Table des matières

Table des figures

Liste des tableaux

Introduction générale

Ces dernières années ont été marquées par une forte évolution des équipements utilisés dans les environnements répartis, notamment dans les technologies des ordinateurs et surtout dans les réseaux à haut débit connectant ces derniers. Nous sommes successivement passés de réseaux locaux à des réseaux à grande échelle (Internet). Cette évolution a abouti à la définition d'une nouvelle technologie, qui se base sur l'infrastructure des grilles.

A travers ces réseaux, des simulations et des expérimentations dans des domaines tels que l'énergie atomique, l'astronomie, la météorologie ou la génomique sont de plus en plus gourmandes en puissance de calcul et génèrent des énormes masses de données à partager (traitement, téléchargement, stockage) par une communauté qui ne cesse d'agrandir. Par exemple, le Centre Européen de la Recherche Nucléaire (CERN[1]) est en train de générer plusieurs Petaoctets de données par an. Plusieurs milliers de physiciens à travers le monde auront besoin d'accéder à ces données pour exécuter beaucoup de calculs. Les grilles permettront de répondre à ces besoins en distribuant les calculs et en partageant les données entre plusieurs sites qui sont reliés par de puissants réseaux. Le domaine des grilles devient très actif et porteur car il existe de nos jours une énorme demande de capacité de calcul, et de stockage. Cependant, il reste encore beaucoup de problèmes à résoudre, tels que les problèmes d'accès équitable aux ressources et de leur partage, d'ordonnancement, de sécurité et de gestion des données partagées.

[1] CERN website : http://wlcg.web.cern.ch.

La réplication est une des techniques souvent utilisée pour résoudre ces problèmes liés à la gestion des données. Elle consiste à créer de multiples copies de la même donnée dans plusieurs ressources de stockage. Le problème de réplication est un problème relativement connu dans les systèmes répartis, mais il pose de nouvelles problématiques dans les environnements à large échelle. Dans la grille de données, où les tailles des données s'élèvent aux Petaoctets et la communauté des utilisateurs est dans l'ordre des milliers (voire des millions) dans le monde entier, la réplication statique ne semble pas appropriée. Un tel système a besoin des stratégies de réplication dynamiques (où la création, la suppression et la gestion des répliques sont effectués automatiquement) qui ont la capacité de s'adapter aux changements dans le comportement des utilisateurs.
Les questions fondamentales auxquelles n'importe quelle stratégie de réplication doit répondre sont :

1. Quand de nouvelles répliques doivent être créées ?
2. Où les répliques doivent être placées ?
3. Quelles données doivent être reproduites ?
4. Choisir la meilleure copie à répliquer (à transférer).
5. Quel est le nombre de répliques dans le système.
6. Comment répliquer ?

Les réponses à ces questions nous mènent vers différentes stratégies de réplication.

Plusieurs stratégies de réplication ont été proposées dans la littérature [Belalem, 07a] [Shorfuzzaman, 10] [Tang, 05] [Xiong, 13].
Chacune d'elles offre des solutions pour répondre aux questions fondamentales de la réplication.

Problématique

Répartir efficacement les traitements et les données sur différents nœuds (ou sites), afin de minimiser les temps de calculs et les temps de transfert ou de téléchargement des données, est un problème critique et complexe. Il est nécessaire de trouver des moyens pour stocker et rendre rapidement accessible d'énormes masses de données souvent demandées. Une solution simple mais mauvaise, consiste à conserver les données à la source, qui risque de devenir rapidement un goulot d'étranglement suite à des téléchargements de données massives et populaires. Les données doivent être répliquées et réparties pour éviter ces goulots d'étranglement et augmenter leur disponibilité. Par exemple, les données sont réparties sur des tiers : des serveurs accessibles à travers des réseaux à haut débit.
Le problème de la répartition dans un cadre statique est déjà un problème difficile. En plus, la grille est utilisée dans un environnement dynamique (débit du réseau variable, accès aux données aléatoire, etc.). La problématique qui se pose est de trouver quelles sont les politiques de réplication et de répartition de données qui soient efficaces dans un tel environnement.

La réplication des données, est largement utilisée dans les grilles de données pour répondre aux exigences et aux défis de la gestion des énormes quantités de données. En effet, elle est utilisée pour améliorer les performances et augmenter la

disponibilité. Pour atteindre ces objectifs, la sollicitation d'une bonne stratégie de réplication de données est nécessaire, pour une gestion efficace des répliques.
Les stratégies de réplication pour les grilles de données est un domaine de recherche actif [Andronikou, 12] [Chervenak, 08] [Elghirani, 09] [Nukarapu, 11]. Un point commun entre ces stratégies est que le plus souvent elles ne considèrent que les cas de lectures seules pour accéder aux données. Par conséquent, ces approches ne sont pas bien adaptées pour les applications où les répliques des données peuvent être modifiées [Perez, 10]. Peu de techniques de réplication de données se sont focalisées sur les cas des mises à jour des répliques dans les grilles de données [Grace, 14]. Et peu d'études ont été faites pour maintenir la cohérence des répliques dans les grilles de données [Mamat, 08] [Senhadji, 13].

Pour avoir une bonne stratégie de réplication, il faut prendre en considération toutes les requêtes d'accès aux données, y compris les requêtes d'écriture sur les données. Donc le problème revient à trouver une politique efficace de gestion de répliques, qui tient en compte la cohérence des répliques.
Etant donné que l'environnement des grilles est très complexe, il est très difficile de prendre des décisions sur la gestion des répliques, notamment sur le moment où de nouvelles répliques doivent être créées, sur le nombre de répliques nécessaires, sur les endroits où les répliques doivent être placées, etc. Par conséquent, si nous voulons améliorer la disponibilité et les performances au sein de la grille, il faut prendre les bonnes décisions sur la gestion des répliques. Or, les systèmes classiques, qui sont passifs, ne permettent pas de réagir pour prendre des bonnes décisions et répondre aux exigences de dynamicité de la grille. Pour surmonter cette limite, il faut donc trouver les moyens qui permettent d'être réactif pour répondre aux changements de la grille et prendre les bonnes décisions afin que la gestion des répliques soit efficace. L'utilisation des agents permettent de satisfaire de tels besoins.

Contribution

Nos travaux de recherche portent sur la proposition et la mise en œuvre d'une stratégie de réplication dynamique, qui tient compte des caractéristiques des grilles de données. A travers ces travaux de recherche, nous essayons d'atteindre deux principaux objectifs :

1. Prendre en compte la cohérence des répliques lors de leur gestion (création, placement, etc.).
2. Intégrer et utiliser les agents dans le système pour l'aider à prendre les décisions sur la gestion des répliques.

Les systèmes distribués et la réplication des données sont reconnus aujourd'hui comme moyens efficaces pour augmenter la disponibilité et la fiabilité des données. En effet la réplication peut contribuer favorablement à l'amélioration des performances, toutefois, ces avantages sont contraints par un problème majeur de cohérence mutuelle des copies. La gestion des copies en terme de propagation des mises à jour est ainsi nécessaire.

La réponse à cette préoccupation constitue le premier objectif dans notre contribution. Pour gérer les différentes répliques d'une donnée, que ce soit pour la création de nouvelles répliques, ou pour le déplacement ou pour la suppression des répliques existantes, nous prendrons toujours en considération la cohérence mutuelle de ces répliques. Car le coût de maintenance de la cohérence des répliques peut s'élever à cause du nombre et du placement des répliques au sein de la grille.
D'autre part, vu que les grilles sont caractérisées par une forte dynamicité (les nœuds peuvent rejoindre et quitter la grille à tout moment), il s'avère très difficile de prendre des décisions sur la gestion des répliques, surtout dans un système passif, où il faut réagir rapidement pour créer, déplacer, ou supprimer les répliques. Par conséquent, le deuxième objectif de notre contribution vise à faire appel aux apports des systèmes multi-agents pour remédier aux problèmes de passivité et améliorer les performances.
Dans ce contexte nous proposons dans cette thèse, une stratégie de réplication dynamique dans les grilles de données basée sur les quorums et sur un système multi-agents. Nous avons choisi la plate-forme JADE[2] pour le développement du système multi-agents.

Pour évaluer notre stratégie, nous avons utilisé la simulation. Elle a l'avantage d'être souple bien qu'elle nécessite plus de calculs. Nous avons opté pour cette technique, pour étudier le comportement de notre approche et la comparer avec d'autres approches existantes.
Nous avons évalué notre stratégie en utilisant deux simulateurs sur lesquelles nous avons intégré nos algorithmes et la plate-forme des systèmes multi-agents JADE. Le premier est notre simulateur de grille de données étendu et amélioré [Belalem, 07a] [Bouharaoua, 07], qui permet de simuler n'importe quelle structure de grille de données de type hiérarchique et plate à la fois (hybride). Le deuxième simulateur que nous avons utilisé est Optorsim[3] [Bell, 03b] [Cameron, 06].

Organisation du manuscrit

La maîtrise de la réplication des données, et des systèmes multi-agents dans les systèmes répartis et particulièrement dans les grilles, et leur compréhension passe en effet par la connaissance de leurs concepts, de leurs stratégies, de leurs algorithmes et de leurs méthodes spécifiques.
C'est pour cela, et dans le contexte de ce travail, cette thèse est synthétisée dans ce manuscrit composé de cinq chapitres :

- Tout d'abord, après la description du contexte, de la problématique et de la contribution dans cette introduction, nous présenterons dans le premier chapitre un état de l'art sur les grilles et sur les systèmes distribués à grandes échelles. Nous présenterons les concepts, les caractéristiques et des exemples sur les grappes de calculateurs (clusters), les systèmes pair-à-pair (P2P), et les grilles informatiques.

[2] JADE website : http://jade.tilab.com.
[3] Optorsim website : http://sourceforge.net/projects/optorsim.

- Le second chapitre sera consacré à un état de l'art sur les systèmes multi-agents. Nous présenterons les notions et les principaux concepts d'agents et de systèmes multi-agents, issus de l'intelligence artificielle distribuée.

- Dans le troisième chapitre, nous présenterons une vue globale sur les notions et les concepts de la réplication et de la cohérence dans les grilles de données. Nous aborderons le sujet dans sa globalité en présentant des techniques et des concepts sur la cohérence et la réplication à grande échelle. Nous présenterons aussi les concepts sur les systèmes de quorums suivis de quelques approches proposées dans des travaux connexes sur la réplication, sur les quorums et sur l'utilisation des agents dans les grilles.

- Le quatrième chapitre sera consacré à notre contribution, où notre stratégie de réplication dynamique basée sur un système multi-agents sera abordée en détail.

- Le cinquième chapitre sera destiné à la discussion de notre approche après l'évaluation de ses performances, où les résultats des expérimentations obtenus par simulations seront présentés et analysés.

Enfin nous terminerons cette thèse par une conclusion dans laquelle nous résumerons l'essentiel de notre travail et nous proposerons quelques perspectives pour des futurs travaux dans ce domaine de recherche.

Chapitre I.
Les grilles informatiques

1. Introduction

Tout a débuté lorsque des chercheurs ont commencé à exploiter les ressources de plusieurs machines inutilisés et qu'ils se sont aperçus que faire coupler tous ces éléments leur faisaient gagner du temps et de l'argent par rapport à l'achat d'un supercalculateur.
Par la suite, ce genre d'expérience a pu être renouvelé en utilisant le réseau Internet et ainsi utiliser les ressources offertes par des machines géographiquement très éloignées.

Nous pourrions se poser la question : pourquoi les technologies des systèmes distribués arrivent soudainement dans notre environnement actuel ? La réponse est très simple : les réseaux évoluent bien plus vite que les solutions de stockages ou la puissance des processeurs. La formidable expansion de la capacité des réseaux a donc permis aux systèmes distribués (grille de calcul, pair à pair, etc.) d'ouvrir ses ailes. Les nouvelles technologies de l'Internet permettent une connectivité universelle sans limite en facilitant le travail en équipe. Grâce aux réseaux actuels, les frontières n'existent plus, les visions de la géographie en sont totalement bouleversées.
Plus encore, le besoin de distribuer les informations à diverses équipes disséminées sur le globe a conduit les chercheurs en informatique à sentir l'utilité de pouvoir gérer à distance, d'avoir des infrastructures distribuées, ainsi que divers services et applications, et c'est là qu'interviennent les systèmes distribués.

Dans ce qui suit, nous présentons trois catégories de systèmes distribués tributaire de leur caractère hétérogène et leur besoin de passage à l'échelle : les grappes de calculateurs (clusters), les systèmes pair-à-pair (P2P), et les grilles informatiques.

2. Les grappes de calculateurs (clusters)

Les grappes de calculateurs (les clusters) représentent une alternative aux multiprocesseurs fortement couplés. L'an 1995, grâce au projet Beowulf de la NASA que le concept de cluster a vu le jour [Salmon, 98]. Un des objectifs de ce projet était le traitement des énormes volumes de données utilisées dans les sciences de l'espace et de la terre par l'exploitation des ordinateurs massivement parallèles. Le premier cluster a été conçu et construit pour étudier des problèmes liés aux grandes quantités de données liées aux calculs sismiques. Ces nouvelles solutions de clusters fournissent des plates-formes très performantes et très fiables avec des prix relativement faibles par rapport à ceux des supercalculateurs, ce qui permet aux utilisateurs d'en bénéficier. Les clusters sont composés d'un ensemble d'ordinateurs (serveurs, PC, etc.) interconnectés entre eux par un réseau. Ils permettent de répondre aux problématiques de haute performance et de haute disponibilité. Leurs utilisation, été un succès pour les moteurs de recherche sur Internet (comme Google par exemple). L'exploitation du parallélisme est devenue plus facile puisque les applications traitées par les grappes sont typiquement de lectures intensives.

2.1. Définition des grappes de calculateurs

Une grappe de calculateurs (cluster) peut être définie comme un ensemble de machines (appelés nœuds) connectés par un réseau fiable et rapide afin de répondre aux problématiques de performance et de disponibilité.
Les clusters nous permettent de répartir les traitements sur différentes machines grâce à l'utilisation optimisée des ressources. Ces ressources qui sont généralement quasi-homogènes (quasi-similarité des systèmes d'exploitation, logiciels, etc.) sont contrôlées par un seul nœud (nœud maître).
Parmi les avantages d'un cluster est son coût de mise en œuvre qui est relativement abordable, ce qui nous permet de ne pas acheter un multiprocesseur qui est très coûteux. En effet, le cluster est vu comme une technologie qui réside dans la mise en place de plusieurs machines interconnectés via un réseau et qui apparaissent comme une seule machine possédant plus de capacités.

2.2. Exemples de clusters

- **Beowulf**

La structure de Beowulf est basée sur une architecture multi-ordinateur utilisée pour la programmation parallèle [Salmon, 98]. C'est une structure homogène dont les composants sont des ordinateurs personnels (PC) bon marché. Elle est composée de plusieurs machines connectées entre elles à travers un réseau. Le cluster Beowulf ne contient aucun composant matériel ou logiciel particuliers (qui lui sont propre). Il est construit en utilisant les composants matériels et logiciels existants (PCs, adaptateurs réseaux standards, système d'exploitation Linux ou tout autre systèmes d'exploitation libre, etc.). Il utilise aussi des bibliothèques telles que PVM (Parallel Virtual Machine) et MPI (Message Passing Interface).

- **MOSIX**

Le cluster MOSIX (Multicomputer Operating System for unIX) est destiné pour l'optimisation des performances du système en équilibrant la charge entre les différentes machines qui le composent [Barak, 98]. Cet équilibrage de charge est réalisé par la migration des processus entre les différents nœuds appartenant au cluster. En effet, MOSIX dispose de mécanismes qui permettent de détecter les différences de charge entre les nœuds et de provoquer les migrations des processus ou des tâches depuis les nœuds fortement chargés vers les nœuds les moins chargés. La migration des processus participe à l'amélioration des performances puisqu'elle peut tirer profit de la meilleure ressource disponible.

3. Système pair à pair (P2P)

Les premiers systèmes pair à pair (P2P : Peer To Peer) n'ont attiré plus d'intérêt qu'avec la naissance de Napster [Saroiu, 03]. En effet, Le premier système pair à pair a vu le jour en 1999 avec l'apparition de la plate-forme Napster dédiée au partage de la musique en format mp3.

De nos jours, l'intérêt des systèmes pair à pair ne cesse de croître. Ils sont utilisés dans de nombreux domaines. Le modèle pair à pair est considéré comme l'un des modèles les plus importants dans les systèmes distribués.
Un système pair à pair représente tout système basé sur un modèle réseau d'égale à égale où tous les nœuds (pairs) ont en général des rôles symétriques, c'est à dire qu'ils jouent à la fois le rôle de demandeur et d'offreur de services (rôle de client et de serveur) (figure I.1).

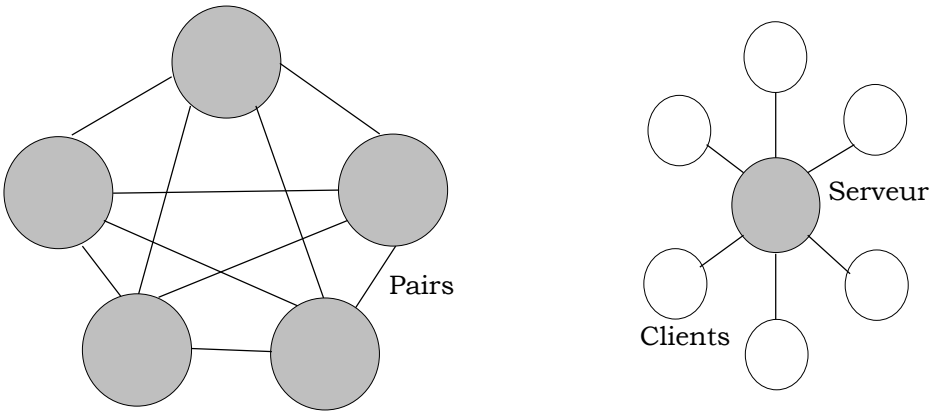

Figure I.1 : Système pair à pair et système client-serveur.

3.1. Définition d'un système pair à pair

Différentes définitions du système pair à pair existent en littérature. Ces définitions ne sont pas parfaitement équivalentes.
Le système pair à pair utilise les ressources d'un système distribué afin d'exécuter de façon décentralisée des tâches particulières sans passer obligatoirement par une entité centrale. Ces ressources peuvent être composées des unités de calcul, des supports de stockage, d'un réseau, etc. Les tâches à exécuter peuvent être des calculs distribués, du partage de données, de la communication et de la collaboration, etc. quant à la décentralisation, elle peut être appliquée soit aux algorithmes, soit aux données, soit aux métadonnées, etc.

3.2. Caractéristiques d'un système pair à pair

Un système pair à pair se distingue par plusieurs caractéristiques [Doyen, 05].

- **Passage à l'échelle**

  Le passage à l'échelle est souvent considéré comme principal intérêt du système pair à pair. L'absence d'une infrastructure fixe dans les systèmes pair à pair facilite leur passage à l'échelle. Les caractéristiques propres au système pair à pair lui garantissent un fonctionnement à large échelle [Oram, 03].

- **Tolérance aux fautes**

  Une des caractéristiques importantes dans les réseaux pair à pair est la tolérance aux fautes. Quand un nœud (pair) défaille dans un modèle pair à pair, la disponibilité du service doit être assurée par les nœuds (pairs) restants.

- **Dynamicité**

  Une autre caractéristique des réseaux pair à pair est la volatilité des nœuds. En effet, les pairs peuvent avoir un comportement dynamique et peuvent se connecter ou se déconnecter librement et à n'importe quel moment [Ben Charrada, 10] [Bonnel, 08]

- **Auto-Organisation**

  Une des caractéristiques importantes dans les réseaux pair à pair est qu'ils peuvent s'auto-organiser. Un système pair à pair est un système dynamique à grande échelle qui nécessite des mécanismes d'auto-organisation afin de permettre aux pairs de fonctionner en mode décentralisé [Ben Charrada, 10]

Le système pair à pair est devenu incontournable dans le domaine de la recherche d'information de la collaboration et du partage de fichiers. Certaines de ces caractéristiques peuvent être considérées comme des avantages offerts par les systèmes pair à pair. En plus de l'accès offert aux grands nombres de ressources dont il dispose, l'administration d'un système pair à pair est transparente et elle est assurée sans avoir recours aux machines dédiées. Aucun des pairs n'est indispensable au fonctionnement du système (c.à.d. la défaillance d'un ou plusieurs pairs n'implique pas la défaillance du système) [Soyez, 05]. En contrepartie, des problèmes de sécurité peuvent être compromis de la part de certains pairs malveillants, et la confidentialité des données échangées peut ne pas être assurée. De plus, le système peut devenir non fiable à cause du comportement volatile des pairs.

3.3. Exemples des systèmes pair à pair

- **Napster[4] [Oram, 03]**

Napster est une application dédiée au partage de fichiers musicaux en format mp3. Elle a été créée en 1999, et elle est considérée comme la première application pair à pair. Napster est basée sur un modèle pair à pair centralisé, où un serveur est utilisé pour stocker les index des fichiers musicaux. Les différents pairs communiquent avec le serveur pour consulter les index des fichiers partagés pour pouvoir ensuite les télécharger.

- **Gnutella [Hughes, 05]**

Gnutella a été créé en 2000. Il est basé sur un modèle pair à pair décentralisé et non structuré. La recherche des fichiers dans Gnutella est complètement décentralisée et ne passe pas par un serveur centralisé. En effet, chaque pair lance sa recherche par propagation de messages (requêtes) aux voisins. Et chaque pair index localement ces propres fichiers pour ensuite les partager avec les autres pairs.

[4] Napster website : https://www.napster.com.

- **Chord [Stoica, 01]**

Chord a été présenté dans le cadre des travaux sur le pair à pair en 2001. Il est basé sur un modèle pair à pair décentralisé et structuré. Dans Chord, les pairs sont organisés selon une topologie en anneau. Cette organisation est basée sur une table de hachage distribuée (DHT : Distributed Hash Table). Chord a été utilisé pour un système de fichiers distribué à l'échelle de l'Internet qui est CFS (Collaborative File System) [Dabek, 01]. Et il a été utilisé pour un service du nom de domaine DNS (Domain Name Service) dans un réseau pair à pair [Cox, 02].

- **Kazaa[5]**

Kazaa est apparu en 2001. Il est basé sur un modèle super pairs qui est un modèle hybride entre le modèle pair à pair non structuré et le modèle client-serveur (voir la figure I.2). Il est comme le modèle client-serveur car certains pairs jouent le rôle des serveurs (super pairs). Et il est comme le modèle pair à pair car les super pairs sont organisés en mode pair à pair proprement dit. La désignation des super-pairs est faite selon certains critères tels que leur puissance de calcul, leur capacité de stockage, etc.

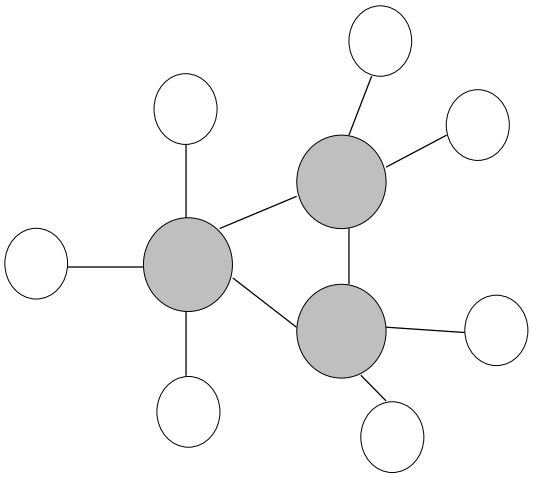

Figure I.2 : Topologie pair à pair hybride (modèle super-pair).

Pour certaines classifications, l'organisation des super-pairs n'est pas décentralisée. Elles considèrent que les modèles pair à pair hybrides (super-pair) ne diffèrent pas des modèles centralisés. Elles considèrent que le modèle centralisé est un cas particulier du modèle super pair et qui possède un seul super-pair.

4. Les grilles informatiques (grid computing)

Imaginez plusieurs millions d'ordinateurs, appartenant à des personnes et à des institutions différentes, soient connectés. Jusque-là, rien de nouveau, c'est le cas de l'Internet. Puis, imaginez que, d'une manière ou d'une autre, vous les faites tous fonctionner comme un ordinateur unique énorme et très puissant. Maintenant c'est

[5] Kazaa website : http ://www.kazza.com.

très différent ; cet extraordinaire et vaste fouillis d'ordinateurs est " la grille", telle que l'imaginent certains.

4.1. Définition d'une grille

Une brève définition de la grille, similaire à celle de la toile, est un ensemble de divers services et applications permettant de partager la puissance informatique, l'information et la capacité de mémoire. Mais la grille fait beaucoup plus qu'établir une simple communication entre les machines. Son objectif ultime est de transformer le réseau mondial des ordinateurs en une ressource immense et unique de capacité de traitement. C'est l'objectif, mais en réalité, la grille est aujourd'hui un "chantier", où des centaines de chercheurs dans le monde entier travaillent sur la mise au point de la technologie sous-jacente.

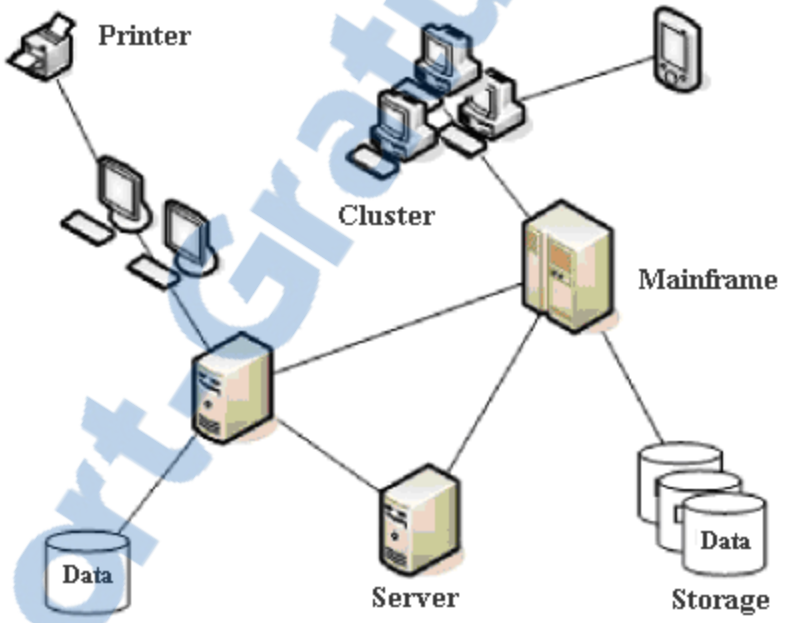

Figure I.3 : Exemple d'un réseau de ressources[6].

Le "grid-computing" consiste en un ensemble de ressources hétérogènes et distribuées, utilisé dans un réseau et vue comme une unique machine de très hautes performances (figure I.3), afin de réaliser rapidement des calculs complexes qui prendraient plusieurs mois voire plusieurs années avec des supercalculateurs classiques.
Cette mutualisation des ressources informatiques a pris le nom de grid-computing (grille de calcul) en référence au "power-grid" (réseau de distribution de l'électricité). Le grid-computing fonctionne en effet comme un réseau électrique dans la mesure où l'énergie électrique est fournie au consommateur sans que celui-ci se préoccupe de savoir où et par qui elle est produite.

[6] http://www-igm.univ-mlv.fr/~dr/XPOSE2006/Jolly_Laskri/grillescalcul.html.

4.2. Les raisons à déployer les grilles

Il est important de savoir quels avantages une grille est en mesure d'offrir que les infrastructures et les technologies actuelles ne sont pas capables d'assurer. Nous présentons par la suite quelques-unes des raisons pouvant amener à déployer une grille.

4.2.1. Exploiter les ressources sous utilisées

Les études montrent que les ordinateurs personnels et les stations de travail sont inactifs la plupart du temps. Le taux d'utilisation varie entre 30% (milieux académiques et industriels) et 5% (machines grand public). Les grilles de calcul permettront ainsi de récupérer les cycles (exploiter les cycles inutilisés) des processeurs afin de faire tourner une application nécessitant une puissance de calcul importante et que les machines qui lui sont dédiées n'arrivent pas à assurer. La conséquence d'une telle utilisation, est la possibilité de diviser les applications, requérant une grande capacité de traitement, en un grand nombre de tâches indépendantes les unes des autres, et ordonnancer ces dernières de manière à partager la charge entre les différentes ressources d'une grille.
Les cycles processeur ne sont pas la seule ressource sous utilisée; souvent les capacités de stockage le sont aussi. Ainsi il est possible qu'une grille agrège toutes ces ressources afin de les partager entre les différents utilisateurs (nous parlons alors de "grille de données" ou "data grid").

4.2.2. Fournir une importante capacité de calcul parallèle

A l'instar de tout système réparti, les grilles de calcul peuvent de la même manière fournir des ressources dont l'utilisation pourra se faire en parallèle. Des hautes performances de calcul résultent de la combinaison de ces puissantes ressources de calcul pour traiter des problèmes qui ne pourraient pas être résolus sur un système unique, ou au moins le faire beaucoup plus rapidement.

4.2.3. Meilleure utilisation de certaines ressources

En partageant les ressources, une grille pourra fournir l'accès à des ressources spéciales comme des équipements spécifiques (microscope électronique, bras robotique, capteurs, télescopes et d'autres appareils) ou des logiciels à prix élevé. Ainsi ces ressources partagées seront mieux utilisées par les utilisateurs, et nous éviterons d'avoir recours à installer du nouveau matériel ou acquérir de nouvelles licences des logiciels.

4.2.4. Fiabilité et disponibilité des services

Du fait que les ressources fédérées par une grille de calcul soient géographiquement dispersées et disponibles en importantes quantités permet d'assurer la continuité du service si certaines ressources deviennent inaccessibles. Les logiciels de contrôle et de gestion de la grille seront en mesure de soumettre la demande de calcul à d'autres ressources.

4.2.5. Travail en commun

Les problèmes axés sur le travail en commun (ou en équipe), également appelés applications coopératives, ont principalement pour objet de permettre et d'améliorer les interactions entre des personnes, en essayant de rapprocher des individus aux fins de collaborations. Ces applications sont souvent structurées autour d'une "organisation commune virtuelle" qui permet l'utilisation de ressources partagées. Dans ces organisations, différents groupes ont en commun un même besoin. Pour réaliser leur objectif, ils doivent effectuer plusieurs types de traitements gourmands en calculs, qui ne peuvent pas être exécutés avec les ressources que possède un seul des participants. Comme exemple, nous pouvons citer les collaborations des chercheurs scientifiques du monde entier à des simulations complexes, des présentations vidéographiques et conférences interactives et simultanées tenues en plusieurs lieux. En outre, plusieurs spécialistes de la grille soutiennent que celle-ci ne sera pleinement exploitée que quand les utilisateurs auront les moyens de construire rapidement et commodément des organisations virtuelles.

4.3. Composants de la grille

Cette section met brièvement en évidence certains principes généraux qui sont à la base de la construction de la grille.
Bien que la grille en soit encore à ses débuts, il existe un consensus remarquable entre les nombreux réalisateurs contribuant à l'élaboration de la technologie de la grille, quant à la façon dont il convient de la structurer.
Parmi les principaux composants sur lesquelles repose la grille, nous citons évidemment le matériel sous-jacent (ordinateurs, réseaux de transmission, appareils spéciaux), les boîtes à outils (tel que Globus qui fournit des protocoles et des services sur lesquels pratiquement tous les grands projets de grille sont fondés), et l'intergiciel qui est la clé du succès de l'informatique en grille. Il est la brique de base, regroupant l'ensemble des éléments logiciels pour la mise en œuvre d'une grille, qui organise et intègre les capacités de calcul dont dispose la grille. Son rôle est d'automatiser toutes les négociations "machine-machine" nécessaires pour unir les ressources de la grille en une unique entité informatique.
Tous ces composants peuvent être englobés dans la définition de l'architecture de la grille.

4.3.1. Architecture de la grille

L'architecture de la grille est souvent décrite en termes de "couches" (figure I.4), dont chacune assure des fonctions spécifiques [Buyya, 02 a].

- **Fabric**

  La couche Fabric désigne l'ensemble de l'infrastructure physique de la grille, incluant les ordinateurs et les réseaux de transmission, les dispositifs de stockage, et les instruments spéciaux comme un télescope ou un détecteur.

- **Ressources et connectabilité**

  Cette couche prend en charge toutes les transactions "spécifiques à la grille" entre des ordinateurs différents et d'autres ressources de la grille. Pour accomplir cette tâche, la grille met en œuvre des protocoles de transmission qui assurent un dialogue des ressources, leur permettant d'échanger des données, et des protocoles d'authentification qui assurent des mécanismes sûrs, de vérification de l'identité des utilisateurs et des ressources.

- **Collective**

  Elle est constituée de services fondés sur des protocoles d'information, qui renseignent sur la structure et l'état des ressources de la grille, et des protocoles de gestion, qui négocient l'accès aux ressources (courtier des ressources, planificateur des tâches des applications pour l'exécution sur des ressources).

- **Application**

  Elle est tributaire de toutes les couches de niveau inférieur pour faire fonctionner la grille (applications bénéficiaires des services fournis par des couches inférieures).

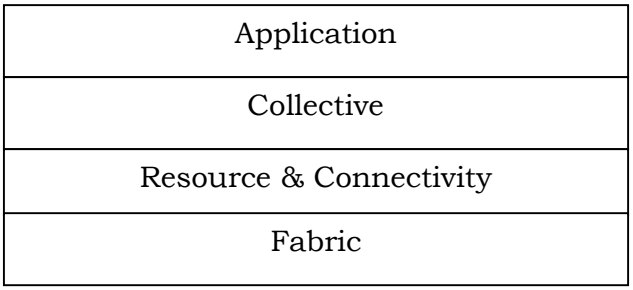

Figure I.4 : Architecture de la grille.

4.3.2. Caractéristiques des grilles

Beaucoup d'aspects caractérisent la grille. Bien sûr, certains aspects caractérisant l'informatique répartie, existaient bien avant. Pourtant, si nous examinons sur quoi les ingénieurs consacrent leur temps et leurs efforts pour que la technologie de la grille soit utilisable, nous constatons qu'il faut répondre à certaines spécifications [Buyya, 02 a]:

- **Hétérogénéité**

  Une des caractéristiques inhérentes aux grilles de calcul est l'hétérogénéité de plusieurs points de vue : matériel, logiciel, politiques de sécurité, contraintes d'exploitation, profil des utilisateurs, etc.

- **Transparence**

  C'est la transparence de cette hétérogénéité, toute la complexité liée aux différences concernant ces aspects doit être invisible pour l'utilisateur final.

- **Autonomie**

  Les contraintes d'exploitation et de qualité de service de chaque site ne doivent pas être sacrifiées par leur intégration à une grille. Donc chaque site doit conserver une totale autonomie concernant ses politiques d'accès, d'exploitation et d'évolution aussi bien côté matériel que logiciel, tout en gardant la compatibilité avec les services offerts à travers la grille. Un juste équilibre entre ces différences doit être trouvé par chaque site participant à une grille de calcul.

