Émulation de mémoire partagée en environnements distribués dynamiques

Vincent Gramoli

EPFL
Station 14, CH-1015 Lausanne, Suisse
Université de Neuchâtel
Rue Emile-Argand 11, CH-2007 Neuchâtel, Suisse
vincent.gramoli@epfl.ch

RÉSUMÉ. *La plupart des systèmes distribués modernes sont à la fois à grande échelle et dynamiques. Dans un tel contexte il est difficile pour les entités participantes de communiquer. La mémoire partagée est une abstraction de communication fiable de ces entités au travers d'opérations de lecture et d'écriture. Cet article présente les défis qui entourent l'émulation de mémoire partagée dans un environnement à passage de message où les nœuds participants ne cessent de quitter le système et où de nouveaux nœuds rejoignent ce système de façon imprévisible. À travers les travaux existants sur les systèmes de quorum, les mécanismes de reconfiguration, et les objets en lecture/écriture probabiliste cet article dresse un état de l'art des approches adoptées pour pallier le dynamisme en passant par la tolérance aux pannes franches et au problème de minimisation des communications.*

ABSTRACT. *Most distributed systems are large-scale and dynamic. Communication has thus become a critical issue and the shared memory, which is a reliable paradigm of communication, allows distributed nodes to exchange information through read and write operations. In this article, I introduce the challenges related to emulating a shared memory in a dynamic message passing environment where nodes constantly leave the system and where new nodes arrive in an unpredictable manner. This article surveys the work on quorum systems and reconfigurable systems, and present solutions that cope with crash failures, dynamism and limited complexity.*

MOTS-CLÉS : *Systèmes pair-à-pair, Cohérence atomique, Mémoire partagée distribuée, Quorum.*

KEYWORDS: *Peer-to-Peer Systems, Atomic Consistency, Distributed Shared Memory, Quorum.*

RSTI - TSI – 30/2011. Algorithmique distribuée, pages 809 à 839

1. Introduction

Les systèmes distribués actuels deviennent de plus en plus hétérogènes. Le dynamisme est ainsi devenu une problématique majeure tant le comportement de leurs entités est imprévisible : mobilité, panne franche, interférence, déconnexion, etc. Cet article étudie les moyens permettant à des nœuds de communiquer dans un tel environnement dynamique. Pour ce faire, nous étudions les solutions proposées pour émuler une communication simple dite à mémoire partagée dans un environnement dit à passage de message où la communication est difficile. Cette émulation permet aux nœuds du réseau de communiquer de façon simple et transparente.

Bien que la réplication des nœuds permette de pallier les pannes franches, elle ne permet pas de pallier l'accumulation de pannes que peut provoquer le dynamisme. La problématique majeure est donc d'adapter le système face aux changements pour permettre à celui-ci de bénéficier des nœuds nouvellement actifs. L'idée commune aux solutions proposées est celle des *quorums*, des ensembles de nœuds qui possèdent au moins un nœud en commun, *i.e.*, leur intersection est non vide. Ces quorums jouent un rôle primordial lorsqu'un client décide d'écrire une information que d'autres clients pourront lire ensuite : ils permettent non seulement à cette information de persister au cours du temps, en dépit de la panne de certains membres d'un quorum, mais ils permettent aussi d'assurer la cohérence de la donnée renvoyée par l'ensemble des membres. Plus précisément, même s'il est possible que certains membres du quorum renvoient une valeur ancienne, la propriété d'intersection assure qu'il existe au moins un membre qui retournera la valeur à jour.

La réplication au sein de quorum est connue depuis 30 ans comme permettant la persistance des données en dépit des pannes franches (Gifford, 1979; Thomas, 1979). Puisque toute nouvelle information est répliquée aux nœuds membres du quorum, la persistance de la donnée dépend de la taille de ce dernier. Cependant la cohérence des lectures et écritures dépend de l'intersection entre quorums. Ainsi l'intersection doit être conservée pour assurer la cohérence en dépit des pannes. Le problème devient plus difficile lorsque les pannes s'accumulent et que leur nombre excède le nombre de nœuds initialement présents dans le système. Pour que les données persistent ou restent cohérentes, il devient alors nécessaire de transférer les données stockées sur d'anciens nœuds sur de nouveaux nœuds et ce continuellement. De plus, ce transfert doit être effectué suffisamment rapidement pour pallier au rapide départ des nœuds. Cette rapidité indique l'intensité du dynamisme, aussi appelé *va-et-vient*. Nous supposerons dans notre modèle qu'il existe à tout moment des nœuds présents dans notre système, cependant nous supposerons également que tous les nœuds peuvent quitter et que d'autres peuvent rejoindre le système rapidement : avec un va-et-vient élevé.

Dans cet article, nous présentons une implémentation simple d'une émulation de mémoire partagée en système à passage de message et montrons l'algorithme correct. Nous introduisons ensuite un mécanisme de reconfiguration pour tolérer un nombre non borné de pannes franches. Finalement nous étudions les améliorations et alternatives possibles pour supporter un fort dynamisme.

Tout au long de cette étude, nous discutons les systèmes de quorum sous-jacents en fonction de leur capacité à tolérer le dynamisme du système. La classification résultante est illustrée en figure 1.

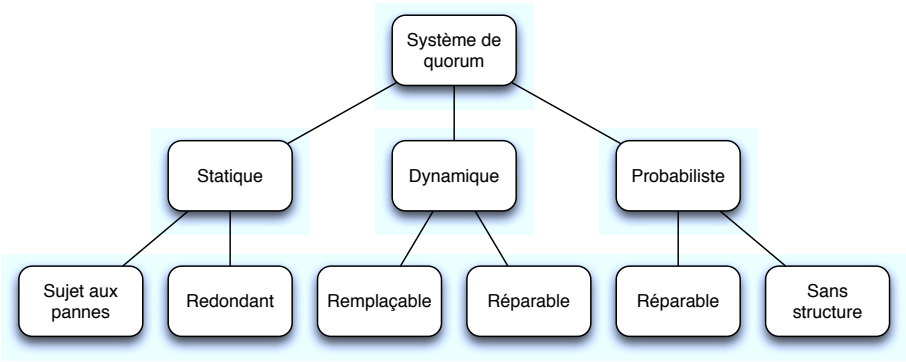

Figure 1. *Classification des systèmes de quorum utilisés pour l'émulation de mémoire partagée*

Les systèmes de quorum statiques sont sujets aux pannes ou tolèrent seulement un nombre borné de pannes. Les premiers sont illustrés dans la section 4.1 pour comprendre la problématique, les seconds sont illustrés dans les sections 4.2 et 4.3. Les systèmes de quorum dynamiques sont soit remplaçables lorsqu'ils se reconfigurent de façon globale, soit réparables lorsque leur reconfiguration est localisée. Les premiers sont illustrés dans la section 5 et les seconds dans la section 6.1. Finalement, les systèmes de quorum probabilistes peuvent également être réparables mais peuvent être « sans structure » et donc ne nécessiter aucune reconfiguration. Ces derniers types sont décrits dans les sections 6.2 et 6.3, respectivement.

L'article est organisé comme suit. La section 2 présente le modèle utilisé et les définitions liées aux quorums. La section 3 présente un algorithme générique utilisant les quorums qui émulent une mémoire partagée dans un modèle à passage de message. La section 4 décrit comment les systèmes de quorum sous-jacents peuvent permettre de tolérer les pannes. La section 5 décrit les solutions permettant d'étendre l'algorithme d'émulation de mémoire partagée pour tolérer un nombre non borné de pannes fréquentes. Finalement, la section 6 présente des alternatives pour réduire la complexité en communication et la section 7 conclut cet article.

2. Modèle et définitions

Dans cette section, nous donnons tout d'abord les définitions préliminaires concernant les quorums avant de décrire le modèle à passage de message que nous étudions dans cet article.

2.1. *Les quorums*

Afin d'assurer la tolérance aux fautes, les objets doivent être répliqués à différents endroits ou nœuds du système. Lorsque des nœuds distants ont la possibilité de modifier un objet à n'importe quel moment, un mécanisme de synchronisation doit être mis en place pour assurer la cohérence en dépit de la concurrence. Un mécanisme permettant d'informer un nœud sur le fait que son opération peut altérer la cohérence a été défini par Gifford (1979) et Thomas (1979). Dans les mécanismes présentés dans ces articles, avant que l'opération d'un client soit effective, le client demande une permission à d'autres nœuds, ceux détenant une copie de l'objet. Cette permission est donnée en fonction de l'ensemble des nœuds qui répondent et de leur réponse : une permission donnée empêche une autre d'être accordée.

Gifford considère un système de votants où un poids global W est partagé parmi tous les votants tels que leur vote influence la décision de donner ou non la permission. Une permission est donnée pour une opération de lecture (réciproquement d'écriture) si la somme des poids des votes reçus est r (réciproquement w), tels que $r + w > W$. Thomas suppose une base de donnée répartie où la copie d'une donnée est répliquée à plusieurs endroits distincts. Plusieurs nœuds peuvent exécuter une opération sur cette base de donnée répliquée en envoyant un message à une copie de la base de donnée, cette copie essayant de récolter la permission d'une majorité de copies pour exécuter l'opération.

En dépit de l'attrait intuitif des systèmes de votes et des majorités, ils ne représentent pas une solution ultime à la cohérence mémoire des systèmes répartis. En effet, Garcia-Molina et Barbara (1985) montrent l'existence d'une généralisation de ces notions. Une notion plus générale que celle des votes valués et que celle des majorités est celle de *quorums*. Les quorums sont des ensembles s'intersectant mutuellement. Un ensemble de tels quorums est appelé un *système de quorum*.

Définition 1 (Système de quorum) *Un* système de quorum $\mathcal{Q}$ *sur un univers* U *est un ensemble sur* U *tel que pour tout* $Q_1, Q_2 \in \mathcal{Q}$*, la propriété d'intersection* $Q_1 \cap Q_2 \neq \emptyset$ *est vérifiée.*

2.2. *Modèle général*

Le système que nous considérons dans cet article consiste en un ensemble de nœuds interconnectés entre eux. Dans ce document, nous supposons qu'un nœud n'a besoin que de connaître l'identifiant d'un autre noeud afin de communiquer avec lui et chaque nœud i connaît un sous-ensemble de nœuds avec lesquels il peut communiquer, ces nœuds sont appelés les voisins du nœud i. Tous les nœuds possèdent un unique identifiant et l'ensemble des identifiants est noté I. La communication est asynchrone, ainsi la transmission des messages peuvent être arbitrairement longue. La communication n'est pas fiable et les messages peuvent être réordonnés et perdus, cependant, si un message arrive à destination, alors il a été envoyé et pas altéré ; aucun message

n'est dupliqué. Le système est dynamique en ce sens que chaque nœud qui est déjà dans le système peut partir à n'importe quel moment tandis qu'un nouveau nœud peut rejoindre le système à n'importe quel moment.

