Pour être César, il faut que tous les chemins viennent de la Domus Augustana †

Karine Altisen[1], Alain Cournier[2], Geoffrey Defalque[2] et Stéphane Devismes[2]

[1] *Univ. Grenoble Alpes, CNRS, Grenoble INP‡, VERIMAG, 38000 Grenoble, France*
[2] *Université de Picardie Jules Verne, MIS, Amiens (France)*

Nous considérons des réseaux dirigés où chaque processus est identifié et connait une borne supérieure sur l'écart maximum des processus à leur descendants. Dans ce type de réseaux, nous étudions les conditions topologiques nécessaires et suffisantes pour résoudre de manière autostabilisante deux problèmes fondamentaux : l'élection de leader et l'unisson synchrone. Cette condition contraint le réseau à n'avoir qu'une seule composante source (c'est-à-dire, une composante fortement connexe où aucun nœud n'a de prédécesseur hors de la composante). Pour démontrer que notre condition est suffisante pour les deux problèmes cités, nous proposons deux algorithmes autostabilisants, dont nous étudions également la complexité.

Mots-clefs : Réseaux dirigés, Autostabilisation, Unisson synchrone, Élection de leader

1 Introduction

La tolérance aux fautes est une préoccupation majeure dans les réseaux modernes. D'une part, ces réseaux regroupent généralement un grand nombre de machines géographiquement éloignées, ce qui implique qu'une intervention humaine pour les réparer peut être difficile, voire impossible. D'autre part, la disponibilité des réseaux doit être maximisée. Cependant, la tolérance aux fautes est une propriété parfois complexe à obtenir et lorsqu'elle est réalisable cela peut être au détriment des performances ; voir [**?**].

L'autostabilisation est un concept fondamental de l'algorithmique distribuée qui qualifie l'aptitude d'un système à atteindre en un temps fini une configuration dite *légitime* à partir de laquelle il a un comportement spécifique, et cela indépendamment de sa configuration initiale. Cette initialisation quelconque peut être vue comme la résultante d'un nombre fini de *fautes transitoires* dans le système. Ainsi, un système autostabilisant tolère naturellement ce type de fautes, c'est-à-dire, des perturbations temporaires et rares du fonctionnement de certains de ses composants (liens de communication ou processus), à condition que celles-ci ne modifient pas le code des processus §. À titre d'exemple, la corruption de messages est généralement considérée comme une faute transitoire.

Les topologies des réseaux modernes, tels que les réseaux de capteurs, les réseaux optiques ou encore l'internet des objets, sont généralement dirigées, c'est-à-dire que l'existence d'un canal permettant la communication d'un processus p vers un processus q n'implique pas nécessairement l'existence d'un canal de communication dans le sens inverse. Par exemple, dans un réseau de capteurs, la communication s'effectue par ondes radio et, de part la disparité des équipements, les portées d'émission/réception des capteurs sont parfois hétérogènes, créant ainsi ce type d'asymétrie.

Les réseaux dirigés sont une généralisation des réseaux bidirectionnels. En effet, dans un réseau dirigé, chaque lien de communication est soit bidirectionnel soit unidirectionnel. Lorsqu'un lien est orienté du nœud p vers le nœud q, p est appelé *prédécesseur* de q. Notez que si p et q sont reliés par un lien bidirectionnel alors ils sont prédécesseurs l'un de l'autre. Dans les réseaux bidirectionnels, la plupart des problèmes sont traités en supposant une topologie connexe. Dans les réseaux dirigés, la connexité se décline en deux notions : la connexité faible et la connexité forte. La plupart des problèmes distribués ne peuvent pas

‡Institute of Engineering Univ. Grenoble Alpes

†Ce travail a été soutenu par le projet ANR SKYDATA (ANR-22-CE25-0008).

§. Ce qui peut être garanti, entre autres, en stockant le code dans la ROM du processus, qui est une mémoire non réinscriptible.

