Mémoires partagées distribuées pour systèmes dynamiques à grande échelle

Vincent Gramoli

Mémoires partagées distribuées pour systèmes dynamiques à grande échelle

Vincent Gramoli
IRISA, INRIA
et Université de Rennes 1,
35042 Rennes, France
tel : +33 (0) 2 99 84 74 53
fax : +33 (0) 2 99 84 71 71
vgramoli@irisa.fr

***Abstract*— La plupart des systèmes distribués modernes sont à la fois à grande échelle et dynamiques. Cet article met en avant le passage des systèmes distribués classiques aux systèmes dynamiques à grande échelle. Bien que la réplication d'un *objet* (i.e., d'une donnée) à plusieurs endroits du réseau tolère les défaillances, cette technique implique un coût considérable à chaque modification de la valeur de l'objet. Ce surcoût est nécessaire pour que la valeur de l'objet reste cohérente. Dans les systèmes à grande échelle qui évoluent dynamiquement, les participants ne peuvent obtenir des informations globales. Un des défis majeurs de ces systèmes est donc de trouver un bon compromis entre la complexité en temps des opérations de lecture et d'écriture, et la complexité en messages requis pour la maintenance de la cohérence atomique. Cet article étudie ce compromis en décrivant différentes mémoires atomiques et en prenant en compte les contraintes imposées par les systèmes dynamiques à grande échelle.**

I. Introduction

A. Contexte de la cohérence des données

Répliquer des *objets* (ou données) accessibles en lecture et en écriture à plusieurs endroits du réseau pose des problèmes de cohérence. Supposons, par exemple, que la valeur v d'un objet ait été modifiée à un endroit du système où l'objet est répliqué. Dans ce cas, les changements doivent être visibles depuis tout endroit du système : en particulier si un client lit la valeur, alors cette opération de lecture doit renvoyer la valeur v ou une valeur plus à jour. Ce scénario simple résume l'idée prépondérante de la cohérence atomique [12], [14], aussi appelée linéarisabilité [11].

Lamport a défini les registres atomiques comme une abstraction de base pour l'implémentation de mémoires partagées. Herlihy et Wing ont proposé la notion de linéarisabilité applicable à des objets supportant différentes spécification (first-in first-out, last-in first-out), Lynch définit l'atomicité en termes de propriétés de trace d'éxecution d'un automate. Ici nous nous intéressons aux efforts menés afin d'implémenter une mémoire atomique assurant une faible latence des lectures et écritures et guarantissant la cohérence à moindre coût.

B. Le quorum comme brique de base

Depuis la fin des années soixante-dix [7], [20], les *quorums*, ensembles s'intersectant mutuellement, ont été utilisés pour assurer la disponibilité des données dans les systèmes distribués. La représentation la plus simple d'un quorum est la majorité. Soit un ensemble de n éléments. Une majorité est un sous-ensemble composé d'au moins $\lceil n/2 \rceil$ éléments. Il existe au moins un élément commun dans tout couple de majorités, l'intersection mutuelle entre de tels quorums est donc assurée.

Les quorums sont fondamentaux dans l'élaboration des mémoires partagées distribuées (MPD) car ils représentent les ensembles de nœuds où un objet (et sa valeur) doit être répliqué. En effet, lire et écrire un objet consiste simplement à accéder un nombre constant de quorums (au moins un). La propriété d'intersection des quorums est utilisée pour assurer qu'au moins un nœud, interrogé lors d'une lecture, peut témoigner de la valeur la plus à jour de l'objet. Par conséquent, la cohérence atomique dans les systèmes distribués est étroitement liée à l'idée d'intersection ensembliste proposée par les quorums. Et, diminuer la taille des quorums permet de diminuer la latence des opérations.

Cet article tente de décrire l'évolution des récentes techniques de mémoires partagées distribuées. Pour cela il présente leur point commun par un algorithme générique et identifie leur caractéristiques pour enfin proposer l'idée de base d'une approche probabiliste qui semble prometteuse.

