La République Algérienne Démocratique et Populaire

وزارة التعليم العالي و البحث العلمي

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie d'Oran Mohamed Boudiaf

Faculté des Sciences

Département d'informatique

Laboratoire LSSD

Spécialité : INFORMATIQUE

Option : Systèmes, Réseaux et Bases de données (SRBDD)

MEMOIRE

présenté par

Mr. KATEB HACHEMI AMAR AMAR

Pour obtenir Le Diplôme de Magister en INFORMATIQUE

Intitulé

COHERENCE DES DONNEES REPLIQUEES SUR LES GRILLES

Devant la commission d'examen composée de :

Président	*Mr BENYETTOU Mohamed*	*Prof*	*USTO*
Rapporteur	*Mme BELBACHIR Hafida*	*Prof*	*USTO*
Examinateur	*Mr DJEBBAR Bachir*	*Prof*	*USTO*
Examinateur	*Mr RAHAL Sid Ahmed*	*MCA*	*USTO*
Examinateur	*Mr BELALEM Ghalem*	*MCA*	*Univ Es-senia*

Année Universitaire 2012 - 2013

La République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie d'Oran Mohamed Boudiaf

Faculté des Sciences
Département d'informatique
Laboratoire LSSD

Spécialité : INFORMATIQUE
Option : Systèmes, Réseaux et Bases de données (SRBDD)

MEMOIRE

présenté par

Mr. KATEB HACHEMI AMAR AMAR

Pour obtenir Le Diplôme de Magister en INFORMATIQUE

Intitulé

COHERENCE DES DONNEES REPLIQUEES SUR LES GRILLES

SOUTENU LE : 12-02-2013

Devant la commission d'examen composé de :

Président	*Mr BENYETTOU Mohamed*	*Prof*	*USTO*
Rapporteur	*Mme BELBACHIR Hafida*	*Prof*	*USTO*
Examinateur	*Mr DJEBBAR Bachir*	*Prof*	*USTO*
Examinateur	*Mr RAHAL Sid Ahmed*	*MCA*	*USTO*
Examinateur	*Mr BELALEM Ghalem*	*MCA*	*Univ Es-senia*

Année Universitaire 2012 – 2013

Remerciements

Le grand remerciement pour mon Dieu, Qui m'a donné le courage et la patience pour mener à bien ce travail.

Je tiens à remercier toutes les personnes qui ont contribué de manière directe ou indirecte à l'aboutissement de ce travail.

En premier lieu, J'exprime ma gratitude à Madame BELBACHIR Hafida mon encadreur, qui m'a confié ce sujet d'actualité. J'apprécie son enthousiasme, sa gentillesse, ses conseils et ses orientations remarquables.

Je tiens à remercier également tous les membres du jury, Mr BENYETTOU Mohamed en tant que président, Mr DJEBBAR Bachir, Mr RAHAL Sid Ahmed et Mr BELALEM Ghalem comme examinateurs qui ont accepté de juger ce travail.

Merci aux membres du laboratoire de Base de données, à mes amis et à tous les enseignants pour leurs efforts tout au long de mes années d'étude.

Merci à Mlle SENHADJI Sarah, pour son aide précieuse afin de réaliser ce travail et pour les communications acceptées que nous avons effectués ensemble.

Enfin, un grand Merci à tous mes proches qui m'ont apporté leurs soutiens durant l'élaboration de ce travail.

Résumé

La cohérence des données répliquées est un problème largement posé dans le domaine de la gestion des bases de données, plus particulièrement dans les grilles de données.

En effet, dans de tels environnements, la disponibilité de la dernière mise à jour d'une donnée répliquée représente un énorme défi à relever, car les nœuds hébergeant cette donnée répliquée peuvent être volatiles.

Dans ce papier nous nous intéressons au problème de la cohérence des répliques dans les grilles de données. Nous proposons une approche qui combine le protocole à quorum et le modèle de cohérence à l'entrée et ceci dans le but d'assurer une meilleure gestion des propagations des mises à jour.

Mots-clé : cohérence, réplication, grille de données.

Abstract

In distributed systems, the consistency of replicated data is one of the most important problems to solve, especially in data grid environment. In such environments, the availability of the latest value of the replicated data represents a big challenge to relieve, because the node that store this replicated data is dynamic.

In this paper we study the problem of the replicated data in data grid systems. We propose an approach based on the quorum protocol and the entry consistency model, in order to assure the replica update propagation.

Key words: consistency, replication, data grid.

Table des matières

Chapitre 3 : Approche Proposée

Chapitre 4 : Implémentation et mise en œuvre de l'approche

Introduction générale

Durant ces dernières années on a remarqué une augmentation intense du volume des informations traitées, ce qui a exigé le besoin du développement des réseaux à grande échelle et le partage géographique des informations sur le monde entier. En conséquence les chercheurs ont pensé à la conception des grilles qui utilisent la puissance de calcul et l'espace de stockage de plusieurs unités de traitement interconnectées par des réseaux.
Dans une grille, les bases de données sont nombreuses, autonomes (sous fort contrôle local), et très hétérogènes en terme de taille et de complexité. De plus, les postes clients peuvent être des terminaux mobiles qui travaillent en mode déconnecté et se synchronisent de temps à autre avec les bases de données sur le réseau. La gestion des données dans un tel contexte pose des problèmes difficiles, car les techniques doivent passer à l'échelle tout en supportant les nouveaux besoins liés à l'autonomie et l'hétérogénéité des données, et à la mobilité des nœuds de la grille.

Le maintien de la cohérence des données est l'une des taches primordiale lors de la conception et la gestion des bases de données. Dans un système centralisé cette tache est plus au moins facilement accomplie car les données se trouvent dans un endroit précis, et les mises à jours sont effectuées localement sans concurrence ce qui facilite la gestion des données. Mais dans les systèmes répartis à grande échelle tel que les grilles, la gestion de la cohérence devient difficile, car les données sont distribuées sur des sites géographiquement distants, les bases de données sont fragmentées sur ces sites, en plus certaines peuvent être répliquées sur plusieurs sites. Les sites sont interconnectés entre eux formant un réseau d'interconnexion et se communiquent via des messages, quelque soit l'architecture choisie, les mises à jour sont effectuées par des requêtes distantes, et les demandes de lectures / écritures sont concurrentes, donc le temps de propagation des mises à jours, la résistance au pannes et la gestion de concurrence sont pris en considération, il faut trouver donc des bons moyens afin de maintenir la cohérence des données lors de leurs mises à jour.

Dans une grille, la réplication permet d'améliorer la disponibilité des données et les performances d'accès aux données. En général, les données répliquées doivent répondre à deux critères de qualité importants : la fraîcheur et la cohérence des données répliquées par rapport aux données de base. La cohérence est assurée quand à chaque instant toutes les copies d'une donnée sont identiques. Mais garantir la cohérence des répliques sur les grilles est un vrai problème, plusieurs recherches ont été effectuées sur ce domaine, d'après ces recherches on a remarqué que le maintien d'une cohérence forte des répliques coute en terme de performance du système, mais si on veut préserver les performances on doit relâcher la cohérence, pour cela le compromis entre cohérence et performance reste jusqu'à maintenant un défi à surmonter.

Dans ce contexte [(*)], nous proposons une approche qui combine le protocole à quorum et le modèle de cohérence à l'entrée et ceci dans le but d'assurer une meilleure gestion des propagations des mises à jour.

Le présent document est constitué de quatre chapitres et de deux annexes.
Le premier chapitre introduit quelques concepts de bas relatifs à la cohérence et à la réplication des données dans les systèmes distribués.
Le deuxième chapitre présente les travaux liés à notre problème celui de la cohérence des répliques sur les grilles.
Le troisième chapitre expose l'approche proposée pour notre problème.

Le quatrième chapitre présente l'implémentation et la mise en œuvre de l'approche proposée et quelques résultats obtenus.
Une conclusion générale ainsi que des perspectives des travaux futurs clôturent ce document.
Enfin les deux annexes, dont la première concerne des concepts de base sur les grilles et la deuxième présente quelques simulateurs de grilles.

(*) : Ce travail a été réalisé dans le laboratoire LSSD et rentre dans une problématique plus générale qui est la gestion des données dans les grille

Chapitre 1 : Concepts de base sur la réplication et la cohérence dans les systèmes distribués

1. Introduction :

Depuis toujours les chercheurs ont essayé de donner plus de performance à leurs systèmes, afin de diminuer les coûts de communications et augmenter la puissance de calcul. Avec l'apparition des grilles de calcul, les problèmes de gestion des données sont devenus de plus en plus compliqués, surtout en ce qui concerne l'équilibrage de la charge des nœuds, la disponibilité des données partagées, la dynamicité et la volatilité des nœuds du réseau.

L'une des solutions proposées pour gérer ces problèmes est la réplication des données. Ce mécanisme est considéré comme un outil très puissant dans les grilles de calcul pour donner plus de performance aux systèmes et une haute disponibilité des données,
La réplication repose sur le concept de propagation des mises à jour, donc dans un environnement largement distribué comme les grilles, la mise à jour effectuée sur une des copies d'une donnée doit être propagée vers toutes les autres copies, ce qui peut engendrer une dégradation des performances ou une incohérence entre la donnée modifiée et ses copies. Pour cela un modèle de gestion de la cohérence doit être mis en place, et doit être bien adapté à la stratégie de réplication utilisée.
Jusqu'à maintenant le compromis entre cohérence et réplication pose toujours des problèmes surtout dans les systèmes à grande échelle comme les grilles.

Dans ce chapitre on va présenter quelques concepts de base concernant la réplication, et la cohérence des données répliquées.

2. La réplication des données :

La réplication des données est un outil très efficace utilisé dans les systèmes distribués, pour donner plus de disponibilité aux données et en conséquence plus de performance d'accès aux données. La réplication consiste à créer des copies d'une donnée et les stocker dans des endroits différents situés dans le réseau.
Les principaux objectifs de la réplication sont l'amélioration des performances des requêtes sur les données locales et surtout l'augmentation de la disponibilité des données.
Les lectures sont exécutées sur le site disposant de la copie la plus proche du client, ce qui diminue le nombre de transfert de données. Quand à la disponibilité des données, la réplication permet de ne plus dépendre d'un site central unique, et utiliser une copie sur un site lorsqu'elle est indisponible sur un autre [EDD 05], [MAT 07].

Bien que très avantageuse, la réplication présente tout de même un inconvénient majeur de cohérence mutuelle des données (toutes les copies d'une même donnée doivent être identiques). Il peut arriver que les copies d'une donnée ne soient pas identiques. Pour cela des mécanismes doivent être mis en place pour assurer que les copies qui divergent à un moment donné, finissent toutes par converger vers le même état.
De plus, la propagation des mises à jour d'une copie vers toutes les autres copies peut avoir parfois un impact négatif sur les performances du système, celle-ci ne doit pas altérer de façon excessive le temps de réponse total.
Dans ce qui suit on va présenter quelles que caractéristiques concernant la réplication dans les systèmes distribués, tel que la mise à jour des copies, les modes de réplication, la convergence, …etc [MAT].

2.1. Mise à jour des copies :

Le problème majeur de la réplication est posé lors de la mise à jour des copies. Une diffusion des mises à jour est appliquée d'une copie aux autres copies, cette diffusion est assurée par le SGBD réparti. Plusieurs techniques de diffusion sont possibles, il y a celles basées sur la diffusion de l'opération de la mise à jour, et celles basées sur la diffusion du résultat de l'opération.

Diffuser le résultat présente l'avantage de ne pas ré exécuter l'opération sur les sites de la copie, mais l'inconvénient est de nécessiter un ordonnancement identique des mises à jour sur tous les sites afin d'éviter la perte des mises à jour [EDD 05].

La mise à jour synchrone est le mode de distribution dans lequel toutes les sous opérations locales effectuées suite à une mise à jour globale sont accomplies pour le compte de la même transaction. Dans le contexte des copies, ce mode de distribution est très utile lorsque les mises à jour effectuées sur un site doivent être prises en compte immédiatement sur les autres sites.

L'avantage de ce type de mise à jour (synchrone) est de garder toutes les données au dernier niveau de la mise à jour. Le système peut alors garantir la fourniture de la dernière version des données quelque soit la copie accédée. L'inconvénient majeur est la nécessité de gérer des transactions multi-sites coûteuses en ressources.

La mise à jour asynchrone représente le mode de distribution dans lequel certaines sous opérations locales effectuées suite à une mise à jour globale sont accomplies dans des transactions indépendantes, en temps différé.
Le temps des mises à jour des copies peut être plus ou moins différé, dans le cas où les transactions de report peuvent être lancées dés que possible ou à des instants fixés.

L'avantage de ce type de mise à jour est la possibilité de mettre à jour en temps choisi des données, tout en autorisant l'accès aux versions anciennes avant la mise à jour.
L'inconvénient est que l'accès à la dernière version n'est pas garanti [MAT 07].

2.2. Modes de Réplication:

Lors du traitement des données répliquées sur un système distribué, en fonction de la propagation des mises à jour et du traitement des copies des données, on distingue en générale deux modes différents de réplication.
Le premier est définit par la symétrie des comportements des copies d'une donnée répliquée. Chaque copie est traitée d'une façon identique aux autres, c'est le cas de la réplication symétrique. Le deuxième mode distingue deux comportements d'une donnée répliquée ; la copie primaire et les copies secondaires. La copie primaire est la seule à effectuer tous les traitements (lecture/écriture). Les copies secondaires, surveillent la copie primaire. En cas de défaillance de la copie primaire, une copie secondaire devient la nouvelle copie primaire, c'est le cas de la réplication asymétrique. [EDD 05].

2.2.1. Réplication asymétrique :

La réplication asymétrique distingue un site maître chargé de centraliser les mises à jour, il est appelé aussi site primaire, il est le seul autorisé à mettre à jour les données et est chargé de diffuser les mises à jour aux autres copies dites secondaires (esclaves). Dans ce mode de réplication, le site primaire effectue les contrôles et garantit l'ordonnancement correct des mises à jour.

Le problème de la gestion de copie asymétrique est la panne du site maître. Dans ce cas, il faut choisir un remplaçant si l'on veut continuer les mises à jour. On aboutit alors à une technique asymétrique mobile dans laquelle le site maître change dynamiquement. Il devient nécessaire de gérer à la fois les problèmes de pannes qui provoquent des échecs de transactions et l'évolution des travaux [EDD 05]. Il existe la réplication asymétrique synchrone et asynchrone :

La réplication asymétrique synchrone utilise un site maître qui pousse les mises à jour en temps réel vers un ou plusieurs sites esclaves (Figure 1.1).

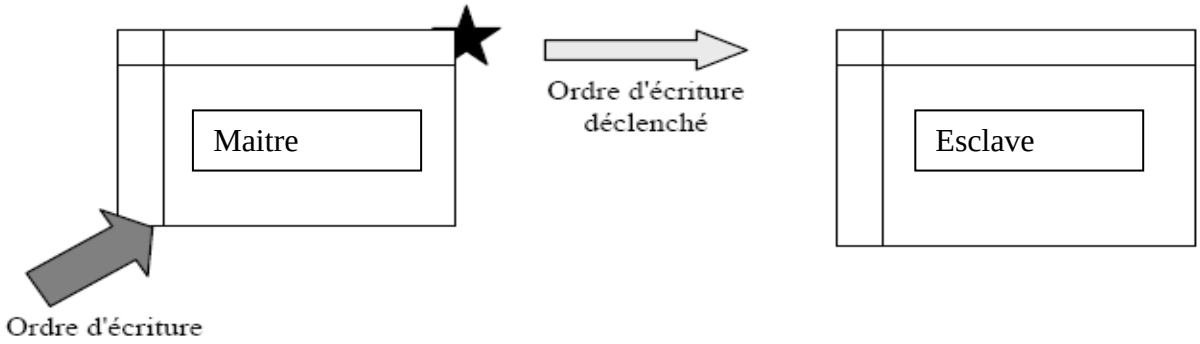


Figure 1.1. Réplication asymétrique synchrone

Dans le cas de la réplication asymétrique asynchrone les mises à jour sont poussées en temps différé via une file d'attente. Les mises à jour seront exécutées ultérieurement, à partir d'un déclencheur externe, une horloge par exemple (Figure 1.2).

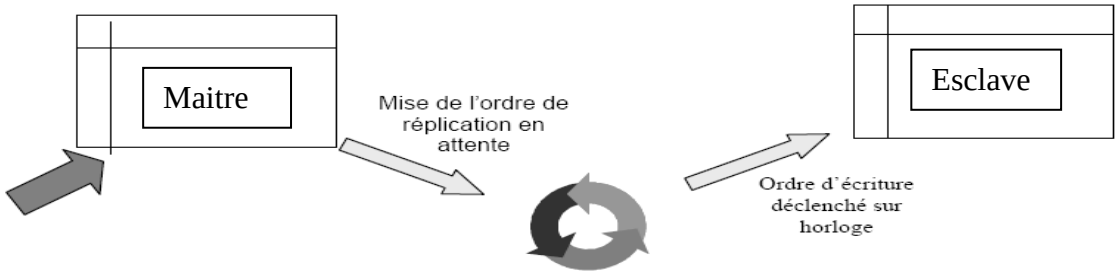


Figure 1.2. Réplication asymétrique asynchrone

2.2.2. Réplication symétrique :

A l'opposé de la réplication asymétrique, la réplication symétrique ne privilégie aucune copie, elle permet des mises à jour simultanées de toutes les copies par des transactions différentes. Dans ce cas chaque copie peut être mise à jour à tout instant et assure la diffusion des mises à jour aux autres copies.
Ce mode pose le problème de concurrence d'accès, en risquant de faire diverger les copies. Un algorithme global de résolution des conflits doit donc être mis en œuvre [EDD 05].

Pour la réplication symétrique aussi on deux modes la réplication symétrique synchrone et asynchrone.

La réplication symétrique synchrone gère des accès concurrentiels aux écritures, toutes les copies sont accessibles à la fois en lecture et en écriture.
Chaque copie possède ses propres triggers qui lors de la réception d'une requête de mise à jour déclenchent le déploiement de cette dernière vers toutes les autres copies. A ce moment entre en jeu le protocole de validation atomique qui doit s'assurer que la transaction est bien validée sur toutes les copies ou sur aucune. [EDD 05] (Figure 1.3).

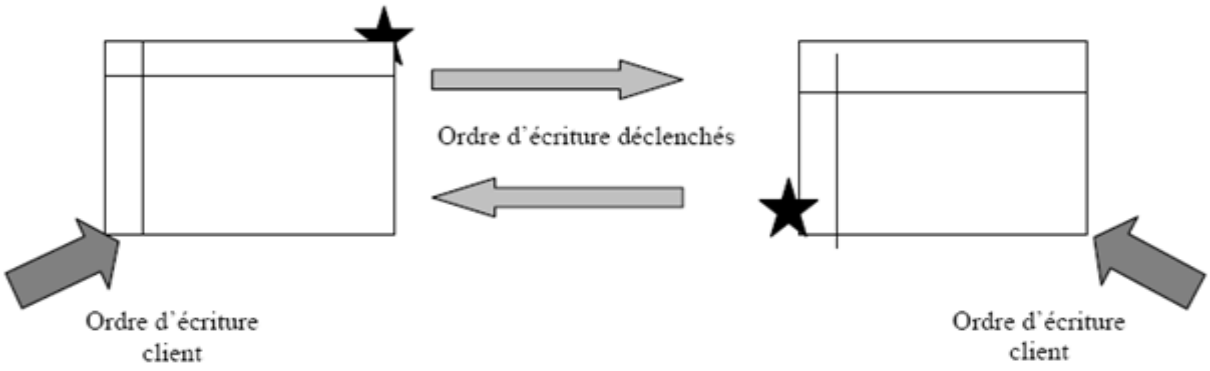


Figure 1.3. Réplication symétrique synchrone

Le principe de la réplication symétrique asynchrone est basé sur trois éléments :
Écritures sur le maître : les écritures initiées par les utilisateurs sont orientées vers un site unique (le site maître), les esclaves n'acceptent que les requêtes de lecture.
Mise en attente : chaque écriture effectuée sur le site maître est sauvegardée dans une file locale ou dans un journal. C'est un processus extérieur au SGBD qui scrute en permanence la file (ou le journal) afin de détecter les modifications apportées au site maître. Ce dernier se charge par la suite de propager ces changements vers les copies secondaires (esclaves) via un déclencheur tel que l'horloge.
Propagation des changements sur l'esclave : les seules écritures admises par les esclaves sont celles effectués par les utilisateurs sur le maître.

Une panne qui intervient sur le site maître avant qu'il n'ait eu le temps d'enregistrer la requête dans une file d'attente ou un journal, entraîne la perte de la mise à jour. (Figure 1.4).

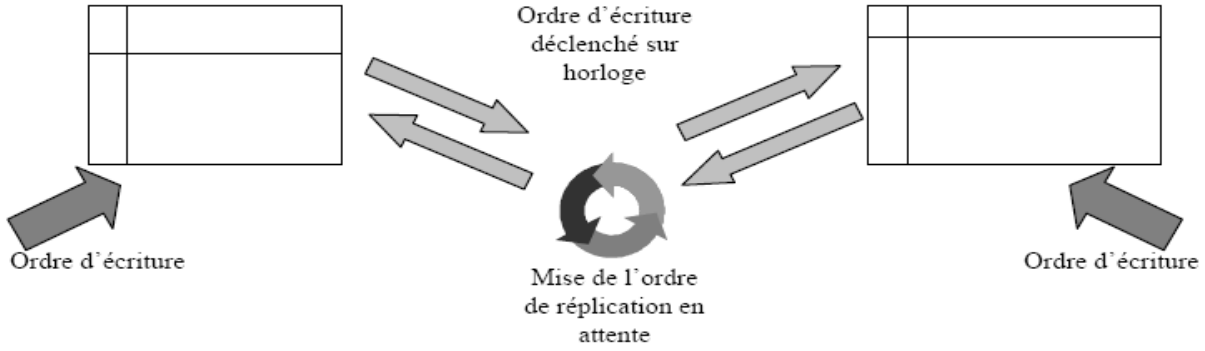


Figure 1.4. Réplication symétrique asynchrone

2.3. Convergence des copies :

La convergence des copies asymétriques est assurée par le site maître. Ce dernier règle les problèmes de concurrence et envoie les mises à jour dans l'ordre où il les applique aux sites secondaires. Encore il faut que ces derniers exécutent les mises à jour dans le bon ordre, ce qui présente quelques difficultés [RIM 07].
Dans le cas des copies symétriques, la convergence est plus difficile à assurer, à cause des mises à jour simultanées qui risquent d'être accomplies dans un ordre différent sur une même donnée. On obtient alors des états différents, ce qui signifie que les copies se divergent, cela est due par le non respect de la sérialisabilité de l'exécution globale des transactions. Ce problème peut être résolu par des mécanismes de contrôle de concurrence, comme par exemple [EDD 05] :

- ***La prévention des conflits par verrouillage des copies :*** Ce mécanisme consiste à demander à toute transaction mettant à jour, d'obtenir un verrou en écriture sur chacune des copies. Les risques de verrous mortels sont alors importants si plusieurs transactions mettent à jour simultanément plusieurs copies. Ces risques peuvent être prévenus en ordonnant l'ordre de visite des sites.

- ***La détection des conflits par ordonnancement des mises à jour :*** L'ordonnancement s'effectue par un mécanisme d'estampillage ou de certification. Le système doit alors utiliser des estampilles synchronisées pour ne pas favoriser un site plutôt qu'un autre. Ici les risques de transactions multiples sont importants.

En asymétrique il est impossible de défaire des transactions validées. Les différentes approches se contentent alors de détecter les conflits et de les reporter vers l'utilisateur qui doit mettre on œuvre une technique de résolution de conflits. On peut utiliser le mécanisme des estampilles décrit précédemment ou le mécanisme de vérification des valeurs avant mise à jour, cela consiste à utiliser la valeur avant mise à jour de l'objet afin de vérifier que la valeur de la copie correspond bien à celle sur laquelle est basée la mise à jour. Chaque message de mise à jour doit donc comporter la valeur avant mise à jour et la valeur après. Dans le cas où la valeur avant est différente de celle figurant dans la base, un conflit est détecté, ce qui veut dire qu'une autre transaction a effectué une mise à jour concurrente [EDD 05].

Après détection des conflits, il faut les résoudre, il s'agit de réconcilier les copies qui ont divergé. La réconciliation peut être purement syntaxique et ne pas prendre en compte la signification des données. Comme on peut aussi utiliser une solution asymétrique en choisissant un site prioritaire. La réconciliation peut faire appel à la signification des données et des mises à jour, c'est le cas d'une procédure de réconciliation sémantique [EDD 05].

2.4. Quelques approches de réplication sur les grilles:

Les approches de réplication se devisent en deux grandes familles : pessimistes et optimistes.

2.4.1. Les approches pessimistes :

Les algorithmes pessimistes interdisent l'accès à une réplique à moins qu'elle soit mise à jours. Les approches pessimistes garantissent la cohérence en prévenant les risques de conflit. L'avantage de ces approches est d'éviter les problèmes liés à la réconciliation [PAS 05].