être résolus dans des réseaux faiblement connexes quelconques car dans de tels réseaux, deux processus peuvent être incapables de communiquer, même indirectement, dans un sens et dans l'autre. D'ailleurs, à notre connaissance, il y a très peu de travaux en autostabilisation dédiés à de tels réseaux ([**?**] étant une exception notable). Ainsi, la majorité des travaux en autostabilisation sur les réseaux dirigés considèrent des réseaux fortement connexes [**?**], voire des cas particuliers de réseaux fortement connexes comme les anneaux unidirectionnels [**?**]. Or, il y a un fossé énorme entre ces deux notions de connexité et la principale question abordée dans cet article est de savoir où placer la frontière entre ce qui est faisable ou non pour deux problèmes donnés.

Contribution. Obtenir des solutions autostabilisantes déterministes nécessitent souvent des identifiants uniques et la connaissance de paramètres globaux du système. Ici, nous considérons le modèle à états (le modèle de calcul le plus couramment utilisé en autostabilisation) et nous supposons des réseaux dirigés où les processus sont identifiés de manière unique et connaissent une borne supérieure α sur l'écart maximum des processus à leurs descendants, l'écart entre un processus p et l'un de ses descendants q étant la longueur minimum d'un chemin reliant p à q.

Dans ce contexte, nous montrons d'abord qu'il existe un algorithme autostabilisant d'élection du leader si et seulement si la topologie du réseau contient exactement une *composante source*, c'est-à-dire, une composante fortement connexe où aucun nœud n'a de prédécesseur hors de la composante. Cette condition nécessaire et suffisante est vraie à la fois sous les hypothèses synchrone et asynchrone. Notez que la preuve de condition suffisante est constructive puisque nous proposons, sous l'hypothèse asynchrone, un algorithme autostabilisant (silencieux) d'élection de leader qui stabilise en $O(\alpha)$ rondes en utilisant $O(\log \alpha + B)$ bits de mémoire par processus (B étant le nombre de bits nécessaire pour stocker un identifiant).

Ensuite, à l'aide de cet algorithme, nous construisons également un algorithme d'unisson synchrone autostabilisant pour les réseaux à composante source unique. L'unisson synchrone est un problème de synchronisation d'horloges : chaque processus dispose d'une horloge locale et à chaque étape, toutes les horloges doivent s'incrémenter modulo K (avec $K \geq 2$) afin de rester synchronisées. (Ce problème nécessite donc de supposer des communications synchrones.) Notre algorithme se stabilise en $O(\alpha)$ rondes synchrones en utilisant $O(\log \alpha + B + \log K)$ bits de mémoire par processus. De ce résultat et des résultats de [**?**], nous déduisons alors que le fait que le réseau doive avoir une composante source unique est à la fois nécessaire et suffisant pour résoudre l'unisson synchrone autostabilisant.

À notre connaissance, les conditions topologiques nécessaires et suffisantes pour réaliser l'autostabilisation dans les réseaux dirigés n'avaient jusqu'à maintenant été que peu étudiées. Au final, nos résultats montrent, peut-être de manière surprenante, que des problèmes classiques, tels que l'élection et l'unisson synchrone —à première vue très différents— peuvent être résolus dans la même classe de réseaux, cette dernière étant strictement plus faible que la classe des réseaux fortement connexes (*n.b.*, un réseau fortement connexe a, par définition, une composante source unique : lui-même).

Plan. Dans la section suivante, nous présentons succintement le modèle à états. Ensuite, par manque de place, nous nous focalisons sur le problème de l'élection de leader : nous justifions brièvement pourquoi notre condition est nécessaire, ensuite nous présentons les idées principales de notre algorithme. Les résultats complets sont disponibles dans notre rapport technique en ligne (`https://hal.science/hal-04434345v1`).