La panne d'un nœud est dite *franche* et est considérée comme un départ, et une réparation est considérée comme une nouvelle arrivée. En d'autres termes, lorsqu'un nœud ré-entre dans le système il obtient un nouvel identifiant et perd ses informations sur le système. Dans la suite nous nommons un nœud qui est dans le système comme étant *actif* et un nœud qui est parti ou tombé en panne comme étant un nœud *inactif*.

Tout objet présent dans le système est accédé *via* des opérations de lecture et d'écriture que peut initier n'importe quel nœud. Si un nœud initie une opération il est appelé *client*. Une opération de lecture a pour but de renvoyer la valeur courante de l'objet tandis qu'une opération d'écriture a pour but de modifier cette valeur. La valeur d'un objet est répliquée sur un ensemble de nœuds qu'on appelle les *serveurs*. N'importe quel nœud peut être client et serveur. Chacune des valeurs v est associée à un « tag » t. Le tag possède un compteur qui indique la version de la valeur et un identifiant qui correspond à l'identifiant du nœud qui a écrit cette valeur. Cet identifiant assure l'existence d'un ordre total sur les tags : le tag t est plus petit que t' si son compteur est plus petit que celui de t' ou si ils sont identiques mais l'identifiant de t est plus petit que celui de t'.

3. Émuler une mémoire partagée en utilisant des quorums

Cette section définit la cohérence atomique que doit satisfaire une mémoire partagée et présente un algorithme générique d'émulation de mémoire partagée en modèle à passage de messages. Pour faciliter sa présentation, cet algorithme est volontairement non tolérant aux pannes. Les sections suivantes indiqueront comment un tel algorithme peut être étendu pour tolérer les pannes et le dynamisme du système.

3.1. *Cohérence mémoire*

Un critère de cohérence mémoire est souvent vu comme un contrat entre une application et une mémoire : si une application vérifie certaines propriétés, la mémoire assure les garanties désirées. De nombreux critères de cohérence mémoire ont été proposés dans la littérature. Une échelle de tolérance est généralement utilisée pour comparer ces différents critères et sur cette échelle, l'atomicité possède le niveau de tolérance le moins élevé parmi tous les critères de cohérence existants. L'atomicité est une propriété assurée par les opérations de lecture et d'écriture telles que leur ordre vérifie la spécification série de l'objet ainsi que la précédence de temps réel.

Bien que l'atomicité soit une propriété intéressante car elle assure que les opérations ont un résultat similaire à celles appliquées à une seule mémoire partagée, des propriétés plus simples à implémenter lui sont parfois préférées. Par conséquent des critères de cohérence plus tolérants mais plus faciles à mettre en œuvre ont été

proposés : l'atomicité faible (Vidyasankar, 1996), la causalité (Lamport, 1978; Hutto *et al.*, 1990), la cohérence PRAM (Lipton *et al.*, 1988), la cohérence de processeurs (Goodman, 1989), etc. L'inconvénient de tels critères est d'autoriser plusieurs nœuds distincts à avoir une vue différente de l'objet, violant une propriété importante pour l'émulation de mémoire partagée : la *sérialisation de copie unique*. D'autres critères alliant plusieurs niveaux de tolérances sont apparus : la cohérence mixte (Agrawal *et al.*, 1994) et la cohérence hybride (Attiya *et al.*, 1998), utilisant un ordre tolérant (Dubois *et al.*, 1986).

La caractéristique de *localité* a été définie par Weihl (1989) comme étant la capacité pour un critère de cohérence à supporter la composition. Ainsi, si les exécutions restreintes à chaque objet vérifient indépendamment le critère *local* de cohérence alors toute l'exécution appliquée à la composition des objets vérifie également le critère de cohérence. L'atomicité est locale (Herlihy *et al.*, 1990), contrairement à la sérializabilité. Ici, nous considérons le critère local de cohérence atomique nous permettant ainsi d'être modulaire. La définition d'atomicité est tirée de (Lynch, 1996, théorème 13.16)[1].

Définition 2 *Soit x un objet accessible en lecture et écriture. Soit H une séquence complète d'invocations et de réponses d'opérations de lecture/écriture appliquées à x. La séquence H satisfait l'atomicité s'il existe un ordre partiel $\prec$ sur les opérations telles que les propriétés suivantes soient vérifiées :*

1) si l'événement de réponse de l'opération op_1 précède l'événement d'invocation de op_2, alors il n'est pas possible que $op_2 \prec op_1$;

2) si op_1 est une écriture et op_2 n'importe quelle autre opération alors soit $op_2 \prec op_1$ soit $op_1 \prec op_2$;

3) la valeur retournée par chaque opération de lecture est la valeur écrite par la dernière opération d'écriture qui précède cette lecture selon $\prec$ (cette valeur est la valeur initiale de l'objet si une telle écriture n'existe pas).

Nous nous intéressons essentiellement à la définition des opérations de lecture et d'écriture effectuées par des clients sur des objets dont ils ont la connaissance. Cet article n'étudie pas la recherche d'objet dans un système. Comme mentionné précédemment, l'atomicité est une propriété locale. Ainsi, dans le reste de cet article nous nous intéressons à un seul objet atomique, la généralisation à un ensemble d'objets supportant aussi l'atomicité étant directe.

1. Le premier point du théorème original se déduisant des autres points, il n'a pas été mentionné ici.

3.2. *Un algorithme simple*

Cette section explique simplement comment utiliser les quorums comme briques de base pour émuler une mémoire partagée.

Domaine	Description
$I \subset \mathbb{N}$	l'ensemble des identifiants des nœuds
V	l'ensemble des valeurs possibles de l'objet
$T \in \mathbb{N} \times I$	l'ensemble des tags possibles

Tableau 1. *Domaine de l'algorithme*

Nous présentons ici un algorithme générique implémentant un objet atomique. Par la suite, nous verrons plusieurs façons d'implémenter l'accès aux quorums et le maintien de ces quorums. Dans cet algorithme, nous ne spécifions volontairement pas les procédures de communication (send et recv) ainsi que le test d'arrêt (is-quorum-contacted). En effet, ces procédures sont spécifiées différemment en fonction de la solution proposée. Par exemple, un client peut connaître l'ensemble du quorum et attendre que tous les éléments du quorum répondent pour décider d'arrêter la procédure visant à le contacter. Ou bien, sans connaître l'ensemble du quorum, le client peut attendre le message du dernier élément contacté pour décider de cet arrêt.

Algorithme 1 État de l'algorithme générique d'objet atomique

1: États de i:
2: $Q_1, ..., Q_k \subset I$, les quorums
3: $\mathcal{Q} = \{Q_1, ..., Q_k\}$, le système de quorums
4: $\mathcal{M}$, un message contenant :
5: $type \in \{\mathsf{GET}, \mathsf{SET}, \mathsf{ACK}\}$, un type de message ;
6: $\langle val, tag \rangle \in V \times T \cup \{\perp\}$, la valeur et son tag ;
7: $seq \in \mathbb{N}$, le numéro de séquence du message.
8: $v \in V, v = v_0$, la valeur de l'objet
9: $t \in T, t = \langle 0, i \rangle$, le tag utilisé composé de :
10: $compteur \in \mathbb{N}$, le compteur des écritures ;
11: $id \in I$, l'identifiant du client écrivain.
12: $s \in \mathbb{N}, s = 0$, le numéro de séquence en cours
13: $tmax \in T$, le tag découvert le plus grand
14: $vlast \in V$, la valeur découverte la plus à jour
15: $tnew \in T$, nouveau tag à écrire
16: $vnew \in V$, nouvelle valeur à écrire

L'algorithme générique implémente un objet atomique supportant des lecteurs multiples et écrivains multiples. Le domaine des variables utilisé dans l'algorithme est présenté dans le tableau 1. Le pseudocode est volontairement abstrait et présente une solution générique utilisant des accès aux quorums. Cet algorithme s'inspire à la fois des travaux de (Attiya *et al.*, 1995) ainsi que de ceux de (Lynch *et al.*, 2002).

Un client exécute une opération en invoquant la procédure Lire ou Ecrire. Chaque serveur posséde une réplique de l'objet et maintient une valeur v et un tag t associé indiquant le numéro de version de v. Ce tag est incrémenté à chaque fois qu'une nouvelle valeur est écrite. Pour assurer qu'un seul tag correspond à une valeur unique, l'identifiant du nœud écrivain (le client qui initie l'écriture) est ajouté au tag comme chiffre de poids faible. Les procédures de lecture (Lire) et d'écriture (Ecrire) sont similaires puisqu'elles se déroulent en deux phases : la première phase Consulte la valeur et le tag associé d'un objet en interrogeant un quorum de serveurs. La seconde phase Propage la valeur $vnew$ et le tag $tnew$ à jour de l'objet au sein d'un quorum de serveurs. Lorsque la consultation termine, le client récupère la dernière valeur écrite ainsi que le tag qui lui est associé. Si l'opération est une écriture, le client incrémente le tag et Propage ce nouveau tag avec la valeur qu'il souhaite écrire. Si, par contre, l'opération est une lecture alors le client Propage simplement la valeur à jour (et son tag associé) qu'il a Consulté précédemment.

Le numéro de séquence s est utilisé comme un compteur de phase pour pallier l'asynchronie et éviter qu'un nœud ne prenne en compte un message périmé. Plus précisément, lorsqu'un nouvel appel à Consulte ou Propage est effectué, ce numéro est incrémenté lorsque la phase débute. Lorsqu'un nœud reçoit un message de numéro de séquence s lui demandant de participer, il Participe en envoyant un message contenant le même numéro de séquence s. Un message reçu ne correspondant pas à la phase courante est donc détecté grace à ce chiffre s et il est ignoré. Chaque nœud reçoit donc des messages de participation avec deux numéros différents si les réponses appartiennent à deux phases distinctes.

3.3. *Preuve de cohérence atomique*

Le théorème 1 prouve que l'algorithme générique, présenté sur les algorithmes 1 et 2, implémente un objet atomique. La preuve utilise l'ordre décrit par les tags et vérifie successivement que cet ordre respecte les trois propriétés de la définition 2. Pour cela, on définit le tag d'une opération op comme étant le tag propagé durant l'exécution de cette opération op (cf. lignes 19 et 24).

Théorème 1 *L'algorithme générique implémente un objet atomique.*

Preuve. La preuve montre successivement que l'ordre des tags sur les opérations vérifie les trois propriétés de la définition 2. Soit $\prec$ l'ordre partiel sur les opérations défini par : $op_1 \prec op_2$ si t_1 et t_2 sont les tags de deux opérations et $t_1 < t_2$, ou bien t_1 est le tag d'une opération d'écriture et t_2 celui d'une opération de lecture et $t_1 = t_2$.

