Coordination de comités instantanément stabilisante †

Borzoo Bonakdarpour[1], Stéphane Devismes[2] et Franck Petit[3]

[1] *Dpt. of Electrical and Computer Engineering, University of Waterloo, Ontario, Canada*
[2] *VERIMAG UMR 5104, Université Joseph Fourier, Grenoble*
[1] *LIP 6 UMR 7606, INRIA REGAL, UPMC Sorbonne Universités, Paris*

Nous nous intéressons à la *coordination de comités* qui consiste à réaliser des rendez-vous de synchronisation entre des groupes de processus. Nous augmentons la définition originale de ce problème, notamment avec des propriétés d'équité et de concurrence. Nous montrons qu'il est impossible d'assurer à la fois l'*équité* et la *concurrence maximale* pour ce problème. Ensuite, nous proposons deux algorithmes instantanément stabilisants pour ce problème. Le premier maximise la concurrence sans garantir l'équité, alors que le second assure l'équité mais ne garantit pas la concurrence maximale. Nous démontrons que le second algorithme reste cependant efficace en terme de niveau de concurrence.

Keywords: Algorithmes répartis, stabilisation instantanée, coordination de comités

1 Introduction

La *coordination de comités* [3] est un problème de synchronisation dans les systèmes distribués qui est généralement présentée par l'allégorie suivante :

> *« Des professeurs d'une université sont répartis en plusieurs comités. Chaque comité est composé d'un ensemble fixé de plusieurs professeurs (au moins deux). De temps à autre, un professeur peut décider de réunir un des comités auquel il appartient. Il se met alors en attente jusqu'à ce qu'une réunion d'un de ses comités puisse se tenir. Toute réunion dure un temps fini. (1) Une réunion d'un comité peut se tenir seulement si tous les membres de ce comité sont en attente. (2) Des réunions de deux comités différents peuvent se dérouler simultanément seulement si elles n'ont aucun membre en commun. (3) Enfin, si tous les membres d'un comité sont en attente, alors au moins un membre finit par participer à une réunion. »*

Les propriétés (1), (2) et (3) sont respectivement appelées les propriétés de *synchronisation*, d'*exclusion* et de *progrès*. Dans le contexte d'un système réparti, les professeurs représentent les *processus* et les *comités* des points de synchronisation (ou rendez-vous).

Dans cet article nous proposons des algorithmes *instantanément stabilisants* résolvant la coordination de comités. La stabilisation instantanée [2] est une technique générale permettant de concevoir des algorithmes tolérants efficacement les pannes transitoires. En effet, après un nombre fini de telles fautes (*e.g.*, corruption mémoire, perte de message, ...), un algorithme instantanément stabilisant retrouve immédiatement un comportement correct, sans intervention extérieure. Les solutions instantanément stabilisantes que nous proposons ont d'autres propriétés intéressantes que nous détaillerons par la suite. Ces propriétés sont liées à l'équité, la concurrence et à la synchronisation.

2 Propriétés

Nous présentons maintenant les différentes propriétés que nous souhaitons intégrer à nos algorithmes. Nous commençons par donner quelques précisions sur la notion de stabilisation instantanée.

† Cet article est un résumé étendu de [1].

Stabilisation instantanée. La *stabilisation instantanée* est souvent comparée à l'auto-stabilisation. Un algorithme auto-stabilisant, après un nombre fini de pannes transitoires, retrouve un comportement correct en un temps fini alors qu'un algorithme instantanément stabilisant retrouve immédiatement un comportement correct. Cependant, un algorithme instantanément stabilisant n'est bien sûr pas insensible aux pannes transitoires. En fait, il garantit uniquement que toute tâche démarrée après la dernière panne sera exécutée correctement à condition qu'aucune nouvelle panne ne se produise. Ainsi, aucune garantie n'est assurée pour les tâches exécutées toutes ou parties durant une période de pannes transitoires.

Par exemple, pour la coordination de comités, toute réunion démarrée après la dernière panne vérifiera toutes les propriétés de la spécification. En revanche, rien ne sera garanti pour les réunions démarrées pendant les fautes, exceptée qu'elles n'interféreront pas avec les réunions démarrées après les fautes.