Ces techniques sont bien adaptées dans les réseaux locaux mais pour les réseaux à grande échelle telle que Internet, les performances d'exécution et de disponibilité ne sont pas assurées pour les raisons suivantes [PAS 05]:

- Les réseaux de large secteur continuent à être longs et incertains. Il y a beaucoup de dispositifs de calcul mobiles et dispositifs de pare-feu avec seulement une connectivité irrégulière d'Internet. Les algorithmes pessimistes fonctionnent mal dans de tels environnements. Par exemple : l'algorithme « primary copy » exige que l'échec d'un emplacement soit exactement détecter de sorte qu'il puisse sûrement réélire un autre primaire quand ce dernier échoue, ce qui ne peut être effectué sur un environnement internet.
- Un autre problème concernant la disponibilité : si on augmente le nombre de répliques le temps de lecture diminue mais le temps d'écriture augmente. Il est donc difficile de supporter les services qui déploient beaucoup de réplique dont les mises à jour sont fréquentes. Internet et beaucoup de services mobiles représentent cette catégorie.
- Quelques activités humaines exigent le partage optimiste de données. Par exemple, dans la technologie coopérative ou le développement de programme, les utilisateurs travaillent concurremment et dans l'isolement relatif les uns des autres. Il vaut mieux permettre les mises à jour concurrentes et réparer les conflits occasionnels après qu'ils se produisent que de verrouiller les données pendant la mise à jour.

Quelques exemples des approches pessimistes [PAS 05] sont donnés dans ce qui suit.

- **RAWA** (Read Any, Write All) : c'est un protocole basique, il consiste à obtenir un verrou exclusif sur toutes les copies avant d'effectuer une écriture sur une des copies.

- **ROWA** (Read Once Write All) : la disponibilité des lectures est améliorée avec ce protocole. Les lectures verrouillent et accèdent à une seule copie, tandis que les écritures continuent à verrouiller et modifier toutes les copies. Toutes les copies sont mises à jour de manière synchrone ce qui implique que toutes les copies sont à jour. Mais si un site tombe en pane, toutes les écritures sont bloquées jusqu'à ce que la panne soit réparée.

- **ROWAA** (Read Once Write All Available) : ce protocole est adapté aux cas de pannes en ne verrouillant que les copies disponibles. Quand une copie reprend sa disponibilité, elle doit d'abord se resynchroniser pour effectuer les mises à jour manquantes. Toutes les copies disponibles sont mises à jour de manière synchrone et les copies indisponibles sont mises à jour de manière asynchrone ce qui implique que toutes les copies disponibles sont à jour et il existe une possibilité de copies non à jour. ROWAA tolère (n-1) pannes et ne supporte pas les partitions réseaux et les pannes de communication.

- **Primary Copy ROWA :** Une copie est désignée comme copie primaire, les autres comme esclaves. Si la copie primaire tombe en panne, une autre est désignée comme nouvelle primaire. On trouve deux variantes : Primaire/Secondaire : Les mises à jour sont faites sur la copie primaire, les copies secondaires sont mises à jour de manière asynchrone ce qui implique que la copie primaire est toujours à jour.
  Primaire/Sauvegarde : Les mises à jour sont faites sur la copie primaire, et une des copies secondaire est désignée comme étant la sauvegarde, celle-ci est responsable de la reprise sur défaillance de la copie primaire. Toutes les autres copies secondaires

sont mises à jour de manière asynchrone ce qui implique que, lorsqu'elle est disponible, la copie primaire est toujours à jour.

- **Indépendant :** Dans cette approche, les mises à jour sont faites sur des copies quelconques. Mais ceci peut engendrer la possibilité d'avoir des copies incompatibles ou incohérentes, pour cela un algorithme de réconciliation est mis en place. Cette approche ne garantie pas que toutes les copies sont à jour.

2.4.2. Les approches optimistes :

Les approches optimistes permettent aux utilisateurs d'accéder à n'importe quelle réplique à tout moment, en supposant que les conflits de mise à jour sont rares. L'idée générale est donc de laisser l'utilisateur lire ou écrire n'importe qu'elle réplique, la mise à jour se propage et les conflits sont résolus après qu'ils se produisent. Ces approches offrent plusieurs avantages par rapport aux approches pessimistes [STE 03] :
Disponibilité : elles fonctionnent bien dans les réseaux lents et incertains, parce qu'ils peuvent propager des mises à jour sans bloquer les accès à toutes les répliques.
Flexibilité de gestion du réseau : elles fonctionnent également bien dans les réseaux dynamiques. Une telle propriété est essentielle dans les environnements à large échelle telle que les grilles de calcul [Annexe 1], dans lesquels les dispositifs peuvent avoir un comportement dynamique.
Scalabilité : les approches optimistes peuvent soutenir un plus grand nombre de répliques, parce que la propagation de mise à jour exige moins de coordination parmi les sites.

Quelques exemples des approches optimistes sont donnés dans ce qui suit [IMI 06].

- **Bayou :** dans cette approche une copie de l'objet partagé est située sur chaque serveur dans le système. Les applications interagissent avec cette copie via une interface de programmation Bayou. Les utilisateurs peuvent donc effectuer des opérations (Lectures/Ecritures) sur cette donnée. Une fois qu'un serveur reçoit une opération, il tente de l'exécuter, pour assurer la convergence des copies. Les serveurs doivent exécuter les opérations dans le même ordre. Cet ordre est décidé par un serveur principal désigné au lancement du système.
  Un journal des opérations exécutées est maintenu par chaque serveur. Ce journal se compose de deux parties : un préfixe qui contient les opérations validées par le serveur principal, qui sont ordonnées définitivement selon un ordre total introduit lors de la validation, la deuxième partie du journal (appelé provisoire) contient le reste des opérations, qui sont ordonnées selon un ordre total définit au fur et à mesure de la réception de nouvelles opérations.
  L'inconvénient de cette approche est le passage à l'échelle, car la mise en place d'un préfixe commun nécessite un ordre total, ce qui limite le passage à l'échelle [IMI 06].

- **IceCube :** cette approche représente un système de réconciliation, dont le but est d'exécuter une combinaison optimale de mises à jour concurrentes.
  Ice cube déduit un journal optimal à partir des journaux des différents sites, contenant un nombre maximum de mises à jour qui ne présentent pas de conflits. Dans cette approche les mises à jour sont modélisées par des actions, une action est composée par :
  - Cibles : identifie les objets modifiés par l'action.
  - Pré-condition : permet de détecter les conflits.

- Opération : permet d'accéder aux objets partagés.
- Données privées : données propres à l'action, comme les paramètres et le type de l'opération.

La réconciliation est effectuée en trois phases :

- Phase d'ordonnancement : avec la combinaison des différentes actions issues des journaux des différents sites, on construit toutes les exécutions possibles.
- Phase de simulation : par l'exécution des différents ordonnancements trouvés dans la phase précédente on vérifie et on ne retient que ceux qui répondent aux contraintes de simulation.
- Phase de sélection : parmi les ordonnancements retenus, on doit choisir un seul, le critère de choix est définit par l'administrateur.

Cette approche assure la convergence des copies par l'imposition d'un ordre total sur l'exécution des opérations [IMI 06].

2.5. Les Protocoles de Réplication :

Les principaux protocoles de réplication utilisés sur des systèmes de communication de groupe sont la réplication passive, la réplication active, la réplication semi-active et la réplication coordinateur cohorte [STE 03], [FOS 01].

2.5.1. Réplication Passive :

Dans ce protocole, une seule copie, appelée copie primaire, reçoit la requête d'un client et l'exécute. Les autre copies, nommées copies secondaires ou copies de sauvegardes, ne sont là que pour prendre le relais en cas de défaillance de copie primaire. Afin d'assurer la cohérence des copies secondaires, la copie primaire leur diffuse régulièrement son nouvel état (un point de reprise). Cette méthode assure que toutes les copies ont la même valeur même si les traitements sont non déterministes car il y a diffusion de l'état de la copie primaire vers les copies secondaires et non exécution des requêtes (Figure 1.5).

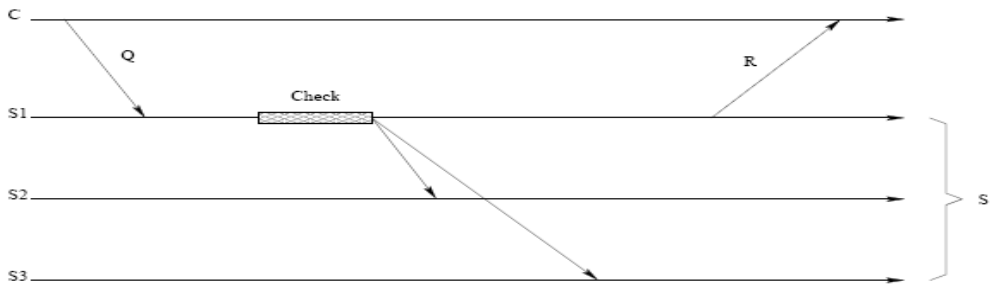


Figure 1.5. Exemple de réplication Passive

2.5.2. Réplication Active :

Dans ce protocole, toutes les copies jouent le même rôle. Elles reçoivent toutes la même séquence totalement ordonnée de requêtes de la part des clients, les exécutent de manière déterministe et renvoient la même séquence totalement ordonnée des réponses (Figure 1.6). La condition pour que toutes les copies reçoivent et exécutent toutes les requêtes dans le même ordre peut être décomposée en deux conditions distinctes :

1. Toutes les copies reçoivent les mêmes requêtes.
2. Toutes les copies exécutent les requêtes dans le même ordre relatif.

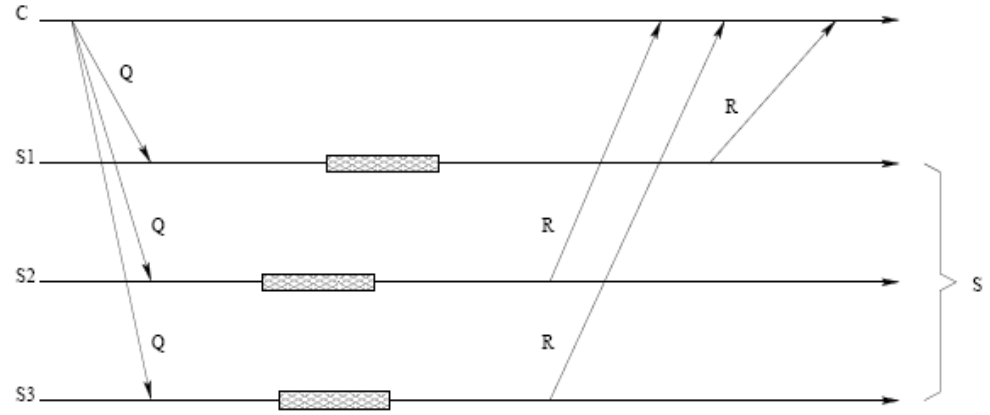


Figure 1.6. Exemple de réplication Active

2.5.3. Réplication Semi-active :

Ce protocole est un protocole de réplication active dans le sens où chaque copie exécute la requête. Par contre, toutes les sources d'indéterminisme sont résolues par le choix d'une copie primaire qui diffuse aux autres copies ses choix. De plus, la copie primaire est la seule à renvoyer les résultats aux clients. La copie primaire est appelée leader et les copies secondaires sont appelées suiveurs. Le leader traite une requête dés qu'il la reçoit. Par contre, un suiveur doit attendre une notification du leader pour pouvoir traiter une requête. Lorsqu'il y a des sources d'indéterminisme, le leader envoie ses choix aux suiveurs (Figure 1.7).

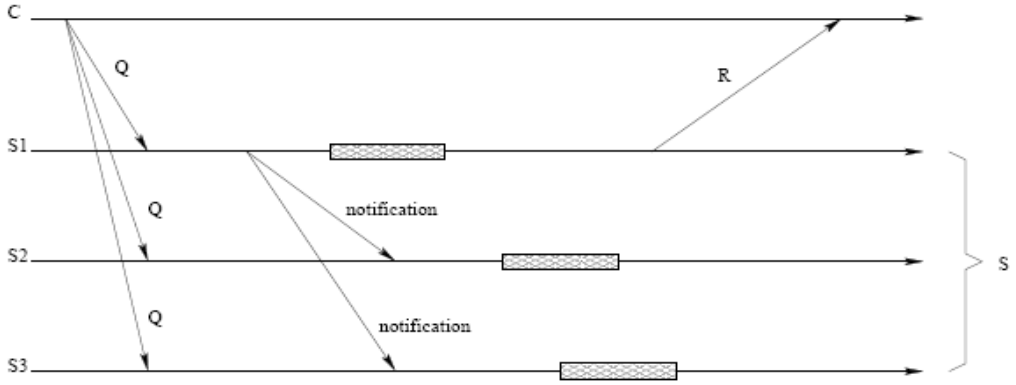


Figure 1.7. Exemple de réplication Semi-active

2.5.4. Réplication Coordinateur/cohortes :

Ce protocole est une hybridation entre la réplication active et la réplication passive. Il distingue une copie primaire appelée coordinateur et des copies secondaires appelées cohortes. Toutes les copies reçoivent la requête, mais le coordinateur est le seul à la traiter. Le coordinateur envoie des points de reprise systématiques aux cohortes (Figure 1.8).

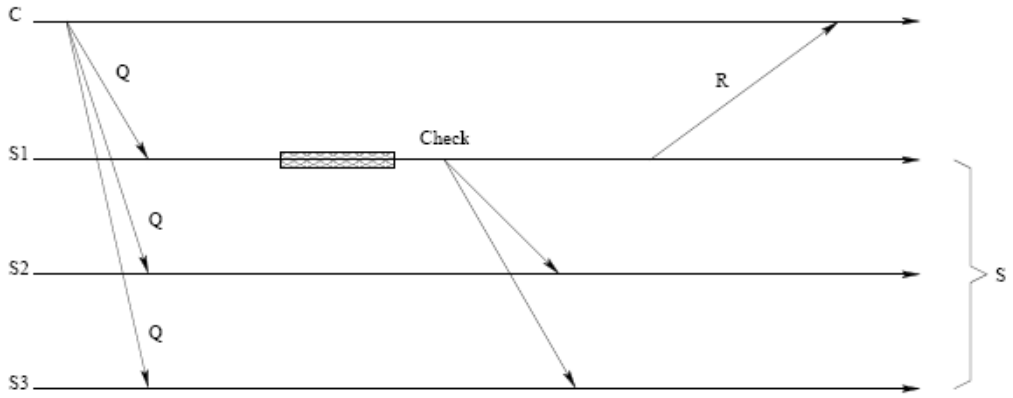


Figure 1.8. Exemple de réplication Coordinateur/cohortes

3. La cohérence :

La cohérence est une propriété qui assure qu'une transaction fait passer une base de données d'un état cohérent à un autre état cohérent grâce au respect des contraintes d'intégrité.

Dans les systèmes distribués la cohérence des répliques est d'assurer deux propriétés fondamentales, la première est qu'à n'importe quel moment toutes les copies d'une certaine donnée sont identiques, la deuxième est que n'importe quelle opération de lecture sur une donnée (ou une copie de donnée) doit retourner la dernière valeur écrite.
Nous présentons dans ce qui suit les concepts de base de la cohérence que nous avons obtenu à partir des documents [PRO 05], [LOI 05], [STE 00] et [SEB 06].

3.1. Concept de transactions :

L'utilisateur manipule la base de données soit à travers le langage de requête, soit à travers des programmes qui font appel au SGBD. L'exécution d'une requête ou d'un programme fait naître au niveau du SGBD une occurrence de transaction.

Une transaction est une séquence d'instructions, modifiant des données (incluant lectures, écritures dans des bases de données différentes) et devant être effectuées toutes en même temps ou pas du tout afin de garantir la cohérence des bases de données (Figure 1.9).
Une transaction peut être simplement l'exécution d'une requête SQL, ou bien celle d'un programme en langage hôte dans lequel on a des appels au langage de requête [PRO 05].

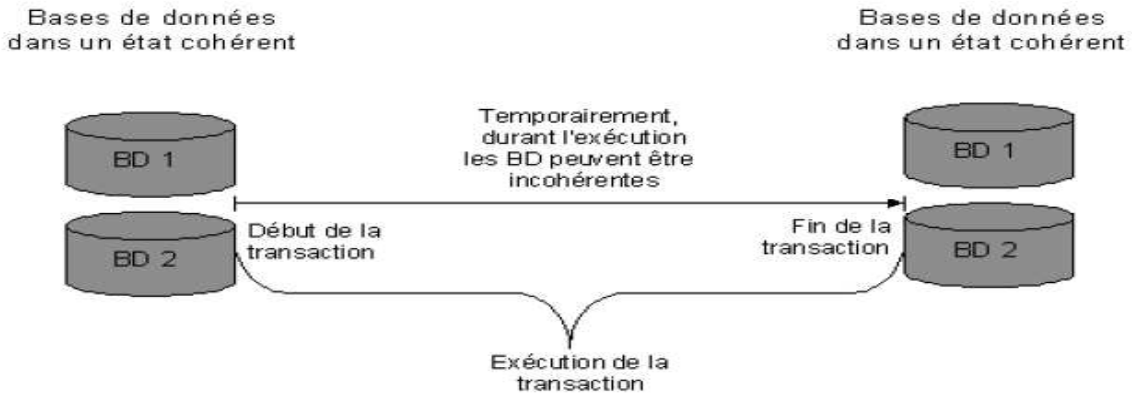


Figure 1.9. Transaction

Dans une base de données, une contrainte d'intégrité d'unicité permet de s'assurer de l'unicité d'un enregistrement pour une table afin de préserver la cohérence de cette base. On distingue deux types de contraintes : les contraintes d'intégrité statiques sont celles qui correspondent à un prédicat sur l'état courant de la base, et les contraintes d'intégrité dynamiques sont celles qui concernent le passage d'un état à un autre.

3.2. Propriétés de transaction :

Le mécanisme de gestion des transactions doit assurer certaines propriétés afin de préserver la cohérence des bases de données lors l'exécution des différentes transactions sur ces dernières. [PRO 05].

- ***Atomicité d'une transaction :*** La gestion de transaction nécessite la propriété d'atomicité d'une transaction c'est à dire d'assurer que les opérations mêmes les plus complexes englobées au sein d'une transaction ne soient perçues que comme une opération unique. De plus il faut que soit toutes les modifications liées à cette opération soient effectuées, soit aucune ne l'est. C'est le système qui est à même d'assurer l'atomicité des transactions. Comme une transaction peut ne pas se terminer normalement (à cause d'une panne par exemple) on définit donc les points : avant et après validation.
  Une transaction ayant atteint son point de validation sera dite validée (elle ne peut plus être remise en cause). Une transaction n'ayant pas encore atteint son point de validation sera dite vivante [PRO 05].

- ***Permanence*** : Le problème de la permanence est de faire en sorte que lorsqu'une transaction a atteint son point de validation, les effets de la transaction soient conservés sur la base quelles que soient les circonstances. La solution est obtenue par l'intermédiaire de fichiers journaux qui conservent la trace des transactions successives qui ont été effectuées sur la base.
  Lors d'une transaction qui effectue une mise à jour sur la base, la base passe d'un ancien état à un nouvel état et on conserve dans le journal, l'identification de la transaction, l'identification des éléments modifiés, leur ancienne valeur et leur nouvelle valeur (Figure 1.10).

Figure 1.10. Réalisation d'une transaction

Lors d'une panne si la transaction doit être défaite on se sert du nouvel état et de l'ancienne valeur se trouvant dans le journal pour reconstituer l'ancien état.

Figure 1.11. Défaire d'une transaction

Si la transaction doit être refaite, il faut partir de l'ancien état et du journal pour reconstruire le nouvel état [PRO 05].

Figure 1.12. Refaire une transaction

- ***Isolation :*** La base de données est découpée en éléments appelés granules (c'est en général l'administrateur en accord avec le concepteur qui fixe le niveau de granularité (n-uplet, page, table). L'isolation d'un élément dans une transaction est obtenue par le mécanisme des verrous (lock), Ceci peut engendrer deux grands types de problèmes :
  Le premier est celui de la famine. Lorsqu'un verrou est relâché sur un élément A, le système choisit parmi les transactions candidates en attente. Si les transactions sont liées au niveau de priorité de l'utilisateur certaines d'entre elles risquent d'attendre longtemps.
  Le deuxième type de problème concerne l'inter blocage. Cette situation se présente lorsqu'un ensemble de transactions attendent mutuellement le déverrouillage d'éléments actuellement verrouillés par des transactions de cet ensemble.
  Le mécanisme de verrouillage permet donc d'isoler un élément pour une transaction, cependant il faut encore que cette utilisation soit faite à bon escient [PRO 05].

- ***Sérialisabilité :*** Une exécution d'un ensemble de transactions est sérialisable si et seulement si elle est équivalente à une exécution séquentielle (ou série) de transactions. Quand les transactions sont arbitraires, la sérialisabilité est la seule propriété qui assure le bon entrelacement.
  La sérialisabilité peut être obtenue en imposant aux transactions que tous les verrouillages précèdent tous les déverrouillages. Les transactions sont alors dites à deux phases : une phase d'acquisition des verrous puis une phase de libération [PRO 05].

3.3. Les modèles de cohérence des données répliquées dans les systèmes répartis:

Les systèmes répartis donnent l'illusion au programmeur que plusieurs mémoires physiquement distribuées n'en forment qu'une.
Dans ce contexte, les mécanismes d'adressage classiques de la mémoire sont conservés et il est du ressort des systèmes répartis d'aller chercher les données, qu'elles soient situées localement sur le site ou distantes. Il n'est alors pas nécessaire d'utiliser des primitives de lecture et d'écriture spécialisées car la mémoire détecte les accès distants et se charge d'effectuer les transferts de données. Les échanges de messages induits par la gestion de la mémoire distribuée entre les différents sites sont masqués par le système.

La répartition des données entre plusieurs sites présente le problème d'ordonnancement des opérations de lecture et d'écriture, c'est à dire la vision que chaque site peut avoir de l'entrelacement des instructions. Ceci est d'autant plus difficile à appréhender que les données puissent être répliquées sur plusieurs sites. La mise en place d'une horloge globale

synchronisée sur tous les sites n'est pas réalisable et la technique de l'horodatage ne permet alors plus d'établir une trace d'exécution unique.

La notion centrale dans un système réparti est le modèle de cohérence utilisé. Un modèle de cohérence s'affiche comme un contrat passé entre le système et le programmeur. Il définit les critères déterminant la valeur retournée par une lecture en fonction des écritures précédentes. Il existe plusieurs modèles de cohérence appartenant aux classes de cohérence sans synchronisation et avec synchronisation [LOI 05].

3.3.1. Les modèles de cohérence sans synchronisation :

Les modèles de cohérence sans synchronisation sont caractérisés par des contraintes fortes entre la dernière écriture et la prochaine lecture. Ils utilisent les seules primitives de lecture et d'écriture. Parmi ces modèles on peut citer [LOI 05], [STE 00] :

- ***Cohérence atomique :*** C'est un modèle idéal où chaque lecture retourne la dernière valeur écrite dans la donnée. Dans les systèmes distribués, le protocole associé nécessite l'utilisation d'une horloge globale, ce qui rend son implémentation difficile. Il garantit un comportement similaire à celui d'un système centralisé.

- ***Cohérence séquentielle :*** modèle formalisé par LAMPORT en 1979 [LAM 79]. Un modèle de cohérence est séquentiel si le résultat de toutes les exécutions est le même que celui où les opérations de tous les processus seraient exécutées dans un ordre séquentiel donné, et dans cette séquence. Les opérations de chaque processus individuel apparaissent dans l'ordre spécifique par leur programme. Il garantit que tous les processus percevront les opérations dans le même ordre, mais il ne garantit pas qu'une lecture obtienne la dernière valeur affectée par une écriture.

- ***Cohérence causale :*** Un modèle de cohérence est causal s'il garantit les deux conditions suivantes [HUT 90] :
  1. Les opérations d'écriture causalement liées doivent être perçues par tous les processus dans le même ordre.
  2. Les opérations d'écriture non causalement liées peuvent être perçues dans des ordres différents, sur des processus différents.

  Ce modèle permet un relâchement de la cohérence séquentielle en fonction de la causalité entre les opérations, et la possibilité que tous les processus perçoivent les opérations d'écriture causalement liées dans le même ordre. Par contre, aucune contrainte sur les opérations d'écriture concurrentes.

- ***Cohérence PRAM :*** Un modèle de cohérence est PRAM s'il garantit que les opérations d'écriture effectuées par un même processus sont perçues par tous les processus dans l'ordre où ces opérations ont été effectuées. Les opérations d'écriture effectuées par différents processus peuvent être perçues par chaque processus dans un ordre différent. Il permet un relâchement du modèle de cohérence causale : certaines transitivités dues à la relation de causalité ne sont pas considérées. C'est un modèle implantable sous forme de files d'opérations d'écriture

- ***Cohérence à la longue :*** Un modèle de cohérence est à la longue s'il garantit que les opérations d'écriture seront propagées ultérieurement. C'est le modèle de cohérence le plus faible. Ce modèle n'impose aucun ordre sur les opérations

3.3.2. Les modèles de cohérence avec synchronisation :

Les modèles de cohérence avec synchronisation utilisent, en plus des primitives de lecture et d'écriture, des primitives manipulant des variables de synchronisation.

Avec les modèles de cohérence sans synchronisation, les opérations pour synchroniser les processus se font par des mécanismes indépendants. Or, en intégrant ces synchronisations avec les modèles de cohérence, il est possible de relâcher des contraintes, et donc d'obtenir un protocole de cohérence moins coûteux et plus performant. Par exemple, il est fréquent de protéger des variables par une exclusion mutuelle entre les processus. Le processus possédant la section critique est donc le seul à lire et à modifier ces variables. Il est donc judicieux d'alléger la cohérence, puisque ces variables ne sont plus partagées à ce moment.

