Quelles Relations entre Lire/Ecrire et Envoyer/Recevoir dans les Systèmes Asynchrones avec Crashs de Processus ?

Mathilde Déprés, Achour Mostefaoui, Matthieu Perrin, Michel Raynal

Quelles Relations entre Lire/Ecrire et Envoyer/Recevoir dans les Systèmes Asynchrones avec Crashs de Processus ?

Mathilde Déprés,[1] Achour Mostéfaoui,[2] Matthieu Perrin[2] et Michel Raynal[3†]

[1]*ENS Paris-Saclay, Université Paris-Saclay*
[2]*LS2N, Nantes Université*
[3]*Univ Rennes IRISA, Inria, CNRS*

Cet article porte sur la puissance en calculabilité des motifs de messages dans les systèmes asynchrones à passage de messages sensibles aux pannes. Il propose et étudie trois motifs de messages de base rencontrés dans ces systèmes, impliquant chacun deux processus, et les compare à leurs équivalents en mémoire partagée. Il est d'abord remarqué qu'un de ces motifs, appelé MP1, n'a pas d'équivalent en mémoire partagée. L'article propose ensuite une nouvelle abstraction de communication appelée *Mutual Broadcast*, qui interdit au motif MP1 de se produire. La puissance de calcul de cette abstraction de diffusion est la même que celle d'un registre lire/écrire atomique.

Mots-clefs : Abstraction de diffusion de messages, Asynchronie, Atomicité, Calcul réparti, Caractérisation, Motifs de lecture/écriture, Motifs de messages, Pannes franches, Registre lire/écrire.

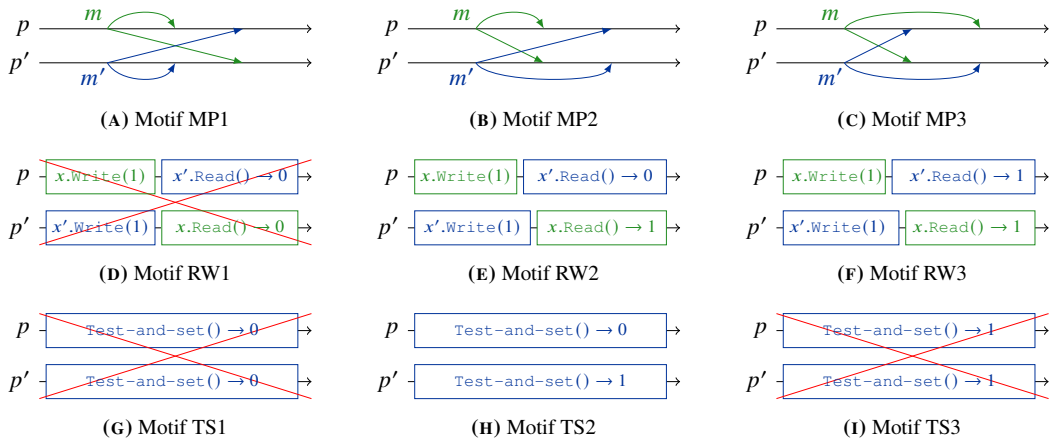
1 Introduction : une vue d'ensemble

De send/receive aux abstractions de coopération. Les opérations send() et receive() constituent le langage machine des réseaux sous-jacents. Ainsi, pour résoudre un problème de calcul distribué, il est habituel de définir d'abord une abstraction de communication appropriée qui facilite la conception d'algorithmes de niveau supérieur. L'ordonnancement des messages FIFO et causal [BJ87, Ray18] sont des exemples de telles abstractions de communication qui facilitent la construction d'objets distribués tels que, par exemple, une mémoire causale sur un système de passage de messages asynchrone [ANB+95]. Une abstraction de communication de haut niveau et très puissante est le broadcast d'ordre total, qui assure que l'ordre de livraison des messages est le même pour tous les processus.

