INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE | INRIA

UNIVERSITÉ DE RENNES 1

INRIA Centre de recherche de Rennes **Université de Rennes 1**
INRIA Futurs Saclay

Mémoire partagée distribuée pour systèmes dynamiques à grande échelle

Thèse présentée devant

l'Université de Rennes 1

pour l'obtention du grade de

Docteur

en

Informatique

par

Vincent Gramoli

soutenue le 22 novembre 2007 devant le jury composé de :

Antonio Fernández	Professeur	Examinateur
Roy Friedman	Professeur	Rapporteur
Anne-Marie Kermarrec	Directrice de recherche	Examinatrice
Michel Raynal	Professeur	Directeur de thèse
Pierre Sens	Professeur	Président du jury
Alex Shvartsman	Professeur	Rapporteur

Year 2007 École Doctorale Matisse
Numéro d'Ordre : 3637

Cette thèse étudie les nouveaux défis du contexte du partage de donnée liés au récent changement d'échelle des systèmes répartis. De nos jours les systèmes répartis deviennent très grands. Cet accroissement concerne non seulement le nombre de personnes qui communiquent dans le monde via leur ordinateur mais aussi le nombre d'entités informatiques personnelles connectées. De tels grands systèmes sont sujets au dynamisme du fait du comportement imprévisible de leurs entités. Ce dynamisme empêche l'adoption de solutions classiques et plus simplement affecte la communication entre des entités distinctes. Cette thèse étudie les solutions existantes et propose des suggestions de recherche pour la résolution du problème fondamental de mémoire partagée distribuée dans un tel contexte dynamique et à grande échelle.

Mots-clefs : Mémoire partagée distribuée, quorum, atomicité, cohérence, reconfiguration, dynamisme, passage à l'échelle.

Table des matières

1 Prérequis **4**
- 1.1 Les quorums . . . 4
- 1.2 Le modèle général . . . 5
- 1.3 Cohérence mémoire . . . 6
  - 1.3.1 Opérations atomiques utilisant l'accès aux quorums . . . 7
  - 1.3.2 Preuve de cohérence atomique . . . 11

2 Tolérer le dynamisme **12**
- 2.1 Introduction . . . 12
- 2.2 La reconfiguration comme le remplacement de quorums . . . 13
- 2.3 Indépendance des opérations . . . 13
- 2.4 La reconfiguration décentralisée . . . 14
- 2.5 Installation et suppression de configurations. . . . 14
- 2.6 Ordonner les configurations en utilisant le consensus . . . 15
- 2.7 Mémoire partagée distribuée dynamique . . . 17

3 Tolérer le passage à l'échelle **18**
- 3.1 Introduction . . . 19
- 3.2 Diminuer la connaissance pour passer à grande échelle . . . 20
- 3.3 Mémoire atomique adaptative . . . 20
- 3.4 Mémoire partagée distribuée passant à l'échelle . . . 21
- 3.5 Opérations atomiques . . . 24
- 3.6 Routage . . . 24
  - 3.6.1 Quorums dynamiques . . . 25
  - 3.6.2 Phase de consultation . . . 26
  - 3.6.3 Phase de propagation . . . 29
- 3.7 Tolérance aux défaillances . . . 29
  - 3.7.1 Réplication des données . . . 29
  - 3.7.2 Détection de défaillances . . . 30
  - 3.7.3 Arrivée d'un nœud . . . 31
  - 3.7.4 Départ d'un nœud . . . 31
- 3.8 Répartition de la charge . . . 32
  - 3.8.1 Charge et Complexité . . . 32
  - 3.8.2 Diagonalisation . . . 33

3.8.3 Auto-expansion 33
3.8.4 Auto-réduction 35
3.9 Conclusion 35

4 Tolérer le dynamisme et le passage à l'échelle **36**
4.1 Introduction 36
4.1.1 Contexte 36
4.1.2 Contributions 37
4.2 Modèle 37
4.3 Relation entre les paramètres clefs du système dynamique 38
4.3.1 Relation entre la taille du noyau q et la probabilité ϵ. . . . 40
4.3.2 Relation entre la taille du noyau q et la période δ 41
4.4 Conclusion 42

5 Conclusion **43**

Introduction

Passage à l'échelle

Les systèmes distribués sont devenus un domaine important de l'informatique. L'intérêt de recherche que suscite un tel domaine n'a cessé d'augmenter durant ces 30 dernières années et maintenant les systèmes distribués ont considérablement changé amenant de nouveaux problèmes. Nous sommes persuadés que la principale cause de ces problèmes est le changement d'échelle récent qu'ont connus les systèmes distribués.

Le besoin de resources a toujours été une motivation importante de la recherche en systèmes distribués. Les capacités de calculs en sont un bon exemple. Une machine multi-processeurs est capable de partager les tâches de calcul sur différents processeurs et des machines distribuées peuvent exécuter de multiples tâches en même temps, une s'exécutant sur chaque machine. A l'heure actuelle, les machines multi-processeurs avec de fortes capacités sont toujours très chères alors qu'une nouvelle forme de collaboration émerge grace à l'Internet et à l'essor de divers réseaux. Cette colaboration atteint des sommets encore inégalés en terme de capacité de calcul. Les premières observations de ce phénomène ont vu le jour avec SETI@home qui, lancé en 1999, fournit actuellement 264 TeraFlops grace à la participation de millions de machines. Ce besoin de resources est un catalyseur de l'expansion des systèmes distribués.

Finalement, le monde d'aujourd'hui est peuplé d'entités électroniques capables de communiquer. Premièrement, le protocole de l'Internet IPv4 a été revu et corrigé en IPv6 pour permettre son utilisation à un plus grand nombre de machines. Deuxièmement, d'autres formes de périphériques communiquant se développent et chaque personne à tendance à utiliser plusieurs de ces périphériques. Finalement, IDC prévoit qu'en 2012, il y aura 17 milliards d'entités électroniques communiquantes dans le monde. De fait, il est naturel de considérer le passage à l'échelle comme un des défis les plus importants des systèmes distribués actuels.

Dynamisme

L'accroissement de la taille des systèmes distribués modifie la manière dont les entités communiquent entre elles. Dans ce contexte, connecter des entités variées

entre elles amènent de nouvelles problématiques liées au comportement indépendant que chacune possède. Ainsi, la dépendance énergétique, la deconnexion volontaire, les malfonctionnements, et autres conséquences de facteurs environnementaux affectent la disponibilité de chaque entité indépendamment. Par conséquent, chaque entité est capable de recevoir des messages ou d'executer des calculs seulement par intermittence. Une telle intermittence rend les systèmes dynamiques. De plus, dans de tels grands systèmes, chaque entité est incapable de rassembler des informations sur le système dans son ensemble, il est donc impossible de prédire ou même de mesurer le dynamisme de tels systèmes. Ce dynamisme rend davantage difficile la communication des entités dans les grands systèmes.

Il existe deux principaux modes de communications dans les systèmes distribués : celui à *mémoire partagée* et celui à *passage de message*. Avec un mémoire partagée, les entités communiquent en lisant et en écrivant sur une seule mémoire. Par passage de message, les entités communiquent en envoyant et recevant des messages dont la transmission est, en général, arbitrairement longue. L'avantage du mode de communication à mémoire partagée est qu'il est plus simple de décrire un algorithme ou d'écrire un programme en utilisant ce mode de communication. L'avantage du mode de communication par passage de message est qu'il est adapté à la robustesse et permet de décrire un systèmes tolérant aux pannes du fait de sa réplication. Ce dernier mode est plus adapté lorsque la communication entre machines est lente comparée au temps de calcul d'une machine. Par conséquent, il est intéressant dans notre contexte de pouvoir émuler une mémoire partagée avec un mode de communication par passage de message. Cette émulation, décrite par l'abstraction de mémoire partagée distribuée, est le sujet de cette thèse.

L'objectif d'une mémoire partagée distribuée (MPD) est d'émuler le fonctionnement d'une mémoire partagée en utilisant un mode de communication à passage de message. Du point de vue du client, la MPD doit fournir des primitives de lecture et d'écriture qui agissent comme si la mémoire était partagée. La difficulté majeure réside dans le fait que les opérations exécutées par des clients distants doivent être ordonnées : un client lit la valeur écrite par un autre client en fonction du temps auquel ont eu lieu les opérations. De façon idéale, l'ordre doit respecter l'ordre de temps réel, cependant il est difficile de synchroniser les horloges des utilisateurs et les opérations peuvent s'exécuter en parallèle. Il est donc nécessaire d'assurer que les opérations respectent un ensemble de règles, appelé critère de cohérence mémoire. Durant les dernières dizaines d'années, les MPD pour les

systèmes statiques avec pannes ont été étudiés. Plus recemment, les MPD pour les systèmes dynamiques ont suscités un intérêt particulier. Maintenant, nous sommes convaincus que les MPD pour grands systèmes dynamiques est d'un intérêt crucial pour la communication dans les systèmes distribués.

Contributions

Cette thèse est divisée selon trois axes principaux, représentés par trois Chapitres principaux.

Le Chapitre 3 s'intéresse essentiellement au problème du dynamisme dans les Mémoires Partagées Distribuées (MPD). Il présente la reconfiguration comme un méchanisme permettant de tolérer le dynamisme du système. Cette reconfiguration assure qu'en dépit des départs et arrivées fréquents, le système reste utilisable et les opérations vérifient à tout moment n'importe quel critère de cohérence.

Le Chapitre 4 se focalise sur les problèmes liés au passage à l'échelle. Il propose d'améliorer le méchanisme de reconfiguration pour qu'il passe à l'échelle en étant efficace. Pour cela, ce Chapitre s'intéresse à la structure des ensembles de serveurs sous-jacents, appelés quorums, et à la façon dont les entités du système communiquent avec ces serveurs.

Le Chapitre 5 s'intéresse à résoudre les problèmes listés précédemment pour tolérer à la fois le passage à l'échelle et le dynamisme. Il essaie à la fois de proposer une solution efficace en la latence des opérations, tolérant le dynamisme et tolérant le passage à l'échelle. Cette solution affaiblit la cohérence mémoire désirée afin de fournir des opérations exactes (vérifiant n'importe quel critère de cohérence) avec forte probabilité.

Plan

La Section 1 donne quelques prérequis nécessaires à la compréhension du reste du document. La Section 2 résume les solutions apportées dans le Chapitre 3 de la thèse. Elle présente rapidement la reconfiguration comme un méchanisme permettant de tolérer le dynamisme du système. La Section 3 donne une version simplifiée de la MPD présentée dans le Chapitre 4 de la thèse. Cette solution permet d'améliorer le méchanisme de reconfiguration pour qu'il passe à l'échelle en étant efficace. La Section 4 s'intéresse au problème de persistance des données

qui sert de brique de base au Chapitre 5 de la thèse. Cette solution essaie à la fois de proposer une solution efficace en la latence d'accès à un noyau (i.e., quorum), tolérant le dynamisme et tolérant le passage à l'échelle. Finalement la Section 5 conclue ce document.

