Table des matières

Table des Figures

Table des Figures

Liste des tableaux

Introduction générale

Au cours de la dernière décennie, les recherches et les évolutions autour des technologies de réseaux, des microprocesseurs et des supports de stockage de données ont fortement contribué à l'émergence de l'informatique distribuée [11]. Les performances de calcul des microprocesseurs et les capacités de stockage de données ne cessent de croître, alors que les nouvelles technologies de réseau ouvrent de nouvelles potentialités pour les communications. Ce contexte a motivé la communauté informatique à s'intéresser aux architectures distribuées à large échelle, afin d'offrir des solutions pour le stockage de données et le calcul réparti à un plus grand nombre d'applications et d'utilisateurs.

En parallèle, l'évolution des systèmes informatiques s'est caractérisée par une tendance très forte vers la décentralisation. La communication, le partage de données, la puissance de calcul et la capacité de stockage sont des besoins fortement exprimés par les nouvelles applications informatiques [12]. Des exemples d'applications couvrant différentes disciplines ont montré le besoin de disposer d'architectures plus flexibles, favorisant le calcul et la distribution de données à grande échelle [47, 50]. Des disciplines aussi diverses que la météorologie, la science des particules, la médecine et la biologie sont des exemples où les applications requièrent de grandes capacités de calcul et de stockage. La tendance à la distribution a entraîné une évolution de l'architecture des systèmes distribués et des outils pour la conception et la mise en œuvre d'applications distribuées.
Parmi ces applications, nous pouvons citer :

- Le Large Hadron Collider, collisionneur de particules du CEEN (Organisation européenne pour la recherche nucléaire) qui produit 400 millions d'événements par seconde, soit l'équivalent de 15 Petaoctets par an, qui seront visualisés et analysés par 6000 physiciens dans le monde [59]
- La recherche de silico de médicaments par l'analyse de ligands, annonce un potentiel de 46 millions de ligands pour l'étude de la maladie de malaria. Les traitements relatifs à cette recherche nécessitent une capacité de 1000 machines/mois et génèrent plusieurs Teraoctets de données [47].
- Les applications de prédiction météorologique qui génèrent un volume de 8 Peta- octets et il est prévu que ce volume augmente les prochaines années [12].

A travers ces quelques exemples, nous remarquons que les demandes en puissance de calcul et en capacité de stockage sont énormes et dépassent largement les capacités informatiques d'un particulier, d'une institution et même d'un pays. Pour faire face à ces exigences, les systèmes à large échelle sont apparus vers la fin des années 90 [11].

De nos jours, les systèmes à large échelle sont devenus des architectures incontournables pour les applications qui utilisent de grands volumes de données et qui demandent beaucoup de puissance de calcul. Ces systèmes offrent la possibilité de partager des ressources (informatiques et non informatiques) multiples et hétérogènes à travers une architecture qui permet de faire interopérer ces ressources d'une manière transparente pour l'utilisateur [49].

Deux types d'architectures se sont imposés parmi les systèmes à large échelle : les grilles qui consistent à interconnecter des grappes d'ordinateurs géographiquement réparties, et les architectures P2P dans lesquelles un ensemble d'ordinateurs individuels coopèrent d'égal à égal. Ces architectures de grande taille remettent en cause un certain nombre de concepts, de méthodes, de techniques et d'outils informatiques à cause de quatre facteurs très importants :

- *L'espace géographique* de leur déploiement qui dépasse largement celui d'un système distribué ;
- Leur *hétérogénéité* qui se situe à plusieurs niveaux : matériel, système, réseaux de communication et environnements de développement ;
- Leur *dynamicité*, notamment du point de vue des ressources.
- Le problème de *sécurité* et *fiabilité* des données qui augmente dans ces infrastructures

Problématiques et motivations :

Dans cette thèse, nous nous intéressons particulièrement aux grilles de données qui permettent de connecter des centaines de machines et de ressources de stockage distribuées à travers le monde. Ce type de grille est utilisé par les applications intensives qui génèrent des quantités importantes de données. Ces applications peuvent avoir besoin de données produites par d'autres applications dans un endroit distant géographiquement. Comme les données représentent la ressource la plus importante dans ce type de grilles, une gestion et un accès efficace à ce volume de données présentent des défis majeurs.

