Stabilisation progressive

Karine Altisen, Stéphane Devismes, Anaïs Durand, Franck Petit

Stabilisation progressive †

Karine Altisen[1], Stéphane Devismes[1], Anaïs Durand[2] et Franck Petit[3]

[1] *VERIMAG UMR 5104, Université Grenoble Alpes, France*
[2] *Technion - Israel Institute of Technology, Haifa, Israel*
[3] *LIP6 UMR 7606, INRIA, UPMC Sorbonne Universités, France*

Cet article est un résumé étendu de [1] dans lequel nous nous intéressons à des réseaux pouvant subir des changements topologiques transitoires. Nous proposons une nouvelle spécialisation de l'autostabilisation adaptée à ce type de réseau : la *stabilisation progressive*. Un algorithme est progressivement stabilisant sous hypothèse de (τ,ρ)-dynamicité s'il est autostabilisant et satisfait la propriété supplémentaire suivante : après au plus τ pas dynamiques vérifiant la condition ρ et se produisant à partir d'une configuration légitime, l'algorithme converge rapidement vers une configuration où une spécification plus faible est satisfaite ; puis il continue à converger progressivement vers des configurations où des spécifications de plus en plus fortes sont vérifiées, et ce, jusqu'à retrouver une configuration légitime vérifiant la spécification initiale du problème. Nous illustrons cette nouvelle propriété en proposant un algorithme progressivement stabilisant de synchronisation d'horloges.

Mots-clefs : Autostabilisation, réseaux dynamiques, unisson.

1 Introduction

L'*autostabilisation* est un paradigme permettant de concevoir des systèmes distribués tolérants aux *fautes transitoires*. Les fautes transitoires sont rares, de durée finie et affectent le contenu du composant du réseau où elles surviennent. La corruption d'une mémoire locale d'un processus ou du contenu d'un message en transit sont deux exemples de fautes transitoires. Après un nombre fini de fautes transitoires et en supposant que ces fautes n'aient pas affecté le code de l'algorithme, un système autostabilisant retrouve de lui-même et en temps fini une configuration dite *légitime* à partir de laquelle son comportement est conforme à sa spécification. Nous considérons ici des *systèmes dynamiques*, c'est-à-dire, des réseaux dans lesquels des changements topologiques peuvent survenir par l'ajout ou la suppression de processus ou de liens de communication. Lorsque les changements topologiques sont détectables (en temps fini et localement par les processus) et de fréquence suffisamment faible, ces évènements peuvent être considérés comme de nature transitoire. En conséquence, tout algorithme autostabilisant conçu pour des topologies quelconques retrouve naturellement un comportement correct après un nombre fini de tels changements topologiques. On appelle *phase de stabilisation* la période nécessaire au système pour qu'il retrouve une configuration légitime après des fautes transitoires. La durée maximale de cette phase est appelée *temps de stabilisation*. Pour beaucoup de problèmes, le temps de stabilisation dépend de paramètres globaux au réseau, comme sa taille ou son diamètre. C'est en grande partie une conséquence du fait que le temps de stabilisation est une mesure de complexité au pire cas. Or, le pire des cas correspond souvent à une exécution dégénérée et donc peu probable. *A contrario*, dans beaucoup de cas, les fautes transitoires sont éparses et leurs effets sur le système sont superficielles. Dans cette optique, plusieurs variantes de l'autostabilisation garantissant un temps de convergence réduit dans certains cas favorables ont été proposées. En particulier, la *superstabilisation* [2] est une variante de l'autostabilisation avec exige une convergence rapide lorsque les fautes transitoires sont limitées à un seul changement topologique (l'ajout ou la supression d'un lien ou d'un processus). En effet, un algorithme *superstabilisant* est un algorithme autostabilisant qui assure deux propriétés supplémentaires lorsqu'un seul changement topologique intervient depuis une configuration légitime : après un tel évènement, le système doit (1) retrouver rapidement (généralement, en un nombre de rondes constant) une configuration légitime, (2) tout en garantissant qu'un prédicat, appelé prédicat de *passage*, est préservé.