1 Prérequis

1.1 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 méchanisme de synchronisation doit être mis en place pour assurer la cohérence en dépit de la concurrence. Un méchanisme permettant d'informer un nœud sur le fait que son opération peut altérer la cohérence a été défini par Gifford [Gif79] et Thomas [Tho79] la même année 1979. Dans les méchanismes 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 gens 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 possède un poids dans 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 (reciproquement w), tels que $r + w > W$. Thomas suppose une base de données distribuée où la copie d'une donnée est répliquée à plusieurs endroits distincts. Plusieurs nœuds peuvent exécuter une opération sur cette base de donnée en envoyant un message à une copie de la base de donnée, cette copie essaie de récolter la permission d'une majorité des 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 distribués. En effet, Garcia-Molina et Barbara [GMB85] 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 quorums s'intersectant mutuellement est

appelé un *système de quorum*.

Definition 1.1 (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.*

1.2 Le modèle général

Le système consiste en un ensemble de nœuds interconnectés entre eux. Dans ce document, nous supposons qu'un nœud n'a besoin que de connaitre l'identifiant d'un autre noeud afin de communiquer avec lui et chaque nœud i connait un sous-ensemble de nœuds avec lesquels il peut communiquer, ces nœuds sont appelés les voisins du nœud i. Tout les nœuds possède un unique identifiant et l'ensemble des identifiants est noté I. La communication est asynchrone, ainsi les messages peuvent être arbitrairement longs. La communication n'est pas fiable, c'est-à-dire les messages peuvent être réordonnés et perdus, cependant, si un message arrive à destination, alors il a été envoyé et pas alteré ; aucun message n'est dupliqué. Le système est dynamique parce 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 considéré comme un départ et une réparation est considérée comme une nouvelle arrivée. En d'autres termes, lorsqu'un nœud re-rentre 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 initié 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 : si le tag t

est plus petit que t' cela signfie que soit son compteur est plus petit que celui de t' soit ils sont identiques mais l'identifiant de t est plus petit que celui de t'.

1.3 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 guaranties désirées. De nombreux critères de cohérence mémoires variés 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 que vérifie 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.

En dépit du fait que les opérations respectant l'atomicité aient un résultat similaire à celles appliquées à une seule mémoire partagée, l'atomicité est difficile à implémenter du fait de sont niveau de tolérance. Par conséquent des critères de cohérence plus tolérants mais plus faciles à mettre en œuvre ont été proposés : l'atomicité faible [Vid96], la causalité [Lam78, HA90], la cohérence PRAM [LS88], la cohérence de processeurs [Goo89]... L'inconvénient de tels critères est d'autoriser plusieurs nœuds distincts à avoir une vue différentes 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 [ACLS94] et la cohérence hybride [AF92], utilisant un ordre tolérant [DSB86].

La caractéristique de *localité* a été définie par Herlihy et Wing 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 a été prouvée comme étant locale, contrairement à la séquentialité [HW90]. Ici, nous considérons le critère local de cohérence atomique.

La définition d'atomicité est tirée du Théorème 13.16 proposé dans [Lyn96].[1]

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

Definition 1.2 *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).*

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* [HW90, Lyn96] : 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.

1.3.1 Opérations atomiques utilisant l'accès aux quorums

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

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

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

TAB. 1 – Domaine de l'algorithme

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

du quorum, le client peut attendre le message du dernier élément contacté pour décider de cet arrêt.

L'Algorithme 1 (présenté Pages 7 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 1. 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 [ABND95] ainsi que de ceux de Lynch et Shvartsman [LS02].

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

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**($\langle vnew, tnew\rangle$)$_i$

25: **Consulte()**$_i$**:**
26: Soit Q un quorum de $\mathcal{Q}$
27: $s \leftarrow s + 1$
28: send$(\mathsf{GET}, \perp, s)$ aux nœuds de Q
29: **Repeat:**
30: **if** recv$(\mathsf{ACK}, \langle v, t\rangle, s)$ 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** $\langle vlast, tmax\rangle$

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.

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

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

Le numéro de séquence s est utilisé comme un compteur de phase pour pallier l'asynchronie et éviter qu'un nœud prenne en compte un message périmé. Plus pécisément, 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.

1.3.2 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 1.2. 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).

Theorem 1.1 *L'Algorithme 1 implémente un objet atomique.*

Proof. La preuve montre successivement que l'ordre des tags sur les opérations vérifie les trois propriétés de la Définition 1.2. 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 [DGLC04] ou bien lorsqu'il est clair que la valeur à jour a déjà été propagée à tout un quorum lorsque la lecture consulte [CGG+05, GAV07].[2] De telles approches présentent des cas particuliers que ne spécifie pas l'Algorithme 1 pour des raisons de simplicité dans la présentation.

2 Tolérer le dynamisme

2.1 Introduction

Reconfigurer une mémoire partagée sans altérer les opérations en cours est difficile. Afin de satisfaire la sûreté et la vivacité des opérations, la reconfiguration ne doit pas violer la cohérence et doit assurer la terminaison des opérations. Cette section s'intéresse à cette problématique et propose une mémoire partagée distribuée intégrant un méchanisme de reconfiguration afin de pouvoir être utilisé dans les systèmes dynamiques.

Cette partie présente les différents problèmes rencontrés pour implémenter une mémoire partagée distribuée dans un contexte dynamique. Pour cela, elle enumère les propositions de solutions adoptées dans le Chapitre 2 de la thèse. Cependant la mémoire résultante de ces propositions n'est pas présentée ici.

[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 [ABND95].

2.2 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 où les pannes s'accumulent. Un méchanisme récurrent doit être effectué afin de tolérer les pannes qui s'accumulent perpétuellement. Ici, nous décrivons un méchanisme 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 méchanisme 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 sans panne. Le but est d'assurer que tous les nœuds de quelques quorums soient toujours acitfs (en général deux comme explicité ultérieurement dans ce document).

Dans [Her86], 4 étapes indiquent ce que peut être un méchanisme de reconfiguration simple. Ceux-ci sont les suivants : *(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.

2.3 Indépendance des opérations

Certains travaux, comme [MA04], proposent de stopper les opérations durant la reconfiguration. Par conséquent une reconfiguration bloquante peut empêcher les opérations de terminer, et les opérations dépendent de la reconfiguration. Dans la suite nous nous intéressons à l'indépendence des opérations par rapport à la reconfiguration.

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

Dans [LS02], Lynch et Shvartsman 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 dans laquelle une nouvelle configuration peut-être installée permettant à plusieurs configurations de cohabiter.
- La mise à jour de la configuration permettant de mettre à jour la dernière configuration installée en rassemblant les informations présentes dans toutes les configurations au sein de la nouvelle configuration.

Les premières tentatives [LS97, ES00] 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 est impossible et le système peut tomber en panne.

2.4 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 [LS02] propose l'utilisation d'un algorithme de consensus permettant à plusieurs participants de se mettre d'accord sur une nouvelle configuraton à installer. Cet algorithme nécessite un algorithme de consensus indépendant, comme Paxos [Lam89] pour assurer qu'une nouvelle configuration est choisie de façon décentralisée.

2.5 Installation et suppression de configurations.

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

Une première implémentation de ces méchanismes dans RAMBO [LS02] permettait d'installer une nouvelle configuration à la demande. Cependant un méchanisme indépendant de recyclage permettait d'informer les clients potentiels que les configurations précédentes étaient devenues obsolètes. Plus précisément, ce méchanisme de recyclage s'exécutait indépendamment afin de supprimer la plus

ancienne des configurations actives si plusieurs configurations actives cohabitaient dans le système.

Dans cette première version de l'algorithme RAMBO, un client doit contacter plusieurs configurations afin d'exécuter ses opérations. Durant le temps ou plusieurs configuration cohabitaient, chaque nœud doit donc contacter non pas un seul quorum par phase de son opération mais un ensemble de quorums : un quorum par configuration active. 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 croit de manière correspondante pour toute opération au fur et à mesure que le temps passe.

Afin d'empêcher le nombre de configurations de croître indéfiniment si la fréquence de recyclage est trop faible, la nouvelle version de RAMBO, appelée RAMBO II [GLS03] intègre un méchanisme de recyclage agressif permettant de supprimer non plus une seule configuration mais un ensemble de configurations en même temps. L'idée est simplement 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 méchanisme empêche le nombre de configurations actives de croître indéfiniment si le recyclage est effectué de façon régulière. Cependant, ce méchanisme reste indépendant de l'installation de reconfiguration et ne permet pas de restreindre le nombre de configuration actives à une petite constante (une ou deux par exemple).

La solution que nous proposons dans la section 2.7 permet de limiter le nombre de configurations en obligeant la suppression de la configuration obsolète à chaque nouvelle installation. Ainsi, cette solution couple l'installation et la suppression et il n'existe jamais plus de deux configurations actives 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'astraire des hypothèses de RAMBO et RAMBO II.

2.6 Ordonner les configurations en utilisant le consensus

Le consensus permet à différents nœuds de se mettre d'accord sur une décision à prendre. Dans notre cas, il est nécessaire que les configurations soient choisies de façon uniforme par tous les nœuds. Étant donné que la suppression des confi-

gurations 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'informations 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 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.

Ainsi, il est 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 uniquement celle-ci. Cela revient donc à assurer un ordre total sur les configurations successivement installée, chaque nœud ne pouvant installée la $i + 1^{eme}$ configuration que si la i^{eme} 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 mimimal et la variante de Paxos permettant ces performances est appelé "Paxos Rapide" [Lam06].

Paxos a tout d'abord été décrit comme une suite de règle informelle dans un rapport technique de 1989, publié neuf ans plus tard. Cette présentation décrit le fonctionnement d'un parlement sur une île grecque, appelée 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. Cette algorithme apparait comme un algorithme permettant d'assurer la cohérence en dépit d'une quantité arbitraire de pannes pourvu que le système retourne à un état où plus de la moitié des participants sont présents.

Figure 1 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. Les accepteurs s'interdisent de participer dans un référendum si ils découvrent un autre référendum avec un identifiant plus large (**Phase 1b**). Les ac-

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 ballot plus à jour s'interdit de participer à tout précédent référendum.
Phase 2a. Après avoir reçu les réponses d'au moins un quorum d'accepteurs, le coordinateur choisi une nouvelle valeur pour son référendum et informe un quorum d'accepteur 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 and 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.

FIG. 1 – Description de l'algorithme Paxos

cepteurs peuvent voter pour un référendum si ils ne se le sont pas interdit avant. Un référendum gagne si et seulement 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 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 informants de la valeur décidée ; cette réception met fin au référendum (**Phase 2b**).