La réplication est une technique souvent utilisée pour résoudre ces problèmes. Elle consiste à créer de multiples copies de la même donnée dans plusieurs ressources de stockage de la grille. Le problème de réplication est un problème relativement connu dans les systèmes distribués, mais il pose de nouvelles problématiques dans les environnements à large échelle tels que :

a. Quels sont les fichiers à répliquer ?
b. Dans quels sites faut-il placer les fichiers candidats à la réplication ? (trouver le bon emplacement)
c. À partir de quel site faut-il transférer un fichier demandé en présence de plusieurs copies (c'est le problème du choix de la meilleure réplique) ?
d. Décider, pour un fichier donné, le nombre minimal (ou maximal) de répliques à créer.

En théorie, La réplication vise principalement à améliorer le temps de réponse des applications, réduire la consommation de la bande passante, augmenter le niveau de disponibilité des données, et équilibrer la charge. En pratique, ces objectifs s'avèrent conflictuels car quand on vise à répliquer les données pour assurer leurs disponibilités, ceci se fait naturellement au détriment de communications entre sites et de transfert de données qui surchargent le réseau, ce qui n'est pas sans conséquence sur le temps de réponse aux requêtes. Paradoxalement, sans le processus de réplication, la disponibilité des données diminue et le temps de réponse augmente, on peut même perdre certaines requêtes faute de disponibilité.

La recherche de stratégies efficaces pour la réplication de données dans les systèmes à grande échelle a été abordée par Ian Foster dès les années 2000 [2,3,5]. Bien que le problème a été largement étudié et plusieurs stratégies de réplication ont été proposées, il demeure toujours un problème d'actualité car il est très difficile, voir illusoire, de définir une stratégie de réplication universelle supportée par n'importe quelle grille de données. Ceci est dû aux caractéristiques de la grande échelle à savoir : l'hétérogénéité des ressources et la dynamicité des sites, ajouté à cela le profil et le comportement des utilisateurs qui peut changer d'un instant à l'autre.

Pour être efficace, une stratégie de réplication doit définir un contexte dans lequel elle s'applique. Ce contexte portera sur les paramètres suivants :

- **Topologie logique de la grille** : la topologie de la grille affecte énormément la stratégie de réplication. En effet, une stratégie de réplication proposée dans le cadre d'une grille hiérarchique (Multi-tiers) ne produira pas les mêmes résultats que pour une grille complètement distribuée.
- **Ressources utilisées** : Une stratégie de réplication proposée dans une grille de données de partage de contenu où le principal objectif est le partage de fichiers en lecture ne se préoccupera pas du problème de gestion de la cohérence des données manipulées. Ce qui n'est pas le cas d'une stratégie autorisant des accès en écritures.
- **Objectifs visés :** Comme mentionné ci-dessus, les objectifs d'une stratégie de réplication sont conflictuels. Une bonne stratégie de réplication doit trouver un compromis pour augmenter les performances générales du système selon le domaine d'application de la grille.

Contributions :

L'objectif visé à travers cette thèse est de pouvoir proposer des stratégies efficaces de réplication dynamique de données dans les grilles de données.

Plusieurs stratégies de réplication ont été proposées dans la littérature. Chacune d'elles offre des solutions pour répondre aux questions fondamentales de la réplication. Mais la plus part des travaux qui existent se limitent aux topologies hiérarchiques au départ [5,8,9,10,13,14,18,23,25], ce qui exclut une large gamme de grilles de topologies différentes. Aussi les solutions sont statiques, du fait qu'elles fixent certains paramètres (chemin de retour, répliques hébergées par la racine) et ne s'adaptent donc pas à la nature dynamique de la grille : la structure arborescente du modèle fait qu'il existe des chemins spécifiques que les messages et les fichiers doivent traverser pour arriver à destination. Un transfert entre deux nœuds de même niveau n'est pas envisageable. De plus, si la racine tombe en panne toute la grille est paralysée.