- **Extensibilité et dynamicité**

  Les grilles doivent reposer sur une infrastructure extensible et dynamique où le nombre de clients et de serveurs évoluent constamment.

- **Disponibilité**

  Définition d'une politique de partage des ressources des propriétaires pour qu'elles soient disponibles.

- **Administration distribuée**

  Pour exécuter des applications sur de tels types d'environnements, il est nécessaire de disposer d'une infrastructure d'administration adaptée. Une telle infrastructure devrait fournir des mécanismes pour la découverte et l'organisation de ressources, ainsi que des outils pour le déploiement et le contrôle à l'exécution des applications.

- **Tolérance aux pannes**

  Le système doit inclure des mécanismes pour la prise en charge des éléments défaillants.

- **Sécurité**

  Le système doit être sécurisé pour les participants. Un comportement malicieux ne doit pas pouvoir corrompre l'application. Un agent externe ne doit pas pouvoir se faire passer pour un utilisateur autorisé de la grille.

Un autre aspect complémentaire, considéré comme caractéristique de la grille, est l'utilisation de normes ouvertes. Il s'agit de convaincre les ingénieurs et les principales sociétés d'informatique, qui réalisent actuellement la grille, d'établir au départ des normes communes pour la grille, nécessaires pour avoir l'assurance que les études et recherches menées partout dans le monde pourront contribuer de façon positive à l'élaboration de la grille.
Actuellement, une norme connue sous le nom OGSA[7] (Open Grid Services Architecture), est considérée comme une référence clé pour les projets d'élaboration des grilles. Une grande partie des milieux scientifiques contribuent à une activité, en cours depuis 2002 au "Global Grid Forum", visant à définir OGSA [Foster, 02].

[7] OGSA website : http://toolkit.globus.org/ogsa.

4.4. Conception d'applications pour la grille

Bien que les grilles offrent plusieurs avantages, seuls certains types d'applications pourront en bénéficier [Ferreira, 03]. Les applications conçues afin de s'exécuter sur plusieurs processeurs en parallèle bénéficieront le plus des grilles de calcul.
La question la plus importante est de savoir si une application pourra être écrite sous forme d'algorithmes s'exécutant en parallèle. En effet certains algorithmes s'exécutent d'une façon séquentielle et ne pourront pas bénéficier de la puissance de calcul parallèle des grilles.
Le travail du concepteur de l'application consistera donc à déterminer quelles sont les parties de l'application susceptibles de tourner en parallèle et d'implémenter les algorithmes correspondants afin de rendre l'application prête à s'exécuter sur une grille.
En informatique, les problèmes de calcul peuvent se classer d'après leur niveau de parallélisme. Un problème peut-il se découper en beaucoup de petits sous-problèmes qui peuvent être traités en parallèle par des processeurs différents ? Si c'est le cas, le calcul pourra considérablement être accélérer en utilisant de nombreux ordinateurs. En littérature, nous parlons du degré de granularité pour classer les problèmes : calcul à grain fin où les sous-problèmes sont très dépendants, et calculs à gros grain dont la dépendance entre les sous-problèmes est très faible.
A première vue, cette distinction semble suggérer que la grille convienne seulement aux calculs à gros grain. Mais, en fait, beaucoup de problèmes scientifiques intéressants nécessitent une combinaison de techniques à grain fin et à gros grain. C'est là que la grille peut être particulièrement puissante.
Pour certaines applications difficiles à décomposer en sous applications parallèles, une grille s'avère utile lorsque nous souhaitons exécuter plusieurs instances de l'application afin de trouver un résultat optimal, par exemple la trajectoire idéale d'une navette spatiale.
Des décisions devront aussi être prises lors de la conception de l'application en ce qui concerne les données à traiter. En effet en décomposant une application, nous devons prendre en considération la nécessité d'envoyer les données à travers la grille à une sous partie de l'application tournant à distance. Le volume de ces données et leur temps de transfert deviennent des facteurs importants dans la performance globale de l'application. Ainsi, si le volume est important, il devient préférable de placer les données le plus proche possible du nœud effectuant les calculs. Et lorsque les mêmes données sont utilisées par plusieurs nœuds il devient nécessaire de les répliquer dans plusieurs endroits de la grille. Ainsi il faut s'arranger à ce que chaque nœud accède aux données les plus proches. Cela illustre l'importance de la présence d'un service permettant de localiser ces données.

4.5. Les domaines d'applications concernés

Plusieurs domaines peuvent potentiellement bénéficier des grilles tels que:

- L'archivage, l'indexation et le traitement de grandes bases de données.
- Les algorithmes coûteux (simulation, modélisation anatomique etc.).
- Le traitement d'images de très grande taille (séquences dynamiques cardiaques).
- …

Donc parmi les grands défis nécessitant l'exploitation des grilles, nous pouvons citer la météorologie et les études sur le changement climatique global, les sciences de l'univers et l'observation de la terre, les simulations et les outils de conception en médecine, chimie, nucléaire ou biologie, sans oublier les domaines de la finance, etc. Hors du secteur scientifique, d'autres groupes de personnes seront suffisamment motivés pour investir dans l'infrastructure de la grille. Ainsi, cette dernière devient progressivement un enjeu économique. De nombreux secteurs seront concernés (surtout dans le domaine de la simulation), nous pouvons citer :

- **Secteur de l'industrie**

  Il utilise la grille d'une part pour faire des simulations fortement consommatrices de puissance de calcul comme par exemple la conception et l'interaction avec un nouveau moteur d'avion Airbus, la conception d'une fusée ou la simulation de collisions d'automobiles; et d'autre part pour collaborer avec les différentes entités d'un groupe qui sont réparties à travers le monde. Comme par exemple, le système BoilerMaker développé au laboratoire national d'Argonne permet à de multiples utilisateurs de collaborer à la conception des systèmes de contrôle des émissions dans les incinérateurs industriels [Foster, 99]. Les différents utilisateurs interagissent entre eux à l'aide d'une simulation d'incinérateur.
  Un autre exemple est celui du fabricant de microprocesseurs AMD (Advanced Micro Devices) qui a utilisé la grille pour exploiter plus d'un millier de ses ordinateurs (géographiquement distribués) pendant les phases de conception leurs microprocesseurs K6 et K7 [Foster, 99].

- **Secteur de l'énergie atomique :**

  Le Centre Européen de la Recherche Nucléaire (CERN) utilise la grille pour la simulation d'explosions nucléaires.

- **Secteur des jeux :**

  Le fabriquant Sony utilise la grille pour la démultiplication de la puissance de ses consoles de jeu Playstation.
  Le système NICE développé à l'Université de l'Illinois à Chicago permet aux enfants de participer à la création et au maintien de mondes virtuels réalistes, pour le divertissement et l'éducation.

- **Secteur de météorologie**

  Les systèmes modernes de prévision météorologique utilisent des données, situées dans divers sites dans le monde, fournies à partir des observations satellitaires. Plusieurs Téraoctets de données peuvent être traitées et générées par ces systèmes.
  Aerospace Corporation a mis au point un système qui utilise des ressources pour des calculs intensifs des données des satellites pour transmettre les résultats d'un algorithme de détection de nuages aux météorologues à distance en temps quasi réel [Foster, 99].

Donc plusieurs domaines peuvent tirer profit des capacités de la grille. En littérature, les domaines d'application des grilles peuvent être classés en cinq catégories [Bote-Lorenzo, 04] [Foster, 99]:

- **Calculs intensifs distribués**

  Les applications qui nécessitent des calculs intensifs distribués utilisent les grilles pour agréger d'importantes quantités de ressources de calcul afin de résoudre des problèmes qui ne peuvent pas être résolus sur un seul système et qui leur étaient auparavant inaccessibles. Ces ressources sont fédérées sous forme d'un système permettant d'effectuer des calculs à large échelle. En fonction de la grille utilisée, ces ressources agrégées pourraient comprendre la majorité des supercalculateurs dans plusieurs pays ou tout simplement tous les postes de travail au sein d'une entreprise.

- **Calcul haut débit**

  Pour le calcul haut débit, la grille est utilisée en exploitant les cycles de processeur inutilisés (souvent depuis des postes de travail inactifs) afin d'ordonnancer un grand nombre de tâches faiblement couplées ou indépendantes.

- **Calcul à la demande**

  Les applications utilisent les fonctionnalités de la grille pour répondre à court terme aux besoins de ressources qui ne peuvent pas être rentables ou qui ne se situent pas localement. Ces ressources peuvent être des calculs, des logiciels, des capteurs spécialisés, etc.

- **Calcul à forte densité de données (Calcul gourmand en données)**

  La grille est utilisée par de telles applications (gourmandes en données), afin de synthétiser et générer d'importantes quantités d'informations (nouvelles données) à partir d'énormes quantités de données géographiquement distribuées (d'énormes bases de données distribuées).

- **Calcul Collaboratif**

  Dans le calcul collaboratif, la grille est utilisée par les applications qui visent principalement à permettre et à améliorer les interactions entre les personnes. Elle permet la collaboration et la communication entre les utilisateurs. Les applications collaboratives sont souvent structurées dans des environnements sous forme d'espace virtuel partagé.

4.6. Différentes topologies de grilles

Les grilles sont répertoriées, d'un point de vue topologique [Ferreira, 03], en trois types par ordre croissant d'étendue géographique et de complexité : Intragrilles (Intragrids : en analogie avec Intranet), Extragrilles (Extragrids : en analogie avec Extranet) et Intergrilles (Intergrids : en analogie avec Internet). La figure I.5 présente une illustration de ces topologies.

4.6.1. Intragrille

La plus simple des grilles est l'intragrille, composée d'un ensemble relativement simple de ressources et de services et appartenant à une organisation unique. Les principales caractéristiques d'une telle grille sont la présence d'un réseau d'interconnexion performant et à haut-débit, d'un domaine de sécurité unique et maîtrisé par les administrateurs de l'organisation et d'un ensemble relativement statique et homogène de ressources. Une entreprise peut être amenée à construire une intragrille pour augmenter la puissance de calcul de ses équipes de recherche et développement tout en maintenant un niveau d'investissement faible en terme de nouvelles infrastructures (comme par exemple AMD lors de la phase de conception de ses processeurs K6 et K7 [Foster, 99]).

4.6.2. Extragrille

Une extragrille étend le modèle en agrégeant plusieurs intragrilles. Les principales caractéristiques d'une telle grille sont la présence d'un réseau d'interconnexion hétérogène haut et bas débit (LAN /WAN), de plusieurs domaines de sécurité distincts, et d'un ensemble plus ou moins dynamique de ressources.
Un exemple d'utilisation est lors d'alliances et d'échanges "Business- to-Business" (B2B) entre entreprises partenaires.

4.6.3. Intergrille

Une intergrille consiste à agréger les grilles de multiples organisations, en une seule grille. Les principales caractéristiques d'une telle grille sont la présence d'un réseau d'interconnexion très hétérogène haut et bas débit (LAN /WAN), de plusieurs domaines de sécurité distincts et ayant parfois des politiques de sécurité différentes et même contradictoires, et d'un ensemble très dynamique de ressources.

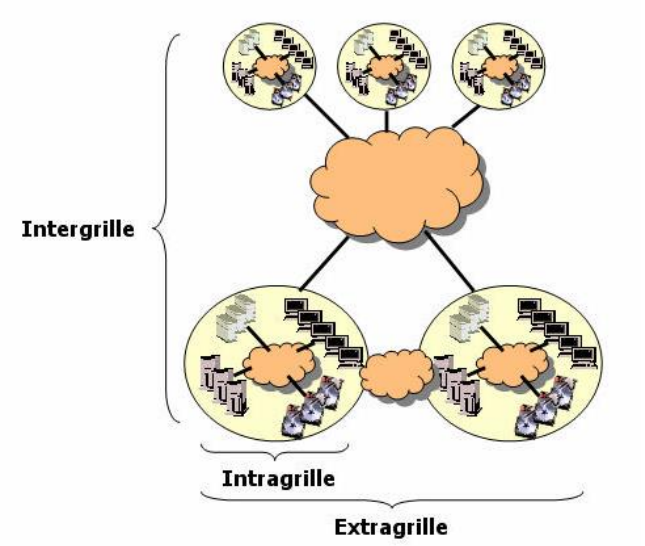

Figure I.5 : Différentes topologies de grilles [Yagoubi, 07].

Les intergrilles seront souvent mises en œuvre lors des grands projets industriels (conception d'un avion par un consortium aéronautique par exemple) ou scientifiques (modélisation de protéines) où plusieurs organisations seront amenées à participer. Avec la naissance de la notion de grille, la notion d'organisation virtuelle (Virtual Organization) apparaît. Les utilisateurs pourront être groupés, selon leurs différents intérêts, dans de telles organisations dont chacune possède sa propre politique. Chaque organisation virtuelle mettra à la disposition de ces utilisateurs un ensemble de ressources sous la forme d'une grille.

4.7. Architectures et modèles économiques de gestion des ressources

Dans des environnements vastement répartis comme les grilles, la gestion des ressources devient un processus complexe à cause de l'étendue géographique et de l'hétérogénéité des ressources et des politiques de gestion et de sécurité. Dans cette section, nous allons tenter de répondre à certaines questions que nous nous sommes posé au début : pourquoi un aussi grand enthousiasme pour les grilles et quelles sont les sources de profits envisageables ? Les fournisseurs, doivent être en mesure de tirer un bénéfice de l'utilisation de leurs ressources par les autres utilisateurs. Dans l'Internet et avec le commerce électronique, les modèles économiques permettant de faire des profits tels que les publicités et les affiliations (pay-per-click) ne peuvent pas être adaptés aux grilles parce que l'utilisateur est le plus souvent une application et non pas un être humain. D'un autre côté, il est impossible de gérer les ressources d'une grille comme dans la vie réelle par des contrats bilatéraux entre tous les utilisateurs et tous les fournisseurs. Pour cela de nouvelles approches pour la gestion des ressources sont nécessaires.
Dans ce qui suit, nous présenterons des architectures et des modèles économiques qui ont été proposés dans la littérature scientifique.

4.7.1. Modèles architecturaux

Le choix du bon modèle architectural de gestion des ressources joue un rôle important dans le succès économique des grilles. Dans [Buyya, 00], les auteurs identifient trois architectures susceptibles de permettre un tel succès :

- Le modèle hiérarchique.
- Le modèle du propriétaire abstrait ("Abstract Owner Model").
- Le modèle de l'économie de marché ("Computational Market/Economy Model").

Ces modèles devront encourager les fournisseurs de ressources à contribuer à la grille, assurer un juste partage de ces ressources entre les utilisateurs et réguler (contrôler) efficacement l'offre et la demande.

1. Modèle hiérarchique

Le modèle hiérarchique est implémenté par la plupart des intergiciels actuels des grilles tels que Globus et Legion. Cette architecture tente de hiérarchiser la gestion des ressources afin de mieux l'optimiser et la rendre plus facile d'un point de vue utilisateurs. Fonctionnellement, cette architecture est divisée en un ensemble de composants passifs et actifs [Buyya, 00].

Les composants passifs étant les ressources proprement dites, les tâches qui devront s'y exécuter, et les ordonnancements de ces tâches sur les ressources.
Les composants actifs sont :

- Les ordonnanceurs (" Schedulers ").
- Les services d'information ("Information Services").
- Les contrôleurs de domaine ("Domain Control Agents").
- Les agents de déploiement ("Deployment Agents").
- Les agents de contrôle d'admission ("Admission Control Agents").
- Les superviseurs ("Monitors").
- Les agents de contrôle des tâches ("Job Control Agents").
- …

La figure I.6 montre ces différents composants.

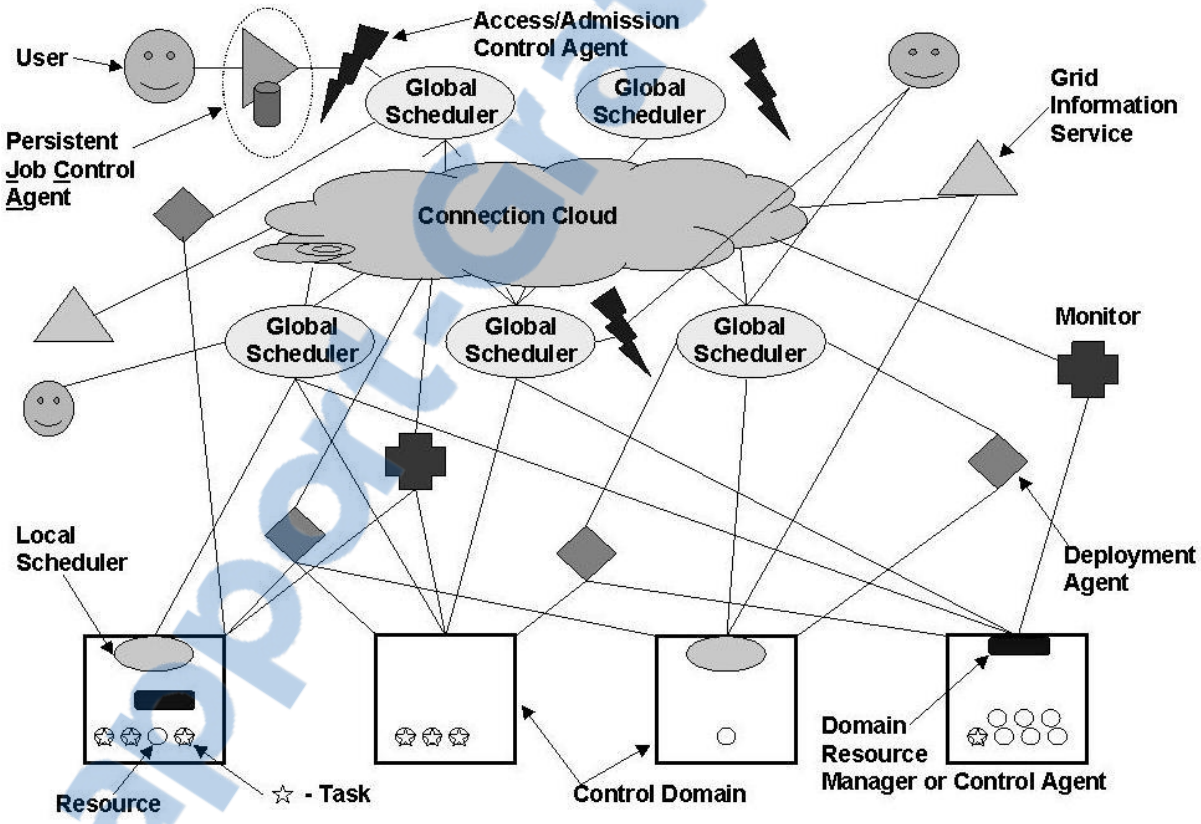

Figure I.6 : Modèle hiérarchique de gestion des ressources [Buyya, 00].

Le modèle hiérarchique facilite l'interaction entre l'utilisateur et le système pour la réservation des ressources. L'utilisateur soumet ses tâches à un agent de contrôle de tâches qui interroge un agent de contrôle d'admission. Ce dernier consulte les services d'information pour connaître l'état des ressources et pouvoir prendre la décision d'autoriser l'admission dans le système. Dans le cas où les tâches sont admises, elles sont soumises aux ordonnanceurs de haut niveau qui se chargent de

contacter les services d'information et les ordonnanceurs de bas niveau afin de localiser les ressources adéquates. Les agents de déploiement s'occupent alors de contacter les contrôleurs de domaine afin d'autoriser le déploiement des tâches sur les ressources.

2. Modèle du propriétaire abstrait

Dans un système aussi complexe et étendu qu'une grille, il est souvent difficile de déterminer quelle entité possède les ressources utilisées. Cela n'est guère nouveau, car dans l'Internet et dans les réseaux de téléphonie les mêmes problèmes se posent. Dans ces réseaux, les utilisateurs ne se soucient presque jamais des vrais propriétaires des ressources utilisées. Par contre, ils sont le plus souvent concernés par des questions telles que l'accès aux ressources, la sécurité, la qualité de service et les mécanismes de comptabilité. Dans le système téléphonique, par exemple, l'entité avec laquelle l'utilisateur négocie l'établissement de l'appel n'est parfois pas le propriétaire des ressources (appels internationaux par exemple). Cependant l'utilisateur traite cette entité abstraitement comme étant le propriétaire de ces ressources car cela facilite la procédure de négociation.
Le plus souvent l'entité que l'utilisateur contacte n'est qu'un courtier représentant les vrais fournisseurs. Cette entité apparaît comme "le seul propriétaire" des ressources de la grille. Elle est appelée "Abstract Owner" ou AO [Buyya, 00].

3. Modèle de l'économie de marché

Les arguments en faveur d'un modèle d'économie de marché pour la gestion des ressources d'une grille sont nombreux. Ce modèle incite les fournisseurs à contribuer leurs ressources car il leur offre la possibilité de tirer des profits (souvent financiers) de l'utilisation qui en sera faite. L'économie est le meilleur régulateur de l'offre et de la demande. Les utilisateurs souhaitent utiliser les ressources en minimisant leurs coûts tandis que les fournisseurs souhaitent maximiser leur retour sur investissement et leur bénéfice. Le modèle de l'économie de marché permet, à travers les négociations entre utilisateurs et fournisseurs, d'atteindre un équilibre entre l'offre et la demande.
D'un point de vue technique, un système de gestion des ressources fournissant les outils et les mécanismes adéquats permettant aux utilisateurs et aux fournisseurs d'exprimer leurs besoins est nécessaire.
La figure I.7 montre un exemple d'un tel système.

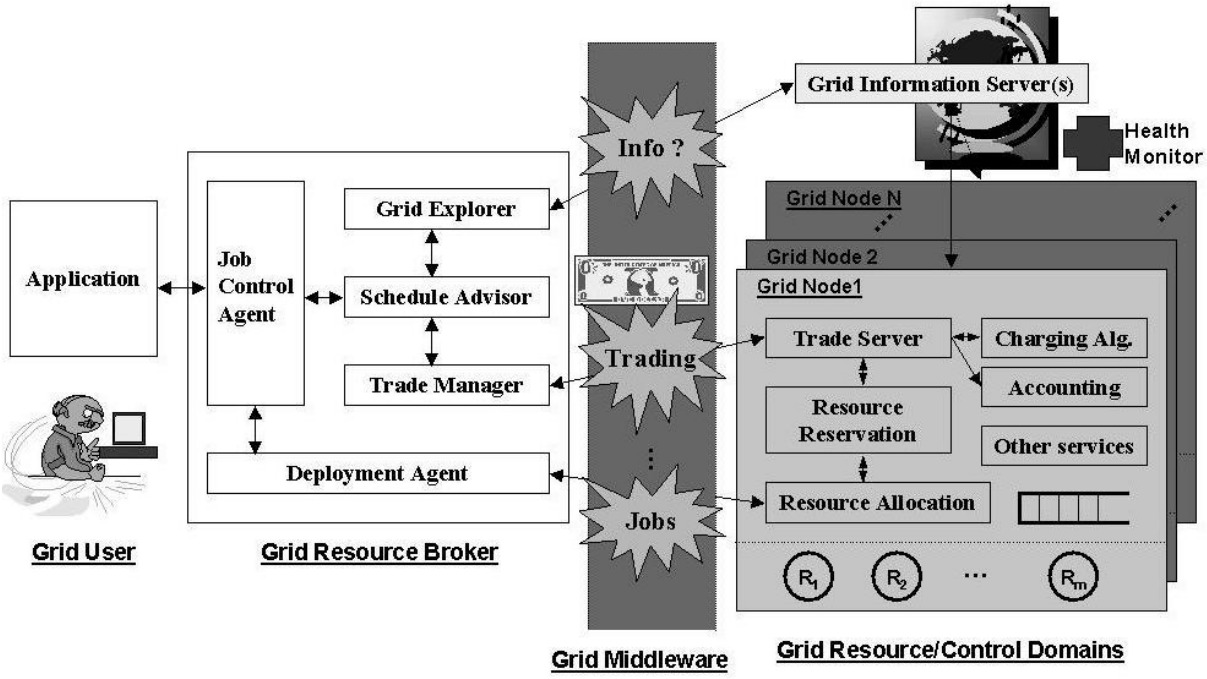

Figure I.7 : Modèle de l'économie de marché [Buyya, 00].

Les principaux composants de cette architecture sont :

- Le courtier ("Grid Resource Broker") : Il constitue l'intermédiaire entre les utilisateurs et les ressources. Il utilise les services de l'intergiciel. Il a la responsabilité de découvrir les ressources, de recevoir les besoins des utilisateurs, de sélectionner les ressources adéquates et d'exécuter les tâches sur ces ressources. Il présente la grille aux utilisateurs comme une ressource unique.
- L'intergiciel de la grille.
- Les ordonnanceurs locaux.

4.7.2. Modèles économiques

Les principes des modèles économiques employés dans les marchés réels peuvent être appliqués aux grilles. Ces modèles se basent soit sur le principe du marchandage soit sur le principe du prix.

Dans le premier, tous les participants doivent posséder des ressources qu'ils contribuent à la grille pour pouvoir accéder à d'autres ressources. Dans le second modèle, les ressources ont un prix que l'utilisateur devra payer afin d'y accéder. Ce prix est déterminé par l'intensité de l'offre et de la demande. Dans un modèle d'économie de marché la présence de plusieurs utilisateurs en compétition pour l'accès aux ressources et de plusieurs fournisseurs en compétition pour la vente de leurs ressources est le facteur qui permet au système d'atteindre un équilibre reflétant l'état de l'offre et de la demande.

Dans un tel système nous trouvons des fournisseurs de services ("Grid Service Providers") jouant le rôle de producteurs, et des courtiers ("Grid Resource Brokers") jouant le rôle de consommateurs. Parmi les modèles économiques employés, nous trouvons [Buyya, 02 b] :

- Le modèle de l'offre et de la demande.
- Le modèle des enchères.
- Le modèle des appels d'offre.
- …

4.8. Les types de grilles

En littérature beaucoup de termes, désignant la fonction spéciale de chaque grille, apparaissent. Différentes terminologies du type de grilles ont été proposées :

- **Grille de calcul (Computing Grid)**

  Cette terminologie a probablement la signification la plus diverse. Ce type de réseaux a pour but d'agréger la puissance avec un double objectif : obtenir plus de performance et réduire les coûts. C'est une extension des systèmes parallèles et distribués à une échelle sans précédent de très forte hétérogénéité, pour subvenir aux besoins d'un calcul très performant [Skillicorn, 02].

- **Grilles de services**

  Cette catégorie devient de plus en plus populaire en sociétés commerciales. Ces dernières aiment agréger les ressources distribuées en une ressource virtuelle commune, et la livrer à leurs clients par des réseaux étendus. Par exemple, les projets de "l'utilitaire du centre de données" de HP et le "e-utilitaire" d'IBM ont l'intention de tirer profit des ressources disponibles à des emplacements géographiquement éloignés pour fournir des services de toile et de stockage à leurs clients dans le monde entier [Graupner, 03].

- **Grilles de données (Data Grid)**

  Ces réseaux fournissent des services spécialisés pour le traitement de très grands ensembles de données qui doivent être transférés et reproduit sur des sites différents.

Si les grilles de calcul ont fait l'objet de nombreux travaux ces dernières années, en revanche, les efforts qui ont été faits pour faire avancer les aspects liés à la gestion des données à très grande échelle ne s'avèrent pas suffisant. Paradoxalement, nous disposons à ce jour d'infrastructures complexes de calcul permettant d'ordonnancer des calculs répartis sur plusieurs sites, alors que le transfert des données vers ces sites est laissé à la charge de l'utilisateur, où au mieux, des fonctionnalités rudimentaires de type transfert de fichiers (FTP) sont proposées. Ceci devient un facteur limitant dans l'exploitation efficace des grilles de calcul.

En 1999, les auteurs de [Chervenak, 00] ont présenté le concept des grilles de données comme une architecture intégrée qui est une spécialisation et une extension d'une grille de calcul. La grille de données a été conçue pour satisfaire les exigences de la combinaison d'une gestion des grandes tailles des ensembles de données, d'une distribution géographiquement éloignée d'utilisateurs et de ressources, et d'une puissance de calcul intensive. Son architecture est aussi développée pour supporter

des travaux dans des environnements hétérogènes et à grande échelle. Conçue sous forme d'un système en couches, elle comporte un service fondamental de "metadata" qui gère l'information sur les données aussi bien que des données reproduites. La différence clef entre une grille de données et une grille classique est la proposition d'un système de répliques et ses sous-systèmes (composants de gestion des répliques et de sélection des répliques) [Chervenak, 00].
Dans ce contexte, il semble important de complémenter les recherches sur les mécanismes de gestion des calculs sur la grille par des recherches sur une gestion efficace des données utilisées dans la grille. L'objectif est de revoir les techniques classiquement utilisées pour la gestion des données partagées et répliquées à l'échelle d'une grappe à la lumière des nouvelles problématiques liées au passage à l'échelle.
Est-ce que les modèles de cohérence et les protocoles utilisés classiquement à l'échelle d'une grappe sont adaptés pour une utilisation à très grande échelle ?
Est-ce que les hypothèses fondatrices de ces modèles restent valides ?
Est-ce que les contraintes de mise en œuvre sont les mêmes ?
De nombreux problèmes peuvent existés. De ce fait le troisième chapitre sera consacré à la présentation de quelques fondements scientifiques sur la gestion des données, pour tenter d'éclaircir quelques problèmes concernant les grilles de données (principes et défis liés à la gestion des données, sous contraintes de performance, de cohérence, et de tolérance aux fautes).

5. Conclusion

Ce chapitre nous a permis de faire un état de l'art sur quelques systèmes distribués à grande échelle. Nous avons présenté certaines notions et certaines caractéristiques sur les grappes de calculateurs, sur les réseaux pair à pair et beaucoup plus de notions et de caractéristiques sur les grilles, puisque la suite de nos chapitres est liée aux grilles.

Quoique certains aient des opinions sceptiques au sujet de la grille, l'avenir de celle-ci est largement tracé d'un point de vue économique, grâce à l'évolution des technologies réseaux. Nous pourrions donc imaginer un avenir axé sur le "tout réseau", avec des utilisateurs ayant seulement un terminal doté d'un minimum de puissance. Il sera ainsi possible de louer de la puissance CPU, de louer des espaces disques et d'augmenter la mémoire d'une façon très simple.

Dans le chapitre suivant, nous allons présenter des concepts et des notions sur les agents et les systèmes multi-agents.

Chapitre II.
Les systèmes multi-agents

1. Introduction

Les Systèmes Multi-Agents (SMA) et les agents autonomes offrent une nouvelle manière de concevoir, d'implanter et d'analyser des systèmes informatiques complexes. Les agents sont utilisés dans une variété d'applications qui ne cesse d'augmenter. Lorsque nous abordons ce domaine, il est vraisemblablement nécessaire de commencer par la définition des termes « agent » et « système multi-agents ». Or, lorsque nous cherchons la définition de ces termes dans la littérature, nous constatons que les définitions sont sensiblement différentes d'un auteur à autre. Les travaux sur les systèmes multi-agents ont vu naître un grand nombre de sous domaines de recherche et la classe des applications concernées par les systèmes multi-agents s'élargit de plus en plus.
Nous présentons dans ce chapitre les notions et les principaux concepts d'agents et de systèmes multi-agents, issus de l'Intelligence Artificielle Distribuée (IAD).

2. Intelligence artificielle distribuée

Au départ les systèmes logiciels étaient considérés comme des entités recevant en entrée des données et fournissant des résultats en sortie. C'est sur cette idée que les systèmes centralisés et séquentiels ont été développés.
Avec l'accroissement de la complexité des problèmes, il est devenu nécessaire de décomposer les systèmes logiciels en plusieurs sous-systèmes moins complexes ayant la capacité d'interagir entre eux. L'utilisation de cette approche décentralisée dans le domaine de l'intelligence artificielle a donné naissance à l'intelligence artificielle distribuée.
Donc, l'intelligence artificielle distribuée est la branche de l'intelligence artificielle classique qui s'intéresse à des comportements intelligents qui sont le produit de l'activité coopérative et des interactions entre plusieurs agents [Benmerzoug, 09].
L'intelligence artificielle distribuée a introduit le concept de système multi-agents qui porte sur le modèle de l'agent dont les caractéristiques sont : la coopération, la coordination et la communication [Tranvouez, 01].

3. Les agents

3.1. Définition d'un agent

Il n'existe pas une définition unique de ce qui est un agent. Plusieurs définitions cohabitent [Bousseta, 98].
C'est la raison pour laquelle le terme agent est utilisé de manière vague.
Ferber dans [Ferber, 95] définit un agent comme étant :
« Une entité autonome, réelle ou abstraite, qui est capable d'agir sur elle-même et sur son environnement, qui dans un univers multi-agents, peut communiquer avec d'autres agents, et dont le comportement est la conséquence de ses observations, de ses connaissances et des interactions avec les autres agents ».
A partir de cette définition, nous pouvons attribuer aux agents des propriétés clés telles que l'autonomie, l'action, la perception et la communication. D'autres propriétés peuvent aussi être attribuées aux agents, en particulier la réactivité, la rationalité, l'engagement et l'intention [Chaib-Draa, 01].

De leur côté, les auteurs dans [Wooldridge, 94] ont proposé la définition suivante :
« Un agent est un système informatique, situé dans un environnement, et qui agit d'une façon autonome et flexible pour atteindre les objectifs pour lesquels il a été conçu ».

La plupart des chercheurs se mettent d'accord pour dire qu'un agent est un programme, possédant une certaine autonomie, de l'intelligence (bases de connaissances ou autres) et des objectifs bien définis. Un agent est situé, c'est-à-dire qu'il possède une représentation de son environnement. Il est donc capable d'interagir avec son environnement, et par conséquent, d'avoir une influence sur ce dernier. Enfin, il est capable de communiquer avec les autres composants du système (par exemple avec les autres agents).

3.2. Caractéristiques d'un agent

A partir de ces définitions sur les agents, nous pouvons extraire plusieurs propriétés ou caractéristiques d'un agent.
Un agent est :

- **Autonome**

  Un agent a un certain degré d'autonomie. Il possède certains états (non-accessibles aux autres agents et composants du système). Et par rapport à ses états, l'agent peut prendre certaines décisions (sans intervention externe directe).

- **Situé**

  Un agent est situé dans son environnement (physique ou virtuel). Il a une représentation de son environnement.

- **Réactif**

  Un agent peut percevoir son environnement via des senseurs. Il peut aussi agir sur son environnement.

- **Social**

  Un agent est capable d'interagir et de communiquer avec les autres agents (par des langages de communication). Il est capable de coopérer pour résoudre des problèmes ou effectuer des tâches.

- **Proactif**

  Un agent est capable de "prendre de l'initiative" pour atteindre son but ou effectuer des tâches (et d'adopter les comportements appropriés).

- **Actif**

  Un agent est toujours actif. Il s'exécute donc nécessairement dans un thread ou un processus indépendants.

- **Apprentissage**

  Un agent est capable d'apprendre et d'évoluer en fonction de cet apprentissage. Il est capable de changer son comportement (en fonction des expériences passées).