2.7 Mémoire partagée distribuée dynamique

Cet algorithme couple l'algorithme d'installation d'une nouvelle configuration avec celui de désinstallation de la configuration devenue, a fortiori, obsolète. Pour la première partie il utilise des échanges de messages entre les quorums afin que les participants décident d'une nouvelle configuration à installer. Pour la seconde partie il utilise également l'échange de message entre quorums afin de passer la valeur à jour de l'objet d'une configuration à une autre pour enlever la configuration obsolète sans perdre l'information importante. Tirant partie

de cet attraits commun aux deux algorithmes, l'algorithme résultant est davantage aggressif que toutes les solutions précédemment évoquées. La reconfiguration se fait rapidement, certains messages des deux parties sont assemblés afin de gagner en performance, et l'algorithme résultant améliore la tolérance aux fautes du fait de sa rapidité.

Néanmoins cette technique suppose que les nœuds accèdent directement à tous les participants d'un quorum. Cela nécessitent pour chaque potentiel client 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 trouvé 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 ou le nombre de clients est très grand, la charge induite peut nécessité également un grand nombre de serveurs. Dans ce cas, le coût en communication nécessaire à chaque opération peut amener des congestions dans le réseau. La Figure 2.7 compare le cas statique au cas dynamique ou dans un premier cas il n'y a pas de reconfiguration alors que dans le second il y a un nombre important de reconfigurations. Dans chaque relevé la taille d'une reconfiguration représente la moitié du nombre total d'instances dans le système. Il est clair que le surcout induit par l'exécution périodique des reconfigurations est faible cependant ce cout dépend étroitement du nombre d'instances de l'algorithme et de la taille des configurations.

D'après ces résultats, la mémoire atomique ne semble pas passer à l'échelle lorsque le nombre de participants dans chaque configuration devient grand. Cela s'explique par la bande passante limitée de tels systèmes et la congestion induite par le grand nombre de message requis.

3 Tolérer le passage à l'échelle

Cette partie présente une version simplifiée de la solution présenté dans le chapitre 3 de la thèse. Cette version est également locale ne nécessitant pas que

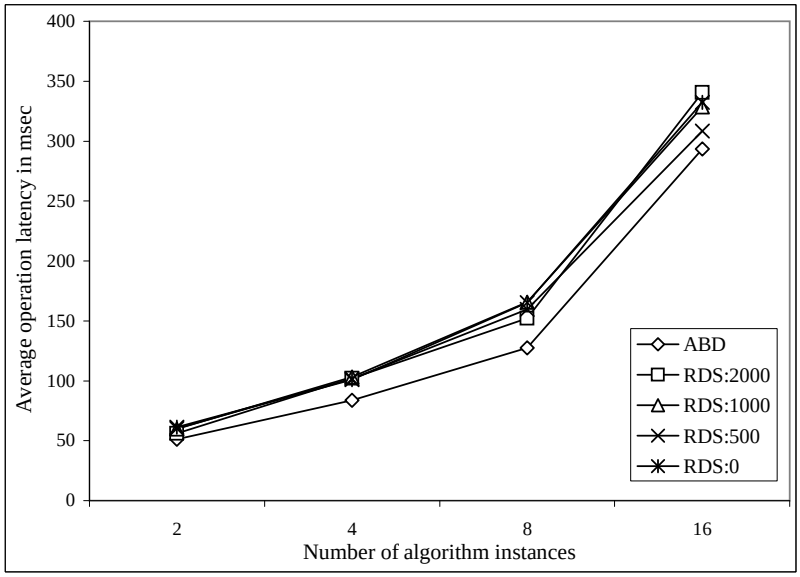

FIG. 2 – Average operation latency as the reconfiguration and the number of participants changes.

les nœuds aient une vue globale du système ou même une vue globale du système de quorums. Cependant cette version simplifiée n'indique pas comment empêché le blocage des opérations.

En fait, le travail présenté ici fait l'objet des prémisses qui ont permis d'aboutir à la solution plus complexe présentée au chapitre 3 de la thèse. Il donne l'idée principale qui a abouti aux résultats du chapitre 3.

3.1 Introduction

À cause des besoins induits par les systèmes 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.

3.2 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 $O(q)$.

3.3 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 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* [NW03].

Récemment, certains auteurs se sont intéressés aux quorums dynamiques [NW03, NN05, AM05], dont les serveurs changent progressivement avec le temps. Dans [AM05], 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 [NW03] 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 » [NN05] 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 p2p, non seulement pour leur dynamisme inhérent mais également pour leur propriété de localité. Par exemple, SAM [AGGV05] et SQUARE [GAV07] utilisent des spécificités dynamiques afin d'adapter la structure des systèmes de quorums aux variations de charges imprévisibles en contexte grande échelle. La structure torique utilisée bénéficie de l'adaptativité du partage d'un plan en sous-zones similairement à CAN [RFH+01] et l'efficacité des quorums en grille. 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 3 représente un accès au quorum : le client C contacte tour-à-tour chacun des serveurs d'un quorum.

3.4 Mémoire partagée distribuée passant à l'échelle

Pour chaque objet un espace de coordonnées réelles est partagé au sein des réplicas présents dans le système à la manière de CAN [RFH+01]. Nous nous intéressons ici à un seul objet et donc à un seul système de quorums. Une plus large mémoire s'étend trivialement comme composition de ces objets atomiques. Dans notre cas, on se donne un espace bidimensionnel $[0,1) \times [0,1)$ dont le premier nœud détenteur de l'objet est initialement responsable. Au fur et à mesure de l'arrivée de nouveaux nœuds, l'espace est de plus en plus divisé. À

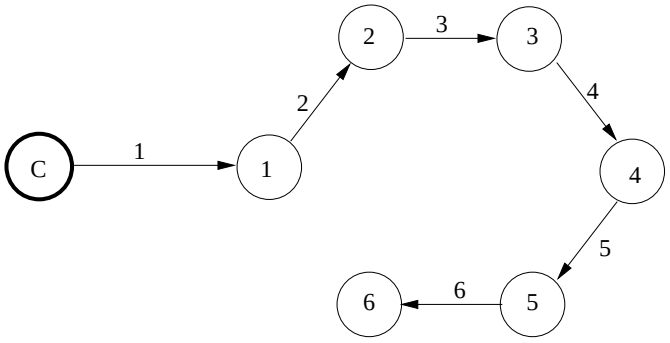

FIG. 3 – L'approche adaptative : le client C accède à un quorum en contactant chaque serveur du quorum tour-à-tour. Les opérations sont executées en temps $O(q)$.

contrario, les départs (ou pannes) de nœuds entraînent une fusion de ces sous-espaces appelés *zones*, et l'union de ces zones représente à tout moment l'espace $[0,1) \times [0,1)$. Deux nœuds sont dits *voisins* si ils sont responsables de zones adjacentes. La figure 4 représente un tore $[0,1) \times [0,1)$ divisés en de multiples zones.

Chaque nœud est représenté par un unique identifiant. On note l'ensemble de ces identifiants $I \subset N$. L'ensemble des valeurs que peut prendre l'objet est noté V et l'ensemble des estampilles associées est T. Chaque réplica conserve une certaine vision de l'objet, c'est-à-dire qu'il conserve la dernière valeur (selon lui) de l'objet et l'estampille associée à l'écriture correspondante. Une estampille est associée à chaque valeur écrite, et est monotoniquement incrémentée à chaque nouvelle écriture. Ainsi l'estampille définit un ordre total sur les opérations d'écritures effectuées. Afin de différencier deux écritures concurrentes effectuées depuis différents nœuds, l'estampille possède un entier de poids faible correspondant à l'identifiant du nœud l'exécutant. Ainsi l'estampille est un couple $\langle compteur, id\_noeud \rangle \in N \times I$.

Le tore utilisé a pour borne b telle que $b.xmin = b.ymin = 0$ et $b.xmax = b.ymax = 1$. Chaque réplica de la mémoire atomique est responsable d'une zone bornée. Cette zone z est délimitée par un rectangle défini par un point inférieur gauche de coordonnées $(z.xmin, z.ymin)$ et un point supérieur droit de coordon-

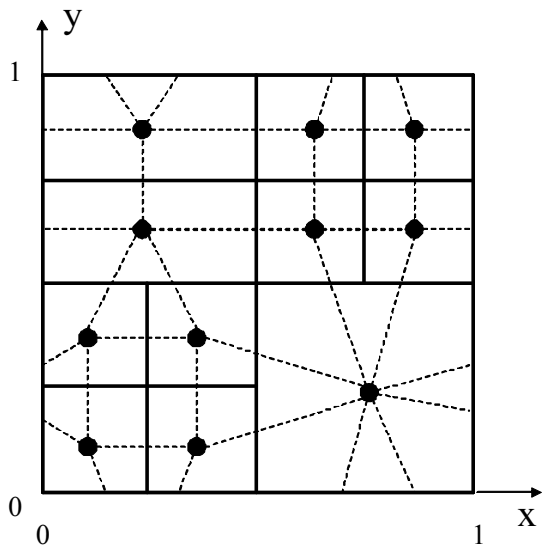

FIG. 4 – La topologie en tore : L'espace de coordonnées est découpé en zones. Chaque zone possède un réplica (représenté par un point) responsable. Deux réplicas sont voisins (liés par des pointillés) si les zones dont ils sont responsables sont adjacentes.

nées $(z.xmax, z.ymax)$. La zone b attribuée au premier nœud correspond donc à l'ensemble des points dont l'abscisse est dans l'intervalle $[b.xmin, b.xmax)$ et l'ordonnée est dans l'intervalle $[b.ymin, b.ymax)$. On utilisera la notation $[z.xmin, z.xmax) \times [z.ymin, z.ymax)$ pour caractériser la zone z.

3.5 Opérations atomiques

Les opérations considérées sont des opérations de lecture et d'écriture. Elles sont divisées en deux phases, à l'instar des opérations non-adaptatives de [ABND95, LS02]. Ces phases sont appelées phase de consultation et phase de propagation.

Une écriture s'effectue seulement sur certains nœuds du système de quorums. Cela implique qu'il existe des nœuds qui ne sont pas à jour lorsqu'une écriture termine. Cependant le protocole d'une opération de lecture assure qu'au moins un nœud à jour soit consulté. Afin de détecter ce nœud parmi l'ensemble des nœuds consultés, il est important d'associer une estampille à la valeur, indiquant son numéro de version : soient e_1 et e_2 deux estampilles associées respectivement aux valeurs v_1 et v_2, si $e_1 < e_2$ alors la valeur v_2 associée est plus à jour que la valeur v_1.

Chaque quorum de consultation intersecte tout quorum de propagation. Ainsi une consultation informe l'initiateur au sujet de la dernière valeur propagée sur n'importe quel quorum de propagation. Cette phase est utile à la lecture pour connaître la valeur à renvoyer, mais elle permet aussi à l'écriture de récupérer la plus grande estampille du système. Ensuite, la phase de propagation permet d'assurer que lorsqu'une opération termine, toute opération suivante ne prendra pas en compte une ancienne valeur ou une estampille non maximale. Si l'opération est une lecture, cette dernière phase propage la paire $\langle$ estampille, valeur $\rangle$ lue à la phase précédente. Dans le cas d'une écriture, cette phase propage la valeur à écrire avec une nouvelle estampille plus grande que celle consultée à la phase précédente. Ces phases sont détaillées dans les sous-sections 3.6.2 et 3.6.3.