La représentation d'une grille la mieux adaptée est un graphe général dans lequel il n'existe pas de nœud central désigné comme nœud racine, et chaque nœud peut être connecté à n'importe quels autres nœuds sans restriction.

Dans le cadre de ce travail, nous considérons qu'une grille est un environnement complètement distribué, nous adoptons donc une architecture décentralisée où les nœuds sont

organisés indépendamment les uns des autres, ce qui offre une flexibilité dans la communication.

Notre contribution consiste en la proposition de quatre (04) stratégies dynamiques de réplication pour les grilles de données :

- Les deux premières stratégies s'attaquent au problème de placement dynamique des répliques en discutant respectivement :
  a. le problème du nombre de répliques à créer (pour la première stratégie) ;
  b. le problème du coût de la réplication (pour la seconde stratégie) ;
- Les deux dernières approches constituent des propositions d'amélioration de stratégies de réplication déjà existantes et connues : Fast spread [5,11,22] et BHR (Bandwidth Hierarchy Replication) [8,13,14]. Au départ, ces deux stratégies ont été développées pour des architectures hiérarchiques. Nous y avons apporté des améliorations pour pouvoir les intégrer dans une topologie hybride.

Enfin, notons que dans une stratégie de réplication, il est bien évident qu'une donnée ne peut être répliquée à l'infini. Pour cela, toute les stratégies que nous avons proposées tiennent compte de ce paramètre et proposent différentes politiques de suppression de répliques (au cas où l'espace de stockage est insuffisant).

Expérimentations :

Afin d'évaluer le comportement des approches proposées et valider les résultats obtenus, nous avons effectué une série d'expérimentations en utilisant le simulateur OptorSim[42]. Nous y avons implémenté nos algorithmes en intégrant des modifications. Le simulateur OptorSim a été développé par European Grid projects. Ce simulateur est écrit en java et constitue un bon environnement pour le test de stratégie dynamique de réplication.

Les stratégies implémentées ont été comparées à d'autres stratégies de réplication en utilisant les métriques suivantes : temps de réponse aux requêtes, nombre de requêtes créées, consommation des ressources réseaux et le nombre de messages échangés.

Organisation du manuscrit :

Les travaux de recherche que nous avons menés pour définir les quatre stratégies de réplication dans les grilles de données, sont synthétisés dans ce document qui est organisé en deux grandes parties, outre une introduction générale et une conclusion.

- **Partie1** : Cette première partie présente un état de l'art des systèmes à large échelle et du problème de réplication sous-jacent. Elle comprend trois chapitres :

  Le chapitre 1 décrit les systèmes P2P, leurs caractéristiques et leur évolution de point de vue topologie. Les domaines d'application de ces systèmes sont présentés à la fin du chapitre.

Le chapitre 2 expose un état de l'art sur les grilles informatiques, il présente les différents types de grille en mettant l'accent sur les grilles de données qui constituent l'infrastructure sur laquelle reposent les approches proposées dans cette thèse.

Le chapitre 3 introduit le problème de réplication en discutant les avantages et les inconvénients de cette technique. Une étude des approches de réplication existantes est donnée à la fin du chapitre.

- **Partie2** : Cette partie décrit la conception et l'implémentation des approches que nous avons proposées pour la réplication de données dans le cadre de ce travail. Elle comprend quatre chapitres. Chaque chapitre décrit en détail l'approche implémentée en exposant :
  i) la problématique ;
  ii) la topologie logique adoptée pour la représentation de la grille ;
  iii) les hypothèses ;
  iv) le modèle fonctionnel utilisé ;
  v) les algorithmes implémentés.
  A la fin de chaque chapitre les résultats des expérimentations sont présentés et analysés.