La discussion en deux phases. Dans les solutions proposées jusqu'à présent, les réunions sont supposées être des évènements atomiques : une fois que les membres d'un comité sont en attente, la réunion démarre et termine pour tous simultanément. Donc, un mécanisme de synchronisation implicite est supposé et ce mécanisme impose que les processus rapides attendent les processus lents pour réaliser la terminaison de la réunion ensemble.

Cette hypothèse est trop forte pour des systèmes supposés asynchrones. Ainsi, nous levons cette hypothèse en supposant qu'une réunion puisse se tenir jusqu'à ce qu'un de ses membres décide de manière autonome de quitter la réunion. Cependant, cela autorise un membre à quitter une réunion immédiatement après son ouverture (si par exemple ce membre n'est pas intéressé par une réunion de ce comité particulier) sans qu'il ait échangé la moindre information avec les autres membres. Cette situation n'étant pas souhaitable, nous imposons que tout membre reste un temps minimum dans la réunion avant de la quitter. Ainsi, les « discussions » lors de la réunion se tiennent en *deux phases* : une phase minimale où chaque membre à l'occasion d'échanger avec les autres, suivie d'une phase libre où la réunion continue jusqu'à ce qu'un des membres décide unilatéralement de quitter la réunion.

Équité. La définition originale de la coordination de comités n'impose pas d'équité entre les professeurs, ni même entre les comités. Certains travaux, hors du cadre de la tolérance aux fautes, se sont penchés sur ce problème, notamment [4, 5].

Dans [4], Joung s'intéresse à l'*équité* entre les professeurs : tout professeur attendant infiniment souvent une réunion, d'un des comités dont il est membre, finit par participer à une réunion (pas nécessairement une réunion du comité qu'il souhaite). Joung montre qu'il est impossible de garantir l'équité entre professeurs lorsque les décisions d'attendre une réunion de comité sont prises de manière strictement autonome, asynchrone et non-prédictible. En particulier, dans ce cas, un professeur peut ne plus jamais souhaiter participer à une réunion de comité sans que les autres en soient informés.

Kumar [5] contourne ce problème en introduisant une hypothèse supplémentaire : il suppose que chaque professeur finit toujours par se mettre en attente d'une nouvelle réunion. Chaque fois que nous considérerons la propriété d'équité, nous ferons la même hypothèse.

Concurrence maximale. Lorsque plusieurs réunions sont prêtes à commencer (tous leurs membres sont en attente), il est souhaitable de démarrer le plus de réunions possibles sans violer la propriété d'exclusion. Or, jusqu'à présent aucune contrainte de concurrence n'avait été défini pour ce problème. Ceci a mené à des solutions très centralisées, où par exemple les réunions sont réalisées séquentiellement grâce à une circulation de jeton.

Il est donc souhaitable de contraindre le problème pour que le nombre de réunions se déroulant simultanément soit le plus grand possible. Nous avons appelée cette propriété la *concurrence maximale*. Informellement, elle impose que si au moins une nouvelle réunion peut démarrer, alors au moins une réunion commence en temps fini, même si aucune réunion en cours ne termine. Ainsi, si toutes les réunions durent un temps arbitrairement long, d'autres réunions démarrent jusqu'à ce que le système soit saturé.

Il serait intéressant d'obtenir des solutions à la fois équitable et permettant une concurrence maximale, dans le cas bien sûr où l'équité est réalisable (c'est-à-dire, en supposant que chaque professeur finit toujours par se mettre en attente d'une nouvelle réunion). Cependant, nous avons prouvé que ces deux propriétés

étaient orthogonales, c'est-à-dire, qu'il n'est pas possible de les vérifier simultanément [‡].

Ainsi, nous avons proposé deux solutions réalisant ces deux propriétés séparément. Toutes deux sont instantanément stabilisantes. Le premier algorithme vérifie les propriétés de discussion en deux phases et de concurrence maximale. L'autre garantit les propriétés de discussion en deux phases et d'équité. Pour ce dernier, nous essayons d'assurer la meilleur concurrence possible à défaut d'être maximale.