C. Contributions

Cet article présente, dans un premier temps, un algorithme simple et générique qui implémente un objet atomique. Nous spécifions les opérations de lecture et d'écriture atomiques utilisant l'accès aux quorums. Cet algorithme met en valeur l'importance de la taille des quorums pour la complexité des opérations. Nous prouvons que cet algorithme implémente un objet atomique sans donner le détail sur la façon dont sont accédés et maintenus les quorums. Nous nous intéressons uniquement à la sûreté et non à la vivacité de cet algorithme.

Dans un second temps, nous nous focalisons sur les deux problèmes mentionnés précédemment causés par l'aspect grande échelle et dynamique du contexte. D'une part, les opérations doivent être exécutées en un minimum de temps malgré le grand nombre de serveurs à contacter. D'autre part, la cohérence doit être assurée en dépit des limitations de bande passante imposées par les systèmes actuels.

Finalement nous décrivons plusieurs moyens existant pour accéder et maintenir les quorums. Nous étudions les mécanismes de reconfiguration permettant d'assurer la cohérence en dépit du dynamisme des nœuds. Nous présentons l'adaptativité comme une solution à l'accès des quorums en nous appuyant sur plusieurs travaux récents. Enfin nous décrivons les techniques probabilistes les mieux adaptées à ce contexte et expliquons pourquoi elles constituent des pistes de recherche intéressantes.

D. Plan

La Section II définit le modèle de l'étude. La Section III propose un algorithme générique de mémoire atomique utilisant un système de quorums. La Section IV propose des solutions face au dynamisme. La Section V présente des solutions au problème de passage à l'échelle. Enfin, la Section VI conclut l'article.

II. Modèle et définitions

Cette Section présente le modèle et définit certains des termes utilisés dans la suite de l'article. Chacune des solutions de cet article utilise le modèle du système présenté dans cette Section.

A. Système distribué dynamique

Le système est composé d'un ensemble de n nœuds, possédant chacun un identifiant unique. Il existe un ensemble d'*objets* (i.e., données) dans le système et chacun possède une valeur initiale v_0. Chaque objet x est répliqué sur des nœuds, appelés les *serveurs* de l'objet x. Tout nœud du système, qu'il soit serveur ou non, peut accéder un objet via des opérations de lecture et d'écriture. On parlera alors de ce nœud comme d'un *client*. Un nœud pourra donc être à la fois client et serveur.

Le système est dynamique, les nœuds peuvent quitter et rejoindre le système à loisir. Néanmoins, lorsqu'un nœud rejoint le système, il est considéré comme nouveau et possède un nouvel identifiant. Par ailleurs, les défaillances qui peuvent survenir dans le système sont des pannes franches. lorsqu'un nœud tombe en panne il n'effectue plus d'action.

B. Quorums et ensemble de quorums

Un *quorum* Q de l'objet x est constitué de q serveurs de x. Un quorum est dit *actif* si l'ensemble des nœuds qui le composent n'ont pas quitté le système et ne sont pas tombés en panne. Le *système de quorums* $\mathcal{Q}$ de l'objet x est l'ensemble des quorums de x. Tous les quorums d'un système de quorums s'intersectent deux-à-deux.

C. Communication entre les nœuds

Nous supposons que la communication entre les nœuds est asynchrone et que les canaux sont fiables. Les messages mettent donc un temps arbitraire mais arrivent finalement à destination et aucun message n'est généré ou dupliqué par les canaux de communication.

D. Atomicité

La définition d'atomicité est tirée du Théorème 13.16 proposé par Lynch dans [14].[1]

Définition 1: Soit un objet x 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 vérifie l'atomicité si 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 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).

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

E. Remarques préliminaires

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.

L'atomicité est une propriété qui supporte la *composition* : la composition de deux objets atomiques est également un objet atomique. 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.

III. Opérations atomiques utilisant l'accès au quorum

Cette Section explique simplement comment utiliser les quorums comme briques de base pour une mémoire atomique distribuée.

A. Spécification de l'algorithme générique

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 les procédures **Propage** et **Consulte**. Ou bien, sans connaître l'ensemble du quorum, le client peut attendre le message du dernier élément contacté pour décider de cet arrêt.