3.6 Routage

Chaque nœud a connaissance de ses voisins. Ainsi chaque nœud conserve l'identité de ces derniers et a la capacité de les contacter. Le routage permet de

définir un chemin suivant lequel une information transite. Dans le cas d'une phase de consultation, le routage assure que la plus grande estampille et la valeur à jour associée rencontrées, sont transmises de voisin en voisin jusqu'à l'initiateur suivant un anneau horizontal du tore. La phase de propagation quant à elle, permet de propager la dernière valeur et son estampille maximale associée aux éléments constituant un anneau vertical du tore. Ces valeur, estampille proviennent de la précédente phase.

Néanmoins, il peut arriver qu'un message soit perdu, ou qu'un des nœuds contactés tombe en panne. Pour pallier à une perte éventuelle de message, l'initiateur envoie régulièrement les messages d'une phase tant qu'il n'a pas appris sa terminaison. Dans le cas où un nœud à contacter tombe en panne, cela revient à des pertes de message avant qu'un voisin prenne la responsabilité de sa zone. Comme détaillé dans les sections suivantes, un des voisins prend la responsabilité de la zone du réplica fautif et en informe ses voisins. Dans le cas où l'initiateur tombe en panne, il est possible que l'information sur la requête reçue soit perdue. Ainsi une opération peut ne pas terminer. À noter que l'atomicité fait l'objet d'une propriété de sûreté et il est nécessaire de supposer que les opérations initiées terminent pour qu'elles soient linéarisables. Enfin l'étude temporelle de ce protocole ne fait pas l'objet de cette section et nous ne présentons pas de résultats sur la progression où la vivacité de celui-ci.

3.6.1 Quorums dynamiques

Dans le contexte des quorums en grille, un quorum est défini comme une ligne ou une colonne. Ainsi il existe deux types de quorums, et n'importe quel quorum d'une ligne intersecte n'importe quel quorum d'une colonne. Dans notre contexte, chaque zone découpant l'espace bidimensionel n'est pas nécessairement carrée. C'est pourquoi nous définissons deux types de quorums légèrement différents. Un quorum de consultation (resp. de propagation) est un ensemble de nœuds définis par une traversée horizontale (resp. verticale) effectuée par le routage précédemment cité.

Un quorum regroupe un ensemble d'éléments en fonction de leur zone. Ainsi les éléments constituant les quorums dépendent des limites de leur

zone au moment où ils sont contactés. Le dynamisme de ces zones induit un dynamisme dans le choix des nœuds d'un quorum. Autrement dit, si un même initiateur est contacté pour la même phase à deux instants différents, les quorums qu'il contacte ne sont pas nécessairement identiques du fait des départs et arrivées de nouveaux nœuds. Les quorums utilisés ici sont dit *dynamiques*.

Pour des raisons évidentes de simplicité les définitions de quorums suivantes ne mentionnent pas de paramètres temporels induit par le dynamisme.

Un quorum de consultation, comme son nom l'indique, est un ensemble de nœuds qui sont consultés afin d'obtenir la valeur et l'estampille qu'ils conservent de l'objet.

Definition 3.1 (Quorum de consultation Q_c) *Un quorum de consultation $Q_c \subset I$, est l'ensemble des nœuds responsables des zones de Z_c telles que*

- $\bigcup_{\forall z \in Z_c} \{[z.xmin, z.xmax)\} = [b.xmin, b.xmax)$
- $\bigcap_{\forall z \in Z_c} \{[z.xmin, z.xmax)\} = \emptyset$
- $\exists z \in Z_c, \forall z' \in Z_c, z.ymin + (z.ymax - z.ymin/2) \in [z'.ymin, z'.ymax)$

Un quorum de propagation est un ensembe de nœuds sélectionnés pour mémoriser l'écriture d'une nouvelle valeur. Ces nœuds mettent à jour la valeur de l'objet et l'estampille associée qu'ils conservent.

Definition 3.2 (Quorum de propagation Q_p) *Un quorum de propagation $Q_p \subset I$, est l'ensemble des nœuds responsables des zones de Z_p telles que*

- $\bigcup_{\forall z \in Z_p} \{[z.ymin, z.ymax)\} = [b.ymin, b.ymax)$
- $\bigcap_{\forall z \in Z_p} \{[z.ymin, z.ymax)\} = \emptyset$
- $\exists z \in Z_p, \forall z' \in Z_p, z.xmin + (z.xmax - z.xmin/2) \in [z'.xmin, z'.xmax)$

La classification des quorums en deux types, où tout quorum d'un type intersecte ceux de l'autre type, a été initialement proposée par Herlihy [Her86].

3.6.2 Phase de consultation

Tout nœud du système peut initier une opération de lecture ou d'écriture à n'importe quel moment sous réserve que toute opération initiée avant ait terminé. Ce nœud doit avoir joint le système et y être toujours actif mais peut également

être un réplica d'un des quorums de la mémoire atomique.

Les deux types d'opérations commencent par une phase de consultation où un quorum de consultation est contacté. Afin que chaque nœud puisse initier une opération, on émet l'hypothèse qu'une fonction prenant les coordonnées réelles d'un point permet de retrouver le responsable de la zone le contenant. Ainsi le nœud peut prendre connaissance d'un réplica arbitrairement choisi et lui envoyer la requête de l'opération.

Tout nœud de ce système de quorum appartient à au moins un quorum de consultation d'après la définition 3.1. Ainsi lorsqu'une requête d'opération parvient à un tel réplica i, appelé l'*initiateur*, i propage selon une traversée horizontale un message contenant (i) son estampille, (ii) sa valeur et (iii) sa zone. Étant donnée que chaque réplica connait ses voisins directs, le message est propagé de voisin en voisin dans un sens horizontal comme indiqué sur la figure 5. Il est possible que plusieurs réplicas soient voisins dans cette direction. Dans ce cas, la traversée est déterminée par une fonction prenant en argument les limites verticales de la zone de l'initiateur ($i.ymin$ et $i.ymax$). Puis j est choisi parmi les voisins dans le sens considéré tel que l'ordonnée du centre de la zone de i appartienne à l'intervalle verticale de la zone de j. Formellement on obtient j tel que :

$$i.ymin + (i.ymax - i.ymax)/2 \in [j.ymin, j.ymax)$$

À chaque fois qu'un nœud reçoit ce message il inspecte le contenu du message. S'il contient une estampille plus grande ou égale à la sienne alors il propage le même message, sinon cela signifie qu'il a connaissance d'une valeur plus à jour ; dans ce cas il remplace l'estampille et la valeur du message par les siennes. Du fait de cette fonction déterministe et de la topologie en anneau parcourue, il est inévitable que l'initiateur i reçoive le message qu'il avait le premier envoyé. De plus, comme l'estampille est mise à jour (si nécessaire) à chaque réception du message, l'initiateur récupère la plus grande estampille et la dernière valeur dont tout élément du quorum de consultation a connaissance. Dans le cas d'une opération d'écriture, l'entier de l'estampille rencontrée est incrémentée, son identifiant devient l'identifiant de celui qui a émis la requête et la valeur devient celle à écrire. Dans le cas d'une opération de lecture la paire $\langle$ estampille, valeur $\rangle$ reste inchangée. Dans les deux cas, la paire résultante est celle utilisée dans la phase de

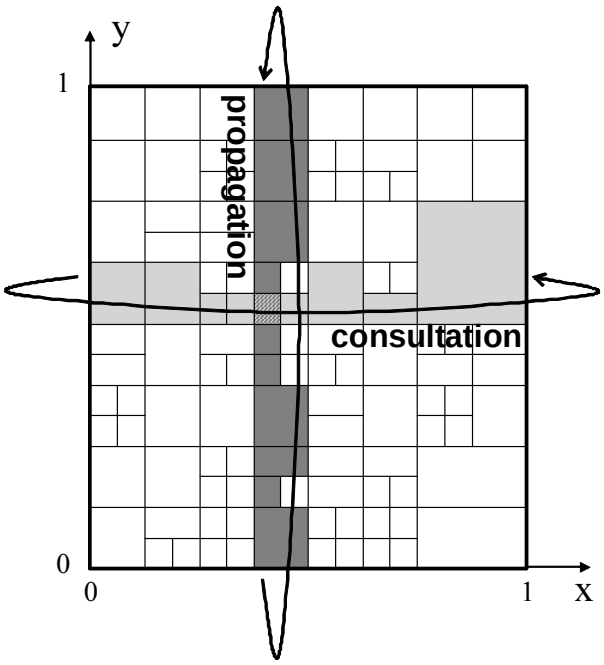

FIG. 5 – Les phases de traversées : Ici sont représentées deux traversées. La première, horizontale, indique les nœuds d'un quorum de consultation contactés et est appelée phase de consultation. La seconde, verticale, indique un quorum de propagation et est appelée phase de propagation.

propagation et son estampille définit l'ordre de l'opération.

3.6.3 Phase de propagation

La seconde phase des opérations est effectuée de façon analogue. L'initiateur envoie un message correspondant à cette phase et contenant la paire ⟨ estampille, valeur ⟩, dans un sens vertical bien précis. La traversée est effectuée par une retransmission de voisin en voisin du même message. À la différence de précédemment, le message n'est pas modifié mais la paire ⟨ estampille, valeur ⟩ conservée localement par chacun des participants est modifiée si une plus récente est reçue (i.e., une estampille plus grande est reçue). Une traversée est effectuée selon un axe vertical de telle manière que tout élément d'un quorum de propagation ait la paire ⟨ estampille, valeur ⟩ à jour.

Claim 3.1 *L'ordre partiel des opérations défini par l'estampille assure la propriété d'atomicité.*

La preuve d'atomicité d'un seul objet dépend essentiellement de l'intersection non vide des deux types de quorums. L'atomicité de la mémoire résulte de la composition des objets atomiques.

3.7 Tolérance aux défaillances

Pour que notre service soit efficace en terme de *qualité de service,* il faut que la disponibilité des données soit toujours assurée. Ainsi le service doit être tolérant aux défaillances.

3.7.1 Réplication des données

Lorsque les données sont centralisées, une panne du système entraîne une indisponibilité du service associé. Certains serveurs adoptent une approche répliquée en utilisant des serveurs-mirroirs. Cette approche ne résoud pourtant pas les problèmes liés aux variations incontrôlables de la charge du système. La particularité d'un grand système est le nombre important de participants potentiels qu'il contient. Notre réplication s'effectue au sein d'un ensemble de ces nœuds participants. Ce regroupement au sein d'un réseau de pairs peut être

basé sur un recoupement de centres d'intérêt : sachant que deux nœuds possèdent des objets similaires il est probable, que la réplication d'un objet du premier nœud sur le second profite à ce dernier. Nous laissons cette remarque de côté, celle-ci ne faisant pas partie de l'objet de cet section.