† Il a été financé en partie par les projets ANR DESCARTES (ANR-16-CE40-0023) et ESTATE (ANR-16-CE25-0009).

Dans cet article, nous suivons une approche similaire à la superstabilisation en considérant des systèmes dynamiques où les changements topologiques sont transitoires. Pour ces systèmes, nous proposons une nouvelle spécialisation de l'autostabilisation appelée *stabilisation progressive sous hypothèse de* (τ,ρ)-*dynamicité*. Un algorithme est progressivement stabilisant sous hypothèse de (τ,ρ)-dynamicité s'il est autostabilisant et satisfait la propriété supplémentaire suivante : après au plus τ pas dynamiques [‡] vérifiant la condition ρ et se produisant à partir d'une configuration légitime, l'algorithme converge rapidement vers une configuration où une spécification plus faible (mais offrant tout de même un minimum de qualité de service) est satisfaite ; puis il continue à converger progressivement vers des configurations où des spécifications de plus en plus fortes sont vérifiées, et ce, jusqu'à retrouver une configuration légitime de la spécification (forte) du problème initial. Bien entendu, une fois que la spécification initiale est vérifiée, l'algorithme peut à nouveau assurer une convergence progressive en cas de nouvelles occurences d'au plus τ pas dynamiques de type ρ. Naturellement, la stabilisation progressive fait sens uniquement si (1) les spécifications intermédiaires sont intéressantes et (2) les convergences vers ces spécifications sont rapides.

Pour illustrer cette nouvelle propriété, nous nous intéressons à trois variantes d'un problème de synchronisation, respectivement appelées *l'unisson fort*, *l'unisson faible* et *l'unisson partiel*. Dans chacun de ces problèmes, chaque processus doit maintenir une horloge locale. Nous nous restreignons à l'étude d'horloges périodiques, chaque horloge étant une variable entière de domaine $\{0,...,\alpha-1\}$ où $\alpha \geq 2$ est appelé la période. Chaque processus doit régulièrement incrémenter son horloge (modulo α) tout en respectant une propriété de sûreté. La sûreté de l'unisson fort impose qu'au plus deux valeurs consécutives d'horloges doivent être présentes à tout moment dans le réseau. L'unisson faible contraint les valeurs d'horloges de voisins à différer d'au plus un incrément. L'unisson partiel est, quant à lui, similaire à l'unisson faible, mais dédié aux réseaux dynamiques puisqu'il exonère de sûreté les nouveaux processus entrant dans le réseau jusqu'à ce qu'ils aient effectué leur premier pas de calcul, la sûreté de l'unisson faible s'appliquant aux autres processus. Nous proposons un algorithme stabilisant progressivement convergeant vers ces spécifications, appelé $\mathcal{DSU}$.

Plan. Dans la section 2, nous décrivons le modèle pour lequel est écrit $\mathcal{DSU}$. Par manque de place, nous présentons uniquement les idées principales de $\mathcal{DSU}$ dans la section 3. Enfin, nous concluons en section 4.

2 Modèle

Nous considérons des réseaux dynamiques bidirectionnels asynchrones, initialement connexes, de n processus où chaque processus peut communiquer directement avec un sous-ensemble d'autres processus appelés *voisins*, ce sous-ensemble variant au cours du temps. Les processus sont anonymes, cependant chaque processus peut distinguer ses voisins *via* un numéro de canal local. Les processus communiquent à l'aide de *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 voisins courants. Les variables d'un processus définissent son état. Une configuration est définie par la topologie du système et l'état des processus. L'exécution d'un algorithme est une suite d'étapes atomiques de calculs de la forme $\gamma_i \mapsto \gamma_{i+1}$ où γ_i et γ_{i+1} sont respectivement les configurations du système avant et après l'étape atomique. Lors de l'étape $\gamma_i \mapsto \gamma_{i+1}$, soit la topologie du système est modifiée, soit des processus changent d'états, soit les deux. Dans les deux derniers cas, il existe des processus — dits *activables* — pouvant modifier leurs états, et un sous-ensemble non-vide de ces processus est activé. Chaque processus activé réalise de manière atomique (1) la lecture de ses propres variables et de celles de ses voisins dans γ_i et (2) l'écriture de son nouvel état.

