Extension de la Hiérarchie de Herlihy aux Systèmes Multi-threads

Matthieu Perrin, Achour Mostéfaoui, Grégoire Bonin

Extension de la Hiérarchie de Herlihy aux Systèmes Multi-threads

Matthieu Perrin[1] et Achour Mostéfaoui[1] et Grégoire Bonin[1] †

[1]*LS2N, Université de Nantes, Faculté des Sciences et Techniques, 2 Chemin de la Houssinière, 44300, Nantes, France*

Dans les systèmes d'exploitation et les langages de programmation modernes adaptés aux architectures informatiques multicœurs, le parallélisme est abstrait par la notion de threads d'exécution. Les systèmes multi-threads ont deux spécificités majeures : 1) de nouveaux threads peuvent être créés dynamiquement pendant l'exécution, il n'y a donc pas de limite sur le nombre total de threads participant à une exécution continue ; 2) les threads ont accès à un mécanisme d'allocation de mémoire qui ne peut pas allouer des tableaux infinis. Cela rend difficile d'adapter certains algorithmes à des systèmes multi-threads, en particulier ceux qui attribuent un registre partagé à chaque processus.

Cet article, qui est une traduction résumée et vulgarisée de précédents travaux [PMB20], explore la puissance de coordination des objets partagés dans les systèmes multi-threads en étendant la hiérarchie de Herlihy pour tenir compte de ces contraintes. Il propose de subdiviser en cinq nouveaux degrés l'ensemble des objets de numéro de consensus infini, selon leur capacité à synchroniser un nombre borné, fini ou infini de processus, avec ou sans la nécessité d'allouer un tableau infini. Il présente ensuite un objet illustrant chaque degré proposé.

Mots-clefs : Allocation mémoire, linéarisabilité, modèles d'arrivée, numéro de consensus, système multi-thread, universalité, wait-freedom.

1 Introduction

Universalité Wait-free. En informatique séquentielle, la notion d'universalité est représentée par une machine de Turing capable de calculer tout ce qui est calculable. Les registres lire/écrire, objets de base d'une machine de Turing, sont donc des objets universels en calcul séquentiel. Dans le contexte des systèmes répartis, nous savons depuis 1985 et le fameux résultat d'impossibilité FLP, que le problème du consensus, dans lequel les processus doivent se mettre d'accord sur l'une des valeurs proposées, n'a pas de solution déterministe dans un système réparti asynchrone où même un seul processus peut tomber en panne [FLP85]. Cette impossibilité n'est pas due à la puissance de calcul des processus pris individuellement, mais plutôt à la difficulté de coordination entre les processus qui composent le système. Les problèmes de coordination et d'accord sont donc au cœur de la calculabilité dans les systèmes répartis.

Un système réparti peut être abstrait comme un ensemble de processus accédant simultanément à un ensemble d'objets concurrents. Les implémentations de ces objets utilisent des registres lire/écrire et des instructions matérielles. Une opération sur un tel objet a une durée ainsi qu'une date de début et une date de fin (si celle-ci se termine). Intuitivement, une ≪ bonne ≫ implémentation d'un objet concurrent doit satisfaire deux propriétés : une condition de cohérence et une condition de progression qui spécifient respectivement la pertinence des résultats retournés, et les garanties sur la vivacité.

La linéarisabilité [HW90] est une condition de cohérence. Elle garantit que toutes les opérations d'une histoire distribuée apparaissent comme si elles avaient été exécutées séquentiellement et instantanément : chaque opération apparaissant à un instant unique entre ses événements de début et de fin. Cela donne l'illusion aux processus d'accéder à un objet physique concurrent.

L'utilisation de verrous dans une implémentation peut provoquer un blocage dans un système où les processus peuvent tomber en panne. Interdire l'utilisation des verrous conduit à plusieurs conditions de

†Ce travail a été financé par le projet ANR français 16-CE25-0005 O'Browser.

progression, parmi lesquelles *wait-freedom* [Her91] et *lock-freedom* [HW90]. Alors que wait-freedom garantit que chaque opération se termine après un temps fini, lock-freedom garantit que, si un calcul s'exécute assez longtemps, au moins un processus progresse. Wait-freedom est donc plus forte que lock-freedom.

Une difficulté majeure du calcul réparti est que les implémentations linéarisables wait-free sont souvent coûteuses, voire impossibles. Le système doit être enrichi d'objets plus sophistiqués ou d'instructions spéciales matérielles. La puissance de coordination des objets, c'est-à-dire leur capacité ou incapacité à résoudre des problèmes de coordination, est donc importante pour la calculabilité dans les systèmes répartis. Dans [Her91], le consensus est prouvé universel. A savoir, tout objet ayant une spécification séquentielle a une implémentation wait-free utilisant uniquement des registres lire/écrire et un certain nombre d'instances de consensus. D'où l'idée d'attribuer à chaque objet un *numéro de consensus* représentant sa capacité à résoudre le consensus. Plus précisément, un objet a le numéro de consensus x s'il est universel dans un système asynchrone composé de x processus, mais pas dans un système composé de $x+1$ processus. Si aucune limite supérieure n'existe sur x, l'objet a un numéro de consensus infini.