Finalement, nous conclurons cette thèse en dressant un récapitulatif des différentes contributions que nous avons apportées au problème de réplication dans les grilles de données. Nous terminerons par relater quelques pistes de recherche qui nous semblent pertinentes par rapport à ce problème.

Partie 1 :
Etat de l'art

Introduction :

Dans cette partie, nous présentons le contexte dans lequel nous avons effectué nos travaux de recherches : Les systèmes à grande échelle et la réplication.

Depuis quelques années, la gestion des données dans un environnement à grande échelle est devenue indispensable pour prendre en compte les besoins des applications informatiques véhiculant et stockant de très grandes masses de données. Ainsi, de nouvelles infrastructures de distribution ont vu le jour pour prendre en charge les deux aspects de dynamicité et de grande échelle : il s'agit de la technologie centralisée appelée Grille de Calcul [1,22,48] et des technologies décentralisées auto –adaptatives P2P [54,55,56].

Une grille informatique est une infrastructure matérielle et logicielle dont le but est de connecter des ressources partagées, distribuées et hétérogènes. Les ressources propres d'un ordinateur qui se connecte à la grille sont mises à la disposition de l'ensemble des autres nœuds. En même temps, l'ordinateur accède de manière transparente pour l'utilisateur, à l'ensemble des ressources de la grille. Celle-ci se comporte alors comme un ordinateur unique. L'objectif de la grille est donc d'apporter à l'utilisateur final la possibilité d'utiliser des ressources distantes ou de lancer une application qui demande beaucoup de ressources non disponibles localement. La nature des ressources partagées est très diverse comme la capacité de calcul, les moyens de stockage, les applications ou encore le matériel scientifique.

Depuis plusieurs années, les systèmes p2p se sont développés et sont devenus populaires avec l'´échange de fichiers via Internet grâce à Napster [85]. La principale caractéristique par rapport à la grille réside dans leur architecture : dans les systèmes p2p, les nœuds possèdent les fonctions à la fois d'un serveur et d'un client (ils sont appelés servent) et peuvent communiquer directement entre eux. Les systèmes p2p sont aussi caractérisés par des communications ad-hoc, c'est à dire des communications point-à-point. Cette décentralisation permet le passage à l'échelle en réduisant la charge des nœuds dédiés. Les systèmes actuels comportent ainsi des millions d'utilisateurs. Les auteurs [84] précisent cependant que les grilles et les systèmes p2p ont tendance à converger vers le même objectif, tout en partant d'un point de départ différent. Ils sont développés dans le but d'être tolérants aux pannes et de supporter une mise à l'échelle importante. Une des approches pour la conception d'applications de grille est donc basée sur le modèle p2p : les serveurs sont supprimés et les services rendus par ces serveurs sont repartis dans la grille. La répartition de la charge sur les nœuds permet d'éviter la distinction de nœuds particuliers, ce qui permet une plus grande tolérance aux pannes.

Dans cette première partie de la thèse, Nous exposons en premier lieu un état de l'art sur les systèmes P2P et les grilles de calcul tout en discutant leurs point communs et leurs divergences. Nous terminons par présenter la technique de réplication, son principe, ses différents types et modèles. Nous relèverons qu'en dépit de la diversité des modèles et des approches, le but reste le même, à savoir l'amélioration des performances des systèmes et la disponibilité des ressources. Nous mettrons l'accent sur les problèmes posés par l'utilisation de la réplication dans les environnements à large échelle.

Chapitre 1
Etat de l'art sur les systèmes P2P

1. Etat de l'art sur les systèmes P2P

1.1 Introduction

Les réseaux peer to peer (P2P) sont considérés aujourd'hui comme l'une des plus importantes sources de partage de données et leur intérêt ne cesse de croître au fur et à mesure qu'ils sont utilisés dans de nombreux domaines.

Un système P2P ou Peer-to-Peer désigne tout système informatique basé sur un modèle réseau d'égale à égale ou chaque nœud est à la fois offreur et demandeur de services (d'où l'appellation de tel nœud "Servent" contraction de "**se**rveur" et de "Cli**ent**") par opposition au modèle Client/serveur où chaque nœud a un rôle bien précis (soit il offre un service ce qui fait de lui un serveur, ou bien il est demandeur de service ce qui fait de lui un client).

