Une Implémentation Optimale pour SCD-Broadcast Byzantin

Vincent Kowalski, Achour Mostefaoui, Matthieu Perrin

Une Implémentation Optimale pour SCD-Broadcast Byzantin

Vincent Kowalski[1] et Achour Mostéfaoui[1] et Perrin Matthieu[1]

[1]*LS2N, Nantes Université*

Dans cet article, nous étudions une abstraction de communication appelée Set-Constrained Delivery Broadcast (SCD-Broadcast) ainsi que sa résilience optimale dans un système de passage de messages sujet aux fautes byzantines. SCD-Broadcast est une abstraction de communication qui offre une propriété d'ordonnancement parmi des ensembles de messages. Cette abstraction permet à chaque processus de diffuser des messages et de délivrer des ensembles de messages reçus de telle manière que si un processus délivre un ensemble contenant un message m avant un ensemble contenant un message m', alors aucun autre processus ne peut délivrer un ensemble contenant m' avant un ensemble contenant m. La première implémentation de SCD-Broadcast a été conçue pour des systèmes distribués sujets aux pannes franches. Par la suite, une nouvelle implémentation de SCD-Broadcast appelée Byzantine-Tolerant Set-Constrained Delivery Broadcast (BSCD-Broadcast) a été adaptée à un contexte sujet aux défaillances byzantines. La résilience de cette implémentation est $t < n/4$ (où t est le nombre maximal de processus pouvant être byzantins et n est le nombre total de processus). Cet article présente un nouvel algorithme qui met en œuvre l'abstraction Byzantine SCD-Broadcast avec une résilience optimale de $t < n/3$.

Mots-clefs : Broadcast primitive, Byzantine process, Message-passing distributed system, Message ordering, Reliable broadcast, Shared memory.

1 Introduction

Les primitives de diffusion (broadcast) sont les moyens de communication de base entre les processus dans les systèmes de passage de messages. Dans la primitive de diffusion la plus simple à laquelle on puisse penser, un processus p qui diffuse un message m envoie simplement m à tous les processus. Afin de tolérer les pannes, la primitive de *diffusion fiable (Reliable Broadcast)* peut être construite sur celle-ci [HT93], garantissant des propriétés plus robustes aux processus qui l'utilisent. La diffusion fiable assure que tous les processus qui ne tombent jamais en panne finiront par recevoir (*deliver*) le même ensemble de messages. En d'autres termes, même si le processus émetteur tombe en panne pendant sa diffusion de m, soit tous les autres processus corrects reçoivent m, soit aucun d'entre eux.

La diffusion fiable peut être enrichie avec des propriétés supplémentaires qui restreignent l'ordre dans lequel différents processus reçoivent des messages concurrents, permettant aux concepteurs de logiciels de séparer les aspects fonctionnels de leur application des problèmes de synchronisation qui peuvent être directement encapsulés dans l'abstraction de diffusion. Ces primitives de diffusion définissent une hiérarchie d'abstractions qui facilitent la conception d'applications de couche supérieure.

Le *Causal Broadcast* [BJ87] est un exemple d'une abstraction de diffusion enrichie d'une propriété d'ordonnancement. Il s'agit d'une primitive de communication qui permet aux processus de diffuser des messages de manière à ce que la réception des messages respectent l'ordonnancement causal entre leurs émissions. Si un message est diffusé causalement avant un autre message (au sens de la relation précédent-introduit par Lamport [Lam78]), alors aucun processus ne peut recevoir le dernier message avant le premier.

Set-Constrained-Delivery Broadcast Récemment, une nouvelle primitive de diffusion appelée Set-Constrained-Delivery Broadcast (SCD-Broadcast) a été proposée [IMPR21]. Elle est plus faible que la diffusion atomique car elle ne permet pas la mise en œuvre du consensus, mais elle est plus forte que la diffusion fiable et causale car elle permet l'émulation d'une mémoire partagée. En effet, elle est équivalente à une abstraction de mémoire, et inversement, cette primitive de diffusion peut être émulée sur un système

de mémoire partagée. On sait qu'une abstraction de mémoire peut être émulée sur un système de passage de messages asynchrone sujet aux pannes de processus uniquement si la majorité des processus ne tombent jamais en panne [ABD95]. De manière similaire, SCD-Broadcast peut être mise en œuvre dans le même contexte uniquement si la majorité des processus ne tombent jamais en panne.