Les modèles de cohérence avec synchronisation sont aussi appelés modèles de cohérence faibles car faisant intervenir le programmeur dans la gestion de la cohérence [LOI 05], [STE 00].
Parmi ces modèles on a :

- ***Cohérence faible :*** Un modèle de cohérence est faible s'il garantit les trois propriétés suivantes :
    1. Les opérations sur les variables de synchronisation se font selon un modèle de cohérence séquentielle ;
    2. L'accès aux variables de synchronisation ne peut se terminer que si toutes les opérations d'écriture et de lecture sont terminées sur tous les sites ;
    3. Les opérations de lecture et d'écriture ne peuvent se faire que si toutes les opérations sur les variables de synchronisation sont terminées sur tous les sites.

  Ce modèle garantie pour un processus : que toutes les modifications effectuées sont reçues par tous les processus et qu'il a reçu toutes les modifications faites par tous les processus, et il permet le regroupement des propagations

- ***Cohérence au relâchement :*** Un modèle de cohérence est au relâchement s'il garantit les trois propriétés suivantes :
    1. Les opérations de lecture et d'écriture ne peuvent se faire que si toutes les opérations d'acquisition précédentes sont terminées sur tous les sites.
    2. L'opération de relâchement ne peut se terminer que si toutes les opérations d'écriture et de lecture sont terminées sur tous les sites.

  Quand un processus effectue une opération de relâchement, les modifications qu'il a effectuées seront observées par tous les processus faisant une acquisition ultérieure. Il existe deux protocoles mettant en œuvre ce modèle. Le premier concerne la cohérence au relâchement impatiente où toutes les modifications entre les deux opérations de synchronisation sont propagées au moment du relâchement à tous les processus, même ceux qui n'effectueront pas d'acquisition ultérieure. Le deuxième concerne la Cohérence au relâchement paresseuse où toutes les modifications entre les deux opérations de synchronisation sont propagées au moment d'une acquisition par un autre processus [CRI 96]. Le surcoût dû à ce protocole est compensé par le nombre et la taille des messages échangés.

- ***Cohérence à l'entrée :*** Basée sur l'association à chaque variable partagée un objet de synchronisation comme le verrou [BRI 93]. Cette stratégie permet de ne transférer que

des mises à jour en rapport avec la donnée, et non plus un morceau de mémoire. l'avantage est qu'une écriture interdit uniquement l'accès à la donnée et non pas l'accès à une zone mémoire où d'autres données peuvent être stockées. L'établissement de la relation entre donnée et verrou est cependant laissé à la charge du programmeur. La cohérence à l'entrée fait la distinction entre les accès en lecture et les accès en écriture grâce à l'utilisation de deux verrous. Si l'écrivain reste unique, les lecteurs sont multiples et peuvent donc lire en parallèle. Une lecture ne peut cependant pas avoir lieu en même temps qu'une écriture sur la même donnée.

- ***Cohérence de portée :*** reprend le principe du modèle précèdent (à l'entrée) et tente d'éviter au programmeur d'effectuer lui-même l'association entre verrous et données [LIV 96]. Cette technique se base sur les instructions de synchronisation déjà présentes dans le programme.
  Lors de l'acquisition d'un verrou par un site, seules les modifications effectuées dans les portées correspondant à ce verrou sont visibles.

3.4. Localisation des données :

Un protocole mettant en œuvre un modèle de cohérence doit pouvoir localiser les données. La localisation d'une donnée revient à trouver le nœud stockant la copie la plus à jour, c'est-à-dire celui hébergeant le processus ayant effectué la dernière écriture [SEB 06].
Ce nœud est appelé *propriétaire* de la donnée. On appelle nœud *gestionnaire* d'une donnée le nœud qui est chargé de localiser son propriétaire. On distingue des gestionnaires *distribués* et des gestionnaires *centralisés* [SEB 06].

3.4.1. Gestionnaires centralisés :

Dans le cas d'un gestionnaire centralisé, toutes les requêtes d'accès à une donnée, les lectures comme les écritures, sont adressées au nœud gestionnaire choisi. Ce nœud est responsable de la gestion de l'ensemble des données. Cette gestion centralisée est bien adaptée à de petites architectures possédant des réseaux d'interconnexion performants comme les grappes de calculateurs. En revanche, elle présente des limites en terme de passage à l'échelle. Au sein des systèmes à grande échelle, les latences des communications entre les nœuds peuvent être grandes. Lorsque des nœuds éloignés en terme de latence du nœud gestionnaire accèdent à une donnée partagée, les échanges réseau entre ces différents nœuds risquent de ralentir l'application dans son ensemble. En plus des limites liées au passage à l'échelle, ces protocoles sont sensibles aux fautes. Si le nœud gestionnaire tombe en panne, l'ensemble des données partagées devient inaccessible [SEB 06].

3.4.2. Gestionnaires distribués :

Les approches basées sur un gestionnaire distribué permettent une décentralisation du gestionnaire. On distingue trois types de gestionnaire [SEB 06] :

- ***Gestionnaire distribué fixe :*** Cette approche permet un meilleur équilibrage de charge grâce à l'utilisation de plusieurs nœuds gestionnaires. Chaque nœud gestionnaire est responsable d'un sous-ensemble des données partagées. Cette approche n'est pas fiable lors du passage à l'échelle et à la tolérance aux fautes.

- ***Gestionnaire distribué à diffusion :*** Cette approche consiste à diffuser les requêtes de localisation à tous les nœuds du système. Le gestionnaire se trouve donc distribué sur l'ensemble des nœuds. Ceci implique de nombreuses diffusions globales, ce qui est très coûteux en terme de communication. Cette approche est conçue pour des systèmes de petite taille et devient impraticable dès que la taille du système augmente. En revanche la localisation des données reste possible en cas de fautes.

- ***Gestionnaire distribué dynamique :*** Cette approche reprend le principe de celle du gestionnaire distribué fixe, sauf que tous les nœuds gèrent la localisation de toutes les données. Cependant, cette localisation n'est pas exacte : les nœuds connaissent seulement un propriétaire *probable*. En cas de changement de propriétaire d'une donnée lors d'un accès en écriture, l'ancien propriétaire transmet la donnée au nouveau et le considère comme le nouveau propriétaire probable. Il pourra ainsi lui retransmettre les requêtes d'accès à la donnée. Ce mécanisme permet de mettre en place un *chaînage* afin de pouvoir systématiquement retrouver la donnée. Ces mécanismes sophistiqués permettent une localisation efficace des données au sein des systèmes réparties. En revanche, là encore, les problèmes liés au passage à l'échelle ainsi qu'à la tolérance aux fautes ne sont pas totalement résolus. En plus le mécanisme de chaînage pose des problèmes. Le chemin réseau à emprunter pour suivre le chaînage peut exploiter des liens à forte latence.

4. Conclusion :

Dans ce chapitre, les différents concepts de base concernant les éléments constituant le problème de notre projet ont été présentés : la réplication et la cohérence. Nous avons essayé d'étudier d'une façon brève chaque élément, en montrant la relation entre ces éléments.
La réplication est un outil efficace pour donner plus de disponibilité aux données, et plus de performance en terme de temps de réponse dans les systèmes distribués. Elle pose un problème majeur, la gestion de cohérence des différentes répliques. En utilisant des modèles de cohérence, ce problème est peut être surmonté, mais lors du passage à l'échelle et l'utilisation des systèmes plus étendus, tels que les grilles de données [Annexe1], ces modèles s'avèrent inefficaces.

Chapitre 2 : Travaux liés au problème de cohérence des répliques sur les grilles

1. Introduction :

Le problème de cohérence des répliques dans les grilles est un domaine très vaste. Plusieurs études et recherches ont été effectuées à ce sujet. Chaque travail traite un problème bien précis, mais touche d'une façon ou d'une autre le problème présenté.
Parmi les travaux les plus liés à notre problème on peut citer ceux de [LOI 06], [IVA 05] et de [MED 06], qui traitent le problème de la gestion de la cohérence dans les grilles avec des approches différentes. Dans ce qui suit nous présentons ces travaux.

2. Extension du modèle de cohérence à l'entrée pour la visualisation dans les applications de couplage de codes sur grilles [LOI 06]:

Ce projet a été réalisé par Sébastien Monnet et Loïc Cudennec (chercheurs d'IRISA/Université de Rennes -PARIS-), dans le cadre du problème de la visualisation des données partagées dans les applications à base de couplage de codes sur les grilles.
L'idée des auteurs est de concevoir un modèle de cohérence adaptable aux grilles de calcul, et qui répond aux exigences de dynamicité et du passage à l'échelle. Le système présenté est basé sur la plate-forme JUXMEM (Juxtaposed Memory) qui reflète une architecture matérielle de la grille de calcul, elle propose le concept de service de partage transparent de données, qui vise essentiellement les applications de calcul distribué sur les grilles. JUXMEM suit une approche hybride inspirée par les systèmes à mémoire virtuellement partagée (MVP) et les systèmes pairs à pairs (P2P).

L'amélioration vise donc le protocole de cohérence qui implémente le modèle de cohérence à l'entrée, en proposant une architecture hiérarchique, ceci permet de minimiser les communications entre les grappes, en introduisant la notion de lecture relâchée comme extension du modèle de cohérence à l'entrée. Ce nouveau type d'opération peut être réalisé sans prise de verrou, en parallèle avec des écritures. En revanche, l'utilisateur relâche les contraintes sur la fraîcheur de la donnée et accepte de lire des versions légèrement anciennes, dont le retard est néanmoins contrôlé.

D'après les auteurs, l'implémentation de cette approche au sein du service de partage de données pour grilles JUXMEM, montre des gains considérables par rapport à une implémentation classique basée sur des lectures avec prise de verrou.
Le problème dans ce travail aborde les applications de couplage des codes. Une application de couplage de code est une application distribuée particulière composée de plusieurs codes éventuellement parallèles qui coopèrent à une même fin. Un calcul global est réalisé à l'aide de plusieurs codes s'exécutant en parallèle sur des grappes de calculateurs différentes. Même si la distribution des codes est effectuée de manière à minimiser les échanges de messages entre les grappes, des données peuvent être partagées entre les différents sites. Le couplage de code est utilisé dans le calcul scientifique haute performance. Ces calculs peuvent durer des temps considérables et il n'est pas souhaitable d'en attendre la fin pour vérifier que tout s'est bien passé. Les données partagées entre les codes permettent de renseigner sur l'état d'avancement du calcul. L'observation d'un calcul doit s'effectuer avec des temps d'accès courts tout en causant le moins d'interactions possibles avec les autres sites tout en minimisant les perturbations du système.

Dans ce travail le problème traité vise l'observation des données intermédiaires dans une application à base de couplage de codes.

L'objectif est d'améliorer le comportement du protocole de cohérence lorsque des observateurs sont utilisés. La contribution principale de ce projet est de proposer une extension du protocole de cohérence qui introduit la possibilité d'effectuer des lectures en parallèle d'une écriture. Ceci est possible si l'observateur accepte d'exploiter des données dont la version est considérée comme ancienne. L'implémentation de cette stratégie a été effectuée dans le cadre de la plate-forme JUXMEM. Des évaluations de performances ont été menées pour montrer que cette solution améliore la vitesse d'accès des observateurs sans dégrader celle des autres sites.

2.1. La stratégie et le modèle proposés :

Les auteurs ont proposé d'implémenter un protocole de cohérence hiérarchique tolérant aux fautes, le système considéré est un modèle hiérarchique composé de grappes de calculateurs, avec l'architecture de JUXMEM qui reflète une architecture de la grille.
L'architecture du service proposé utilise la notion de groupes de pairs afin de prendre en compte l'architecture physique sous-jacente (Figure 3.1).

Le principe de base de cette architecture est la fédération des grappes, les pairs localisés dans une même grappe sont regroupés dans un groupe cluster, l'ensemble des groupes cluster forme un groupe de plus haut niveau appelé le groupe JUXMEM. Ce groupe rassemble tous les pairs participant au service, il présente donc une architecture hiérarchique à deux niveaux : le niveau global et le niveau local (aussi appelé niveau grappe).
Chaque pair dans les différents groupes joue un rôle bien précis, on trouve des paires gestionnaires, fournisseurs et clients.

Il y a aussi deux groupes data utilisés dans ce service, le groupe de données locale (LDG) et le groupe de données globales (GDG).
Le LDG est un groupe de pairs de type fournisseur situés dans un même groupe cluster et possédant une copie d'une même donnée. Il est le représentant local du protocole de cohérence gérant la donnée au sein du groupe cluster dans lequel il se trouve, ce qui veut dire que la cohérence à l'intérieur du groupe locale est maintenue en se basant sur les nœuds constituants ce groupe

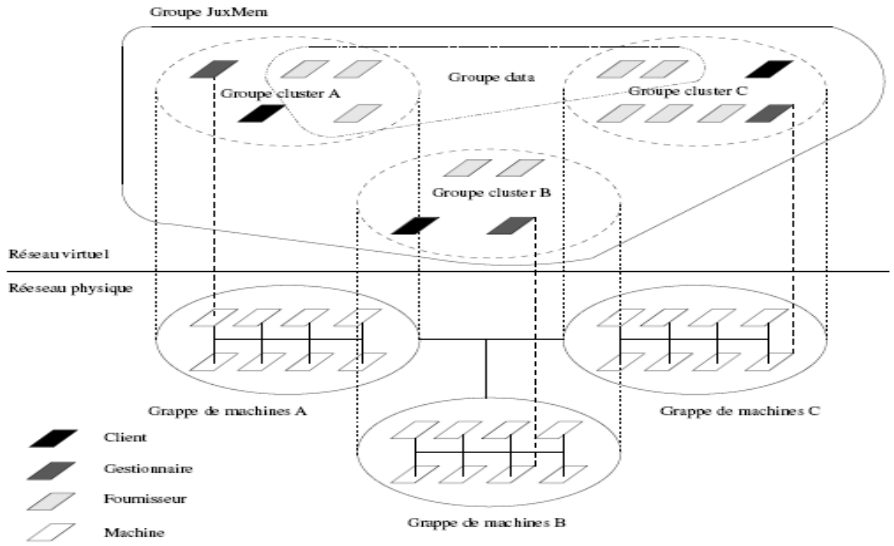


Figure 3.1. Architecture JUXMEM

Le GDG est un groupe constitué par les LDG représentants une même donnée (Figure 3.2).

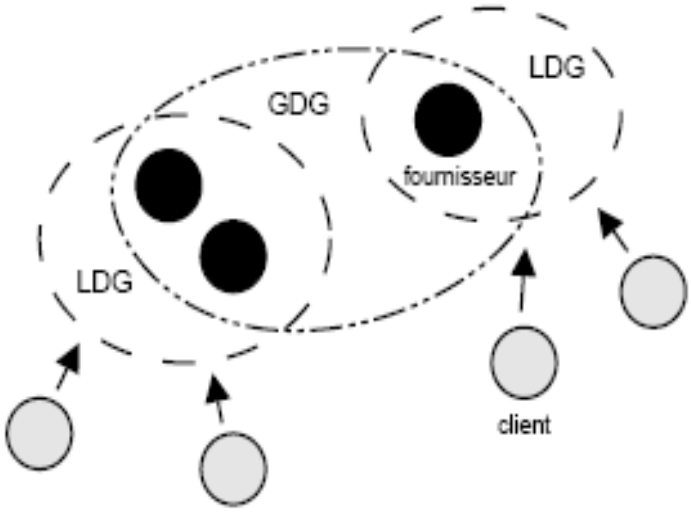


Figure 3.2. Groupes LDG et GDG du service JUXMEM

Le service proposé utilise des mécanismes de réplication. Les entités qui doivent être répliquées sont les entités critiques du protocole de cohérence comme le représentant local de chaque LDG et le représentant globale de chaque donnée (le GDG). Trois modes de réplication peuvent être utilisés : réplication passive, active ou semi-active.
La réplication est basée sur des concepts tels que la composition de groupe, la diffusion atomique

Protocole de composition de groupe : ce type de protocole a pour objectif de gérer les groupes de nœuds hébergeant les copies de référence des données. Il permet de gérer un ensemble de nœuds ayant un intérêt commun (un groupe). Chaque nœud appartenant à un groupe maintient à jour la liste de membres de ce groupe, la composition de ces listes évolue quand de nouveaux nœuds joignent ou quittent le groupe.

Diffusion atomique : ce qui exige l'utilisation d'un mécanisme de réplication pessimiste afin d'assurer qu'une copie à jour reste disponible dans le système, chaque mise à jour entraîne donc une mise à jour de tous les membres du groupe.

Le modèle de cohérence choisi est le modèle de cohérence à l'entrée, il est basé sur la particularité d'associer un objet de synchronisation spécifique à chaque donnée. Cette particularité permet un gain de performance au niveau du protocole de cohérence. En effet, lorsqu'un processus entre en section critique pour accéder à une donnée, par exemple pour lire la valeur de la donnée avant de la mettre à jour, il doit le faire via l'objet de synchronisation associé à cette donnée. Ainsi, sur un nœud, le protocole de cohérence n'a qu'à garantir la cohérence des données accédées au sein de la section critique associée. Cela permet d'éviter des mises à jour non nécessaires : seuls les processus qui déclarent qu'ils vont accéder aux données seront concernés. De plus ne seront mises à jour que les données qui seront effectivement accédées.

Deux contraintes sont exigées par ce modèle, la première est que chaque donnée partagée doit être associée à un objet de synchronisation spécifique, et la deuxième est que les accès non-exclusifs (en lecture seule) doivent être distingués des accès exclusifs (où il y a mise à jour de la valeur de la donnée), la distinction entre les deux types d'accès est faite en utilisant des

primitives différentes : « acquire_read » pour entrer en section de code où l'on effectue des accès non-exclusifs, et « acquire » pour entrer en section critique comprenant des accès de type exclusif.

2.2. L'amélioration du service :

La proposition d'amélioration est donnée par la notion de lecture relâchée. Le scénario considéré met en œuvre un site observateur dont le rôle est de lire la donnée partagée avec des temps d'accès courts sans pour autant dégrader les performances des autres sites. La première idée présentée dans cette amélioration consiste à utiliser des copies de la donnée considérées comme anciennes et détenues par le client ou son LDG.

Cette stratégie permet d'exploiter des données déjà présentes sur le site client ou très proches en terme de distance réseau. La deuxième idée vise à diminuer les temps d'attente imposés par le protocole de cohérence en ne prenant plus le verrou en lecture, en conséquence autoriser ces lectures d'un genre particulier en même temps que les écritures. Ces lectures sont nommées lectures relâchées et la primitive associée est « rlxRead ». Le modèle de cohérence à l'entrée ne garantit qu'une donnée soit à jour que lors de l'acquisition de son verrou. Dans ce modèle, pour un observateur qui n'acquiert pas le verrou, le fait d'utiliser la donnée contenue dans son propre cache ou celle disponible sur son LDG ne peut garantir que la donnée soit à jour. L'approche considérée propose d'autoriser ce type de lectures non-synchronisées tout en contrôlant la fraîcheur de la donnée. Ceci est possible en bornant l'écart entre la version retournée et la version la plus récente. Ainsi pour chaque lecture relâchée, l'application spécifie le nombre de versions de retard autorisées avant qu'une donnée ne soit considérée comme périmée.

La notion de fenêtre de lecture a pour but d'exprimer le retard entre la version la plus récente et celle retournée par la lecture relâchée, par la mise en œuvre de deux paramètres qui prennent en compte les deux niveaux de l'hiérarchie du protocole de cohérence :

- La constante D, propre à chaque donnée, exprime le nombre de fois que le LDG peut transmettre successivement le verrou en écriture, sans effectuer de mise à jour du GDG. Si D = 0 alors le LDG doit propager les modifications après chaque relâchement du verrou en écriture. Dans ce cas tous les LDG ont la même version de la donnée.
- Le paramètre w est spécifié par le client lors de chaque appel à la primitive de lecture relâchée « rlxRead ». C'est la fenêtre de lecture. Elle exprime l'écart total maximum entre la version la plus récente et celle retournée par la lecture relâchée.

Les distances D et w sont positives ou nulles et respectent le fait que w soit supérieur ou égal à D. La différence w − D correspond à la distance maximale entre la version détenue par le LDG du client et celle retournée par la lecture relâchée. A titre d'exemple, si D = 3 alors un LDG pourra distribuer successivement le verrou en écriture au plus 3 fois sans propager les modifications à tous les membres du GDG. Si w = 4, la version de la donnée lue par un client effectuant une lecture relâchée retourne soit la dernière version de la donnée détenue par son LDG, soit la version précédente. La notion de dernière version fait référence à la dernière version propagée par le LDG ayant le verrou en écriture. C'est à dire qu'elle peut déjà être ancienne de D versions.

2.3. Mises en œuvre et résultats obtenus :

La grappe utilisée pour les expérimentations fait partie de Grid'5000 et est composée de 64 nœuds bi-processeurs AMD Opteron cadencés à 2.2GHz, pourvus de 2Go de RAM et interconnectés par un réseau Ethernet gigabit. Le développement du protocole de cohérence a été réalisé avec la version de JUXMEM basée sur JXTA 2.3.2 pour J2SE (JVM 1.4.2 Sun Microsystems). Le programme développé pour l'évaluation des performances est un programme reposant sur la librairie JUXMEM. Il met en jeu les fonctionnalités du protocole de cohérence dans le cadre d'un scénario particulier. Une donnée partagée est accédée par des clients supportant les rôles d'écrivain, lecteur et observateur. Chaque LDG est composé d'un fournisseur auquel se rattache un ou plusieurs clients. L'écrivain et le lecteur sont associés au premier LDG alors que l'observateur est attaché à un deuxième LDG.
Chaque participant (écrivain, lecteur ou observateur) effectue 50 itérations d'accès aux données partagées de la manière suivante :
Pour l'écrivain : prise de verrou en écriture, l'écriture et le relâchement du verrou en écriture.
Pour le lecteur : prise de verrou en lecture, la lecture et le relâchement du verrou en lecture.
Pour l'observateur : la lecture relâchée directement
Les paramètres du système sont la taille de la donnée partagée dont les valeurs prises sont 1Ko et 1Mo, les paramètres de la fenêtre de lecture D et w.
Les résultats obtenus sont données comme suit :
Temps d'accès moyens (1ko, sans lecture relâchée pour l'observateur)
Ecrivain : 19 ms
Lecteur : 19 ms
Observateur : 19 ms
Temps d'accès moyens (1ko, D=0, w=0)
Ecrivain : 19 ms
Lecteur : 19 ms
Observateur : 9 ms
Temps d'accès moyens (1Mo, D=0, w=0)
Ecrivain : 69 ms
Lecteur : 63 ms
Observateur : 44 ms
Temps d'accès moyens (1Mo, D=3, w=6)
Ecrivain : 60 ms
Lecteur : 61 ms
Observateur : 23 ms

D'après les résultats obtenus les auteurs ont remarqué que le fait d'utiliser la primitive d'accès « rlxRead » (modèle de cohérence à l'entrée étendu) par l'observateur a permis d'obtenir un temps d'observation moyen 2 fois plus petit que celui où on n'utilise pas cette primitive (modèle de cohérence à l'entrée simple). Les évaluations de l'extension du protocole de cohérence ont montré que les lectures relâchées permettaient un gain de temps considérable par rapport à une lecture avec prise du verrou.

3. Reconfiguration dynamique de coterie structurée en arbre [IVA 05]:

Le travail a été réalisé par Ivan Frain, Jean-Paul Bahsoun et Abdelaziz M'zoughi (Institut de Recherche en Informatique de Toulouse), l'objectif est d'adapter un protocole de cohérence à quorum à un environnement distribué à large échelle comme les grilles de calcul et qui répond aux exigences de la dynamicité des nœuds composants ces systèmes.

Dans des environnements distribués aussi dynamiques que les grilles informatiques où les nœuds peuvent quitter le réseau volontairement ou involontairement (cas des pannes), leurs charges peuvent varier au cours du temps, il est important que les mécanismes employés pour la gestion des données soient capables de s'adapter à cette dynamicité.

Il y a des protocoles de cohérence à quorum qui prennent en compte les pannes des nœuds du réseau mais ils ne prennent pas en compte le concept de la variation de la charge des nœuds. Les auteurs introduisent dans ce travail une reconfiguration basée sur des permutations élémentaires des nœuds de l'arbre constituant la coterie. Grâce à cette reconfiguration, ils permettent d'améliorer le débit d'opérations traitées par le système pour donner des résultats meilleurs par rapport à un système ne faisant pas de reconfiguration.

La réplication dans les grilles de calcul est une exigence majeure, cependant optimiser les accès aux répliques et maintenir la cohérence entre elles restent un problème très important, pour cela les auteurs se basent sur un protocole à quorum qui réduit le nombre de messages à échanger pour effectuer une opération de lecture/écriture.

Le système considéré par les auteurs met en œuvre un protocole de cohérence à quorum basé sur la reconfiguration dynamique qui, non seulement assure le maintien de la cohérence entre les données répliquées du système, mais aussi peut s'adapter au changement des charges des nœuds qui le composent.