Pour mesurer le temps de stabilisation, nous utilisons la notion de *ronde* : la première ronde d'une exécution termine dès lors que tous les processus continûment présents et activables depuis le début de l'exécution ont été activés, la seconde ronde commence alors et termine selon les mêmes règles.

3 Unisson progressivement stabilisant

Notre algorithme utilise trois paramètres : α, μ et β. Ces paramètres doivent vérifier les conditions suivantes : $\alpha \geq 2$, $\mu \geq \max(2,N)$ (où N est une borne supérieure sur le nombre de processus du réseau), β doit être divisible par α et $\beta > \mu^2$. Chaque processus p maintient deux horloges locales : une horloge dite *externe*, $p.c \in \{0,...,\alpha-1\} \cup \{\perp\}$, et une horloge dite *interne*, $p.t \in \{0,...,\beta-1\}$.

‡. Nous appelons *pas dynamique* un pas du système où des changements topologiques interviennent.

L'algorithme commence par faire converger les horloges internes des processus vers un unisson faible. Les valeurs d'horloges externes sont calculées à partir des précédentes pour obtenir l'unisson fort. Pour cela, nous utilisons la notion de *délai* : le délai $d_\beta(x,y)$ entre deux valeurs entières x et y est égal à $\min((x-y) \bmod \beta, (y-x) \bmod \beta)$. Nous utilisons aussi la relation $\preceq_{\beta,\mu}$ entre deux entiers x et y définie par $x \preceq_{\beta,\mu} y \equiv ((y-x) \bmod \beta) \leq \mu$.

Quand un processus p détecte que l'unisson faible est localement correct, c'est-à-dire quand le délai entre son horloge interne et celles de ses voisins est au plus 1, il peut incrémenter celle-ci modulo β si elle est en retard ou égale à celle de tous ses voisins (autrement dit, si $p.t \preceq_{\beta,\mu} q.t$, pour tout voisin q de p). *A contrario*, quand p détecte localement que l'unisson faible n'est pas vérifié, nous avons deux cas : soit le délai entre p et ses voisins est inférieur ou égal à μ et dans ce cas, p se comporte comme précédemment ; soit l'horloge interne de p est trop en retard vis-à-vis de celles de ses voisins, (*i.e.*, délai $> \mu$), et $p.t$ est réinitialisée à 0. Ce mécanisme permet au système de stabiliser vers une configuration où toutes horloges internes vérifient la sûreté de l'unisson faible.

Pour que les horloges externes stabilisent vers un unisson fort, l'horloge externe de chaque processus p est recalculée chaque fois que celui-ci modifie son horloge interne. La valeur de $p.c$ est maintenue égale à $\lfloor \frac{\alpha}{\beta} p.t \rfloor$: $p.c$ représente la valeur de $p.t$ normalisée entre 0 et $\alpha - 1$ et les paramètres α, β et μ sont contraints de telle façon qu'un écart d'au plus $N-1$ entre deux horloges internes devient un écart d'au plus 1 entre leurs deux horloges externes associées. Ainsi, notre algorithme est dans une configuration légitime lorsque tout processus p vérifie $p.c \neq \perp$ et $p.c = \lfloor \frac{\alpha}{\beta} p.t \rfloor$ et $d_\beta(p.t, q.t) \leq 1$, pour tout voisin q (la valeur particulière $\perp$ sera justifiée plus tard). Dans ces conditions, le délai entre les horloges internes de deux processus quelconques est borné par $n-1$ (avec n, le nombre de processus dans le réseau), donc le délai entre les horloges externes de deux processus quelconques est borné par 1 : les horloges externes vérifient l'unisson fort.