4. Les systèmes multi-agents

Les systèmes multi-agents ne sont pas récents et ils représentent actuellement un champ de recherche très actif. Ce domaine est l'intersection de plusieurs domaines : en particulier des systèmes distribués, du génie logiciel et surtout de l'intelligence artificielle.
Un système multi-agents est un système distribué composé d'un ensemble d'agents. Contrairement aux approches de l'intelligence artificielle, qui simulent dans une certaine mesure les capacités du raisonnement humain, les systèmes multi-agents permettent de modéliser un ensemble d'agents, qui interagissent généralement suivant des modes de coopération, de concurrence ou de coexistence [Brandolese, 00] [Chaib-draa, 02] [Moulin, 96].

4.1. Définitions

Les systèmes multi-agents constituent une nouvelle technique de modélisation qui place l'objet d'étude au centre de sa démarche. Ces modèles représentent les actions individuelles, les interactions entre les acteurs et les conséquences de ces interactions sur la dynamique du système.
Les systèmes multi-agents héritent de l'intelligence artificielle distribuée les modes de communication et de concertation entre agents. Ils reprennent les idées d'autonomie et d'émergence du résultat final à partir des interactions individuelles [Chaib-draa, 01].

Selon Ferber dans [Ferber, 95] :
« Un système multi-agents est un système composé des éléments suivants :

- Un environnement, c'est à dire un espace disposant généralement d'une métrique,
- Un ensemble d'objets situés dans l'espace, ils sont passifs, ils peuvent être perçus, détruits, créés et modifiés par les agents,
- Un ensemble d'agents qui sont les entités actives du système,
- Un ensemble de relations qui unissent les objets entre eux,
- Un ensemble d'opérations permettant aux agents de percevoir, de détruire, de créer, de transformer et de manipuler les objets,
- Un ensemble d'opérateurs chargés de représenter l'application de ces opérations et la réaction du monde à cette tentative de modification (les lois de l'univers) ».

Cette définition présente une description globale de la structure d'un système multi-agents. Elle propose une description des principes de la philosophie multi-agents sans que les domaines d'applications puissent être limitant.
Les idées directrices de conception d'un système multi-agents doivent être dissociées des domaines d'application. Souvent dans ces applications, les agents représentent des êtres vivants.
Cette proximité peut mener vers une généralisation, des principes spécifiques aux sociétés tels que la communication ou la reproduction, à tous les systèmes multi-agents.

4.2. Types d'agents

A partir des définitions des agents, deux courants ont émergé conduisant à distinguer agents réactifs et agents cognitifs [Bousseta, 98]. En fonction de leur environnement, de leur représentation et de leur mode de fonctionnement, les agents peuvent être réactifs ou cognitifs.

4.2.1. Agent réactif

Un agent réactif agit en réponse à des stimuli de son environnement, sans connaissance d'un objectif. Ses actions sont régies par des règles prédéfinies de comportements. L'agent réactif ne dispose que d'un protocole et d'un langage de communication réduits. Ses capacités répondent uniquement à la loi stimulus/actions : l'occurrence de chaque événement déclenche l'exécution d'une action prédéfinie.
Dans un système d'agents réactifs, il n'est pas nécessaire que chaque agent soit individuellement intelligent pour atteindre un comportement global intelligent. Cette approche propose la coopération d'un grand nombre d'agents de faible granularité [Ferber, 95].

4.2.2. Agent cognitif

Un agent cognitif possède une base des connaissances comprenant des informations sur ses objectifs et sur son environnement. En fonction des informations disponibles, il agit selon des plans explicites lui permettant d'atteindre ses objectifs [Ferber, 95]. Un ensemble d'agents cognitifs ressemble à un groupe d'individus qui doivent coopérer et négocier pour accomplir leurs buts. Les analogies sociales amènent les chercheurs dans ce domaine, à s'appuyer sur des travaux de sociologie lors de la mise au point des protocoles de négociation [Ferber, 95].
Un système d'agents cognitifs est constitué d'un petit nombre d'agents de forte granularité ; chaque agent est apparenté à un système expert plus ou moins évolué. Les actions de ces agents seront ainsi « réfléchies » du fait qu'elles seront basées sur les connaissances de l'agent sur lui-même, sur les autres agents et sur son environnement et les objectifs qui le guident.

4.2.3. Agent hybride

La distinction entre agents cognitifs, capables de raisonner, et agents réactifs, qui ne font que réagir immédiatement aux stimuli perçus, a fait apparaitre l'intérêt des agents hybrides.
La figure II.1 illustre une architecture typique, d'un agent hybride, constituée de couches distinctes. Ces architectures sont principalement conçues pour permettre à des agents cognitifs d'intégrer des capacités réactives.

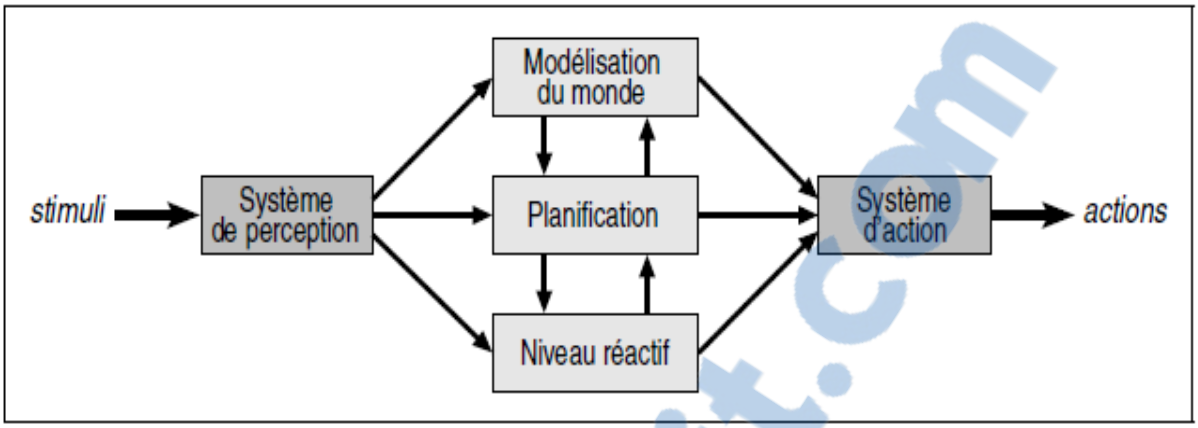

Figure II.1 : Architecture typique d'agent hybride [Richard, 01].

Dans ce type d'architecture, un agent est composé de plusieurs couches arrangées selon une hiérarchie. La plupart des architectures se composent de trois couches.
Au niveau le plus bas de l'architecture, nous trouvons une couche exclusivement réactive où les décisions sont prises en se basant sur des données brutes de l'environnement (qui proviennent des senseurs).
La couche intermédiaire fait abstraction de ces données brutes. Elle travaille avec une vision située au niveau des connaissances de l'environnement. La couche supérieure se charge des aspects sociaux et comportementaux de l'environnement (raisonnement tenant compte des autres agents).

4.3. Autres approches d'agent

4.3.1. Agent autonome

L'approche des agents autonomes ne suppose pas l'existence de plusieurs agents dans le cadre d'un système multi-agents. L'agent est considéré comme un collaborateur intelligent capable d'apprendre des habitudes de l'utilisateur pour le décharger de certaines tâches répétitives [Bousseta, 98].
Parmi les travaux qui illustrent mieux cette approche, nous pouvons citer le système Telescript développé par General Magic [Tardo, 96]. L'agent est une procédure informatique, associée à un état courant et pouvant migrer d'une machine à une autre pour continuer son exécution.
L'agent WBI (Web Browser Intelligence) d'IBM dont le but est d'assister l'utilisateur dans sa recherche d'information sur le Web en se servant de l'historique des navigations [Barrett, 97].

4.3.2. Agent social

Cette approche considère que le concept d'agent ne prend tout son intérêt que dans un système multi-agents. L'agent a un comportement individuel et collectif. Le caractère individuel d'un agent se manifeste à travers des objectifs qui lui sont propres et qu'il essaye dans la mesure du possible d'atteindre avec les ressources et les compétences dont il dispose [Bousseta, 98].
Le caractère collectif se manifeste sous forme d'interactions entre les agents. Ces interactions peuvent être de différents types : coordination, coopération, négociation.

4.4. Les intérêts des systèmes multi-agents

Les modèles informatiques classiques s'appuient généralement sur des équations différentielles et reposent sur des relations de cause à effet. Ils sont très puissants mais présentent cependant une faiblesse quant à la mise en évidence des rapports entre différents niveaux [Ferber, 95].
L'approche multi-agents a une philosophie centrée sur une représentation directe des individus et des comportements.

- Elle permet de représenter plusieurs niveaux : l'individu, des groupements d'individus et l'ensemble du système. L'évolution du système au niveau supérieur doit émerger des interactions entre les individus.
- Cette approche est particulièrement bien adaptée à la simulation des systèmes complexes dont le fonctionnement global émerge des actions des individus. Les systèmes multi-agents permettent de faire vivre virtuellement des agents autonomes sur ordinateur et d'y effectuer des expériences difficiles, voire impossibles, à mener dans la réalité, d'où la qualification de laboratoires virtuels [Treuil, 97].
- Elle offre une grande flexibilité pour la programmation. L'utilisation d'entités individualisées et la programmation des processus localement dans différents modules apportent une grande flexibilité.
- Le langage utilisé est souvent un langage orienté objet permettant le développement des programmes de façon modulaire. Les modifications ne nécessitent pas de large restructuration du programme.
- Le modèle choisi peut être facilement adapté à des cas différents en ajoutant soit des agents soit des comportements. Les modèles sont ainsi facilement évolutifs.
- Un autre avantage est de pouvoir manipuler à la fois des paramètres quantitatifs et des paramètres qualitatifs (des comportements).

5. Interaction et communication entre les agents d'un SMA

Parfois, l'idée de la communication est liée à celle d'interaction. La communication la plus évoluée nécessite un langage commun qui manipule des symboles. Cependant, l'action sur l'environnement et la modification de ce dernier par un agent influencent les actions des autres agents. Dans ce cas, c'est une interaction qui s'est produite et non pas une communication. Un système peut évoluer vers la résolution d'un problème sans qu'il y ait communication directe entre les agents (figure II.2).

5.1. Les interactions

L'action et l'interaction constituent les éléments moteurs de la structuration d'un système dans son ensemble. Une interaction est une mise en relation dynamique de deux ou plusieurs agents [Bousseta, 98]. L'interaction entre les agents représente plus qu'un simple échange de messages. Un des aspects d'interaction est la conversation. Elle est basée sur un échange partagé et conventionné de messages.

Les conversations entre les agents sont souvent structurées suivant des schémas typiques appelés protocole d'interaction. Un protocole précise "qui peut dire quoi à qui et les réactions possibles à ce qui est dit".

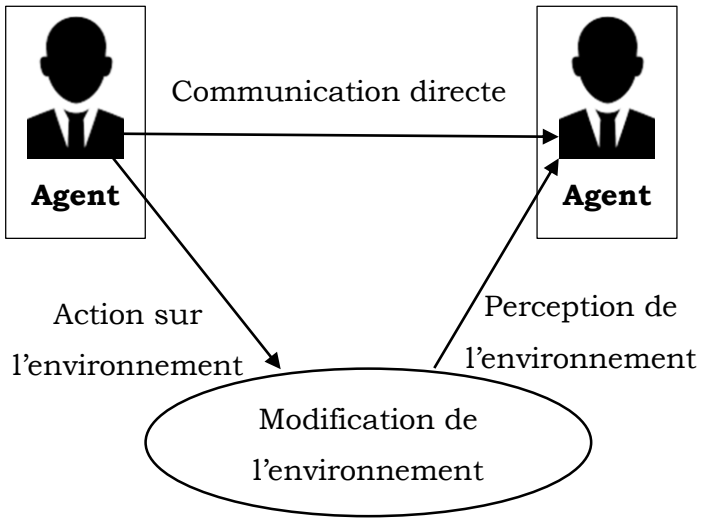

Figure II.2 : Interactions entre deux agents par modification de l'environnement ou par communication directe.

Les protocoles d'interaction permettent de définir une séquence causale des messages communiqués entre les agents. Ils décrivent comment les agents doivent réagir aux messages reçus durant les interactions.
Les aspects de ces protocoles d'interaction diffèrent selon le type des agents (agents concurrents ou agents ayant des buts communs). Donc, Il existe différents types de protocoles d'interaction. Nous citons :

- Les protocoles de coordination.
- Les protocoles de coopération.
- Les protocoles de négociation.

5.1.1. La coordination

Elle se traduit par un comportement individuel qui vise à satisfaire ses propres intérêts tout en essayant d'atteindre l'objectif global du système. Nous citons le protocole du réseau contractuel (Contract-net) [Smith, 80] comme exemple des protocoles de coordination [Davis, 83]. L'intérêt de ce protocole est qu'il réalise la coordination des tâches parmi les agents en assurant l'allocation la plus optimale possible.

Plusieurs exemples de coordination issus de notre vie peuvent être citer : des personnes qui parlent à tour de rôle en se passant un micro, l'échange des balles entre des jongleurs avec lesquelles ils jonglent, le déplacement d'un objet lourd par des déménageurs, etc.

D'après Malone dans [Malone, 87], l'allocation de ressources rares et la communication de résultats intermédiaires sont des composantes fondamentales de la coordination entre agents. Les agents doivent être capables de communiquer entre eux de telle sorte qu'ils puissent échanger les résultats intermédiaires. Et pour l'allocation des ressources partagées, ils doivent en plus être capables de faire des transferts de ressources. Cela peut imposer certains comportements à des agents particuliers.

5.1.2. La coopération

Une technique courante des protocoles de coopération consiste à décomposer une tâche en sous-tâches puis à les répartir entre les différents agents. L'avantage de cette technique est qu'elle réduise la complexité de la tâche initiale. Cependant, elle peut engendrer des conflits entre les agents suite aux interactions qui peuvent y avoir entre les sous-tâches.
Après la décomposition de la tâche en sous-tâches, ces dernières peuvent être attribuées aux agents selon différents mécanismes de répartition. Généralement, les mécanismes utilisés sont basés sur :

- L'économie du marché : où les sous-tâches sont allouées aux agents selon le principe de l'offre et de la demande (achat et vente).
- La planification : où certains agents planificateurs sont responsabilisés pour répartir les sous-tâches.
- L'élection : où les sous-tâches sont attribuées à des agents suite à un vote ou un accord.

5.1.3. La négociation

Les protocoles de négociation sont utilisés dans les cas où les objectifs des agents sont différents. Un protocole de négociation est généralement considéré comme une stratégie de résolution, utilisée pour parvenir à un accord entre les agents qui visent à résoudre les conflits. Donc, la négociation est un mécanisme de résolution de conflit [Moulin, 96]. Dans [Durfee, 90], Durfee et al. définissent la négociation comme étant le processus qui vise à améliorer les accords sur des points de vue communs ou des plans d'action, en échangeant de manière structuré des informations pertinentes. Cette amélioration est réalisée en réduisant les inconsistances et l'incertitude. Et selon les auteurs dans [Chu-Caroll, 95], la résolution des conflits peut s'effectuer dans deux contextes différents : favorable ou défavorable. Dans le cas d'un contexte favorable, tous les agents sont prêts à faire les concessions nécessaires pour parvenir à un accord. Dans le cas d'un contexte défavorable, le principal objectif pour au moins un des agents, est de parvenir à un compromis qui le convient même au détriment des autres agents. Certains chercheurs en intelligence artificielle distribuée considèrent la négociation comme étant un mécanisme de coordination entre un groupe d'agents. Elle peut aussi être considérée comme une coopération dont l'aboutissement est un accord.

5.2. La communication

La communication est la forme d'interaction consciente la plus élémentaire dans un système multi-agents et constitue l'ossature sur laquelle des modes d'interaction plus évolués s'appuient. La communication dans un système multi-agents suppose l'existence d'un médium physique (un réseau informatique par exemple), d'un médium linguistique (langage d'expression) et un mode de diffusion. Le médium physique dépend des choix techniques d'implémentation (architecture du réseau, types de connections, protocoles de communication).
Le médium linguistique nécessite la spécification d'un langage commun à un ensemble d'agents leur permettant d'interpréter un message. Le langage KQML (Knowledge Query Manipulation Language), développé dans le cadre du projet DARPA [Finin, 94], largement utilisé dans les systèmes multi-agents ainsi que le langage ACL (Agent Communication Language) proposé par FIPA[8] (Foundation for Intelligent Physical Agents), reposent sur cette théorie.
Les modes de diffusion se distinguent en deux grands types : les modes anonymes et les modes nominatifs.

- Le mode de diffusion anonyme peut s'effectuer à travers un tableau noir (Blackboard System) [Corkill, 91]. Les agents envoient leurs informations ou requêtes à une zone de connaissance (ou mémoire) partagée que chaque agent peut consulter. Les agents lisent ainsi tous les messages inscrits dans le tableau noir mais ne réagissent qu'à ceux qui les concernent [Chaib-draa, 02].

- A l'opposé, dans le mode de diffusion nominatif ou par message, un agent envoie "personnellement" un message à un autre agent en utilisant son adresse.

6. Quelques applications des systèmes multi-agents

Les systèmes multi-agents permettent de modéliser une grande quantité de systèmes relevant de domaines d'application très variés. Pour illustrer ce fait, nous allons d'abord évoquer quelques applications relevant du domaine industriel, puis de domaines divers [Boussebough, 11] [Chaib-draa, 95].

6.1. Applications industrielles

- La conception et l'ingénierie simultanée : il s'agit de produire de meilleurs produits en un temps court. A cet effet, toutes les étapes du cycle de vie du produit (définition du cahier des charges, conception, élaboration des méthodes, mises en fabrication…) sont considérées aussi tôt que possible.

- Le système d'information et le CIM (Computer Integrated Manufacturing) : il est de plus en plus prouvé que la productivité des entreprises est plus limitée par l'information que par le travail ou le capital. Un exemple de gêne à la circulation de l'information provient des logiciels des différentes parties de l'entreprise qui ne

[8] FIPA website : http://www.fipa.org.

peuvent pas "discuter" entre eux. Dans l'approche des systèmes multi-agents, chacune de ces parties peut être vue comme un agent.

- La distribution du système de production (multi-sites) : chaque site de production est autonome, mais doit tenir compte des décisions des autres sites.

- La gestion de la production : le problème consiste à faire cohabiter des objectifs qui ont des termes plus ou moins longs.

- L'ordonnancement et le pilotage d'un système de production : comment réordonnancer dynamiquement un atelier qui subit des événements perturbateurs. Ce réordonnancement est une résolution coopérative et distribuée de problèmes entre les différentes entités de l'entreprise. La résolution part de la plus petite entité (le poste de travail), et tente de résoudre le problème en mettant en jeu progressivement de plus en plus d'entités (machines identiques, puis atelier, etc.).

- Les chaînes logistiques : ce sont des réseaux de fournisseurs, d'usines, d'entrepôts, de centres de distribution et de détaillants à travers lesquels des matières premières sont acquises, transformées et livrées au consommateur. L'objectif est d'optimiser les performances de ces chaînes logistiques. Pour cela, les niveaux de décision stratégique, tactique et opérationnelle sont mis en jeu pour que chaque maillon opère de manière intégrée à l'ensemble de la chaîne.

6.2. Autres applications

Parmi les autres domaines dans lesquels les systèmes multi-agents se révèlent utiles, nous citons :

- Les transports : une approche multi-agents a été appliquée pour résoudre les problèmes de planification et de négociation dans les systèmes de transport de marchandises par camions. Les conducteurs de camion et leur compagnie respective y sont vus comme des agents intelligents, autonomes et rationnels. L'objectif de l'étude est d'atteindre un compromis qui minimise le coût global des transports en appliquant successivement différentes méthodes de coopération. A cet effet, les conducteurs négocient à travers leur compagnie leurs propositions en effectuant des ventes et des achats de commandes.

- La biologie : une étude vise à étudier l'émergence d'un comportement global intelligent à partir du comportement basique de chaque agent. Cette étude a montré comment un groupe de termites au comportement très simple peut rassembler en une seule pile des copeaux de bois. En outre, elle a aussi étudié la propagation d'une épidémie dans une population composée d'agents sains et d'agents malades.

7. Plates-formes de développement des systèmes multi-agents

La réalisation des systèmes multi-agents rencontre des difficultés qui sont essentiellement liées à la complexité de ces systèmes. En plus des problèmes des systèmes distribués et concurrents, ces systèmes rencontrent d'autres problèmes liés à leurs caractéristiques (dynamique des organisations et flexibilité des interactions). Pour que leur réalisation soit plus accessible, des plates-formes multi-agents (environnements de développement) ont été conçues et développées à partir des architectures et des langages existants. Les plates-formes multi-agents facilitent le développement des systèmes multi-agents. Elles fournissent les outils nécessaires pour la création et la manipulation des agents et pour l'interaction entre les agents dans un environnement spécifique. Ces environnements de développement sont les meilleurs moyen pour réaliser des applications orientées agents.

Plusieurs plates-formes de développement existent. Nous allons citer quelques exemples fournis comme logiciels libres : MACE, ZEUS, MADKIT, SWARM et JADE.

- MACE [Gasser, 87] : la plate-forme MACE (Multi-Agent Computing Environment) est considérée parmi les premiers environnements de conception et d'expérimentation des systèmes multi-agents. D'après Ferber dans [Ferber, 95], l'impact de MACE est considérable sur les recherches ultérieures en intelligence artificielle distribuée et toutes les plates-formes actuelles de développement des systèmes multi-agents sont des descendants directs ou indirects de MACE.

- ZEUS [Nwana, 99] : est une plate-forme multi-agents conçue et réalisée par le laboratoire de "British Telecom[9]" pour développer des applications collaboratives. La plate-forme ZEUS est développée en langage Java. Elle génère automatiquement le code Java des agents spécifiés graphiquement. Elle permet l'observation du système pendant son exécution. Par conséquent, elle facilite un développement rapide des systèmes multi-agents.
L'environnement de ZEUS comporte trois bibliothèques : une bibliothèque de composants d'agents, une bibliothèque d'outils de création graphique d'agents et une bibliothèque d'agents spécialisés (agents de visualisation, serveurs de noms, agents de débogage, etc.) [Richard, 01]. Les agents de ZEUS servent principalement à la planification des tâches, car leur rôle est typiquement de raisonner quand et comment activer ou désactiver des systèmes externes.
Plusieurs applications ont été développées en utilisant ZEUS, comme les ventes aux enchères et la simulation de la fabrication des ordinateurs.

- MADKit[10] (Multi-Agent Development Kit) [Gutknecht, 00] est une plate-forme développée par le laboratoire LIRMM (Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier). C'est une plate-forme générique de conception et d'exécution de systèmes multi-agents. MADKit est développée en langage Java et

[9] ZEUS : développé par "Agent Research Program" du "British Telecom Intelligent Research Laboratory".
[10] MADKit website: http://www.madkit.org.

repose sur un modèle organisationnel (modèle AGR : Agent – Groupe - Rôle), plutôt que sur une architecture d'agent ou sur un modèle d'interaction [Gutknecht, 00].
Le modèle AGR repose sur les notions d'agent, de groupe et de rôle :
Le rôle est une abstraction d'une fonction ou d'un service. Un rôle est local à un groupe. Un agent joue un ou plusieurs rôles et appartient à un ou plusieurs groupes d'agents. Un agent peut changer de rôle et peut rejoindre un groupe ou le quitter de manière dynamique.

- SWARM[11] est une plate-forme multi-agents pour développer des applications de simulation. Elle a été développée en langages Objective C et Java. SWARM est dédiée aux applications de simulation qui utilisent des systèmes multi-agents. Elle offre des outils dédiés à supporter la conception de simulateurs multi-agents. SWARM ne vise pas un domaine d'applications particulier. Au contraire, elle a été conçue pour pouvoir offrir des outils logiciels assez génériques pour permettre d'implémenter des applications qui utilisent les agents dans la simulation.

- JADE[12] (Java Agent DEvelopment framework) [Bellifemine, 99] est une plate-forme multi-agents développée par le laboratoire TILAB (Telecom Italia LABoratories). Elle est écrite en langage Java, ce qui permet aux développeurs d'implémenter leurs agents en Java pour l'exploiter. Elle fournit un environnement de développement et d'exécution des systèmes multi-agents conformément aux normes de FIPA. Ces composantes de base sont une plate-forme agents et un paquet logiciel pour le développement des agents. Le paquet logiciel comprend : un environnement d'exécution (runtime environnement) où vivent les agents, une bibliothèque de classes qui peut être utilisée par les programmeurs pour développer leurs agents, et un ensemble d'outils graphiques permettant d'administrer et de contrôler les activités des agents. JADE offre un ensemble de services nécessaires à toute application conforme aux standards de FIPA : un gestionnaire des agents (AMS : Agent Management System) qui enregistre et gère les agents, un annuaire de services (DF : Directory Facilitator) qui contient les descriptions des agents et les services qu'ils offrent et un service de gestion des communications (ACC : Agent Communication Channel) qui gère les communications entre les agents.

8. Conclusion

Dans ce chapitre, nous avons présenté une vue globale sur une approche qui est actuellement un champ de recherche très actif, l'approche "Agents". Cette discipline issue du domaine de l'intelligence artificielle, s'intéresse aux notions d'agents et de systèmes multi-agents.
Cette approche est très adaptée à la conception de systèmes où plusieurs éléments indépendants doivent coopérer pour utiliser des ressources distribuées et hétérogènes soit sur un réseau local soit sur un réseau à grande échelle tel que l'Internet.

[11] SWARM website: http://www.swarm.org/index.html.
[12] JADE website: http://jade.tilab.com.

Nous avons aussi cité quelques plates-formes qui permettent d'implémenter des systèmes multi-agents. Ce genre de plates-formes est un ensemble d'outils nécessaire à la construction et à la mise en service d'agents au sein d'un environnement spécifique, et ce genre d'outils peut se trouver sous forme d'environnement de programmation ou d'applications aidant à la programmation d'un système multi-agents.
Certaines plates-formes (comme JADE) sont entièrement implémentées en Java, et répondent aux spécifications FIPA. Elles fournissent un grand nombre de classes qui implémentent le comportement des agents qu'elles créent.

Dans le chapitre suivant, nous présenterons les concepts et les techniques de la réplication et de la cohérence des données dans les systèmes distribués à grande échelle.

Chapitre III.
La réplication dans les grilles de données

1. Introduction

La quantité des données scientifiques produites par des simulations ou rassemblées des expériences à grande échelle est généralement très grande. De telles données ont tendance à être géographiquement stockées à travers des réseaux étendus pour des collaborations à grande échelle. La notion des grilles a été proposée pendant plusieurs années, se concentrant principalement sur l'utilisation efficace de la puissance de calcul globale et l'optimisation de stockages. Cependant, la performance de la grille, où l'utilisation efficace des ressources de stockage est ignorée, sera considérablement dégradée si la majorité des applications exécutées dans la grille sont intensives de données. Pour surmonter ces problèmes, des techniques diverses ont été développées et incorporées dans une infrastructure améliorée nommée "grille de données". La technique de réplication dans les grilles de données est la plus utilisée dans cet environnement. Elle est un des sujets de recherche les plus populaires et une méthode largement acceptée pour améliorer la disponibilité des données et la tolérance aux pannes dans les grilles. D'un autre côté, les approches basées sur les agents peuvent aussi augmenter la disponibilité des données et résister aux défaillances. En effet, certaines techniques introduisant les notions des agents ont été développées et utilisées dans les grilles, afin d'améliorer les performances de la grille.
Dans ce qui suit, nous allons présenter une vue globale sur les techniques et les concepts de la réplication des données à grande échelle. Nous présenterons aussi les concepts sur les systèmes de quorums suivis de quelques travaux connexes sur les quorums et sur l'utilisation des agents pour la réplication dans les grilles de données.

2. Réplication et cohérence dans les systèmes répartis

Tout d'abord, nous allons brièvement présenter les différentes formes de localisation des données dans les systèmes répartis avant d'entamer la notion de réplication.
La localisation d'une donnée peut prendre diverses formes dans le cadre des systèmes répartis. Une donnée peut être centralisée c'est à dire localisée sur un seul site. Une autre approche consiste à partitionner (fragmenter) la donnée. Certains sites possèdent alors une partie de la donnée et il faut donc recomposer ces parties pour connaître la valeur de cette donnée. Une troisième approche, qui est l'objet de cette section, consiste à répliquer la donnée.

2.1. Réplication

La réplication consiste à créer plusieurs copies d'un même objet de telle sorte que l'ensemble de ces copies représente un seul objet logique qui représente l'objet initial avant réplication.
Une telle solution favorise d'une part les lectures qui sont alors toutes locales, et d'autre part la résistance aux défaillances des sites; en effet tant qu'il reste un site possédant une copie de cette donnée, cette dernière est accessible depuis les autres sites.
Dans les systèmes répartis, les deux approches de réplication de références sont les réplications active et passive. Chacune de ces réplications a ses propres avantages et

peut être complémentaire à l'autre. Nous présentons une brève description de ces deux techniques ci-dessous :

a) Réplication Active (active replication ou state machine approach)

C'est une technique générale pour la gestion de réplication qui n'a aucun contrôle centralisé [Schneider, 93]. Toutes les copies de l'objet répliqué jouent le même rôle : elles reçoivent chaque requête, la traitent, mettent à jour leur état et envoient une réponse au client. Puisque les invocations sont toujours envoyées à chaque réplique, l'échec d'une d'entre elles est transparent au client.

b) Réplication Passive (passive replication ou primary/backups replication)

C'est une technique qui désigne une copie comme primaire, et les autres comme secondaires ou de sauvegardes (backups) [Budhiraja, 93]. Les clients envoient leurs requêtes au primaire seulement. La primaire exécute la requête et met à jour les autres copies (de sauvegardes). Ainsi la réplication passive exige l'appui d'application complémentaire pour la primaire pour mettre à jour l'état des autres copies. Si la copie primaire échoue, une des sauvegardes prend la relève.

2.2. Cohérence des données

La réplication est une technique de mise en œuvre qui, comme nous l'avons indiqué, a pour but d'améliorer les performances en lecture et la résistance aux défaillances.
La difficulté de la réplication réside dans la préservation d'un même état pour les différentes copies de la donnée, qui ne doivent en effet pas diverger les unes par rapport aux autres, nous parlons de cohérence mutuelle des copies qui doit être garantie par des protocoles de réplication.
Un protocole de réplication gère la cohérence entre les différentes copies d'un même objet tout en ayant pour objectif d'améliorer les performances du système. Le maintien de la cohérence a un coût. Il faut faire un compromis entre le coût et la qualité de la cohérence. En effet, la performance peut être améliorée au détriment de la fiabilité. Ce sont deux objectifs antagonistes. D'une part, l'obtention d'une bonne fiabilité, nécessite une cohérence forte ce qui pénalise les performances. D'autre part, l'obtention de bonnes performances, nécessite le relâchement de la cohérence, ce qui pénalise la fiabilité.

La difficulté de la réplication réside dans la cohérence mutuelle des copies de chaque donnée. Autrement dit, elle réside dans la mise en œuvre des opérations de lecture et d'écriture. Un protocole de réplication doit garantir que l'utilisateur ne perçoit pas les différentes répliques d'une donnée (transparence). Tout doit donc se passer comme s'il y avait une seule copie, quel que soit le nombre de copies d'une donnée et l'implémentation des opérations de lecture et d'écriture sur ces copies.
Tout protocole de gestion de données à copies multiples doit garantir que toutes les copies d'une même donnée apparaissent à l'utilisateur comme si celle-ci n'était pas répliquée [Raynal, 92].

2.2.1. Modèles de cohérence

Le protocole de cohérence assure l'exécution des opérations des clients et la cohérence mutuelle des copies conformément à un comportement défini par un modèle de cohérence. Il représente une implémentation particulière d'un modèle de cohérence. En littérature, nous trouvons deux principales catégories de modèles de cohérence selon le type de cohérence mis en œuvre : les modèles de cohérence forte (stricte) et les modèles de cohérence faible (relâchée).

a) La cohérence stricte

Elle est réalisée par les algorithmes pessimistes, qui préfèrent réduire la disponibilité au profit d'une cohérence stricte jugée indispensable.

b) La cohérence faible

Elle est réalisée par les algorithmes optimistes qui jugent que l'efficacité et surtout la disponibilité sont beaucoup plus importantes qu'une cohérence stricte (jugée trop pénalisante).

Un grand nombre de protocoles de contrôle de la réplication, basés sur l'un de ces modèles, ont été proposés, témoignant ainsi de l'intense activité de recherche qui règne dans ce domaine. Donc les protocoles (ou algorithmes) proposés sont, généralement classés en deux catégories [Belalem, 07b] [Saito, 05]: les algorithmes pessimistes et les algorithmes optimistes qui différent sur ces points :

- Les algorithmes pessimistes garantissent le maintien de la cohérence (forte) des données en cas de défaillances (panne de machines ou du réseau de communication). Cependant, les algorithmes optimistes peuvent relâcher cette contrainte de cohérence et laisser les copies diverger, afin d'augmenter la disponibilité des données.
- Dans la plupart des cas, les algorithmes pessimistes sont basés sur des algorithmes complexes qui, même en absence de pannes, peuvent s'avérer très coûteux. Ils privilégient la cohérence des données ce qui va réduire leur disponibilité. Contrairement aux algorithmes optimistes qui privilégient la disponibilité et les performances, et par conséquent la cohérence des données ne sera pas garantie.
- Les algorithmes optimistes donnent peu d'importance aux partitionnements du réseau du point de vue des applications, et supposent que ces partitions sont rares, et laissent les accès et les mises à jour s'exécuter librement. Toutefois, il est indispensable d'effectuer un contrôle des copies pour déterminer si un conflit s'est produit. Si c'est le cas, une phase de correction est déclenchée pour résoudre ce conflit. Alors que les algorithmes pessimistes considèrent que les partitionnements du réseau sont dramatiques pour les applications et il faut les traiter rigoureusement.

- Dans les algorithmes pessimistes la synchronisation des accès aux copies est réalisée avant les mises à jour, tandis que dans les algorithmes optimistes cette synchronisation est réalisée après les mises à jour.

Si les modifications sur une donnée sont immédiatement répercutées sur ces copies, la cohérente est dite forte. Et si la propagation des modifications n'est pas visible immédiatement, la cohérence est dite faible. Dans ce cas, certaines modifications concurrentes peuvent être perdues.

3. Caractéristiques des protocoles de réplication

Afin de pouvoir éclaircir les principes des protocoles de réplication, nous présentons dans cette section leurs principales caractéristiques jugées importantes.

3.1. Nombre de copies concernées par une requête

Chaque protocole de réplication a ses propres contraintes sur le nombre de copies à consulter avant de pouvoir répondre à une requête de lecture ou d'écriture. Par exemple, certains protocoles écrivent sur toutes les copies avant de répondre à une requête d'écriture, et ils n'ont besoin de consulter qu'une seule copie pour répondre à une requête de lecture. D'autres protocoles, qui utilisent le principe du numéro de version, ne répondent à une requête d'écriture que lorsque plus que la moitié des copies sont mises à jour. Et pour répondre à une requête de lecture, il est suffisant de consulter exactement la moitié des copies [Drapeau, 03].