3 Algorithmes instantanément stabilisants

Nous supposons que deux processus (professeurs) peuvent communiquer directement ensemble s'il existe au moins un comité où ils sont tous les deux membres, dans ce cas les deux professeurs sont dits *voisins*. Nos algorithmes sont écrits dans *le modèle à états*. Dans ce modèle, les communications sont réalisées *via* des variables 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. Les variables d'un processus définissent son état. Les exécutions de nos algorithmes sont des suites d'étapes atomiques : à chaque étape, s'il existe des processus souhaitant modifier leurs états, un sous-ensemble 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 sans hypothèse d'équité particulière.

Nos algorithmes utilisent les identités des processus. De plus, ils utilisent tous deux une circulation de jeton pour assurer le progrès dans le premier cas et l'équité dans le second. Ces deux propriétés étant des propriétés de vivacité (et pas de sûreté), nous pouvons utiliser n'importe quelle circulation de jeton auto-stabilisante [§]. Nous donnons maintenant les idées principales de nos algorithmes. Pour plus de détails voir [1].

Algorithme réalisant la concurrence maximale. Dans notre algorithme, chaque processus maintient une variable d'état $S_p \in \{\mathsf{libre}, \mathsf{recherche}, \mathsf{prêt}, \mathsf{ok}\}$, un booléen T_p et un pointeur de comité P_p. Nous expliquons maintenant comment nous réalisons la coordination de comités en fonction de l'état des processus.

Etat libre. Lorsqu'un processus p n'est pas demandeur de réunion, il vérifie $S_p = \mathsf{libre}$. Lorsqu'il le souhaite, il se met en attente de réunion en changeant son état de libre à recherche. Dans le même temps, il initialise son pointeur de comité P_p à $\perp$.

Etat recherche. Ensuite, p se met à la recherche d'un comité disponible, prêt à se réunir. p montre son intérêt pour un comité dont tous les processus q vérifient $S_q = \mathsf{recherche}$ en le désignant avec P_p.

Pour obtenir un accord (local) sur les réunions de comité à démarrer, nous utilisons des priorités basées sur la circulation de jeton. Lorsqu'un processus en attente est le plus prioritaire dans son voisinage, il sélectionne un comité dont tous les membres sont en attente (si un tel comité existe), puis s'y tient. Les processus non prioritaires en attente sélectionnent le comité choisi par leur voisin en attente le plus prioritaire.

Nous expliquons maintenant notre système de priorité : Chaque processus informe ses voisins sur le fait qu'il détient un jeton ou pas grâce au booléen T_p. Lorsqu'un processus détient un jeton, il est plus prioritaire que ses voisins pour choisir une réunion à commencer. Dans le cas où plusieurs processus détiennent un jeton (uniquement durant la phase de stabilisation de la circulation de jeton auto-stabilisante), le processus le plus prioritaire est celui ayant l'identifiant le plus grand parmi les détenteurs de jeton.

Un processus relâche son jeton dans deux cas : (1) quand il quitte une réunion ou (2) quand il est dans une situation où il n'est pas sûr de pouvoir un jour démarrer une réunion, c'est-à-dire, dans chacun des comités où il est membre au moins un processus q vérifie $S_q \neq \mathsf{recherche}$. Notez que c'est à cause de ce dernier cas que notre algorithme n'est pas équitable.

Ensuite, pour garantir un niveau de concurrence maximale, nous devons aussi permettre aux comités de se réunir quand tous leurs membres q vérifient $S_q = \mathsf{recherche}$ et qu'il n'y a aucun détenteur de jeton vérifiant $S = \mathsf{recherche}$ dans le voisinage. Dans ce dernier cas, le processus le plus prioritaire est celui en attente ayant l'identité la plus grande.

---

‡. *cf.*, `http://www.ece.uwaterloo.ca/~bbonakda/ipdps11.pdf` pour les preuves et définitions formelles.

§. Les algorithmes de circulation de jeton sont généralement écrits pour des réseaux enracinés, ainsi nous devons composer notre circulation de jeton avec un algorithme d'élection de leader pour obtenir une circulation de jeton pour réseaux identifiés.

Etat prêt. Une fois que tous les processus d'un comité vérifient $S =$ recherche et désignent ce comité avec leur pointeur, la réunion peut être démarrée. Chaque processus du comité change son état de recherche à prêt. La réunion commence dès que tous les membres ont changé leur état à prêt.