Ainsi un certain ensemble de nœuds possèdent une réplication de la donnée et si une partie stricte d'entre eux tombent en panne au même instant, l'objet reste accessible en dépit des pannes. Comme mentionné dans la sous-section 3.6.1, chaque opération consiste à contacter des quorums. Ainsi le service est assuré si et seulement si le quorum contacté est disponible. Or, la définition de nos quorums et l'auto-adaptation de notre système assure l'existence d'un quorum de consultation et un quorum de propagation s'il reste au moins un réplica actif.

Également, lorsque les pannes s'accumulent un mécanisme d'auto-réparation assure le remplacement de réplicas par une réplication active au sein de nœuds extérieurs.

3.7.2 Détection de défaillances

La détection de défaillance permet au système de s'auto-adapter pour que les quorums nécessaires aux opérations restent disponibles. En effet lorsqu'un nœud tombe en panne, sa zone n'a plus de responsable. Ainsi il existe des nœuds du système de quorums d'où part une traversée qui n'aboutira pas. Ces traversées sont celles qui doivent passées par le nœud fautif. L'inactivité de ce nœud bloque ainsi la traversée.

Pour pallier à ce problème un mécanisme de détection de faute est utilisé. Celui-ci repose sur un échange de messages périodiques, qualifiés de *battements de cœur*. L'absence de réception d'un tel message durant une certaine période de temps indique au récepteur que l'émetteur est tombé en panne. Chacun des messages de battement de cœur sert également à mettre à jour la table des voisins, c'est-à-dire pour chacun d'entre eux, leur zone, les identifiants de leurs voisins, l'estampille et la valeur qu'ils conservent.

3.7.3 Arrivée d'un nœud

On suppose qu'à la création du système un seul nœud est détenteur de l'objet. Ainsi il crée son système de quorum tel qu'il soit l'unique responsable de toute la zone $[b.xmin, b.xmax) \times [b.ymin, b.ymax)$.

Lorsqu'un nœud décide de joindre le système, il contacte un nœud déjà présent, attend son accusé-réception et devient actif. À ce stade le nœud est seulement un candidat potentiel pour faire partie du système de quorum. Ce n'est que lorsque le système décide de s'étendre par le mécanisme d'auto-reconfiguration, qu'un réplica i choisi parmi les candidats potentiels, un nœud j pour y forcer la réplication. Puis i partage sa zone en deux parties égales, et en attribue une au nouveau réplica j. Il informe j de la zone dont il est responsable et de l'identité des réplicas dont les zones sont adjacentes à celle de j, i.e., ses voisins. Cette intégration d'un nouveau nœud est détaillée dans la sous-section 3.8.3.

3.7.4 Départ d'un nœud

Il est important de constater qu'une traversée ou une diagonalisation (cf. sous-section 3.8.2) peut être retarder entre le moment où un voisin concerné tombe en panne et le moment où le voisinnage est redéfinit. Le protocole qui suit est similaire à celui proposé dans la table de hachage CAN [RFH+01]. Lorsqu'un réplica i est considéré comme étant en panne, le système décide de s'auto-reconfigurer afin de permettre aux quorums d'être toujours accessibles. Pour cela un nœud voisin ne recevant pas de messages de *battements de cœur* de la part de i le considère comme fautif. À noter qu'il existe plusieurs voisins détectant cette panne. Un voisin j est choisi parmi ceux-ci selon un critère simple : le nœud ayant découpé sa zone en dernier est le nœud choisi. À noter qu'un nœud peut éventuellement être responsable de plusieurs zones afin d'assurer que leur forme reste rectangulaire. Ensuite la zone dont le réplica choisi est responsable, est étendue avec la zone du réplica qui a quitté le système. Le réplica considéré connait l'identité des voisins du nœud parti, et contacte ses voisins diagonaux pour récupérer la paire $\langle$ estampille, valeur $\rangle$ (ou une plus à jour) des voisins verticaux. Enfin si l'estampille découverte est plus récente, il change sa paire locale en celle découverte et informe ses voisins de sa nouvelle zone. Le mécanisme de traversée ou de diagonalisation éventuellement en attente est alors poursuivi.

3.8 Répartition de la charge

Chacun des réplicas présents dans le système de quorums subit une charge due aux nombres de requêtes qu'il reçoit durant une courte période de temps. D'après le fonctionnement des opérations, on remarque que lorsqu'un réplica effectue une opération en tant qu'initiateur, il provoque une charge quasiment équivalente sur chacun des membres de son quorum. Ainsi si la charge d'un nœud est c à un instant, on sait que $\Omega(c)$ sera attribué à chacun des nœuds des quorums auxquels il appartient dans le cas où toute requête est traitée.

3.8.1 Charge et Complexité

Il existe un compromis entre la charge et la complexité du système. Lorsque les quorums contiennent en moyenne un grand nombre de réplicas, la complexité en nombre de messages d'une traversée est plus importante. Cependant lorsque leur taille moyenne est petite, cela signifie que la charge globale du système est répartie sur un nombre moins important de réplicas et la charge moyenne d'un réplica est d'autant plus grande. Ici, nous nous efforçons d'utiliser des quorums dont l'intersection avec un autre quorum contient exactement un réplica. Ainsi, à condition d'avoir une stratégie d'accès uniforme en tout point du système de coordonnées, le système de quorums atteint n réplicas avec des quorums de taille $O(\sqrt{n})$. Et, la complexité de contact d'un quorum – définie par le temps et le nombre de messages nécessaires pour contacter un quorum – est optimale comme montré dans [NW03].

Un problème se pose lorsque la charge d'un nœud devient trop importante. Cela signifie que le taux de requêtes reçues devient supérieur au taux de traitement des opérations du nœud. On ajoute au modèle précédent une file permettant à un réplica de stocker les requêtes reçues en attendant de les traiter. Après un certain délai de garde ces requêtes sont traitées. Un nombre seuil de requêtes enregistrées définit une surcharge du réplica. Ce nombre reste inférieur à la capacité de la queue.

3.8.2 Diagonalisation

Si un réplica surchargé reçoit une requête alors il transmet directement la requête à un autre nœud. D'après la remarque précédente, on sait qu'il est inutile pour un réplica i d'envoyer la requête à un membre j d'un même quorum. En effet, si j accepte de traiter la requête, il a davantage de chance d'être surchargé puisque i le sollicitera pour les opérations dont certaines requêtes sont dans sa queue. Réciproquement, j acceptant cette requête pourra éventuellement demander la participation de i dans l'opération, entraînant une augmentation de la charge de i celui-ci étant déjà surchargé.

Afin de contacter des quorums différents nous définissons un parcours diagonal parmi les réplicas du système de quorums. De plus, afin d'assurer la détection d'un système surchargé, ce parcours suit une diagonale parallèle à celle traversant le tore du point $(xmin, ymin)$ au point $(xmax, ymax)$. Ainsi tout réplica dont la zone est traversée par cette droite, est contacté. Si le réplica contacté est un voisin indirect du dernier nœud à avoir tester sa charge, alors il vérifie sa charge à son tour. S'il n'est pas surchargé, il insère la requête dans sa queue. Sinon il retransmet la requête et le scénario est répété. Cette stratégie est représentée sur la figure 6.

3.8.3 Auto-expansion

Comme le mécanisme poursuit sur une trajectoire parallèle à la diagonale, l'initiateur de cette diagonalisation reçoit finalement le message si aucun des réplicas contactés ne décide d'initier l'opération. Ainsi, l'initiateur sait que tous les replicas testés étaient surchargés. Dans ce cas, une approximation sur la charge apparemment élévée du système est faite et l'initiateur décide d'étendre le système de quorums.

Lorsqu'un réplica i décide d'étendre le système de quorum, il fait appel à un nœud du système hors du système de quorum. Autrement dit, un nœud actif mais n'étant pas membre des quorums. Il est raisonnable de supposer l'existence d'un tel nœud car la charge du système de quorum est proportionnelle au nombre total de nœuds actifs par rapport au nombre de réplicas du système de quorums. Or, si les nœuds actifs du système sont uniquement les réplicas membres des quorums et que le système est en phase d'expansion cela signifie que les réplicas suffisent par leur requête à se surcharger eux-mêmes. Plus précisément, avec

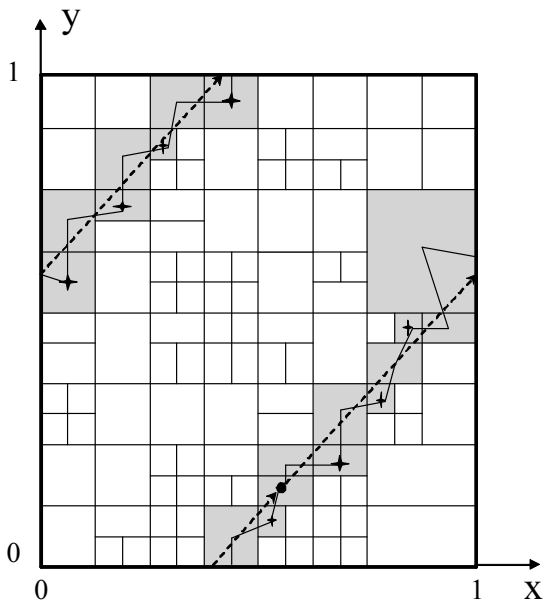

FIG. 6 – La diagonalisation : Les zones grisées sont celles des réplicas impliqués dans la diagonalisation. Le point indique l'initiateur de la diagonalisation et les croix représentent les autres réplicas inspectant leur charge. Dans ce scénario tous les réplicas réprésentés par des croix ou un point sont surchargés. Ainsi la diagonalisation s'achève seulement lorsque le message retourne a l'initiateur.

$\tau_r(p)$, le taux de requêtes que peut délivré un nœud pendant une période p, $\tau_t(p)$ le taux maximal de traitement des requêtes effectuées par un nœud durant la même période et tq la taille maximale d'un quorum, alors une surcharge durable n'est possible que lorsque $\tau_r(p) > \frac{\tau_t(p)}{tq}$. Le terme « durable » est utilisé pour pallier à l'irrégularité dans la réception des messages induit par l'asynchronisme. Ce terme signifie durant une période assez longue pour que tout message non perdu arrive à destination. Dans la suite, on suppose l'existence d'un tel nœud lorsqu'une expansion a lieu.

Une fois que i trouve un nœud j actif mais non membre du système de quorums, i réplique l'objet en copiant la valeur de l'objet qu'il possède et l'estampille associée en j. Il informe également j qu'il fait partie de ses voisins. Alors i découpe sa zone en deux parties égales selon un axe horizontal. Ce choix est effectué par simplicité mais il peut s'agir d'une découpe variable afin de favoriser la rapidité de certaines opérations comme présenté dans [AGGV05]. Une des deux zones obtenues, est alors attribuée à j, et i met à jour sa zone de responsabilité avec la seconde zone obtenue. L'identité des voisins de i devenus voisins de j du fait de cette découpe est également transmise à j par i.