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

TABLE I
Domaine de l'algorithme

L'Algorithme 1 (présenté Pages 3 et 4) implémente un objet atomique supportant des lecteurs multiples et écrivains multiples. Le domaine des variables utilisé dans l'algorithme est présenté dans la Table I. Le pseudocode est volontairement haut niveau et présente une solution générique utilisant des accès aux quorums. Cet algorithme s'inspire à la fois des travaux de Attiya, Bar-Noy et Dolev [3] ainsi que de ceux de Lynch et Shvartsman [15].

Algorithm 1 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

17: **Lire()$_i$:**
18: $\langle v, t \rangle \leftarrow$ **Consulte**()$_i$
19: **Propage**($\langle v, t \rangle$)$_i$
20: **Return** $\langle v, t \rangle$

21: **Écrire(**$vnew$**)$_i$:**
22: $\langle v, t \rangle \leftarrow$ **Consulte**()$_i$
23: $tnew \leftarrow \langle t.compteur + 1, i \rangle$
24: **Propage**($\langle vnew, tnew \rangle$)$_i$

Un client exécute une opération en invoquant la procédure **Lire** ou **Écrire**. À chaque serveur de l'objet est maintenue une valeur v et un tag t associé indiquant le numéro de version de la valeur. 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 (**Écrire**) 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. La seconde phase **Propage** la valeur $vnew$ et le tag $tnew$ à jour de l'objet au sein d'un quorum. 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 comp-

25: **Consulte()$_i$:**
26: Soit Q un quorum de $\mathcal{Q}$
27: $s \leftarrow s + 1$
28: send(GET, $\perp, s$) aux nœuds de Q
29: **Repeat:**
30: **if** recv(ACK, $\langle v, t\rangle, s$) de j **then**
31: $repondant \leftarrow repondant \cup \{j\}$
32: **if** $t > tmax$ **then**
33: $tmax \leftarrow t$
34: $vlast \leftarrow v$
35: **Until** is-quorum-contacted($repondant$)
36: **Return** $\langle vlast, tmax\rangle$

37: **Propage($\langle v, t\rangle$)$_i$:**
38: Soit Q un quorum de $\mathcal{Q}$
39: $s \leftarrow s + 1$
40: send(SET, $\langle v, t\rangle, s$) aux nœuds de Q
41: **Repeat:**
42: **if** recv(ACK, $\perp, s$) de j **then**
43: $repondant \leftarrow repondant \cup \{j\}$
44: **Until** is-quorum-contacted($repondant$)

45: **Participe()$_i$ (activer à la réception de $\mathcal{M}$ par j):**
46: **if** recv($\mathcal{M}$) du nœud j **then**
47: **if** $\mathcal{M}.type =$ GET **then**
48: send(ACK, $\langle v, t\rangle, \mathcal{M}.seq$) à j
49: **if** $\mathcal{M}.type =$ SET **then**
50: **if** $\mathcal{M}.tag > t$ **then**
51: $\langle v, t\rangle = \mathcal{M}.\langle val, tag\rangle$
52: send(ACK, $\perp, \mathcal{M}.seq$) à j

teur de phase pour pallier l'asynchronie et éviter qu'un nœud 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.

B. Preuve de cohérence atomique

Le Théorème suivant prouve que l'Algorithme 1 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 1. 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 1 implémente un objet atomique.

Preuve. La preuve montre successivement que l'ordre des tags sur les opérations vérifie les trois propriétés de la Définition 1. Soit $\prec$ l'ordre partiel sur les opérations définis par : $op_1 \prec op_2$ si et seulement 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 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 diffèrent 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 soient $op_1, ..., op_k$, 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 1 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 concourament écrite est retournée [6], [21] ou bien lorsqu'il est clair que la valeur à jour a déjà été propagée à tout un quorum lorsque la lecture consulte [5], [8].[2] De telles approches présentent des cas particuliers que ne spécifie pas l'Algorithme 1 pour des raisons de simplicité dans la présentation.