$\mathcal{DSU}$ est un algorithme stabilisant progressivement sous hypothèse de (1, BULCC)-dynamicité. Un pas dynamique satisfait la condition BULCC si après ce pas (1) le réseau contient au plus N processus, (2) reste connexe et (3) si $\alpha > 3$ alors tout nouveau processus entrant est voisin d'au moins un processus présent avant le pas dynamique, à moins que tous les processus présents avant le pas dynamique n'aient disparu. La condition (1) est nécessaire pour avoir une solution avec des horloges bornées. Nous avons prouvé que la condition (2) est nécessaire quelle que soit la période. Enfin, nous avons prouvé que la condition (3) est nécessaire à partir de $\alpha > 5$ (pour les périodes 4 et 5, nous avons exhibé des cas pathologiques mais nous n'avons pas pu prouver de condition de nécessité). Notez qu'un pas dynamique de type BULCC peut contenir plusieurs changements topologiques. Nous étudions maintenant plusieurs scénarios de pas dynamiques $\gamma_i \mapsto \gamma_{i+1}$ vérifiant la condition BULCC où γ_i est une configuration légitime de notre algorithme.

Ajout de liens. Supposons que $\gamma_i \mapsto \gamma_{i+1}$ ne contienne que des ajouts de liens. L'ajout d'un lien peut rompre la sûreté de l'unisson faible sur les horloges internes comme le montre la figure 1. Cependant le délai entre deux horloges internes de processus nouvellement voisins reste borné par $n-1$. Ainsi, l'ajout de lien ne provoque pas de réinitialisation. De plus, puisque les incréments sont contraints par le voisinage, les ajouts de liens n'ont pour effet que de renforcer ces contraintes. Ainsi, le délai entre deux horloges internes de processus arbitrairement éloignés reste borné par $n-1$ et, en conséquence, le délai entre leurs horloges externes reste borné par 1 : la sûreté de l'unisson fort reste satisfaite.

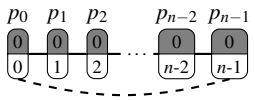


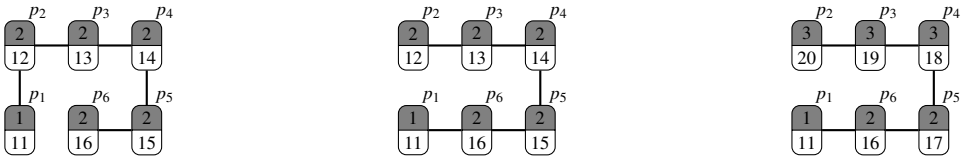
FIGURE 1: Ajout d'un lien.

Suppression de processus et de liens. Supposons que $\gamma_i \mapsto \gamma_{i+1}$ ne contienne que des suppressions de liens et de processus. Par définition de BULCC, le réseau reste connexe. Ainsi, les contraintes entre les processus toujours voisins sont maintenues : le délai entre les horloges internes de deux voisins reste borné par 1 et donc le délai entre les horloges internes (resp. externes) de deux processus quelconques reste borné par $n-1$ (resp. 1), ainsi l'unisson fort reste satisfait.

Ajout de liens couplé à la suppression de processus et de liens. Supposons maintenant que $\gamma_i \mapsto \gamma_{i+1}$ contienne à la fois des ajouts de liens et des suppressions ou ajouts de processus. Dans ce cas, l'unisson fort peut être violé après quelques pas à partir de γ_{i+1}. La figure 2 montre un tel exemple. L'ajout du lien $\{p_1, p_6\}$ crée entre les deux nouveaux voisins un délai plus grand que 1 (ici 5). De plus, la suppression du lien

$\{p_1, p_2\}$ relâche les contraintes sur l'horloge interne de p_2 : p_2 peut maintenant incrémenter sans attendre p_1. En conséquence, après quelques pas, le système peut atteindre la configuration décrite en figure 2(c), où le délai entre les horloges internes de p_1 et p_2 est maintenant de 9. Puisque la valeur des horloges externes est calculée à partir des horloges internes, l'unisson fort est violé, lui aussi : $p_1.c = 1$, $p_6.c = 2$ et $p_2.c = 3$, il existe trois valeurs différentes dans les horloges externes.