Dans le contexte des systèmes distribués sujets aux défaillances byzantines, au plus t processus parmi les n processus du système ont le potentiel non seulement de tomber en panne, mais aussi de manifester un comportement arbitraire [PSL80]. La même hiérarchie de primitives de diffusion a également été étudiée dans des systèmes sujets aux défaillances byzantines. La diffusion fiable byzantine a été proposée dans [Bra84]. Contrairement au contexte des pannes où elle peut être mise en œuvre quel que soit le nombre de processus qui tombent en panne, dans le contexte byzantin, le nombre maximal t de processus byzantins dans le système ne peut pas dépasser $t < n/3$.

Objectif Récemment, la diffusion Set-Constrained-Delivery (SCD-Broadcast) a été étendue au contexte byzantin dans [ART19]. Alors que toute implémentation de SCD-Broadcast requiert $t < n/2$ dans le modèle avec pannes, la seule adaptation connue au contexte byzantin nécessite $t < n/4$. D'un autre côté, les implémentations connues de registres partagés sur un système de passage de messages sujet aux défaillances byzantines n'ont besoin que de $t < n/3$ [IRRS16, MPRJ17]. Comme SCD-Broadcast est censée être équivalente à une abstraction de mémoire (l'une peut être émulée par l'autre), la question demeure ouverte quant à savoir si $t < n/4$ est une borne inférieure pour implémenter la primitive SCD-Broadcast dans le contexte byzantin.

Cet article introduit à un travail répondant à cette question de manière négative : $t < n/4$ n'est pas nécessaire pour implémenter la primitive Set-Constrained-Delivery Broadcast dans le contexte byzantin. Pour ce faire, il propose une implémentation supportant $t < n/3$, satisfaisant à la fois la borne inférieure pour une émulation de registre et la borne inférieure pour la diffusion fiable byzantine, dans un système de passage de messages asynchrone sujet aux défaillances byzantines. En d'autres termes, la borne $t < n/3$ est contraignante pour SCD-Broadcast byzantine.

2 Modèle de communication

Dans cet article, nous considérons le modèle classique de calcul asynchrone avec propension aux fautes byzantines et communication par messages.

Entités informatiques Le système est composé d'un ensemble de n processus séquentiels, notés p_1, p_2, ..., p_n. Ces processus sont asynchrones, ce qui signifie que chaque processus progresse à sa propre vitesse, pouvant être arbitraire, pouvant varier au cours de n'importe quelle exécution, et restant toujours inconnue des autres processus. Chaque processus p_i a accès à son propre identifiant i qui peut être utilisé dans le code.

Modèle de défaillance Parmi les n processus du système, on suppose qu'au plus t processus peuvent présenter un comportement byzantin. Un comportement byzantin est caractéristique d'un processus ne suivant pas son algorithme et agissant de manière arbitraire [PSL80] : il peut commencer dans un état arbitraire, cesser de s'exécuter à tout moment (ce comportement est appelé un crash), effectuer des transitions d'état arbitraires, tenter de communiquer des valeurs arbitraires ou différentes à différents processus, etc. Un processus byzantin est également appelé un processus *défaillant*, et un processus qui ne commet aucune défaillance (c'est-à-dire un processus non byzantin) est appelé un processus *correct*. Dans cet article, nous considérons qu'au moins les deux tiers des processus sont corrects, c'est-à-dire $t < n/3$.