IV. La tolérance au dynamisme

Le dynamisme d'un systéme s'exprime par un changement continue de son état. Ces changements posent des problèmes pour assurer la cohérence de l'information. Pour rémédier à un nombre de changements bornés il suffit de répliquer l'information davantage. Cependant d'autres mécanismes, prenant en compte les contraintes physiques, doivent être mis en œuvre pour pallier à des changements qui s'accumulent.

A. Réplication et tolérance aux fautes

Certains travaux de recherche ont pour objet l'émulation de mémoire partagée dans les modèles à passage de message, aussi appelée Mémoire Partagée Distribuée (MPD). Par exemple, Attiya, Bar-Noy et Dolev [3] proposent un algorithme robuste de mémoire avec des lecteurs multiples et un écrivain unique. Cet algorithme spécifie une lecture en deux phases et une écriture en une phase. Chacune de ces phases consiste en un échange de messages entre le client et une majorité de nœuds. Plus récemment, des opérations de lecture ne nécessitant qu'une seule phase ont été proposées dans [5], [6], [8], [21].

B. Reconfiguration et tolérance au dynamisme

La robustesse dans les systèmes dynamiques est plus difficile à obtenir. Alors qu'utiliser une majorité de nœuds permet d'obtenir un algorithme robuste lorsque le nombre de fautes est borné, dans les systèmes dynamiques, le nombre de défaillances peut croître indéfiniment. En conséquence, un mécanisme de *reconfiguration* a été introduit dans RAMBO [15] afin d'assurer la robustesse lorsque les défaillances s'accumulent. Le processus de reconfiguration remplace régulièrement l'ensemble des serveurs par des nouveaux nœuds actifs.

La reconfiguration remplace le système de quorums par un nouveau système de quorums. Ce remplacement de système de quorums, s'effectue de façon transparente pour l'utilisateur, aucune opération n'est stoppée durant la reconfiguration. La réplication de quorums est suffisante pour pallier aux défaillances qui se produisent entre les reconfigurations consécutives.

Bien sûr, les reconfigurations doivent être suffisamment fréquentes pour que le nombre de défaillances s'accumulant restent sous le seuil critique toléré (typiquement, au moins un quorum doit rester acif). 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 du *Reconfigurable Distributed Storage* (RDS) [5].

RDS [5] propose un mécanisme de reconfiguration rapide. Le protocole de reconfiguration qui y est présenté utilise une version optimisée de l'algorithme Paxos [4], [13], 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 quorums 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 messages.

La reconfiguration rapide est utile pour la tolérance aux défaillances dans les systèmes distributés dynamiques. Plus précisément, 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 se stabiliser durant suffisamment longtemps.

C. Reduire les bavardages

Les solutions proposées ci-dessus possèdent un inconvénient lorsqu'elles sont deployées à grande échelle. En

[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 (cf. [3]).

effet, dans un système tel que l'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 sus-citées, les nœuds connaissent le système de quorums 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 quorums. 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 noeuds soient au courant du nouveau système de quorums est $O(n^2)$.

Dans [10], les auteurs proposent une amélioration significative dans le but de réduire la complexité de 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* si il se considère localement comme faisant partie d'un quorum à jour—un quorum du dernier système de quorums installé.

Lorsqu'une reconfiguration a lieu, les informations relatives au nouveau système de quorums 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 surcroit, la complexité de 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 sur un des propriétaires du système de quorums actuel bien que ces systèmes se succèdent au fil du temps. L'hypothèse utilisée est qu'au moins un propriétaire de chaque système de quorums 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 quorums 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 1 où $q = 6$. 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.

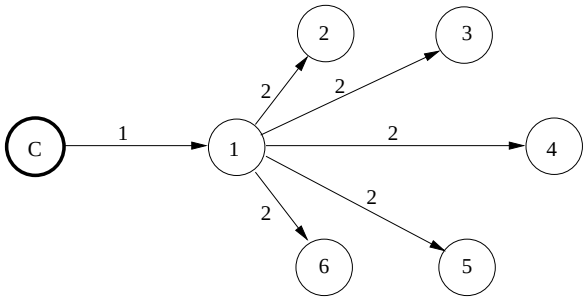

Fig. 1. L'approche par connaissance globale : le client C effectue chaque opération de lecture et écriture en temps $O(1)$. Tous les serveurs (ou presque) du quorum sont contactés en parallèle.