Le problème visé dans ce travail est la cohérence de données répliquées dans un système distribué à large échelle, deux notions sont introduites :

- **Protocole à quorum :** les auteurs ont proposé ce type de modèles de cohérence à cause de son efficacité dans les systèmes distribués, avec la réduction du nombre de messages échangés entraînés lors d'une opération de lecture ou d'écriture. Il faut noter aussi que certains des protocoles à quorum prennent en compte les pannes des nœuds.
- **Reconfiguration dynamique :** la reconfiguration consiste à effectuer des changements sur l'arbre qui représente une structure logique des nœuds, elle permet d'adapter le protocole proposé avec les changements des charges des nœuds constituant cet arbre.

L'idée des auteurs dans ce travail est de proposer un procédé de reconfiguration de coterie permettant d'améliorer le débit d'opérations de lecture/écriture traitées par le système lorsque la charge des nœuds évolue au cours du temps et génère un déséquilibre.

3.1. Le modèle proposé :

Le modèle de cohérence utilisé dans ce travail est basé sur un protocole à quorum. Ces protocoles permettent d'optimiser l'accès aux données en utilisant des mécanismes de lecture/écriture entraînant le plus petit nombre de messages à échanger par opération.
Un quorum est l'ensemble minimum de nœuds, possédant une réplique, à contacter pour effectuer une opération de lecture ou d'écriture sur une donnée.
Une coterie est l'ensemble de tous les quorums disponibles. On peut distinguer deux types de quorums : les quorums à consensus et les quorums structurés. Les premiers permettent de ne contacter qu'un petit nombre de répliques, qui peut atteindre jusqu'à (n/2) messages (n étant le nombre de répliques). Les protocoles à quorums structurés permettent de diminuer encore plus le nombre de messages à échanger. C'est le cas des protocoles : à arbre : log2(n) messages, hiérarchique $n^{0.63}$ messages, ou à grille : ($\sqrt{n}$) messages.

Dans des environnements dynamiques comme les grilles de calcul, la charge des nœuds peut évoluer au cours du temps, si des nœuds tombent en panne ou quittent le réseau, le nombre de messages échangés peut augmenter très rapidement, le protocole employé doit agir en conséquence.

Le modèle proposé par les auteurs est une grille composée d'un ensemble de nœuds P, à chaque nœud est associée une charge de travail notée par X(P), chaque nœud possède une réplique de donnée accédée en lecture ou écriture. Un quorum Q représente l'ensemble minimum de nœuds P, possédant une réplique de donnée, entraînés dans une opération de lecture ou d'écriture sur une donnée. Une coterie C est l'ensemble de tous les quorums possibles pour un groupe de répliques.
Tous les protocoles à quorum sont tous régis par deux propriétés qui sont les propriétés d'intersection et de minimalité.

Intersection : $\forall Q1, Q2 \in C : Q1 \cap Q2 \neq \emptyset$.

Minimalité : $\forall Q1, Q2 \in C : Q1 \not\subset Q2$.

Dans ce travail les auteurs ont introduit la notion de la charge d'un quorum et la charge d'une coterie.

La charge d'un quorum $Y(Q) = Max(X(P) / P \in Q)$

La charge de coterie $S(C) = Somme (Y(Q) / Q \in C)$

Le modèle choisi par les auteurs pour représenter leur protocole de cohérence à quorum, est une structure de coterie en arbre.
Les processeurs sont organisés logiquement en arbre. Les processeurs sont des nœuds ou des feuilles de l'arbre. Une opération de lecture ou d'écriture est effectuée sur un quorum de la coterie. Un quorum est obtenu en prenant tous les processeurs de n'importe quel chemin allant de la racine à une feuille de l'arbre (Figure 3.3).

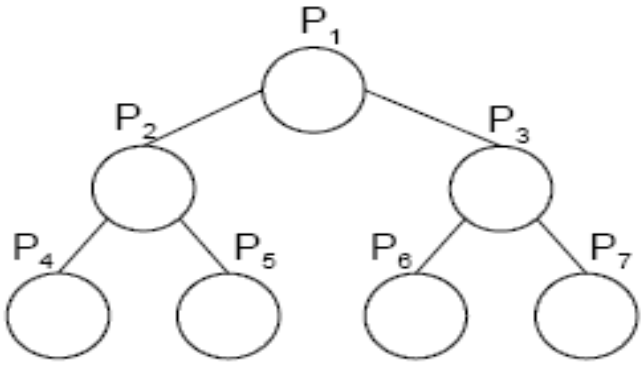


Figure 3.3. Coterie structurée en arbre

Ce protocole est classé parmi les protocoles à quorum structuré. Les propriétés d'intersection et de minimalité sont bien vérifiées. De plus, ce protocole propose un nombre inférieur de message à échanger (log2 (n)).

La notion de permutations élémentaires introduite par les auteurs dans ce travail permet la reconfiguration de la coterie structurée en arbre.
La reconfiguration d'une nouvelle coterie est effectuée par l'application d'une ou de plusieurs permutations élémentaires (Figures 3.4 et 3.5). On applique une permutation élémentaire sur une coterie afin d'obtenir une nouvelle coterie moins chargée. Il faut d'abord trouver deux nœuds dans l'arbre qui sont parents : un père et son fils ; tels que la charge du fils soit

inférieure à la charge du père. Une permutation élémentaire consiste à permuter le nœud père avec le nœud fils s'ils satisfont la contrainte de charge. Cela a pour effet de positionner le processeur le moins chargé au-dessus du processeur le plus chargé.

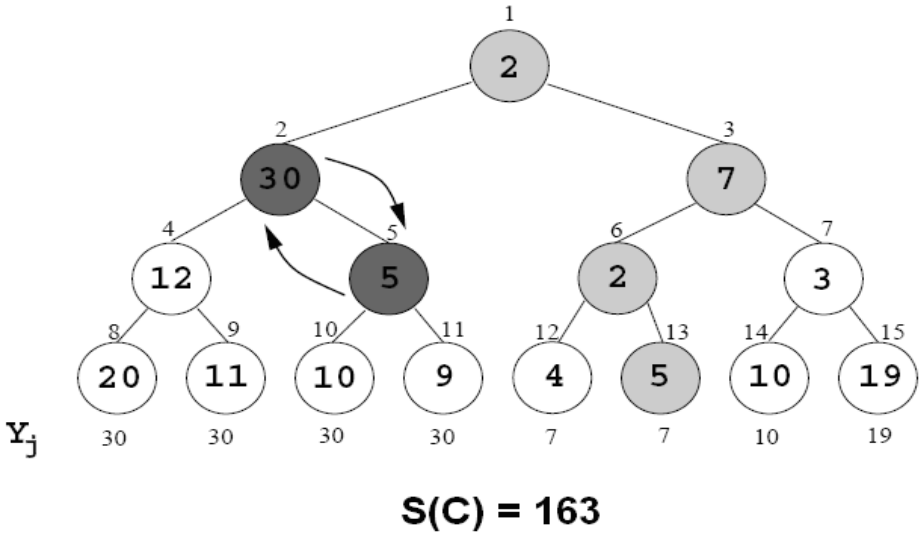


Figure 3.4. Charge de la coterie

Dans l'exemple illustré dans la figure 3.4 les cercles (nœuds de l'arbre) représentent des processeurs, les valeurs à l'intérieur des cercles correspondent aux différentes charges des nœuds, le nombre au-dessus des cercles représente le numéro du nœud, un quorum est l'ensemble des nœuds partant de la racine vers une feuille de l'arbre, par exemple : {1, 3, 6, 13} et la charge de ce quorum est égale à 7.
La valeur sous chaque feuille de l'arbre représente la valeur de la charge du quorum contenant cette feuille, la charge de la coterie est donnée par S(C) et elle est égale à la somme des charges de tous les quorums de l'arbre.
Dans l'exemple plusieurs permutations élémentaires sont possibles, par exemple le nœud 2 et le nœud 5 (père et fils) comme on le voit dans la figure 3.4, la charge du fils X(5)=5 est inférieure à la charge du père X(2)=30, on effectue donc la permutation entre ces deux nœuds et on obtient une nouvelle coterie D tel que $S(D) \leq S(C)$ (Figure 3.5).

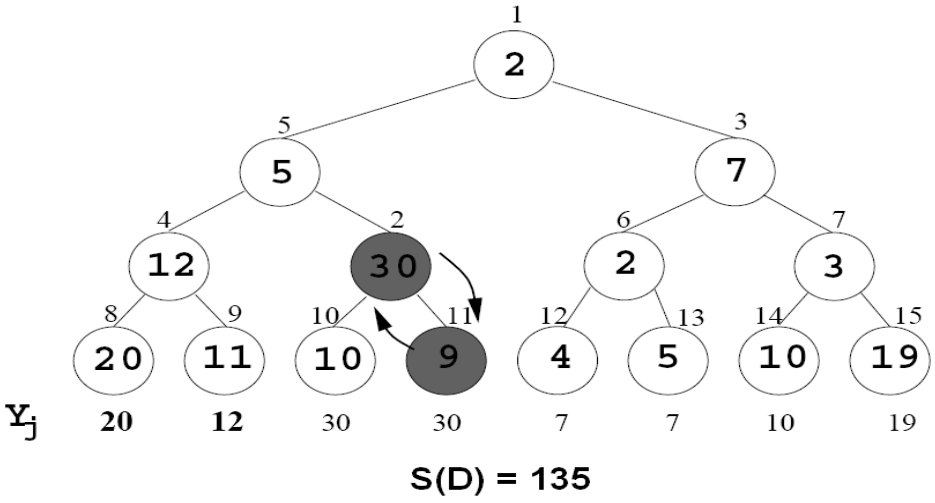


Figure 3.5. Permutation élémentaire entre le nœud 2 et le nœud 5

La procédure consiste à effectuer plusieurs permutations élémentaires successives jusqu'à ce qu'il n'ait plus de permutations à effectuer. La coterie obtenue aura la plus petite charge (Figure 3.6).

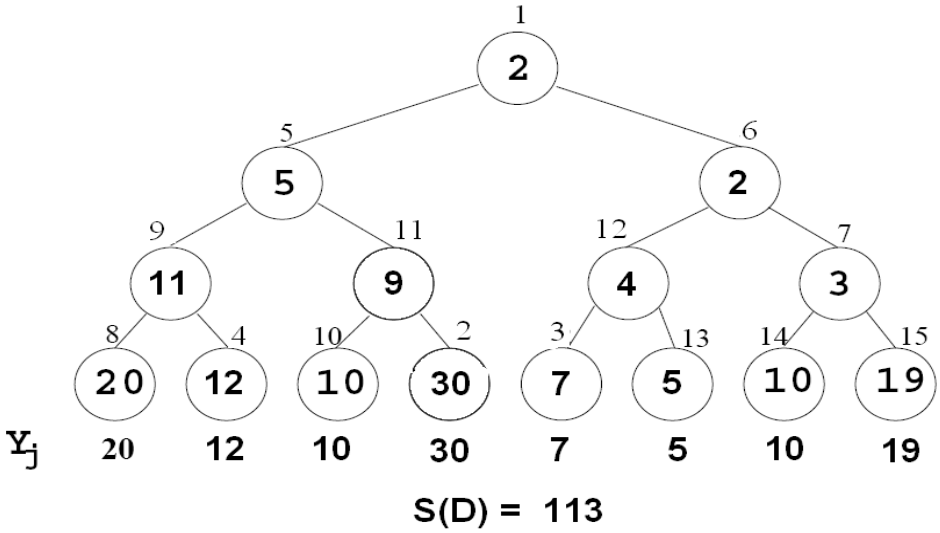


Figure 3.6. Réduction de la charge de la coterie

3.2. Evaluation de l'approche et résultats :

Afin d'évaluer l'approche présentée dans ce travail les auteurs ont proposé un algorithme qui combine le service de lecture / écriture avec un service de reconfiguration dynamique de la coterie structurée, l'évaluation a été effectuée à l'aide du simulateur NEKO [Annex2].
Une opération de lecture ou d'écriture d'une donnée est effectuée via l'interface fonctionnelle du service lecture/écriture atomique et elle est décomposée en deux phases : la phase de requête où un quorum est contacté en lecture et chacun des nœuds renvoie la valeur et la version de la donnée, ainsi que les informations concernant la coterie courante. Une fois toutes les réponses collectées, la version la plus récente de la donnée est utilisée. La phase de propagation où un quorum est contacté en écriture afin de propager la nouvelle valeur et la nouvelle version de la donnée. Les deux phases sont effectuées systématiquement pour une lecture ou une écriture de la donnée.
La reconfiguration est effectuée via l'interface de gestion, elle peut être effectuée en même temps que les opérations de lecture ou d'écriture, et elle est effectuée par un reconfigureur qui peut être soit un nœud élu, soit un nœud dédié. Le protocole de reconfiguration est découpé en trois phases. La phase d'installation, le reconfigureur contacte un ensemble de processeurs, il s'agit de l'union de deux quorums (un en lecture et l'autre en écriture) ; en envoyant la nouvelle configuration. Les processeurs renvoient au reconfigureur la valeur et la version de la donnée qu'ils possèdent. Lorsque toutes les réponses sont arrivées, le reconfigureur entre dans une phase de propagation identique à celle du protocole de lecture/écriture précédent. Enfin, la phase d'acquittement, permet de confirmer la mise en place de la nouvelle configuration (Figure 3.7).

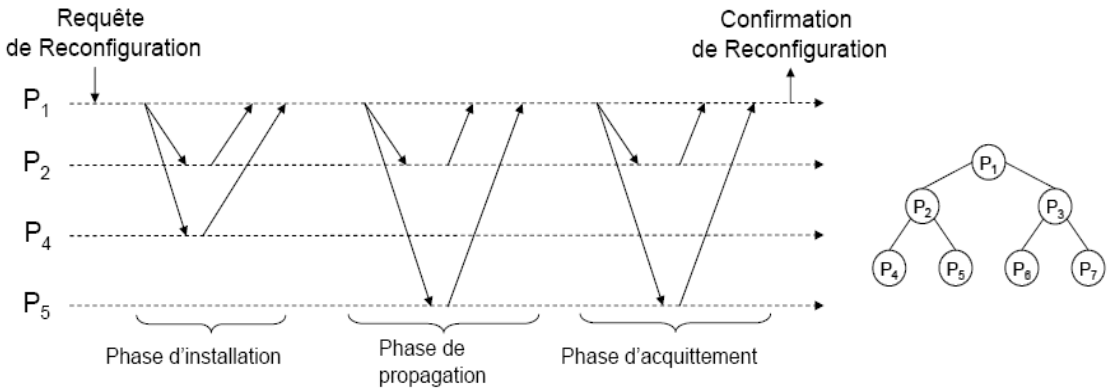


Figure 3.7. Interface de gestion : Reconfiguration

Afin de compléter et améliorer le service présenté, les auteurs ont ajouté trois fonctionnalités complémentaires :

- **Une politique d'information :** consiste à récupérer les informations de la charge des processeurs du quorum assemblé pendant la phase de propagation d'une opération de lecture ou d'écriture. La récupération des informations de charge n'aura pas un coût supplémentaire car l'idée des auteurs est de profiter des messages déjà échangés par le protocole de lecture / écriture.
- **Une politique de sélection :** permet de définir au mieux le moment de reconfiguration, par défaut la reconfiguration est déclenchée après chaque opération de lecture ou écriture.
- **Une politique de reconfiguration** : permet de choisir le nombre et l'ordre des permutations élémentaires à effectuer, en fonction de l'état de la coterie et l'état du réseau.

L'implémentation de l'algorithme étendu a été réalisée sur le simulateur NEKO. Les paramètres prises en considération sont le nombre de requêtes initiées par session (la durée pendant laquelle un nœud a une charge constante) et le débit d'opérations traitées par le système par session, la charge des nœuds du système évolue dans le temps de façon aléatoire, et cette évolution est indépendante des opérations de lecture / écriture effectuées dans le système.
D'après les auteurs, les résultats obtenus ont montré l'efficacité d'implémenter la reconfiguration élémentaire, surtout lorsque le nombre de requêtes initiées dépasse 50.
Les auteurs ont remarqué que le système effectuant des reconfigurations permet de traiter jusqu'à 25% d'opérations en plus (pour cent requêtes) par rapport au système n'effectuant pas de reconfiguration.

4. Un protocole de gestion de cohérence dans un environnement de simulation Optorsim [MED 06]:

Ce travail a était réalisé par [MED 06], dont l'objectif a été d'étendre le simulateur de grilles OptorSim par un module de gestion de la cohérence.
Les auteurs, dans ce travail, ont utilisé un modèle de grille hiérarchique à deux niveaux, modélisé par un arbre en anneau, la conception du modèle a été réalisée à l'aide du simulateur OptorSim et le concept de réplication optimiste a été introduit. Bien que les approches pessimistes de réplication assurent une cohérence forte des répliques, ces approches ne sont pas adaptables sur des réseaux à grande échelle tels que les grilles, à cause de l'incertitude des

réseaux, les distances importantes qui séparent les différents nœuds, la présence des équipements de filtrage et de sécurité qui ralentissent la propagation des mises à jour. Un nombre très grand de répliques rend les écritures plus lentes et plus difficiles.
Les auteurs ont donc proposé d'utiliser une combinaison entre deux approches de réplication : optimiste et pessimiste, et ont intégré quelques mécanismes constituants un gestionnaire de cohérence dans une grille de calcul simulée avec OptorSim.

4.1. Le modèle proposé :

La topologie proposée pour modéliser une grille de calcul dans l'environnement OptorSim est un modèle hiérarchique à deux niveaux (Figure 3.8). Le modèle hiérarchique vise à distribuer les communications et les responsabilités, donc à diminuer le nombre des requêtes adressées à un groupe de nœuds. Il est constitué de groupement d'arbres de deux niveaux. Chaque arbre hiérarchique suppose une racine où les méta-données sont centralisées. Cette racine est identifiée par le représentant des fils de l'arbre, pour participer à la gestion de la cohérence globale. Cette architecture en hiérarchie permet facilement le passage à l'échelle. Le niveau bas du modèle hiérarchique est constitué d'un ensemble de sites, chaque site est composé d'un groupe de nœuds selon une disposition quelconque (anneau, étoile). Un nœud représente un élément de calcul qui est associé à un élément de stockage. Le niveau haut est un groupement des représentants des sites, dont l'objectif est d'assurer le maintien de la cohérence pessimiste.

Le processus du maintien de cohérence combine entre les deux approches : pessimiste et optimiste. La démarche proposée implémente l'approche optimiste à l'intérieur des sites et l'approche pessimiste entre les représentants des sites.

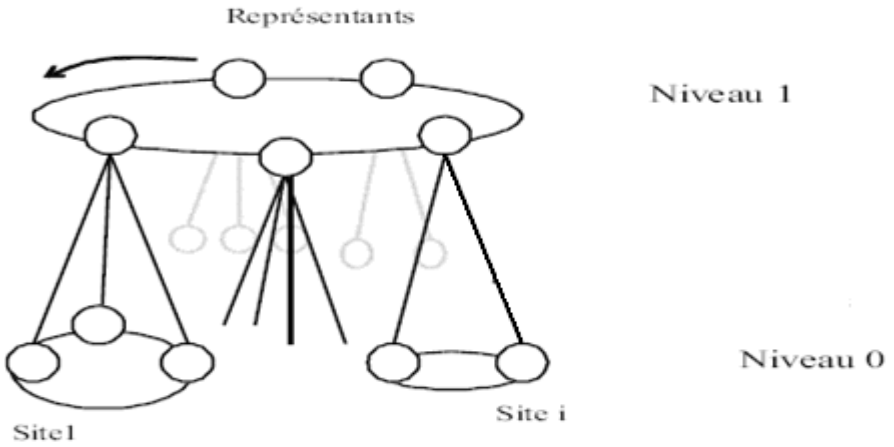


Figure 3.8. Modèle de la grille à deux niveaux

L'approche de réplication optimiste garantit l'accès n'importe quand et à n'importe quelle réplique du système, mais en conséquence, elle doit laisser parfois leur contenu diverger temporairement. Ceci est dû à la propagation des mises à jours retardée. Les accès concurrents entre les différents nœuds du système peuvent générer des désordres d'exécution des opérations de mises à jour en plus des conflits.
La cohérence est réalisée par la combinaison de quatre mécanismes :

- **La propagation des mises à jour** : la propagation épidémique est souhaitable : n'importe quel site communique avec les autres et effectue les opérations de ses propres mises à jour, il annonce par la suite les mises à jour aux autres sites.

- **L'ordonnancement** : définit la politique de commande de mise à jour. Il est nécessaire parce que les différents sites peuvent recevoir les mêmes mises à jour dans un ordre différent. L'ordonnancement a deux buts :
  1-Appliquer les mises à jour rapidement pour réduire la latence.
  2-Respecter les intentions des utilisateurs et éviter la confusion.

- **Détection et résolution des conflits** : les conflits se produisent parce que les utilisateurs mettent à jours les données sans connaitre les autres mises à jour effectuées par d'autres utilisateurs d'autres sites. Dés qu'un conflit est détecté il faut remplacer les mises à jour contradictoires, la politique «last writer Win » choisit la mise à jour la plus récente.

Pour la propagation des mises à jour dans les différents sites du système un modèle de réplication à multi-maîtres est utilisé, dans ce modèle, les sites sont interconnectés par un réseau. Chaque site possède une copie des objets partagés. Sur un site, une réplique peut être modifiée au moyen d'opérations. Quand une réplique est modifiée sur un site, l'opération correspondante est immédiatement exécutée sur ce site, puis propagée aux autres sites pour y être ré-exécutée. Lorsque deux répliques de deux sites différents sont modifiées en parallèle, les répliques divergent. Il est donc possible d'observer au même moment une valeur sur un site qui est différente sur un autre site. Le modèle doit donc assurer la convergence des répliques.

Un des mécanismes utilisés pour gérer ce genre de situation est la règle de THOMAS. Cette règle est utilisée pour assurer la convergence dans Usenet. Par exemple on considère trois sites interconnectés par un réseau informatique (Figure 3.9). Chaque site contient une copie de la même donnée une chaîne de caractère S par exemple. La règle de Thomas associe à chaque réplique sur un site i une estampille Ti (h, s) où h est une heure et s un numéro de site.

Figure 3.9. Réseau de trois sites

Pour modifier sa réplique, un site génère une opération de mise à jour contenant la nouvelle valeur et une estampille. Cette estampille contient l'heure courante du site et le numéro de site. Cette opération est exécutée immédiatement sur le site puis propagée aux sites voisins. Par exemple l'opération op1=set(S,"AXB", (13h, 1)) est générée à un moment donné par le site1.

Lorsqu'une opération est reçue sur un site, ce dernier compare l'estampille de l'opération à celle de sa réplique. Si l'estampille de l'opération est plus récente alors il exécute l'opération et met à jour l'estampille, sinon, il ignore l'opération. D'après les auteurs, une estampille T1 (H1, S1) est plus récente qu'une estampille T2 (H2, S2) si : (H1 > H2) ou ((H1 = H2) et (S1 > S2)).

De cette manière la règle de THOMAS gère les opérations concurrentes, si on considère un scénario où il y a ce cas (Figure 3.10), op1 et op2 sont deux opérations concurrentes exécutées localement et propagées aux sites voisins.

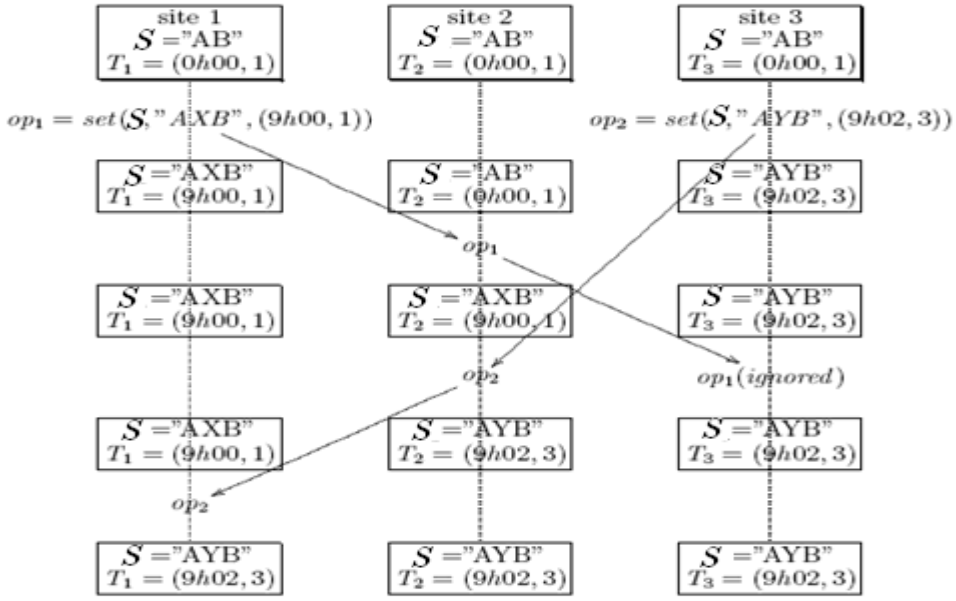


Figure 3.10. Convergence des copies avec la règle de THOMAS

Quand op1 arrive sur le site 2, son estampille (9h00, 1) étant plus récente que celle de la réplique (0h00, 1), op1 est exécutée. La valeur de S sur le site 2 devient "AXB" et l'estampille est mise à jour. A l'arrivée d'op2 sur le site 2, l'estampille d'op2 est comparée avec celle de la réplique. Comme l'estampille (9h02, 3) est plus récente que l'estampille (9h00, 1), op2 est exécutée et la valeur devient "AYB".