Problématique. Cette dernière décennie, d'abord avec des systèmes pair-à-pair, puis avec des programmes multi-threads sur des machines multicœurs, l'hypothèse d'un système fermé avec un nombre fixe n de processus et où chaque processus connaît les identifiants de tous les processus est devenue trop restrictive. Dans les systèmes multi-threads, de nouveaux processus peuvent être créés et démarrés pendant l'exécution, donc bien que le nombre de processus à chaque instant soit fini, il n'y a pas de limite sur le nombre total de processus pouvant participer à des exécutions de longue durée.

Une autre spécificité des systèmes multi-threads doit être prise en compte. Les threads partagent un espace mémoire commun dans lequel ils peuvent allouer un nombre (virtuellement) illimité mais fini d'emplacements mémoire pour instancier des enregistrements de structures de données ou des tableaux finis. En particulier, lorsqu'aucune limite n'est connue sur le nombre de threads dans une exécution, attribuer un registre à chacun d'eux n'est pas anodin. Une conséquence de cet article est que le maintien de structures de données extensibles telles que les listes chaînées nécessite une puissance de coordination qui n'est pas nécessairement fournie par tous les objets qui ont un numéro de consensus infini.

Les deux aspects notés ci-dessus ont un impact important sur quels algorithmes peuvent être implémentés dans des systèmes multi-threads et quels algorithmes ne le peuvent pas, et donc sur le pouvoir de coordination des objets partagés : dans [AMW11], Afek, Morrison et Wertheim ont présenté un objet appelé *pile d'itérateurs* (noté `IStack`) qui a un numéro de consensus infini, mais ne peut pas être utilisé pour résoudre le consensus quand une infinité de processus peuvent rejoindre au fil du temps une exécution. Le présent article répond à la question suivante : comment comparer la puissance de coordination des objets partagés dans des systèmes multi-threads ?

Approche. En suivant la même approche que dans [Her91], nous proposons de comparer la puissance de coordination des objets partagés en fonction du nombre maximal de processus qu'ils sont capables de synchroniser, y compris dans des situations où l'ensemble des processus participants est initialement inconnu ou peut changer pendant une exécution. Plus précisément, nous différencions les modèles de calcul en fonction des restrictions sur l'arrivée des processus. Dans ces modèles, n'importe quel nombre de processus peut tomber en panne, mais de nouveaux processus peuvent également rejoindre le réseau lors d'une exécution. Lorsqu'un processus rejoint un tel système, il n'est pas connu des processus déjà en cours d'exécution. Quatre modèles d'arrivée sont distingués dans [Agu04] :

- Le modèle classique, M_1^n où le nombre n de processus est fixe et peut apparaître dans le code.
- Le modèle à arrivées bornées, M_1, dans lequel au plus n processus peuvent participer, où n est connu des processus au début de chaque exécution, mais peut varier d'une exécution à l'autre.
- Le modèle d'arrivées finies, M_2, dans lequel un nombre fini de processus participe à chaque exécution.
- Le modèle d'arrivées infinies, M_3, où de nouveaux processus peuvent arriver pendant toute l'exécution, bien que le nombre de processus présents dans le système reste fini à chaque instant.

De plus, l'impossibilité d'allouer des tableaux infinis est un facteur limitant important qui limite la puissance de calcul de certains objets dans les systèmes multi-threads. Nous étudions également la puissance de coordination des objets partagés selon la possibilité, ou non, d'allouer des tableaux infinis. Nous proposons donc la hiérarchie bidimensionnelle présentée sur la figure 1. Dans cette hiérarchie, les objets partagés sont