2 Modèle

Nous considérons des réseaux dont la topologie est un graphe dirigé de n nœuds où les nœuds représentent les processus et les arcs les liens de communications unidirectionels. Pour tout arc (p, q), q est dit *successeur* de p et p est dit *prédécesseur* de q.

Le modèle à états est une abstraction du modèle à messages dans lequel les échanges d'informations sont réalisés *via* des *variables localement partagées* : chaque processus détient un nombre fini de variables dans lesquelles il peut lire et écrire ; de plus, il peut lire les variables de ses prédécesseurs dans le réseau. L'exécution d'un algorithme est composée d'une succession de pas de calcul atomiques. À chaque pas de calcul, chaque processus détermine en fonction de son état et de celui de ses prédécesseurs s'il est activable, c'est-à-dire s'il doit modifier ses variables. Ensuite, si au moins un processus est activable, alors un adversaire,

le *démon*, choisit un sous-ensemble non vide de processus activables. En un pas atomique, les processus choisis sont activés et mettent à jour leurs variables. Nous considérons ici deux types de démons. Le démon *distribué inéquitable*, le plus général : il active les processus de manière asynchrone sans aucune hypothèse sur l'équité. Le démon *synchrone* active tous les processus activables à chaque pas de calcul.

Le temps de stabilisation est le temps maximal pour atteindre une configuration légitime. Nous l'exprimons en nombre de rondes : la première ronde se termine dès lors que tous les processus qui étaient continûment activables depuis le début de l'exécution ont exécuté au moins une action. La seconde ronde commence à la fin de la première, *etc*.

3 Élection de Leader

Dans un réseau identifié, l'élection de leader consiste à accorder l'ensemble des processus sur l'identité de l'un d'entre eux.

Condition nécessaire. Pour rappel, notre condition impose l'existence d'une composante source unique dans le réseau. Pour montrer que cette condition est nécessaire pour l'élection de leader, nous avons tout d'abord prouvé que les processus d'une composante source S doivent nécessairement élire l'identité d'un processus de S. En effet, puisque, par définition, aucun prédécesseur des processus de S n'est en dehors de la composante, les processus de S ne peuvent acquérir d'informations (une identité, par exemple) sur un processus hors de S. À partir de ce résultat, nous procédons par contradiction : si le réseau comporte au moins deux composantes sources A et B, les processus de A éliront finalement une identité différente de celle calculée par les processus de B puisque, par définition, A et B n'ont pas de processus en commun.

Condition suffisante. Pour démontrer que notre condition topologique est suffisante, nous proposons un algorithme autostabilisant d'élection de leader stabilisant, sous l'hypothèse d'un démon distribué inéquitable, dans tous réseaux à composante source unique. Notre algorithme est en fait une composition de trois instances d'un même algorithme générique, `C-MAI`.

Composition. Nous notons $B \circ A$ la composition entre deux algorithmes A et B. Dans $B \circ A$, A et B sont exécutés concurremment et B utilise la sortie de A dans ses calculs. Nous imposons que B n'écrive pas dans les variables de A. De plus, nous modifions le code de B de sorte qu'un processus ne puisse exécuter une action de B que si aucune action de A n'est activable.

Notre opérateur de composition $\circ$ est associatif à droite : la composition $C \circ B \circ A$ correspond à la composition entre C et $B \circ A$. Ainsi, localement à chaque processus, A sera prioritaire sur B et C, et B sera lui-même prioritaire sur C.