3.2. Détermination des copies concernées par la mise à jour

Parmi les copies d'un objet, il existe celles qui peuvent être modifiées par des requêtes, et celles qui peuvent être modifiées uniquement par le protocole de réplication. La détermination de ces copies, définie par deux approches extrêmes [Drapeau, 03], est un point essentiel : l'approche maître-esclaves ou primaire-secondaire, et l'approche copies identiques.

a) Approche maître-esclaves

Chaque objet répliqué possède une copie dite maîtresse, les autres sont des copies esclaves. Une requête d'écriture ne peut être effectuée que sur la copie maîtresse, qui diffusera à son tour les modifications aux copies esclaves.
Dans la figure III.1.a la copie maîtresse (primaire) reçoit les requêtes d'écriture, puis les diffuse aux copies esclaves (secondaires). Les copies esclaves ne répondent qu'aux requêtes de lecture. Cette approche facilite le maintien de la cohérence mais elle produit des goulots d'étranglement sur la copie maîtresse. Des variantes proposent qu'il y ait plusieurs copies maîtresses en même temps pour éliminer les goulots d'étranglement. Elles permettent d'améliorer les performances et la tolérance aux fautes.

b) Approche copies identiques

Dans cette approche, le comportement de toutes les copies est identique (comportement de copie maîtresse). Toutes les copies peuvent répondre aux requêtes de lecture et d'écriture. Chaque copie qui traite une requête d'écriture, diffusent les mises à jour aux autres copies.
Dans la figure III.1.b les copies 1, 2, 3 et 4 traitent les requêtes d'écriture et diffusent ensuite aux autres copies les mises à jour. Le principal point négatif de cette approche est que des conflits peuvent se produire suite à des écritures simultanées sur des copies différentes d'un même objet.

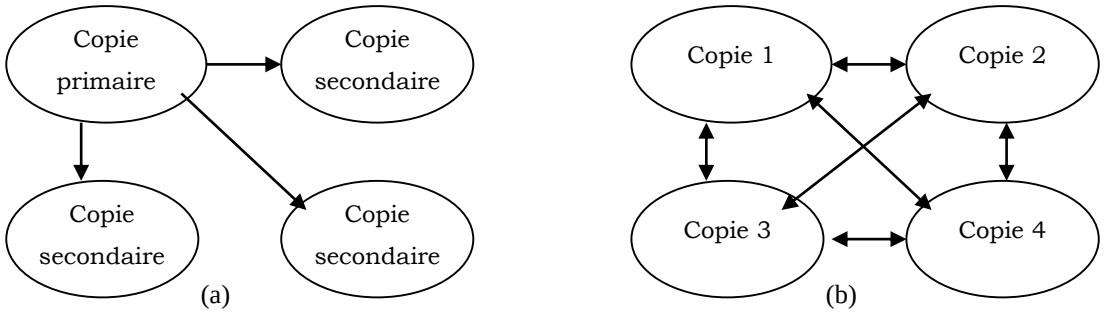

Figure III.1 : Approches des mises à jour : maître-esclaves et copies identiques.

3.3. Moment de la synchronisation

Les mises à jour des différentes copies peuvent se faire simultanément sur toutes les copies ou d'abord sur une copie et ensuite sur les autres. Dans [Drapeau, 03], les auteurs proposent, sous le terme de "condition de cohérence", une classification de sept différents moments de déclenchement de la synchronisation :

a) Conditions sur le délai.

Elles expriment la durée maximale que le protocole peut attendre avant la propagation d'une mise à jour. Par exemple, pour un délai maximum de 60 secondes, toutes les mises à jour d'une copie doivent être propagées vers une autre copie avant l'expiration de celui-ci.

b) Conditions sur la périodicité.

Ces conditions portent aussi sur le temps. Elles expriment qu'une copie d'un objet doit être mise à jour avec la dernière valeur de l'objet toutes les *n* unités de temps, que l'objet ait été modifié ou non. Cette approche est avantageuse que la première du fait de l'écartement des pertes de messages de mise à jour dues au réseau ou aux pannes des nœuds.

c) Conditions sur le moment.

Ces conditions sont un cas particulier des conditions de périodicité. Elles expriment qu'une copie d'un objet doit être mise à jour avec la dernière valeur de l'objet à un moment. Par exemple tous les jours à 8h.

d) Conditions sur la version.

Ces conditions spécifient le nombre de modifications pouvant avoir lieu sur une copie avant qu'une autre copie soit mise à jour.

e) Conditions numériques.

Ces conditions portent sur la valeur d'un objet si elle est numérique. Elles permettent de limiter l'écart entre les valeurs des différentes copies. En connaissant la valeur de deux copies, nous pouvons vérifier ces conditions en calculant, par exemple, les différences absolues, relatives ou exprimées en pourcentage.

f) Conditions sur les objets.

Ces conditions portent sur la structure des objets. Trois types de conditions sont définis. x' doit être mise à jour avec la dernière valeur de x lorsque :

- au moins i sous objets de la copie x ont été modifiés,
- au moins i % des sous objets de x ont été modifiés ou
- le sous objet a de x a été modifié, depuis la dernière mise à jour de x'.

g) Conditions d'événements.

Ces conditions portent sur les événements de déclenchement des mises à jour des copies. Les conditions précédentes peuvent être exprimées par un modèle d'événements assez riche.

3.4. Initiative de la mise à jour

Deux approches existent pour le rafraîchissement des copies : soit les copies sources sont les initiatrices des propagations des mises à jour, soit les copies cibles demandent les mises à jour aux copies sources.

a) Rafraîchissement à la poussée ("push").

La propagation des mises à jour à l'initiative de la copie source vers toutes les copies cibles. Des problèmes de passage à l'échelle peuvent être posés dans certains cas, où des messages inutiles de synchronisation peuvent être envoyés à des copies qui ne seront pas consultées.

b) Rafraîchissement à la demande ("pull").

La propagation des mises à jour à l'initiative de la copie cible. Cette approche permet de réduire la charge du réseau en ne propageant que les dernières modifications. En revanche, les copies cibles ne peuvent pas savoir si une mise à jour est nécessaire. En plus si elles interrogent trop souvent la copie source cette approche peut s'avérer inintéressante.

3.5. Capture des mises à jour

La capture est utilisée pour détecter et sélectionner les changements sur une copie afin de les propager aux autres copies. Une façon simple de faire, consiste à consulter

la copie afin de connaître son dernier état. Une deuxième façon de faire, consiste à enregistrer les modifications dans un journal (log) ou une copie ombre (shadow). Une troisième façon consiste à utiliser les triggers (déclencheurs) où la modification d'une donnée répliquée le déclenche.

3.6. Topographie de mises à jour

Différents protocoles basés sur différentes topographies de propagation des mises à jour entre les copies peuvent existés. Ces protocoles se justifient quand des copies doivent être mises à jour avant d'autres ou qu'il est avantageux de s'appuyer sur la topographie du réseau. Nous pouvons ainsi imaginer différentes topographies de propagation des mises à jour entre les copies. Des protocoles peuvent propager une mise à jour d'une copie en la diffusant à toutes les autres copies (figure III.2.a). D'autres protocoles propagent les mises à jour de copie en copie (figure III.2.b), ou vers un ensemble de copies où chacune d'elles les propage à son tour vers un autre ensemble (figure III.2.c). Certains protocoles construisent des chemins particuliers entre les copies que doivent suivre les mises à jour (figure III.2.d).

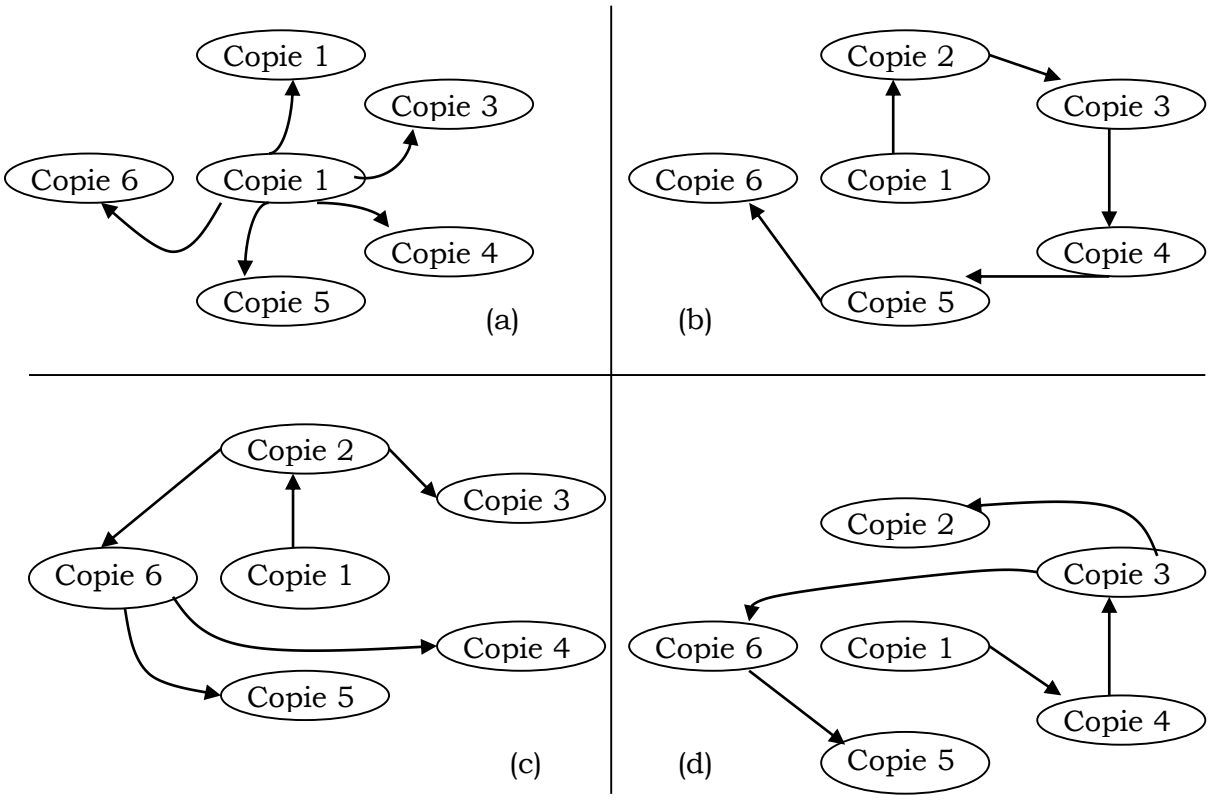

Figure III.2 : Différentes topographies de mise à jour.

3.7. Gestion des fautes

Généralement la tolérance aux fautes dans un protocole de réplication est réaliser en deux phases : la détection des fautes et la récupération [Drapeau, 03]. Certains protocoles de réplication sont en état de supporter les fautes survenant sur les copies et d'autres non. Quatre types de fautes sur les copies peuvent survenir : les fautes par arrêt, les partitions réseau, les fautes par valeur et les fautes byzantines.

Une fois qu'une faute est détectée, la copie soupçonnée est supprimée du groupe des copies gérées par le protocole jusqu'à ce qu'elle retrouve un comportement normal. Avant de réintégrer une copie ayant à nouveau un comportement normal au groupe, il est nécessaire qu'elle rattrape son retard sur les autres copies.

3.8. Degrés de réplication et leurs emplacements

Il est aussi nécessaire de décider du nombre de copies qui sont nécessaires dans le système, à quels moments en créer de nouvelles, et où les placées.
Cette gestion des copies peut être statique ou dynamique.

3.9. Transparence à la réplication

Le protocole de réplication peut offrir plus ou moins de la transparence. Si un protocole n'offre pas la transparence à la réplication, chaque copie sera manipulée explicitement. Nous parlons alors d'objets physiques. Par contre si la transparence est offerte à la réplication, Nous n'aurons pas conscience du fait que les objets sont répliqués. Cette abstraction permet de référencer l'ensemble des copies d'un même objet sous une seule désignation [Drapeau, 03].

4. La réplication à base des quorums

Dans les systèmes répartis, la cohérence mutuelle des différentes copies d'une même donnée est gérée par les protocoles de réplication. Certains protocoles de réplication sont basés sur les quorums. Nous présentons dans ce qui suit le concept de quorum, nécessaire pour la suite dans cette thèse.

En littérature, le quorum est défini comme le nombre de votants nécessaire pour qu'une élection soit valable. Ainsi, puisque chaque donnée à plusieurs copies, nous pouvons appliquer ce principe pour pouvoir manipuler ou consulter une donnée. Donc, l'idée de base est d'avoir l'autorisation de plusieurs copies (quorum) avant de réaliser et valider une opération de lecture ou d'écriture sur une donnée. Par exemple, pour une donnée ayant n copies, nous pouvons spécifier qu'il faut l'autorisation de plus de $n/2$ copies pour s'assurer qu'une modification sera toujours faite sur une majorité de copies.
Dans ce qui suit, nous allons présenter les règles de lecture et d'écriture sur les copies multiples d'une donnée en appliquant le principe du quorum.

La difficulté de la réplication, comme nous l'avons indiqué, réside dans la mise en œuvre des opérations de lecture et d'écriture. Une donnée x peut être accédée par des opérations de lecture et d'écriture issues de plusieurs sites et éventuellement simultanées. Il s'agit donc en fait de résoudre le problème des lecteurs-rédacteurs sur les différentes copies de la donnée x.

Pour toute opération de lecture et d'écriture sur la donnée x, deux règles (règles de lectures et d'écritures) sont associées aux lecteurs-rédacteurs [Raynal, 92]:

R1 : toute exécution d'une opération d'écriture sur la donnée *x* exclut toute autre exécution simultanée d'une opération de lecture ou d'écriture sur la donnée *x*.

***R2* :** toute exécution d'une opération de lecture sur la donnée *x* peut être simultanée avec d'autres exécutions d'opérations de lecture sur la donnée *x*.

Ces deux règles garantissent la cohérence de la donnée *x*. Elles ne précisent que la synchronisation qui doit exister entre les opérations de lecture et les opérations d'écriture pour que la donnée *x* puisse être cohérente. Plusieurs mises en œuvre de ces deux règles sont possibles dans le cadre des systèmes centralisés. Elles reposent sur des extensions de solutions au problème de l'exclusion mutuelle. Dans le contexte de la réplication, il faut également préciser une troisième règle qui représente la cohérence atomique et qui donne la signification de la valeur rendue en résultat par une lecture.

R3- la valeur rendue par une opération de lecture sur la donnée *x* est la dernière valeur qui a été écrite sur la donnée *x*.
« Un modèle de cohérence est atomique si une opération de lecture sur une donnée retourne la valeur affectée par la dernière opération d'écriture sur cette donnée » [Drapeau, 03].

4.1. Quorums de lecture et d'écriture

Supposons qu'un client veut effectuer une opération de lecture sur la donnée *x*. Il demande à un certain nombre de copies (ou de nœuds puisqu'il y a une copie par nœud) leur permission. Les permissions seront restituées une fois que le client termine l'opération de lecture, car les copies ayant autorisé le client sont considérées comme verrouillées. Nous considérons que chaque copie n'est dotée que d'une seule permission (attribution uniforme). Les permissions, comme nous l'avons indiqué, sont assimilables à des votes. Nous parlons alors d'un système de votes et nous disons que l'opération de lecture exige "r" votes, ou nous pouvons dire aussi que le quorum de lecture vaut "r". C'est de même pour les écritures, il faut que toute écriture sur la donnée *x* requiert "w" votes pour pouvoir s'exécuter.
Avec $R \cap W \neq \varnothing$.
Cette formule garantit qu'il existe au moins une copie appartenant à l'intersection de tout quorum de lecture *R* et de tout quorum d'écriture *W*. D'où nous concluons que tout quorum de lecture *R* comprend toujours une copie dotée de la dernière valeur qui a été écrite (copie courante).

Le respect de la règle *R3* pourra être implémenté en utilisant les numéros de version. Où chaque copie lui est associée un numéro de version qui sert à indiquer son actualité. La copie dotée du plus grand numéro de version sera la copie à jour ou la copie courante. Il peut y avoir plusieurs copies dotées du même numéro de version, comme il peut y avoir plusieurs copies dotées des numéros de version différents.
L'opération de lecture sur la donnée *x* retourne une copie, parmi les *r* copies auxquelles elle a demandé la permission de lire, dotée du plus grand numéro de version. L'opération d'écriture sur la donnée *x* met à jour les w copies et leurs associe

le nouveau numéro de version. Ce dernier est égal au plus grand numéro de version, obtenu des w copies auxquelles l'opération a demandé la permission d'écrire, plus 1.

Un exemple illustratif sur les quorums de lecture et d'écriture est présenté par la figure III.3, où le nombre de copies d'une donnée est *12.* Dans la figure III.3.a, puisque le nombre de quorum d'écriture *w=10,* il est impossible d'effectuer deux opérations d'écritures simultanément. Par contre dans la figure III.3.b, il peut y avoir un conflit entre deux écritures (exécution simultanée de deux écritures), puisque chaque écriture peut acquérir les 6 permissions nécessaires (*6+6 =12*). Dans la figure III.3.c, nous pouvons lire d'une seule copie mais il faut avoir toutes les permissions pour écrire.

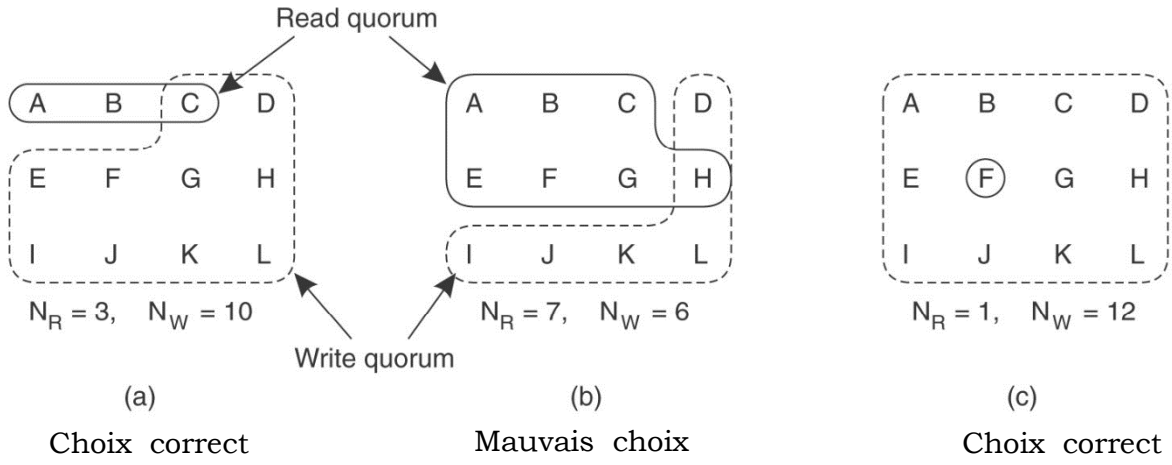

Figure III.3 : Exemple illustratif sur les quorums de lecture et d'écriture[13].

5. Les protocoles basés sur les quorums

L'objectif de ces protocoles de réplication des données, est de réduire le nombre de copies (taille du quorum) nécessaires pour accéder (en lecture ou en écriture) à une donnée tout en offrant une haute disponibilité, et en garantissant la cohérence des répliques des données.
De nombreux protocoles basés sur les quorums existent (le Quorum à Consensus, le Quorum à Arbre, le Quorum à Grille, etc.), et qui sont principalement destinés aux systèmes de base de données [Abawajy, 14]. Ces protocoles ainsi que le protocole ROWA (Read One Write All) sont considérés comme les plus populaires. Ils se distinguent les uns des autres principalement par le nombre de répliques impliquées dans l'exécution d'une opération de lecture ou d'écriture.

5.1. ROWA

Le protocole ROWA est un protocole simple [Bernstein, 84]. Comme son nom l'indique, il lui suffit qu'une seule réplique pour lire une donnée, mais il exige l'écriture sur toutes les répliques d'une donnée.

[13] http://www.inf.ufpr.br/aldri/disc/slides/SD712_lect12.pdf.

Le point fort de ce protocole est la fiabilité des opérations de lecture. Ses avantages sont les faibles coûts et la haute disponibilité pour les lectures. Cependant, ses inconvénients sont les coûts élevés et la non tolérance aux pannes pour les opérations d'écriture, puisque toutes les répliques doivent être mises à jour pour chaque opération d'écriture, (si la mise à jour d'une seule copie échoue toute l'opération d'écriture échoue).

5.2. Quorum à Consensus

Le principe du Quorum à Consensus (connu aussi sous le nom du vote majoritaire) se base sur le suffrage (le vote) [Thomas, 79]. Pour aborder le principe de ce protocole, nous considérons les notations suivantes :
Soient :
n : le nombre des copies d'une donnée,
Qr : quorum de lecture,
Qw : quorum d'écriture,
Pour ce protocole de vote majoritaire, la mise en œuvre des règles de lectures et d'écritures *R1* et *R2* (décrites dans la section 4), définissant les contraintes d'exclusion qui garantissent la cohérence du contenu de la donnée x, peut être faite en imposant des contraintes sur les quorums de lecture Qr et d'écriture Qw associés à la donnée x [Raynal, 92] (les quorums de lecture et d'écriture doivent satisfaire les contraintes suivantes):

C1 : $Qr + Qw > n$.
C2 : $2Qw > n$.

La première contrainte garantit l'exclusion entre une opération de lecture et une opération d'écriture sur la même donnée, alors que la deuxième contrainte garantit l'exclusion entre deux opérations d'écritures sur la même donnée.
Ces deux contraintes précisent que l'exécution simultanée d'une opération d'écriture et d'une autre opération ne peut se faire, car il faudrait pour cela plus de vote qu'il n'en existe.
En plus, le respect de la première contrainte garantie que le quorum de lecture Qr comprend toujours une copie courante.

En cas d'écriture, ce protocole améliore les coûts et offre une plus grande disponibilité par rapport à ROWA. Cependant, en cas de lecture les coûts sont plus élevés à cause du nombre de copies impliquées.

5.3. Quorum à Arbre

Ce protocole est basé sur une structure arborescente logique, les répliques sont logiquement disposées en une structure arborescente [Agrawal, 90]. Une opération d'écriture exige un quorum formé de la racine, la majorité de ses enfants, la majorité des enfants de chacun de ces enfants, et ainsi de suite. Par conséquent, deux opérations d'écriture simultanées ont au moins une réplique commune à chaque niveau de l'arbre. Le quorum de lecture est formé de la racine. Si la racine n'est pas disponible, le quorum sera formé de la majorité de ses enfants. Si un nœud de cette

majorité d'enfants est indisponible, il sera remplacé par la majorité de ses enfants et ainsi de suite.

5.4. Quorum à Grille

Dans ce protocole, les répliques sont logiquement disposées dans structure à grille [Cheung, 92]. La forme la plus simple du Quorum à Grille est la grille rectangulaire. Les n répliques d'une donnée sont organisées logiquement dans une grille rectangulaire de dimension de r lignes et c colonnes ($n=r*c$). Un quorum de lecture Qr est formé d'une copie de chaque colonne ($Qr=c$). Un quorum d'écriture Qw requiert une colonne entière et une copie de chaque colonne restante ($Qw=r+c-1$).

5.5. Votes pondérés

Ce protocole est dérivé du protocole du Quorum à Consensus. Dans le but de réduire les coûts des mises à jour et des communications, des valeurs non négatives peuvent être attribuées aux nœuds possédant une copie [Gifford, 79]. Ces valeurs représentent les poids de chaque nœud dans l'ensemble du système. Un nœud considéré comme bon aura une valeur plus élevée qu'à celle d'un autre nœud considéré comme moins bon.
Le Quorum à Consensus (décrit dans la section 5.2) est équivalent à l'attribution uniforme d'un vote à chaque copie. Il est possible de le généraliser en utilisant des votes pondérés, en attribuant au nœud possédant une copie i (ou simplement à la copie i) un vote de poids $P_i \geq 0$.
Les votes pondérés permettent de particulariser les nœuds (les copies) selon des critères liés à la charge, à la résistance aux défaillances, aux partitions réseaux, etc.

Soit :
n : le nombre des copies d'une donnée,

$\forall \quad 1 \leq i \leq n \qquad\qquad poidtotal = \sum_{i=1}^{n} P_i$

$$poidmaj = \left\lceil \frac{poidtotal + 1}{2} \right\rceil$$

Les deux contraintes *C1* et *C2* (décrites dans la section 5.2) deviennent :

C1' : $Qr + Qw > poidtotal$

C2' : $Qw \geq poidmaj$

6. La réplication dans les grilles de données

Bien que les grilles de données se concentrent sur la gestion de données, l'infrastructure de la grille d'aujourd'hui peut coupler la planification des tâches avec la planification de la réplication, comme elle peut les découpler. Un préplacement des répliques basé sur des modèles d'utilisation a été proposé, pour prévoir les exigences des utilisateurs pour les placements des tâches et aussi pour les ensembles de

données. Dans [Ranganathan, 02 b], des algorithmes et des concepts, pour un planificateur de réplication découplé, sont présentés pour adresser les placements. La prémisse est de découpler les besoins des réplications en ressources des besoins des applications en ressources. Cela exige alors aux applications de la grille d'envoyer les tâches aux planificateurs externes. Les planificateurs externes informent alors les planificateurs d'ensemble de données des ensembles à utiliser, ce qui permet à ces derniers (planificateurs d'ensemble de données) de créer automatiquement des répliques à travers la grille.
Donc un des services d'optimisation essentiels d'une grille de données est le service de réplication de données. Ce service cherche à améliorer le trafic du réseau en copiant des données massivement accédées à des emplacements appropriés et gérant leurs utilisations. Le mécanisme de réplication, en plus de la gestion des différentes répliques, essaye de déterminer quand, où et quelles données reproduire à travers la grille. D'où la nécessite d'avoir un système de gestion de réplication décentralisé.
Ces trois dimensions de ce que, où et quand reproduire des données sont complètement déterminées par la politique de la grille. Cette politique peut certainement être influencée par plusieurs facteurs environnementaux. Nous pouvons citer : les modèles d'accès aux données, les performances des réseaux, l'environnement des applications, etc. Dans ce contexte, nous allons présenter quelques concepts liés à la réplication à grande échelle.

6.1. Propriétés désirables pour un système de réplication dans les grilles de données

Nous avons mentionné précédemment que les principaux buts de la réplication sont la disponibilité des données et la tolérance aux pannes. Dans ce qui suit, nous présenterons l'ensemble des propriétés désirées d'un système de réplication, qui peut être utilisé pour évaluer les solutions proposées :

- **Dynamisme**

  La réplique peut être créée et supprimée dynamiquement quand des besoins surgissent.

- **Sûreté**

  La réplique doit être créée solidement tel qu'aucune fuite d'information à un tiers et aucune corruption de données n'arrivent.

- **Efficacité**

  La réplique doit être créée dans un temps et avec une quantité de ressources raisonnables.

- **Adaptabilité**

  Le processus de réplication doit manipuler les diverses vitesses des réseaux, un environnement de stockage hétérogène, les diverses vitesses de traitement et les défaillances.

- **Flexibilité**

  La réplique doit être capable de joindre et laisser la grille quand il est nécessaire.

- **Choix de la réplique**

  Dans un environnement où des répliques multiples sont disponibles, l'existence d'un service de sélection de réplique est nécessaire pour choisir la réplique qui peut correspondre aux exigences de la qualité de service.

- **Cohérence**

  Différents degrés de cohérence doivent être fournis dans un environnement où les mises à jour des répliques sont nécessaires.

- **Scalabilité**

  Le système de réplication doit être capable de manipuler un grand nombre de répliques et la création simultanée des répliques.

6.2. Les modèles d'accès

Les emplacements où les répliques sont placées affectent l'efficacité de la stratégie de réplication. La localisation joue un rôle essentiel lors de la réplication, ce qui contribue à améliorer le temps d'accès global et la disponibilité des données dans l'environnement de la grille [Ranganathan, 01].
Quand les clients demandent des données, les modèles de ces demandes, ou les modèles d'accès, peuvent présenter des propriétés de localité diverses :

- **Localité temporelle**

  Les données récemment accédées vont probablement être accédées de nouveau.

- **Localité géographique (localité des utilisateurs)**

  Les données récemment accédées par un utilisateur vont probablement être accédées par les utilisateurs voisins.

- **Localité spatiale (localité des données)**

  Les données près d'une donnée récemment accédée vont probablement être accédées.

7. Le processus de réplication

Le but principal d'utiliser la réplication est de réduire la latence d'accès et la consommation de la largeur de bande. La réplication peut aussi aider dans l'équilibrage des charges et peut améliorer la fiabilité en créant des copies multiples des mêmes données. La réplication statique peut être employée pour réaliser quelques bénéfices mentionnés ci-dessus, mais son inconvénient est qu'elle ne peut pas s'adapter aux changements dans le comportement des utilisateurs (et par conséquent le changement de l'environnement).
Dans la grille de données, les tailles des données s'élèvent aux Téraoctets et le nombre des utilisateurs s'élèvent aux milliers voire même aux millions dans le monde entier. Dans un tel environnement, la réplication statique ne semble pas appropriée, d'où la

nécessité d'avoir des stratégies de réplication dynamiques (la création, la suppression et la gestion des répliques s'effectuent automatiquement) qui ont la capacité de s'adapter aux changements de la grille.

N'importe quelle stratégie de réplication doit répondre à quatre questions fondamentales :

1) Quand les répliques doivent être créées ?
2) Où les répliques doivent être placées ?
3) Quelles répliques doivent être sélectionnées ?
4) Comment répliquer ?

Ces considérations sont complètement déterminées par la politique de la grille. Différentes stratégies de réplications peuvent résulter en réponse à ces questions.

7.1. Moment de création des répliques

La création des répliques est un aspect important dans une stratégie de réplication dynamique. Le processus de création de réplique peut être divisé en deux sous processus :

a) Vérifier si de nouvelles répliques sont nécessaires.
b) Créer effectivement de nouvelles répliques.

Ces décisions sont prises en se basant sur les coûts et les bénéfices de la création des nouvelles répliques.

Une approche suggérée dans [Ranganathan, 02 a], pour la découverte de "quand de nouvelles répliques sont nécessaires ?", est basée sur des contrôles périodiques. Cette périodicité peut être changée selon le statut du réseau. Conformément aux derniers contrôles effectués, l'intervalle de temps des contrôles peut s'accroître s'il n'y a aucun changement du statut du réseau, et vice versa, l'intervalle de temps peut s'amoindrir s'il existe des changements.
L'inconvénient de cette méthode est que des contrôles non nécessaires sont à chaque fois effectués sur des données moins populaires.
Une approche alternative est de vérifier le statut de réplication d'une donnée seulement quand la donnée est demandée. Par exemple, chaque site peut consulter l'historique des demandes des données dont il possède (données stockées sur ce site), et quand le nombre des demandes excède un seuil, le système peut exécuter le contrôle ou créer simplement une réplique. Cette approche réduit le nombre des contrôles inutiles et améliore les performances du système.

7.2. Placement des répliques

Un autre aspect important dans une stratégie de réplication, que nous examinerons, est où placer les répliques ? Un placement différent d'une réplique a un effet différent sur l'amélioration du système [Bouharaoua, 07].
Généralement, les stratégies de réplication placent la nouvelle réplique suivant le coût global relatif au placement ou suivant la qualité de service offerte par ce nouveau placement. Elles estiment le coût de placement de la réplique ou la qualité de service de chaque site appartenant à l'ensemble des sites candidats, pour déterminer le site

le plus approprié au placement de la réplique. L'ensemble des sites candidats est l'ensemble des sites potentiels qui répondent à ces critères :

1) Le site ne contient pas de copie de la donnée à répliquer.
2) La capacité de stockage du site est suffisante.
3) Le temps de transfert entre le site et les utilisateurs potentiels est raisonnable.

7.3. Sélection des répliques

Un autre aspect d'une stratégie de réplication, est le choix des répliques. La sélection des répliques est utilisée pour deux objectifs différents qu'il faut distinguer :

- d'une part elle est utilisée pour répondre aux besoins d'un utilisateur,
- et d'autre part, elle est utilisée pour désigner la copie, parmi l'ensemble des copies, à être reproduite.

Dans le cas du premier objectif, et après la découverte d'une liste de répliques, le choix de la meilleure réplique, qui répondra aux besoins des utilisateurs, est effectué soit par les applications des utilisateurs soit par le middleware (la stratégie elle-même). Ainsi, la sélection d'une réplique est définie comme le processus de choix d'une réplique parmi l'ensemble des répliques disponibles, basé sur les performances (temps de transfert, QoS, etc.) et les caractéristiques d'accès aux données.
Dans le deuxième cas, où plusieurs copies de la même donnée existent, le choix de la copie candidate (la copie à être reproduite) est basé sur le coût de stockage. Généralement, la copie candidate est une copie récente (cohérente) qui se trouve dans le site le plus proche du site où la nouvelle copie sera placée.
Certaines stratégies n'effectuent pas de sélection. Toutes les copies sont reproduites à partir de la copie originale.

La sélection d'une réplique, dans certaines stratégies, se base sur la prévision des performances et de l'état du réseau [Vazhkudai, 02]. Cette prévision peut être réalisée par le service d'état du réseau *NWS* (Network Weather Service) [Wolski, 98].

7.4. La manière de répliquer

Pour répondre à la question de comment répliquer, le processus de création effective d'une réplique dépend de la structure et de l'état de l'objet à répliquer. La structure de l'objet peut être indivisible ou composée (fichier simple, un objet lié à ses références, base de données relationnelle, etc.), tandis que l'état de l'objet peut être constitué de code, de données et éventuellement d'un état (contexte) d'exécution.

8. Classification des stratégies de réplication

Dans la littérature, plusieurs classifications des stratégies de réplication existent. Elles sont classifiées selon plusieurs critères. Elles peuvent être classées selon l'architecture de la grille adoptée. Chaque architecture de grille de données différente a des propriétés différentes, et ces propriétés nécessitent différentes stratégies de réplication. Il existe des stratégies dédiées aux grilles hiérarchiques et d'autres dédiées aux grilles pair à pair ou hybrides, etc. [Tos, 15].
Une autre classification des stratégies se base sur le critère de périodicité, qui définit le moment du déclenchement du processus de réplication. Nous distinguons des

stratégies de réplication périodiques (déclenchées à chaque intervalle de temps) et des stratégies non périodiques (déclenchées à la demande).
Une autre possibilité de classification se base sur la prise de décision. Nous distinguons deux classes de stratégies : centralisées et décentralisées.
Dans ce qui suit, nous présentons les classifications des stratégies de réplication existantes [Tos, 15]. Bien que chaque stratégie de réplication soit différente, elles peuvent avoir des caractéristiques communes en ce qui concerne certains aspects.