Même si un tel modèle existe depuis les premiers pas de l'informatique, les premiers systèmes P2P n'ont suscité d'intérêt qu'avec l'apparition de la première plateforme de partages de musique en l'occurrence Napster[1], ce qui par la suite a donné naissance à une myriade d'applications informatiques basées sur le modèle P2P.

1.2 Définition d'un système P2P

L'absence de standardisation du modèle P2P, laisse libre court à la multiplication de définitions très variées. Le terme peer to peer est utilisé dans de nombreux contextes, parfois avec différentes significations. Ainsi dans la littérature nous retrouvons une multitude de définitions. Elles ne sont pas toutes parfaitement équivalentes : nous pouvons citer :

Definition1 "Peer to-Peer computing is the sharing of computer resources and services by direct exchange between systems." Intel P2P Working Group [56].

Definition2 *"P2P is a way of structuring distributed applications such that the individual nodes have symmetric* roles. Rather than being divided into clients and servers each with quite distinct roles (such as Web clients vs. Web servers), in P2P applications a node may act as both a client and a server."William Yeager, P2P Working Group[57].

Definition3 "A distributed network architecture may be called a P2P network, if the participants share a part of their own hardware resources (processing power, storage capacity, network link capacity, printers). These shared resources are necessary to provide the Service and content offered by the network (e.g. file sharing or shared workspaces for collaboration). They are accessible by other Peers directly, without passing intermediary entities. The participants of such a network are thus resource providers as well as resource requestors." Pr Rüdiger Schollmeier, Université technique de Munich (TUM)[53].

Definition4 "P2P is a class of applications that takes advantage of resources (storage,cycles, content, human presence) available at the edges of the Internet. Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, P2P nodes must operate outside the DNS system and have

1: http://www.napster.com

significant or total autonomy from central servers." Clay Shirky, écrivain et consultant spécialisé dans les technologies Internet[58].

Il ressort de toutes ces définitions, qu'un système P2P se rapporte à une classe de systèmes et d'applications qui utilisent des ressources distribuées afin de rendre un service de façon totalement décentralisée. Un système est dit P2P lorsqu'il autorise la communication directe entre entités d'un réseau, sans passer obligatoirement par une autorité centrale, telle qu'un serveur.

1.3 Caractéristiques d'un système P2P

Un système P2P se distingue par plusieurs caractéristiques :

1.3.1 Partage de ressources

C'est une caractéristique essentielle des systèmes P2P. En effet, les systèmes P2P sont nés du besoin des applications fortement consommatrices de ressources qui prennent diverses formes, matérielles ou logicielles tel que : Capacité de stockage, puissance de calcul, services, etc. Ces applications offrent des performances médiocres sur des architectures Client/serveur où seuls les serveurs sont surexploités alors que les ressources des machines clientes sont inutilisées. La solution a été de concevoir un système où tout participant coopère en mettant en commun ses ressources et opérant aussi bien comme serveur que client : le P2P [53,54].

1.3.2 Décentralisation

La première caractéristique d'un système P2P est la décentralisation. Un tel système est dépourvu d'entité centrale de coordination pour l'organisation du réseau, ou pour l'utilisation de ressources, ce qui implique une communication directe entre les nœuds pour toutes les tâches demandées. Aucun nœud n'a de contrôle central sur l'autre. Un tel système n'est pas facile à concevoir d'où l'existence de systèmes hybrides qui ne sont pas entièrement décentralisés [51,55].

1.3.3 Passage à l'échelle

Le passage à l'échelle d'un système p2p est souvent décrit comme l'intérêt principal. L'un des premiers avantages de la décentralisation est la facilité du passage à l'échelle due à l'absence de nœuds centraux qui se chargent de la synchronisation et de la coordination, ceci explique l'intérêt suscité par le P2P. Une définition récurrente du principe du passage à l'échelle est la suivante : "la résistance au changement d'échelle dans un réseau P2P correspond à la capacité propre du système à maintenir son efficacité inchangée lors de l'arrivée ou du départ d'un certain nombre de nœuds" [54,55].