V. Passer à l'échelle de l'Internet

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 principe de *localité*. La localité est la qualité d'un algorithme à tirer parti de l'information située dans un voisinage proche plutôt que de 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 évévements dynamiques : la réparation d'une défaillance, bien qu'en moyenne plus fréquente que la reconfiguration, a un coût plus faible.

A. Diminuer la connaissance pour passer à grande échelle

Chaque nœud maintient, désormais, une information restreinte sur le système, le mécanisme de réparation est exécuté localement, et donc, avec une complexité plus faible que dans le cas d'une reconfiguration. Supposons, par exemple, qu'un des serveurs quitte le système de quorums, il sera seulement nécessaire à ses voisins de réparer cette panne, soit en agissant à sa place, soit en répliquant la donnée à un autre nœud pour qu'il devienne 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 au simple voisinage 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 $O(q)$.

B. 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 découvert 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 [19].

Récemment, certains auteurs se sont intéressés aux quorums dynamiques [1], [18], [19], dont les serveurs changent progressivement avec le temps. Dans [1], 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. D'autre part, le Dynamic Path [19] 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 acollées sont voisins. Les cellules se réadaptent automatiquement en fonction du placement des nouveaux nœuds ou des nœuds qui partent.

Le système de quorums dynamiques « et/ou » [18] 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 mémoires distribuées partagées utilisent ces nouveaux types de quorums pour leur adaptation aux systèmes dynamiques à grande échelle, non seulement pour leur dynamisme inhérent mais également pour leur propriété de localité. Par exemple, SAM [2] et SQUARE [8] utilisent des spécificités dynamiques afin d'adapter la structure des systèmes de quorums aux variations de charges imprévisibles en contexte grande échelle. De plus, ces solutions utilisent une grille supportant des départs et des 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 2 représente un accès au quorum : le client C contacte tour-à-tour chacun des serveurs d'un quorum.

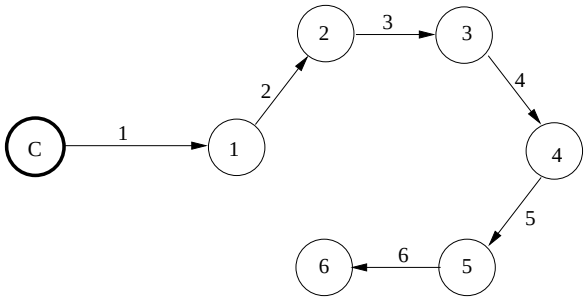

Fig. 2. L'approche adaptative : le client C accède à un quorum en contactant chaque serveur du quorum tour-à-tour. Les opérations sont executées en temps $O(q)$.

Certaines méthodes adaptatives étendent la notion de localité à un nombre de nœuds logarithmique en la taille du système. Ce type d'approche [17] permet d'effectuer des opérations en temps $O(\log q)$.

C. Mémoire atomique probabiliste

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, dans [5] la reconfiguration termine si le système se stabilise et que les temps des messages sont bornés. Aussi, [2] nécessite la présence de détecteurs de fautes parfait.

Récemment et face aux difficultés rencontrées pour assurer des propriétés déterministes, des solutions probabilistes ont vu le jour.

Dans [1], Abraham et al. 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 forte probabilité. Ce résultat est atteint en exécutant $O(\sqrt{n})$ marches aléatoires, de longueur $O(\log n)$ chacune, effectuées en parallèle. Ainsi le temps nécessaire pour accéder un quorum est de $O(\log n)$ délais de message (avec $q = \sqrt{n}\log n$). La Figure 3 présente simplement l'accès à un quorum en utilisant deux marches aléatoires parallélisées.