8.1. Réplication statique ou dynamique

Les stratégies de réplication peuvent être classées en deux groupes, à savoir la réplication statique et dynamique. Dans la réplication statique, toutes les décisions concernant la stratégie de réplication sont prises avant que le système ne soit opérationnel et resterons fixes tout au long de son fonctionnement [Cibej, 05] [Xiong, 13]. Dans la réplication dynamique, "quoi, quand et où répliquer ?" sont décidés en réponse aux changements (modifications) effectués sur la grille de données [Chang, 08] [Nicholson, 08].
Dans un environnement statique, où les nœuds ne peuvent pas rejoindre ou quitter la grille, la réplication statique peut être intéressante. Dans un environnement dynamique où les nœuds sont libres de rejoindre ou de quitter la grille, et où les modèles d'accès aux données changent avec le temps, la réplication dynamique est bien appropriée en raison de sa nature d'adaptabilité.

8.2. Réplication centralisée ou décentralisée

Chaque étape dans la réplication des données, nécessite d'avoir des informations sur l'état de l'environnement de la grille. Sur la base de ces informations, qui seront éventuellement traitées et analysés, des décisions sur la réplication seront prises. Toutes ces tâches (collecte, traitement et mesure à prendre envers la réplication) peuvent être sous le contrôle d'une seule entité ou plusieurs entités. Une classification peut être faite sur la base de ce contrôle [Amjad, 12] [Ma, 13].
Dans les stratégies de réplication centralisée, un nœud central contrôle tous les aspects de la réplication de données. Toutes les décisions de réplication sont prises par ce nœud. En revanche, dans l'approche décentralisée, il n'existe aucun mécanisme de contrôle central. Les nœuds eux-mêmes décident de la manière dont la réplication va se produire.
Chaque approche a ses avantages et ses inconvénients. La réplication centralisée est plus facile à mettre en œuvre et généralement plus efficace, car une seule entité est responsable de toutes les décisions et possède des connaissances sur tous les aspects de la grille de données. D'autre part, l'autorité centrale est également un point de défaillance et n'est donc pas idéale pour la fiabilité et la tolérance aux pannes. La réplication décentralisée est bonne pour la fiabilité car il n'y a pas de point de défaillance unique dans le système.
En l'absence d'un contrôle central, aucun nœud ne peut contenir des informations complètes sur l'intégralité de la grille de données. Dans une stratégie de gestion de réplication décentralisée, la coordination d'un événement de réplication est généralement effectuée avec la collaboration d'un certain nombre de nœuds.

8.3. Classification basée sur une fonction objective

Dans certaines stratégies, la réplication de données est vue comme un problème d'optimisation d'une fonction objective. Considérant que chaque stratégie de réplication de données vise à minimiser ou à maximiser certains objectifs, il est possible de faire une classification par rapport à la définition de cette fonction objective [Mokadem, 15].
Une approche populaire pour définir une fonction objective consiste à définir des objectifs de localité de données [Tang, 05]. Dans cette approche, l'objectif principal est de placer les répliques aussi près que possible des demandeurs (de préférence localement).
Pour ne pas augmenter la localité de toutes les données demandées, certaines stratégies modifient leur objectif de localité en utilisant des heuristiques pour augmenter sélectivement la localité des seules données populaires [Shorfuzzaman, 10].

D'autres fonctions objectives sont basées sur un modèle de coût [Andronikou, 12] [Mansouri, 13]. La décision de réplication dépend de ce modèle de coût, qui peut inclure de nombreux paramètres à prendre en compte, notamment des statistiques d'accès aux données, la disponibilité, la bande passante, la taille des données, la latence du réseau, la capacité du stockage, etc.
Le problème de placement des répliques est un problème NP-complet. C'est un problème de minimisation d'une fonction de coût, selon un certain nombre de contraintes (bande passante, capacité de stockage, capacité de calcul, charge supportée, nombre de répliques, temps de réponse, taux de disponibilité, etc.) [On, 03].

Les mécanismes du marché et les comportements économiques sont également évalués [Belalem, 08] [Goel, 07]. Dans les modèles économiques, les données sont traitées comme des biens échangeables sur le marché. Lors d'un événement de réplication, les clients ont tendance à acheter des données à partir de sites distants offrant le prix le plus bas et les sites distants essaient de vendre leurs données pour le plus grand profit.

8.4. Réplication par poussée (push) ou à la demande (pull)

Dans toute stratégie de réplication, deux acteurs principaux sont impliqués dans la réplication d'un objet. Le premier est le serveur qui contient l'objet, et le second est le client qui a besoin de cet objet localement. La classification par push ou par pull se concentre sur lequel de ces deux acteurs déclenche l'événement de réplication [Dogan, 09] [Steen, 10].
Dans la réplication basée sur un push, l'événement de réplication est déclenché par l'expéditeur qui pousse l'objet vers le nœud demandeur. Les serveurs reçoivent des demandes d'un certain nombre de clients, ils nécessitent donc suffisamment d'informations sur l'état du système pour pouvoir déclencher des événements de réplication.
Dans la réplication basée sur un pull, l'événement de réplication est déclenché par le nœud demandeur. La mise en cache côté client est également considérée comme une

réplication par pull, du fait que dans cette forme de mise en cache, les clients décident de stocker temporairement et localement un objet [Steen, 10].
Puisque l'événement de réplication est réalisé à la demande, les réplications basées sur le pull peuvent être considérées comme des approches réactives, par rapport aux réplications basées sur les push qui sont souvent proactives.

9. Travaux connexes

Les stratégies de réplication dans les grilles de données est un domaine de recherche très actif [Andronikou, 12], [Nukarapu, 11]. Deux domaines, liés aux problèmes de la réplication des données, font l'objet de plusieurs travaux de recherche : les domaines liés aux problèmes de création de sélection et de placement des répliques, et les domaines liés aux problèmes de la gestion de la cohérence des répliques.

Plusieurs travaux sur le placement des répliques, sur le nombre optimal des répliques et sur la sélection des répliques ont été proposés. Le plus souvent, ces stratégies ne considèrent que les cas de lectures seules lors des accès aux données. Par conséquent, elles ne sont pas adaptées pour les applications qui accèdent aux données en mode écriture [Perez, 10]. Et peu d'études sur la cohérence des répliques dans les grilles de données ont été effectuées [Grace, 14] [Mamat, 08].

Dans cette section, nous présentons quelques travaux sur les protocoles de réplication des données où nous nous intéressons principalement aux travaux basés sur les quorums. Ensuite nous présenterons d'autres travaux basés sur les notions d'agents dans les grilles.

9.1. Systèmes de quorums

Afin d'améliorer les performances d'un protocole de réplication de données, il est important de ne pas négliger quelques facteurs tels que : le nombre de répliques utilisées, les sites où les répliques seront déposées et comment maintenir la cohérence de ces répliques.
Dans ce qui suit, nous présentons quelques travaux qui ont proposé des protocoles de réplication de données basés sur les quorums et qui prennent en considération les facteurs cités ci-dessus pour atteindre leurs objectifs.

Dans [Abawajy, 14] les auteurs proposent un nouveau protocole de réplication de données basé sur les quorums dans le but de réduire le coût des mises à jour des données, de fournir une haute disponibilité et de maintenir la cohérence des données, avec des bons temps de réponse et des petites tailles de quorums de lecture et d'écriture. Ils ont développé un algorithme nommé *DDG* (Data Duplication on Grid) en décomposant logiquement le réseau en régions (zones). Chaque région regroupe les sites qui appartiennent à la même partition géographique. Cette décomposition est sous forme de structure logique à deux dimensions. Où chaque ligne de cette structure représente une région du réseau et les éléments de la même ligne contiennent les sites géographiquement proches. L'algorithme *DDG* n'autorise qu'une réplique par région. Le nombre de répliques pour cette approche ne doit pas dépasser $N = R = \sqrt{S}$, où R représente le nombre de régions (zones) dans la grille et S représente

le nombre de sites dans la grille. Puisque cet algorithme de réplication est limité par le nombre de répliques, il ne peut répliquer les données que dans très peu d'endroits, ce qui représente une limite pour cette approche, car elle fournit une faible disponibilité surtout dans le cas des défaillances des sites possédant les répliques.

Dans [Basmadjian, 11], un protocole nommé *A2DS* (Arbitrary 2D Structured) a été proposé. Il peut être adapté à n'importe quelle structure bidimensionnelle. Les répliques sont organisées logiquement dans n'importe quelle structure 2D (droite, triangle, carré, trapèze et rectangle). Les auteurs ont aussi introduit une méthode pour obtenir d'autres structures 2D (hexagone et octogone) par la composition de plusieurs structures 2D basiques. Dans ce protocole, un quorum d'écriture est formé de toutes les répliques de n'importe quel niveau de la structure 2D, et un quorum de lecture est formé de n'importe quelle réplique de chaque niveau de la structure 2D.
Un autre protocole appelé *ReadTwoWriteMajority* a également été proposé, où les répliques sont arrangées logiquement dans une structure rectangulaire pour un nombre pair de répliques, et sont arrangées logiquement dans une structure trapézoïdale pour un nombre impair de répliques. Ce protocole a été proposé afin de contourner les inconvénients du protocole ROWA (liés aux opérations d'écriture) et de préserver ses avantages (liés aux opérations de lecture). Les auteurs ont réalisé une analyse comparative sur l'adaptation de ces protocoles aux différentes structures 2D.

Les auteurs dans [Latip, 13] ont présenté un protocole de réplication à trois quorums nommé *TQR* (Triple Quorum Replication). Comme son nom l'indique, ce protocole est structuré sur l'intersection de trois groupes de quorums. Il vise à réduire les tailles des quorums et à maintenir une haute disponibilité des données même lorsque la taille du réseau augmente. Dans ce travail, les auteurs se sont focalisés sur la sélection des copies à répliquer (quelle copie est sélectionnée pour être répliquée) en proposant une nouvelle stratégie de contrôle de répliques. Cette approche est limitée par plusieurs hypothèses sur le nombre de nœuds dans un quorum, la similarité des quorums de lecture et des quorums d'écriture, la même disponibilité pour les deux opérations de lecture et d'écriture. Ces auteurs ont étudié et expérimenté que les performances de la disponibilité et ils ont négligé les performances des temps de réponses et des coûts des cohérences.

Les auteurs dans [Choi, 12] proposent un nouveau protocole de réplication appelé Protocole Dynamique et Hybride qui combine entre le protocole du Quorum à Grille et le protocole de Quorum à Arbre. La structure logique utilisée pour ce protocole est arborescente composé des deux structures logiques du Quorum à Grille et du Quorum à Arbre. C'est une topologie basée sur trois paramètres de configuration : hauteur de l'arbre, profondeur de la grille et nombre de descendants. Les auteurs précisent par leur étude analytique que pour obtenir une grande disponibilité en lecture, la hauteur de l'arbre et le nombre de descendants doivent être réduits et la profondeur de la grille doit augmenter. Pour obtenir une grande disponibilité en écriture, la hauteur de l'arbre et la profondeur de la grille doivent être réduites, et le nombre de descendants doit augmenter. Dans ce travail, les auteurs n'ont étudié leur

protocole que sur un nombre réduit de nœuds, et aucune précision n'a été faite sur le passage à grande échelle.

Dans [Senhadji, 13], les auteurs proposent une stratégie dynamique d'équilibrage de charge et de gestion de la cohérence des répliques basée sur le système de quorums pour améliorer les performances. Les nœuds contenant des répliques sont représentés dans une structure logique d'arbre binaire. Afin de maintenir la cohérence des répliques d'une donnée, lorsqu'une demande d'accès en lecture ou en écriture est adressée à une réplique, un quorum est construit à partir de l'arborescence par le chemin depuis la feuille de l'arbre jusqu'à sa racine. Pour améliorer les performances, les auteurs complémentent la cohérence des répliques, par une stratégie d'équilibrage de charge suivant l'état de chaque nœud du quorum. Cette stratégie est basée sur la permutation entre les nœuds des ensembles des quorums afin de réduire la charge et le temps de communication des requêtes de lecture/écriture.

9.2. Les systèmes d'agents

Dans cette section, nous présentons quelques travaux qui ont proposé des approches basées sur les agents pour gérer les répliques dans les grilles.

Les auteurs dans [Vashisht, 14] proposent un algorithme nommé *EDRA* (Efficient Dynamic Replication Algorithm), basé sur l'utilisation des agents pour optimiser les performances dans les grilles. La grille est décomposée logiquement en régions et en sous-régions formant une structure hiérarchique. Les agents sont placés dans les régions et les sous-régions. Ils permettent de recueillir et de conserver des informations nécessaires pour l'ordonnancement des tâches et pour la sélection de la meilleure réplique. Les informations recueillies (telles que la disponibilité des nœuds, la fréquence d'accès, la capacité de calcul des nœuds, etc.) sont locales à chaque région ou sous-région.

Dans [Bell, 03a], les auteurs ont proposé une stratégie de réplication basée sur un modèle économique, qui optimise à la fois la sélection des répliques pour l'exécution des tâches et la création dynamique des répliques dans la grille. Dans cette approche, des agents d'optimisation, situés dans les sites de la grille, ont été utilisés. Ces agents utilisent un protocole basé sur les enchères pour sélectionner la réplique optimale d'un fichier, et une fonction de prédiction pour prendre les meilleures décisions concernant la réplication des fichiers locaux.
Les fichiers sont considérés comme des produits sur le marché et sont négociés, par les différents agents d'optimisation de chaque site de la grille, en fonction des demandes provenant des travaux en cours. Ces fichiers sont vendus par les *SE* (Storage Element) soit aux *CE* (Calcul Element) soit à d'autres *SE*. Les *CE* tentent de minimiser les coûts d'achat des fichiers, tandis que les *SE* tentent de maximiser leurs profits. Les *CE* et les *SE* interagissent avec des agents d'optimisation intelligents qui effectuent le raisonnement requis.

Les auteurs dans [Zhang, 10] ont proposé une approche de réplication décentralisée basée sur l'intelligence en essaim (Swarm intelligence [Beni, 93]) en utilisant des agents dans la grille de données. Chaque site de la grille possède un seul agent. Les agents participent dans la sélection de la copie du fichier qui doit être répliquée, dans la localisation du site où la copie du fichier sera répliquée ou migrée, et aussi dans l'estimation de la charge qui peut être transférée vers un autre site.

Dans [Senthilnathan, 12], les auteurs proposent un mécanisme de réplication basé sur plusieurs agents pour gérer la cohérence entre les répliques des fichiers de manière efficace, en propageant judicieusement les dernières mises à jour. Ce mécanisme de réplication est basé sur trois types d'agents : un agent responsable du maintien de la stabilité du système en offrant la tolérance aux pannes, un agent responsable du maintien de la stabilité du nœud et un agent de mise à jour qui est responsable de la cohérence entre les nœuds.

Lee et Yang [Lee, 05] ont proposé un service de cohérence basé sur les agents. Les agents interviennent dans deux modules : le module de service de cohérence (GCSM: Grid Consistency Service Module) et le module de transaction (GTM : Grid Transaction Module). Ces deux modules GCSM et GTM coordonnent entre eux pour maintenir la cohérence et l'intégrité des répliques.

Dans [Naseera, 09], les auteurs proposent un algorithme basé sur les agents pour le placement des répliques. Les agents sont utilisés pour déterminer le site candidat pour le placement des répliques. Un agent est déployé sur chaque site contenant les copies principales des données partagées. Pour créer une réplique, chaque agent hiérarchise les ressources de stockage de la grille en leur attribuant des priorités en fonction des configurations des ressources, de la bande passante et des demandes d'accès aux répliques sur leurs sites, puis crée une réplique dans les emplacements appropriés.

10. Conclusion

Pour déterminer la stratégie de réplication, les informations sur l'environnement des applications et le modèle d'accès aux données, sont des facteurs importants à être considérés. Une politique de réplication efficace dans un environnement peut ne pas être efficace ou peut causer des effets négatifs dans un autre environnement.
Il existe beaucoup de facteurs qui influencent le choix de la stratégie de réplication. Il n'existe aucune stratégie de réplication universelle qui améliorera tous les aspects du système dans n'importe quel environnement.

Dans ce chapitre nous avons présenté une synthèse sur les notions de la réplication dans les systèmes à large échelle. Nous avons aussi présenté les protocoles de réplication basés sur les quorums. Nous avons mis en évidence les problèmes liées à la réplication des données. Vu que les objectifs de la réplication sont conflictuels (disponibilité des données vs fiabilité des données), deux grands problèmes suscitent l'intérêt des chercheurs : le premier est lié au placement efficace des répliques et le second est lié à la gestion de la cohérence mutuelle des répliques. Nous avons

également cités quelques travaux qui se sont intéressés aux protocoles de réplication basés sur les quorums et aux systèmes d'agents dans les grilles.

Dans le prochain chapitre, nous allons présenter et détailler notre approche de réplication dans les grilles de données, basée sur les agents et les quorums.

Chapitre IV.
La réplication à base d'un système multi-agents

1. Introduction

Le nombre de répliques, la cohérence de ces répliques, ainsi que les nœuds qui contiennent les répliques sont des notions très importantes lors de l'élaboration d'une stratégie de réplication. En plus, si les stratégies de réplication sont dynamiques et se basent sur les quorums, le calcul des tailles des quorums doit être effectué à chaque fois que le nombre de répliques change. Cette opération est coûteuse surtout à large échelle (grille de données). Ce qui influence les performances du système tel que le temps de réponse.
Dans ce contexte, et pour minimiser ce coût et améliorer les performances de la réplication, nous contribuons par une nouvelle approche de réplication basée sur une nouvelle notion des quorums, que nous avons appelé Quorum Imbriqué (IQ). Cette approche est supervisée par un système multi-agents qui prend en considération à la fois les problèmes liés à la cohérence et les problèmes liés à la réplication.

2. Principe

Dans une grille de données, une stratégie de réplication dynamique déplace, ajoute ou supprime les répliques dynamiquement. Le placement des répliques lors de leur ajout et leur suppression influencent forcément les tailles des quorums. Par conséquent ces tailles doivent être, elles aussi, gérées de manière dynamique.
Cependant, la définition dynamique des ensembles de quorums (formation des quorums) est coûteuse. Car à chaque fois que le nombre des copies change nous devons recalculer les tailles des quorums de lecture et d'écriture, avant qu'aucune nouvelle opération de lecture ou d'écriture ne soit effectuée. Donc le problème revient à minimiser le nombre de ces calculs. Ce qui nous amène à réduire les tailles des ensembles de quorums. Pour cela nous réduisons le nombre des copies impliquées dans un quorum en les distribuant en groupes de copies (distribution en clusters de nœuds). Par conséquent le nombre de copies impliquées dans les opérations de lecture ou d'écriture sera réduit. Ce qui nous permet de réduire les temps des mises à jour et d'améliorer les temps de réponses lors des accès aux données.
Cette distribution des copies dans des groupes nous permet de gérer la cohérence mutuelle des copies et aussi d'en tirer profit pour prendre des décisions sur la création, le placement et le nombre des répliques.
En effet, étant donné que chaque groupe de copies peut avoir un leader, et puisque les agents sont des entités actives (et non passives), nous pouvons attribuer à chaque leader un agent et profiter de leurs utilisation. Cet ensemble d'agents pourra coopérer pour gérer la réplication et la cohérence des répliques afin d'améliorer les performances [Bouharaoua, 17].

Dans ce qui suit, nous commençons par présenter le principe de formation des groupes sur la base des quorums (imbriqués) ensuite nous expliquerons l'utilisation des systèmes multi-agents dans notre approche.

3. Imbrication des quorums

Soient :
n : le nombre de copies d'une donnée distribuées dans une grille de données.
Qr : le quorum de lecture.
Qw : le quorum d'écriture.

En appliquant le protocole du Quorum à Consensus, nous obtenons :
La taille du quorum de lecture $Qr=\left\lceil\frac{n}{2}\right\rceil$.
La taille du quorum d'écriture $Qw=\left\lceil\frac{n+1}{2}\right\rceil$.

Si le nombre des copies change, ces tailles de quorums doivent être mise à jour par toutes les entités responsables de la réplication (service de consistance des répliques qui sont géographiquement éloignées au sein de la grille).
Cette mise à jour est coûteuse ce qui diminue les performances (temps de réponses…). Pour minimiser ce coût afin d'améliorer les performances, nous proposons une technique qui consiste à minimiser le nombre de fois où les tailles des quorums doivent être recalculés.
Pour cela, nous répartissons l'ensemble des répliques en groupes, par conséquent les tailles des quorums appliqués sur ces groupes seront considérablement diminuées. Chaque groupe aura ses propres répliques, d'où la possibilité d'appliquer le principe du protocole du Quorum à Consensus sur les groupes, ensuite nous appliquons une deuxième fois le Quorum à Consensus sur les répliques à l'intérieur de chaque groupe choisi séparément (i.e. les groupes appartenant aux quorums des groupes).

Exemple 1 :

Dans la figure IV.1, *22* répliques sont répartis en 3 groupes.
Le groupe 1 contient *8* répliques et les deux groupes 2 et 3 contiennent *7* répliques chacun.
En appliquant le Quorum à Consensus, nous obtiendrons : la taille du quorum de lecture $Qr=11$ et la taille du quorum d'écriture $Qw=12$.
Par contre en utilisant notre technique nous obtiendrons pour le premier quorum appliqué sur les groupes : un quorum de lecture $Qr=2$ et un quorum d'écriture $Qw=2$ (donc nous devrons choisir *2* groupes parmi *3* pour former un quorum de lecture ou d'écriture).
Ensuite à l'intérieur de chaque groupe choisi nous obtiendrons un quorum de lecture $Qr=4$ et un quorum d'écriture $Qw=4$ ou au maximum $Qw=5$.
Donc le quorum de lecture global $Qr=8=4+4$ et le maximum du quorum d'écriture global $Qw=9=5+4$ (il se peut que Qw soit égal seulement à $8=4+4$).
Ainsi, les tailles des quorums sont réduites par rapport au Quorum à Consensus. La différence des tailles des quorums entre les deux approches égale au minimum *6* (*3* pour le quorum de lecture et au minimum *3* pour le quorum d'écriture).

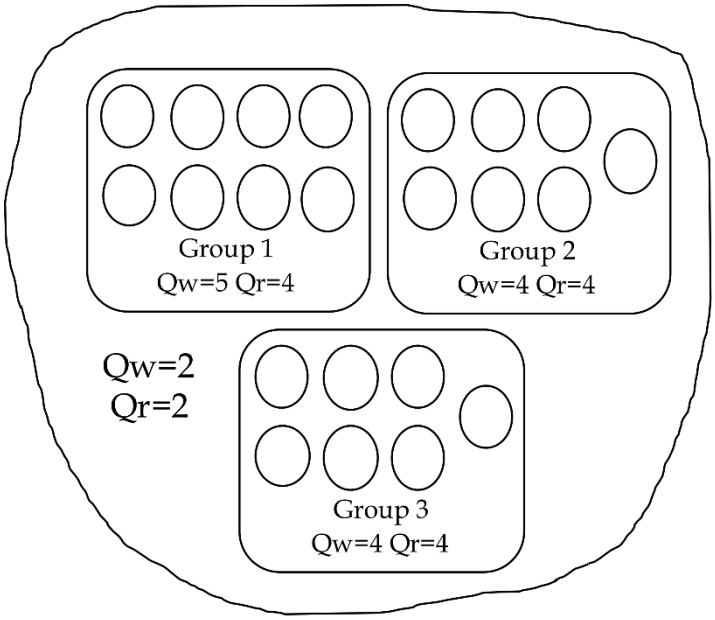

Figure IV.1 : Exemple de répartition des répliques en groupes.

D'un autre côté, nous réduisons le coût des mises à jour des tailles des quorums si le nombre de répliques change.
Par exemple si une réplique est ajouté au groupe 1, le nombre des répliques sera *9*, et nous mettrons à jour que les tailles des quorums de ce groupe seulement (*Qr=5* et *Qw=5*), et non pas pour tout le système de la grille.
En plus, cette mise à jour peut être faite de manière asynchrone si le groupe concerné n'est pas sollicité.
Il est à noter que même les mises à jour des répliques peuvent être faites de manière asynchrone pour les groupes qui sont en retard et qui ne sont pas sollicités.

4. Définition des tailles des quorums

Dans ce qui suit, nous définissons les tailles des quorums de notre approche (Quorum Imbriqué).
Nous utiliserons le mot cluster au lieu du mot groupe.

Soient :
n : le nombre de répliques.
Nclt : le nombre total des clusters dans la grille tel que $Nclt > 2$.
rep_clt_i : le nombre de répliques dans le cluster *i* tel que $rep\_clt_i > 2$ et $1 \leq i \leq Nclt$.
QCw: quorum d'écriture au sein de la grille (appliqué sur les clusters).
QCr: quorum de lecture au sein de la grille (appliqué sur les clusters).
Qw_i: quorum d'écriture au sein du cluster *i* (appliqué sur les répliques du cluster *i*).
Qr_i: quorum de lecture au sein du cluster *i* (appliqué sur les répliques du cluster *i*).

Les quorums obtenus en appliquant le Quorum Imbriqué sont :

Le quorum des clusters concerné par une opération d'écriture est :

$$QCw = \left\lceil \frac{Nclt + 1}{2} \right\rceil \quad (1)$$

Le quorum des clusters concerné par une opération de lecture est :

$$QCr = \left\lceil \frac{Nclt}{2} \right\rceil \quad (2)$$

Le quorum des répliques du cluster i concernées par une écriture est :

$$Qw_i = \left\lceil \frac{rep\_clt_i + 1}{2} \right\rceil \quad (3)$$

Le quorum des répliques du cluster i concernées par une opération de lecture est :

$$Qr_i = \left\lceil \frac{rep\_clt_i}{2} \right\rceil \quad (4)$$

Le quorum global d'écriture (quorum des répliques de la grille concernées par une opération d'écriture) est :

$$Qw = \sum_{i=1}^{\left\lceil \frac{Nclt+1}{2} \right\rceil} \left\lceil \frac{rep\_clt_i + 1}{2} \right\rceil \quad (5)$$

Le quorum global de lecture (quorum des répliques de la grille concernées par une opération de lecture) est :

$$Qr = \sum_{i=1}^{\left\lceil \frac{Nclt}{2} \right\rceil} \left\lceil \frac{rep\_clt_i}{2} \right\rceil \quad (6)$$

Démonstration

Nous allons démontrer que les tailles des quorums obtenues par notre méthode du Quorum Imbriqué sont meilleures que celles obtenues par la méthode du Quorum à Consensus.

Nous avons :
Le quorum d'écriture du Quorum à Consensus est :

$$Qw = \left\lceil \frac{n + 1}{2} \right\rceil \quad (7)$$

Le quorum de lecture du Quorum à Consensus est :

$$Qr = \left\lceil \frac{n}{2} \right\rceil \quad (8)$$

$Nclt > 2$
$rep\_clt_i > 2 \quad \forall \ 1 \leq i \leq Nclt$

a. Quorum global (lecture + écriture)

La somme du nombre des répliques de chaque cluster égal au nombre total des répliques de la grille.

$$\sum_{i=1}^{Nclt} rep\_clt_i = n \tag{9}$$

$$\Rightarrow \sum_{i=1}^{Nclt} \left(\left\lceil \frac{rep\_clt_i}{2} \right\rceil + \left\lfloor \frac{rep\_clt_i}{2} \right\rfloor \right) = n \tag{9'}$$

$$\Rightarrow \sum_{i=1}^{Nclt} \left\lceil \frac{rep\_clt_i}{2} \right\rceil + \sum_{i=1}^{Nclt} \left\lfloor \frac{rep\_clt_i}{2} \right\rfloor = n \tag{9"}$$

$$\Rightarrow \sum_{i=1}^{\left\lceil \frac{Nclt}{2} \right\rceil} \left\lceil \frac{rep\_clt_i}{2} \right\rceil + \sum_{i=\left\lfloor \frac{Nclt}{2} \right\rfloor + 1}^{Nclt} \left\lceil \frac{rep\_clt_i}{2} \right\rceil + \sum_{i=1}^{\left\lceil \frac{Nclt+1}{2} \right\rceil} \left\lfloor \frac{rep\_clt_i}{2} \right\rfloor + \sum_{i=\left\lfloor \frac{Nclt+1}{2} \right\rfloor + 1}^{Nclt} \left\lfloor \frac{rep\_clt_i}{2} \right\rfloor = n \tag{9'''}$$

Nous avons

$$\sum_{i=1}^{\left\lceil \frac{Nclt+1}{2} \right\rceil} 1 < \sum_{i=\left\lfloor \frac{Nclt}{2} \right\rfloor + 1}^{Nclt} \left\lceil \frac{rep\_clt_i}{2} \right\rceil + \sum_{i=\left\lfloor \frac{Nclt+1}{2} \right\rfloor + 1}^{Nclt} \left\lfloor \frac{rep\_clt_i}{2} \right\rfloor \tag{10}$$

Nous déduisons de (9"") et (10) que

$$\sum_{i=1}^{\left\lceil \frac{Nclt}{2} \right\rceil} \left\lceil \frac{rep\_clt_i}{2} \right\rceil + \sum_{i=1}^{\left\lceil \frac{Nclt+1}{2} \right\rceil} \left\lfloor \frac{rep\_clt_i}{2} \right\rfloor + \sum_{i=1}^{\left\lceil \frac{Nclt+1}{2} \right\rceil} 1 < n \tag{11}$$

$$\Rightarrow \sum_{i=1}^{\left\lceil \frac{Nclt}{2} \right\rceil} \left\lceil \frac{rep\_clt_i}{2} \right\rceil + \sum_{i=1}^{\left\lceil \frac{Nclt+1}{2} \right\rceil} \left(\left\lfloor \frac{rep\_clt_i}{2} \right\rfloor + 1 \right) < n \tag{11'}$$

$$\Rightarrow \sum_{i=1}^{\left\lceil \frac{Nclt}{2} \right\rceil} \left\lceil \frac{rep\_clt_i}{2} \right\rceil + \sum_{i=1}^{\left\lceil \frac{Nclt+1}{2} \right\rceil} \left\lceil \frac{rep\_clt_i + 1}{2} \right\rceil < n \tag{11"}$$

Nous avons

$$n < n + 1 = \left\lceil \frac{n+1}{2} \right\rceil + \left\lceil \frac{n}{2} \right\rceil \tag{12}$$

Nous déduisons de (11") et (12) que

$$\Rightarrow \sum_{i=1}^{\left\lceil \frac{Nclt}{2} \right\rceil} \left\lceil \frac{rep\_clt_i}{2} \right\rceil + \sum_{i=1}^{\left\lceil \frac{Nclt+1}{2} \right\rceil} \left\lceil \frac{rep\_clt_i + 1}{2} \right\rceil < \left\lceil \frac{n+1}{2} \right\rceil + \left\lceil \frac{n}{2} \right\rceil \tag{13}$$

Donc, la taille globale des quorums (lecture + écriture) du Quorum Imbriqué est inférieure à celle du Quorum à Consensus.

b. Quorum de lecture

Nous déduisons de (9) que

$$\Rightarrow \sum_{i=1}^{\left\lceil \frac{Nclt}{2} \right\rceil} rep\_clt_i + \sum_{i=\left\lceil \frac{Nclt}{2} \right\rceil+1}^{Nclt} rep\_clt_i = n \quad (14)$$

Nous avons

$$\sum_{i=1}^{\left\lceil \frac{Nclt}{2} \right\rceil} 1 < \sum_{i=\left\lceil \frac{Nclt}{2} \right\rceil+1}^{Nclt} rep\_clt_i \quad (15)$$

Nous déduisons de (14) et (15) que

$$\sum_{i=1}^{\left\lceil \frac{Nclt}{2} \right\rceil} rep\_clt_i + \sum_{i=1}^{\left\lceil \frac{Nclt}{2} \right\rceil} 1 < n \quad (16)$$

$$\Rightarrow \sum_{i=1}^{\left\lceil \frac{Nclt}{2} \right\rceil} (rep\_clt_i + 1) < n \quad (16')$$

$$\Rightarrow \sum_{i=1}^{\left\lceil \frac{Nclt}{2} \right\rceil} \frac{rep\_clt_i + 1}{2} < \frac{n}{2} \quad (16'')$$

Nous avons

$$\frac{n}{2} \leq \left\lceil \frac{n}{2} \right\rceil \quad (17)$$

$$\left\lceil \frac{rep\_clt_i}{2} \right\rceil \leq \frac{rep\_clt_i + 1}{2} \quad (18)$$

Nous déduisons de (16''), (17) et (18) que

$$\sum_{i=1}^{\left\lceil \frac{Nclt}{2} \right\rceil} \left\lceil \frac{rep\_clt_i}{2} \right\rceil < \left\lceil \frac{n}{2} \right\rceil \quad (19)$$

La taille de quorum de lecture du Quorum Imbriqué est alors inférieure à celle du Quorum à Consensus.

c. Quorum d'écriture

Pour le quorum d'écriture, il peut y avoir des cas où la taille du quorum obtenue par le Quorum Imbriqué n'est pas meilleure à celle obtenue par le Quorum à Consensus. Et cela lorsque le nombre des répliques qui n'interviennent pas dans le quorum d'écriture (restantes) est inférieur ou égal au nombre des clusters.