Le minimum parmi les entrées des ancêtres. `C-MAI` suppose que chaque processus p a une entrée $\mathcal{I}(p)$ appartenant à un domaine ordonné. Le but consiste alors pour chaque processus p à calculer dans une variable de sortie, $p.M$, la valeur de l'entrée la plus petite parmi celle de ses ancêtres dans le réseau, c'est-à-dire l'ensemble $Anc(p)$ des processus ayant un chemin atteignant p. Notez que puisque p est atteignable depuis lui-même par un chemin vide, $p \in Anc(p)$. En plus de $p.M$, le processus p calculera dans la variable $p.d$ l'écart depuis un ancêtre q ayant la valeur de $p.M$ comme entrée (c'est-à-dire $\mathcal{I}(q) = p.M$). Par définition, l'écart de tout ancêtre de p à p est borné par α. Donc, le domaine de $p.d$ sera $[0..\alpha]$. Dans la suite, nous appelerons *clé* de p le couple $p.\mathcal{K} = (p.M, p.d)$. La *clé propre* de p sera $(\mathcal{I}(p), 0)$. À cause de la configuration initiale quelconque, certains processus peuvent avoir dans leur variable M une valeur qui n'est l'entrée d'aucun de leurs ancêtres, une telle valeur sera appelée *fausse valeur* et, par extension, une clé contenant une fausse valeur sera appelée *fausse clé*.

Le principe général de notre algorithme est une simple propagation de minimum. Chaque processus p calcule sa clé $p.\mathcal{K}$ pour qu'elle soit la valeur minimale (dans l'ordre lexicographique) parmi sa clé propre $(\mathcal{I}(p), 0)$ et celles de l'ensemble $\{(q.M, q.d + 1) \mid q \in \Gamma^-(p) \wedge q.d < \alpha\}$ où $\Gamma^-(p)$ est l'ensemble des prédécesseurs de p. Cet ensemble contient les clés de chaque prédécesseur de p avec un écart augmenté de 1 : en effet, si p adopte la valeur $q.M$, alors l'écart à l'ancêtre a ayant $q.M$ comme entrée est l'écart de a jusqu'à q augmenté de 1. Cependant, les clés ayant atteint l'écart α sont exclues car, par définition, $p.d$ ne peut prendre la valeur $\alpha + 1$. Un effet secondaire de ses exclusions est la suppression ultime des fausses valeurs. Considérons une fausse clé (v, e) avec une valeur d'écart e minimum. Le porteur de cette clé, p modifiera nécessairement $p.\mathcal{K}$ car $v \neq \mathcal{I}(p)$ et aucun de ses précédesseurs n'a $(v, e - 1)$ comme valeur de

clé. Cependant, la fausse valeur v peut être propagée aux descendants de p avant que p ne change de valeur. Lors de cette propagation, l'écart associé à v augmente. Dans le pire des cas, l'écart atteindra α avant que v ne soit exclu et finalement disparaisse.

Ainsi, `C-MAI` stabilise en $O(\alpha)$ rondes. Nous définissons ensuite notre algorithme d'élection comme la composition $\texttt{E} \circ \texttt{A} \circ \texttt{MI}$, où `E`, `A` et `MI` sont les trois instances de `C-MAI` décrites ci-dessous.

Instance `MI`. Dans cette instance, l'entrée de chaque processus est sa propre identité. Ainsi, `MI` permet à chaque processus p de calculer la plus petite identité parmi ses ancêtres (p inclus). Puisque cette instance est la plus prioritaire et que les autres instances ne peuvent pas modifier ses variables, `MI` stabilise dans les $O(\alpha)$ premières rondes. Un exemple des valeurs finales calculées par `MI` est donné dans la configuration (a) de la figure 1. Par définition, dans l'unique composante source, les ancêtres de chaque processus correspondent à l'ensemble des processus de la composante. Donc, ultimement la sortie de chaque processus de la composante source sera l'identité la plus petite d'un processus de la composante, notée mid. Le but des deux autres instances est alors d'assurer que tous les processus du réseau élisent finalement l'identité mid.