Lorsqu'op1 arrive sur le site 3, son estampille (9h00, 1) est comparée par rapport à l'estampille de la réplique (9h02, 3). Puisque l'estampille de l'opération est plus ancienne, l'opération est ignorée et la valeur de S ne change pas. Quand op2 arrive sur le site 1, elle est exécutée et la valeur devient "AYB". Au final, toutes les répliques ont bien convergé vers la même valeur "AYB".

Afin de réduire le taux des conflits et augmenter les performances des communications entre les répliques, les auteurs ont proposé d'adapter une topologie de communication, parmi les topologies qui peuvent être utilisées. Dans ce cadre la topologie étoile, où un site centrale est désigné pour gérer les communications entre les différents sites, le taux des conflits est réduit ainsi que la propagation des mises à jours est distribuée rapidement mais le nœud centrale peut devenir trop chargé et s'il tombe en panne ça devient un point d'échec.

Une autre topologie est utilisée celle à deux-tiers, elle est une généralisation de la topologie étoile, les sites sont divisés en deux catégories : les plus connectés « nœuds du noyau » désignés pour la gestion des communications et transfert des mises à jours, et les moins connectés « nœuds mobiles » communiquent avec les autres sites via les nœuds du noyau aux quels qui sont rattachés. Cette topologie garantie une meilleure gestion mais elle est difficile à mettre en œuvre.

4.2. Evaluation de l'approche et résultats :

Afin d'évaluer l'approche proposée, les auteurs ont intégré dans le simulateur OptorSim un module de gestion de la cohérence, le modèle de la grille utilisée est une architecture hiérarchique à deux niveaux : le niveau haut regroupe des nœuds dits « représentants », et à chaque représentant est rattaché un ou plusieurs nœuds qui sont regroupés dans le niveau bas. Chaque nœud du système représente un élément de calcul qui est associé à un élément de stockage.
Le module de gestion de cohérence comporte plusieurs algorithmes, parmi ces algorithmes on peut citer :

- **L'algorithme optimiste** : basé sur l'application du modèle de réplication optimiste. Son objectif principal est d'assurer la cohérence entre un représentant et les nœuds qu'il rattache. Donc il préserve une cohérence faible dans le niveau bas, Il est exécuté d'une façon distribuée sur chaque site du système. Dans cet algorithme l'utilisateur peut choisir entre deux systèmes à utiliser :
  1. Système multi-maîtres : avec ce système tous les nœuds peuvent exécuter les requêtes de lecture ou d'écriture d'une façon indépendante. Ils communiquent ensuite avec les autres sites pour propager les mises à jour.
  2. Système à simple maître : avec ce système les opérations de lecture peuvent être exécutées sur n'importe quel nœud, par contre pour les opérations d'écriture il faut contacter le maître de la réplique concernée, et la propagation des mises à jour aux autres nœuds est effectuée via ce nœud (le maître).

  Dans les deux cas de systèmes, la propagation des mises à jour est effectuée après l'exécution de la requête de lecture ou d'écriture.

- **L'algorithme pessimiste :** Cet algorithme permet d'assurer la cohérence au niveau haut du modèle, les nœuds de ce niveau possèdent des informations méta-données, ils sont appelés représentants. Un Super-Maître est désigné, à partir de tous les représentants (appelé aussi le Super-Représentant) en fonction de son estampille (celui qui contient la dernière version de la donnée). La cohérence de ce niveau est forte, l'ensemble des nœuds de cette couche converge immédiatement vers la même réplique du Super-Maître.

- **L'algorithme hybride :** Tout au long de la simulation, L'algorithme hybride permet de synchroniser entre les algorithmes Optimiste et Pessimiste. Via cet algorithme le lancement de l'algorithme optimiste est effectué sur chaque site, si le système choisi est celui à simple maître, cet algorithme se charge pour designer le maître comme représentant, sinon (système à multi-maîtres) un représentant est élu. Cet algorithme assure le transfert de la donnée du nœud qu'il l'a modifié vers le représentant, et depuis ce dernier vers les autres représentants.

Les paramètres que les auteurs ont utilisés pour évaluer leur travail sont :

- Le compte de conflits : qui représente le degré de conflits rencontrés sur le système, ce paramètre est incrémenté à chaque fois que l'algorithme implémentant la règle de THOMAS est exécuté. C'est le nombre de conflit par rapport au nombre des sites du système, ce paramètre a permis aux auteurs d'évaluer les performances entre un système qui utilise le gestionnaire de cohérence et un autre qui ne l'utilise pas.

- La distance : la distance est liée au nombre de mises à jour, elle est égale à la différence entre le nombre de mises à jour maximale et celui minimale dans les différents sites. D'après les auteurs ce paramètre est influencé par l'implémentation du système à simple maître ou un système multi-maître, il est aussi relié par l'application du gestionnaire de cohérence.

D'autres paramètres ont été utilisés dans l'évaluation comme le nombre de sites et de répliques utilisées, le temps de simulation, la moyenne de compte de conflits et la moyenne de distance sur chaque site.

D'après les auteurs, l'implémentation du gestionnaire de cohérence a permis d'augmenter les performances en ce qui concerne le nombre de conflits et la distance des mises à jour par rapport au nombre de sites exploités sur le système (Figure 3.11).

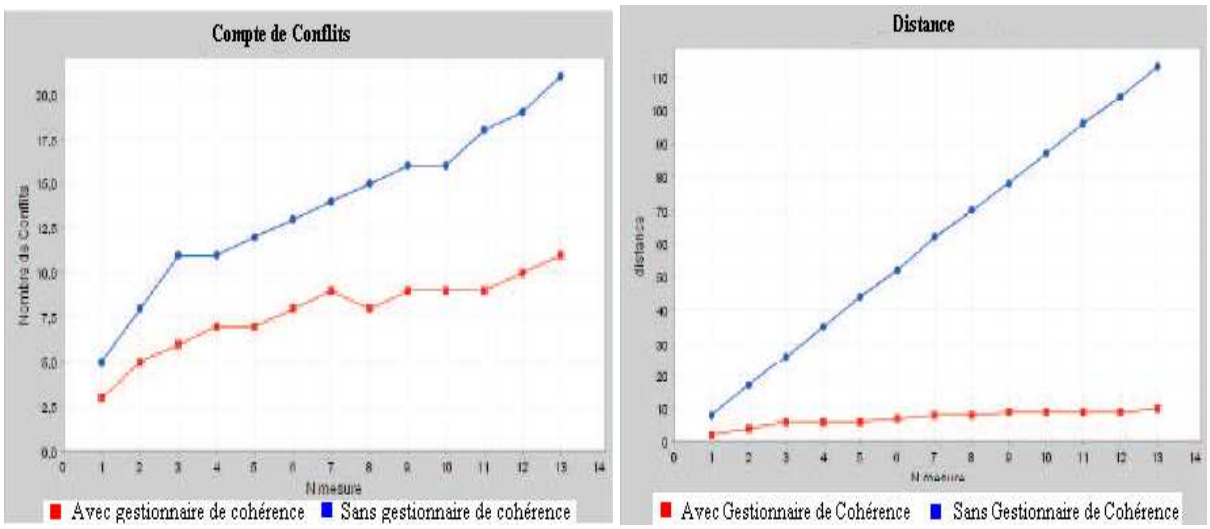


Figure 3.11. Résultats de compte de conflits et de distance

5. Discussions :

Après avoir présenté quelques travaux, nous voulons présenter quelles que propositions à relever pour chaque travail, que nous allons essayer d'utiliser par la suite dans l'implémentation de notre approche.
Dans le premier travail, le point qui nous a intéressé est celui que les auteurs ont proposé pour l'amélioration de leur service où ils ont utilisé la primitive de lecture relâchée, ce qui permet des accès en lecture en parallèle avec une écriture. Ceci permet un accès à une donnée en lecture même si elle est verrouillée en écriture, tout en contrôlant l'écart entre la version la plus récente et la version retournée de la donnée par une borne de fraicheur spécifique au système. Ce que nous pouvons proposer pour notre projet est d'utiliser des versions pour chaque réplique sur chaque nœud du système, comme ça on donne la possibilité d'accéder à une données autant de fois qu'on le souhaite selon le nombre de versions autorisées, tout en appliquant un protocole de réconciliation qui permet de maintenir la cohérence entre les versions d'une même donnée.

Dans le deuxième travail, les auteurs ont utilisé un protocole de cohérence à quorum structuré en arbre combiné avec un modèle de cohérence forte (atomique). L'idée que nous souhaitons appliquer, celle que les auteurs ont mentionné dans les perspectives de leur travail, est

d'utiliser le protocole à quorum avec un modèle de cohérence relâché. Le modèle que nous avons choisi pour notre approche est le modèle de cohérence à l'entrée.

Dans le troisième travail les auteurs ont utilisé la règle de THOMAS qui exige, lors des propagations des mises à jour concurrentes, que le dernier qui écrit gagne, c'est-à-dire la version de donnée qui va être prise en compte est celle retournée par le nœud qui l'a envoyé le dernier en terme de temps de modification. Ce comportement entraine des pertes d'information. A chaque fois qu'on a un cas similaire, la version qui va être propagée est celle retournée par le nœud qui a effectué la dernière modification, par conséquent les versions retournées par les autres nœuds vont être ignorées. Ce que nous proposons pour améliorer cette règle est de contrôler la version qui va être propagée par l'introduction de deux variables : la priorité des nœuds et la marge de fraicheur. Lorsqu'on rencontre un cas de mises à jour concurrentes on décide, en utilisant ces deux variables comme nous allons le présenter dans le chapitre suivant, quelle version va être propagée.

6. Conclusion :

On a pu dans ce chapitre faire une étude générale sur quelques travaux liés à notre problématique, même si chacun traite un problème bien précis et qui sort un peu du cadre du problème proposé, par exemple le premier vise la visualisation des applications de couplage de codes, par la proposition d'une extension du modèle de cohérence à l'entrée afin d'améliorer le service de partage des données répliquées, le deuxième vise l'amélioration des accès en lecture/écriture par la proposition d'une approche qui permet la reconfiguration dynamique d'une coterie structurée en arbre afin de diminuer sa charge et de maintenir la cohérence des répliques, le troisième traite le problème de réplication sur une plate-forme de simulation de grilles (OptorSim), l'idée proposée est de combiner entre deux modes de réplication : optimiste et pessimiste, afin de fournir un gestionnaire de cohérence des répliques sur une grille de calcul.

Dans le chapitre suivant, nous exposons l'approche que nous proposons, basée sur les propositions émises dans la partie discussions.

Chapitre3 : Approche Proposée

1. Introduction :

D'après ce qu'on a vu dans le chapitre précédent, par la présentation de quelques travaux liés au problème, maintenir la cohérence des répliques de données sur un environnement distribué comme les grilles de calcul n'est pas une tache facile, surtout en ce qui concerne le compromis entre cohérence et performance.

Dans le premier travail [LOI 06] présenté dans le chapitre précédent, les auteurs ont utilisé un protocole de cohérence implémentant un modèle de cohérence à l'entrée, qui est un modèle relâche basé sur l'utilisation de primitive de verrou pour accéder aux données en lecture ou écriture. Même si le problème visé était celui de la visualisation (observation) dans les applications de couplage de codes, le choix du modèle de cohérence à l'entrée était basé sur le fait que dans ce modèle le verrou concerne seulement la partie de la donnée qu'on veut accéder en lecture ou en écriture. De plus le verrou en lecture permet l'accès de plusieurs lecteurs en même temps par contre un seul rédacteur peut accéder à la donnée à la fois. Les auteurs ont amélioré leur travail par la suite par l'introduction de lecture relâchée ce qui permet de lire des versions légèrement anciennes (avec le contrôle de la fraîcheur de la donnée) d'une donnée même si elle est verrouillée en écriture.

Dans le deuxième travail, les auteurs ont montré l'efficacité d'implémenter les protocoles à quorum sur les grilles de calcul. L'approche proposée est basée sur la combinaison entre un protocole à quorum en arbre et un service de lecture / écriture atomique. Ceci permet de conserver la cohérence des répliques par la garantie de retourner la dernière version de la donnée par chaque lecture, et par la garantie aussi qu'au moins dans un quorum, tous les nœuds contiennent des répliques identiques. L'amélioration de l'approche a été effectuée par l'introduction de reconfiguration dynamique de coteries (une coterie est l'ensemble de tous les quorums) par le contrôle des charges dues aux opérations de mises à jour sur le système, et donc offrir une performance au service.

Dans le troisième travail, les auteurs ont proposé une approche qui utilise une hybridation entre le mode de réplication optimiste et le mode pessimiste.
Pour la gestion des mises à jour concurrentes ils ont utilisé la règle de THOMAS qui implémente la politique « le dernier qui écrit gagne » ç à d la mise à jour du dernier nœud qui a effectué l'écriture va être prise en considération et va être propagée.

L'idée de base de l'approche qu'on veut proposée, est de combiner un protocole à quorum en arbre avec un modèle de cohérence relâché, et utiliser une extension de la règle de THOMAS, basée sur les notions de priorité et de versions de données pour gérer les mises à jour concurrentes.

2. le problème posé et le modèle proposé :

Le problème vise le maintien de la cohérence des répliques de données sur une grille de calcul.
La grille de calcul est un ensemble de nœuds interconnectés par un réseau (Figure 4.1). Elle est utilisée afin de réaliser une tache bien précise (calcul scientifique par exemple). Les nœuds sont des éléments de calcul (PC, Serveurs, …) qui stockent dans leurs unités de stockage une ou plusieurs répliques de données utilisées pour réaliser le travail considéré.

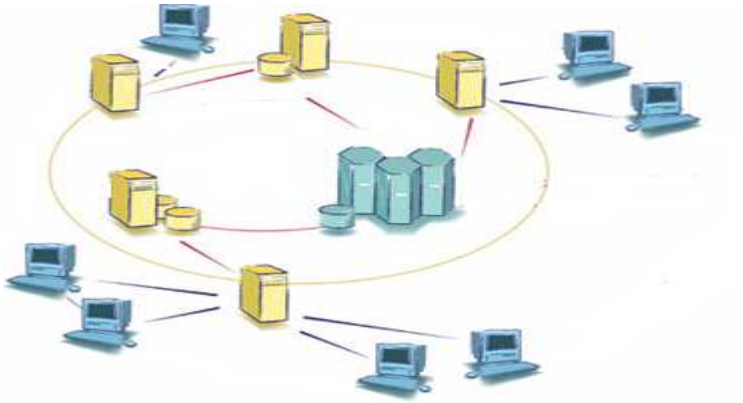


Figure 4.1. Exemple de grille de calcul

La grille est caractérisée par le concept dynamique, chaque nœud de la grille peut quitter le réseau ou le rejoindre à n'importe quel moment, en plus les données sur les nœuds peuvent être supprimées ou déplacées vers d'autres nœuds.
La réplication de données est nécessaire sur de tels systèmes afin d'augmenter la performance et la disponibilité des données. On trouve donc des données répliquées sur certains nœuds. De plus ces données peuvent être partagées par les différents nœuds.
Les données peuvent être des fichiers ou des répertoires de fichiers (des bases de données par exemple), le système doit donc suivre un modèle pour le placement des répliques sur les différents nœuds de la grille.
Le modèle qu'on a proposé pour organiser les nœuds de la grille est une organisation hiérarchique, ce qui permet de réduire le coût de communication et d'échanges d'informations entre les nœuds.

Exemple : on considère une grille composée de 15 nœuds qui partagent 4 fichiers w, x, y et z. Chaque nœud contient une ou plusieurs répliques des quatre fichiers, la matrice (Figure 4.2) représente la correspondance entre les nœuds et les répliques qu'ils contiennent.
Par exemple chaque fichier est répliqué sur 7 nœuds.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
W	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**0**
X	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**
Y	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**0**
Z	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**1**

Figure 4.2. Matrice de correspondance

Un élément de la matrice M [i, j] = 1 si le nœud j contient la réplique i et 0 sinon.

L'approche proposée pour le maintien de la cohérence utilise une combinaison entre le protocole à quorum structuré en arbre et le modèle de cohérence à l'entrée Un quorum est l'ensemble minimum de nœuds qui contiennent une réplique de donnée, nécessaires pour une opération de lecture ou écriture. Une coterie est l'ensemble de tous les quorums possibles. Le modèle de cohérence à l'entrée est basé sur l'idée qu'un utilisateur demandant une opération de lecture ou d'écriture doit acquérir un verrouillage de la partie de la donnée concernée et

relâcher le verrou à la fin de l'opération. Pour chaque donnée du système on a une coterie (Figure 4.3)

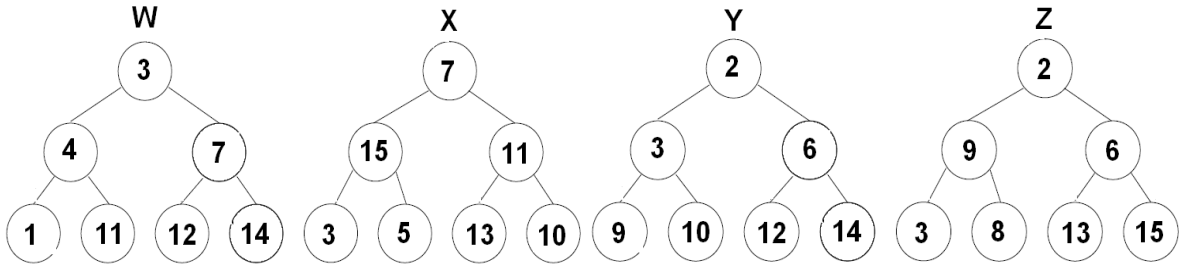


Figure 4.3. Coteries structurées en arbre

Un nœud est identifié par un numéro et une priorité, Chaque réplique sur un nœud est identifiée par une liste de versions (maximum 03 versions pour des raisons de simplification).

Une version est une valeur de réplique identifiée par un quadruplé (N, P, E, V)

N : le numéro du nœud qui a effectué la dernière modification sur la version.

P : la priorité du nœud N.

E : l'estampille qui représente le temps de la dernière modification de la version.

V : la valeur de la version.

Exemple : Supposons une donnée X de type chaîne de caractères. Une réplique de X sur un nœud est représentée par une liste de versions, comme par exemple : {(6, 1, 5, ax), (2, 5, 9, xd), (13, 7, 8, yc)}
La priorité est un paramètre dynamique lié aux différents nœuds de la grille. Il reflète la fiabilité des nœuds, il est calculé en fonction de plusieurs paramètres, comme le nombre d'accès en lecture et/ou en écriture à un nœud, la puissance de calcul, l'autonomie du nœud, la représentation du nœud dans un groupe de travail, la moyenne des pannes, … Plus la valeur de ce paramètre est élevée moins le nœud sera prioritaire.
La valeur de la version désigne le contenu de la réplique pour la version considérée, donc une réplique peut avoir jusqu'à trois valeurs différentes à un moment donné.

On propose de définir des états d'un nœud : Libre (L), Occupé (O) et Bloqué (B) comme il est illustré dans la figure 4.4.

Un nœud est libre si toutes les versions qu'il contient sont libres.

Un nœud est occupé s'il contient au moins une version verrouillée.

Un nœud est bloqué si toutes les versions qu'il contient sont verrouillées.

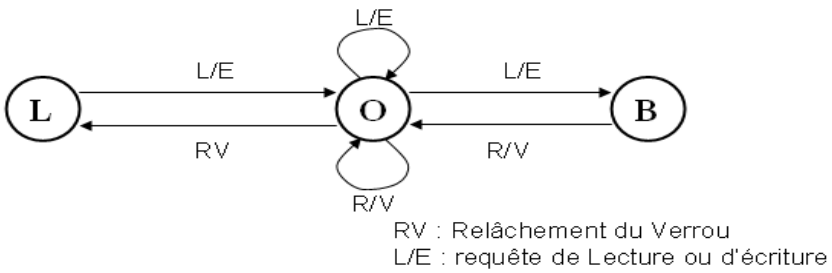


Figure 4.4. Etats d'un nœud d'une coterie

Supposons que l'état initial d'un nœud est libre (L).

Si une requête est adressée à ce nœud, la version choisie pour effectuer l'opération est verrouillée et le nœud passe à l'état occupé (O). Si après cela ce nœud reçoit une autre requête, soit il reste dans le même état s'il existe encore des versions libres de la réplique, soit il passe à l'état bloqué (B).

Si le nœud est à l'état bloqué et une des versions vient d'être libérée il repasse à l'état Occupé, si d'autres versions sont libérées, soit il reste dans le même état s'il reste encore des versions verrouillées, soit il repasse à nouveau à l'état libre si toutes les verrous de toutes les versions de la réplique sont relâchées (toutes les versions sont libres).

NB : l'état bloqué est le seul état où un nœud ne peut accepter des requêtes de lecture ou d'écriture.

Nous présentons dans La figure 4.5 un exemple de coterie relative à la donnée W (Figure 4.3), sur lequel nous illustrons notre approche. Cette coterie est constituée de sept nœuds numérotés. Chaque nœud contient une réplique de la donnée concernée (supposons que c'est une chaîne de caractères), elle est identifiée par une liste de trois versions.

Par exemple, dans le nœud 3, la réplique est identifiée par l'ensemble de versions : {(6, 1, 5, ax), (2, 5, 9, xd), (13, 7, 8, yc)}. La première version (6, 1, 5, ax) a été modifiée par le nœud 6 qui a une priorité égale à 1, au temps t=5, et la valeur de cette version est ‘’ax’’.

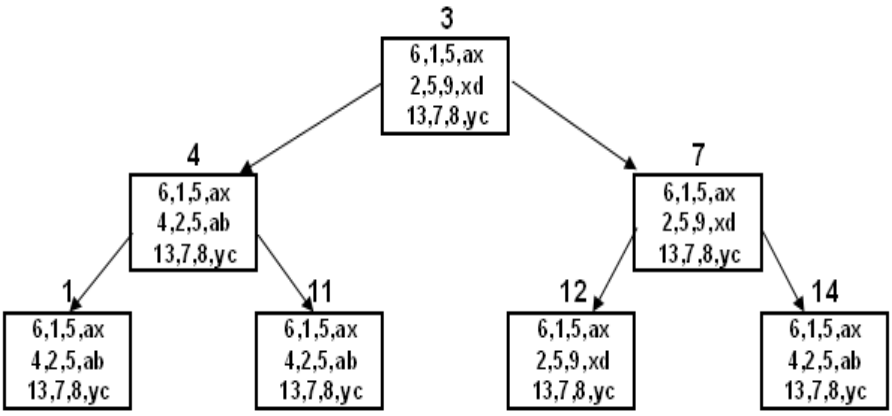


Figure 4.5. Modèle de coterie pour la donnée W

3. Les Services utilisés pour assurer la cohérence :

La cohérence dans l'approche qu'on propose est assurée par trois principaux services : Le service de lecture / écriture, le service de détection et résolution des conflits et le service de réconciliation et convergence des copies, qui doivent fonctionner en coopération de tel sorte à préserver la convergence des répliques distribuées sur l'ensemble des nœuds de la grille et d'assurer qu'une lecture retourne la version la plus récente qui a été écrite.

3.1. Le service de lecture / écriture :

Ce service est chargé de gérer les opérations (lectures, écritures et propagations des mises à jour) émises par les différentes requêtes du système, il est basé sur l'utilisation de deux principaux algorithmes, l'algorithme d'écriture et l'algorithme de lecture.

3.1.1. Algorithme d'écriture :

L'algorithme d'écriture permet à un nœud de la grille d'effectuer une mise à jour sur une donnée. Il est exécuté sur chaque nœud qui demande une mise à jour.

Algorithme d'écriture

```
Paramètres d'entrée : N (Numéro du nœud), D (identificateur de la donnée)
DEBUT
01 : choix = Faux
02 : Contacter la coterie correspondante à la donnée D
03 : TANT QUE (choix == Faux)
04 :     SI (il existe un nœud libre) ALORS
05 :         choix = Juste
06 :         Choisir ce nœud libre
07 :     SINON
08 :         SI (il existe un nœud occupé) ALORS
09 :             choix = Juste
10 :             Choisir ce nœud occupé
11 :         FIN SI
12 :       FIN SI
13 : FIN TANT QUE
14 : Désigner un quorum contenant le nœud choisi
15 : Sur le nœud choisi Lancer l'algorithme de choix de la  version la plus  ancienne de D
       /*décrit dans la section 3.2.1*/
16 : Verrouiller la version choisie en écriture
17 : Effectuer l'écriture sur la version choisie par le nœud N
18 : Relâcher le verrou en écriture
19 : POUR (chaque nœud du quorum) FAIRE
         Exécuter l'algorithme de propagation
FIN
```

Exemple d'écriture : Supposons qu'au temps t=9 le nœud 2 dans le réseau a émis une requête d'écriture de la donnée « w » pour lui affecter la valeur « xd » (Figure 4.6). La coterie correspondante à w est contactée, et le quorum {3, 7, 12} est désigné car il contient le nœud 7 qui est libre. On va donc effectuer l'opération, en appliquant l'algorithme du choix de la version la plus ancienne. Dans l'exemple c'est la version (4, 2, 5, ab), on la verrouille en écriture, on effectue l'écriture, on relâche le verrou en écriture et on propage la mise à jour aux nœuds du quorum 3 et 12.