1) D'une part, l'exécution séquentielle de la procédure de propagation indique que lorsqu'une opération op_1 termine alors tous les nœuds d'au moins un quorum (cf. ligne 44) ont reçu la dernière valeur à jour et le tag t_1 de l'opération op_1. D'autre part, la phase de consultation de toute opération op_2 consulte tous les nœuds d'au moins un quorum (cf. ligne 35). Sans perte de généralité, fixons $Q_1 \in \mathcal{Q}$ et $Q_2 \in \mathcal{Q}$, ces deux

Algorithme 2 Lecture/écriture de l'algorithme générique d'objet atomique

```
17: Lire()_i:
18:   ⟨v, t⟩ ← Consulte()_i
19:   Propage(⟨v, t⟩)_i
20:   Return ⟨v, t⟩

21: Ecrire(vnew)_i:
22:   ⟨v, t⟩ ← Consulte()_i
23:   tnew ← ⟨t.compteur + 1, i⟩
24:   Propage(⟨vnew, tnew⟩)_i

25: Consulte()_i:
26:   Soit Q un quorum de 𝒬
27:   s ← s + 1
28:   send(GET, ⊥, s) aux nœuds de Q
29:   Repeat:
30:     if recv(ACK, ⟨v, t⟩, s) de j then
31:       repondant ← repondant ∪ {j}
32:       if t > tmax then
33:         tmax ← t
34:         vlast ← v
35:   Until is-quorum-contacted(repondant)
36:   Return ⟨vlast, tmax⟩

37: Propage(⟨v, t⟩)_i:
38:   Soit Q un quorum de 𝒬
39:   s ← s + 1
40:   send(SET, ⟨v, t⟩, s) aux nœuds de Q
41:   Repeat:
42:     if recv(ACK, ⊥, s) de j then
43:       repondant ← repondant ∪ {j}
44:   Until is-quorum-contacted(repondant)

45: Participe()_i (activer à la réception de ℳ):
46:   if recv(ℳ) du nœud j then
47:     if ℳ.type = GET then
48:       send(ACK, ⟨v, t⟩, ℳ.seq) à j
49:     if ℳ.type = SET then
50:       if ℳ.tag > t then
51:         ⟨v, t⟩ = ℳ.⟨val, tag⟩
52:       send(ACK, ⊥, ℳ.seq) à j
```

quorums respectifs. Par la propriété d'intersection des quorums, il existe au moins un nœud j, tel que $j \in Q_1 \cap Q_2$, possédant un tag supérieur ou égal à t_1. Étant donné que le tag propagé dans op_2 est supérieur (si op_2 est une écriture, ligne 24) ou égal (si op_2 est une lecture, ligne 19) à celui propagé en j, $op_2 \not\prec op_1$.

2) Si op_2 est une lecture alors il est clair que $op_1 \prec op_2$ ou $op_2 \prec op_1$ par définition de $\prec$. Maintenant supposons que op_1 et op_2 soient deux opérations d'écriture. En raisonnant par l'absurde, supposons que les deux opérations aient le même tag. Premièrement, si les deux opérations sont initiées au même nœud i alors le compteur de leur tag diffère ce qui contredit l'hypothèse. Deuxièmement, admettons que les opérations soient initiées à des nœuds i et j tels que $i \neq j$. Par hypothèse de départ, chaque nœud possède un identifiant unique. L'identifiant est utilisé comme numéro de poids faible dans le tag, ainsi même avec un compteur égal les tags sont différents. Par conséquent, l'hypothèse est à nouveau contredite.

3) Soit op, une opération de lecture et soit $op_1, ..., op_k$, les écritures qui précèdent op, et op_h l'opération parmi ces écritures possédant le plus grand tag. Premièrement, op_h est telle que $\forall \ell \leq k$, $op_\ell \prec op_h$, par définition de $\prec$. Deuxièmement, la valeur retournée par l'opération op de lecture est associée au plus grand tag rencontré durant la phase de consultation (cf. lignes 32–34). Ainsi op renvoie la valeur écrite par la dernière écriture op_h.

□

L'algorithme générique présente les principes communs aux méthodes utilisées pour implémenter des objets atomiques. Cependant, certaines améliorations permettent à une opération de lecture de terminer après l'accès à un seul quorum. Par exemple, lorsqu'une valeur plus ancienne que la valeur concouramment écrite est retournée (Dutta *et al.*, 2004) ou bien lorsqu'il est clair que la valeur à jour a déjà été propagée à tout un quorum lorsque la lecture consulte (Chockler *et al.*, 2009; Gramoli *et al.*, 2007a).[2] De telles approches présentent des cas particuliers que ne spécifie pas l'algorithme générique pour des raisons de simplicité dans la présentation.

4. Tolérer les pannes franches *via* réplication

Cette section présente comment choisir les systèmes de quorum utilisés dans l'émulation de mémoire partagée afin qu'ils tolèrent un nombre borné de pannes.

4.1. *Panne du système de quorum*

Certains systèmes de quorum peuvent souffrir d'une seule panne et une de leur propriété cruciale est leur disponibilité (Naor *et al.*, 1998). La *disponibilité* d'un système de quorum est la probabilité qu'il a de tomber en panne, étant donné la probabilité que chacun de ses nœuds a de tomber en panne. Si l'intersection de deux quorums est vide, alors ces deux quorums ne sont plus considérés comme *actifs*. Par exemple, si tous les nœuds situés dans l'intersection des quorums Q_1 et Q_2 tombent en panne, alors les

2. Nous supposons ici le modèle plus général avec lecteurs multiples et écrivains multiples. Une opération d'écriture ne nécessitant qu'un seul accès dans le cas d'un seul écrivain (Attiya *et al.*, 1995).

quorums Q_1 et Q_2 sont considérés également comme étant en panne. Si tous les quorums intersectent exactement le même nœud i, alors la panne du nœud i provoque la panne du système de quorum. Nous présentons ici, deux systèmes de quorum qui ne tolèrent pas de panne franche.

4.1.1. *Un système de quorum en étoile*

Afin de mieux comprendre comment un système de quorum peut ne pas tolérer les pannes franches, nous proposons d'étudier le système de quorum en étoile comme défini dans (Holzman *et al.*, 1995). Chacun de ses quorums s'interesectent en un seul nœud commun, qui représente le centre d'une étoile. En dépit de la réplication aux membres du quorum, l'intersection n'est pas elle-même répliquée. Ainsi la cohérence ne peut être assurée lorsqu'une panne franche a lieu. La définition formelle du système de quorum en étoile est la suivante.

Définition 3 (Système de quorum en étoile) *Soit l'univers* $U = \{u_1, u_2, ..., u_n\}$. *Le* système de quorum en étoile *est un système de quorum* $\mathcal{Q}$ *constitué de* $n-1$ *quorums* $\{u_1, u_2\}, \{u_1, u_3\}, ..., \{u_1, u_n\}$.

À partir de la définition 3, il est facile de remarquer qu'une seul panne franche peut provoquer la panne du système de quorum en étoile. En effet, si le nœud u_1 tombe en panne, alors plus aucun quorum n'est actif et le système de quorum tombe en panne.

4.1.2. *Un système de quorum en arbre*

Une illustration un peu différente de ce problème est le système de quorum en arbre tel qu'il a été défini dans (Agrawal *et al.*, 1990). Ce système de quorum est différent puisqu'il correspond a un système *biquorum* où il n'est pas nécessaire d'avoir une intersection non vide entre tout couple de quorums. Plus précisément, il est composé de deux types de quorums tels que chaque quorum d'un type intersecte tout quorum de l'autre type. Ce type de systèmes de quorum, qualifié de lecture/écriture (Herlihy, 1986), est particulièrement intéressant dans le cas des opérations en deux phases ou les quorums utilisés lors de la première (resp. seconde) phase n'ont pas besoin d'intersecter les quorums utilisés dans les autres premières (resp. secondes) phases. Dans le cas du système de quorum en arbre, un quorum de lecture se définit par une fonction récursive qui commence à la racine et retourne soit le nœud courant soit la majorité des fils du nœud courant dans l'arbre. Un quorum d'écriture se définit simplement comme prenant, à chaque hauteur de l'arbre, une majorité des nœuds présents et en faisant l'union de ces majorités.

Comme tous les quorums d'écriture contiennent la racine, tous les quorums d'écriture tombent en panne dès que la racine de l'arbre est en panne. Si cela se produit, alors plus aucun quorum d'écriture n'intersecte tous les quorums de lecture.

Il est intéressant pour un système de quorum de supporter les pannes franches. C'est pourquoi il faut soit qu'un quorum tolère la panne d'un de ses nœuds ou alors il

faut que le système tolère la panne d'un de ces quorums. Dans le premier cas, l'intersection doit contenir plus d'un seul nœud pour qu'elle reste non vide en dépit d'une panne. Dans le second cas, il doit exister suffisamment d'intersections pour que le système de quorum persiste en dépit de la panne d'une de ces intersections. Ainsi, il existe deux solutions possibles : une intersection large ou des intersections nombreuses, comme indiquées dans la section suivante.

4.2. *Répliquer l'intersection pour tolérer une seule panne*

Les intersections sont le point important des systèmes de quorum. Pour tolérer les pannes, il est nécessaire que plusieurs nœuds constituent ces intersections de deux façons possibles :

– intersections larges : l'intersection entre n'importe quel couple de quorums a une cardinalité supérieure à 1, ainsi si un des nœuds de l'intersection tombe en panne, les quorums restent actifs ;

– intersections nombreuses : les quorums sont répliqués de façon à ce que plusieurs intersections existent. Si plus aucun nœud n'est actif au sein d'une intersection entre deux quorums, alors il existe une autre intersection entre un autre couple de quorums.

Contrairement aux apparences, ces deux solutions ne sont pas équivalentes, nous indiquons dans la suite comment l'une ou l'autre de ces solutions apparaît comme étant la meilleure en fonction des pannes considérées.

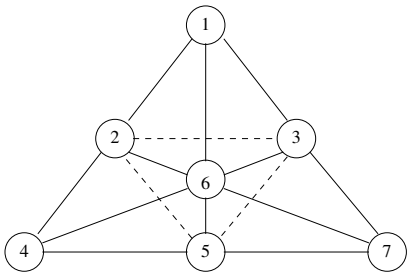

Figure 2. *Un système de quorum en plan-projeté fini avec des intersections nombreuses ; le système contient 7 quorums chacun de taille 3, où chaque élément appartient exactement a 3 quorums :* $\{\{1,2,4\},\{1,3,7\},\{1,5,6\},\{2,3,5\},\{2,6,7\},\{3,4,6\},\{4,5,7\}\}$

Lorsque l'on considère les pannes malicieuses, où le comportement d'un nœud peut sortir de sa spécification initiale, alors il est préférable d'utiliser la première de ces deux solutions. Malkhi et Reiter (2004) définissent les systèmes de quorum *masquant* comme étant des systèmes de quorum ou chaque intersection à une cardinalité au moins égale à $2f + 1$, où f est le nombre de pannes malicieuses supportées. Dans le but d'obtenir une information à jour, il est nécessaire qu'une majorité de réponses

soient correctes, et $2f+1$ est bien le nombre minimum de nœuds communs à contacter pour être certain d'obtenir $f+1$ réponses correctes.

Contacter de larges intersections présente un inconvénient majeur car cela implique de devoir contacter davantage de nœuds lors d'un accès à un quorum, puisqu'une intersection large nécessite que la taille des quorums associés soit large elle aussi. Contrairement aux modèles à pannes malicieuses, le modèle à pannes franches n'a pas besoin d'intersections larges et les intersections nombreuses sont suffisantes. Ainsi, dans la suite de cet article, nous nous intéresserons plus particulièrement aux systèmes de quorum avec intersections nombreuses.