Instance `A`. Dans cette instance, chaque processus vérifie si tous ses ancêtres ont choisi la même identité dans `MI`. Pour ce faire, l'entrée de chaque processus p est une fonction booléenne qui teste si p a calculé dans `MI` la même sortie que ses prédécesseurs. Ainsi, cette entrée deviendra constante après la stabilisation de `MI` et, à partir de cette entrée, la sortie calculée par p dans `A` vaudra ultimement 1 si et seulement si tous les ancêtres de p ont calculé la même identité dans `MI`. Puisque, par définition tous processus possèdent au moins un ancêtre dans la composante source, $O(\alpha)$ rondes après la stabilisation de `MI`, la sortie de chaque processus sera 1 si et seulement si sa sortie dans `MI` est mid ; *cf.*, configuration (b) de la figure 1 pour un exemple de valeurs finales calculées par `A`.

Instance `E`. Dans cette dernière instance, l'entrée de chaque processus est le couple de valeur (b, i) où b est l'inverse du bit de sortie calculé dans `A` et i est l'identité calculée dans `MI`. Grâce à l'inversion de la sortie de `A`, après la stabilisation de `A`, l'entrée la plus petite (dans l'ordre lexicographique) sera $(0, mid)$. En particulier, l'entrée de chaque processus de la composante source sera $(0, mid)$. Puisque chaque processus a au moins un ancêtre dans l'unique composante source, après $O(\alpha)$ rondes supplémentaires, la sortie de chaque processus dans `E` sera $(0, mid)$: l'identité élue sera mid, comme illustré dans la configuration (c) dans la figure 1.

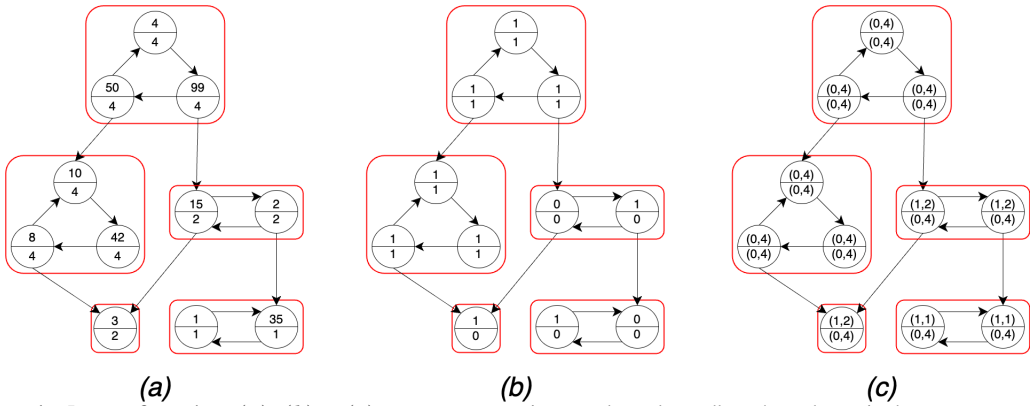

FIGURE 1 : Les configurations (a), (b) et (c) montrent respectivement les valeurs d'entrée et de sortie de `MI`, `A` et `E` après la stabilisation de notre algorithme d'élection. Les rectangles rouges avec bords arrondis délimitent les composantes fortement connexes. Pour chaque configuration, l'entrée de chaque processus est donnée dans la partie supérieure du cercle et la sortie dans la partie inférieure.

Références

[1] Y. Afek and A. Bremler-Barr. Self-stabilizing unidirectional network algorithms by power supply. *Chic. J. Theor. Comp. Sci.*, 1998.

[2] K. Altisen, A. Cournier, G. Defalque, and S. Devismes. Self-stabilizing synchronous unison in directed networks. In *ICDCN'23*.

[3] S. Bernard, S. Devismes, M. Gradinariu Potop-Butucaru, and S. Tixeuil. Optimal deterministic self-stabilizing vertex coloring in unidirectional anonymous networks. In *IPDPS'09*.

[4] B. Chen, H. Yu, Y. Zhao, and P. B. Gibbons. The cost of fault tolerance in multi-party communication complexity. *J. ACM*, 2014.

[5] S. Huang and T. Liu. Self-stabilizing 2^{m}-clock for unidirectional rings of odd size. *Distributed Comput.*, 1999.