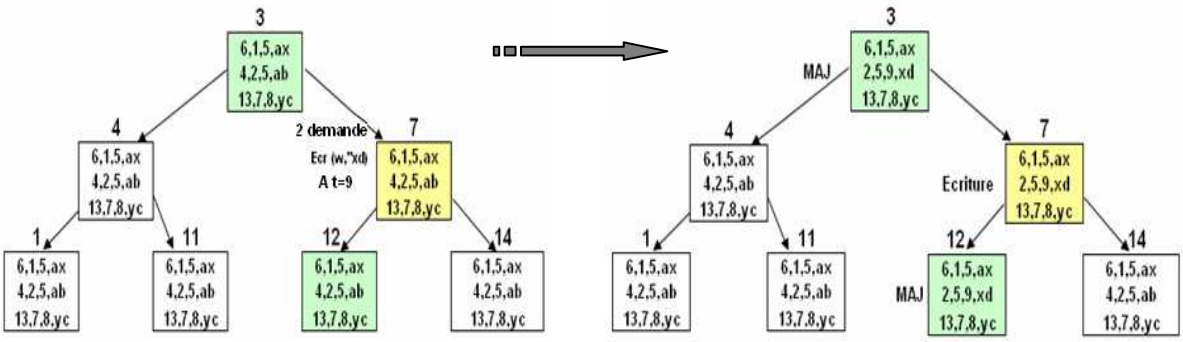


Figure 4.6. Exemple d'écriture

3.1.2. Algorithme de Lecture :

L'algorithme de lecture permet à un nœud de la grille d'effectuer une lecture d'une donnée en retournant au nœud demandeur la version la plus récente de cette donnée. Il est exécuté sur chaque nœud qui demande une lecture.

NB : même en lecture, lorsqu'on choisi la version la plus récente, si il ya divergence alors l'algorithme se charge de propager la version la plus récente aux nœuds du quorum choisi.

Algorithme de lecture

Paramètres d'entrée : N (Numéro du nœud), D (identificateur de la donnée)
DEBUT
01 : choix = Faux
02 : Contacter la coterie correspondante à la donnée D
03 : **TANT QUE** (choix == Faux)
04 : **SI** (il existe un nœud libre) **ALORS**
05 : choix = Juste
06 : Choisir ce nœud libre
07 : **SINON**
08 : **SI** (il existe un nœud occupé) **ALORS**
09 : choix = Juste
10 : Choisir ce nœud occupé
11 : **FIN SI**
12 : **FIN SI**
13 : **FIN TANT QUE**
14 : Désigner un quorum contenant le nœud choisi
15 : **POUR** (chaque nœud du quorum) **FAIRE**
16 : Lancer l'algorithme de choix de la version la plus récente
/*décrit dans la section 3.2.2*/
17 : **FIN POUR**
18 : **POUR** (chaque nœud du quorum) **FAIRE**
19 : **SI** (il y divergence avec la version choisie) **ALORS**
20 : Exécuter l'algorithme de propagation
21 **FIN SI**
22 : **FIN POUR**
23 : Verrouiller la version choisie en lecture

24 : Effectuer la lecture de la version choisie par le nœud N
25 : Relâcher le verrou en écriture
FIN

Exemple de lecture : Supposons qu'à un moment donné un nœud dans le réseau a émis une requête de lecture de la donnée « w » (Figure 4.7). La coterie correspondante à w est contactée et le quorum {3, 4, 1} qui contient le nœud 4 qui est libre est désigné. On effectue la lecture, en appliquant l'algorithme du choix de la version la plus récente. Dans l'exemple c'est la version (2, 5, 9, xd). Comme il y a divergence (cette version ne figure pas dans les nœuds 4 et 1), on doit la propager vers les nœuds qui ne la contiennent pas par l'algorithme d'écriture de propagation. On verrouille donc cette version en lecture, on effectue la lecture, on relâche le verrou en lecture et on retourne la valeur lue au nœud qui l'a demandé.

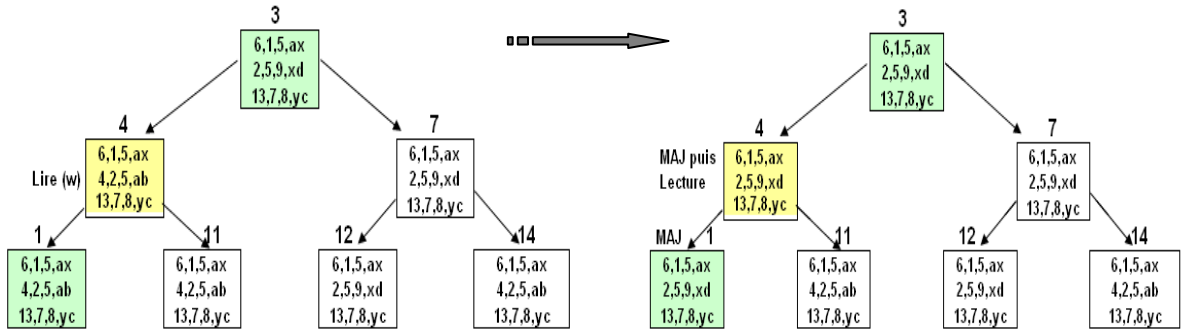


Figure 4.7. Exemple de lecture

3.1.3. Algorithme de propagation :

Il est exécuté sur chaque nœud qui a reçu la version propagée (la plus récente)**.**

Algorithme de propagation
Paramètres d'entrée : V (La version Propagée) **DEBUT** 01 : **POUR** (chaque Version de D) **FAIRE** 02 : **SI** (Version est libre) **ALORS** 03 : Version = V 05 : **SINON** 06 : **SI** (Version est verrouillée en lecture) **ALORS** 07 : Attendre la libération de Version 08 : Version = V 09 : **SINON** 10 : Ignorer l'opération de propagation 11 : **FIN SI** 12 : **FIN SI** **FIN**

3.2. Le service de détection et de résolution des conflits :

Les conflits rencontrés dans notre service sont dus d'une part à des propagations des mises à jour concurrentes, et aussi au choix de la version la plus ancienne à écraser lors d'une écriture, ou au choix de la version la plus récente à récupérer lors d'une lecture. Dans la règle de THOMAS qui exige que le dernier qui écrit gagne.

Si on est dans des systèmes qui traitent des données considérées comme importantes, qui sont modifiées par des nœuds ou processus privilégiés, la règle de THOMAS s'avère inutile. Pour cela on a introduit la notion de priorité pour chaque nœud. Cette priorité est attribuée automatiquement à chaque nœud selon un critère donnée comme le groupe de travail par exemple.

On a utilisé un paramètre supplémentaire : « La marge de fraicheur » qui est un paramètre fixé par le système. Ce paramètre permet de contrôler le choix de la version la plus ancienne à écraser lors de l'écriture ou lors de mises à jour concurrentes, ou la version la plus récente à récupérer lors de la lecture. Si on trouve, par exemple, deux versions la première avec un estampille grand mais une priorité faible, la deuxième version avec un estampille petit mais une priorité élevée, et si on est dans le cas d'écriture et on choisi directement la première à écraser (en favorisant la priorité), on risque que la prochaine lecture ne retourne pas la version la plus récente de la donnée concernée. Si on choisi directement la deuxième à écraser (en se basant sur la règle de THOMAS) on risque de perdre une donnée importante.

Le même principe est appliqué pour le cas où on veut effectuer une lecture et on veut choisir la version la plus récente de la donnée.

La marge de fraicheur permet de résoudre ce problème, comme il est mentionné dans les algorithmes 3.2.1 et 3.2.2.

Les algorithmes qu'on a proposés pour gérer ces cas de conflits sont : l'algorithme de choix de la version la plus ancienne et l'algorithme de choix de la version la plus récente.

3.2.1. Algorithme de choix de la version la plus ancienne :

Cet algorithme est exécuté lors d'une mise à jour, pour choisir la version à écraser, ou bien dans le cas où un nœud reçoit deux mises à jour concurrentes.

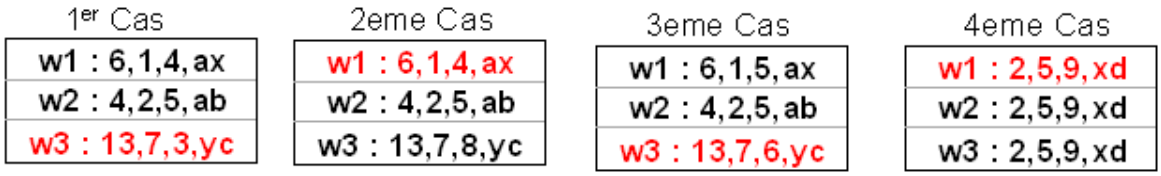
Algorithme de choix de la version la plus ancienne

```
Paramètres d'entrée : D (identificateur de la donnée)
DEBUT
    POUR (chaque version de D)
        Choisir V1 dont la priorité est la plus faible
        SI (il existe plusieurs versions avec la même priorité) ALORS
            Choisir parmi ces versions celle avec l'estampille le plus petit
        FIN SI
        Choisir V2 dont l'estampille est le plus petit
        SI (il existe plusieurs versions avec la même estampille) ALORS
            Choisir parmi ces versions celle avec la priorité la plus faible
        FIN SI
        SI (V1 == V2) ALORS
            Retourner V1 (ou V2)
        SINON
          SI (V1.Estampille - V2.Estampille > Marge de Fraicheur) ALORS
```

```
            Retourner V2
        SINON
            Retourner V1
        FIN SI
    FIN SI
FIN
```

Exemple :

Si on considère par exemple une copie de la réplique (w) sur le nœud 1 et la marge de fraîcheur égale à trois, alors la version choisie pour être remplacée est définie pour chaque cas en suivant l'algorithme précédent :



Dans le premier cas V1=w3 et V2=w3, on a donc V1=V2, par conséquent w3 est la version à remplacer.

Dans le 2eme cas on a V1=w3 et V2=w1 et on a Estampille (V1) - Estampille (V2) > 3 donc w1 sera remplacée.

Dans le 3eme cas on a V1=w3 et V2=w2 et on a Estampille (V1) - Estampille (V2) < 3 donc w3 sera remplacée.

Dans le 4eme cas toute les versions sont identiques, w1 est choisie aléatoirement pour être remplacée.

3.2.2. Algorithme de choix de la version la plus récente :

Cet algorithme est lancé sur chaque nœud du quorum lors de la lecture afin de retourner la version la plus récente de la donnée à lire.

Algorithme de choix de la version la plus récente

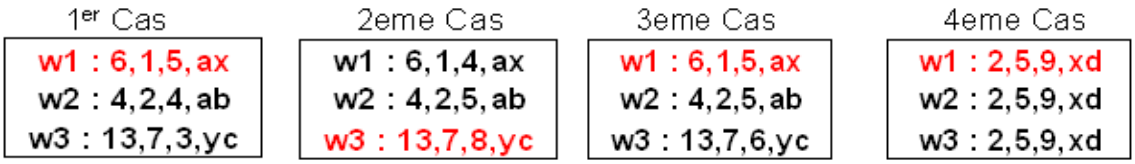
```
Paramètres d'entrée : D (identificateur de la donnée)
DEBUT
    POUR (chaque version de D)
        Choisir V1 dont la priorité est la plus élevée
        SI (il existe plusieurs versions avec la même priorité) ALORS
            Choisir parmi ces versions celle avec l'estampille le plus grand
        FIN SI
        Choisir V2 dont l'estampille est le plus grand
        SI (il existe plusieurs versions avec la même estampille) ALORS
            Choisir parmi ces versions celle avec la priorité la plus élevée
        FIN SI
        SI (V1 == V2) ALORS
            Retourner V1 (ou V2)
        SINON
```

```
            SI (V2.Estampille – V1.Estampille > Marge de Fraicheur) ALORS
                Retourner V2
            SINON
                Retourner V1
            FIN SI
          FIN SI
FIN
```

Remarque : cet algorithme est valable pour le cas de lecture, lorsqu'on doit récupérer la version la plus récente pour effectuer la lecture.

Exemple :

Si on considère par exemple un nœud recevant trois mises à jours en même temps et il qu'il est obligé de choisir une seule parmi ces trois versions. On suppose que la marge de fraîcheur égale à trois (03), la version qui sera donc choisie comme étant la plus récente est définie pour chaque cas en suivant l'algorithme précédent.



Dans le premier cas V1=w1 et V2=w1, on a donc V1=V2, par conséquent w1 est la version la plus récente.

Dans le 2eme cas on a V1=w1 et V2=w3 et on a Estampille (V2) - Estampille (V1) > 3 donc w3 est la version la plus récente.

Dans le 3eme cas on a V1=w1 et V2=w3 et on a Estampille (V2) - Estampille (V1) < 3 donc w1 est la version la plus récente.

Dans le 4eme cas toute les versions sont identiques, w1 est choisie aléatoirement comme la version la plus récente.

3.3. Le service de réconciliation et convergence des répliques :

L'approche proposée ne garantie pas d'une manière rigoureuse qu'à un moment donné sur chaque nœud toutes les versions d'une réplique soient identiques, on doit donc assurer la convergence des versions vers la même copie du même nœud, et aussi les mêmes répliques sur tous les nœuds de la grille.
Le mécanisme de réconciliation permet donc d'assurer l'« intra-cohérence » sur chaque nœud et l'« inter-cohérence » entre les différents nœuds de la grille.
Ce mécanisme est exécuté en deux parties :

- La première exige que le système soit au repos afin de relever la version la plus récente parmi toutes les répliques sur tous les nœuds de la coterie, et la propager vers les autres nœuds.
- La deuxième sur chaque nœud, elle exige que le nœud en question soit libre, afin de converger les versions de la réplique qu'il la contient vers la version la plus récente.

NB : Les deux parties peuvent être exécutées de façon indépendante.

4. Conclusion :

On a vu dans ce chapitre les principaux idées et concepts de l'approche proposée pour la gestion du problème de cohérence des répliques sur les grilles de calcul, ainsi que les mécanismes et les algorithmes qui représentent les différents aspects de cette approche.
Dans le chapitre suivant on va mettre en place ces mécanismes à l'aide d'un simulateur de grilles de calcul, on étudiera le comportement de l'approche par différents essais.

Chapitre 4 : Implémentation et mise en œuvre de l'approche

1. Introduction :

Nous avons présenté dans les chapitres précédents les notions de base concernant la réplication et la cohérence des répliques sur les grilles. Nous avons vu aussi quelques travaux liés au problème de maintien de la cohérence des répliques sur les grilles. L'objectif de notre projet était de trouver une approche qui permet de maintenir la cohérence des répliques sur les grilles tout en prenant en compte les caractéristiques fondamentales des grilles comme la volatilité et le passage à l'échelle. L'idée de base de l'approche que nous avons proposée pour le problème de cohérence sur les grilles est inspirée des travaux [LOI 06], [IVA 05] et de [MED 06]. Nous avons combiné le protocole de cohérence à quorum avec le modèle de cohérence à l'entrée et nous avons étendu la règle de THOMAS, en se basant sur les notions de priorité et de versions de données pour gérer les mises à jour concurrentes.

Comme les grilles sont des systèmes distribués à grande échelle, avec un nombre de nœuds très importants et une zone géographique distante, l'implémentation de l'approche sur une grille réelle s'avère difficile, pour cela nous avons utilisé un simulateur de grille [Annexe 2] afin de tester et implémenter l'approche que nous avons proposé.
Nous allons présenter dans ce chapitre les démarches de la réalisation, la mise en œuvre de notre approche et les résultats obtenus.

2. Le simulateur GRIDSIM :

GRIDSIM est un des simulateurs les plus utilisés dans les expérimentations de test des algorithmes sur les systèmes distribués et les grilles de calculs [ROB 06], nous l'avons choisi pour notre projet à cause de plusieurs facteurs. Parmi ces facteurs on peut citer [ROB 06]:

- C'est un simulateur à événements discrets, ce qui permet l'envoi et la réception des messages entre les entités constituants la grille ou le système étudiés.
- Il n'est pas destiné seulement pour étudier les grilles, il est utilisé sur les systèmes utilisant les notions de producteurs et consommateurs.
- Il utilise la notion d'entité. Les entités peuvent être des processeurs, des utilisateurs, des ressources, des informations …etc.
- Il offre la possibilité de créer des réseaux en utilisant les différentes entités et de tester de façon efficace et flexible leurs comportements.
- Il simplifie l'implémentation des algorithmes proposés sur les modèles de grilles.
- Il est basé sur un langage de programmation à usage général celui de JAVA, ce qui assure la portabilité des applications.
- Il fournit des modèles des composants réels des réseaux tels que les routeurs et les nœuds.
- Il permet la modélisation et la simulation des ressources et des applications d'ordonnancement dans des environnements distribués et parallèles.

3. L'environnement de développement et le modèle proposé :

3.1. L'environnement de développement :

Nous avons choisi pour développer notre approche un PC équipé par un processeur pentium 4 CPU 3Ghz, et 2Go de RAM, avec un système d'exploitation XP Pro, et l'outil de développement NetBeans V6.1. Nous avons utilisé la version 5.2 de GRIDSIM.

3.2. Le modèle proposé :

Le modèle choisi est une organisation hiérarchique des nœuds en se basant sur les paramètres fournis par le simulateur GRIDSIM. Les nœuds, qui sont des entités dans le simulateur GRIDSIM, sont interconnectés par des routeurs sous forme hiérarchique, dont le but est seulement de diminuer le cout des échanges dans le réseau, mais il faut noter que cette organisation est totalement indépendante de l'approche que nous avons proposée. L'approche proposée est valable pour n'importe quelle autre organisation.

L'initialisation du système est donnée par les paramètres d'entrée suivants:

- N : le nombre de nœuds de la grille.
- D : le nombre de données utilisées sur le système.
- V : le nombre de versions autorisées.
- NBROP : le nombre des opérations de lectures/écritures qui seront effectuées par les nœuds durant la simulation.
- Ta : la taille de chaque donnée.

NB : dans notre projet nous avons pris une taille identique pour toutes les données du système.

En résultats on obtient les paramètres de sortie suivants :

- Nbr Lecture : le nombre de lectures effectuées par nœud.
- Nbr Ecriture : le nombre d'écritures effectuées par nœud.
- Attente : le délai d'attente moyen par nœud. Il représente le délai d'attente des requêtes de lectures/écritures lorsqu'elles sont mises en attente.
- Temps de Simulation : représente le temps total entre le début et la fin de simulation.
- L'espace de stockage utilisé : représente l'espace de stockage utilisé sur l'ensemble des nœuds de la grille.
- Taux de cohérence : représente le pourcentage des répliques récentes par rapport à l'ensemble de toutes les répliques d'une certaine donnée. Une réplique est récente si elle est identique à celle qui a été dernièrement écrite. C'est-à-dire :
  Taux de cohérence de D = (NBR des répliques récentes de D / NBR total des répliques de D)*100.

Chaque nœud du système peut contenir plusieurs répliques de données utilisées dans le système, les données sont répliquées aléatoirement sur les différents nœuds.

La simulation de l'approche passe par trois principales phases :

Première Phase : c'est la phase d'initialisation. Au lancement du système GRIDSIM se charge de la création de la grille et des entités qui la conçoivent, ainsi que l'interconnexion entre elles, puis le système effectue une réplication aléatoire des données sur les nœuds, ainsi que la duplication des versions sur chaque nœud.

Deuxième Phase : c'est la phase d'exécution. Durant la simulation, chaque nœud du système effectue des opérations de lecture/écriture jusqu'à atteindre le nombre NBROP, tous les services de l'approche proposée sont exécutés dans cette phase (Le service de lecture / écriture, le service détection et résolution des conflits et le service de réconciliation et convergence des copies) définis dans le chapitre précédent.

Troisième Phase : c'est la phase d'affichage des résultats. Dans cette phase les paramètres de sortie sont affichés pour montrer à l'utilisateur les résultats de la simulation.

4. Expérimentations :

Afin de tester l'approche que nous avons proposée, nous avons réalisé plusieurs expérimentations :

4.1. Expérimentation 01 : Le délai d'attente

La première expérimentation concerne le test du délai d'attente globale du système, ce délai est égale à la somme de tous les délais d'attente de chaque nœud de la grille.
D'après plusieurs essais nous avons constaté que le délai d'attente dépend du nombre de nœuds de la grille, du nombre de versions autorisées ainsi que du nombre d'opérations effectuées par le système. Pour cela nous avons effectué deux tests et dans tous les deux nous avons fixé le nombre de données du système à 10 données avec une réplication aléatoire.
Dans le premier test nous avons utilisé un nombre de nœuds égale à 50 (Figure 4.1 et 4.3). Dans le deuxième un nombre de nœuds choisi égale à 150 (Figure 4.2 et 4.4). Nous avons changé dans chacun le nombre d'opérations à effectuer (Nbr OP) ainsi que le nombre de versions autorisées (Nbr VR).
Les tableaux suivants montrent le comportement de notre approche en variant les différents paramètres considérés.

Nbr OP / Nbr VR	50	150	300	500	800	1000	1500
1	31	203	421	723	1181	1481	2276
2	1	82	187	354	577	721	1089
3	0	48	127	224	379	463	730
5	0	17	66	126	219	275	423

Figure 4.1. Délai d'attente (us) avec le nombre de nœuds = 50 nœuds

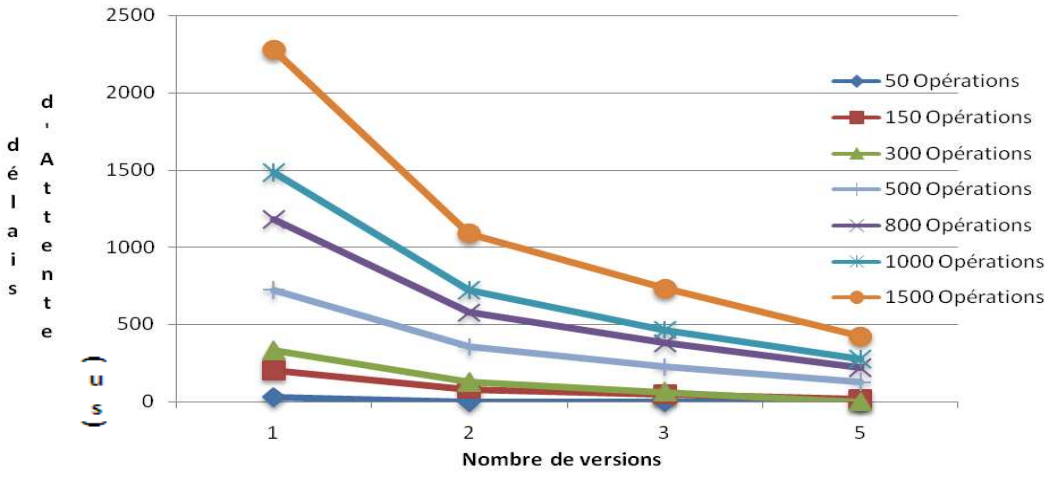


Figure 4.2. Diagramme relatif au Délai d'attente avec Nbr Nods = 50

Nbr OP / Nbr VR	150	300	500	800	1000	1500
1	96	332	580	1035	1272	2180
2	1	129	241	469	598	1032
3	0	64	145	294	373	688
5	0	1	49	148	194	386

Figure 4.3. Délai d'attente (us) avec le nombre de nœuds = 150

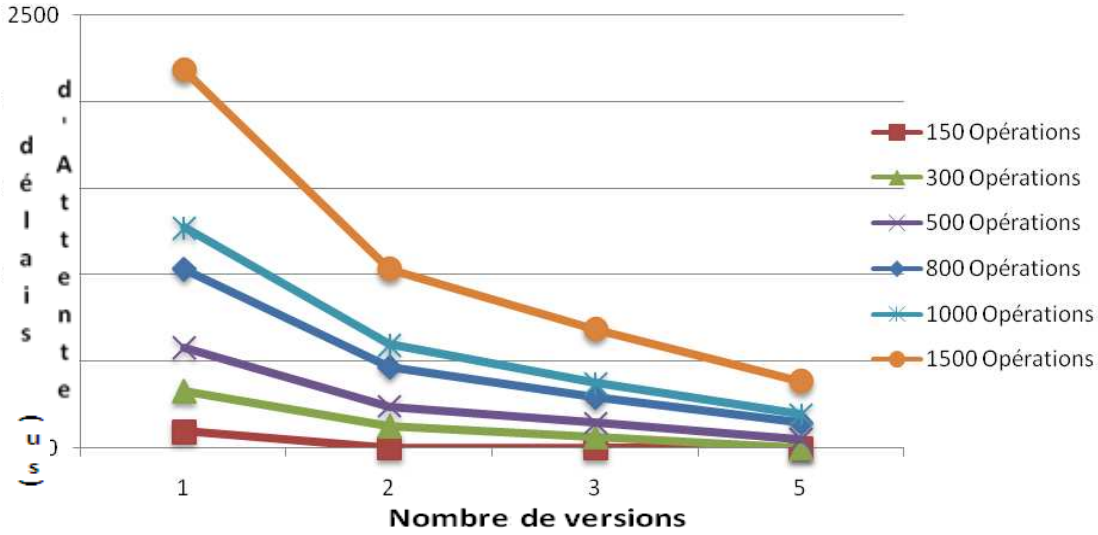


Figure 4.4. Diagramme relatif au Délai d'attente avec Nbr Nods = 150

D'après les résultats obtenus des deux tests précédents on remarque que le délai d'attente diminue avec l'augmentation du nombre de versions autorisées, mais il augmente avec l'augmentation du nombre d'opérations effectuées. Comme on remarque aussi une légère diminution avec l'augmentation du nombre de nœuds de la grille, l'explication de ces résultats est que l'augmentation du nombre de versions donne plus de possibilité aux nœuds d'effectuer les opérations lectures/écriture sans attendre en prenant des versions non verrouillées.