Pour que la taille du quorum obtenue par notre approche ne soit pas meilleure à celle obtenue par le Quorum à Consensus, il faut que :

$$\sum_{i=1}^{\left\lfloor\frac{Nclt+1}{2}\right\rfloor} \left\lceil\frac{rep\_clt_i + 1}{2}\right\rceil \geq \left\lceil\frac{n+1}{2}\right\rceil \quad (20)$$

$$\Rightarrow \sum_{i=1}^{\left\lfloor\frac{Nclt+1}{2}\right\rfloor} \left(\left\lfloor\frac{rep\_clt_i}{2}\right\rfloor + 1\right) \geq \left\lfloor\frac{n}{2}\right\rfloor + 1 \quad (20')$$

$$\Rightarrow \sum_{i=1}^{\left\lfloor\frac{Nclt+1}{2}\right\rfloor} \left\lfloor\frac{rep\_clt_i}{2}\right\rfloor + \sum_{i=1}^{\left\lfloor\frac{Nclt+1}{2}\right\rfloor} 1 \geq \left\lfloor\frac{n}{2}\right\rfloor + 1 \quad (20'')$$

$$\Rightarrow \sum_{i=1}^{\left\lfloor\frac{Nclt+1}{2}\right\rfloor} \left\lfloor\frac{rep\_clt_i}{2}\right\rfloor + \left\lceil\frac{Nclt+1}{2}\right\rceil \geq \left\lfloor\frac{n}{2}\right\rfloor + 1 \quad (20''')$$

$$\Rightarrow \sum_{i=1}^{\left\lfloor\frac{Nclt+1}{2}\right\rfloor} \left\lfloor\frac{rep\_clt_i}{2}\right\rfloor + \left\lceil\frac{Nclt+1}{2}\right\rceil - 1 \geq \left\lfloor\frac{n}{2}\right\rfloor \quad (20'''')$$

Nous avons

$$\left\lceil\frac{Nclt+1}{2}\right\rceil - 1 = \left\lfloor\frac{Nclt}{2}\right\rfloor \quad (21)$$

Nous déduisons de (20'''') et (21) que

$$\sum_{i=1}^{\left\lfloor\frac{Nclt+1}{2}\right\rfloor} \left\lfloor\frac{rep\_clt_i}{2}\right\rfloor + \left\lfloor\frac{Nclt}{2}\right\rfloor \geq \left\lfloor\frac{n}{2}\right\rfloor \quad (22)$$

$$\Rightarrow \sum_{i=1}^{\left\lfloor\frac{Nclt+1}{2}\right\rfloor} \left\lfloor\frac{rep\_clt_i}{2}\right\rfloor \geq \left\lfloor\frac{n}{2}\right\rfloor - \left\lfloor\frac{Nclt}{2}\right\rfloor \quad (22')$$

$$\Rightarrow 2 \sum_{i=1}^{\left\lfloor\frac{Nclt+1}{2}\right\rfloor} \left\lfloor\frac{rep\_clt_i}{2}\right\rfloor \geq 2\left(\left\lfloor\frac{n}{2}\right\rfloor - \left\lfloor\frac{Nclt}{2}\right\rfloor\right) \quad (22'')$$

Nous avons

$$\left|(n - Nclt) - 2\left(\left\lfloor\frac{n}{2}\right\rfloor - \left\lfloor\frac{Nclt}{2}\right\rfloor\right)\right| \leq 1 \quad (23)$$

$$\sum_{i=1}^{\left\lceil\frac{Nclt+1}{2}\right\rceil} rep\_clt_i \geq 2 \sum_{i=1}^{\left\lceil\frac{Nclt+1}{2}\right\rceil} \left\lfloor\frac{rep\_clt_i}{2}\right\rfloor \quad (24)$$

Nous déduisons de (22"), (23) et (24) que

$$\Rightarrow \sum_{i=1}^{\left\lceil\frac{Nclt+1}{2}\right\rceil} rep\_clt_i \geq n - Nclt - 1 \quad (25)$$

Nous déduisons de (9) que

$$\sum_{i=1}^{\left\lceil\frac{Nclt+1}{2}\right\rceil} rep\_clt_i + \sum_{i=\left\lceil\frac{Nclt+1}{2}\right\rceil+1}^{Nclt} rep\_clt_i = n \quad (26)$$

Le reste des répliques qui n'interviennent pas dans le quorum d'écriture peut être formulé par la somme suivante :

$$\sum_{i=\left\lceil\frac{Nclt+1}{2}\right\rceil+1}^{Nclt} rep\_clt_i$$

Nous déduisons de (25) et (26) que

$$\sum_{i=\left\lceil\frac{Nclt+1}{2}\right\rceil+1}^{Nclt} rep\_clt_i \leq Nclt + 1 \quad (27)$$

Donc, pour que la taille du quorum d'écriture du Quorum Imbriqué soit supérieur ou égal à celui du Quorum à Consensus, il faut que le nombre des répliques restantes (qui n'interviennent pas dans le quorum d'écriture) soit inférieur ou égal au nombre de clusters plus 1.

Résultat et discussion

A partir de ce résultat obtenu de cette dernière inéquation (formule 27), nous pourrons dire que notre proposition est appropriée pour les systèmes à large échelle (grille de données) où la taille des clusters est grande. Même pour les systèmes à petit échelle, où la taille des clusters est petite, notre proposition restera fonctionnelle sauf pour des cas rares (voir annexe A).

Exemple 2 :

Voici un exemple illustratif qui compare entre les deux approches (Quorum à Consensus et Quorum Imbriqué).
Soit n =70 le nombre de répliques d'une donnée réparties dans 7 clusters selon la table IV.1.

En appliquant la formule (1), nous obtenons le quorum d'écriture sur les clusters : QCw=4=$\left\lceil\frac{Nclt+1}{2}\right\rceil$.

En appliquant la formule (2), nous obtenons le quorum de lecture sur les clusters : $QCr=4=\left\lceil\frac{Nclt}{2}\right\rceil$.
En appliquant les formules (3) et (4), nous obtenons les quorums de lecture et d'écriture au sein de chaque cluster (tableIV.1).

Cluster	**Nombre de répliques**	**Quorum d'écriture du cluster *i*** $\left\lceil\frac{rep\_clt_i+1}{2}\right\rceil$	**Quorum de lecture du cluster *i*** $\left\lceil\frac{rep\_clt_i}{2}\right\rceil$
1	13	7	7
2	10	6	5
3	8	5	4
4	11	6	6
5	12	7	6
6	7	4	4
7	9	5	5

Table IV.1 : Taille des quorums au sein des clusters.

Le calcul des différents quorums d'écriture et de lecture en appliquant les formules (5) et (6) pour le Quorum Imbriqué et (7) et (8) pour le Quorum à Consensus, est résumé par la table IV.2.

	Quorum d'écriture	**Quorum de lecture**
Quorum à Consensus	36	35
Quorum Imbriqué (total)	26	24

Table IV.2 : Quorum Imbriqué vs Quorum à Consensus.

Nous remarquons que les quorums obtenus par notre approche sont meilleurs que ceux obtenus par le Quorum à Consensus.

5. Formation des groupes (clusters)

Pour minimiser les tailles des quorums nous devons répartir les répliques en groupes. Et pour cela nous répartissons l'ensemble des nœuds de la grille en groupes (clusters).
Dans les systèmes distribués, il existe plusieurs méthodes pour regrouper un ensemble de nœuds et pour déterminer leurs leaders [Alumuttairi, 10] [Mitton, 06] [Sharma, 10] [Souli-Jbali, 15] [Yu, 05]. Elles peuvent être basées sur différentes métriques tels que les latences de réseau, la bande passante, l'espace de stockage libre, le voisinage, la disponibilité, etc.

Puisque le voisinage et la disponibilité des nœuds sont deux paramètres très importants dans les grilles et pour améliorer les performances, nous proposons dans notre approche deux méthodes de regroupement : la première est basée sur le voisinage et la deuxième est basée sur la disponibilité des nœuds [Bouharaoua, 17].

Il est à noter que les formations des groupes par ces deux méthodes sont effectuées indépendamment des répliques et de leurs emplacements (formation des groupes de tous les nœuds de la grille et non des répliques).

5.1. Regroupement à base de voisinage

La formation des clusters par cette méthode se fait sur la base du nombre de voisins d'un nœud, où le nœud qui possède le plus de voisins est susceptible d'être le leader de son cluster.
La formation des clusters s'effectue comme suit :

- Chaque nœud possède : un identifiant qui le désigne et une valeur qui désigne le nombre de ses voisins.
- Parmi l'ensemble des nœuds, le nœud possédant le plus grand nombre de voisins est sélectionné.
- En cas d'égalité entre plusieurs nœuds, le nœud ayant le plus petit identifiant sera choisi.
- Un cluster est formé à partir de ce nœud et de tous ses voisins.
- Le nœud choisi est élu leader de ce cluster.
- Recalculer le nombre de voisins pour le reste des nœuds (qui ne sont pas encore regroupés).
- Refaire les étapes précédentes.
- Dans le cas où des nœuds orphelins existent (les nœuds restant n'appartenant à aucun cluster), nous affectons chaque nœud orphelin au même cluster que l'un de ses voisins (ayant le plus petit identifiant).

Exemple 3 :

Dans l'exemple de la figure IV.2, chacun des nœuds 4 et 9 possède 4 voisins (voir la table IV.3).
Le nœud 4 est choisi (puisque 4 < 9).

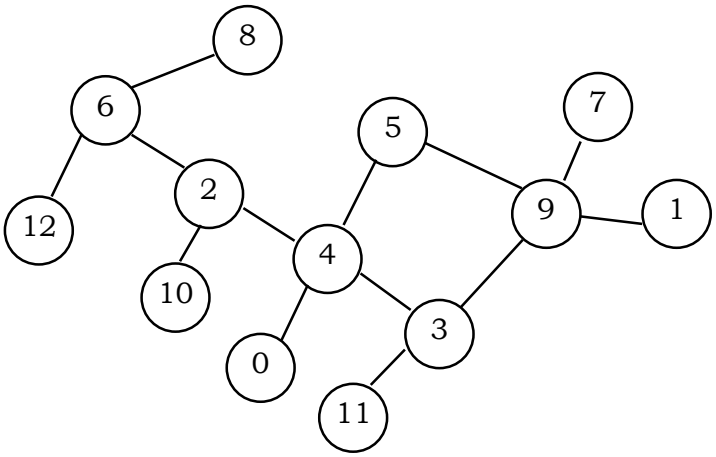

Figure IV.2 : Exemple d'un ensemble de nœuds et leurs voisins.

Nœud	Nombre de voisins
0	1
1	1
2	3
3	3
4	4
5	2
6	3
7	1
8	1
9	4
10	1
11	1
12	1

Table IV.3 : Nombre de voisins par nœud.

A partir du nœud 4, le premier cluster, contenant ce nœud et tous ses voisins, sera formé (il contiendra les nœuds 0, 2, 3, 4, et 5).
Le même travail sera effectué avec le reste des nœuds (qui n'appartiennent pas au cluster formé).
Nous recalculons le nombre de voisins. Le nœud 6 et le nœud 9 ont le même nombre de voisins, chacun possède 2 voisins, le nœud 6 sera choisi (puisque 6<9).
Le deuxième cluster contiendra les nœuds 6, 8 et 12. Et le troisième cluster contiendra les nœuds 1, 7 et 9.
Après avoir recalculé le nombre de voisins, les nœuds 10 et 11 deviendront des nœuds orphelins (aucun voisin). Le nœud 10 sera affecté au même cluster que celui du nœud 2, et le nœud 11 sera affecté au même cluster que celui du nœud 3. Par conséquent, le premier cluster contiendra tous les nœuds 0, 2, 3, 4, 5, 10 et 11. La figure IV.3 schématise la formation de ces trois clusters.

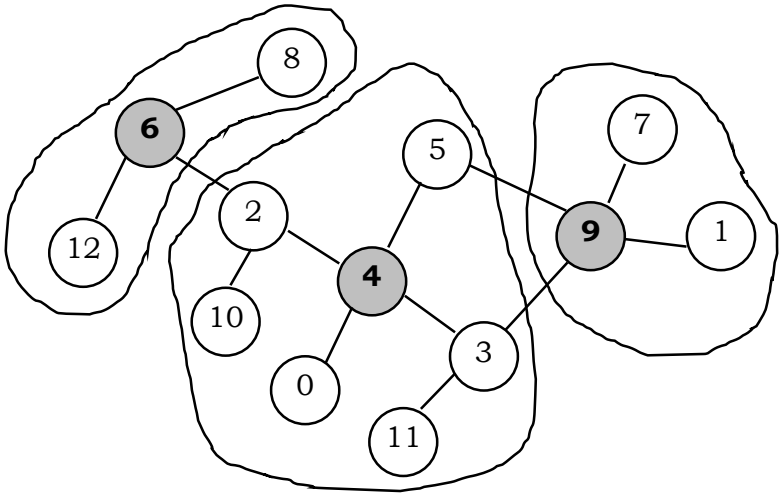

Figure IV.3 : Exemple de formation des clusters à base de voisinage.

Cette méthode favorise les nœuds qui ont le plus de voisins, cela signifie que ces nœuds (s'ils possèdent une réplique) peuvent servir beaucoup de nœuds (ceux qui sont proches).

L'algorithme 1 décrit le fonctionnement du regroupement à base de voisinage.

Algorithm 1 : regroupement à base de voisinage

```
1 :  E,C: set of nodes
2 :  x, n : nodes
3 :  BEGIN
4 :  E← All Nodes              /* initialization : E contains all nodes */
5 :  while E ≠ ∅ do
6 :     Max←0
7 :     for any n ∈ E do       /*Looking for the node having the maximum neighbors*/
8 :        if number_of_neighbors(n) > max then
9 :           max←number_of_neighbors(n)
10 :          x←n
11 :       end if
12 :    end for
13 :    if max > 0 then
14 :      C←{x}+{neighbors(x)}
15 :    else                          /* orphans node */
16 :       C←{n}
17 :    end if
18 :      Add C to set of clusters    /* formation of clusters*/
19 :      E←E-C
20 : end while
21 : END
```

La Complexité d'exécution de cet algorithme est $O(n^2/4)$, où n désigne le nombre de nœud dans la grille. Le pire des scénarios pour cet algorithme est que chaque nœud possède *1* voisins, d'où l'obtention comme résultats de $n/2$ clusters (notons que ce cas est irréaliste). Donc, la complexité d'exécution au pire des cas est $O(n/2.(n/2+1))$ avec $n/2.(n/2+1)=2+4+...+n$. En contrepartie, dans le meilleur des cas la complexité d'exécution est $O(1)$, et cela quand tous les nœuds sont voisins à un nœud donné (qui représentera le leader d'un seul cluster).

5.2. Regroupement à base de la disponibilité

La formation des clusters par cette méthode se fait sur la base de la disponibilité des nœuds, où le nœud le plus disponible est susceptible d'être le leader de son groupe. La formation des clusters s'effectue comme suit :

- Chaque nœud possède : un identifiant qui le désigne et une valeur qui désigne sa disponibilité.
- Le nœud qui possède la plus grande valeur de disponibilité est sélectionné.
- En cas d'égalité entre plusieurs nœuds, le nœud ayant le plus petit identifiant sera choisi.

Cette tâche s'effectuera en trois étapes :

Etape 1 :

Pour chaque nœud, nous procédons comme suit : nous désignons le nœud le plus disponible (appelé chief) parmi ce nœud et ses voisins.
Après avoir terminé cette étape nous obtiendrons une liste contenant tous les nœuds et leurs chiefs.

Etape 2 :

Ensuite, à partir de ces nœuds et leur chief (les plus disponibles), nous déterminons les leaders des clusters.
Pour chaque nœud, nous pointons sur son chief, et pour ce dernier (chief), nous pointons encore une fois sur son chief et ainsi de suite, jusqu'à qu'un chief pointe sur lui-même. Ainsi, ce dernier chief obtenu (qui pointe sur lui-même) sera désigné comme leader de ce nœud. Lorsque nous terminons cette étape pour tous les nœuds nous obtiendrons une liste contenant tous les nœuds et leurs leaders.

Etape 3 :

Ensuite, nous formons les clusters à partir de ces nœuds et leurs leaders. Chaque ensemble de nœuds ayant le même leader forment un cluster. Le nombre de clusters formés sera égal au nombre de leaders obtenus.

Exemple 4 :

Soit l'ensemble des nœuds de la figure IV.4 et leurs valeurs de disponibilité selon la table IV.4.a.

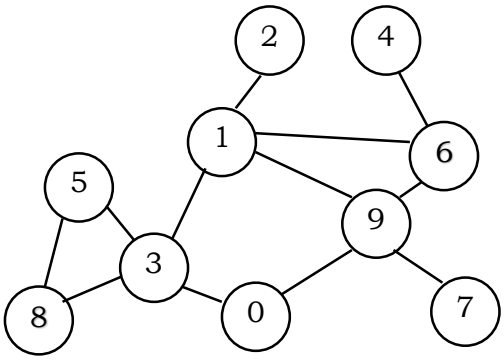

Figure IV.4 : Exemple d'un ensemble de nœuds.

Etape 1 : Détermination des chiefs.
Pour le nœud 0 le nœud le plus disponible parmi l'ensemble des nœuds {0, 3, 9} est le nœud 3, puisque sa disponibilité égale 70% parmi leur valeurs de disponibilité respectives {35%, 70%, 67%}.
Nous refaisons le même travail pour tous les nœuds. Lorsque nous terminons cette étape pour tous les nœuds, nous obtiendrons les résultats (les chiefs) de la table IV.4.b.

Nœud	Disponibilité
0	35
1	70
2	60
3	70
4	70
5	90
6	95
7	50
8	95
9	67

(a)

Nœud	Chief (nœud le plus disponible)
0	3
1	6
2	1
3	8
4	6
5	8
6	6
7	9
8	8
9	6

(b)

Table IV.4 : Ensemble des nœuds, leur disponibilité et leur chief.

Etape 2 : Détermination des leaders.
Ensuite, à partir de ces nœuds et leur chief, nous déterminons les leaders. Le chief du nœud 0 est 3, le chief du nœud 3 est le nœud 8 et ce dernier (le nœud 8) est le chief de lui-même. Ainsi, le leader du nœud 0 sera le nœud 8.
Nous refaisons le même travail pour tous les nœuds. Lorsque nous terminons cette étape pour tous les nœuds nous obtiendrons deux leaders : le nœud 6 et le nœud 8 ; (voir la table IV.5).

Nœud	Disponibilité	Chief	Leader
0	35	3	8
1	70	6	6
2	60	1	6
3	70	8	8
4	70	6	6
5	90	8	8
6	95	6	6
7	50	9	6
8	95	8	8
9	67	6	6

Table IV.5 : Ensemble des nœuds et leur leader.

Etape 3 : Formation des clusters
Ensuite les nœuds ayant le même leader forment le même cluster. Nous obtiendrons deux clusters représentés par la figure IV.5.

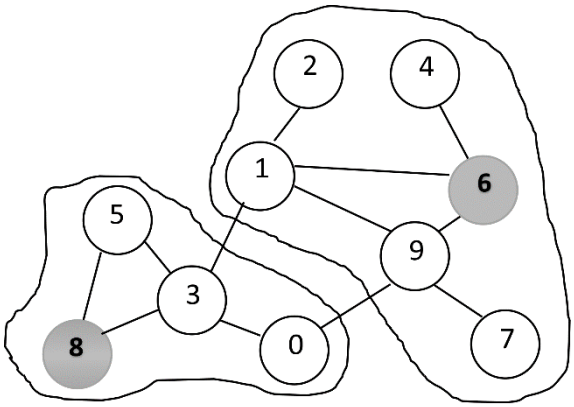

Figure IV.5 : Exemple de formation des clusters à base de la disponibilité.

Cette méthode favorise les nœuds les plus disponibles, ce qui améliore la disponibilité des données au sein de la grille si ces nœuds possèdent une réplique.

L'algorithme 2 décrit la méthode de regroupement à base de la disponibilité.

Algorithm 2 : regroupement à base de la disponibilité

```
1 :  E,C: set of nodes
2 :  BEGIN
3 :  E← All Nodes                    /* initialization : E contains all nodes */
4 :  x, n, y : nodes
5 :  for any n ∈ E do                /*Looking for the chief of n*/
6 :    max← availability (n)
7 :    y←n
8 :    for any x ∈ neighbours(n)
9 :       if availability (x) > max then
10 :        max← availability (x)
11 :        y←x
12 :      end if
13 :   end for
14 :   Chief(n) ←y
15 : end for
16 : for any n ∈ E do                /*Looking for the leader of cluster */
17 :   x←n
18 :   while x ≠ chief(x) do
19 :      x ←chief(x)
20 :      if  leader(x) ≠ null then
21 :        x ← leader (x)
22 :      end if
23 :   end while
24 :   leader (n)←x
```

```
25 :  end for
26 :  while E ≠ ∅ do                    /* formation of clusters*/
27 :    C ← ∅
28 :    n←leader(first node in E)
29 :    for any x ∈ E do
30 :        if leader(x)= n then
31 :          C←C+{x}
32 :        end if
33 :    end for
34 :    Add C to set of clusters
36 :    E←E-C
37 :  end while
38 :  END
```

La Complexité d'exécution de cet algorithme est $O(n^2)$, où n désigne le nombre de nœud dans la grille. Le pire des scénarios pour cet algorithme est que chaque nœud possède *n-1* voisins, d'où l'obtention comme résultats d'un seul cluster et d'un seul leader (notons que ce cas est irréaliste). Lors de la détermination des chiefs (étapes : 5 à 15), la complexité d'exécution au pire des cas est $O(n.(n-1))$. En contrepartie, la complexité d'exécution est $O(1)$ pour l'étape de détermination des leaders (étapes :16 à 25) et pour l'étape de formation des clusters (étapes : 26 à 37), puisqu'il existe qu'un seul cluster.

6. Formation des quorums et traitement des requêtes

A chaque fois que le nombre de répliques est modifié (ajout ou suppression de répliques) les tailles des quorums du protocole du Quorum à Consensus doivent être mis à jour pour l'ensemble du système. Par contre dans notre approche, les mises à jour des quorums ne seront faites qu'à l'intérieur du cluster où le nombre de répliques a été modifié.

La formation des quorums et le traitement des requêtes sont effectués par la coopération d'un ensemble d'agents, où nous affectons à chaque cluster un agent qui représentera son leader (appelé agent intra-cluster), en plus d'un agent qui coordonne les tâches entre les différents clusters (appelé agent inter-cluster). Chaque agent intra-cluster aura une vue globale sur son cluster (voir la figure IV.6), et l'agent inter-cluster aura une vue globale sur la grille de données (sur tous les clusters de la grille de données).

Donc, nous distinguons deux types d'agents :

L'agent inter-cluster qui est responsable de :

- La formation des clusters.
- Le calcul de la taille des quorums de lecture et d'écriture au sein de la grille (appliqués sur les clusters) en appliquant les formules (1) et (2).
- Le verrouillage et le déverrouillage des répliques des nœuds leaders impliqués dans un quorum lors des accès aux répliques (lecture/écriture).
- Diffusion des mises à jour aux leaders en cas d'une opération d'écriture.

Les agents intra-cluster qui représentent les nœuds leader des clusters (i.e. : chaque cluster possède un agent intra-cluster).
Chaque agent est responsable de :

- La mise à jour de la taille des quorums de lecture et d'écriture au sein de son cluster, en appliquant les formules (3) et (4).
- Le verrouillage et le déverrouillage des nœuds de son cluster impliqués dans un quorum lors des accès aux répliques (lecture/écriture).
- La mise à jour des différentes répliques de son cluster en cas d'une opération d'écriture.

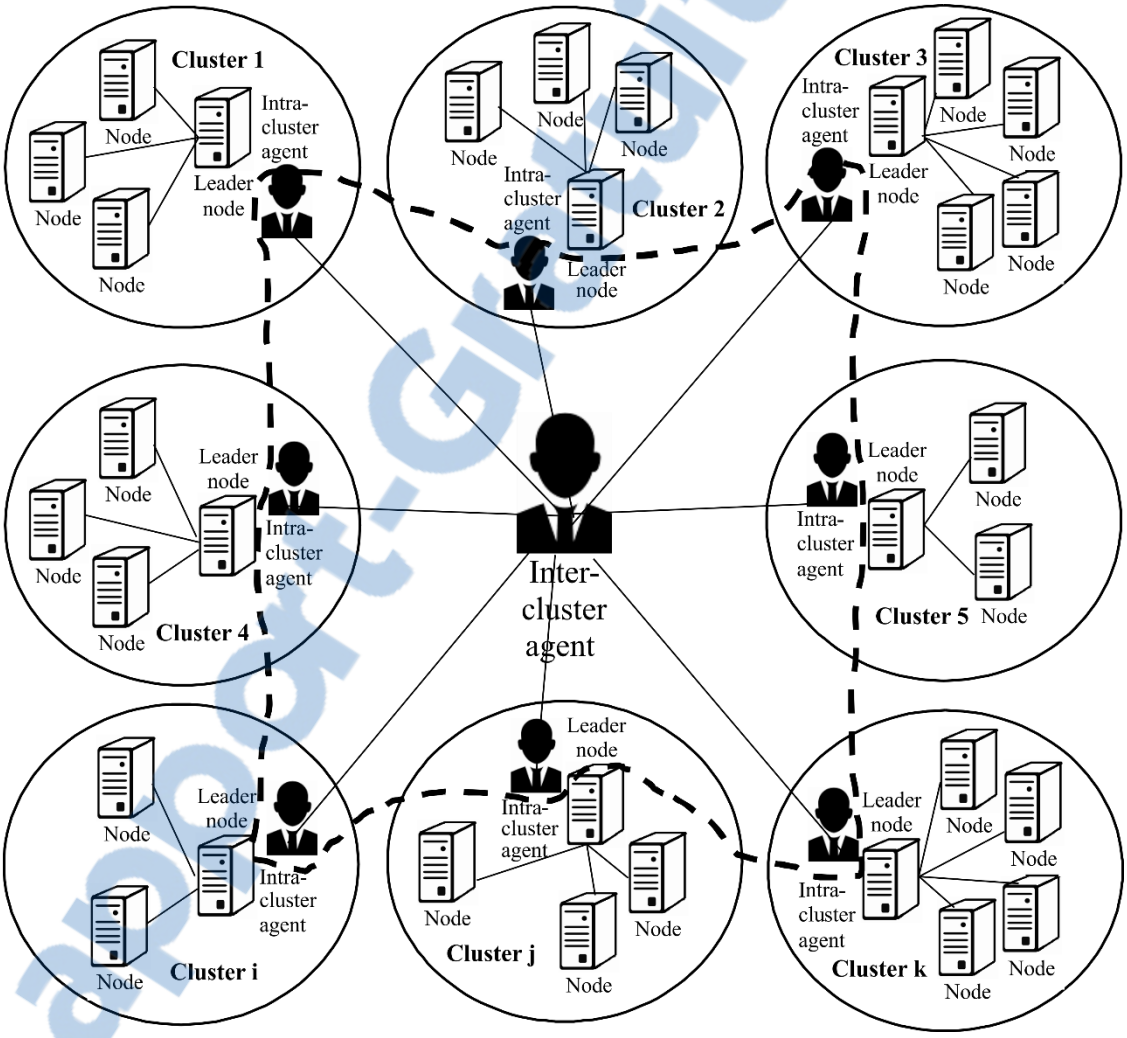

Figure IV.6 : Représentation schématique des nœuds, des clusters et des agents dans l'environnement de la grille.

Chaque réplique d'une donnée au sein de la grille est dotée d'un numéro de version qui sert à déterminer la réplique la plus récente. Quand un nœud demande l'accès en lecture ou en écriture à une donnée, une requête qui demande les numéros de version des différentes répliques est formulée dans un premier temps. Une fois la réplique la plus récente est déterminée (ayant le plus grand numéro de version), un

accès effectif (en lecture ou en écriture) est effectué sur le nœud possédant cette réplique (la plus récente).
Alors, lorsqu'un nœud demande l'accès à une donnée, il envoie sa requête au leader de son cluster. Une fois que son leader reçoit la requête, l'agent intra-cluster la transmet à l'agent inter-cluster.
L'agent inter-cluster envoie une requête à chaque agent intra-cluster, lui demandant de déterminer la réplique la plus récente au sein de son cluster. Chaque agent intra-cluster localise la réplique ayant la dernière version, ensuite il transmet l'adresse de cette réplique ainsi que sa version à l'agent inter-cluster.
L'agent inter-cluster collecte les réponses des différents agents intra-cluster, et lorsque le quorum (appliqué sur les clusters) sera atteint, il détermine la réplique la plus récente dans toute la grille et il transmet le numéro de version et l'adresse de cette réplique à l'agent intra-cluster qui a émis la requête (l'agent du cluster du nœud qui voulait accéder à la donnée).
Une fois l'opération d'accès réalisée, chaque agent intra-cluster libère les répliques verrouillées de son cluster, et l'agent inter-cluster libère les leaders verrouillés. En cas d'écriture, la mise à jour des répliques ainsi que leur versionement (nouvelles versions) doit être réalisée avant le déverrouillage.

Pour localiser la réplique la plus récente au sein de chaque cluster, chaque agent intra-cluster envoi une requête à chaque nœud de son cluster et qui possède une réplique, lui demandant sa version et son adresse. Ensuite il collecte les réponses des différents nœuds, et quand le quorum (appliqué sur les répliques de son cluster) sera atteint, il détermine la réplique la plus récente dans son cluster et il transmet le numéro de version et l'adresse de cette réplique à l'agent inter-cluster.

Le diagramme d'activité de la figure IV.7 illustre cette coopération entre agents pour répondre à une requête d'accès à une donnée émise par un nœud du cluster "*i*".

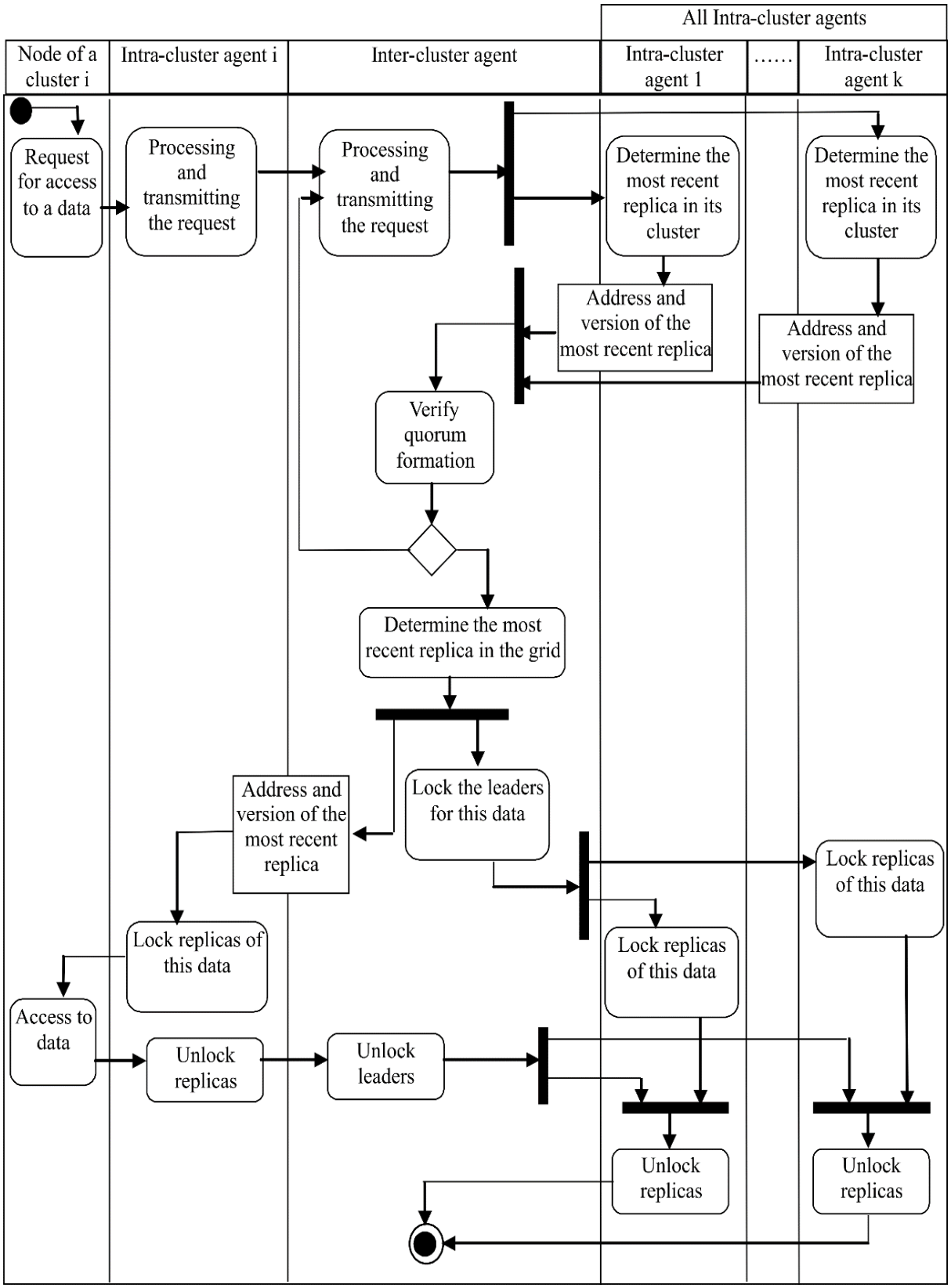

Figure IV.7 : Diagramme d'activité de coopération entre agents.

Le code suivant décrit l'algorithme à exécuter par les agents pour déterminer la réplique la plus récente lors d'une opération d'accès à une donnée.

Algorithm 3 : détermination de la réplique la plus récente

```
1:  C: set of element
2:  x: element
3:  Q: integer                          /* size of read or write quorum*/
4:  numb_resp ,version: integer
5:  BEGIN
6:  send request to any element ∈ C              /* for get latest version*/
7:  numb_resp←0
8:  version←0
9:  while numb_resp< Q  do
10:   Wait (response)                   /* wait for some time the answers  else failure */
11:   Lock(response.element)            /* lock the element for reading or writing*/
12:   numb_resp←numb_resp +1
13:   if version< response.version then
14:     version← response.version
15:     x← response.element             /*record the element containing the latest version*/
16:   end if
17: end while
18: Transmit (x.@ , x.version)          /*Transmit the address and the version of element x*/
19: END
```

Le code de l'algorithme 3 peut être exécuté en deux modes : soit en mode intra-cluster par un agent intra-cluster ou en mode inter-cluster par l'agent inter-cluster.
Dans cet algorithme, le mot "*element*" est générique, il peut représenter : soit un nœud possédant une réplique au sein d'un cluster si l'algorithme est exécuté en mode intra-cluster ; ou soit un leader d'un cluster si l'algorithme est exécuté en mode inter-cluster.
Dans le cas d'une requête d'accès en lecture, la variable *Q* représente la taille du quorum de lecture et la fonction *Lock* verrouille les répliques en mode lecture. Toutefois, dans le cas d'accès en écriture, la variable *Q* représente la taille du quorum d'écriture et la fonction *Lock* verrouille les répliques en mode écriture.

7. Les autres tâches des agents

D'autres tâches sont ajoutées aux responsabilités des agents :

- Afin d'améliorer les performances, la détermination de la réplique la plus récente ainsi que son adresse au sein d'un cluster peut être effectuée de manière asynchrone. En effet, quand un leader est libre et n'est pas sollicité, son agent intra-cluster peut déterminer la réplique la plus récente de son cluster.

- L'agent intra-cluster doit mettre à jour toutes les répliques de son cluster. Cette mise à jour peut être effectuée de manière asynchrone.

- Les nouvelles répliques sont créées dans un cluster par son agent intra-cluster. Cette création peut aussi être effectuée de manière asynchrone.

- L'agent intra-cluster doit décider si une nouvelle réplique doit être créée ou pas, si le nombre de répliques dans son cluster est limité.
  Dans certains cas, où le nombre de requêtes d'accès en écriture aux répliques est élevé, il est préférable de limiter (ou même de diminuer) le nombre de répliques, afin d'améliorer les coûts de la cohérence (réduction des coûts des mises à jours des répliques).

- L'agent intra-cluster peut supprimer une réplique jugée très ancienne.
  Dans un cluster, si l'agent intra-cluster constate que la différence entre le numéro de version d'une réplique et le numéro de version de la réplique à jour est trop grande, alors l'agent pourra choisir entre la suppression ou la mise à jour de la réplique en question.
  La décision de suppression ou de mise à jour d'une réplique peut être basée sur deux paramètres : l'ancienneté de la réplique et le nombre maximal de répliques autorisées dans le cluster.
  L'ancienneté d'une réplique peut être déterminée en utilisant un seuil. Si la différence entre le numéro de version d'une réplique et le numéro de version de la réplique actuelle est supérieure à un seuil donné, alors la réplique ancienne peut être supprimée.

- Lors du placement d'une nouvelle réplique, l'agent intra-cluster choisi le nœud le plus disponible dans son cluster (à part le leader).

- L'agent inter-cluster et les agents intra-cluster peuvent aussi coopérer pour répliquer une donnée.
  Lors d'une réplication, les agents peuvent facilement coopérer pour par exemple trouver la réplique la plus récente et choisir l'emplacement adéquat pour la nouvelle réplique, etc.