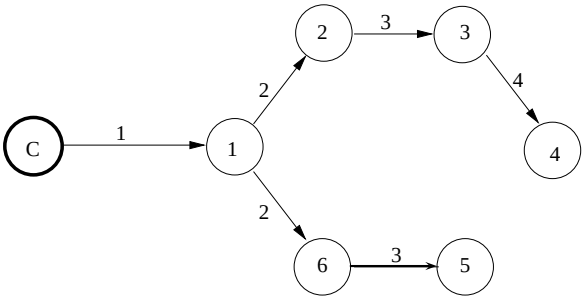

Fig. 3. L'approche par marches aléatoires : deux marches aléatoires sont exécutées en parallèle afin de contacter l'ensemble des serveurs du quorum. Le temps d'accès est donc linéaire en la taille des marches aléatoires.

Malkhi et al. définissent un système de quorums probabilistes [16]. Appelés systèmes de quorums ϵ-*intersectant*, ils contiennent 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.

Bien que la taille de ces quorums ne peut pas être considérablement diminuée, contrairement au méthodes précédentes, les quorums probabilistes apportent des améliorations en terme 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 ?

D. Solutions sans structure

Toutes les solutions précédemment décrites ont un point commun : elles utilisent une couche de communication 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 strucure.

Par conséquent, diminuer les coûts de communication qu'impliquent le maintien de la cohérence nécessite de définir une mémoire atomique sans structure. Dans [9], Gramoli et al. évoquent une façon de préserver une donnée en dépit du dynamisme sans utiliser de structure. Leur approche est aussi bien généralisable à la cohérence des données. L'idée fondamentale de cet article 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.

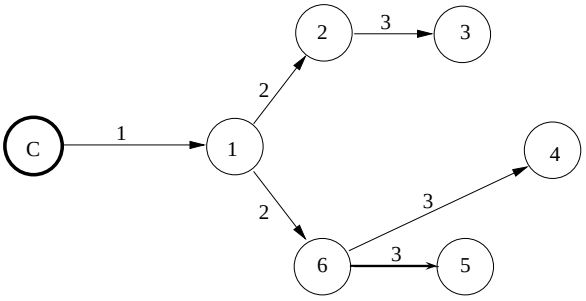

Fig. 4. L'approche par dissémination : des messages sont disséminés au sein des nœuds afin de contacter les serveurs du quorum. Le temps d'accès est donc au plus logarithmique en le nombre de nœuds à contacter.

L'idée maîtresse est temporelle dans le sens où un objet subsiste si les opérations sont suffisamment fréquentes par rapport au dynamisme du système—le taux auquel les nœuds rentrent dans le système et quittent le système. En d'autres termes, si un objet est écrit (mis-à-jour ou modifié) suffisamment souvent sur un nombre suffisant de serveurs alors l'objet subsiste. Cette technique peut être utilisée afin de définir des quorums ne nécessitant aucun mécanisme de réparation. La Figure 4 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 utilisant un bavardage régulier pour obtenir des résultats similaires en diminuant les effets de congestion possibles.

VI. CONCLUSION

A. Résumé

Cet article classifie un éventail de résulats obtenus dans le contexte des solutions de mémoire partagée distribuée. L'article met en évidence les deux défis majeurs rencontrés dans les systèmes dynamiques à grande

Méthodes	Inconvénients	Avantages
Structurée reconfigurable (RAMBO [15], RDS [5])	Messages	Temps
Structurée adaptative (SAM [2], SQUARE [8])	Temps	Messages
Structurée probabiliste (Abraham et al. [1])	Guarantie probabiliste	Temps
Non structurée (Gramoli et al. [9])	Guarantie probabiliste	Temps et messages

TABLE II

INCONVENIENTS ET AVANTAGES DES DIFFÉRENTES MÉTHODES.

échelle : la complexité en temps des opérations ainsi que la complexité de communication résultant du maintien de la cohérence des données. Les solutions décrites témoignent de l'évolution des techniques durant les vingt dernières années et présentent les solutions probabilistes comme des techniques appropriées pour les systèmes dynamiques à large échelle.

B. Futures recherches

Un premier problème intéressant serait celui de réduire la taille des quorums probabilistes. Un second serait de modéliser le dynamisme du système de façon probabiliste afin de comparer les techniques existantes pour le maintien de la cohérence des données. Une solution pourrait être simplement l'étude de l'évolution du nombre de données à jour en fonction du dynamisme du système et du type d'algorithme utilisé.