Maekawa (1985) suggère des systèmes de quorum avec des intersections nombreuses mais toutes de cardinalité 1. Le système est plus précisément composé de m^2+m+1 nœuds où $m=p^k$, p est un nombre premier et k est une constante, et il contient $m+1$ quorums chacun de taille $m+1$. La figure 2 décrit un tel système de quorum, où $m=2$, qui illustre les intersections nombreuses.

4.3. *Tolérer davantage de pannes*

Une autre propriété interessante est celle d'intersections disjointes. Cette propriété est assurée par un système de quorum, si et seulement si, ses quorums s'intersectent en des nœuds disjoints. Les intersections disjointes présentent une propriété indépendante de celles d'intersections larges ou nombreuses. Voici la définition formelle de la propriété des intersections disjointes :

Définition 4 (Intersections disjointes) *Un système de quorum $\mathcal{Q}$ défini sur un univers U vérifie les* intersections disjointes *si pour tout triplet de quorum $Q_1, Q_2, Q_3 \in \mathcal{Q}$, on a $Q_1 \cap Q_2 \cap Q_3 = \emptyset$.*

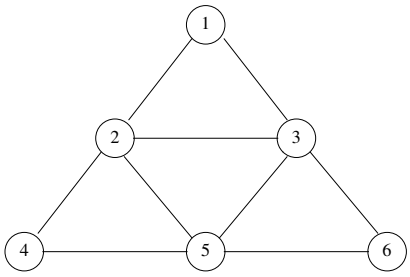

Figure 3. *Un système de quorum basé sur le 3-gon vérifiant les intersections disjointes ; le système contient 4 quorums, chacun de taille 3 et où chaque élément appartient à au plus deux quorums :* $\{\{1,2,4\},\{1,3,6\},\{2,3,5\},\{4,5,6\}\}$

La figure 3 présente un exemple de système de quorum 3-gon qui vérifie les intersections disjointes. Ce système de quorum a été proposé dans (Kuo *et al.*, 1997). Il est

important de remarquer que certains systèmes de quorum dit n-gon ne sont pas disjoints (pour certaines valeurs de $n \neq 3$), le 3-gon étant un cas particulier pour illustrer la propriété des intersections disjointes.

La tolérance aux pannes de systèmes de quorum vérifiant les intersections disjointes est intéressante puisqu'au moins $|\mathcal{Q}|$ nœuds doivent tomber en panne pour que le système soit en panne lorsque chaque nœud appartient à plusieurs quorums. Un système de quorum, dit en grille (Lee *et al.*, 2003) car il consiste en un biquorum de lignes et de colonnes où chaque ligne de la grille intersecte chaque colonne, respecte aussi la propriété des intersections disjointes. Pour un système de quorum en grille, avec $Q_1, Q_2, Q_3 \in \mathcal{Q}$ nous avons $Q_1 \cap Q_2 \cap Q_3 = \emptyset$ alors que tout couple de quorums de types différents ont une intersection non vide. Si tous les $|\mathcal{Q}|$ nœuds situés sur la diagonale de cette grille tombent en panne, alors plus aucun quorum n'est actif et le système tombe en panne.

5. Tolérer un nombre non borné de pannes franches *via* reconfiguration

Le dynamisme du système peut induire l'accumulation de pannes puisque de nouveaux nœuds rejoignent sans cesse le système. Il est donc crucial de tolérer l'accumulation de pannes franches. Cette section propose des techniques pour en tolérer une quantité non bornée.

5.1. *La reconfiguration comme le remplacement de quorums*

Dans les systèmes dynamiques, le nombre de pannes est potentiellement infini. Alors que la réplication permet de tolérer un nombre limite de pannes, elle ne permet pas de tolérer les modèles de pannes propres aux systèmes dynamiques. Dans un système dynamique, un mécanisme doit être effectué régulièrement afin de tolérer les pannes qui s'accumulent perpétuellement. Ici, nous décrivons un mécanisme possible appelé *reconfiguration*.

Les opérations sont divisées en phases qui consistent pour un client à contacter un quorum (généralement à plusieurs reprises). Ces opérations, basées sur les quorums, doivent pouvoir s'exécuter et respecter la cohérence imposée. Ainsi le mécanisme de reconfiguration ne doit pas affecter les opérations de lectures écritures en les bloquant ou en violant l'atomicité. La reconfiguration consiste à remplacer les quorums par des nouveaux quorums sans panne. Le but est d'assurer que tous les nœuds de quelques quorums soient toujours actifs (en général deux suffisent, comme expliqué ultérieurement).

Herlihy (1986) divise en quatre étapes ce que peut être un mécanisme de reconfiguration simple. Celles-ci sont les suivantes : *(i)* récupérer l'état courant de l'objet ; *(ii)* stocker cet état ; *(iii)* initialiser une nouvelle configuration ; *(iv)* mettre à jour la nouvelle configuration, pour pouvoir enlever les précédentes.

5.2. *Indépendance des opérations*

Certains travaux, comme (Martin *et al.*, 2004), proposent de stopper les opérations durant la reconfiguration. Par conséquent une reconfiguration bloquante peut empêcher les opérations de terminer, et les opérations dépendent ainsi de la reconfiguration. Dans la suite nous nous intéressons à l'indépendance des opérations par rapport à la reconfiguration.

Afin que les opérations soient indépendantes et puissent terminer indépendamment du fonctionnement de la reconfiguration, cette dernière doit informer les opérations sur la bonne configuration à utiliser. Lorsqu'une configuration est nouvellement installée, les opérations doivent l'utiliser plutôt qu'une configuration antérieure. Cependant comme des opérations peuvent être en cours d'exécution lorsqu'une nouvelle configuration est installée, l'opération (ou tout au moins sa phase courante qui lui sert à contacter un quorum) doit recommencer son exécution pour utiliser la nouvelle configuration et les nouveaux quorums associés.

Lynch et Shvartsman (2002), proposent une mémoire reconfigurable dont les opérations sont indépendantes de la reconfiguration. Ils définissent deux procédures distinctes à la base de toute reconfiguration :

– l'installation de configuration introduisant une nouvelle configuration et permettant à plusieurs configurations de cohabiter ;

– la mise à jour de la configuration permettant de rassembler les informations présentes dans toutes les configurations au sein de la nouvelle configuration.

Les premières tentatives (Lynch *et al.*, 1997; Englert *et al.*, 2000) consistent à utiliser un reconfigureur centralisé. Dans ce cas, un seul nœud initie la reconfiguration et s'assure de son bon fonctionnement. L'inconvénient de cette centralisation est qu'elle peut être victime d'une seule panne. Si le reconfigureur tombe en panne, alors la reconfiguration devient impossible et le système n'est plut tolérant au dynamisme.

5.3. *La reconfiguration décentralisée*

Afin d'assurer que la reconfiguration puisse s'exécuter en dépit des pannes, il est nécessaire de l'exécuter de façon décentralisée. RAMBO (Lynch *et al.*, 2002) propose l'utilisation d'un algorithme de consensus permettant à plusieurs participants de se mettre d'accord sur une nouvelle configuration à installer. Cet algorithme requiert un algorithme de consensus indépendant, comme Paxos (Lamport, 1989) pour assurer qu'une nouvelle configuration soit choisie de façon décentralisée.

Comme évoqué précédemment, il y a deux mécanismes au sein de la procédure de reconfiguration. Un algorithme de reconfiguration doit installer une nouvelle configuration utilisable afin que l'algorithme puisse continuer en dépit du dynamisme, mais aussi supprimer les configurations précédentes afin d'assurer que l'algorithme arrête d'utiliser des configurations devenues obsolètes. Une implémentation de ces méca-

nismes dans RAMBO permet d'installer une nouvelle configuration à la demande. Cependant le mécanisme de recyclage permettant d'informer les clients potentiels que les configurations précédentes sont devenues obsolètes est indépendant. Plus précisément, ce mécanisme de recyclage ne dépend pas de l'introduction d'une nouvelle configuration. Ne pas activer ce mécanisme au bon moment conduit soit à multiplier le nombre de configurations présentes dans le système soit à communiquer inutilement.

5.4. *Suppression de configurations*

Dans cette première version de l'algorithme RAMBO, un client doit contacter plusieurs configurations afin d'exécuter ses opérations. Durant la période où plusieurs configurations cohabitent, chaque nœud doit contacter non pas un seul quorum par phase de son opération mais un ensemble de quorums : un quorum par configuration présente. De plus, le nombre de configurations actives cohabitantes peut croître arbitrairement si le taux de recyclage n'est pas assez élevé, ainsi la complexité en nombre de message croît de manière correspondante pour toute opération au fur et à mesure que le temps passe.

Afin d'empêcher le nombre de configurations de croître infiniment si la fréquence de recyclage est trop faible, la nouvelle version de RAMBO, appelée RAMBO II (Gilbert *et al.*, 2003) intègre un mécanisme de recyclage agressif permettant de supprimer non plus une seule configuration mais un ensemble de configurations en même temps. L'idée est d'identifier la dernière configuration, de récupérer les informations contenues dans les anciennes configurations, de transmettre ces informations à la dernière configuration avant de supprimer toutes les configurations sauf la dernière. Ce mécanisme empêche le nombre de configurations présentes de croître infiniment si le recyclage est effectué de façon régulière. Cependant, ce mécanisme reste indépendant de l'installation de nouvelle configuration et ne permet pas de restreindre le nombre de configurations présentes à une petite constante (une ou deux par exemple).

RAMBO III (Gramoli, 2004) améliore les précédentes versions en utilisant les accès aux quorums à la fois pour obtenir un consensus sur le choix de la nouvelle configuration, et pour reconfigurer le système. Ce couplage repose sur la combinaison de l'algorithme Paxos (Lamport, 1989) avec l'algorithme de RAMBO permettant ainsi de limiter le nombre de configurations non-utilisées et d'accroître substantiellement sa tolérance au dynamisme.

La solution proposée dans *Reconfigurable Distributed Storage* (RDS) (Chockler *et al.*, 2009) repose sur RAMBO III et permet de limiter le nombre de configurations en couplant l'installation et la suppression pour qu'il n'existe jamais plus de deux configurations présentes dans le système au même moment. De fait, chaque phase exécutée par un client ne doit jamais contacter plus de deux quorums ce qui permet de s'abstraire des hypothèses de RAMBO et RAMBO II concernant l'activité d'anciens quorums.

5.5. *Ordonner les configurations en utilisant le consensus*

Le consensus permet à différents nœuds de se mettre d'accord sur une décision à prendre. Il est nécessaire que les configurations soient choisies de façon uniforme par tous les nœuds. Étant donné que la suppression des configurations obsolètes ainsi que l'exécution des opérations utilisent des échanges de messages entre quorums, il est naturel d'utiliser un algorithme de consensus également basé sur l'échange d'information entre quorums. Paxos est un algorithme proposé par Lamport qui permet de résoudre le problème du consensus en utilisant des échanges de messages entre quorums. Celui-ci permet à certains nœuds de proposer des nouvelles configurations et de laisser décider les autres nœuds par un système de vote. La particularité est, d'une part, qu'il est impossible pour les nœuds de décider de deux configurations différentes du fait de ce système de vote. D'autre part, si les propositions cessent à un moment donné de rentrer en conflit alors une décision uniforme est prise par l'ensemble des nœuds participant à l'algorithme.