- La distribution des copies en groupes, et la coopération entre agents nous permet d'attribuer à chaque cluster sa propre politique de réplication. Et chaque agent intra-cluster peut répliquer selon la politique de réplication de son cluster.

8. Conclusion

Dans ce chapitre, nous avons présenté notre approche qui gère à la fois la réplication et la cohérence mutuelle des répliques. Elle est basée sur les quorums et sur la coopération entre agents. La conception de ces agents est réalisée à l'aide d'un système multi-agents. La cohérence mutuelle des répliques dans notre approche est basée sur les quorums. Et le système multi-agents gère la réplication (création,

placement, etc.) et la cohérence des différentes copies, dans le but d'améliorer les performances.

Toutes ces considérations sont conceptuelles et doivent être appuyées par des résultats pratiques. Dans le chapitre suivant, et en s'appuyant sur la simulation pour évaluer et étudier le comportement de notre stratégie, nous allons implémenter et comparer notre approche avec d'autres approches.

Chapitre V. Evaluation des performances

1. Introduction

Dans ce chapitre, nous allons étudier le comportement et évaluer les performances de notre approche à travers une série de simulations.
Nous commençons par la présentation des outils utilisés : simulateur et plate-forme des systèmes multi-agents. Ensuite, nous présentons et nous analysons les résultats de nos expérimentations. A partir de ces expérimentations nous montrons l'efficacité de notre solution.

Dans le chapitre précèdent, nous avons présenté deux méthodes de regroupement des nœuds : regroupement à base de la disponibilité et regroupement à base de voisinage. Pour évaluer notre approche, nous effectuerons une série d'expérimentations en utilisant deux simulateurs (un simulateur par méthode de regroupement) :
Le premier simulateur est "GREP-SIM" [Belalem, 07a] [Bouharaoua, 07], utilisé pour le regroupement à base de la disponibilité.
Le deuxième simulateur est "Optorsim[14]" [Bell, 03b] [Cameron, 06], utilisé pour le regroupement à base de voisinage.

2. Environnement de travail et paramètres de simulation

Les simulations ont été réalisées sous le système "Windows 7" installé sur une machine dotée d'un Processeur Intel Core I3 de 2.5 GHz, et d'une RAM de 2 Go.
L'implémentation de nos algorithmes ont été développés en langage Java en utilisant l'environnement de programmation "Eclipse".
Comme plate-forme des systèmes multi-agents, nous avons opté pour JADE[15]. Il nous a facilité l'implémentation de nos agents grâce à sa bibliothèque de classes (paquet logiciel), et il nous a facilement permis de l'intégrer dans les deux simulateurs, puisqu'il est développé en Java.

Les expérimentations avec les deux simulateurs ont été effectuées, en utilisant les paramètres définis dans la table V.1.

Nombre de données	20
Nombre de nœuds	500
Tailles des données	De 100 MB à 2 GB
Nombre de répliques	De 1 à 100 par donnée
Nombre de requêtes	1000

Table V.1 : Paramètres de simulation.

[14] Optorsim website : http://sourceforge.net/projects/optorsim.
[15] JADE website : http://jade.tilab.com.

3. Premier scénario de simulation

Dans cette première simulation, nous allons évaluer notre approche en se basant sur le paramètre de disponibilité des nœuds.

3.1. Environnement et outils de simulation pour le premier scénario

Pour évaluer notre approche, nous avons utilisé notre simulateur "GREP-SIM" étendu et amélioré [Bouharaoua, 07] [Belalem, 07a]. Ce simulateur nous permet de générer une topologie hiérarchique et plate à la fois (hybride). La topologie arborescente est inspirée de l'architecture de grille de données "CERN[16]".

Dans ce simulateur, chaque nœud dans la grille est capable de spécifier sa capacité de stockage, la performance de son processeur, les données répliquées localement, et l'historique détaillé des requêtes transitant par ce nœud (le passage d'une requête par ce nœud lors de son acheminement d'un client vers son nœud serviteur). Ce simulateur, nous permet d'ajouter, de supprimer et de placer les répliques de manière dynamique. Par conséquent, le nombre de répliques ne restera pas fixe (variable) durant la simulation. Ce qui nous permettra d'évaluer notre approche dans un environnement dynamique où le nombre de répliques change, et la comparer à deux autres approches : l'approche ROWA et surtout l'approche du Quorum à Consensus. Car avec cette dernière, c'est tout le système de la grille qui va être influencé par le changement du nombre de répliques, contrairement à notre approche où seulement le cluster où le nombre de répliques a changé qui va être influencé.

Nous avons effectué des modifications sur ce simulateur afin d'implémenter nos algorithmes et afin d'intégrer les agents dans le système. L'implémentation de ces agents est réalisée à l'aide de la plate-forme des systèmes multi-agents JADE.
Dans cette version modifiée du simulateur, chaque nœud possède une valeur indiquant la probabilité de sa disponibilité.

3.2. Modèle de grille simulé

La figure V.1 montre un exemple réduit de la topologie de la grille simulée. Elle est composé de 12 nœuds (un nœud racine et 11 nœuds intermédiaires) et de 12 feuilles représentant les clients. Ce simulateur nous permet de générer une topologie hiérarchique et plate avec un grand nombre de nœuds (500 nœuds dans la simulation ci-dessous). Cela nous permet également de générer et de distribuer des répliques des différentes données.

[16] CERN website : http://wlcg.web.cern.ch.

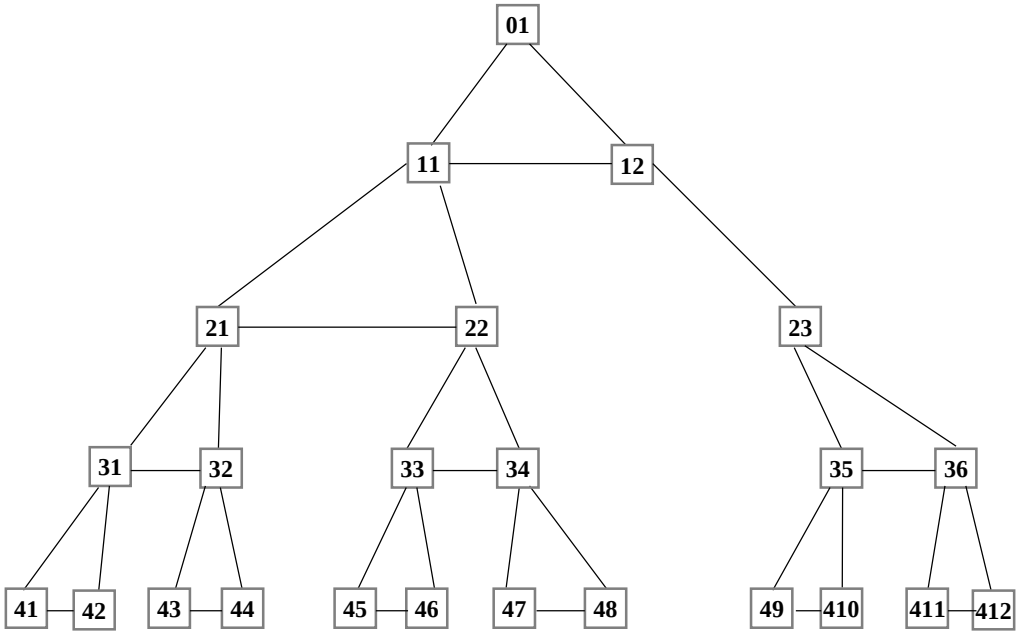

Figure V.1 : Structure de la grille simulée.

3.3. Expérimentations et résultats pour le premier scénario

Lors des expérimentations, le simulateur commence par la formation des clusters. Ensuite une distribution aléatoire des répliques sera effectuée. A partir de cette distribution et pour le même ordre chronologique d'arrivée des requêtes des clients, le simulateur calcule les temps de réponses pour chaque requête (lecture ou écriture) en appliquant les trois approches : ROWA, Quorum à Consensus (QC) et Quorum Imbriqué (IQ). Ainsi nous pourrons comparer notre approche aux deux autres approches.

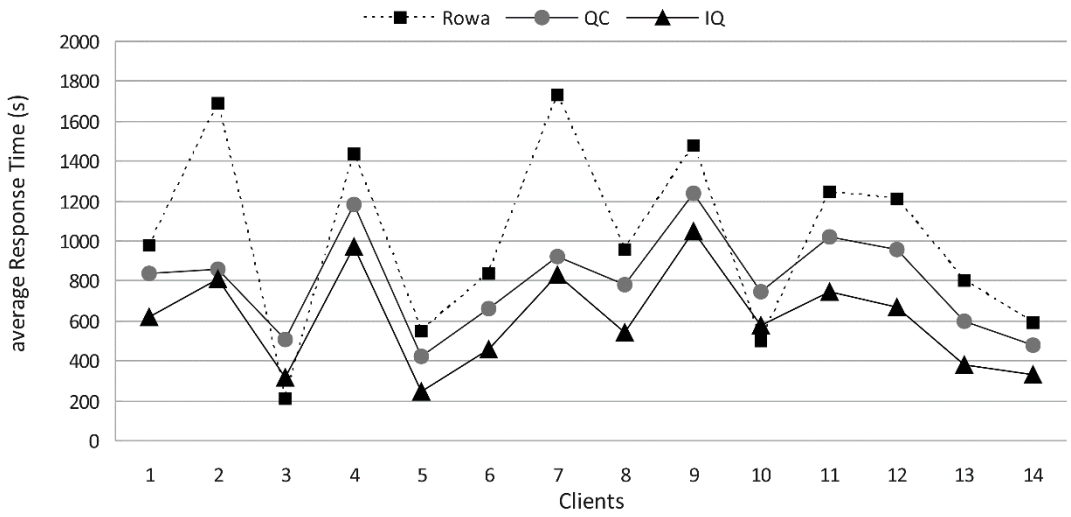

Figure V.2 : Temps de réponse moyen par clients.

Les résultats obtenus et modélisés par le graphe de la figure V.2 représentent les temps de réponses moyens de quelques clients (14 premiers clients). Elle montre que les performances sont nettement améliorées en utilisant notre approche. En effet, suivant les résultats obtenus dans la table V.2, nous constatons une diminution de 39,92% du temps de réponse moyen d'IQ par rapport à ROWA et de 23,82% par rapport à QC.

Approches	ROWA	QC	IQ
Moyenne générale des temps de réponses (seconde).	1017	802	611

Table V.2 : Moyenne générale des temps de réponses par clients.

Nous remarquons que pour les clients 3, et 10, les meilleurs temps de réponse sont obtenus par ROWA, car ces clients ont émis un nombre de requête d'écriture négligeable par rapport aux nombre de requêtes de lecture.

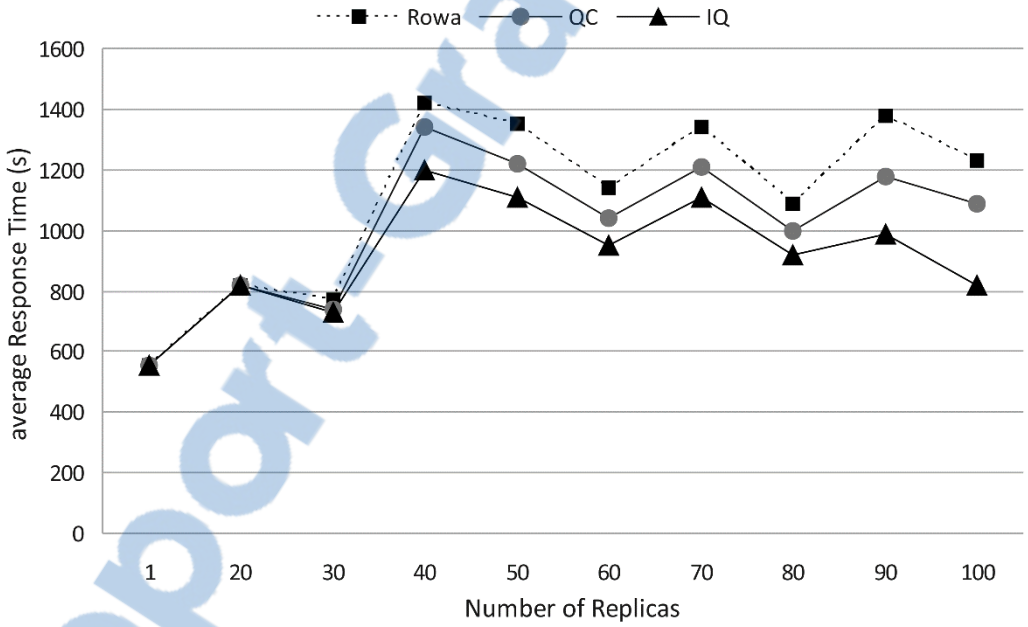

Figure V.3 : Temps de réponse moyen par nombre de répliques.

Les résultats obtenus et modélisés par le graphe de la figure V.3 représentent les temps de réponses moyens par nombre de répliques. Elle montre que notre approche améliore nettement les performances. Les résultats obtenus dans la table V.3 montrent que le temps de réponse moyen d'IQ vaut 17,12% de moins que celui de ROWA, et vaut 9,80% de moins que celui de QC.

Approches	ROWA	QC	IQ
Moyenne générale des temps de réponses (seconde).	1110	1020	920

Table V.3 : Moyenne générale des temps de réponses par nombre de répliques.

Les résultats suivants montrent où notre approche est vraiment intéressante. Les résultats obtenus et modélisés par le graphe de la figure V.4.a représentent les temps de réponses moyens par nombre de requêtes d'écriture (nombre de requêtes d'écritures parmi 1000 requêtes). Nous constatons selon les résultats obtenus dans la table V.4 que l'approche IQ améliore les temps de réponse moyen de 75,40% par rapport à celui de ROWA, et améliore les temps de réponse moyen de 52,05% par rapport à celui de QC.

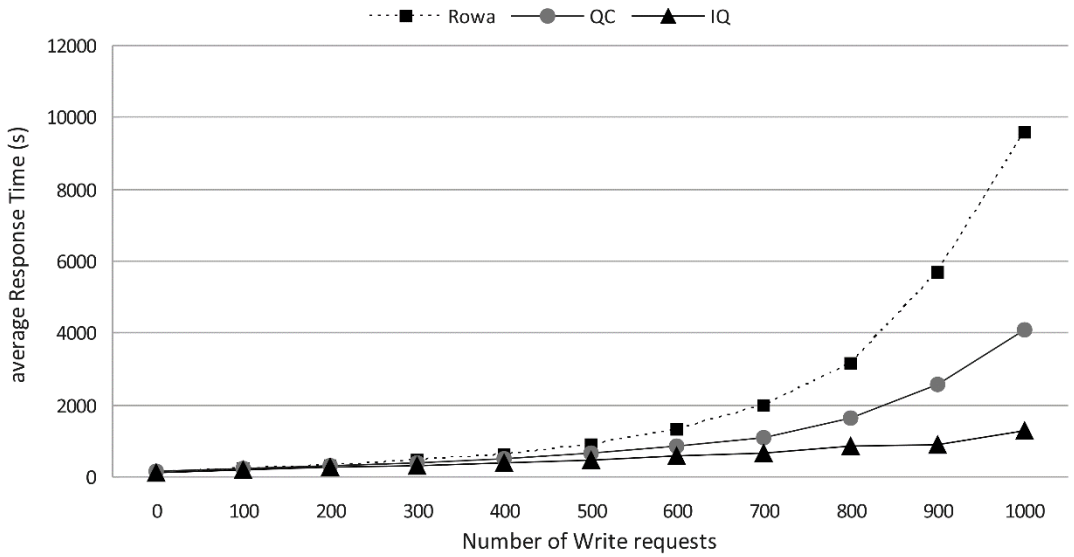

Figure V.4.a : Temps de réponse moyen par type de requêtes.

Approches	ROWA	QC	IQ
Moyenne générale des temps de réponses (seconde).	2423	1243	596

Table V.4 : Moyenne générale des temps de réponses par type de requêtes.

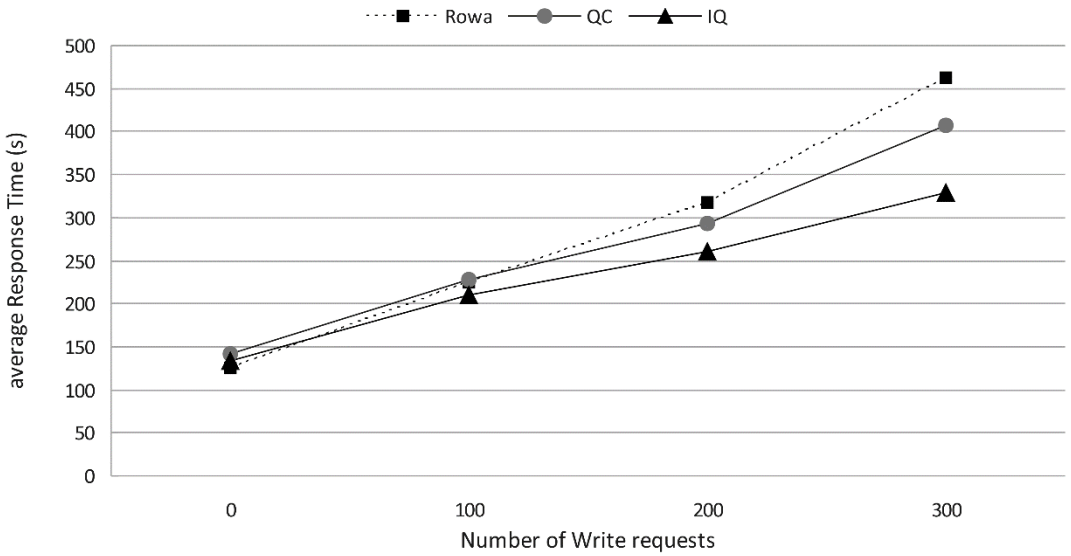

Figure V.4.b : Temps de réponse moyen par type de requêtes (figure détaillée).

La figure V.4.b (figure détaillée de la figure V.4.a) montre que si le nombre de requêtes d'écritures est nul ou très faible, l'approche ROWA est meilleure que l'approche IQ, mais l'approche IQ reste toujours meilleure que QC.

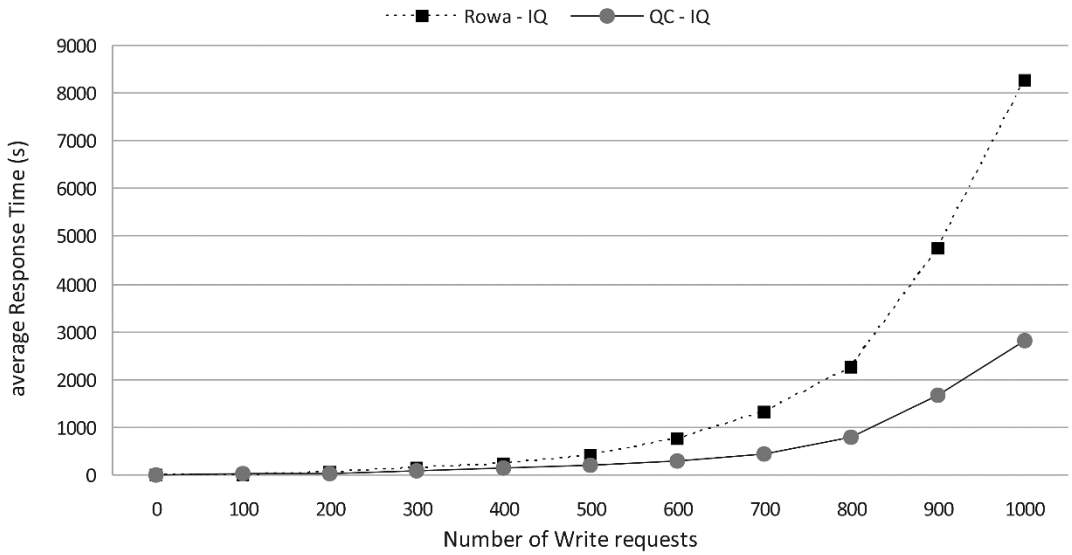

Figure V.4.c : Ecart des temps de réponse moyen par type de requêtes.

Nous remarquons aussi que notre approche est meilleure que les deux autres approches, puisque l'écart, entre les temps de réponses moyens d'IQ d'un côté et de ROWA et QC d'un autre côté, devient de plus en plus important quand le nombre de requêtes d'écriture augmente. Cet écart est presque exponentiel comme illustré dans la figure V.4.c.

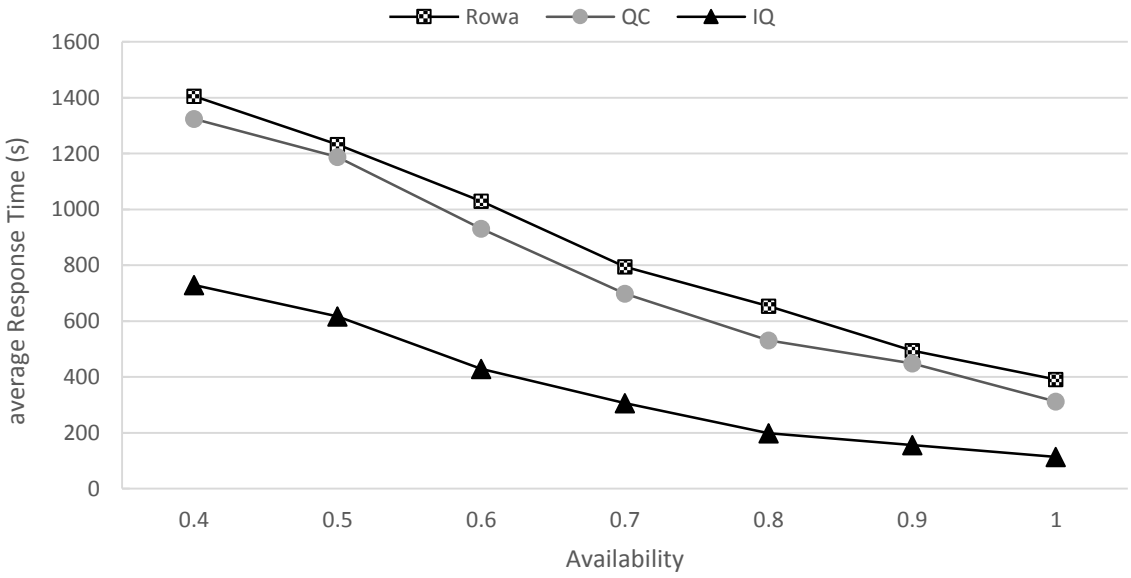

Figure V.5 : Temps de réponse moyen par disponibilité des nœuds.

Les résultats obtenus et modélisés par le graphe de la figure V.5 représentent les temps de réponses moyens par disponibilité des nœuds. Les résultats obtenus dans la table V.5 montrent que le temps de réponse moyen d'IQ vaut 57,53% de moins que celui de ROWA, et vaut 53.09% de moins que celui de QC.
Cette amélioration s'explique par le fait que la répartition des nœuds, dans notre approche, est basée sur la disponibilité et par le fait que les agents place les répliques dans les nœuds les plus disponibles.

Approches	ROWA	QC	IQ
Moyenne générale des temps de réponses (seconde).	857	776	364

Table V.5 : Moyenne générale des temps de réponses par disponibilité des nœuds.

4. Deuxième scénario de simulation

Pour mieux évaluer notre approche, il serait pertinent d'effectuer d'autres expérimentations sur un autre simulateur.
Dans cette deuxième simulation, nous évaluons notre approche en se basant sur le paramètre de voisinage des nœuds.

4.1. Environnement et outils de simulation pour le deuxième scénario

Cette fois ci, nous changeons le simulateur pour évaluer notre approche. Nous utilisons le simulateur "Optorsim". C'est un simulateur pour les environnements de grille de données. Il est a été développé et écrit en langage Java, dans le cadre du projet Européen des grilles de données (European Data Grid project) [Cameron, 06], pour simuler les différentes techniques de réplication.
Ce simulateur nous a permis d'évaluer notre approche dans un environnement dynamique et aussi la comparer aux deux autres approches (ROWA et Quorum à Consensus).
Nous avons aussi modifié ce simulateur afin d'intégrer la plate-forme des systèmes multi-agents JADE et afin d'implémenter nos algorithmes.

4.2. Expérimentations et résultats pour le deuxième scénario

De même que le premier scénario de simulation, la formation des clusters, précède la distribution des répliques. Et de même que la simulation précédente, nous calculons les temps de réponses pour chaque requête (lecture ou écriture) en appliquant les trois approches. Ensuite nous comparons notre approche avec les deux autres.

Les résultats obtenus et modélisés par le graphe de la figure V.6 représentent les temps de réponses moyens par nombre de répliques. Notre approche améliore nettement les performances. La table V.6 montre que le temps de réponse moyen d'IQ vaut 18,36% de moins que celui de ROWA, et vaut 10,32% de moins que celui de QC.

Approches	ROWA	QC	IQ
Moyenne générale des temps de réponses (seconde).	1160	1056	947

Table V.6 : Moyenne générale des temps de réponses par nombre de répliques du deuxième scénario.

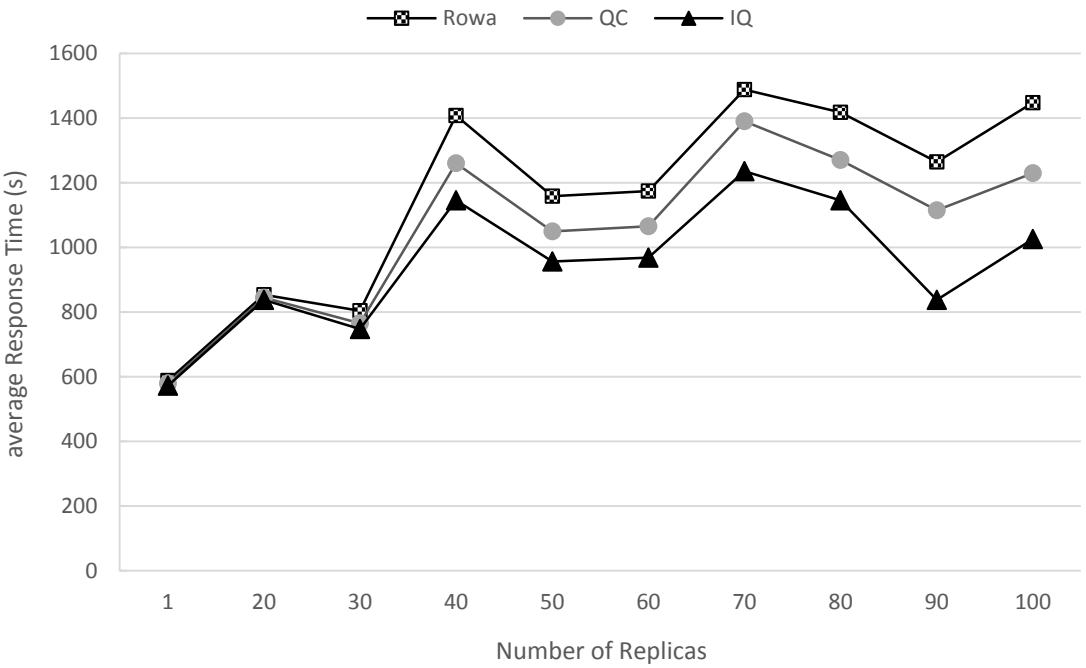

Figure V.6 : Temps de réponse moyen par nombre de répliques du deuxième scénario.

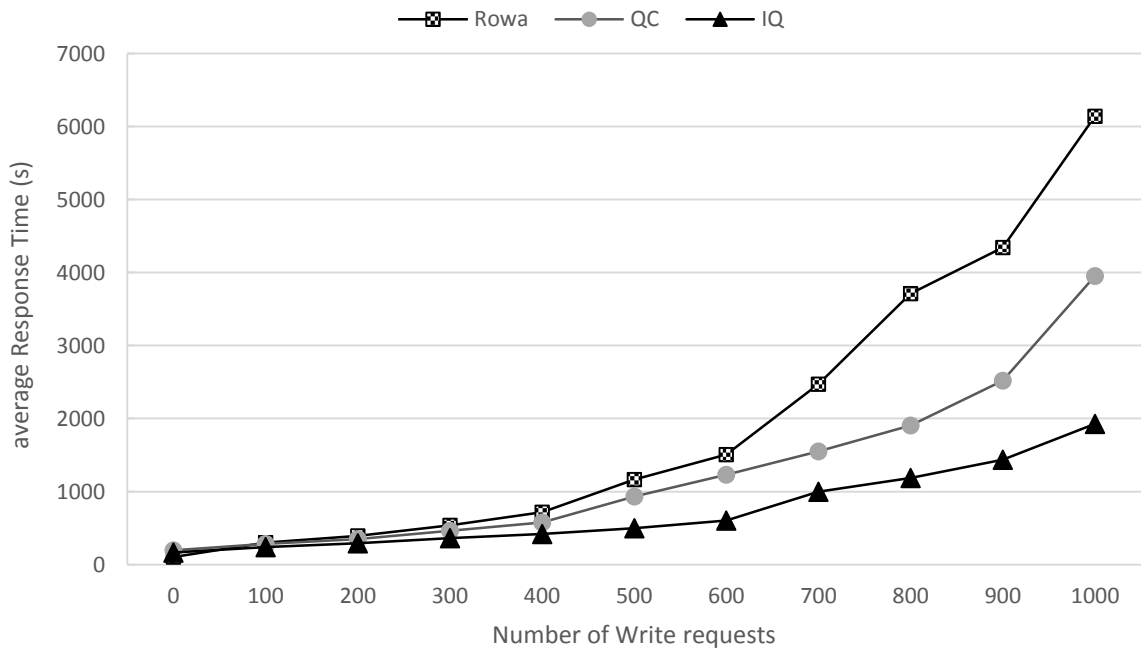

Figure V.7 : Temps de réponse moyen par type de requêtes du deuxième scénario.

Les résultats modélisés par le graphe de la figure V.7 représentent les temps de réponses moyens par nombre de requêtes d'écriture. Les requêtes d'écritures sont émises parmi 1000 requêtes. Nous constatons dans la table V.7 que l'approche d'IQ améliore les temps de réponse moyen de 62,08% par rapport à celui de ROWA, et améliore les temps de réponse moyen de 41,84% par rapport à celui de QC.

Approches	ROWA	QC	IQ
Moyenne générale des temps de réponses (seconde).	1946	1269	738

Table V.7 : Moyenne générale des temps de réponses par type de requêtes du deuxième scénario.

5. Conclusion

Dans ce chapitre, nous avons réalisé une série d'expérimentations afin d'évaluer l'approche proposée. Pour une meilleure évaluation, nous avons utilisée deux simulateurs différents, qui ont été modifié afin d'intégrer nos algorithmes et le système multi-agents.

Plusieurs tests ont été effectués selon plusieurs paramètres de configuration. Les résultats expérimentaux obtenus avec les deux simulateurs ont révélé une amélioration des performances. En effet, les temps de réponses obtenus par notre approche "Quorum Imbriqué" sont nettement meilleurs que les temps de réponses obtenus par les deux autres approches. Ces performances résultent du fait que dans notre stratégie, la cohérence mutuelle des copies est prise en compte en même temps que la réplication des données et en se basant sur les agents. Elles résultent aussi du fait que les tailles des quorums dans notre approche sont significativement plus faibles que celles des deux autres approches.

Conclusion générale

Dans ce travail, nous avons contribué à la réplication des données dans les grilles de données. Nous avons conçu une approche basée sur les quorums et sur un système multi-agents pour améliorer les performances dans la grille. Cette technique de quorum proposée, appelée Quorum Imbriqué, a été formulé en un sujet de minimisation de la taille des quorums de lecture et d'écriture, pour minimiser le nombre de répliques impliquées dans les opérations de lecture ou d'écriture, en se basant sur la répartition des répliques en groupes.

La réplication permet d'améliorer les performances d'accès aux données de la grille. Bien qu'elle soit avantageuse, elle présente comme même un inconvénient majeur, à savoir la cohérence mutuelle des répliques où toutes les répliques d'une même donnée doivent être identiques. Donc, dans n'importe quelle stratégie de réplication, le choix de la politique de réplication et le choix de la politique de gestion de la cohérence sont des facteurs cruciaux dont dépendent les performances du système de la grille.
Dans ce contexte, notre contribution porte sur deux volets :
Le premier volet concerne la gestion des répliques et la cohérence de ces répliques. En effet, nous avons proposé une stratégie de réplication qui traite à la fois les problèmes liés à la gestion des répliques (placement, nombre, sélection, etc.), et les problèmes liés à la cohérence des répliques.
Le deuxième volet concerne l'intégration des agents pour aider le système à prendre les décisions sur la gestion des répliques.

Notre approche dépend essentiellement de la distribution de l'ensemble des nœuds en clusters. Nous avons présenté deux méthodes de regroupement des nœuds : l'une est basée sur le paramètre de voisinage des nœuds et l'autre est basée sur le paramètre de disponibilité des nœuds.
Ces distributions, nécessaires pour l'application de notre technique de Quorum Imbriqué, servent à la fois pour la réplication et pour la cohérence. Elles nous ont aussi permis d'en profiter pour octroyer à chaque cluster un agent qui représentera son leader, et qui servira dans la prise des décisions pour la gestion des répliques, en coopérant avec les autres agents.

Comme la plupart des travaux connexes, notre approche vise à minimiser les tailles de quorum. Cependant, dans ces travaux, les trois points suivants ne sont pas considérés en même temps, à savoir le nombre de répliques, la mise à l'échelle et les performances de temps de réponse. Dans notre travail, nous prenons en compte ces trois points simultanément. En effet, nous avons proposé une approche de gestion des répliques dans les systèmes distribués à grande échelle (tels que les grilles de données), où le nombre de répliques varie (non fixe) et qui vise à améliorer les performances en diminuant la taille des quorums.

Pour évaluer notre approche, nous avons opté pour la simulation. Notre approche est comparée à deux autres approches basées sur les techniques de ROWA et du Quorum à Consensus pour gérer la cohérence.
Les résultats obtenus montrent que notre approche (Quorum Imbriqué) améliore les performances d'accès aux données dans les systèmes à grande échelle tels que les grilles de données. En effet, ces résultats montrent que notre approche est considérablement meilleure que les deux autres approches (ROWA et QC), puisque les tailles des quorums de lecture et d'écriture dans notre approche sont significativement plus réduites que les deux autres approches.

Perspectives

Ce travail nous permettra d'envisager quelques perspectives de recherche intéressantes.
Dans le futur, nous projetons d'étendre ce travail dans plusieurs directions.

- Etude de la contribution du clustering à notre approche :
  L'élargissement du champ d'étude sur le regroupement des nœuds vers les techniques et les algorithmes de clustering. Nous avons constaté qu'une distribution intelligente des répliques améliorera davantage notre approche.

- Décision distribuée :
  Il serait judicieux d'enrichir notre approche par le principe de collaboration et de négociation distribuée entre agents.

- L'élargissement du champ d'étude sur les quorums :
  L'ajout des techniques de pondération des votes à la formation des quorums.