1.3.4 Anonymat

L'anonymat se traduit par la possibilité donnée aux utilisateurs d'utiliser le système P2P sans se préoccuper des ramifications juridiques ou autres, ainsi que de lutter contre toutes formes de censure. Nous pouvons distinguer six formes d'anonymat : d'auteur, d'éditeur, de

serveur, de document, de lecteur et de requête. Plusieurs applications implémentent cette caractéristique car elle permet de ne pas être identifiable sur un réseau. [59]

1.3.5 Autonomie

Les systèmes P2P sont dédiés aux architectures à large échelle. Une telle architecture est fortement instable non seulement à cause des pannes mais surtout à cause de la liberté d'un Peer à se connecter ou à se déconnecter à tout moment, ou de disposer de ses ressources comme il l'entend. Donc en évitant d'avoir une gestion centralisée, un système P2P offre un degré d'autonomie considérable aux utilisateurs en réalisant les traitements localement et librement sur leurs Peers contrairement aux autres architectures [51,60].

1.3.6 Connectivité Ad Hoc

Le comportement des usagers influe considérablement sur la disponibilité des peers. Les usagers peuvent à leurs gré se connecter ou se déconnecter de manière spontanée et donc imprévisible. Du fait de la volatilité des Peers, des mécanismes de duplication et de synchronisation sont mis en place pour pallier à ce problème. L'absence d'un peer ou d'une ressource ne doit pas être considérée comme une faute [55].

1.3.7 Auto-Organisation

De par la nature décentralisée d'un système P2P et de ses utilisateurs (Autonome et Connectivité Ad Hoc), l'auto organisation du système est la solution la plus envisageable, ainsi chaque Peer maintient une vue locale et le système s'auto organise au gré des interactions entre Peers. Cette caractéristique prend encore plus de sens lors du passage à l'échelle d'un système aussi dynamique, en effet une gestion centralisée serait couteuse en terme de moyens [55,60].

1.3.8 Robustesse et Tolérance aux pannes

La Robustesse d'un système se résume par sa capacité à maintenir une stabilité même en cas de défaillances de certains de ces composants, cela se traduit dans les systèmes P2P par leurs capacités à fonctionner et à se stabiliser même en cas d'erreur, ou de déconnexion de certains Peers.

Les systèmes P2P pallient au problème du point de défaillance présent dans les architectures client-serveur, puisque leur aspect décentralisé leur permet de continuer à offrir les services auxquels ils sont dédiés même après la disparition de certains de leurs participants. [54]

1.3.9 Performance

La performance est un souci significatif dans les systèmes P2P. Ces derniers visent à améliorer leur performance par l'agrégation de nouvelles ressources (capacités de stockage et puissances de calcul). Cependant, en raison de la nature décentralisée de ces modèles, la performance est influencée par trois types de ressources : traitements, stockage et gestion du réseau. En particulier, les délais de communication sont très significatifs dans les réseaux à grande échelle. Dans ce cas, la bande passante est un facteur important lorsqu'il s'agit de

propager un grand nombre de messages ou d'échanger plusieurs fichiers entre les peers. Il existe trois approches principales pour optimiser la performance d'un système p2p :

1.3.9.1 La réplication

Cette approche consiste à placer les données le plus proche possible des Peers qui les demandent afin de réduire le nombre de messages émis, ainsi que de maintenir un certain niveau de disponibilité de la ressource.

1.3.9.2 Le Caching

Il consiste à stocker un fichier sur l'ensemble des nœuds intermédiaires entre le nœud qui demande le fichier et le nœud qui répond à la requête, cette technique permet de réduire le temps de recherche d'un fichier ainsi que le nombre de messages échangés, ce qui se traduit concrètement par une diminution conséquente de la latence de la communication.