Modèle de communication Les différents processus communiquent en échangeant des messages via des canaux de communication bidirectionnels. Ces canaux connectent chaque paire de processus de telle sorte que n'importe quel processus peut identifier l'émetteur d'un message et aucun processus ne peut se faire passer pour un autre processus. L'envoi d'un message est asynchrone et fiable. "Asynchrone" signifie qu'il n'y a pas de limite de délai de transfert des messages, et "fiable" signifie que les canaux ne créent, ne dupliquent pas et ne modifient pas l'information, et que tous les messages envoyés aux processus corrects finiront par être reçus.

3 Propriétés du SCD-Broadcast Byzantin

Une primitive de communication Byzantine SCD-Broadcast a été définie dans [ART19]. Il s'agit d'une extension au contexte byzantin de la primitive de communication SCD-Broadcast initiale proposée pour le modèle de défaillance par crash dans [MPRJ17]. Cette abstraction de communication offre deux opérations, `scd_broadcast()` et `scd_deliver()`. L'opération `scd_broadcast`(m) permet à un processus de diffuser un message, tandis que `scd_deliver()` fournit à un processus appelant un ensemble non vide de paires $\langle m, j\rangle$ composées d'un message m et de l'identifiant du processus p_j qui a diffusé m en utilisant `scd_broadcast()`. D'où le nom de **Set**-Constrained-Delivery Broadcast. "Contraint" fait référence au fait qu'il existe des contraintes d'ordre sur les ensembles délivrés aux différents processus. Nous supposons que les processus corrects invoquent régulièrement `scd_deliver()` pour vérifier s'ils ont reçu de nouveaux messages.

Nous faisons une distinction entre un message et son contenu : chaque message est unique, au sens où il n'est diffusé qu'une fois et livré au plus une fois par chaque processus correct, même si deux messages différents peuvent avoir le même contenu. Pour faciliter l'exposition, nous supposons que `scd_broadcast` prend des messages en tant qu'arguments, même si dans les implémentations pratiques, `scd_broadcast` créerait probablement un nouveau message à partir de son contenu donné en argument.

Nous disons qu'un processus p_i diffuse (scd-broadcast) un message m lorsque p_i invoque `scd_broadcast`(m). De même, nous disons que p_i délivre (scd-deliver) un ensemble de messages $mset$ lorsque `scd_deliver()` renvoie $mset$. Par un léger abus de langage, nous disons que p_i délivre (scd-deliver) un message m d'un processus p_j si p_i délivre (scd-deliver) un ensemble de messages $mset$ contenant la paire $\langle m, j\rangle$.

Plus précisément, la diffusion SCD est définie par les cinq propriétés suivantes :

Validité. Si un processus correct p_i délivre (scd-deliver) un message m provenant d'un processus correct p_j, alors p_j a diffusé (scd-broadcast) m.

Intégrité. Un message m peut être délivré (scd-deliver) au plus une fois par chaque processus correct.

Terminaison. Soit m un message diffusé (scd-broadcast) par un processus correct p_i. Le message m sera éventuellement délivré (scd-deliver) par p_i.

Cohérence. Si un processus correct délivre (scd-deliver) un message m provenant d'un processus quelconque p_j, alors tous les processus corrects délivreront éventuellement (scd-deliver) m provenant de p_j (p_j peut être correct ou byzantin).

Ordre. Soit p_i un processus correct qui délivre (scd-deliver) en premier un ensemble contenant un message m et délivre ensuite un autre ensemble contenant un message m'. Alors, aucun processus correct ne peut délivrer (scd-deliver) un ensemble contenant m' et ensuite un autre ensemble contenant m.

4 Algorithme

Afin de comprendre le fonctionnement de l'algorithme de manière intuitive, ce dernier peut être séparé en deux parties avec des rôles distincts.

Reliable Broadcast La première partie reprend le principe exact du reliable broadcast de Bracha [Bra84]. Elle permet d'assurer qu'à la fin de cette première étape, si un message a été reçu par un processus correct, tous les processus corrects finiront par recevoir ce même message. Ce reliable broadcast est utilisé pour envoyer le message à communiquer ainsi qu'un numéro de séquence, attribué par le processus émetteur. Nous nous assurons également que chaque processus ne puisse scd-broadcast qu'un seul message à la fois. Si un processus tente de broadcaster deux messages en même temps, le deuxième message sera mis en attente tant que le premier n'aura pas été scd-deliver.