Du côté lecture/écriture. Les registres de lecture/écriture (RW) (c'est-à-dire, les cellules d'une machine de Turing) sont au centre des algorithmes distribués lorsque les processus communiquent par une mémoire partagée. Ainsi, un problème fondamental est la construction de registres RW atomiques sur un système de passage de messages asynchrone sujet aux pannes. Ce problème a été résolu par Attiya, Bar-Noy et Dolev qui ont présenté dans [ABND95] un algorithme basé sur send/receive (ABD) pour une telle construction, et ont prouvé que, d'un point de vue opérationnel, de telles constructions sont possibles si et seulement si au plus $t < n/2$ processus peuvent tomber en panne. La construction ABD est basée sur l'utilisation explicite de numéros de séquence, de quorums et d'un schéma send/receive (utilisé une fois pour écrire et deux fois pour lire). Les quorums sont utilisés pour réaliser des barrières de synchronisation.

Objectif. Le but de cet article est de comparer deux modèles de calcul réparti classiques, tous deux composés d'une collection de processus asynchrones sujets aux pannes : le modèle à passage de messages, et le modèle en mémoire partagée. Nous nous proposons de comparer ces modèles sous l'angle des motifs de communication qu'ils permettent entre les processus.

†Ce travail a été partiellement soutenu par le projet régional français "Étoile Montante en Pays De La Loire" consacré aux aspects de calculabilité des abstractions de diffusion, et le projet ANR français ByBLoS (ANR-20-CE25-0002-01) dédié à la conception modulaire de blocs de construction pour des applications multi-utilisateurs à grande échelle sans confiance.



(A) Motif MP1 **(B)** Motif MP2 **(C)** Motif MP3

(D) Motif RW1 **(E)** Motif RW2 **(F)** Motif RW3

(G) Motif TS1 **(H)** Motif TS2 **(I)** Motif TS3

FIGURE 1 : Les trois motifs de communication en passage de messages, en mémoire partagée, et avec test-and-set

2 Trois motifs de communication binaires de base

Motifs de communication par messages. Considérons deux processus p et p' qui échangent des messages de façon concurrente, à savoir, p envoie un message m à lui-même et à p', tandis que p' envoie le message $m' \neq m$ à lui-même et à p. Selon l'ordre de livraison des messages par chaque processus, il y a exactement trois cas à considérer (échanger p et p' ne donne pas lieu à de nouveaux motifs de messages).

Motif de messages MP1. Ce cas est représenté sur la Figure 1a. Il s'agit d'un motif symétrique dans lequel chaque processus délivre son propre message en premier, puis le message de l'autre processus. Ce motif capture le cas où, lorsqu'un processus livre son propre message, il n'a aucune information sur le fait que les autres processus diffusent ou non un message.

Motif de messages MP2. Ce cas est représenté sur la Figure 1b. Il décrit un motif asymétrique du point de vue de la livraison des messages dans lequel à la fois p et p' livrent d'abord m diffusé par p puis m' diffusé par p'. Dans ce motif, les deux processus p et p' livrent les messages dans le même ordre (un motif analogue se produit lorsque nous échangeons le rôle de p et de p').

Motif de messages MP3. Ce cas est représenté sur la Figure 1c. De manière similaire à MP1 il s'agit d'un motif symétrique dans le sens où p délivre d'abord le message m' de p' puis son message m, tandis que p' délivre d'abord le message m de p puis son message m'. La différence fondamentale entre MP1 et MP3 réside dans le fait que lorsque p (respectivement p') délivre son propre message, il a déjà livré le message envoyé par l'autre processus p' (respectivement p).

Des messages à la mémoire partagée. Pour comprendre la signification profonde et la portée des trois précédents motifs de communication basés sur les messages, considérons leurs « équivalents » dans un contexte où p et p' coopèrent par l'intermédiaire de registres lire/écrire atomiques à un bit initialisés à 0. Le registre x, écrit par p et lu par p', correspond à m. Le registre x', écrit par p' et lu par p correspond à m'. Dans chaque cas, le motif de lecture/écriture représenté au milieu de la Figure 1 simule le motif de messages au-dessus de lui. Plus précisément, nous avons ce qui suit.

Motif de lecture/écriture RW1. La Figure 1d correspond au motif de message MP1 : p écrit 1 dans x et lit la valeur initiale de x', à savoir, la valeur 0, tandis que p' écrit 1 dans x' et lit la valeur initiale de x, à savoir, la valeur 0.