Il est ainsi possible, en utilisant Paxos, de permettre aux noeuds de se mettre d'accord sur une configuration commune à installer telle que tous les nœuds n'utiliseront que celle-ci. Cela revient donc à assurer un ordre total sur les configurations successivement installées, chaque nœud ne pouvant installer la $i + 1^{\text{e}}$ configuration que si la i^{e} a déjà été installée. Bien sûr l'inconvénient est que des conflits entre les votants empêchent la décision d'être prise empêchant l'installation de la nouvelle configuration. Mais lorsque le système stabilise et que l'échange des messages est relativement rapide alors la décision est prise très rapidement car Paxos permet de résoudre le consensus en deux échanges de message (après que des configurations aient été proposées) lorsque de bonnes conditions pratiques sont réunies. Ce délai a été prouvé comme étant minimal et la variante de Paxos permettant ces performances est appelée « Paxos Rapide » (Lamport, 2006).

Paxos a tout d'abord été décrit comme une suite de règles informelles dans un rapport technique de 1989, publié neuf ans plus tard. Cette présentation décrit le fonctionnement d'un parlement ancien sur l'île grecque de Paxos où les députés travaillent à mi-temps. Ce papier explique comment les décrets peuvent être votés bien que les députés ne soient pas tous dans la chambre du parlement au même moment et peuvent ainsi avoir du mal à communiquer. Cet algorithme permet d'assurer la cohérence en dépit d'une quantité arbitraire d'absences pourvu que le système retourne à un état où plus de la moitié des participants sont présents.

La figure 4 décrit la séquence normale des messages envoyés lors d'un référendum dans l'algorithme Paxos. Une phase y est représentée par un échange de message et chaque nœud peut jouer plusieurs rôles parmi proposeur, accepteur et appreneur. Les accepteurs s'interdisent de participer à un référendum s'ils découvrent un autre référendum avec une valeur plus grande indiquant qu'il est plus récent (phase 1b). Les accepteurs peuvent voter pour un référendum s'ils ne se le sont pas interdis avant. Un référendum gagne si un quorum d'accepteur l'a voté, mais les accepteurs d'un même quorum peuvent voter pour différents référendums. Si cela se produit, alors

Phase 1a. Un coordinateur envoie un nouveau référendum a un quorum de proposeurs.

Phase 1b. Lorsque le message est reçu, chaque proposeur répond en envoyant la valeur du plus grand référendum (s'il en existe un) pour lequel il a voté. Chacun des proposeurs qui découvre un référendum plus à jour s'interdit de participer à tout référendum précédent.

Phase 2a. Après avoir reçu les réponses d'au moins un quorum d'accepteurs, le coordinateur choisit une nouvelle valeur pour son référendum et informe un quorum d'accepteurs du choix de cette valeur pour ce référendum.

Phase 2b. Tout accepteur qui découvre cette valeur et qui ne s'est pas interdit de voter à ce référendum, peut voter pour celui-ci et donner cette valeur aux appreneurs (et au coordinateur). Lorsque les appreneurs apprennent qu'un quorum d'accepteurs a voté pour cette valeur, ils décident de cette valeur.

Figure 4. *Description de l'algorithme Paxos*

un nouveau référendum avec un identifiant plus grand doit être démarré pour assurer qu'à un moment un référendum gagne. Le consensus est résolu lorsque les appreneurs reçoivent un message les informant de la valeur décidée ; cette réception met fin au référendum (phase 2b).

5.6. *Reconfiguration décentralisée rapide*

Bien sûr, les reconfigurations doivent être suffisamment fréquentes pour que le nombre de défaillances s'accumulant reste sous le seuil critique toléré (typiquement, au moins un quorum doit resté actif). Autrement dit, plus le mécanisme de reconfiguration est rapide, plus le temps de remplacement de serveurs défaillants est court, ce qui rend le système davantage robuste. Cet aspect est la principale motivation de RAMBO III et RDS. qui proposent un mécanisme de reconfiguration rapide. Le protocole de reconfiguration qui y est présenté utilise la version optimisée Paxos Rapide, un algorithme de consensus pour permettre au système de choisir une nouvelle configuration à installer. L'algorithme remplace suffisamment de serveurs pour assurer qu'à tout moment au moins un quorum est actif.

Plus précisément, un système de quorum est élu par un quorum resté actif, puis les anciens quorums sont remplacés par de nouveaux quorums actifs. L'originalité de cet algorithme réside dans la superposition des deux mécanismes. Puisque décider d'une nouvelle configuration et remplacer l'ancienne configuration par une nouvelle sont deux problèmes utilsant les quorums, le couplage de ces deux mécanismes résulte en un algorithme aussi rapide que le moins rapide des deux mécanismes sous-jacents. Par conséquent lorsque le système stabilise et qu'un leader est élu, RDS reconfigure le système en trois délais de message. La figure 2 résume la complexité en temps des opérations et de la reconfiguration dans les algorithmes présentés ci-avant. RDS améliore substantiellement les autres algorithmes en minimisant la complexité en temps.

Algorithme	Latence min. des operations	Latence max. de configuration	Delai max. de suppression de config.
RAMBO	4δ	$10d + \epsilon$	$4(s-1)d + \epsilon$
RAMBO II	4δ	$10d + \epsilon$	$4d + \epsilon$
RDS	2δ	$5d + \epsilon$	0

Tableau 2. *Complexité en temps d'exécution de* RAMBO, RAMBO II, *et* RDS *; la lettre δ (réciproquement d) est une borne inférieure (réciproquement supérieure) sur le délai de message, s est le nombre de configurations présentes et $\epsilon = O(1)$ est une constante indépendante du délai des messages*

La reconfiguration rapide est utile pour la tolérance aux défaillances dans les systèmes distribués dynamiques. Plus le système renouvelle rapidement l'ensemble des serveurs actifs et plus le système est tolérant aux défaillances. Cependant, ce mécanisme peut paraître trop complexe pour un système où les ressources sont limitées et où le système doit stabiliser durant suffisamment longtemps.

De plus, cette technique suppose que les nœuds accèdent directement à tous les participants d'un quorum. Cela nécessite pour chaque client potentiel de connaître tous les éléments d'au moins un quorum. La tolérance aux fautes résultant de la réplication des quorums, il est même préférable que chaque client potentiel connaisse l'ensemble des participants de plusieurs quorums. Ainsi pour un petit nombre de fautes se produisant durant le lapse de temps nécessaire à la reconfiguration, un client pourra plus facilement trouver un quorum dont l'ensemble des participants sont toujours actifs.

Cette connaissance globale peut avoir des conséquences dramatiques sur les performances de l'algorithme lorsque la taille et le nombre de quorums tend à devenir très grand. En effet, dans un système où le nombre de clients est très grand, la charge induite peut nécessiter également un grand nombre de serveurs. Dans ce cas, le coût en communication nécessaire à chaque opération peut amener des congestions dans le réseau.

6. Des alternatives efficaces en communication

Dans cette section, nous nous intéressons aux coûts en communication liés à l'émulation de la mémoire partagée. Nous proposons des solutions pour diminuer cette complexité.

6.1. *Éviter les communications à l'ensemble des participants*

À cause de l'intérêt croissant pour les systèmes dynamiques à grande échelle, une approche radicalement différente a été adoptée. Cette approche est basée sur le prin-

cipe de *localité*. La localité est la qualité d'un algorithme à tirer parti de l'information située dans un voisinage proche plutôt que l'information éloignée. L'idée est de permettre à chaque nœud de conserver une partie de l'information du système en fonction de son emplacement dans le réseau des communications. Par conséquent, un algorithme utilisant la localité minimise l'impact induit par les événements dynamiques : la réparation d'une défaillance, bien qu'en moyenne plus fréquente que la reconfiguration, a un coût plus faible.

6.1.1. *Réduire les bavardages*

Les solutions proposées précédemment possèdent un inconvénient lorsqu'elles sont déployées à grande échelle. En effet, dans un système tel qu'internet, où la bande passante est limitée, la communication entre un grand nombre de nœuds provoque des congestions au sein du réseau. Dans les approches citées précédemment, les nœuds clients connaissent le système de quorum de l'objet afin de pouvoir exécuter une opération. Le client effectue une opération en envoyant un message simultanément à l'ensemble des nœuds du système de quorum. Comme la reconfiguration assure qu'à tout moment un quorum est actif, le client reçoit finalement un message de la part de chaque serveur d'au moins un quorum. L'opération se poursuit éventuellement par une seconde phase d'échange entre le client et un quorum. Par conséquent, le nombre de messages requis à chaque reconfiguration pour que tous les n nœuds soient au courant du nouveau système de quorum est $O(n^2)$.

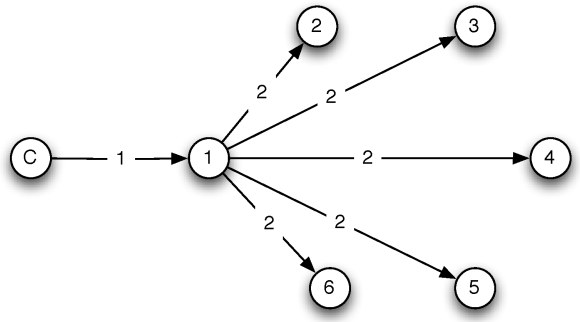

Figure 5. *Approche par connaissance globale : le client C effectue chaque opération de lecture et écriture en temps $O(1)$; tous les serveurs (ou presque) du quorum sont contactés en parallèle*

Gramoli, Musial et Shvartsman (2005) proposent une amélioration substantielle dans le but de réduire la complexité en communication de quadratique à linéaire en n. Pour cela, des rôles spécifiques sont assignés aux serveurs et des quorums particuliers doivent être utilisés. Un nœud a un rôle de *propriétaire* s'il se considère localement comme faisant partie d'un quorum à jour – un quorum du dernier système de quorum installé.

Lorsqu'une reconfiguration a lieu, les informations relatives au nouveau système de quorum sont disséminées uniquement entre propriétaires. Ainsi la diminution en nombre de messages est proportionnelle au carré de la différence entre le nombre de nœuds et le nombre de propriétaires. Cette amélioration constitue une première étape pour le passage à grande échelle. Lorsque le nombre de nœuds devient grand, diminuer le nombre de serveurs permet de diminuer le nombre de propriétaires et donc diminue, de surcroît, la complexité en communication d'une reconfiguration.

Cependant, malgré la diminution du coût de communication pour la reconfiguration, tout nœud doit pouvoir trouver l'objet sur lequel effectuer les opérations. Pour cela, tout client doit pouvoir envoyer sa requête à un des propriétaires du système de quorum actuel bien que ces systèmes se succèdent au fil du temps. L'hypothèse utilisée est qu'au moins un propriétaire de chaque système de quorum reste actif. Lorsqu'un client contacte un système qui a été remplacé, il contacte au moins un nœud de ce système permettant de retrouver le système de quorum successeur. Par conséquent les anciens propriétaires permettent petit à petit de recontacter les propriétaires actuels. Cette propriété reste difficile à satisfaire dans un système fortement dynamique.