Propriété d'ordre Une fois l'étape du Reliable Broadcast terminée, la seconde partie s'occupe de la propriété d'ordre propre au SCD-Broadcast.

Lorsqu'un processus p_i reçoit un message ayant passé le Reliable Broadcast, il lui attribue un numéro de séquence et l'envoie à tous les autres processus. Ainsi, les autres processus peuvent savoir dans quel ordre p_i place les différents messages qu'il a reçus après le Reliable Broadcast.

Chaque processus tient note de tous les messages qu'il a reçus de la part des autres processus ainsi que des numéros de séquence qui leur sont associés. Si suffisamment de numéros de séquence ont été reçus pour un même message m (au moins $n - t$ réceptions) et qu'aucun message m' pouvant être délivré n'a été identifié, tel que la majorité des processus corrects ($\frac{n+t}{2}$ processus) a placé m' comme plus récent que m, alors m est délivré dans le même ensemble que chaque message satisfaisant ces mêmes conditions.

5 Pour aller plus loin

En fournissant une implémentation de SCD-Broadcast avec une résilience de $t < \frac{n}{3}$, SCD-Broadcast atteint une résilience comparable à celle de différentes abstractions de mémoire. Cependant, bien que certaines abstractions de mémoire soient équivalentes en termes de calculabilité à SCD-Broadcast dans un système soumis à des pannes, d'autres, comme l'objet Read/Snapshot, perdent en puissance dans un contexte incluant des acteurs byzantins et deviennent moins puissantes que SCD-Broadcast. Un article détaillant et prouvant l'algorithme discuté ici a été présenté lors d'EDCC 2024.

Références

[ABD95] Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. Sharing memory robustly in message-passing systems. *J. ACM*, 42(1) :124–142, 1995.

[ART19] Alex Auvolat, Michel Raynal, and François Taïani. Byzantine-tolerant set-constrained delivery broadcast. In *23rd Int. Conf. on Principles of Distributed Systems, OPODIS 2019, Neuchâtel, Switzerland*, volume 153 of *LIPIcs*, pages 6 :1–6 :23, 2019.

[BJ87] Kenneth P. Birman and Thomas A. Joseph. Reliable communication in the presence of failures. *ACM Trans. Comput. Syst.*, 5(1) :47–76, 1987.

[Bra84] Gabriel Bracha. An asynchronou [(n-1)/3]-resilient consensus protocol. In Tiko Kameda, Jayadev Misra, Joseph G. Peters, and Nicola Santoro, editors, *Proc. of the 3rd ACM Symposium on Principles of Distributed Computing, Vancouver, Canada*, pages 154–162. ACM, 1984.

[HT93] Vassos Hadzilacos and Sam Toueg. Distributed systems. *Fault-Tolerant Broadcasts and Related Problems*, pages 97–145, 1993.

[IMPR21] Damien Imbs, Achour Mostéfaoui, Matthieu Perrin, and Michel Raynal. Set-constrained delivery broadcast : A communication abstraction for read/write implementable distributed objects. *Theor. Comput. Sci.*, 886 :49–68, 2021.

[IRRS16] Damien Imbs, Sergio Rajsbaum, Michel Raynal, and Julien Stainer. Read/write shared memory abstraction on top of asynchronous byzantine message-passing systems. *J. Parallel Distributed Comput.*, 93-94 :1–9, 2016.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7) :558–565, 1978.

[MPRJ17] Achour Mostéfaoui, Matoula Petrolia, Michel Raynal, and Claude Jard. Atomic read/write memory in signature-free byzantine asynchronous message-passing systems. *Theory Comput. Syst.*, 60(4) :677–694, 2017.

[PSL80] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2) :228–234, 1980.