Etat ok. Ensuite, chaque processus exécute la première phase de la discussion puis change son état à ok. Une fois que tous les processus ont exécuté la première phase de discussion (l'état de tous les processus est ok), la deuxième phase se déroule jusqu'à ce qu'un processus décide unilatéralement de terminer la réunion en la quittant. Dans ce cas, il change son état à libre, réinitialise son pointeur et relâche son jeton s'il en possède un.

Enfin, il faut noter que pour éviter tout interblocage, nous avons dû mettre en place des actions de correction. Lorsqu'un processus est dans un état ne pouvant être obtenu qu'après une faute, il réinitialise son pointeur et repasse à l'état recherche à moins qu'il soit dans l'état libre. Pour le détail de ses actions, voir [1].

Algorithme équitable. Notre second algorithme assure l'équité entre les processus, tout en assurant un niveau de concurrence intéressant (pour l'équité entre les comités, voir [1]). Cet algorithme reprend le principe des priorités du premier algorithme. Aussi, nous expliquons ici uniquement les différences.

Dans cet algorithme, chaque processus utilise les variables S_p, P_p et T_p avec la même sémantique que dans l'algorithme précédent. Cependant, cette fois-ci, un processus relâche un jeton uniquement lorsqu'il quitte une réunion.

Lorsqu'un processus p reçoit un jeton, il passe à l'état recherche et sélectionne un comité ε avec son pointeur. Cependant, contrairement à l'algorithme précedent, les membres du comité sélectionné ne sont pas forcément dans l'état recherche. Ensuite, p ne change pas son pointeur jusqu'à ce qu'une réunion du comité ε commence. Par hypothèse, les autres membres du comité ε finissent par être dans l'état recherche, et p ayant la plus grande priorité, ils choisissent ε. Puis, l'algorithme fonctionne comme le précédent.

Pour obtenir la meilleure concurrence possible, tous les processus qui ne sont pas membres du comité ε sélectionné par le détenteur du jeton ne doivent pas attendre après des membres d'ε même lorsque la réunion d'ε n'est pas encore commencée. En effet, ces processus ne participeront à aucune réunion avant de faire une réunion du comité ε. Ainsi, nous ajoutons une variable B qui indique si un processus est *bloqué* ou pas. Un processus est bloqué s'il n'est pas en réunion et qu'il est membre du comité ε sélectionné par le pointeur du détenteur du jeton. Ainsi, les processus qui ne sont pas membres du comité ε essaient de démarrer des réunions de comité n'ayant aucun membre bloqué.

Analyse du niveau de concurrence. Pour mesurer le niveau de concurrence, nous avons introduit une nouvelle mesure : le *degré de concurrence équitable*. Pour effectuer cette mesure, nous considérons un algorithme équitable et nous supposons que les réunions de comité organisées par cet algorithme durent indéfiniment. Ensuite, nous comptons le nombre de comités en réunion, une fois que le système arrive à saturation.

Soit $\mathcal{H}$ l'hypergraphe où les nœuds sont les processus et les hyper-arêtes représentent les comités. Nous avons montré qu'une fois le système saturé, les comités en réunion représentent un couplage maximal du sous-hypergraphe de $\mathcal{H}$ induit par ses sommets non-bloqués (alors qu'avec la concurrence maximale, nous aurions obtenu un couplage maximal de $\mathcal{H}$). Ainsi, le niveau de concurrence obtenue est proche de la concurrence maximale.

Références

[1] B. Bonakdarpour, S. Devismes, and F. Petit. Snap-stabilizing committee coordination. In *Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2011. to appear.

[2] A. Bui, A. K. Datta, F. Petit, and V. Villain. Snap-stabilization and PIF in tree networks. *Distributed Computing*, 20(1) :3–19, 2007.

[3] K. M. Chandy and J. Misra. *Parallel program design : a foundation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1988.

[4] Y.-J. Joung. On fairness notions in distributed systems : I. a characterization of implementability. *Information and Computation*, 166(1) :1–34, 2001.

[5] D. Kumar. An implementation of n-party synchronization using tokens. In *Distributed Computung Systems (ICDCS)*, pages 320–327, 1990.