La complexité d'accès à un quorum est représentée sur la figure 5 où la taille q des quorums est $q = 6$. Les cercles numérotés indiquent les nœuds d'un quorum et le cercle noté C indique le client y accédant. Il est clair que l'accès à un quorum s'effectue en temps constant puisqu'un message est nécessaire pour contacter un propriétaire et un message est nécessaire pour consulter les autres serveurs du quorum. Comme les opérations nécessitent un nombre constant d'accès aux quorums, chaque opération s'effectue dans ce cas en temps constant.

6.1.2. *Diminuer la connaissance pour passer à grande échelle*

En restreignant encore davantage l'information que chaque nœud maintient sur le système, un mécanisme de réparation peut être exécuté avec un coût moindre que dans le cas d'une reconfiguration. Supposons, par exemple, qu'un des serveurs quitte le système de quorum, il sera seulement nécessaire à ses voisins, les nœuds pouvant communiquer directement avec lui, de réparer cette panne soit en agissant à sa place soit en répliquant la donnée à un autre nœud qui deviendra un serveur remplaçant.

L'inconvénient direct d'une telle solution est l'augmentation de la latence des opérations. Bien que la complexité des messages soit diminuée, la complexité en temps d'une opération est augmentée. En effet, diminuer la connaissance des nœuds à un voisinage restreint empêche de contacter les quorums en un nombre constant de messages. Dans ce cas, le premier serveur contacté devra contacter le serveur suivant et ainsi de suite. Le nombre de messages requis sera alors proportionnel à la taille du quorum $O(q)$.

6.1.3. *Mémoire atomique adaptative*

Il est intéressant de noter que le paradigme de la réparation locale ou globale est lié au paradigme du routage réactif ou proactif. La réparation locale est telle que seul un

routage réactif peut être adopté durant la phase d'accès aux quorums. En effet, aucun serveur n'est connu lorsque le routage commence. Les serveurs sont ainsi découverts au fur et à mesure de l'exploration. À l'opposé, la réparation globale nécessite un routage proactif où l'ensemble des serveurs est connu lorsque l'accès au quorum débute. Les algorithmes utilisant la première de ces deux techniques sont appelés *adaptatifs* alors que ceux utilisant la seconde sont appelés *non adaptatifs* comme définis dans (Naor *et al.*, 2003).

Récemment, certains auteurs se sont intéressés aux quorums dynamiques (Naor *et al.*, 2003; Nadav *et al.*, 2005; Abraham *et al.*, 2005), dont les serveurs changent progressivement avec le temps. Dans (Abraham *et al.*, 2005), un nouveau nœud est inséré par un ajout de lien dans une structure en graphe De Bruijn, tandis que la localité est définie de la manière suivante : si deux nœuds sont reliés par un lien du graphe alors ils sont voisins. Par ailleurs, le Dynamic Path (Naor *et al.*, 2003) définit des quorums dans une couche de communication logique utilisant des cellules de Voronoi pour déterminer les relations de voisinage : deux nœuds détenant deux cellules accolées sont voisins. Les cellules se réadaptent automatiquement en fonction du placement des nouveaux nœuds ou des nœuds qui partent.

Le système de quorum dynamiques « et/ou » (Nadav *et al.*, 2005) consiste en une structure arborescente binaire dont les feuilles représentent les serveurs. Un quorum est choisi en partant de la racine et en descendant alternativement un chemin (ou) ou les deux chemins (et) allant vers les feuilles. Les feuilles aux terminaisons des chemins choisis constituent les serveurs d'un quorum. Les quorums sont dynamiques puisque durant le parcours réactif de l'arbre, des nœuds peuvent être supprimés ou ajoutés.

Des émulations de mémoire partagée en modèle à passage de message utilisent ces nouveaux types de quorums pour leur adaptation aux systèmes dynamiques à grande échelle, non seulement du fait de leur dynamisme inhérent mais également du fait de leur propriété de localité. Par exemple, SAM (Anceaume *et al.*, 2005) et SQUARE (Gramoli *et al.*, 2007a) utilisent des spécificités dynamiques afin d'adapter la structure des systèmes de quorum aux variations de charges imprévisibles en contexte grande échelle. De plus, ces solutions utilisent une grille supportant des départs et arrivées de nœuds ainsi le nombre de voisins est constant et la réparation locale ne nécessite qu'un nombre constant de messages. Ces solutions reposent sur un algorithme adaptatif pour accéder aux quorums et les opérations nécessitent $O(q)$ messages successifs. La figure 6 représente un accès au quorum : le client C contacte tour à tour chacun des serveurs d'un quorum.

Certaines méthodes adaptatives étendent la notion de localité à un nombre de nœuds logarithmique en la taille du système. Ce type d'approche (Muthitacharoen *et al.*, 2004) permet d'effectuer des opérations en temps $O(\log q)$.

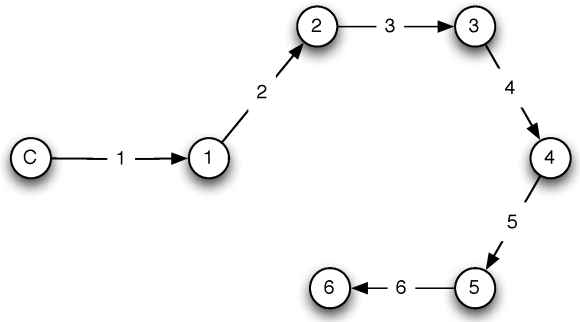

Figure 6. *Approche adaptative : le client C accède à un quorum en contactant chaque serveurs du quorum tour à tour ; les opérations sont exécutées en temps $O(q)$*

6.2. *Assurer les intersections avec forte probabilité*

Étant donné les différentes solutions étudiées précédemment, il semble difficile d'obtenir à la fois des opérations rapides et une cohérence assurée sans surcoût de communication. De plus, afin de préserver les conditions de vivacité, les solutions présentées émettent des hypothèses fortes. Par exemple, Chockler *et al.* (2009) présupposent que le système stabilise et que les temps des messages sont bornés pour que la reconfiguration termine. Aussi, Anceaume *et al.* (2005) requièrent la présence de détecteurs de fautes parfaits. Récemment et face aux difficultés rencontrées pour assurer des propriétés déterministes, des solutions probabilistes ont vu le jour.

Abraham *et al.* (2005) proposent une couche de communication logique en graphe De Bruijn afin d'assurer un degré constant. Cette structure est dynamique, comme mentionnée précédemment, et chaque événement dynamique nécessite une réparation locale impliquant $O(\log n)$ messages avant que la structure stabilise à nouveau (car tous les n nœuds font partie de la structure). Les quorums de cette structure s'intersectent avec grande probabilité. Ce résultat est atteint en exécutant $O(\sqrt{n})$ marches aléatoires en parallèle, de longueur $O(\log n)$ chacune. Ainsi le temps nécessaire pour accéder un quorum est de $O(\log n)$ délais de message (avec $q = \sqrt{n}\log n$). La figure 7 présente simplement l'accès à un quorum en utilisant deux marches aléatoires en parallèle.

Malkhi *et al.* (2001) définissent un système de quorum probabiliste. Appelé système de quorum ϵ-*intersectant*, il contient des quorums qui s'intersectent avec probabilité $1 - \epsilon$. Ces solutions sont basées sur l'accès à un certain nombre de nœuds choisis aléatoirement tels que lors de deux accès, il soit probable qu'un nœud soit contacté deux fois, autrement dit que l'intersection existe. De tels quorums sont essentiellement définis par leur taille q et cette taille est par définition élevée.

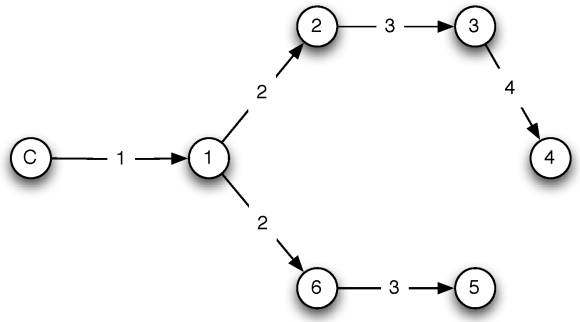

Figure 7. *Approche par marches aléatoires : deux marches aléatoires sont exécutées en parallèle afin de contacter l'ensemble des serveurs du quorum ; le temps d'accès est donc linéaire en la taille des marches aléatoires*

Bien que la taille de ces quorums ne peut pas être considérablement diminuée, contrairement au méthodes précédentes, les quorums probabilistes apportent des améliorations en termes de complexité et posent de nouvelles questions : Quels seraient les critères de cohérence envisageables dans les systèmes dynamiques à grande échelle ? Faut-il définir un nouveau critère de cohérence atomique probabiliste ou doit-on se contenter de cohérence faible ?

6.3. *Affaiblir la cohérence forte*

De nombreux travaux sont dédiés à la définition formelle de critères de cohérence. Parmi ceux-ci, l'atomicité est le critère le plus fort (définition 2). Ainsi si une implémentation vérifie ce critère alors cette même implémentation vérifie tout autre critère.

L'atomicité présente de nombreux avantages tels que la propriété de localité comme mentionné précédemment. D'autres critères moins forts on vu le jour. La sûreté et la régularité (Lamport, 1986) sont deux propriétés qu'il est plus facile d'assurer que l'atomicité. Ces deux propriétés nécessitent qu'une opération de lecture qui n'est pas en concurrence avec une opération d'écriture retourne la dernière valeur écrite. L'atomicité faible nécessite au contraire que si deux opérations de lecture retourne une même valeur, alors toute lecture ultérieure renvoie la même valeur ou une valeur plus à jour.

Un classe de cohérence intéressante, appelée hybride (Attiya *et al.*, 1998) bénéficie des critères faibles et forts à la fois. Elle définit des ordres forts et faibles sur les opérations. Les opérations ordonnées fortement ont un ordre sur lequel tous les nœuds du système sont d'accord alors que les opérations ordonnées faiblement ont un ordre qui peut être différent en fonction du point de vue d'un nœud. Un critère de cohérence

fournissant à la fois l'ordre faible et fort en fonction de l'objet qui est accédé a été défini par Dubois *et al.* (1988; 1990). Tous ces critères de cohérence sont déterministes, ainsi pour qu'une implémentation vérifie la cohérence hybride, chaque type d'opération doit être défini à l'avance comme étant fort ou faible mais ne peut pas être les deux à la fois.