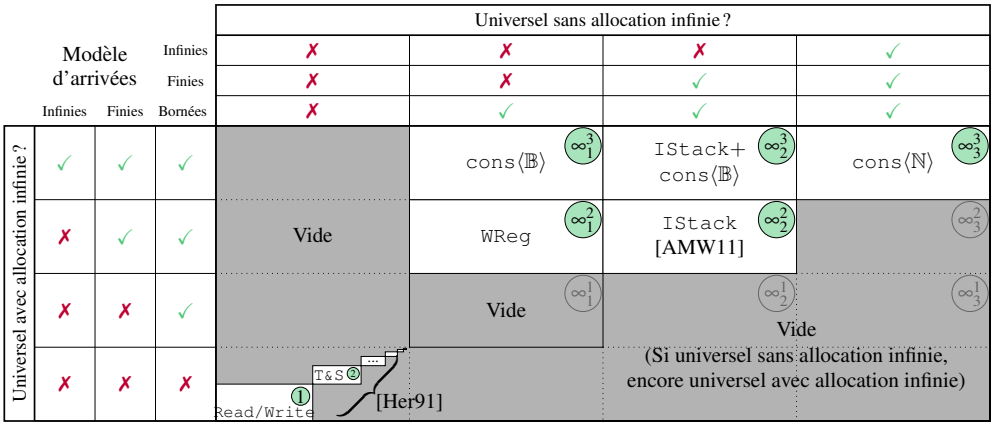

FIGURE 1: Hiérarchie de Herlihy étendue : dans un système multi-thread, il est impossible d'implémenter un objet O_1 à partir d'instances d'un objet O_2 et de registres lire/écrire, si O_2 est plus à gauche ou plus bas que O_1. Les cercles verts montrent le numéro de consensus de chaque degré.

triés horizontalement en fonction de leur universalité dans les modèles M_1^n, M_1, M_2 et M_3 lorsque l'allocation de mémoire infinie n'est pas disponible, et verticalement en fonction de leur capacité à le faire dans les modèles MA_1^n, MA_1, MA_2 et MA_3, où il est possible d'allouer des tableaux infinis. Dans la suite de cet article, nous explorons s'il existe ou non un objet remplissant chaque degré de la hiérarchie.

2 Contributions

Universalité du consensus dans M_3. Pour prouver l'universalité du consensus dans le modèle à arrivées bornées, Herlihy a introduit la notion de construction universelle. Il s'agit d'un algorithme générique qui, étant donnée la spécification séquentielle d'un objet, en produit une implémentation concurrente. Pour garantir la terminaison, les implémentations wait-free nécessitent un mécanisme d'aide : chaque processus doit aider les autres processus à terminer leur opération courante avant de terminer la sienne. Pour cela, chaque processus doit pouvoir annoncer ses opérations aux processus qui veulent l'aider. Cependant, étant donné l'infinité du nombre de processus susceptibles de participer, il n'est pas raisonnable de supposer que chacun peut écrire dans un registre dédié qui peut être lu par tous les autres. Pour résoudre ce problème, nous proposons une nouvelle structure de données : un *journal faible* qui permet aux processus d'y ajouter une valeur et de récupérer la séquence des valeurs ajoutées précédemment en utilisant une nouvelle technique que nous appelons *aide passive*. Quand un processus gagne un consensus, il crée une sous-liste pour héberger les invocations concurrentes qui ont perdu sur le même objet consensus. Comme un nombre fini de processus sont arrivés au moment où le consensus a été gagné, un nombre fini de processus vont essayer de s'insérer dans la sous-liste, ce qui garantit la terminaison. Le journal faible est wait-free mais pas linéarisable, car il ne garantit pas la propriété d'inclusion entre les séquences retournées. Cependant, cela est suffisant pour montrer l'universalité du consensus dans M_3.

L'allocation infinie n'est pas nécessaire dans M_1. L'article original sur la hiérarchie de Herlihy [Her91] ne mentionne pas de limitation due à l'impossibilité d'allouer des tableaux infinis. Nous avons démontré que, dans le contexte des modèles à arrivées bornées, l'allocation infine n'est jamais un facteur décisif pour déterminer l'universalité. Ce résultat se base sur l'observation que, dans MA_1^n, tout algorithme wait-free du consensus binaire ne peut accéder qu'à un nombre borné de configurations dans n'importe quelle exécution (et par conséquent d'objets partagés), et peut donc être transposé dans M_1^n. Le consensus binaire étant universel dans M_1^n, tout objet universel dans MA_1^n l'est aussi dans M_1^n.

Cela implique que notre hiérarchie coïncide avec celle de Herlihy pour les objets avec un numéro de consensus fini, ce qui justifie notre choix de conserver le même terme *numéro de consensus* pour notre hiérarchie. D'un autre côté, notre hiérarchie affine celle de Herlihy pour les objects avec un numéro de consensus infini. Nous disons d'un objet qu'il a pour numéro de consensus ∞_x^y, pour $x, y \in \{1,2,3\}$, s'il est universel dans M_x et dans MA_y mais pas dans M_{x+1} (si $x \neq 3$) ni dans MA_{y+1} (si $y \neq 3$). Comme la possibilité d'accéder à des tableaux infinis n'est jamais préjudiciable, aucun objet n'a un numéro de consensus ∞_x^y

pour $y < x$. Par exemple, la pile d'itérateurs a pour numéro de consensus ∞_2^2 car elle est universelle dans les modèles d'arrivées finies, mais pas infinies.