4.2. Expérimentation 02 : Le taux de cohérence

Dans la deuxième expérimentation nous testons le taux de cohérence globale du système, c'est le taux moyen des taux de cohérence de chaque donnée.
Les essais que nous avons effectués montrent que le taux de cohérence dépend du nombre de nœuds de la grille.
Nous avons effectué un seul test pour étudier le comportement de l'approche concernant ce paramètre, nous avons donc fixé le nombre de données du système à 10 et le nombre d'opération à effectuer par le système à 1500 opérations, et nous avons fait varier à chaque

fois, le nombre de nœuds (Nbr Nods) ainsi que le nombre de versions autorisées (Nbr VR), nous avons obtenus les résultats montrés dans les figures 4.5 et 4.6.

Nbr Nods / Nbr VR	10	50	100	200	500	800	1000	1500
1	58%	28%	17%	11%	5%	3%	2%	1%
2	63%	27%	17%	10%	4%	3%	2%	1%
3	65%	28%	19%	10%	4%	3%	2%	1%
5	60%	27%	18%	11%	5%	3%	2%	1%

Figure 4.5. Variation du taux de cohérence

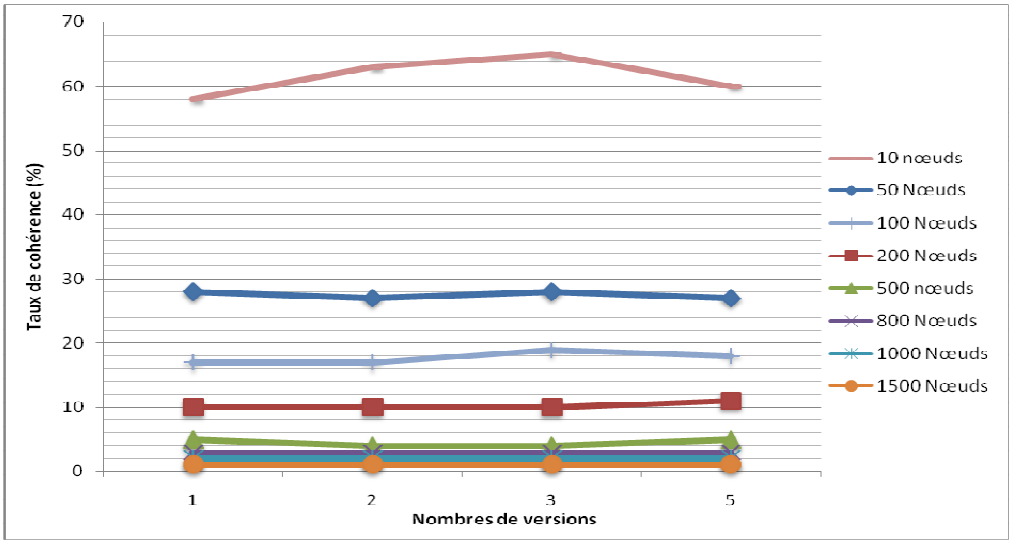


Figure 4.6. Diagramme relatif au test du taux de cohérence

Les résultats obtenus de ce test montrent que le taux de cohérence est lié directement au nombre de nœuds de la grille, plus le nombre de nœuds de la grille est élevé plus le taux de cohérence est bas, ceci est expliqué par le nombre de répliques total du système qui augmente avec l'augmentation du nombre de nœuds de la grille.

4.3. Expérimentation 03 : L'espace de stockage

Dans la troisième expérimentation nous avons testé un paramètre que nous avons considéré comme essentiel, celui de l'espace de stockage.
L'espace de stockage globale est égale à la somme des espaces de stockage utilisés sur chaque nœud.

Les essais montrent que ce paramètre est lié au nombre de données du système, la taille de ces données ainsi que le nombre de versions.
Pour notre test nous avons fixé le nombre de nœuds à 500 et le nombre d'opérations à effectuer à 1000. Nous avons effectué deux tests. Dans le premier test nous avons utilisé une taille identique pour toutes les données du système égale à 100 Mo (Figure 4.7 et 4.8). Dans le

deuxième une taille de données égale à 500 Mo (Figure 4.9 et 4.10). Nous avons changé dans chacun le nombre de données (Nbr DS) ainsi que le nombre de versions autorisées (Nbr VR).

Nbr DS / Nbr VR	10	20	50	100	500
1	30	54	139	270	1204
2	52	99	241	556	2755
3	83	160	401	791	3529
5	155	234	586	1132	5088

Figure 4.7. L'espace de stockage (Go) avec taille de donnée = 100Mo

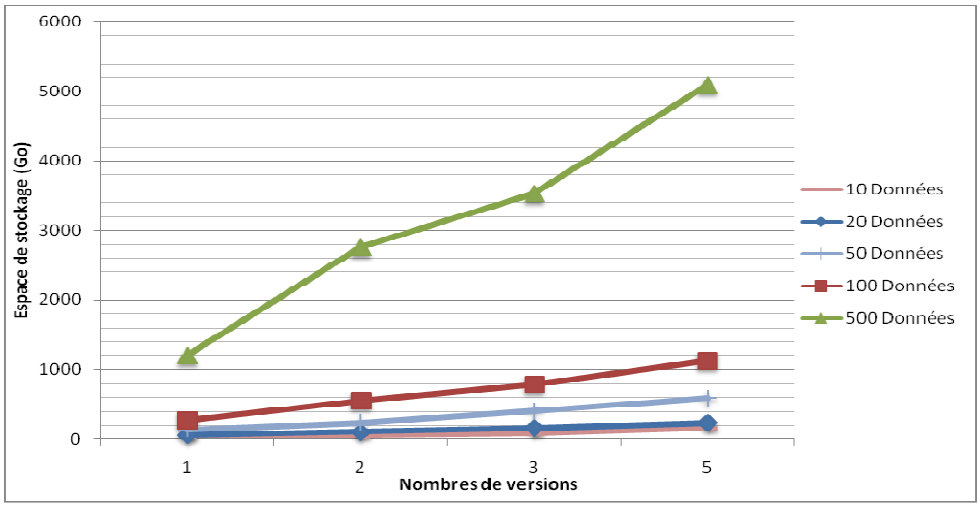


Figure 4.8. Diagramme relatif à l'espace de stockage avec taille de donnée = 100Mo

Nbr DS / Nbr VR	10	20	50	100	500
1	154	302	634	1298	6601
2	276	545	1167	2719	11805
3	451	786	2046	4014	15214
5	755	1232	3690	5987	32360

Figure 4.9. L'espace de stockage (Go) avec taille de donnée = 500Mo

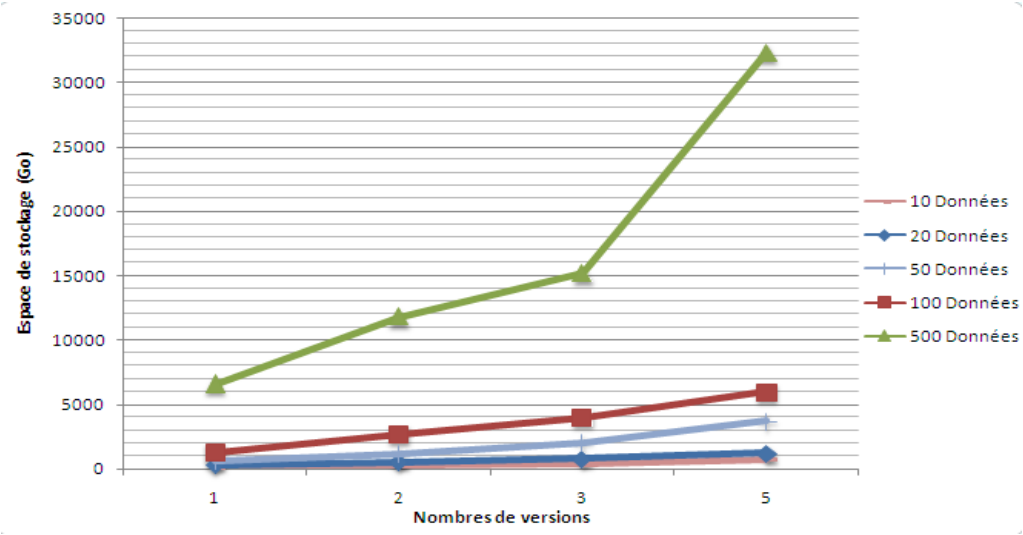


Figure 4.8. Diagramme relatif à l'espace de stockage avec taille de donnée = 100Mo

Nous avons vu dans la première expérimentation que l'augmentation du nombre de versions peut nous aider à maitriser le délais d'attente des opérations, donc à bien gérer la cohérence, en revanche les résultats obtenus de ce test ne sont pas satisfaisants en ce qui concerne les performances du système, l'espace de stockage augmente d'une façon très importante avec l'augmentation du nombre de versions et le nombre des données du système, ce qui peut dégrader le temps de réponse de ce dernier et donc ses performances.

4.4. Expérimentation 04 : Le temps de simulation

Dans cette expérimentation nous avons testé le comportement de l'approche vis-à-vis des performances du système en ce qui concerne le temps de simulation. Il faut noter que le temps de simulation du système est égal à la somme de plusieurs paramètres. Parmi ces paramètres nous avons choisis les plus importants : le délai d'attente des requêtes (que nous avons exprimé dans la première expérimentation), le temps de récolte qui est le temps nécessaire pour contacter la coterie correspondante à la donnée qu'on veut lire ou écrire, pour rechercher un nœud libre ou occupé et pour rechercher la version la plus récente ou la plus ancienne, le temps de propagation qui est le temps nécessaire pour propager la version récente sur un nœud vers les autres nœuds.
Pour ce test nous avons fixé le nombre de nœuds à 10, le nombre de données à 10 et le nombre d'opérations à 300. Nous avons fait varier le nombre de versions (Nbr VR) et à chaque fois nous avons mesuré le temps de récolte (TR), le temps de propagation TP ainsi que le temps de simulation TS. Les résultats obtenus sont montré dans les figures : 4.9 et 4.10.

Nbr VR / TEMPS	1	2	3	5
TR	511	1056	1634	2605
TP	2122	4424	6099	11779
TS	3059	5709	7882	14469

Figure 4.9. Variation du temps de simulation

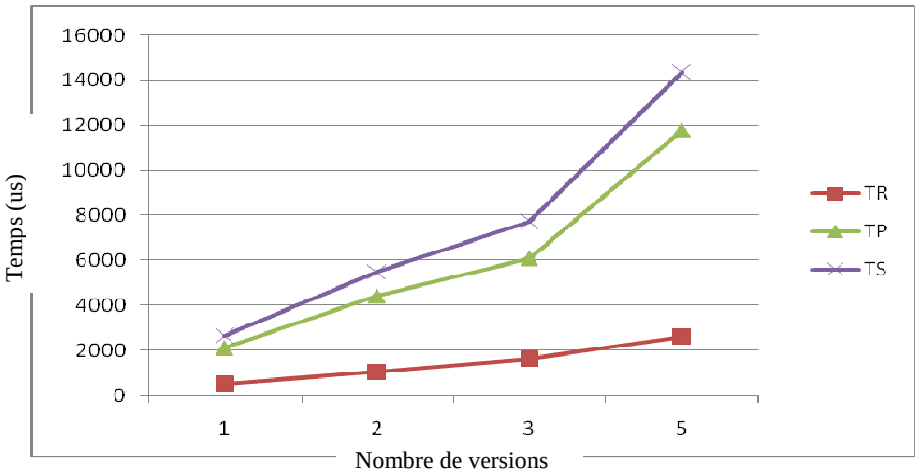


Figure 4.10. Diagramme relatif au test du Temps de simulation

Nous avons constaté dans cette expérimentation que l'utilisation d'un nombre important de versions augmente considérablement le temps de récolte ainsi que le temps de propagation ce qui augment en conséquence le temps de simulation, ce que nous pouvons dire que l'augmentation du nombre de versions autorisées ne permet pas d'obtenir des résultats satisfaisants en ce qui concerne le temps de simulation est donc ce n'ai pas à la faveur des performances du système.

5. Conclusion :

Dans ce chapitre nous avons fait plusieurs expérimentations pour évaluer notre approche. Nous avons pu voir l'influence des différents paramètres proposés sur le comportement de l'approche. D'après les résultats obtenus, nous avons pu constater que les paramètres utilisés dans notre approche ont donné une vision générale sur le comportement de l'approche. Si l'augmentation du nombre de versions a permis d'une part à diminuer le délai d'attente des requêtes, d'autre part il a augmenté considérablement le temps de simulation ainsi que l'espace de stockage du système, ce qui affecte négativement sur les performances du système. Nous avons vu que l'approche peut supporter un nombre important de nœuds et de données et elle fonctionne parfaitement, mais comme nous avons vu, le taux de cohérence diminue dans ces conditions.

L'avantage de notre approche est qu'elle permet le contrôle de cohérence en ce qui concerne le délai d'attente et le taux de cohérence en variant les paramètres nécessaires. Mais en revanche si on utilise un nombre de données et/ou un nombre de versions élevés on risque d'avoir un taux de cohérence réduit et de diminuer les performances du système avec l'augmentation de l'espace de stockage ce qui peut influencer négativement sur le temps de réponse du système.

Conclusion générale et perspectives :

Nous avons étudié à travers ce document le problème de cohérence des données répliquées sur les grilles. Après avoir introduit des notions fondamentales concernant la réplication et la cohérence sur les systèmes distribués, nous avons cité quelques travaux liés au problème étudié, où nous avons pris de chacun des idées que nous avons utilisées par la suite pour élaborer l'approche que nous avons proposé pour le problème de cohérence des répliques sur les grilles.

Le principe de base de l'approche proposée est de combiner le protocole de cohérence à quorum avec un modèle de cohérence relâché, et d'étendre la règle de THOMAS avec l'introduction de nouveaux paramètres tels que la priorité des nœuds et la marge de fraicheur afin de donner plus de contrôle sur les mises à jour des répliques à propager lors de conflits.

A l'aide du simulateur GRIDSIM nous avons implémenté notre approche sur un modèle de grille que nous avons choisi, les résultats obtenus montrent l'influence des paramètres choisis sur le comportement de l'approche concernant le délai d'attente des requêtes ainsi que le taux de cohérence.

Les résultats obtenus sont satisfaisants et montrent l'avantage de notre approche en ce qui concerne le contrôle de cohérence par la réduction du délai d'attente des requêtes et l'augmentation du taux de cohérence en utilisant des paramètres avec des nombres moyens de nœuds de la grille et de données du système . Mais nous avons constaté que si on utilise un nombre important des nœuds de la grilles ou/et un nombre important des données utilisées par le système, ça diminue les performances de l'approche avec la réduction du taux de cohérence et l'augmentation de l'espace de stockage et le temps de simulation, ce qui influence négativement sur le temps de réponse du système.

Comme perspectives pour les futurs travaux Nous proposons d'améliorer l'approche par un service de gestion de la dynamicité, parce qu'une grille est caractérisée par la dynamicité où des nœuds peuvent quitter le réseau et des nouveaux peuvent le rejoindre à n'importe quel moment. Cette idée s'avère intéressante si nous voulons implémenter l'approche sur une grille réelle.

Dans notre travail nous avons étudié de façon générale le problème de cohérence des répliques sur les grilles, nous pouvons implémenter l'approche en ce basant sur des problèmes bien précis tels que l'équilibrage de la charge des nœuds.

Nous proposons de combiner le protocole à quorum structuré en arbre avec d'autres modèles de cohérence relâchés comme le modèle de cohérence de portée.

Nous avons utilisé dans notre approche le protocole à quorum structuré en arbre. Il existe d'autres protocoles à quorum structurés comme les protocoles à quorum hiérarchique et en grille, nous proposons de mener une étude comparative en évaluant l'approche avec l'utilisation des différents protocoles possibles.

La réplication choisie dans notre travail est la réplication aléatoire, nous pouvons évaluer l'approche en utilisant d'autres modes de réplication et faire des comparaisons afin de voir si le mode de réplication influence les performances de l'approche proposée de gestion de la cohérence.

Bibliographie

[ABB 05] : Abbes Heithem, « Introduction aux grilles de calcul », journée du parallélisme, UTIC Université de Paris Nord, le 28/04/2005.

[AUR 09] : Aurélien Ortiz « Contrôle de la concurrence dans les grilles informatiques. Application au projet ViSaGe », Thèse de doctorat à l'Université Toulouse III - Paul Sabatier, Décembre 2009.

[BRI 93] : Brian N. Bershad, Mattew J. Zekauskas, and Wayne A. Sawdon. « The Midway distributed shared memory system », In Proceedings of the 38th IEEE International Computer Conference (COMPCON '93), Los Alamitos, CA, February 1993.

[CHE 02] : M. Chetty and R. Buyya, « Weaving electrical and computational grids: How analogous are they? » Conférence : Computing in Science and Engineering, USA, May/June 2002.

[EDD 05] : Eddy Meylan, « Bases de données : Réplication des données » Support de cours. Haute Ecole spécialisée de Suisse Occidentale, Informatique de Gestion, Février 2005.

[FOS 01] : Foster Ian, « Identifying Dynamic Replication Strategies for a High Performance Data Grid ». The Second International Workshop on Grid Computing Springer-Verlag London, UK (2001).

[FOS 02] : Foster Ian, « What is the Grid? A Three Point Checklist », Argonne National Laboratory & University of Chicago, 2002.

[GUI 06] : Guillaume DESMOTTES, « Déploiement et configuration des intergiciels européens de grilles de calcul », mémoire de licence en informatique, Université Libre de Bruxelles, 2006.

[HUT 90] : P. Hutto and M. Ahamad. Slow memory « Weakening consistency to enhance concurrency in distributed shared memories ». In Proceedings of the 10th International Conference on Distributed Computing Systems (ICDCS '90), May 90.

[IMI 06] : IMINE Abdessamad, « Réplication optimiste vers l'édition collaborative dans les réseaux pair à pair », Thèse de doctorat, Ecole doctorale IAEM LORRAINE, Décembre 2006.

[IVA 05] : Ivan Frain, Jean-Paul Bahsoun, Abdelaziz M'zoughi, « Reconfiguration Dynamique de Coterie Structurée en Arbre » rapport de recherche, Institut de Recherche en Informatique de Toulouse 2005.

[JOL 07] : Jolly Aurélien et Smaël Laskri, « les grilles informatiques », exposé de rapport de recherche, Master Informatique/Réseaux, Institut d'informatique et d'électronique : GASPARD MONGE (Paris Est : Marne la vallée), 2007.

[LAM 79] : L. Lamport. « How to make a multiprocessor computer that correctly executes multiprocessprograms ». Projet de recherche sur IEEE Transactions on Computer September 1979.

Bibliographie

[LIV 96] : Liviu Iftode, Jaswinder Pal Singh, and Kai Li. Scope consistency « A bridge between release consistency and entry consistency ». In Proceedings of the 8th ACM Annual Symposium on Parallel Algorithms and Architectures (SPAA '96), Padova, Italy, June 1996.

[LOI 05] : Loïc Cudennec, « Modèles et protocoles de cohérence des données en environnement volatil », Rapport de Recherche en Informatique de l'IFSIC, Université de Rennes 1, Février - juin 2005.

[LOI 06] : Loïc Cudennec, Sébastien Monnet, « Extension du modèle de cohérence à l'entrée pour la visualisation dans les applications de couplage de codes sur grilles » rapport de recherche, IRISA : Université de Rennes 1, 2006.

[LOU 04] : Louvet Violaine, « Grilles de calcul » notes de cours, Centre National de Recherche de LILLE, 2004.

[MAT 07] : Matthieu Exbrayat, « Bases de Données Réparties : Concepts et Techniques », notes de cours. ULP Strasbourg. Décembre 2007.

[MED 06] : Meddeber Meriem et Kalfadj Fatima, « Un protocole de gestion de Cohérence dans un environnement de simulation Optorsim », Projet de Fin d'Etudes d'ingéniorat, Centre universitaire Mustapha Stambouli - Mascara - Institut d'informatique, 2006.

[PAS 05] : Pascal Molli, Gérald Oster, « Replication des données » Rapport de recherche, LORIA, INRIA Lorraine ECOO Project, Novembre 2005.

[PRO 05] : «Support de cour : Bases de données avancées, Chapitre 8: Protection des SGBD », Université du Sud Toulon Var, 2005.

[RIM 07] : Rim Moussa, « Systèmes de Gestion de Bases de Données Réparties & Mécanismes de Répartition avec Oracle » Rapport de recherche, Ecole Supérieure de Technologie et d'Informatique à Carthage 2007.

[ROB 06] : Robert Walter et Robert Samborski, « Simulating the grig : GridSim » exposé de recherche, Grid Seminar 2, (SE 2.0, 703822), WS 2006/07.

[SEB 06] : Sébastien Monnet, «Gestion des données dans les grilles de calcul : support pour la tolérance aux fautes et la cohérence des données », projet PARIS, IRISA, Rennes École doctorale : MATISSE, Composante universitaire : IFSIC, 2006.

[SOU 03] : SOUEID Toni, « Grilles de calcul : Etat de l'art » Rapport de recherche, ENST (Ecole Nationale Supérieur des Télécommunications), Juin 2003.

[STE 00] : Stéphane Drapeau, « Modèles de Cohérence » notes de cours. France Télécom R&D, Mars 2000.

[STE 03] : Stéphane DRAPEAU, «RS2.7 : un canevas adaptable de services de duplication», thèse de doctorat, Institut National Polytechnique de GRENOBLE, Juin 2003.

Travaux réalisés dans le cadre de Magister

Publications

«**Maintaining Replica Consistency with Load Balancing strategy in grid systems**». Publié en ligne dans le journal "AWERProcedia Information Technology and Computer Science" http://www.world-education-center.org/index.php/P-ITCS/issue/current/showToc, Vol 02 PP : 268 à 273.

Auteurs: S.SENHADJI, A.KATEB, H.BELBACHIR.

Conférence **INSODE2012** IZMIR TURQUIE.

Site : http://www.insode.org/

Communications Acceptées

«**Replica consistency in data grid systems**». Acceptation, en session orale.

Auteurs: A.KATEB, S.SENHADJI, H.BELBACHIR.

Conférence **ICICS 2011** à Singapore.

Site : http://www.icics.org/2011/

«**Replica consistency in data grid systems**». Acceptation, en session orale.

Auteurs: A.KATEB, S.SENHADJI, H.BELBACHIR.

Conférence **GCA World Comp 2011** à Las Vegas, USA.

Site : http://www.world-academy-of-science.org/worldcomp11/ws

«**Maintaining Replica Consistency in Data Grid systems**». Acceptation, en session orale.

Auteurs: A.KATEB, S.SENHADJI, H.BELBACHIR.

Conférence **WOTIC 2011** à Casablanca Maroc.

Site : http://www.wotic.org/

Annexe 1 : Les Grilles

1. Introduction :

L'idée des grilles de calcul est d'effectuer des calculs partagés ou distribués, et d'exploiter pleinement les ressources de l'intégralité des composants de la grille (serveurs, PC, etc.). C'est une forme d'informatique distribuée, basée sur le partage dynamique des ressources entre des participants, des organisations et des entreprises dans le but de pouvoir les mutualiser, et faire ainsi exécuter des applications de calcul intensif ou des traitements de très gros volumes de données. Nous donnons dans ce qui suit quelques concepts et définitions concernant les grilles [ABB 05].

2. Définition et concepts :

2.1. Définition :

Une simple définition des grilles de calcul peut être donnée sous la forme suivante [FOS 02] : Une grille de calcul est définie par trois points essentiels :

- Des ressources informatiques dont leur administration n'est pas centralisée,
- Des méthodes utilisées qui sont standardisées,
- Des ressources dont la qualité de service n'est pas assurée.

Les composants d'une grille de calcul sont [JOL 07] :

- **Partagés** : ils sont mis à la disposition des différents utilisateurs de la grille et éventuellement pour différents usages applicatifs.
- **Distribués** : ils sont situés dans des lieux géographiques différents.
- **Hétérogènes** : ils sont de toute nature, différent par exemple par le système d'exploitation ou le système de gestion des fichiers.
- **Coordonnés** : ils sont organisés, connectés et gérés en fonction des besoins (objectifs) et contraintes (environnements). Ces dispositions sont souvent assurées par un ou plusieurs ordonnanceurs.
- **Non-contrôlés** (**autonomes**) : les ressources ne sont pas contrôlées par une unité commune. Contrairement à un cluster, les ressources sont hors de la portée d'un moniteur de contrôle.

2.2. Objectifs des grilles de calcul :

Parmi les buts recherchés par le déploiement des grilles de calcul on peut noter [LOU 04]:

- Exploitation parfaite des ressources.
- Offre de calcul à parallélisme embarrassant.
- Division des gros problèmes en un grand nombre de taches indépendantes.
- Utilisation pour les applications qui nécessitent une grande capacité de traitement.
- Utilisation pour les modèles complexes où les applications nécessitent de hautes performances de calcul (astrophysique, secteurs aérospatial, modélisation climatique, économie, etc.).
- Réduction de la difficulté des problèmes à grand volume de données (physique des hautes énergies et les observations satellites par exemple).