D'autres critères de cohérence incluent la notion d'aléa. Par exemple dans (Afek *et al.*, 1995), une valeur ancienne peut être retournée en fonction du type courant de l'objet accédé. Par ailleurs, certains travaux ne posent pas de contrainte sur la latence des opérations mais assurent que lorsqu'une opération de lecture termine, elle renvoie une valeur à jour (Shavit *et al.*, 1998). Aussi, les registres aléatoires (Lee *et al.*, 2005) permettent de retourner des valeurs anciennes tant que n'importe quelle valeur écrite est lue ou réécrite ultimement. Finalement, un critère de cohérence appelé atomicité probabiliste (Gramoli *et al.*, 2007b) autorise toute opération à échouer en renvoyant une valeur qui n'est plus à jour, mais seulement avec très faible probabilité.

Ce dernier critère de cohérence peut être utilisé pour viser un niveau de qualité de service exprimé par la probabilité que les opérations effectuées par ce service aient du succès. Pour formaliser cette notion de qualité qui n'est plus déterministe, il nous faut modifier le critère de cohérence défini au début de cet article (définition 2) en *atomicité probabiliste*. Tout d'abord nous rappelons, les points 1 et 3 de la définition initiale :

– $(\pi_1, \pi_2)-$*ordre* : si la réponse d'une opération π_1 précède l'invocation de l'opération π_2 alors il n'est pas possible d'avoir $\pi_2 \prec \pi_1$;

– $(\pi_1, \pi_2)-$*retour* : la valeur retournée par une opération de lecture π_2 est la valeur écrite par la dernière écriture la précédant π_1 (par définition de $\prec$) et s'il n'existe pas de telle opération d'écriture alors la valeur retournée est la valeur par défaut.

La définition d'atomicité probabiliste est la suivante.

Définition 5 (Atomicité probabiliste) *Soit x un objet accessible en lecture et écriture. Soit H une séquence complète d'invocations et de réponses de lecture/écriture appliquées à x. La séquence H satisfait l'atomicité probabiliste s'il existe un ordre partiel $\prec$ sur les opérations à succès tel que les propriétés suivantes soient vérfiées :*

1) soit π_1 une opération à succès. N'importe quelle opération π_2 satisfait $(\pi_1, \pi_2)-$ordre avec forte probabilité (si π_2 ne le satisfait pas, alors il est considéré comme sans succès) ;

2) si π_1 est une opération d'écriture et π_2 une autre opération, alors soit $\pi_2 \prec \pi_1$ ou bien $\pi_1 \prec \pi_2$;

3) soit π_1 une opération à succès. N'importe quelle opération π_2 satisfait $(\pi_1, \pi_2)-$retour avec forte probabilité (si π_2 ne le satisfait pas, alors il est considéré comme sans succès).

Il est important de mentionner qu'un tel critère de cohérence probabiliste est inutile dans beaucoup de cas de figure. En effet, des systèmes critiques ne peuvent

se permettre des garanties probabilistes mais nécessitent des garanties déterministes. Aussi, la faible probabilité qu'une opération échoue peut avoir un impact sur les opérations ultérieures. Par exemple, une opération d'écriture qui n'écrirait que partiellement pourrait avoir du succès si une opération de lecture retournait cependant sa donnée. Dans ce cas, doit-on considérer que les opérations de lecture retournant des valeurs anciennes ont échoué ou devrait-on considérer que toutes les opérations ont eu du succès. Une alternative à cette ambiguïté serait de combiner des opérations d'écriture déterministes (n'échouant jamais) et des opérations de lectures probabilistes.

Il existe cependant des applications concrètes, dans les systèmes à grande échelle, pouvant profiter d'un tel critère de cohérence probabiliste. En effet, l'échange d'informations (renseignements géographiques ou musicaux) entre entités mobiles qui se rencontrent de façon imprévue ne nécessite nullement de garanties déterministes. Par ailleurs, rechercher une information apparue au sein d'un système pair-à-pair renvoie souvent des résultats variables voir incohérents car un fichier trouvé à un moment donné peut avoir disparu un instant plus tard. L'atomicité probabiliste garantit une certaine qualité de service suffisante pour nombre d'applications qui privilégient davantage l'échange quantitatif à l'échange qualitatif d'information.

6.4. *Solutions sans structure*

Toutes les solutions décrites précédemment ont un point commun : elles utilisent une couche de communication logique structurée. Étant donné les contraintes imposées par de telles structures, des coûts de communication sont nécessaires pour assurer que les données soient cohérentes ou même subsistent. En effet la solution principale est de réagir aux événements dynamiques par une réparation de la structure.

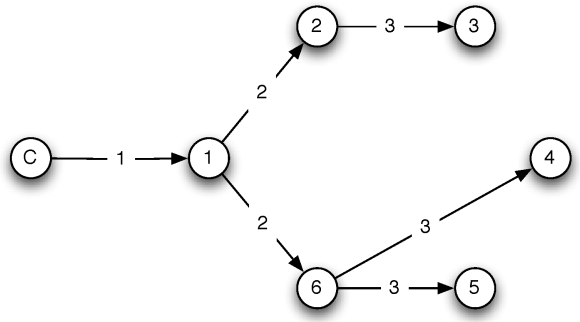

Figure 8. *L'approche par dissémination : des messages sont disséminés au sein des nœuds afin de contacter les serveurs du quorum ; le temps d'accès est donc au plus logarithmique en le nombre de nœuds à contacter*

Par conséquent, diminuer les coûts de communication qu'impliquent le maintien de la cohérence nécessite de définir une mémoire atomique sans structure logique. Gramoli *et al.* (2006; 2007b) évoquent une façon de préserver l'intersection en dépit du dynamisme sans utiliser de structure. Leur approche est donc généralisable à la cohérence des données. Leur idée fondamentale est d'assurer la subsistance d'un noyau constitué de serveurs contenant la donnée critique (*i.e.*, la donnée subsistante ou la dernière valeur écrite) afin que n'importe quel nœud puisse y accéder à tout moment avec forte probabilité. Un travail ultérieur utilisant des quorums temporels (Gramoli *et al.*, 2007b) propose dans un modèle dynamique, légèrement différent de (Gramoli *et al.*, 2006), un système assurant l'atomicité probabiliste (définition 5).

L'idée maîtresse est temporelle dans le sens où un objet subsiste si les opérations sont suffisamment fréquentes par rapport au va-et-vient. En d'autres termes, si un objet est écrit (mis à jour ou modifié) suffisamment souvent sur un nombre suffisant q de serveurs alors l'objet subsiste. Cette technique peut être utilisée afin de définir des quorums ne nécessitant aucun mécanisme de réparation. La figure 8 représente comment accéder de tels quorums avec un protocole de dissémination au sein du réseau. D'une autre façon, il est également possible de définir des protocoles épidémiques utilisant un bavardage régulier pour obtenir des résultats similaires en diminuant les effets de congestion possibles.

Probabilité d'intersection	Va-et-vient $C = 1 - (1 - c)^{\Delta}$	Taille du quorum $q(n = 10^3)$	$q(n = 10^4)$	$q(n = 10^5)$
99 %	statique	66	213	677
	10 %	70	224	714
	30 %	79	255	809
	60 %	105	337	1071
	80 %	143	478	1516
99.9 %	statique	80	260	828
	10 %	85	274	873
	30 %	96	311	990
	60 %	128	413	1311
	80 %	182	584	1855

Tableau 3. *La taille du quorum en fonction de la taille du système et du va-et-vient*

Gramoli *et al.* (2006) montrent que lorsque la portion des nœuds à quitter et rejoindre le système par unité de temps est c, et que la période maximale entre deux opérations est de δ unités de temps, alors on peut paramétrer la taille des quorums afin d'obtenir la qualité de service désirée. Le tableau 3 retranscrit les résultats présentés dans (Gramoli *et al.*, 2006) où C représente la portion des nœuds du système qui ont été remplacés entre deux opérations, durant une période Δ et avec un va-et-vient c. Il est intéressant d'observer que la taille des quorums doit être légèrement augmentée pour tolérer le dynamisme tout en assurant la même probabilité d'intersection que dans un environnement statique. Similairement, lorsque deux fois plus de nœuds sont

remplacés un quorum doit simplement être augmenté de quelques nœuds afin de garantir la même probabilité d'intersection.

7. Conclusion

Cet article présente les défis qui entourent l'émulation de mémoire partagée dans un environnement à passage de message où les nœuds participants ne cessent de quitter le système et où de nouveaux nœuds rejoignent ce système de façon imprévisible. À travers les travaux existants sur les systèmes de quorum, les mécanismes de reconfiguration et les objets en lecture/écriture probabilistes nous avons dressé un état de l'art des approches pour pallier le dynamisme.

Le résultat de notre étude est une classification des systèmes de quorum statiques, dynamiques et probabilistes qui régissent la réplication des données pour tolérer les pannes franches, le va-et-vient et minimiser les coûts associés à la communication qu'induit une telle tolérance.

Je suis convaincu que les systèmes de quorum probabilistes peuvent être adaptés pour différents besoins. Bien que certaines applications, dites critiques, nécessitent des garanties fortes pour les utilisateurs, d'autres types d'applications distribuées visent de gros volumes de communication au détriment des garanties. Typiquement, les réseaux dynamiques proposent un échange d'information dont le volume ne cesse de croître alors même que leur intérêt décroît. L'utilisation d'internet à travers les réseaux sociaux en est un exemple puisqu'il place la quantité d'information devant la pertinence des échanges. Ceux-ci regroupent notamment les réseaux pair-à-pair. Les réseaux mobiles permettent déjà de mettre en relation une clientèle ciblée avec les informations de proximité, et là encore le but est de privilégier la communication de masse par rapport à la non redondance ou l'aspect à jour du message lui-même. Il semble donc primordial pour ces nouveaux types d'applications d'étudier l'émulation de mémoire partagée avec des critères de cohérence plus faibles que ceux étudiés par le passé au profit du passage à l'échelle.

Remerciements

Je souhaite remercier Michel Raynal, Roy Friedman et Alex Shvartsman pour leurs commentaires éclairés lors de la rédaction de ma thèse (Gramoli, 2007) dont ce travail est tiré.

8. Bibliographie

Abraham I., Malkhi D., « Probabilistic quorums for dynamic systems », *Distributed Computing*, vol. 18, n° 2, p. 113-124, 2005.

Afek Y., Greenberg D. S., Merritt M., Taubenfeld G., « Computing with faulty shared objects », *J. ACM*, vol. 42, n° 6, p. 1231-1274, 1995.

Agrawal D., Abbadi A. E., « The tree quorum protocol : an efficient approach for managing replicated data », *Proceedings of the 16th Int'l Conference on Very Large Databases*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, p. 243-254, 1990.

Agrawal D., Choy M., Leong H. V., Singh A., « Mixed consistency : a model for parallel programming », *Proceedings 13th Annual Symposium on Principles of Distributed Computing*, p. 101-110, 1994.

Anceaume E., Gradinariu M., Gramoli V., Virgillito A., « P2P Architecture for Self* Atomic Memory », *Proceedings of 8th Int'l Symposium on Parallel Architectures, Algorithms and Networks*, p. 214-219, Dec., 2005.

Attiya H., Bar-Noy A., Dolev D., « Sharing memory robustly in message-passing systems », *J. ACM*, vol. 42, n° 1, p. 124-142, 1995.

Attiya H., Friedman R., « A correctness condition for high-performance multiprocessors », *SIAM Journal on Computing*, vol. 27, n° 6, p. 1637-1670, 1998.