Aucun objet n'a ∞_1^1 pour numéro de consensus. Nous avons ensuite démontré qu'aucun objet n'a ∞_1^1 pour numéro de consensus. En effet, un tel objet permettrait de résoudre le consensus entre n processus, pour tout n connu à l'avance ; mais un tel objet serait alors également universel dans MA_2, et aurait donc un numéro de consensus au moins égal à ∞_1^2 : nous avons exhibé un algorithme permettant de résoudre le consensus dans MA_2, dans lequel, à chaque ronde k, tous les processus avec un identifiant inférieur à k cherchent à se mettre d'accord sur un objet partagé initialisé à l'avance. L'algorithme s'arrête si le consensus a été atteint entre les k participants de la ronde avant que n'arrive un nouveau processus avec un identifiant supérieur à k, ce qui finit forcément par se produire dans les modèles à arrivées finies.

Les registres fenêtrés ont ∞_1^2 pour numéro de consensus. Ainsi, les objets qui ont ∞_1^2 pour numéro de consensus sont les plus faibles à pouvoir résoudre le consensus entre n objets, pour tout n. Un objet qui peut naturellement jouer ce rôle est le registre fenêtré [MPR18] (noté k-`WReg`). Un registre fenêtré de taille k a deux opérations : une écriture et une lecture, qui retourne la liste ordonnée des k dernières valeurs écrites, les valeurs manquantes étant remplacées par des valeurs par défaut. Un registre fenêtré de taille k a un numéro de consensus égal à k, donc si des registres fenêtrés de toutes tailles sont disponibles, le consensus peut être résolu dans M_1 et MA_2.

Réciproquement, il a été remarqué dans [AMW11] qu'avoir accès à des objets de numéro de consensus n pour tout n n'était pas suffisant pour résoudre le consensus dans MA_3. Les arguments qui y sont développés, qui s'appuient sur la notion classique de valence, peuvent être adaptés aux registres fenêtrés.

La preuve la plus compliquée est que les registres fenêtrés ne sont pas universels dans M_2. L'idée est semblable à un résultat de complexité récent [EGSZ16] : au moins $\lceil \frac{n}{k} \rceil$ registres fenêtrés de taille k sont nécessaires pour résoudre le consensus obstruction-free entre n processus. En précisant les arguments, on se rend compte que ces registres fenêtrés doivent être créés à l'initialisation du système, car les processus sont ensuite incapables de se mettre d'accord sur quelles références vers de nouveaux objets sont valides. Ainsi, n'importe quel algorithme de consensus obstruction-free se basant sur des registres fenêtrés possède une borne sur le nombre de processus qu'il peut synchroniser, ce qui le rend incompatible avec M_2.

Le consensus booléen a ∞_1^3 pour numéro de consensus. Le consensus binaire (noté $\texttt{cons}\langle\mathbb{B}\rangle$) où seules les valeurs **true** et **false** peuvent être proposées, a depuis longtemps été étudié comme un équivalent au consensus multivalué dans M_1. Nous avons étendu ces résultats dans MA_3. Réciproquement, la preuve d'impossibilité concernant les registres fenêtrés dans M_2 peut être adaptée au consensus binaire : il faut au moins k instances du consensus binaire pour synchroniser 2^k processus.

Objets avec ∞_2^3 pour numéro de consensus. La composition de deux objets de numéro de consensus respectif ∞_1^3 et ∞_2^2 ne peut avoir que ∞_2^3 ou ∞_3^3 pour numéro de consensus. Nous avons démontré que la composition du consensus binaire et la pile d'itérateurs n'est pas universelle dans M_3, et donc que son numéro de consensus est ∞_2^3. Parce qu'elle combine des techniques de preuve très différentes, cette démonstration est notre contribution la plus difficile techniquement.

Références

[Agu04] M. K. Aguilera. A pleasant stroll through the land of infinitely many creatures. *ACM Sigact News*, 2004.

[AMW11] Y. Afek, A. Morrison, and G. Wertheim. From bounded to unbounded concurrency objects and back. In *Proc. ACM Symposium on Principles of Distributed Computing*, 2011.

[EGSZ16] F. Ellen, R. Gelashvili, N. Shavit, and L. Zhu. A complexity-based hierarchy for multiprocessor synchronization. In *Proc. ACM Symposium on Principles of Distributed Computing*, 2016.

[FLP85] M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 1985.

[Her91] M. Herlihy. Wait-free synchronization. *ACM TOPLAS*, 1991.

[HW90] M. Herlihy and J. M. Wing. Linearizability : A correctness condition for concurrent objects. *ACM TOPLAS*, 1990.

[MPR18] A. Mostefaoui, M. Perrin, and M. Raynal. A simple object that spans the whole consensus hierarchy. *Parallel Processing Letters*, 2018.

[PMB20] Matthieu Perrin, Achour Mostefaoui, and Grégoire Bonin. Extending the wait-free hierarchy to multi-threaded systems. In *Proc. ACM Symposium on Principles of Distributed Computing*, 2020.