- Assurance de la fiabilité et la disponibilité des services par la dispersion géographique des ressources et le déploiement des mécanismes de contrôle de gestion.

2.3. Constitution d'une grille de calcul :

La grille est un agrégat de machines provenant de divers domaines administratifs, reliées par un réseau, pour former une plate-forme que l'on peut qualifier d'ordinateur virtuel très puissant. Beaucoup d'applications sont capables d'exploiter l'infrastructure d'une grille : les technologies de collaboration, l'exploration de données, le calcul parallèle ou encore les bases de données distribuées.
La proposition la plus simple est de considérer la grille comme un ensemble de deux entités de nœuds : Clients et Serveurs [JOL 07].
Les serveurs fournissent des tâches à réaliser ou des données à traiter aux clients. Ils servent alors d'Ordonnanceurs afin d'organiser le traitement et recomposent les résultats. L'agrégation des retours des clients permet la création d'un résultat final (Figure 2.1).
Les clients, proposent leur puissance de calcul ou de stockage à la grille afin de créer une sorte de supercalculateur virtuel.
En général, on retrouve des environnements hétérogènes, les clients peuvent se retrouver avec des systèmes d'exploitation différents. Ils peuvent être aussi physiquement différents (Serveurs, Pc, clusters, PDA, calculatrices, tout ce qui a une puce de calcul et qui peut accéder à un réseau). Le principe de la grille de calcul consiste en sa capacité à gérer ces machines qui ne se trouvent pas dans un même lieu. Ces derniers peuvent très bien se trouver à des endroits différents, avec des connexions au réseau complètement différentes (par Internet, réseau local, VPN...).

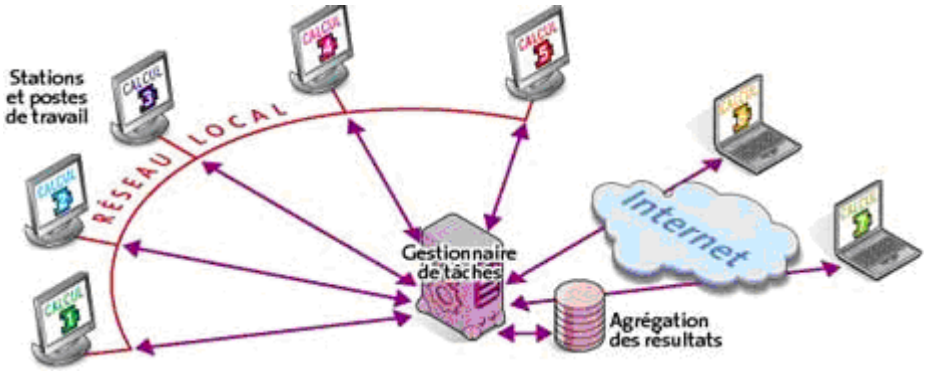


Figure 2.1. Exemple de constitution d'une grille

2.4. Avantages et inconvénients des grilles de calcul :

C'est évident qu'un tel système doit apporter des avantages majeurs [JOL 07].
Le calcul en grille permet le travail parallélisé de plusieurs machines sur un même problème. Les clients proposent leur puissance de travail sur des tâches que le serveur leur a confié. Ces tâches sont indépendantes entre elles, ce qui a pour intérêt d'individualiser le traitement des données, tout en évitant les communications inutiles entre machines.
Les clients participant à la grille de calcul peuvent se situer dans des environnements complètement différents. En effet, il n'est pas nécessaire de posséder le même système d'exploitation, la même puissance de calcul, la même connexion au réseau...

Les clients ne sont pas forcément obligés d'être tout le temps connectés au serveur. En effet, la grille de calcul n'a pas pour obligation d'assurer une qualité de service entre les clients et les serveurs, contrairement aux clusters, qui a besoin, en temps réel, de partager le travail entre les machines.
Le principe de délocaliser la puissance de calcul permet de profiter d'une puissance de calcul exceptionnelle [ABB 05].

Chaque système a des avantages et des inconvénients [JOL 07].
Il est évident qu'un tel grand système, avec un nombre de ressources très important et distribuées sur des zones géographiques éloignées, est difficile à gérer.
Les clients ne possèdent pas une administration centralisée. En effet, ces derniers possèdent chacun des administrateurs qui sont indépendants de l'organisation qui fournit les données de calcul.
Cela a pour conséquence de ne pas pouvoir contrôler l'ensemble des machines de son réseau de calcul. Ce qui peut engendrer des difficultés de contrôler l'intégrité des résultats des traitements.
Dans les grilles de calcul l'outil de réplication de données est largement utilisé, il faut noter donc que maintenir la cohérence des répliques n'est pas une tache facile.
Un des gros problèmes de ce système est l'impossibilité de prévoir à l'avance les ressources réelles qui seront présentes et disponibles à un instant donné. On ne peut donc savoir exactement combien de temps une série de calculs va prendre.

2.5. Architecture des grilles de calcul :

Le modèle d'architecture adapté pour les grilles est un modèle en couches. Les couches hautes (axées sur l'utilisateur) et les couches basses (plus orientées vers les ordinateurs et les réseaux) [CHE 02] (Figure 2.2).

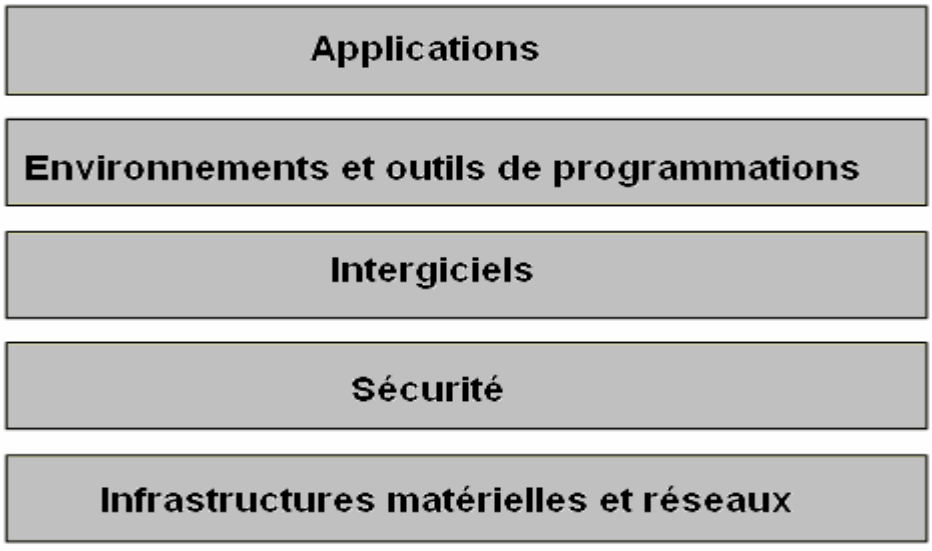


Figure 2.2. Architecture d'une grille de calcul

2.5.1. La couche « Infrastructures matérielles et réseaux » :

Au fond de l'architecture en couches nous avons des ressources et des équipements interconnectées à travers des réseaux locaux avec des distances variées. Cette infrastructure matérielle comprend notamment des PC, des stations de travail, des grappes de calcul (clusters), des équipements de stockage, des bases de données, des équipements spéciaux, les

ressources d'une grille peuvent varier des plus simples (machines de même architecture, même système d'exploitation, réseau d'interconnexion homogène) au plus compliquées (différentes architectures de machines, plusieurs systèmes d'exploitation, étendue géographique, différentes politiques de sécurité, plusieurs administrateurs …).

2.5.2. La couche « Sécurité » :

Cette couche fournit les mécanismes de sécurité nécessaires à la protection des ressources. Dans les grilles de calcul, la sécurité est un souci beaucoup plus important que dans les environnements d'informatique répartie traditionnels. Le nombre important de ressources et leur étendue géographique constituent un facteur important. Ainsi il est fondamental de bien identifier les ressources à protéger et les degrés de protection nécessaires à appliquer. Parmi les mesures de protection on trouve ceux normalement employés dans les réseaux (pare-feu, détection d'intrusion …) ainsi que ceux spécifiques aux environnements répartis et aux grilles.

2.5.3. La couche intergiciel :

Elle regroupe les intergiciels (middleware) nécessaires à la gestion des ressources, la coordination de l'accès, l'ordonnancement des tâches, etc. Nous expliquons ci-dessous brièvement les principales fonctions assurées par ces intergiciels [GUI 06]:

***Ordonnancement** (scheduling) :* un Ordonnanceur (scheduler) devra trouver la machine la plus appropriée pour faire tourner les tâches qui lui sont soumises. On peut trouver des Ordonnanceurs plus simple comme allocation de type round-robin et d'autres plus compliqué comme Ordonnancement à base de priorités. Les Ordonnanceurs réagissent à la charge de la grille. Ils déterminent le taux de charge de chaque ressource afin de bien ordonnancer les prochaines tâches. Ils peuvent être organisés en hiérarchie avec certains interagissant directement avec les ressources, et d'autres (méta-ordonnanceurs) interagissant avec les Ordonnanceurs intermédiaires. Ils peuvent aussi superviser, gérer et prendre des décisions concernant le déroulement d'une tâche jusqu'à sa terminaison.

***Réservation** :* il est possible dans les grilles de calcul de réserver les ressources à l'avance et ceci afin de garantir une certaine qualité de service et de respecter certaines échéances. Pour cela l'intergiciel devra fournir un système de réservation permettant aux utilisateurs d'exprimer leurs besoins en ressources et d'effectuer la réservation.

***Services d'information et d'annuaire** :* une grille est un environnement où le nombre et la charge ressources sont dynamiques. Un objectif lors de la conception d'une grille est de rendre les ressources facilement accessibles à tous les processus. Il est alors nécessaire de fournir des mécanismes permettant d'enregistrer et de découvrir ces ressources et d'identifier leur état.

***Service de nom** :* Comme dans tout système réparti, une grille devra permettre de référencer ses ressources d'une façon standard et uniforme. Une grille de calcul devra fonctionner d'une façon à s'adapter à des technologies matérielles ou logiciels. Un environnement de grille idéal fournira ainsi un accès transparent aux ressources.

2.5.4. La couche « Environnements et outils de programmation » :

Cette couche regroupe tous les outils pouvant aider les développeurs à concevoir et écrire des applications pouvant tourner sur la grille. On y trouve plus particulièrement des compilateurs, des librairies, des outils de conception d'applications parallèles ainsi que des interfaces de programmation que les développeurs d'applications pourront utiliser.

2.5.5. La couche Applications :

Cette couche regroupe les applications qui sont de nature variée : projets scientifiques, médicaux, financiers, ingénierie …

2.6. Propriétés liées à l'architecture d'une grille de calcul :

Quelque soit l'architecture proposée pour la grille de calcul, elle représente des caractéristiques essentielles telles que l'hétérogénéité des ressources, le réseau d'interconnexion, le passage à l'échelle, la dynamicité et la capacité de stockage [AUR 09].

2.6.1. Hétérogénéité des ressources :

La grille est caractérisée par une forte hétérogénéité des ressources. Elle est composée de simples pc de bureau, de clusters ou encore de superordinateurs. De plus, les machines diffèrent de par leur architecture, leur système d'exploitation, leur réseau et leur système de stockage. On retrouve ici un problème critique d'interopérabilité qui doit être géré pour que les systèmes puissent opérer ensemble. Ce sont les intergiciels de grille que nous présentons un peu plus loin, qui assurent ce rôle. De plus, une ressource n'est pas forcément dédiée à la grille. En effet, si on prend le cas d'un ordinateur de bureau, son utilisateur peut tout à fait se connecter à la grille et partager ainsi ses ressources, tout en utilisant sa machine pour d'autres travaux.

2.6.2. Réseau d'interconnexion :

Les ressources de la grille sont regroupées en domaines administratifs géographiquement distants. Bien que le réseau au sein d'un domaine, communication intra-site, soit relativement rapide, le problème vient essentiellement de celui reliant les différents sites, communication inter-site. En effet, ce dernier est caractérisé par une très forte latence. La bande passante quant à elle varie : sur une grille dont le WAN est Internet, la bande passante est très limitée, car elle est partagée par beaucoup d'utilisateurs. Par contre sur une plate-forme dédiée, le réseau reliant les divers sites dispose d'une large bande passante, qui n'est utilisée que pour des applications grille.

2.6.3. Passage à l'échelle :

Le nombre de machines connectées à la grille est très variable et les performances risquent potentiellement d'être dégradées lorsque l'on augmente ce nombre. Si l'on conçoit un algorithme en se basant sur un nombre de nœuds N, il n'est pas dit que celui-ci soit aussi efficace, lorsque l'on passe à N+100 nœuds par exemple. L'architecture de la grille doit donc assurer la préservation des performances, même avec l'augmentation du nombre constituant la grille.

2.6.4. Dynamicité de la grille :

La grille est très sensible aux pannes. Il est très possible qu'un nœud s'arrête ou redémarre pendant l'exécution d'une tache dans de la grille. Vu le grand nombre de machines, la probabilité pour que l'une d'entre elles tombe en panne est naturellement élevée [ABB 05]. Dans beaucoup de services (Ordonnanceur de tâches, service d'information grille, contrôle de la concurrence ...), l'architecture de la grille met en œuvre une hiérarchie pour permettre à la grille de continuer à fonctionner, même lorsqu'une ou plusieurs ressources deviennent indisponibles. Pour cela, certains nœuds se voient attribuer un rôle plus important : on les nomme généralement des nœuds maîtres. Ils sont chargés de prendre les décisions adéquates dans les situations critiques, pour permettre au système de ne pas se bloquer. Lorsque c'est un nœud maître qui tombe en panne, un mécanisme est mis en place pour en élire un nouveau. Les nœuds ne possédant pas ces privilèges sont appelés des nœuds esclaves. Ils ne prennent aucune décision critique, sauf lorsqu'ils sont promus au rang de maître après une élection. Cette hiérarchie permet de s'adapter à la dynamicité de la grille, pour en extraire le maximum de performance.

2.6.5. Capacité de stockage :

La capacité de stockage dans une grille est un concept primordial. En effet, les machines participant à la grille pourront fournir une partie de la capacité de stockage nécessaire au fonctionnement de la grille. Les capacités de stockage dans une grille pourront être utilisées afin d'augmenter la capacité offerte, la performance, l'efficacité de partage et la fiabilité.

2.7. Différents types de grille :

En générale, selon l'étendue géographique et la complexité des grilles de calcul, on peut dire qu'il existe trois types de grilles [SOU 03]:

2.7.1 Intragrilles :

Elles sont caractérisées par une échelle relativement petite, une gestion centralisée et un objectif commun. Elles sont principalement caractérisées par la présence d'un réseau d'interconnexion haut débit, une politique de sécurité unique et maîtrisées par les administrateurs, et un ensemble de ressources relativement statique et homogène.

2.7.2. Extragrilles :

C'est une agrégation de plusieurs intragrilles. Ses principales caractéristiques sont : la présence d'un réseau d'interconnexion hétérogène « haut et bas débit (LAN/WAN), plusieurs domaines de sécurité distincts, et ensemble de ressources plus ou moins dynamique.

2.7.3. Intergrilles :

Consiste à agréger les grilles de multiples organisations, en une seule grille. Elles sont caractérisées par la présence d'un réseau d'interconnexion très hétérogène haut et bas débit (LAN/WAN), plusieurs domaines de sécurité distincts et ayant parfois différentes politiques de sécurité, et un ensemble de ressources très dynamique. Les utilisateurs de la grille pourront être groupés selon leur différents intérêts dans des organisations dites

Annexe 1 : Les Grilles

« Organisations Virtuelles »; chacune possédant sa propre politique et mettant un ensemble de ressources à la disposition de ces utilisateurs.

Annexe 2 : Les Simulateurs de Grilles

1. Introduction :

La gestion et l'ordonnancement des ressources dans un système réparti à grande échelle est complexe et exige des outils très importants pour analyser les algorithmes avant de les appliquer aux vrais systèmes. Pour comparer la performance entre plusieurs algorithmes, les conditions expérimentales doivent nécessairement être les mêmes. Or assurer la même évolution des multiples composants d'un environnement distribué dans le contexte de la grille est impossible : un trop grand nombre de phénomènes amène à un non déterminisme de la plateforme de tests. Il est d'usage de les simuler ce qui permet d'implémenter autant que nécessaire les expériences et de constater par exemple l'influence d'un paramètre en particulier sur les résultats issus des simulations. En conséquence, beaucoup d'outils standard et spécifiques à l'application ont été établis [MED 06].
L'avantage fondamental des simulateurs est leur indépendance à la plateforme d'exécution [MED 06].
La simulation d'un mécanisme d'un système réparti composé d'un nombre très important de nœuds sur un PC simple est possible grâce à la modélisation de ce système et son implémentation sur un simulateur

2. Quels que simulateurs de grilles de calcul :

Il existe plusieurs catégories de simulateurs qui se diffèrent entre eux en fonction de plusieurs paramètres tels que l'environnement de la grille et/ou du réseau d'interconnexion, les caractéristiques des utilisateurs du système, les ressources de calcul et de stockage, le type et la taille des données manipulées, les plates-formes et environnements de programmation, …etc. [MED 06].
Voici quels que simulateurs de grilles de calcul qui sont largement utilisés dans ce domaine.

2.1. SimGrid :

SimGrid est parmi les outils de simulation les plus populaires pour la recherche dans le domaine des grilles. Il est basé sur la simulation conduite par évènement. Il fournit un ensemble d'abstractions et de fonctionnalités pour construire un simulateur correspondant aux applications et aux caractéristiques d'infrastructures. Dans SimGrid, des ressources sont modélisées par leur taux de latence et de service. La topologie est entièrement configurable. Toutes les actions (calculs ou communications) sont référenciées comme tâches. Son objectif est de pouvoir simuler un environnement d'exécution relativement réaliste pour comparer deux algorithmes. Cependant, parce que SimGrid est limité à une entité d'ordonnancement simple et à des systèmes en temps partagé, il est difficile de simuler les utilisateurs en concurrence multiple, les applications et les Ordonnanceurs chacun avec leurs propres politiques.
SimGrid est employé pour simuler le temps de compilation des algorithmes, où toutes les décisions d'établissement du programme sont prises avant l'exécution, il est aussi utilisé pour simuler le temps d'exécution des programmes où certaines décisions sont prises pendant l'exécution [MED 06].

2.2. Bricks :

Bricks est un système d'évaluation des performances pour des algorithmes et des systèmes informatiques à grande échelle. Sa structure comprend deux grandes parties : la partie environnement global de simulation et la partie unité d'ordonnancement. Il permet la

simulation de divers comportements tels que les algorithmes d'ordonnancement de ressource et les topologies de réseaux dans les systèmes informatiques.
Il a été destiné pour certains travaux d'analyse et de performance d'ordonnancement et d'algorithmes de répliques dans le contexte des grilles de données pour la physique de haute énergie.

2.3. GridSim :

GridSim est un simulateur d'évènement discret, il utilise aussi des entités supplémentaires dites Brokers. Il contrôle plusieurs abstractions : utilisateurs, Brokers, ressources, informations de service des grilles, entrées/sorties.
Les Brokers reçoivent les tâches soumises par des utilisateurs et mettent en application leur algorithme d'ordonnancement. Puisque plusieurs utilisateurs concurrents pour le même ensemble de ressources, les Brokers doivent trouver les différences entre les exigences d'utilisateurs. Ces différences doivent être acceptables pour tous les utilisateurs.
Les ressources sont décrites avec le nombre de processeurs, le coût de traitement, performance, la politique interne d'ordonnancement et la charge de travail.
GridSim fait la différence entre les entrées et les sorties, en contrôlant les deux d'une manière séparée. Ceci fournit un moyen d'exprimer les différences de performance entre la communication des paramètres et les résultats. Dans un certain sens, GriSim est un simulateur de plus haut niveau comparé à SimGrid. Il est fondamentalement conçu pour étudier des interactions et des interférences entre les décisions d'ordonnancement prises par les Brokers.
GridSim est principalement employé pour étudier les algorithmes d'optimisation de coût du temps pour les applications d'ordonnancement dans les grilles hétérogènes, considérant le traitement économique de ressources distribuées avec les contraintes de la date limite et du budget.

2.4. GangSim :

GangSim simule une gestion d'infrastructure où l'attribution de la ressource individuelle est déterminée à partir des interactions entre les attributions des politiques dans et à travers les Organisations Virtuelles (VO). C'est un simulateur discret, évaluant périodiquement l'état de tous les composants simulés. Les composants simulés par GangSim sont : les sites, les VO, les Ordonnanceurs (interne, externe et données), les points de surveillance de données, les points d'application de politique du site et les points d'application de politique de VO. En plus de ses capacités (Traitement, intra et inter site, réseau et stockage), un site est définit par une politique d'ordonnancement qui définit les possibilités d'allocation de ressource pour chaque VO. Une VO est un groupe d'utilisateurs qui soumettent des ensembles de travaux. Chaque groupe est représenté par une politique considérée par VO pour l'allocation de ressource. L'Ordonnanceur représente les points où les décisions d'ordonnancement sont prises. GangSim considère plusieurs coûts liés aux composants simulés : le temps d'entrée dans la file d'attente d'Ordonnanceur, le temps d'assignement pour le site, le temps de transfert de site, le temps pour l'assignement du nœud et le temps pour le transfert du travail.
GangSim est utilisé pour simuler les travaux synchrones, où tout les VO soumettent leurs travaux presque en même temps, et asynchrones où les VO soumettent leurs charges de travail à des différents moments.

2.5. EdgSim :

Les composants d'EdgSim fonctionnant aux sites simulés, passent l'information entre eux, coordonnent le transfert de grands fichiers de données requis pour exécuter un job, et distribuent les travaux efficacement entre les machines disponibles. Il utilise le concept d'évènements discrets. Ceci signifie essentiellement que n'importe quel changement d'état dans le système simulé est placé dans une file d'attente, et les évènements sont alors traités dans l'ordre chronologique. Dans une exécution d'EdgSim, chaque élément de stockage a un quota de fichiers de données. Ces fichiers sont inscrits avec le catalogue de réplique. Leurs noms logiques sont également passés à l'objet d'interface utilisateur. Toutes les ressources disponibles sont inscrites avec le Ressource Broker. Après cette phase d'initialisation, l'interface utilisateur commence à produire des travaux. Ceux-ci exigeront un certain nombre de cycles d'unité centrale de traitement. Ils exigeront également un ensemble de fichiers de données, choisi aléatoirement de ceux disponibles dans l'Elément de Stockage (SE). Ces travaux sont produits pendant une période, selon une distribution donnée, et la simulation fonctionne jusqu'au traitement du dernier travail. Afin d'exécuter les travaux, des Eléments de Calcul sont mise en place (CE), un travail est assigné par Broker à une machine avec un CE libre, ou mis en attente jusqu'à ce qu'un CE devient disponible. Tous les fichiers de données qui ne sont pas présents seront demandés par la machine, et le SE concerné essayera de les acquérir. Une fois que tous les dossiers ont été exécutés, le travail sera accompli puis enlevé du système. Chaque site possède des travailleurs reliés par un NIC (Network Information Center) qui gère leurs connections avec les autres travailleurs des autres sites.

2.6. OptorSim :

OptorSim modélise les interactions des composants individuels d'une grille de données. Sa conception dérive directement de l'architecture du projet DataGrid. OptorSim considère les concepts suivants :

- Les sites fournissent des ressources de traitement et/ou de stockage de données.
- Une Ressource Broker pour l'ordonnancement des travaux.
- Un routeur est sans ressource de traitement ou de stockage.

Les travaux s'exécutent sur les ressources de traitement et utilisent les données stockées dans les ressources de stockage. OptorSim permet la description de la topologie de réseau en énumérant les liens entre les sites et leur largeur de bande disponible. Avant le début de la simulation l'utilisateur peut configurer la topologie de la grille et la tabulation des travaux. Les différents algorithmes d'expédition des Ressources Broker et de la gestion d'optimisation de réplique sont établis dans un document de paramètres, en attendant, d'autres paramètres peuvent être assignés, par exemple le modèle de transfert de fichiers, la distribution de document d'initialisation, le temps de processus. Après la simulation, on peut avoir des statistiques des résultats.
OptorSim a été destiné pour évaluer plusieurs stratégies de réplication et de les comparer, parmi ces stratégies on a : aucune réplication, réplication sans conditions, réplication avec suppression des fichiers les plus anciens si l'espace est insuffisant.

2.7. NEKO :

NEKO est une infrastructure destinée aux algorithmes distribués. Les processus interagissent par échange de messages. Développé sous java, il est extensible et facile à utiliser et permet une simulation ou une exécution sur une grille réelle. Une variété de réseaux simulés ou réels sont supporté et une librairie des algorithmes de tolérance aux fautes est aussi incluse.

Annexe 2 : Les Simulateurs de Grilles

NEKO a été développé par "the Distributed Systems Lab" à l'institut fédéral de technologie de suisse en Lausanne (EPFL), par deux étudiants "Xavier Défago" and "Péter Urbán". Son Développement a été continué à l'institut des technology avancées au Japan(JAIST) par le group DDSG, et l'université polytechnique de Madrid, LSD lab. Depuis 2000 NEKO a été développé et utilisé par différents utilisateurs. Le principe de base dans NEKO est que toutes les informations de configuration sont centralisées. Les fichiers de configuration décrivent le type du réseau, le nombre et l'emplacement des processus etc.