REFERENCES

[1] I. Abraham and D. Malkhi. Probabilistic quorums for dynamic systems. *Distributed Computing*, 18(2) :113–124, 2005.

[2] E. Anceaume, M. Gradinariu, V. Gramoli, and A. Virgillito. P2P architecture for self* atomic memory. In *Proc. of 8th IEEE International Symposium on Parallel Architectures, Algorithms and Networks (I-SPAN'05)*, pages 214–219, Dec. 2005.

[3] H. Attiya, A. Bar-Noy, and D. Dolev. Sharing memory robustly in message-passing systems. *J. ACM*, 42(1) :124–142, 1995.

[4] R. Boichat, P. Dutta, S. Frølund, and R. Guerraoui. Reconstructing paxos. *SIGACT News*, 34(2) :42–57, 2003.

[5] G. Chockler, S. Gilbert, V. Gramoli, P. Musial, and A. Shvartsman. Reconfigurable distributed storage for dynamic networks. In *Proc. of 9th International Conference on Principles of Distributed Systems (OPODIS'05)*, volume 3974 of *LNCS*, pages 351–365. Springer, Dec. 2005.

[6] P. Dutta, R. Guerraoui, R. R. Levy, and A. Chakraborty. How fast can a distributed atomic read be ? In *Proc. of the 23th annual symposium on Principles of distributed computing (PODC'04)*, pages 236–245, New York, NY, USA, 2004. ACM Press.

[7] D. K. Gifford. Weighted voting for replicated data. In *Proceedings of the seventh ACM Symposium on Operating systems principles (SOSP'79)*, pages 150–162. ACM Press, 1979.

[8] V. Gramoli, E. Anceaume, and A. Virgillito. SQUARE : Scalable quorum-based atomic memory with local reconfiguration. In *Proc. of the 22th Annual ACM Symposium on Applied Computing (SAC'07)*. ACM Press, 2007. to appear.

[9] V. Gramoli, A-M. Kermarrec, A. Mostefaoui, M. Raynal, and B. Sericola. Core persistence in peer-to-peer systems : Relating size to lifetime. In *Proceedings of the International OTM Workshop on Reliability in Decentralized Distributed systems (OTM'06)*, volume 4218 of *LNCS*, pages 1470–1479. Springer, Oct. 2006.

[10] V. Gramoli, P. Musial, and A. Shvartsman. Operation liveness and gossip management in a dynamic distributed atomic data service. In *Proc. of the 18th International Conference on Parallel and Distributed Computing Systems (PDCS'05)*, Sept. 2005.

[11] M. P. Herlihy and J. M. Wing. Linearizability : a correctness condition for concurrent objects. *ACM Trans. on Programming Languages and Systems (TOPLAS)*, 12(3) :463–492, 1990.

[12] L. Lamport. On interprocess communication, part II : Algorithms. *Distributed Computing*, 1 :86–101, 1986.

[13] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2) :133–169, 1998.

[14] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.

[15] N. Lynch and A. Shvartsman. RAMBO : A reconfigurable atomic memory service for dynamic networks. In *Proc. of 16th International Symposium on Distributed Computing (DISC'02)*, pages 173–190, 2002.

[16] D. Malkhi, M. Reiter, A. Wool, and R. Wright. Probabilistic quorum systems. *Information and Computation*, 170(2) :184–206, 2001.

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

[18] U. Nadav and M. Naor. The dynamic and-or quorum system. In Pierre Fraigniaud, editor, *Proc. of 19th International Symposium on Distributed Computing (DISC'05)*, volume 3724 of *LNCS*, pages 472–486, September 2005.

[19] M. Naor and U. Wieder. Scalable and dynamic quorum systems. In *Proc. of the 22th annual symposium on Principles of distributed computing (PODC'03)*, pages 114–122. ACM Press, 2003.

[20] R. H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *ACM Trans. Database Syst.*, 4(2) :180–209, 1979.

[21] M. Vukolic and R. Guerraoui. How fast can a very robust read be ? In *Proceedings of the 25th annual symposium on Principles of distributed computing (PODC'06)*, 2006.