- Nous pourrons également élargir le champ d'étude vers l'équilibrage de charge, et vers la probabilité de la disponibilité des nœuds pour améliorer les performances.

- Implémentation dans une grille réelle :
  Nous projetons de valider notre approche proposée sur des grilles de données réelles.

- Enfin, nous projetons d'étudier et d'adapter nos propositions dans un environnement de Cloud.

Bibliographie

[Abawajy, 14] Abawajy, J. H., & Deris, M. M. (2014). Data replication approach with consistency guarantee for data grid. *IEEE Transactions on Computers*, *63*(12): 2975-2987.

[Agrawal, 90] Agrawal, D., & El Abbadi, A. (1990). The Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data. In *VLDB***,** Vol. 90, pp. 243-254.

[Alumuttairi, 10] Almuttairi, R. M., Wankar, R., Negi, A., & Rao, C. R. (2010). Replica selection in data grids using preconditioning of decision attributes by k-means clustering (K-RSDG). In *Second Vaagdevi International Conference on Information Technology for Real World Problems (VCON' 2010),* pp. 18-23.

[Amjad, 12] Amjad, T., Sher, M., & Daud, A. (2012). A survey of dynamic replication strategies for improving data availability in data grids. *Future Generation Computer Systems*, *28*(2): 337-349.

[Andronikou, 12] Andronikou, V., Mamouras, K., Tserpes, K., Kyriazis, D., & Varvarigou, T. (2012). Dynamic QoS-aware data replication in grid environments based on data "importance". *Future Generation Computer Systems*, *28*(3): 544-553.

[Barak, 98] Barak, A., & La'adan, O. (1998). The MOSIX multicomputer operating system for high performance cluster computing. *Future Generation Computer Systems*, *13*(4-5): 361-372.

[Barrett, 97] Barrett, R., Maglio, P. P., & Kellem, D. C. (1997). Web browser intelligence: Opening up the web. In *Compcon'97. Proceedings, IEEE*, pp. 122-123.

[Basmadjian, 11] Basmadjian, R., & de Meer, H. (2011). An Arbitrary 2D Structured Replica Control Protocol. In *OASIcs-OpenAccess Series in Informatics*, Vol. 17, pp.157-168. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[Belalem, 07a] Belalem, G., & Bouharaoua, F. (2007). Dynamic strategy of placement of the replicas in data grid. *The 9th International Conference Parallel Computing Technologies, PaCT 2007, Pereslavl-Zalessky, Russia, September 3-7, 2007, LNCS 4671, pp. 496-506.*

[Belalem, 07b] Belalem, G., & Slimani, Y. (2007). A hybrid approach to replica management in data grids. *International Journal of Web and Grid Services* (IJWGS), Inderscience, *3*(1): 2-18.

[Belalem, 08] Belalem, G., Yagoubi, B., & Bouamama, S. (2008). An Approach Based on Market Economy for Consistency Management in Data Grids with OptorSim Simulator. *International Journal of Information Technology and Web Engineering (IJITWE)*, *3*(3):1-16.

[Bell, 03a] Bell, W. H., Cameron, D. G., Carvajal-Schiaffino, R., Millar, A. P., Stockinger, K., & Zini, F. (2003). Evaluation of an economy-based file replication strategy for a data grid. In *International Symposium on Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM,* pp. 661-668.

[Bell, 03b] Bell, W. H., Cameron, D. G., Millar, A. P., Capozza, L., Stockinger, K., & Zini, F. (2003). Optorsim: A grid simulator for studying dynamic data replication strategies. *International Journal of High Performance Computing Applications*, *17*(4): 403-416.

[Bellifemine, 99] Bellifemine, F., Poggi, A., & Rimassa, G. (1999). JADE–A FIPA-compliant agent framework. In *Proceedings of PAAM*, Vol. 99, No. 97-108, p. 33.

[Ben Charrada, 10] Ben Charrada, F., Ounelli, H., & Chettaoui, H. (2010). An efficient replication strategy for dynamic data grids. In *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC'2010),* pp. 50-54.

[Beni, 93] Beni, G., & Wang, J. (1993). Swarm intelligence in cellular robotic systems. In *Robots and Biological Systems: Towards a New Bionics?* pp. 703-712. Springer, Berlin, Heidelberg.

[Benmerzoug, 09] Benmerzoug, D. (2009). *Modèles et outils formels pour l'intégration d'applications d'entreprises,* Thèse de Doctorat, Décembre 2009, Université Pierre et Marie Curie.

[Bernstein, 84] Bernstein, P. A., & Goodman, N. (1984). An algorithm for concurrency control and recovery in replicated distributed databases. *ACM Transactions on Database Systems (TODS)*, *9*(4) :596-615.

[Bonnel, 08] Bonnel, N. (2008). *Adapnet: Stratégies adaptatives pour la gestion de données distribuées sur un réseau P2P,* Thèse de Doctorat, Décembre 2008, Université de Bretagne Sud.

[Bote-Lorenzo, 04] Bote-Lorenzo, M., Dimitriadis, Y., & Gómez-Sánchez, E. (2004). Grid characteristics and uses: a grid definition. In *Grid Computing*, pp. 291-298. Springer Berlin/Heidelberg.

[Bouharaoua, 07] Bouharaoua, F. (2007). *Stratégie de réplication dynamique dans les grilles de données*, These de Magister, Juin 2007, Université Abdelhamid Ibn Badis, Mostaganem, Algérie.

[Bouharaoua, 17] Bouharaoua, F., & Belalem, G. (2017). A quorum-based intelligent replicas management in data grids to improve performances. *Multiagent and Grid Systems*, *13*(2) :143-161.

[Boussebough, 11] Boussebough, I. (2011). *Les systèmes multi-agents dynamiquement adaptables*, Thèse de Doctorat, Juillet 2011, Université Mentouri, Constantine, Algérie.

[Bousseta, 98] Boussetta, S. (1998). *Stratégies d'exécution de plans d'un système multiagent*, Thèse de Doctorat, Paris 9.

[Brandolese, 00] Brandolese, A., Brun, A., & Portioli-Staudacher, A. (2000). A multi-agent approach for the capacity allocation problem. *International Journal of Production Economics*, *66*(3):269-285.

[Budhiraja, 93] Budhiraja, N., Marzullo, K., Schneider, F. B., & Toueg, S. (1993). The primary-backup approach. *Distributed systems*, *Vol. 2*, pp. 199-216.

[Buyya, 00] Buyya, R., Chapin, S., & DiNucci, D. (2000). Architectural models for resource management in the grid. *Grid Computing - GRID 2000*, pp.18-35.

[Buyya, 02 a] Buyya, R. (2002). *Economic-based Distributed Resource Management and Scheduling for Grid Computing*, Thèse de Doctorat, Avril 2002 , Monash University, Melbourne, Australia.

[Buyya, 02b] Buyya, R., Abramson, D., Giddy, J., & Stockinger, H. (2002). Economic models for resource management and scheduling in grid computing. *Concurrency and computation: practice and experience*, *14*(13-15):1507-1542.

[Cameron, 06] Cameron, D. G., Schiaffino, R. C., Ferguson, J., Millar, P., Nicholson, C., Stockinger, K., & Zini, F. (2006). OptorSim v2.1 installation and user guide.

[Chaib-draa, 01] Chaib-Draa, B., Jarras, I., & Moulin, B. (2001). Systèmes multi-agents: principes généraux et applications. *Principes et architectures des systèmes multi-agents*. Ed. Hermès.

[Chaib-draa, 02] Chaib-Draa, B., & Gageut, L. (2002). Aspects formels des Systèmes Multi-Agents. *Organisation et applications des systèmes multi-agents, Hermès*, pp.109-132.

[Chaib-draa, 95] Chaib-draa, B. (1995). Industrial applications of distributed AI. *Communications of the ACM, 38(11):* 49-53.

[Chang, 08] Chang, R. S., & Chang, H. P. (2008). A dynamic data replication strategy using access-weights in data grids. *The Journal of Supercomputing*, *45*(3): 277-295.

[Chervenak, 00] Chervenak, A., Foster, I., Kesselman, C., Salisbury, C., & Tuecke, S. (2000). The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of network and computer applications*, *23*(3): 187-200.

[Chervenak, 08] Chervenak, A., Schuler, R., Kesselman, C., Koranda, S., & Moe, B. (2008). Wide area data replication for scientific collaborations. *International journal of high performance computing and networking*, *5*(3), 124-134.

[Cheung, 92] Cheung, S. Y., Ammar, M. H., & Ahamad, M. (1992). The grid protocol: A high performance scheme for maintaining replicated data. *IEEE Transactions on Knowledge and Data Engineering*, *4*(6):582-592.

[Choi, 12] Choi, S. C., & Youn, H. Y. (2012). Dynamic hybrid replication effectively combining tree and grid topology. *The Journal of Supercomputing*, *59*(3):1289-1311.

[Chu-Caroll, 95] Chu-Carroll, J., & Carberry, S. (1995). Conflict detection and resolution in collaborative planning. In *International Workshop on Agent Theories, Architectures, and Languages*, pp. 111-126. Springer, Berlin, Heidelberg.

[Cibej, 05] Cibej, U., Slivnik, B., & Robic, B. (2005). The complexity of static data replication in data grids. *Parallel Computing*, *31*(8):900-912.

[Corkill, 91] Corkill, D. D. (1991). Blackboard systems. *AI expert, 6(9)*:40-47.

[Cox, 02] Cox, R., Muthitacharoen, A., &Morris, R. (2002). Serving DNS using Chord. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, LNCS 2429, pp. 155–165.

[Dabek, 01] Dabek, F., Kaashoek, M. F., Karger, D., Morris, R., & Stoica, I. (2001). Wide-area cooperative storage with CFS. In *ACM SIGOPS Operating Systems Review,* 35(5):202-215.

[Davis, 83] Davis, R., & Smith, R. G. (1983). Negotiation as a metaphor for distributed problem solving. *Artificial intelligence*, *20*(1):63-109.

[Dogan, 09] Dogan, A. (2009). A study on performance of dynamic file replication algorithms for real-time file access in data grids. *Future Generation Computer Systems*, *25*(8):829-839.

[Doyen, 05] Doyen, G. (2005). *Supervision des réseaux et services pair à pair*, Thèse de Doctorat, Décembre 2005, Université Henri Poincaré-Nancy I.

[Drapeau, 03] Drapeau, S. (2003). *RS2.7 : un canevas Adaptable de Services de Duplication*, Thèse de Doctorat, Juin 2003, Institut National Polytechnique de Grenoble-INPG.

[Durfee, 90] Durfee, E. H., & Montgomery, T. A. (1990). A Hierarchical Protocol for Coordinating Mulitagent Behaviors. In *AAAI*, pp. 86-93.

[Elghirani, 09] Elghirani, A., Subrata, R., & Zomaya, A. Y. (2009). Intelligent scheduling and replication: a synergistic approach. *Concurrency and Computation: Practice and Experience*, *21*(3):357-376.

[Ferber, 95] Ferber, J. (1995). *Les systèmes multi-agents: vers une intelligence collective*. InterEditions.

[Ferreira, 03] Ferreira, L., Berstis, V., Armstrong, J., Kendzierski, M., Neukoetter, A., Takagi, M., Bing-Wo, R., Amir, A., Murakawa, R., Hernandez O., & Magowan, J. (2003). Introduction to grid computing with globus. *IBM Redbooks*, *9*.

[Finin, 94] Finin, T., Fritzson, R., McKay, D., & McEntire, R. (1994). KQML as an agent communication language. In *Proceedings of the third international conference on Information and knowledge management*, pp. 456-463.

[Foster, 02] Foster, I., Kesselman, C., Nick, J. M., & Tuecke, S. (2002). Grid services for distributed system integration. *Computer*, *35*(6):37-46.

[Foster, 99] Foster, I., & Kesselman, C. (1999). I. Foster and C. Kesselman. *Computational Grids.* In The Grid: Blueprint for a Future Computing Infrastructure.Morgan Kaufmann Publishers, San Francisco, USA, pp. 15-52.

[Gasser, 87] Gasser, L., Braganza, C., & Herman, N. (1987). MACE: A flexible testbed for distributed AI research. In *Distributed Artificial Intelligence, Volume I*, pp. 119-152.

[Gifford, 79] Gifford, D. K. (1979). Weighted voting for replicated data. In *Proceedings of the seventh ACM symposium on Operating systems principles*, pp. 150-162.

[Goel, 07] Goel, S., & Buyya, R. (2007). Data replication strategies in wide-area distributed systems. Chapter Book, In *Enterprise service computing: from concept to deployment* (pp. 211-241). IGI Global.

[Grace, 14] Grace, R. K., & Manimegalai, R. (2014). Dynamic replica placement and selection strategies in data grids—a comprehensive survey. *Journal of Parallel and Distributed Computing*, *74*(2):2099-2108.

[Graupner, 03] Graupner, S., Pruyne, J., & Singhal, S. (2003). Making the utility data center a power station for the enterprise grid. *HP Laboratories Technical Report, HPL-2003-53*, *131*.

[Gutknecht, 00] Gutknecht, O., & Ferber, J. (2000). The MadKit agent platform architecture. In *Workshop on Infrastructure for Scalable Multi-Agent Systems at the International Conference on Autonomous Agents*, pp. 48-55.

[Hughes, 05] Hughes, D., Coulson, G., & Walkerdine, J. (2005). Free riding on Gnutella revisited: the bell tolls?. *IEEE distributed systems online*, *6*(6):1-18.

[Latip, 13] Latip, R., Mizan, T., Ghazali, N. F., Kadir, R. A., & Hanandeh, F. A. (2013). Replica Control Protocol: Triple Quorum Replication (TQR) in *the 5th International Conference on* Data Grid, *Computer Science and Information Technology (CSIT), March 2013,* pp. 303-307.

[Lee, 05] Lee, H. M., & Yang, C. H. (2005). A distributed backup agent based on grid computing architecture. In *Proceedings of the 9th international conference on Knowledge-Based Intelligent Information and Engineering Systems-Volume Part II*, pp. 1252-1257.

[Ma, 13] Ma, J., Liu, W., & Glatard, T. (2013). A classification of file placement and replication methods on grids. *Future Generation Computer Systems*, *29*(6):1395-1406.

[Malone, 87] Malone, T. W. (1987). Modeling coordination in organizations and markets. *Management science*, *33*(10) :317-1332.

[Mamat, 08] Mamat, A., Radi, M., Deris, M. M., & Ibrahim, H. (2008). Performance of update propagation techniques for data grid. In *International Conference on Computer and Communication Engineering, ICCCE 2008,* pp. 332-335.

[Mansouri, 13] Mansouri, N., & Dastghaibyfard, G. H. (2013). Enhanced dynamic hierarchical replication and weighted scheduling strategy in data grid. *Journal of Parallel and Distributed Computing*, *73*(4):534-543.

[Mitton, 06] Mitton, N., Busson, A., & Fleury, E. (2006). Efficient broadcasting in self-organizing sensor networks. *International Journal of Distributed Sensor Networks*, *2*(2):161-187.

[Mokadem, 15] Mokadem, R., & Hameurlain, A. (2015). Data replication strategies with performance objective in data grid systems: a survey. *International Journal of Grid and Utility Computing*, *6*(1):30-46.

[Moulin, 96] Moulin, B., & Chaib-Draa, B. (1996). An overview of distributed artificial intelligence. *Foundations of distributed artificial intelligence*, *1*, pp.3-55.

[Naseera, 09] Naseera, S., & Murthy, K. M. (2009). Agent based replica placement in a data grid environment. In *First International Conference on Computational Intelligence, Communication Systems and Networks, 2009. CICSYN'09,* pp. 426-430.

[Nicholson, 08] Nicholson, C., Cameron, D. G., Doyle, A. T., Millar, A. P., & Stockinger, K. (2008). Dynamic data replication in lcg 2008. *Concurrency and Computation: Practice and Experience*, *20*(11):1259-1271.

[Nukarapu, 11] Nukarapu, D., Tang, B., Wang, L., & Lu, S. (2011). Data replication in data intensive scientific applications with performance guarantee. *IEEE Transactions on Parallel and Distributed Systems*, *22*(8):1299-1306.

[Nwana , 99] Nwana, H. S., Ndumu, D. T., Lee, L. C., & Collis, J. C. (1999). ZEUS: a toolkit for building distributed multiagent systems. *Applied Artificial Intelligence*, *13*(1-2):129-185.

[On, 03] On, G., Schmitt, J., & Steinmetz, R. (2003). The effectiveness of realistic replication strategies on quality of availability for peer-to-peer systems. In *Proceedings. Third International Conference on Peer-to-Peer Computing, P2P'2003,* pp. 57-64.

[Oram, 03] Oram, A. (2003). Peer-to-peer: Harnessing the power of disruptive technologies. *SIGMOD Record*, *32*(2) : 57-58.

[Perez, 10] Perez, J. M., García-Carballeira, F., Carretero, J., Calderón, A., & Fernández, J. (2010). Branch replication scheme: A new model for data replication in large scale data grids. *Future Generation Computer Systems*, *26*(1):12-20.

[Ranganathan, 01] Ranganathan, K., & Foster, I. T. (2001). Identifying Dynamic Replication Strategies for a High-Performance Data Grid. In *Proceedings of the Second International Workshop on Grid Computing* (pp. 75-86). Springer-Verlag.

[Ranganathan, 02 a] Ranganathan, K., Iamnitchi, A., & Foster, I. (2002). Improving data availability through dynamic model-driven replication in large peer-to-peer communities. In *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid,* pp. 376-376.

[Ranganathan, 02 b] Ranganathan, K., & Foster, I. (2002). Decoupling computation and data scheduling in distributed data-intensive applications. In *Proceedings. 11th IEEE International Symposium on High Performance Distributed Computing, 2002, HPDC-11 2002,* pp. 352-358.

[Raynal, 92] Raynal, M. (1992). *Gestion des données réparties: problèmes et protocoles*. Tome3 : Une introduction aux principes des systèmes répartis Ed. Eyrolles.

[Richard, 01] Richard, N. (2001). *Description de comportements d'agents autonomes évoluant dans des mondes virtuels*, Thèse de Doctorat, Octobre 2001, Ecole Nationale Supérieure des Télécommunications, Spécialité : Informatique et Réseaux.

[Saito, 05] Saito, Y., & Shapiro, M. (2005). Optimistic replication. *ACM Computing Surveys (CSUR)*, *37*(1):42-81.

[Salmon, 98] Salmon, J., Stein, C., & Sterling, T. (1998). Scaling of Beowulf-class distributed systems. In *Proceedings of the 1998 ACM/IEEE conference on Supercomputing*, pp. 1-13.

[Saroiu, 03] Saroiu, S., Gummadi, K. P., & Gribble, S. D. (2003). Measuring and analyzing the characteristics of Napster and Gnutella hosts. *Multimedia systems*, *9*(2):170-184.

[Schneider, 93] Schneider, F. B. (1993). Replication management using the state-machine approach. *Distributed systems*, *Vol. 2*, pp.169-198.

[Senhadji, 13] Senhadji, S., Kateb, A., & Belbachir, H. (2013). A load balancing strategy for replica consistency maintenance in data grid systems. *Informatica*, *37*(3): 345-353.

[Senthilnathan, 12] Senthilnathan, T., & Purusothaman, T. (2012). A multi-agent based data replication mechanism for mobile grid. *American Journal of Applied Sciences*, *9*(4): 542-552.

[Sharma, 10] Sharma, P., Kant, K., & Chauhan, N. (2010). A comparative study of cluster-based data replication techniques for MANETs. *International Journal of Information Technology*, *2*(2):665-667.

[Shorfuzzaman, 10] Shorfuzzaman, M., Graham, P., & Eskicioglu, R. (2010). Adaptive popularity-driven replica placement in hierarchical data grids. *The Journal of Supercomputing*, *51*(3):374-392.

[Skillicorn, 02] Skillicorn, D. B. (2002). Motivating computational grids. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02),* pp. 401-406.

[Smith, 80] Smith, R. G. (1980). The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on computers*, 29(12):1104-1113.

[Souli-Jbali, 15] Souli-Jbali, R., Hidri, M. S., & Ayed, R. B. (2015). Dynamic data replication-driven model in data grids. In *IEEE 39th Annual Conference Computer Software and Applications (COMPSAC),* Vol. 3, pp. 393-397.

[Soyez, 05] Soyez, O. (2005). *Stockage dans les systèmes pair à pair*, Thèse de Doctorat, Mai 2005, Université de Picardie Jules Verne.

[Steen, 10] Steen, M.V., & Pierre, G. (2010). Replicating for performance: case studies. Chapter 5, in *Replication: Theory and Practice*, B. Charron-Bost, F. Pedone, A. Schiper (Eds), LNCS 5959, Springer, pp. 73-89.

[Stoica, 01] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., & Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, *31*(4):149-160.

[Tang, 05] Tang, M., Lee, B. S., Yeo, C. K., & Tang, X. (2005). Dynamic replication algorithms for the multi-tier data grid. *Future Generation Computer Systems*, *21*(5):775-790.

[Tardo, 96] Tardo, J., & Valente, L. (1996). Mobile agent security and Telescript. In *Compcon'96.'Technologies for the Information Superhighway'Digest of Papers*, 25-28 Feb. 1996, pp. 58-63.

[Thomas, 79] Thomas, R. H. (1979). A majority consensus approach to concurrency control for multiple copy databases. *ACM Transactions on Database Systems (TODS)*, *4*(2):180-209.

[Tos, 15] Tos, U., Mokadem, R., Hameurlain, A., Ayav, T., & Bora, S. (2015). Dynamic replication strategies in data grid systems: a survey. *The Journal of Supercomputing*, *71*(11):4116-4140.

[Tranvouez, 01] Tranvouez, E. (2001). *IAD et ordonnancement, une approche coopérative du réordonnancement par systèmes multi-agents*, Thèse de Doctorat, Mai 2001, Aix-Marseille 3.

[Treuil, 97] Treuil, J. P., & Mullon, C. (1997). Expérimentation sur mondes artificiels: pour une réflexion méthodologique. *Editions Scientifiques et Médicales ELSEVIER*.

[Vashisht, 14] Vashisht, P., Kumar, R., & Sharma, A. (2014). Efficient dynamic replication algorithm using agent for data grid. *The Scientific World Journal*, *Volume 2014, ID767016,* pp. 1-10.

[Vazhkudai, 02] Vazhkudai, S., Schopf, J. M., & Foster, I. (2002). Predicting the performance of wide area data transfers. In *Proceedings International Parallel and Distributed Processing Symposium, IPDPS 2002,* pp. 1-10.

[Wolski , 98] Wolski, R. (1998). Dynamically forecasting network performance using the network weather service. *Cluster Computing*, *1*(1):119-132.

[Wooldridge, 94] Wooldridge, M., & Jennings, N. R. (1994). Agent theories, architectures, and languages: a survey. In *International Workshop on Agent Theories, Architectures, and Languages*, pp. 1-39.

[Xiong, 13] Xiong, F., ZHU, X. X., HAN, J. Y., & WANG, R. C. (2013). QoS-aware replica placement for data intensive applications. *The Journal of China Universities of Posts and Telecommunications*, *20*(3):43-47.

[Yagoubi, 07] Yagoubi, B. (2007). Modèle d'équilibrage de charge pour les grilles de calcul. *Revue Africaine de la Recherche en Informatique et Mathématiques Appliquées*, *Vol. 7*, pp.1-19.

[Yu, 05] Yu, H., Martin, P., & Hassanein, H. (2005). Cluster-based replication for large-scale mobile ad-hoc networks. In *International Conference on Wireless Networks, Communications and Mobile Computing,* Vol. 1, pp. 552-557.

[Zhang, 10] Zhang, P., Xie, K., Ma, X., Li, X., & Sun, Y. (2010). A replication strategy based on swarm intelligence in spatial data grid. In *18th International Conference on Geoinformatics,* pp.1-5.

Annexe A.

A.1. Partie entière

Pour $x \in R$ l'unique entier n tel que $x \in [n ; n+1[$ s'appelle la partie entière de x.

1. Notation

a. Partie entière :

Pour tout $x \in R$,
La partie entière (ou partie entière inférieure) de x est notée : $\lfloor x \rfloor = E(x) = \underline{E}(x)$.
La partie entière $\lfloor x \rfloor$ est le plus grand entier inférieur ou égal à x.

b. Partie entière par excès :

Pour tout $x \in R$,
La partie entière par excès (ou partie entière supérieure) de x est notée : $\lceil x \rceil = \overline{E}(x)$.
La partie entière par excès $\lceil x \rceil$ est le plus petit entier supérieur ou égal à x.

2. Propriétés générales

1. Tout $x \in R$, vérifie les propriétés suivantes :
$\lfloor x \rfloor \leq x < \lfloor x \rfloor + 1$
$\lceil x \rceil - 1 < x \leq \lceil x \rceil$
$\lfloor x + 1 \rfloor = \lfloor x \rfloor + 1$

2. Soit $x \in R$, $n \in N$ *et* $m \in N$ tel que :
$n = \lfloor x \rfloor$
$m = \lceil x \rceil$
Alors x, n et m vérifient les propriétés suivantes :
$n \leq x < n + 1$.
$m - 1 < x \leq m$.
$x - 1 < n \leq x$.
$x \leq m < x + 1$.

3. Tout $n \in N$ vérifie les propriétés suivantes :
$n = \left\lfloor \frac{n}{2} \right\rfloor + \left\lceil \frac{n}{2} \right\rceil$.
$n + 1 = \left\lceil \frac{n+1}{2} \right\rceil + \left\lceil \frac{n}{2} \right\rceil$.
$\left\lceil \frac{n+1}{2} \right\rceil = \left\lfloor \frac{n}{2} \right\rfloor + 1$.

A.2. Présentation des cas pour les quorums

Nous allons présenter les quatre cas possibles, où la taille du quorum d'écriture du Quorum Imbriqué n'est pas meilleure que celle du Quorum à Consensus.
Rappelons qu'il faut que le nombre des répliques restantes (qui n'interviennent pas dans le quorum d'écriture) soit inférieur ou égal au nombre de clusters plus 1.

Le reste des répliques qui n'interviennent pas dans le quorum d'écriture sera noté par ***RepR***.

1. Premier cas (nombre de clusters *Nclt* =3)

Nombre de répliques	23	24
Distribution des répliques par clusters	10;10;3	10;10;4
Nombre de clusters intervenant dans le quorum d'écriture	2	2
Nombre de répliques intervenant dans le quorum d'écriture	20=10+10	20=10+10
Nombre de clusters restants.	1	1
Nombre de répliques restantes (*RepR).*	3	4
Taille du quorum d'écriture d'IQ	12	12
Taille du quorum d'écriture de QC	12	13
Résultats	**IQ = QC** *RepR≤Nclt+1*	IQ < QC *RepR≤Nclt+1*

2. Deuxième cas (Nombre de clusters *Nclt* =4)

Nombre de répliques	33	34	35	36
Distribution des répliques par clusters	10;10;10;3	10;10;10;4	10;10;10;5	10;10;10;6
Nombre de clusters intervenant dans le quorum d'écriture	3	3	3	3
Nombre de répliques intervenant dans le quorum d'écriture	30=10+10+10	30=10+10+10	30=10+10+10	30=10+10+10
Nombre de clusters restants.	1	1	1	1
Nombre de répliques restantes (*RepR).*	3	4	5	6
Taille du quorum d'écriture d'IQ	18	18	18	18
Taille du quorum d'écriture de QC	17	18	18	19
Résultats	**IQ > QC** *RepR≤Nclt+1*	**IQ = QC** *RepR≤Nclt+1*	**IQ = QC** *RepR≤Nclt+1*	IQ < QC *RepR>Nclt+1*

3. Troisième cas (nombre de clusters *Nclt* =6)

Nombre de répliques	46	47	48
Distribution des répliques par clusters	10;10;10;10;3;3	10;10;10;10;4;3	10;10;10;10;4;4
Nombre de clusters intervenant dans le quorum d'écriture	4	4	4
Nombre de répliques intervenant dans le quorum d'écriture	40=10+10+10+10	40=10+10+10+10	40=10+10+10+10
Nombre de clusters restants.	2	2	2
Nombre de répliques restantes (*RepR).*	6	7	8
Taille du quorum d'écriture d'IQ	24	24	24
Taille du quorum d'écriture de QC	24	24	25
Résultats	**IQ = QC** *RepR≤Nclt+1*	**IQ = QC** *RepR≤Nclt+1*	IQ < QC *RepR>Nclt+1*

4. Quatrième cas (nombre de clusters *Nclt* =8)

Nombre de répliques	59	60
Distribution des répliques par clusters	10;10;10;10;10;3;3;3	10;10;10;10;10;4;3;3
Nombre de clusters intervenant dans le quorum d'écriture	5	5
Nombre de répliques intervenant dans le quorum d'écriture	40=10+10+10+10+10	40=10+10+10+10+10
Nombre de clusters restants.	3	3
Nombre de répliques restantes (*RepR).*	9	10
Taille du quorum d'écriture d'IQ	30	30
Taille du quorum d'écriture de QC	30	31
Résultats	**IQ = QC** *RepR≤Nclt+1*	IQ < QC *RepR>Nclt+1*

Nous remarquons que lorsque *RepR >Nclt+1,* le Quorum Imbriqué devient meilleure que le Quorum à Consensus.

Pour *Nclt* =5, le Quorum Imbriqué restera meilleure que le Quorum à Consensus, même dans le cas où *RepR*=*Nclt*+1 (3x2=5+1).

Pour *Nclt* = 7, le nombre de clusters restants égal 3 alors *RepR* sera au minimum égal à 9 =3x3=7+2, puisque $rep\_clt_i > 2 \quad \forall \quad 1 \leq i \leq Nclt$.

Pour les autres cas, impossible que le nombre de répliques restantes *RepR* soit inférieur ou égal à *Nclt+1* (*RepR* ≤ *Nclt* + 1). A partir de *Nclt* ≥ 9, *RepR* sera toujours supérieur à *Nclt*+1 (*RepR* ≥ *Nclt* + 1).

ملخص.

في وقتنا الحاضر، يتم استخدام النسخ المتماثل للبيانات على نطاق واسع في النظم ذات النطاق الواسع مثل شبكات البيانات لتلبية مطالب وتحديات إدارة كميات هائلة من البيانات. يتم استخدام النسخ المتماثل للحد من تكلفة الوصول إلى البيانات والحد من الكمون، لتحسين التوافر والأداء، وضمان الموثوقية في الشبكات. ولا يمكن تحقيق هذه الأهداف إلا من خلال استراتيجية جيدة لاستنساخ البيانات، والتي تعالج أيضا مشكلة الموثوقية للنسخ المتماثلة. ومع ذلك، فإن معظم استراتيجيات النسخ المتماثل المقترحة لا تتعامل إلا مع حالات القراءة فقط عند الوصول إلى البيانات، وهناك عدد قليل من النهج التي تحقق في حالات الكتابة في شبكات البيانات.

في هذا السياق، نقترح في هذه الأطروحة استراتيجية ديناميكية للنسخ المتماثل في شبكات البيانات على أساس النصاب، والتي تأخذ في الاعتبار الحالات التي يسمح فيها التحديثات. وبالإضافة إلى ذلك، نظرا لتعقيدات مثل هذه البيئة، فإنه من الصعب جدا اتخاذ قرارات بشأن إدارة النسخ المتماثلة .لمعالجة هذا النقص وكي تكون إدارة النسخ المتماثلة فعالة، يستند صنع القرار في استراتيجيتنا على نظام متعدد الوكلاء.

تبين نتائج تقييم الأداء فعالية اقتراحنا.

الكلمات الرئيسية: شبكة البيانات، النسخ المتماثل للبيانات، موضع البيانات، نظام متعدد الوكلاء، النصاب.

Résumé.

De nos jours, la réplication des données est largement utilisée dans les systèmes à grande échelle tels que les grilles de données pour répondre aux exigences et aux défis de la gestion d'énormes quantités de données. En effet, la réplication est utilisée pour réduire le coût d'accès aux données et les temps de latence, pour améliorer la disponibilité et les performances et pour assurer la fiabilité dans les grilles. Ces objectifs ne peuvent être atteints que par une bonne stratégie de réplication de données, qui adresse aussi le problème de la fiabilité des répliques. Cependant, la plupart des stratégies de réplication proposées ne traitent que les cas de lectures seules lors des accès aux données, et peu d'approches étudient les cas d'accès en écritures dans les grilles de données.

Dans ce contexte, nous proposons dans cette thèse une stratégie de réplication dynamique dans les grilles de données basée sur les quorums, qui prend en considération les cas où les mises à jour sur les répliques sont autorisées. En plus, étant donné la complexité d'un tel environnement, il est très difficile de prendre des décisions sur la gestion des répliques. Pour remédier à cette limite et pour que la gestion des répliques soit efficace, la prise des décisions dans notre stratégie repose sur un système multi-agents.

Les résultats de l'évaluation des performances montrent l'efficacité de notre proposition.

Mots clés : Grille de données, réplication de données, placement de données, système multi-agents, quorum.

Abstract.

Nowadays, data replication is widely used in large-scale systems such as data grids to meet the requirements and challenges of managing huge amounts of data. Indeed, replication is used to reduce the data access cost and latency, to improve availability and performance, and to ensure reliability in grids. These objectives can be achieved only by a good data replication strategy, which also addresses the problem of replica reliability. However, most proposed replication strategies deal read-only case when accessing data, and few approaches studies write access cases in data grids.

In this context, we propose in this thesis a dynamic replication strategy in data grids based on quorums, which takes into account the cases where updates on replicas are allowed. In addition, given the complexity of such environment, it is very difficult to make decisions on replica management. To overcome this limitation and for effectiveness replica management, decision making in our strategy is based on a multi-agent system.

The results of performance evaluation show the effectiveness of our proposition.

Keywords: Data grid, data replication, data placement, multi-agent system, quorum.

Résumé

De nos jours, la réplication des données est largement utilisée dans les systèmes à grande échelle tels que les grilles de données pour répondre aux exigences et aux défis de la gestion d'énormes quantités de données. En effet, la réplication est utilisée pour réduire le coût d'accès aux données et les temps de latence, pour améliorer la disponibilité et les performances et pour assurer la fiabilité dans les grilles. Ces objectifs ne peuvent être atteints que par une bonne stratégie de réplication de données, qui adresse aussi le problème de la fiabilité des répliques. Cependant, la plupart des stratégies de réplication proposées ne traitent que les cas de lectures seules lors des accès aux données, et peu d'approches étudient les cas d'accès en écritures dans les grilles de données. Dans ce contexte, nous proposons dans cette thèse une stratégie de réplication dynamique dans les grilles de données basée sur les quorums, qui prend en considération les cas où les mises à jour sur les répliques sont autorisées. En plus, étant donné la complexité d'un tel environnement, il est très difficile de prendre des décisions sur la gestion des répliques. Pour remédier à cette limite et pour que la gestion des répliques soit efficace, la prise des décisions dans notre stratégie repose sur un système multi-agents. Les résultats de l'évaluation des performances montrent l'efficacité de notre proposition.

Mots clés :
Grille de données; Grille de calcul; Réplication de données; Placement de données; Agent; Système multi-agents; Quorum; Cohérence; Protocole de réplication; Systèmes distribués.