Motif de lecture/écriture RW2. La Figure 1e correspond au motif MP2 : p écrit 1 dans x puis lit la valeur initiale 0 de x', tandis que p' écrit 1 dans x' puis lit la valeur 1 fraîchement écrite dans x.

Motif de lecture/écriture RW3. La Figure 1f correspond au motif MP3 : chaque processus écrit d'abord « sa » variable (x pour p, x' pour p'), puis lit la nouvelle valeur 1 dans l'autre variable.

Il est facile de voir que les motifs de lecture/écriture RW2 et RW3 produisent la même coopération que les motifs de messages MP2 et MP3, respectivement. Au contraire, alors que le motif MP1 peut se produire dans un système asynchrone par passage de messages, le motif RW1 ne peut pas se produire en mémoire partagée. Cela est dû au fait que, dans RW1, l'écriture de x par p et l'écriture de x' par p' sont linéarisées [HW90, Lam86] et, comme un processus écrit un registre RW avant de lire l'autre registre, il est impossible que les deux opérations de lecture retournent 0.

Du côté du test-and-set. Pour comparaison, les Figures 1g, 1h et 1i considèrent trois motifs de communication où les deux processus coopèrent à travers l'instruction spéciale Test&Set sur un registre atomique. Pour rappel, Test&Set écrit la valeur 1 dans le registre booléen (initialisé à 0) et retourne la valeur écrasée. Ainsi, le premier appel retourne 0, et tous les suivants retournent 1. De manière similaire aux motifs de messages et aux motifs de lecture/écriture, nous pouvons définir trois motifs test-and-set, en fonction de quels processus obtiennent 0 (le même résultat que lors d'une exécution solo) et quels processus obtiennent 1 (comme si leur opération était linéarisée en second).

Motif de test-and-set TS1. La Figure 1g correspond au motif RW1, dans lequel les deux processus obtiennent 0, et donc au motif de message MP1.

Motif de test-and-set TS2. La Figure 1h correspond au motif asymétrique RW2, dans lequel un processus obtient 0 et l'autre processus obtient 1. Ainsi, il correspond également au motif MP2.

Motif de test-and-set TS3. La Figure 1i correspond au motif symétrique RW3, dans lequel les deux processus obtiennent 1, et donc au motif de message MP3.

Cette fois, seul le motif de communication asymétrique TS2 est admis. Le fait que le motif TS3 soit impossible est une différence majeure entre les registres de lecture/écriture et les registres de test-and-set, qui peut être utilisée pour résoudre le consensus entre deux processus.

3 Des motifs de communication aux primitives de communication

Au cœur de l'approche. Le fait qu'il n'y a pas de motif de lecture/écriture correspondant au motif de messages MP1 est une différence fondamentale entre la coopération/communication en passage de messages et la coopération/communication en mémoire partagée, qui est implicitement utilisée pour résoudre de nombreux problèmes de coopération/synchronisation dans les systèmes en mémoire partagée. Le plus célèbre est le motif « écrire d'abord puis lire » utilisé dans tous les algorithmes d'exclusion mutuelle [RT22]. Lorsqu'un processus veut entrer en section critique, il commence par lever un drapeau pour informer les autres processus qu'il commence à concourir, et seulement ensuite il lit les drapeaux (qui sont levés ou baissés) des autres processus. L'ordre total imposé par l'atomicité sur les levées de drapeau empêche RW1 de se produire.

Mutual Broadcast. Pour comprendre l'importance de l'impossibilité du motif MP1dans les algorithmes de coordination en mémoire partagée, nous introduisons une nouvelle primitive de communication appelée *Mutual Broadcast* (en abrégé MBroadcast). En plus des propriétés habituelles des abstractions de communication (seuls des messages émis peuvent être reçus, les messages ne sont pas dupliqués, et les messages émis par les processus corrects sont reçus par tous les processus corrects), MBroadcast assure la propriété d'ordre suivante entre toute paire de processes :

Mutual Ordering. Pour toute paire de processus p et p', si p mbroadcast un message m et p' mbroadcast un message m', il est impossible que p mdelivre m avant m' et p' mdelivre m' avant m.