Chockler G., Gilbert S., Gramoli V., Musial P., Shvartsman A., « Reconfigurable Distributed Storage for Dynamic Networks », *Journal of Parallel and Distributed Computing*, vol. 69, n° 1, p. 100-116, 2009.

Dubois M., Scheurich C., « Memory Access Dependencies in Shared-Memory Multiprocessors », *IEEE Trans. Softw. Eng.*, vol. 16, n° 6, p. 660-673, 1990.

Dubois M., Scheurich C., Briggs F., « Memory access buffering in multiprocessors », *Proceedings of the 13th Annual Int'l Symposium on Computer Architecture*, IEEE Computer Society Press, Los Alamitos, CA, USA, p. 434-442, 1986.

Dubois M., Scheurich C., Briggs F. A., « Synchronization, Coherence, and Event Ordering in Multiprocessors », *Computer*, vol. 21, n° 2, p. 9-21, 1988.

Dutta P., Guerraoui R., Levy R. R., Chakraborty A., « How fast can a distributed atomic read be ? », *Proceedings of the 23th Annual Symposium on Principles of Distributed Computing*, ACM Press, New York, NY, USA, p. 236-245, 2004.

Englert B., Shvartsman A. A., « Graceful quorum reconfiguration in a robust emulation of shared memory », *Proceedings of Int'l Conference on Distributed Computer Systems*, p. 454-463, 2000.

Garcia-Molina H., Barbara D., « How to assign votes in a distributed system », *J. ACM*, vol. 32, n° 4, p. 841-860, 1985.

Gifford D. K., « Weighted voting for replicated data », *Proceedings of the 7th ACM Symposium on Operating systems principles*, ACM Press, p. 150-162, 1979.

Gilbert S., Lynch N. A., Shvartsman A. A., RAMBO II : Implementing atomic memory in dynamic networks, using an aggressive reconfiguration strategy, Technical report, LCS, MIT, 2003.

Goodman J. R., Cache Consistency and Sequential Consistency, Technical Report n° 61, SCI Committee, Mar., 1989.

Gramoli V.*, « RAMBO III : Speeding-up the reconfiguration of an atomic memory service in dynamic distributed system »*, Master's thesis, Université Paris Sud - Orsay, France, Sep., 2004.

Gramoli V., Distributed Shared Memory for Large-Scale Dynamic Systems, PhD thesis, Université Rennes 1, 2007.

Gramoli V., Anceaume E., Virgillito A., « SQUARE : Scalable Quorum-Based Atomic Memory with Local Reconfiguration », *Proceedings of the 22nd ACM Symposium on Applied Computing*, p. 574-579, Mar., 2007a.

Gramoli V., Kermarrec A.-M., Mostefaoui A., Raynal M., Sericola B., « Core Persistence in Peer-to-Peer Systems : Relating Size to Lifetime », *Proceedings of the Int'l Workshop on Reliability in Decentralized Distributed systems*, vol. 4218 of *LNCS*, Springer, p. 1470-1479, Oct., 2006.

Gramoli V., Musial P., Shvartsman A., « Operation Liveness and Gossip Management in a Dynamic Distributed Atomic Data Service », *Proceedings of the 18th Int'l Conference on Parallel and Distributed Computing Systems*, Sept., 2005.

Gramoli V., Raynal M., « Timed Quorum Systems for Large-Scale and Dynamic Environments », *Proceedings of the 11th Int'l Conference on Principles of Distributed Systems*, vol. 4878, Springer-Verlag, p. 429-442, 2007b.

Herlihy M., « A quorum-consensus replication method for abstract data types », *ACM Trans. Comput. Syst.*, vol. 4, n° 1, p. 32-53, 1986.

Herlihy M. P., Wing J. M., « Linearizability : a correctness condition for concurrent objects », *ACM Trans. Program. Lang. Syst.*, vol. 12, n° 3, p. 463-492, 1990.

Holzman R., Marcus Y., Peleg D., « Load Balancing in Quorum Systems », *Proceedings of the 4th Int'l Workshop on Algorithms and Data Structures*, Springer-Verlag, London, UK, p. 38-49, 1995.

Hutto P. W., Ahamad M., « Slow Memory : Weakening Consistency to Enhance Concurrency in Distributed Shared Memories », *Proceedings of the 10th Int'l Conf. on Distributed Computing Systems*, p. 302-311, May, 1990.

Kuo Y., Huang S., « A geometric Approach for constructing Coteries and k-Coteries », *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, p. 402-411, 1997.

Lamport L., « Time, Clocks, and the Ordering of Events in a Distributed System », *Communications of the ACM*, vol. 21, n° 7, p. 558-565, July, 1978.

Lamport L., « On interprocess communication, Part II : Algorithms », *Distributed Computing*, vol. 1, p. 86-101, 1986.

Lamport L., The Part-time Parliament, Technical Report n° 49, Digital SRC, Sept., 1989.

Lamport L., « Fast Paxos », *Distributed Computing*, vol. 19, n° 2, p. 79-103, Oct., 2006.

Lee H., Welch J. L., « Randomized Registers and Iterative Algorithms », *Distributed Computing*, vol. 17, n° 3, p. 209-221, 2005.

Lee H., Welch J. L., Vaidya N. H., « Location Tracking Using Quorum in Mobile Ad Hoc Networks », *Ad Hoc Networks*, vol. 1, n° 4, p. 371-381, 2003.

Lipton R. J., Sandberg J. S., PRAM : A Scalable Shared Memory, Technical Report n° CS-TR-180-88, Dept. of Computer Science, Princeton University, Sept., 1988.

Lynch N., *Distributed Algorithms*, Morgan Kaufmann Publishers, 1996.

Lynch N. A., Shvartsman A. A., « Robust emulation of shared memory using dynamic quorum-acknowledged broadcasts », *Proceedings of 27th Int'l Symposium on Fault-Tolerant Computing*, p. 272-281, 1997.

Lynch N., Shvartsman A., « RAMBO : A reconfigurable atomic memory service for dynamic networks », *Proceedings of 16th Int'l Symposium on Distributed Computing*, p. 173-190, 2002.

Maekawa M., « A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems », *ACM Trans. Comput. Syst.*, vol. 3, n° 2, p. 145-159, 1985.

Malkhi D., Reiter M., « Byzantine Quorum Systems », *Distributed Computing*, vol. 11, n° 4, p. 203-213, 2004.

Malkhi D., Reiter M., Wool A., Wright R., « Probabilistic quorum systems », *Information and Computation*, vol. 170, n° 2, p. 184-206, 2001.

Martin J.-P., Alvisi L., « A Framework for Dynamic Byzantine Storage », *Proceedings of the Int'l Conference on Dependable Systems and Networks*, IEEE Computer Society, p. 325, 2004.

Muthitacharoen A., Gilbert S., Morris R., Etna : a fault-tolerant algorithm for atomic mutable dht data, Technical Report n° MIT-LCS-TR-993, Massachussetts Institute of Technology, June, 2004.

Nadav U., Naor M., « The Dynamic And-Or Quorum System », *in* P. Fraigniaud (ed.), *Proceedings of 19th Int'l Symposium on Distributed Computing*, vol. 3724 of *LNCS*, p. 472-486, Sept., 2005.

Naor M., Wieder U., « Scalable and dynamic quorum systems », *Proceedings of the 22th Annual symposium on Principles of Distributed Computing*, ACM Press, p. 114-122, 2003.

Naor M., Wool A., « The Load, Capacity, and Availability of Quorum Systems », *SIAM Journal on Computing*, vol. 27, n° 2, p. 423-447, 1998.

Shavit N., Zemach A., « Combining funnels : a new twist on an old tale... », *Proceedings of the 17th Annual Symposium on Principles of Distributed Computing*, ACM Press, New York, NY, USA, p. 61-70, 1998.

Thomas R. H., « A Majority consensus approach to concurrency control for multiple copy databases », *ACM Trans. Database Syst.*, vol. 4, n° 2, p. 180-209, 1979.

Vidyasankar K., « Weak atomicity : A helpful notion in the construction of atomic shared variables », *Journal of Engineering Sciences of the Indian Academy of Sciences*, vol. 21, p. 245-259, 1996.

Weihl W. E., « Local Atomicity Properties : Modular Concurrency Control for Abstract Data Types », *ACM Trans. Program. Lang. Syst.*, vol. 11, n° 2, p. 249-283, 1989.

Article reçu le 1^{er} avril 2009.
Accepté après révisions le 18 février 2010.

Vincent Gramoli *effectue ses recherches à l'EPFL et à l'Université de Neuchâtel. Ses travaux portent sur la cohérence des données dans les systèmes distribués dynamiques et plus récemment sur des modèles transactionnels permettant de rendre la programmation concurrente simple et réutilisable. Il a obtenu son doctorat à l'Université Rennes 1 et a été affilié à l'IRISA, l'INRIA Saclay, University of Connecticut et Cornell University.*

SERVICE ÉDITORIAL – HERMES-LAVOISIER
14 rue de Provigny, F-94236 Cachan cedex
Tél. : 01-47-40-67-67
E-mail : revues@lavoisier.fr
Serveur web : http://www.revuesonline.com

ANNEXE POUR LE SERVICE FABRICATION

A FOURNIR PAR LES AUTEURS AVEC UN EXEMPLAIRE PAPIER
DE LEUR ARTICLE ET LE COPYRIGHT SIGNE PAR COURRIER
LE FICHIER PDF CORRESPONDANT SERA ENVOYE PAR E-MAIL

1. ARTICLE POUR LA REVUE :
   RSTI - TSI – 30/2011. Algorithmique distribuée

2. AUTEURS :
   Vincent Gramoli

3. TITRE DE L'ARTICLE :
   Émulation de mémoire partagée en environnements distribués dynamiques

4. TITRE ABRÉGÉ POUR LE HAUT DE PAGE MOINS DE 40 SIGNES :
   Objet atomique en systèmes dynamiques

5. DATE DE CETTE VERSION :
   25 décembre 2011

6. COORDONNÉES DES AUTEURS :

   – adresse postale :
   EPFL
   Station 14, CH-1015 Lausanne, Suisse

   Université de Neuchâtel
   Rue Emile-Argand 11, CH-2007 Neuchâtel, Suisse

   vincent.gramoli@epfl.ch
   – téléphone : +41 21 693 8125
   – télécopie : +41 21 693 75 70
   – e-mail : vincent.gramoli@epfl.ch

7. LOGICIEL UTILISÉ POUR LA PRÉPARATION DE CET ARTICLE :
   LaTeX, avec le fichier de style `article-hermes.cls`,
   version 1.23 du 17/11/2005.

8. FORMULAIRE DE COPYRIGHT :
   Retourner le formulaire de copyright signé par les auteurs, téléchargé sur : `http://www.revuesonline.com`

SERVICE ÉDITORIAL – HERMES-LAVOISIER
14 rue de Provigny, F-94236 Cachan cedex
Tél. : 01-47-40-67-67
E-mail : revues@lavoisier.fr
Serveur web : http://www.revuesonline.com