3.8.4 Auto-réduction

Si pendant une trop longue période de temps, définie préalablement en fonction de l'application, aucune requête n'a été effectuée sur un réplica, ce réplica peut décider de provoquer une réduction du système de quorums afin de diminuer la complexité de contact d'un quorum. Pour cela le réplica concerné cherche directement parmi ses voisins le dernier à avoir divisé sa zone et l'informe de son départ. L'auto-reconfiguration s'effectue comme mentionné dans 3.7.4 une fois que la faute a été détectée.

3.9 Conclusion

Le Chapitre 3 reprend la mémoire présentée ici et la développe. Le résultat est une mémoire qui s'adapte face aux variations de charge et aux dynamismes des participants mais qui assure que les opérations puissent terminées les unes indépendamment des autres.

4 Tolérer le dynamisme et le passage à l'échelle

4.1 Introduction

L'idée principale du Chapitre 4 de la thèse est d'affaiblir les contraintes déterministes afin d'améliorer les performance d'une mémoire distribuée partagée. Ce chapitre propose un affaiblissement de la propriété d'atomicité où certaines des garanties qui devaient être assurées dans tous les cas, sont maintenant assurées avec une forte probabilité. Cet affaiblissement du critère de cohérence permet de trouver une solution tolérant à la fois le dynamisme et le passage à grande échelle en contrepartie de cet affaiblissement.

Dans la section qui suit nous proposons une solution à un problème plus simple que celui de la cohérence mémoire. Comme il est difficile de résumer la définition de la mémoire, le critère de cohérence qu'elle assure ainsi que la preuve de cette mémoire, nous donnons un apperçu d'une solution permettant d'assurer la persistance des données dans un système dynamique à grande échelle. La résolution de ce problème fourni une brique de base permettant de définir la mémoire présentée dans le Chapitre 4.

4.1.1 Contexte

Maintenir la persistance d'une donnée dans un système distribué est une nécessité pour beaucoup d'applications. Bien que de nombreuses solutions aient été proposées dans un cadre statique, cela reste un probléme ouvert dans un cadre dynamique. Un système dynamique est un système ou les participants (*nœuds*) quittent et (re)joignent le système arbitrairement souvent. Pour pallier non seulement au dynamisme et au passage à l'échelle, les garanties probabilistes y remplacent naturellement les garanties déterministes fortes. Quantifier les garanties qu'il est possible d'obtenir probabilistiquement reste primordiale pour des applications dynamiques à grande échelle.

Plus précisément, un des problèmes fondamentaux d'un tel contexte consiste à assurer en dépit du dynamisme, que les données critiques ne soient pas perdues. L'ensemble des nœuds détenant une copie de la donnée critique est parfois appelé un *noyau*. Plusieurs noyaux peuvent coéxister, chacun associé à une donnée spécifique. Pourvu qu'un noyau reste suffisamment longtemps présent dans le système, la donnée peut être transmise de nœuds en nœuds à l'aide d'un pro-

tocole de « transfert de données » aboutissant à la création d'un nouveau noyau. Cependant, l'utilisation d'un tel protocole nécessite de faire un choix sur la fréquence du transfert de données pour éviter un surcoût mais assurer que la donnée ne disparaisse pas.

4.1.2 Contributions

Cet section assure probabilistiquement la maintenance d'un noyau. Les paramètres pris en considération sont la taille du noyau, le pourcentage de nœuds qui rentrent et sortent par unité de temps et la durée d'observation du système. Soit S le système à un temps τ. Il est composé de n nœuds dont q nœuds représentent le noyau Q d'une donnée critique. Soit S', le système au temps $\tau + \delta$. D'après l'évolution du système certains nœuds de S peuvent avoir quitté le système au temps $\tau + \delta$. Une question importante est la suivante : Étant donné un ensemble Q' de q nœuds de S' quel est la probabilité que Q et Q' s'intersectent ? Cette section exprime cette probabilité sous forme d'une fonction dont les paramètres caractérisent le système dynamique.

Travaux similaires. Des travaux portent sur le maintien du routage [RD01] en système dynamique. Néanmoins le degré de réplication nécessaire dépend complètement du nombre de pannes ou de départs : une donnée répliquée $k + 1$ fois tolère seulement k pannes/départs. D'autres travaux [MTK06] étudient comment accéder à une donnée avec forte probabilité mais ne prennent pas en compte le dynamisme dans l'analyse.

4.2 Modèle

Le système est composé de n nœuds. Il est dynamique, dans le sens où cn nœuds rejoignent et quittent le système par unité de temps. Ces nœuds peuvent être vus comme étant remplacés, c représente donc le *va-et-vient* par nœud et par unité de temps. Un nœud quitte le système soit en se déconnectant volontairement ou lors d'une panne définitive. Lorsqu'un nœud rejoint le système il est considéré comme nouveau et la connaissance qu'il possédait du système est considérée comme perdue.

4.3 Relation entre les paramètres clefs du système dynamique

Cette section présente la relation des différents paramètres du système. Le Lemme suivant donne la portion C de nœuds qui sont remplacés après une période de δ unités de temps en fonction du va-et-vient c.

Lemma 4.1 *Soit C la portion de nœuds initiaux qui est remplacée après δ unités de temps. On obtient $C = 1 - (1 - c)^{\delta}$.*

Ce Lemme est facilement prouvable par récurrence sur δ (cf. [GKM+06]). Dans la suite de cette section on appellera α, le nombre de nœuds remplacés dans le système après δ unités de temps avec $\alpha = \lceil Cn \rceil = \lceil (1 - (1 - c)^{\delta})n \rceil$. Étant donné un noyau de q nœuds qui détiennent la donnée au temps τ, le Théorème suivant nous donne la probabilité de ne pas trouver cette donnée en contactant aléatoirement q nœuds après une période de δ unités de temps et le va-et-vient de α noeuds.

Theorem 4.2 *Soient $x_1, ..., x_q$, des nœuds du système au temps $\tau' = \tau + \delta$. La probabilité qu'aucun de ces nœuds n'appartient au noyau initial est*

$$\frac{\sum_{k=a}^{b} \left[\binom{n+k-q}{q} \binom{q}{k} \binom{n-q}{\alpha-k} \right]}{\binom{n}{q}\binom{n}{\alpha}},$$

avec $\alpha = \lceil (1 - (1 - c)^{\delta})n \rceil$, $a = \max(0, \alpha - n + q)$ et $b = \min(\alpha, q)$.

Proof. Le problème à résoudre peut être modélisé de la façon suivante : Une urne contient n boules telles que, initiallement, q parmi les n sont vertes et $n - q$ sont noires. Les boules vertes représentent le noyau initial $Q(\tau)$ et sont représentées par l'ensemble $\mathcal{Q}$. On tire aléatoirement et uniformément $\alpha = \lceil Cn \rceil$ boules de l'urne, on les peint en rouge et on les replace dans l'urne. Ces boules représentent les nœuds initiaux qui ont été remplacés après δ unités de temps.

Ainsi, l'état du système obtenu au temps $\tau' = \tau + \delta$ est différent. Soit $\mathcal{A}$, l'ensemble des boules peintes en rouge et soit $\mathcal{Q}'$ le noyau $\mathcal{Q}$ après que certaines boules aient été peintes en rouge. Soit $\mathcal{E}$ l'ensemble des boules vertes, $\mathcal{E} = \mathcal{Q}' \setminus \mathcal{A}$ au temps τ' et soit β le nombre de boules dans l'ensemble $\mathcal{Q}' \cap \mathcal{A}$. Nous savons

que β possède une loi de distribution hypergéométrique, i.e., pour $a \leq k \leq b$ où $a = \max(0, \alpha - n + q)$ et $b = \min(\alpha, q)$, on a :

$$\Pr[\beta = k] = \frac{\binom{q}{k}\binom{n-q}{\alpha-k}}{\binom{n}{\alpha}}. \quad (1)$$

Finalement, on tire aléatoirement et successivement sans remise, q boules $x_1, ..., x_q$ de l'urne (système au temps τ'). Le problème consiste à calculer la probabilité de l'événement {aucune des boules sélectionnées $x_1, ..., x_q$ n'est verte}, pouvant s'écrire $\Pr[x_1 \notin \mathcal{E}, ..., x_q \notin \mathcal{E}]$.

Comme $\{x \in \mathcal{E}\} \Leftrightarrow \{x \in \mathcal{Q}'\} \cap \{x \notin \mathcal{Q}' \cap \mathcal{A}\}$, on obtient (en prenant le contraire) $\{x \notin \mathcal{E}\} \Leftrightarrow \{x \notin \mathcal{Q}'\} \cup \{x \in \mathcal{Q}' \cap \mathcal{A}\}$, ainsi on peut conclure $\Pr[x \notin \mathcal{E}] = \Pr[\{x \notin \mathcal{Q}'\} \cup \{x \in \mathcal{Q}' \cap \mathcal{A}\}]$. Étant donné que les événements $\{x \notin \mathcal{Q}'\}$ et $\{x \in \mathcal{Q}' \cap \mathcal{A}\}$ sont disjoints, on obtient $\Pr[x \notin \mathcal{E}] = \Pr[x \notin \mathcal{Q}'] + \Pr[x \in \mathcal{Q}' \cap \mathcal{A}]$. Le système contient n boules. Le nombre de boules dans $\mathcal{Q}'$, $\mathcal{A}$ et $\mathcal{Q}' \cap \mathcal{A}$ est égal à q, α et β, respectivement. Par conséquent, on obtient, Comme il n'y a pas de répétition, on obtient simplement de la même manière,

$$\begin{aligned}
\Pr[x_1 \notin \mathcal{E}, ..., x_q \notin \mathcal{E} \,/\, \beta = k] &= \sum_{k=a}^{b} \prod_{i=1}^{q} \left(1 - \frac{q-k}{n-i+1}\right), \\
&= \sum_{k=a}^{b} \frac{\binom{n-q+k}{q}}{\binom{n}{q}}, \\
\Pr[x_1 \notin \mathcal{E}, ..., x_q \notin \mathcal{E}] &= \sum_{k=a}^{b} \frac{\binom{n-q+k}{q}}{\binom{n}{q}} \Pr[\beta = k], \\
&= \frac{\sum_{k=a}^{b} \left[\binom{n+k-q}{q}\binom{q}{k}\binom{n-q}{\alpha-k}\right]}{\binom{n}{q}\binom{n}{\alpha}}.
\end{aligned}$$

□

4.3.1 Relation entre la taille du noyau q et la probabilité ϵ.

Étant donné une valeur C requise par le concepteur d'une application, d'autres paramètres influent sur le coût du maintien d'un noyau ou la garantie d'avoir un tel noyau. Le surcoût est facilement mesurable en utilisant le nombre q de nœuds à contacter.