Cependant, dans le pire des cas, le délai entre deux horloges internes de processus voisins reste borné par $n-1$, donc (1) ce délai maximum reste borné par $n-1$ dans les pas suivants car aucune horloge interne n'est réinitialisée et (2) le système converge vers une configuration où ce délai maximum est au plus 1. Ainsi, le délai entre deux horloges externes de deux voisins reste d'au plus 1 : les horloges externes ne vérifient pas l'unisson fort, mais continuent à vérifier l'unisson faible tout en reconvergeant vers l'unisson fort.



(a) Configuration légitime.

(b) Le lien $\{p_1, p_2\}$ disparait et le lien $\{p_1, p_6\}$ est créé.

(c) Après plusieurs pas, l'unisson fort n'est plus respecté.

FIGURE 2: Exemple d'ajout et suppression de liens : $\mu = 6$, $\alpha = 7$ et $\beta = 42$.

Ajout de processus. Supposons pour finir que des processus (notés p) entre dans le système : dans ce cas, p initialise $p.t$ à 0 et $p.c$ à $\perp$. Par définition, et d'après les discussions précédentes, l'unisson partiel est immédiatement satisfait. Pour assurer que l'unisson faible soit réalisé en au plus une ronde, chaque processus entrant, p, est activable pour affecter une valeur d'horloge à $p.c$. Puisque la condition `BULCC` est vérifiée, p a nécessairement au moins un voisin q tel que $q.c \neq \perp$. L'horloge $p.t$ est alors affectée à la valeur minimum des voisins q vérifiant $q.c \neq \perp$ et $p.c$ est initialisé en fonction de la valeur prise par $p.t$ comme expliqué précédemment. Notez que notre algorithme doit rester autostabilisant, ainsi nous devons éviter un interblocage dans le cas où, par exemple, le pas dynamique ne vérifie pas `BULCC` : nous affectons à la fois $p.t$ et $p.c$ à 0 dans le cas où aucun voisin q de p ne vérifie $q.c \neq \perp$.

4 Conclusion

$\mathcal{DSU}$ est progressivement stabilisant car, après un pas dynamique de type `BULCC` depuis une configuration légitime de l'unisson fort, il satisfait immédiatement la spécification de l'unisson partiel, puis converge en au plus 1 ronde vers une configuration légitime de l'unisson faible et enfin converge en au plus $(\mu+1)\mathcal{D}_1 + 1$ rondes supplémentaires (où $\mathcal{D}_1$ est le diamètre du réseau après le pas dynamique) vers une configuration légitime de l'unisson fort. Nous rappelons que $\mathcal{DSU}$ est aussi autostabilisant, donc après tout autre type de fautes transitoires (par exemple, plusieurs pas dynamiques), il stabilise vers une configuration légitime de l'unisson fort, mais sans garantie supplémentaire.

Comme le montre notre cas d'étude, la stabilisation progressive est une généralisation de la superstabilisation. Par exemple, le fait que l'unisson partiel soit immédiatement vérifié après le pas dynamique peut être assimilé à un prédicat de passage. L'intérêt de notre approche est donc d'obtenir des solutions plus efficaces que l'autostabilisation dans des cas où il est difficile d'obtenir la superstabilisation. Par exemple, très peu d'algorithmes superstabilisant ont été proposés pour résoudre des spécifications non-silencieuses (à notre connaissance il existe des algorithmes superstabilisants pour l'exclusion mutuelle mais pour des topologies en anneau uniquement [3]). Ce manque vient sans doute de la difficulté d'obtenir des solutions réalisant une propriété aussi forte, ce qui fait de la stabilisation progressive une alternative intéressante.

Références

[1] K. Altisen, S. Devismes, A. Durand, and F. Petit. Gradual stabilization under τ-dynamics. In *Euro-Par 2016*, pages 588–602, 2016.

[2] S. Dolev and T. Herman. Superstabilizing protocols for dynamic distributed systems. *Chicago J. Theor. Comput. Sci.*, 1997, 1997.

[3] Y. Katayama, E. Ueda, H. Fujiwara, and T. Masuzawa. A latency optimal superstabilizing mutual exclusion protocol in unidirectional rings. *J. Parallel Distrib. Comput.*, 62(5) :865–884, 2002.