En d'autres termes, Mutual Broadcast interdit au motif de messages MP1 de se produire. De manière intéressante, la puissance de calcul de cette abstraction de diffusion est la même que celle d'un registre de lecture/écriture atomique. En d'autres termes, elle constitue la première caractérisation des registres RW en termes de motifs binaires de messages. En fait, empêcher le motif de message MP1 de se produire sans limiter le nombre de pannes de processus est « équivalent » à l'hypothèse $t < n/2$ sans contraintes sur les motifs d'échange de messages, dans le sens où les deux hypothèses empêchent la partition et permettent ainsi de construire des registres RW atomiques malgré les pannes et l'asynchronie. Ainsi, MBroadcast peut être vu comme une manière d'abstraire les quorums, pour les cacher à la couche supérieure.

Pair Broadcast. De la même manière, nous introduisons une deuxième primitive de communication appelée *Pair Broadcast* (en abrégé PBroadcast). La propriété d'ordre ajoutée à PBroadcast, appelée *Pair Ordering*, renforce Mutual Ordering en interdisant MP3 en plus de MP1. Autrement dit, seul le motif MP2 peut se produire entre toute paire de processus.

Pair Ordering. Pour toute paire de processus p et p', si p mbroadcast un message m et p' mbroadcast un message m', si p et p' délivrent tous les deux m et m', alors ils les délivrent dans le même ordre.

Conformément à l'intuition, PBroadcast, qui ne permet qu'au motif MP2 de se produire, est équivalent en calculabilité à l'opération Test&Set(), qui ne permet qu'au motif TS2 de se produire. Ceci est similaire au fait qu'empêcher le motif MP1 rend MBroadcast équivalent à une mémoire partagée, dans laquelle le motif RW1 est impossible.

4 Pour aller plus loin

La définition de MBroadcast a été proposée dans un article court à PODC 2023 [DMPR23a]. Ensuite, la discussion sur les motifs de messages et les motifs de lecture/écriture a été présentée à DISC 2023 [DMPR23b]. La version complète de cet article est disponible en ligne [DMPR23c]. Elle comprend plus de discussions sur l'interprétation de MBroadcast et les preuves des résultats exposés ci-dessus, ainsi que de nombreux algorithmes illustrant comment l'utilisation de MBroadcast permet d'aboutir à des solutions très simples de problèmes de coordination classiques en passage de messages.

Références

[ABND95] Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. Sharing memory robustly in message-passing systems. *Journal of the ACM (JACM)*, 42(1) :124–142, 1995.

[ANB+95] Mustaque Ahamad, Gil Neiger, James E Burns, Prince Kohli, and Phillip W Hutto. Causal memory : Definitions, implementation, and programming. *Distributed Computing*, 9(1) :37–49, 1995.

[BJ87] Kenneth P Birman and Thomas A Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems (TOCS)*, 5(1) :47–76, 1987.

[DMPR23a] Mathilde Déprés, Achour Mostéfaoui, Matthieu Perrin, and Michel Raynal. Brief announcement : The mbroadcast abstraction. In *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing*, pages 282–285, 2023.

[DMPR23b] Mathilde Déprés, Achour Mostéfaoui, Matthieu Perrin, and Michel Raynal. Send/receive patterns versus read/write patterns in crash-prone asynchronous distributed systems. In *37th International Symposium on Distributed Computing (DISC 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.

[DMPR23c] Mathilde Déprés, Achour Mostefaoui, Matthieu Perrin, and Michel Raynal. Send/Receive Patterns versus Read/Write Patterns : the MB-Broadcast Abstraction (Extended Version). working paper or preprint, May 2023.

[HW90] Maurice P Herlihy and Jeannette M Wing. Linearizability : A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(3) :463–492, 1990.

[Lam86] Leslie Lamport. On interprocess communication : part i : basic formalism. *Distributed computing*, 1 :77–85, 1986.

[Ray18] Michel Raynal. *Fault-tolerant message-passing distributed systems : an algorithmic approach*. Springer, 2018.

[RT22] Michel Raynal and Gadi Taubenfeld. A visit to mutual exclusion in seven dates. *Theoretical Computer Science*, 919 :47–65, 2022.