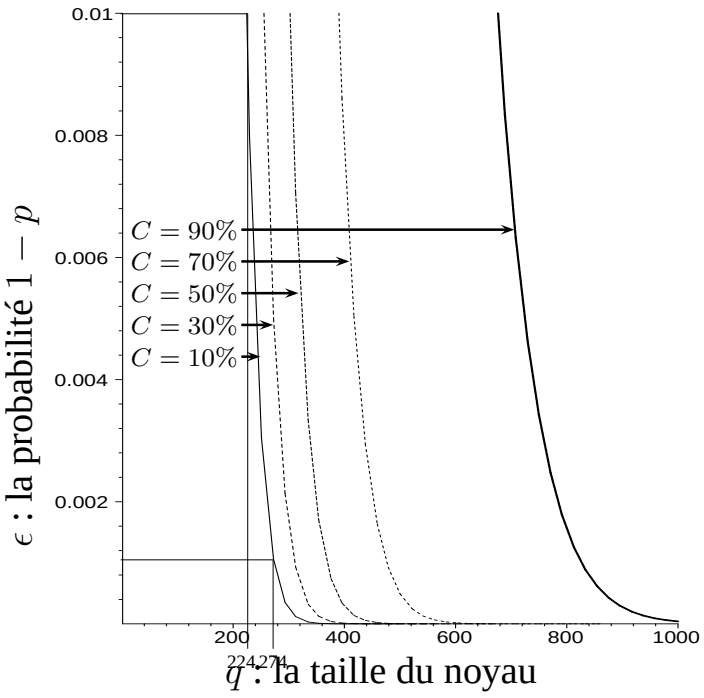

FIG. 7 – Probabilité ϵ de ne pas contacter un noeud du noyau en fonction de sa taille q.

On considère la valeur ϵ donnée par le Théorème 4.2. Cette valeur peut être interprétée de la façon suivante : $p = 1 - \epsilon$ est la probabilité que, au temps $\tau' = \tau + \delta$, une des q requêtes exécutée (aléatoirement) par un nœud s'adresse à un nœud du noyau. Un problème important est alors de savoir quel est la relation entre ϵ et q et amène la question : Dans quelle mesure, augmenter q diminue-t'il ϵ ?

Cette relation est décrite sur la Figure 7 pour un système de $n = 10^4$ nœuds. Chaque courbe correspond à un différent taux de nœuds ayant quitté le système. La courbe montre que lorsque $10^{-3} \leq \epsilon \leq 10^{-2}$, la probabilité $p = 1 - \epsilon$ croît rapidement vers 1. Par exemple, la courbe représentant $C = 10\%$ montre qu'un noyau de $q = 224$ (resp. $q = 274$) nœuds assure une probabilité d'intersection de $1 - \epsilon = 0.99$ (resp. 0.999).

On remarque, que le résultat obtenu est similaire au paradoxe de l'anniversaire qui est paradoxal au sens de l'intuition et non de la logique. Pour que la probabilité que deux personnes dans la même pièce soient nés le même jour (toute année confondue) soit plus grande que $1/2$, il suffit qu'il y ait 23 personnes dans cette pièce. Lorsqu'il y a 50 personnes dans la pièce la probabilité est de 97% et devient 99.9996% pour 100 personnes.

Dans notre cas, ce phénomène se traduit par une forte augmentation de la probabilité d'accéder à un nœud du noyau lorsque la taille du noyau est seulement légèrement augmentée. Ainsi le concepteur du système peut augmenter la probabilité de contacter un nœud du noyau au prix d'un faible surcoût.

4.3.2 Relation entre la taille du noyau q et la période δ

Précédemment, on a montré qu'une application doit fixé C pour pouvoir définir le nombre q de nœuds à contacter permettant d'obtenir une probabilité p de réussite. Il existe un dernier compromis qu'un concepteur d'application doit faire : en fonction d'une probabilité $p = 1-\epsilon$, il s'agit de décider de la taille q, au dépend de la période δ après laquelle les q nœuds sont contactés. Ainsi, il est nécessaire de comparer précisément q à δ pour une valeur ϵ fixée.

Nous mettons en relation la taille et la durée de vie d'un noyau lorsque la probabilité de contacter un nœud du noyau est de 99% ou $99,9\%$. Ces valeurs ont été choisies comme pouvant refléter les choix d'un concepteur d'applications. Pour chacune des deux probabilités, nous avons représenté, sur la Figure 8, la taille d'un quorum en fonction du nombre de nœuds ayant quitté le système durant la période δ. La Figure 8 met en valeur cette relation dans un système statique et dans un système dynamique (pour différentes valeurs de C). Le système statique implique qu'aucun nœud ne quitte ou ne rejoint le système durant la période δ alors que dans le système dynamique une certaine portion C des nœuds quitte et rejoint le système durant cette période δ. Dans un souci de simplicité de présenta-

Probabilité d'intersection	Va-et-vient $C = 1 - (1 - c)^{\delta}$	Taille du noyau $q(n = 10^3)$	$q(n = 10^4)$	$q(n = 10^5)$
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

FIG. 8 – La taille du noyau en fonction de la taille du système et du taux de va-et-vient.

tion, nous avons représenter C plutôt que δ. Les résultats obtenus mènent à deux observations intéressantes.

D'une part, lorsque δ est assez grand pour que 10% des nœuds soient remplacés, alors la taille du noyau nécessaire est étonamment proche de celle du cas statique (873 contre 828 lorsque $n = 10^5$ pour une probabilité de 0,999). De plus, $q = 990$ est suffisant lorsque C passe à 30%. D'autre part, lorsque la période δ est suffisamment large pour que 80% du système soit remplacé, le nombre de nœuds q à contacter reste bas comparé à la taille du système. Par exemple, dans le cas ou 6 000 nœuds sur 10 000 sont remplacés, alors seulement 413 nœuds doivent être contactés pour obtenir une intersection avec probabilité 0,999.

4.4 Conclusion

Maintenir une donnée critique dans un système ou les nœuds partent et arrivent dynamiquement est un problème difficile. Dans cet section, nous avons défini la notion de noyau persistant permettant de maintenir la donnée indépendamment de la structure sous-jacente utilisée.

Nos résultats apportent une aide aux concepteurs d'application pour ajuster les paramètres en fonction du dynamisme et de la garantie recherchée. Un de nos

résultats montre qu'augmenter légèrement la taille du noyau suffit à augmenter la garantie de beaucoup, même avec un fort dynamisme. Ce travail ouvre la voie à de nouvelles recherches tant dans le domaine de la cohérence mémoire que l'évaluation de l'intensité du va-et-vient.

De façon plus générale, la persistence des données est une brique de base pour la résolution de mémoire partagée distribuée. Lorsque la donnée persiste au fil du temps, elle reste lisible par l'ensemble des clients. Tant que cette donnée est lisible les opérations de lecture et d'écriture sont exécutables assurant la disponibilité de la mémoire. Nous avons présenté la résolution du problème de persistence des données car cette résolution nous a amené à définir la mémoire partagée distribuée tolérant à la fois le dynamisme et le passage à l'échelle présentée dans le Chapitre 4 de la thèse.

5 Conclusion

Cette thèse met l'accent sur le fait que les systèmes dynamiques à grande échelle sont très complexes : Premièrement, dans un tel contexte il est impossible d'émuler efficacement une mémoire partagée distribuée (MPD) qui vérifie des critères déterministes et qui tolère le dynamisme et le passage à l'échelle. Deuxièmement, les MPD probabilistes sont faites pour ce genre d'environement.

Mauvaise nouvelle ?

Cette thèse identifie un important compromis qui empêche d'émuler efficacement une MPD déterministe qui tolère le dynamisme et qui passe à l'échelle. Soit tolérer le dynamisme nécessite beaucoup de messages, ou bien assurer le passage à l'échelle nécessite une grande latence des opérations. Comme exemple de ces deux extrêmes, deux solutions ont été présentées :

1. L'algorithme RDS présenté au Chapitre 4 tolère le dynamisme en permettant de décider rapidement de la configuration de quorums remplaçante. Cet algorithme exécute efficacement les opérations qui peuvent ne durer que deux délais de transmission. Cependant, cette solution nécessite que le degré du graphe de communication soit grand pour que les opérations s'effectuent rapidement. Dans un grand système, chacun des événements

dynamiques impliquent de mettre à jour l'état d'un grand nombre de nœuds en échangeant un trop grand nombre de messages.

2. Dans le Chapitre 3, Square fournit une mémoire qui tolère les requêtes imprévisibles d'un grand nombre de clients. La mémoire auto-adapte ses ressources en fonction des variation de la charge. La reconfiguration n'est pas très couteuse puisque le degré du graphe de communication est petit. Il en résulte que l'accès à un quorum requiert de nombreux messages successifs et exécuter une opération peut être très long. Ce phénomène est renforcé lorsque la charge s'accroît et que la mémoire s'agrandit, menant à des quorums plus grands et donc une latence des opérations plus importante.

Classification de systèmes de quorum

Dans le but de trouver le meilleur système de quorum pour une MPD dans un environement à grande échelle et dynamique, cette thèse a étudié en détail les systèmes de quorums existant. Il en résulte une nouvelle classification des systèmes de quorums ainsi qu'elle apparait sur la figure 9.

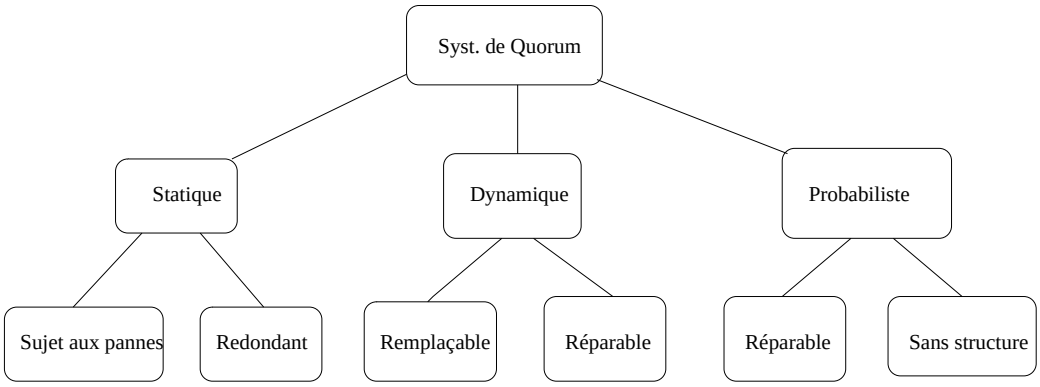

FIG. 9 – Classification de systèmes de quorum.

