couleur) d'une part, et que le nombre initial de conflits est d'au plus n d'autre part. Ce raisonnement est formalisé par l'utilisation d'un arbre probabiliste de type Galton-Watson. Les détails de l'argumentaire se trouvent dans [BDPBT08].

5 Conclusion

Nous avons étudié la complexité de l'auto-stabilisation de tâches locales dans des réseaux unidirectionnels. Contrairement aux réseaux bidirectionnels classiques, la complexité du coloriage déterministe dépend maintenant de paramètres globaux (n états et n pas de calculs par processus) dans le cas déterministe. Nous avons montré que l'approche probabiliste était une alternative intéressante, permettant de moduler mémoire utilisée et temps de convergence.

Références

[BDPBT08] Samuel Bernard, Stéphane Devismes, Maria Gradinariu Potop-Butucaru, and Sébastien Tixeuil. Bounds for self-stabilization in unidirectional networks. Technical report, INRIA, May 2008.

[DGT04] Ajoy K. Datta, Maria Gradinariu, and Sébastien Tixeuil. Self-stabilizing mutual exclusion with arbitrary scheduler. *The Computer Journal*, 47(3) :289–298, 2004. **runner up Wilkes best paper award**.

[Dol00] S. Dolev. *Self-stabilization*. MIT Press, March 2000.

[GT00] Maria Gradinariu and Sébastien Tixeuil. Self-stabilizing vertex coloring of arbitrary graphs. In *International Conference on Principles of Distributed Systems (OPODIS'2000)*, pages 55–70, Paris, France, December 2000.

[¶] Par abus de langage, nous appelons degré d'un réseau unidirectionnel, le degré du graphe simple non-orienté sous-jacent.