Cette classification donne un rangement assez naturel des systèmes de quorums représentant leur faculté d'adaptation à un système dynamique à grande échelle. Premièrement, tout à gauche apparaissent les systèmes de quorums qui

peuvent tomber en pannes lorsqu'un de leurs éléments tombe en panne (le système de quorum en étoile en est un exemple). À leur droite apparait les systèmes de quorums redondants qui tolère un nombre borné de pannes mais qui ne peuvent pas non plus tolérer le dynamisme (citons par exemple le système de quorum en grille [Mae85]). Puis, les systèmes de quorum reconfigurables peuvent tolérer un nombre arbitraire de pannes alors qu'ils ne passent pas à l'échelle (par exemple RDS [CGG+05]). Les systèmes de quorum réparables qui eux tolèrent un grand nombre de pannes sans surcout en communication ne fournissent pas un service optimale (e.g. square [GAV07]). Finalement, les systèmes de quorum sans structure (ainsi que les systèmes de quorum temporisés [GKM+06]) apparaissent comme étant la solution la plus prometteuse tolérant à la fois le dynamisme et le passage à l'échelle.

En fait, la classification est la suivante. Premièrement, les systèmes de quorum sujets aux pannes et les quorums systèmes redondants sont statiques. Deuxièment, les systèmes de quorums reconfigurable et remplaçable sont tous deux dynamiques. Troisièmement, certains systèmes de quorums remplaçables et sans structure sont probabilistes. La distinction principale se fait entre les systèmes de quorums déterministes (les systèmes de quorum statiques et dynamiques) et les systèmes de quorum probabilistes puisque ces deux classes de systèmes de quorum offrent des garanties complètement différentes. Une piste de recherche intéressante serait de définir des quorums sans structure assurant des propriétés déterministes. Ce type de systèmes de quorum tolèrerait donc un fort dynamisme tout en assurant des garanties fortes.

En conclusion de cette classification, nous sommes convaincus que les systèmes de quorums sans structure peuvent être adaptés pour différents besoins. Afin de résoudre des problèmes d'exclusion mutuelle [Ray86], de consensus [Lam06], de recherche de données [MTK06], ou de proposer d'autres applications basées sur les quorums, les chercheurs pourraient tirer partie des propriétés de passage de l'échelle et de tolérance aux dynamismes de tels systèmes de quorum.

Bonne nouvelle ?

Cette thèse dit que la cohérence probabiliste donne des moyens de définir des conditions acceptables tout en permettant des mémoires avec des performances

idéales. Ceci est essentiellement un contrecoup du compromis précédemment mentioné qui détériore les performances lors de l'émulation d'une MPD déterministe : soit la congestion peut provoquer des pertes de requêtes importantes, soit les opérations sont lentes. Il y a deux arguments majeures en faveur de la cohérence probabiliste :

1. Le premier argument est que les systèmes dynamiques à grande échelle ne peuvent être modélisés avec des nœuds dont les comportements sont dépendants les uns des autres. En effet, il n'est pas raisonnable de penser à un systèmes dynamique dans lequel beaucoup de participants agissent ensemble de telle façon que durant une courte période de temps, seulement un petit nombre de nœuds quittent le système. Si le système est très grand, les participants potentiels ont tendance à agir d'autant plus indépendamment et quittent ou rejoignent à des temps arbitraires. Au contraire donc, beaucoup de participants peuvent partir ou rejoindre le système en même temps, même si cela reste peu probable. Il est donc bien plus réaliste de considérer que chaque nœud agit indépendamment et qu'il existe une petite probabilité pour que certains nœuds quittent le système au même moment. Dans ce cas, les garanties qui peuvent être assurées le sont avec une probabilité associée. Le Chapitre 5 a présenté la cohérence atomique probabiliste comme un critère de cohérence prometteur qui permet à toute opération d'être atomique avec une grande probabilité.
2. Le second argument est qu'il existe des implémentations de systèmes de quorum temporisés (SQT), avec des critères probabilistes, qui dépassent les performances des solutions déterministes. L'implémentation de SQT presentée au Chapitre 5 atteint des opérations plus rapides que celles de la solution du Chapitre 4, Square. Plus généralement, toute solution qui réduit la latence des opérations de Square en l'adaptant avec un degré plus grand dans le graphe de communication nécessite une reconfiguration plus couteuse, alors que ce méchanisme est absent des SQT. De plus, SQT améliore les performance de RDS en utilisant un degré constant au niveau du graphe de communication et en mettant le rôle de la reconfiguration au niveau des opérations. Ainsi chaque nœud communique seulement avec un nombre constant de voisins et aucune reconfiguration coûteuse n'est nécessaire.

Finalement, vers une application effective de MPD dans un environement à grande échelle dynamique, le Chapitre 5 a aussi mis en valeur que la notion de probabilité peut être traduite en terme de qualité de service exploitable par les développeurs d'applications.

Références

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

[ACLS94] D. Agrawal, M. Choy, H.V. Leong, and A. Singh. Mixed consistency : a model for parallel programming. In *Proc. 13th ACM Symposium on Principles of Dist. Computing*, pages 101–110, 1994.

[AF92] Hagit Attiya and Roy Friedman. A correctness condition for high-performance multiprocessors (extended abstract). In *STOC '92 : Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 679–690, New York, NY, USA, 1992. ACM Press.

[AGGV05] Emmanuelle Anceaume, Maria Gradinariu, Vincent Gramoli, and Antonino Virgillito. P2P architecture for self* atomic memory. In *Proceedings of the 8th International Symposium on Parallel Architectures, Algorithms and Networks*, pages 214–219. ISCA, Dec. 2005.

[AM05] Ittai Abraham and Dahlia Malkhi. Probabilistic quorum systems for dynamic systems. *Distributed Computing*, 18(2) :113–124, 2005.

[CGG+05] Gregory Chockler, Seth Gilbert, Vincent Gramoli, Peter Musial, and Alexander Shvartsman. Reconfigurable distributed storage for dynamic networks. In *Proceedings of 9th International Conference on Principles of Distributed Systems*, pages 214–219, December 2005.

[DGLC04] Partha Dutta, Rachid Guerraoui, Ron R. Levy, and Arindam Chakraborty. How fast can a distributed atomic read be ? In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing (PODC'04)*, pages 236–245, New York, NY, USA, 2004. ACM Press.

[DSB86] M. Dubois, C. Scheurich, and F. Briggs. Memory access buffering in multiprocessors. In *ISCA '86 : Proceedings of the 13th annual international symposium on Computer architecture*, pages 434–442, Los Alamitos, CA, USA, 1986. IEEE Computer Society Press.

[ES00] B. Englert and A.A. Shvartsman. Graceful quorum reconfiguration in a robust emulation of shared memory. In *Proc. of International Conference on Distributed Computer Systems*, pages 454–463, 2000.

[GAV07] Vincent Gramoli, Emmanuelle Anceaume, and Antonino Virgillito. Square : Scalable quorum-based atomic memory with local reconfiguration. In *Proceedings of the 22nd ACM Symposium on Applied Computing (SAC'07)*, pages 574–579. ACM Press, mar 2007.

[Gif79] David K. Gifford. Weighted voting for replicated data. In *Proceedings of the seventh ACM symposium on Operating systems principles*, pages 150–162. ACM Press, 1979.

[GKM+06] Vincent Gramoli, Anne-Marie Kermarrec, Achour Mostefaoui, Michel Raynal, and Bruno Sericola. Core persistence in peer-to-peer systems : Relating size to lifetime. In *Proceedings of the On-The-Move International Workshop on Reliability in Decentralized Distributed Systems*, volume 4278 of *LNCS*, pages 1470–1479. Springer, Oct. 2006.

[GLS03] S. Gilbert, N. Lynch, and A.A. Shvartsman. RAMBO II : Implementing atomic memory in dynamic networks, using an aggressive reconfiguration strategy. Technical report, LCS, MIT, 2003.

[GMB85] Hector Garcia-Molina and Daniel Barbara. How to assign votes in a distributed system. *J. ACM*, 32(4) :841–860, 1985.

[Goo89] James R. Goodman. Cache consistency and sequential consistency. Technical Report 61, SCI Committee, March 1989.

[HA90] P. W. Hutto and M. Ahamad. Slow memory : Weakening consistency to enhance concurrency in distributed shared memories. In *Proc. of the 10th Int'l Conf. on Distributed Computing Systems (ICDCS)*, pages 302–311, May 1990.

[Her86] Maurice Herlihy. A quorum-consensus replication method for abstract data types. *ACM Trans. Comput. Syst.*, 4(1) :32–53, 1986.

[HW90] 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.

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

[Lam89] L. Lamport. The part-time parliament. Technical Report 49, Digital SRC, September 1989.

[Lam06] L. Lamport. Fast paxos. *Distributed Computing*, 19(2) :79–103, Oct. 2006.

[LS88] R. J. Lipton and J. S. Sandberg. Pram : A scalable shared memory. Technical Report CS-TR-180-88, Dept. of Computer Science, Princeton University, September 1988.

[LS97] N. Lynch and A.A. Shvartsman. Robust emulation of shared memory using dynamic quorum-acknowledged broadcasts. In *Proc. of 27th Int-l Symp. on Fault-Tolerant Comp.*, pages 272–281, 1997.

[LS02] N. Lynch and A.A. Shvartsman. RAMBO : A reconfigurable atomic memory service for dynamic networks. In *Proc. of 16th International Symposium on Distributed Computing*, pages 173–190, 2002.

[Lyn96] N.A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.

[MA04] Jean-Philippe Martin and Lorenzo Alvisi. A framework for dynamic byzantine storage. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN)*, page 325. IEEE Computer Society, 2004.

[Mae85] Mamoru Maekawa. A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. *ACM Trans. Comput. Syst.*, 3(2) :145–159, 1985.

[MTK06] Ken Miura, Taro Tagawa, and Hirotsugu Kakugawa. A quorum-based protocol for searching objects in peer-to-peer networks. *IEEE Transactions on Parallel and distributed Systems*, 17(1), January 2006.

[NN05] U. Nadav and M. Naor. The dynamic and-or quorum system. In Pierre Fraigniaud, editor, *Distributed algorithms*, volume 3724

of *Lecture Notes In Computer Science*, pages 472–486, September 2005.

[NW03] M. Naor and U. Wieder. Scalable and dynamic quorum systems. In *Proceedings of the 22th annual symposium on Principles of distributed computing (PODC'03)*, pages 114–122. ACM Press, 2003.

[Ray86] M. Raynal. *Algorithms for mutual exclusion.* MIT Press, Cambridge, MA, USA, 1986.

[RD01] Antony Rowstron and Peter Druschel. Pastry : Scalable, distributed object location and routing for large-scale peer-to-peer systems. In Rachid Guerraoui, editor, *Middleware 2001*, volume 2218 of *LNCS*, pages 329–350. Springer-Verlag, 2001.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of the ACM SIGCOMM*, pages 161–172, 2001.

[Tho79] Robert H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *ACM Trans. Database Syst.*, 4(2) :180–209, 1979.

[Vid96] K. Vidyasankar. Weak atomicity : A helpful notion in the construction of atomic shared variables. *SADHANA : Journal of Engineering Sciences of the Indian Academy of Sciences 21*, pages 245–259, 1996.