1.3.9.3 Le routage intelligent et l'organisation du réseau

Pour réaliser entièrement le potentiel des systèmes peer to peer, il est important de comprendre et d'explorer les interactions sociales entre les peers. Plusieurs études ont été menées afin d'améliorer les stratégies de recherche d'information dans les réseaux peer to peer et leurs performances. [54]

1.3.10 Réduction des coûts

Offrir des services au moyen de serveurs dédiés dans les architectures client-serveur est plus coûteux que de les offrir en exploitant les ressources inutilisées des stations existantes. En effet, les serveurs des architectures centralisées coûtent cher et requièrent un entretien continuel, tandis que les dispositifs faisant office de Peers sont abordables et quasi autonomes.

1.4 Topologies des systèmes P2P

En théorie un système P2P est conçu autour de machines qui communiquent directement entre elles d'égale à égale d'une manière entièrement décentralisée. En pratique ce n'est pas toujours le cas, vu les difficultés rencontrés lors de la conception des premiers systèmes (manque de maitrise de la technologie P2P, difficulté de passage à l'échelle).

Depuis leur émergence à la fin des années 90, les systèmes peer to peer ont beaucoup évolué et se sont diversifiés dans leur architecture. On peut classifier les réseaux peer to peer en trois générations [28]:

1.4.1 Première Génération : Architecture Centralisée

Comme montré dans la figure 1.1, dans cette architecture, les Peers s'organisent autour d'un nœud central qui fait office de serveur dédié uniquement à l'indexation des ressources et à la mise à jour des métadonnées relatives aux Peers, par contre la communication se fait directement entre les peers ce qui différencie cette architecture par rapport à l'architecture Client/serveur.

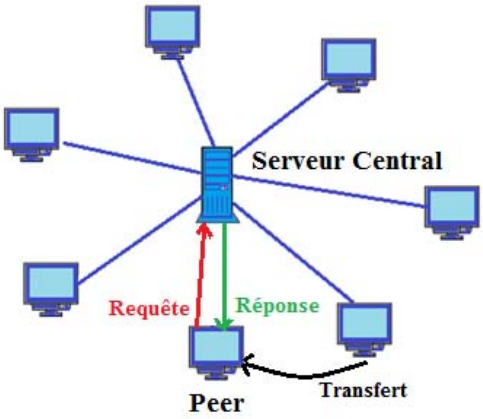


Figure 1.1. Architecture P2P Centralisée

1.4.1.1 Mode de fonctionnement

- Chaque Peer communique au serveur la liste des ressources qu'il possède.
- Un Peer qui est à la recherche d'une ressource bien spécifique envoie une requête d'index au serveur pour avoir la liste des Peers détenteurs de cette ressource (Identifiant, adresse, etc).
- Le Peer demandeur de la ressource communique directement (il envoie une requête de donnée) avec les autres Peers qui la détiennent en se basant sur la liste précédemment reçue.

1.4.1.2 Avantages

- L'existence du serveur central facilite l'administration et le contrôle du système.
- Facilité et rapidité de recherches des données.

1.4.1.3 Inconvénients

- La présence d'un serveur central rend l'architecture sensible à la défaillance, au partitionnement du réseau et aux attaques.
- Aucun anonymat n'est possible, puisque chaque utilisateur est identifié sur le serveur.
- La non-scalabilité, puisque cette architecture supporte difficilement l'évolution rapide des clients. L'augmentation des peers devrait être suivie par la multiplication de serveurs.
- La taille de la base de données du serveur évolue proportionnellement par rapport au nombre de clients.

Exemple : Napster [85]

Crée en 1999 par Shawn Fanning, c'est la première application P2P a grand succès au point d'ouvrir cet horizon à une multitude d'applications. Elle consiste en une plateforme de partage de morceaux musicaux entre les différents utilisateurs. Un utilisateur partage des morceaux MP3 sur le réseau, en échange il peut télécharger les morceaux disponibles sur les

autres machines. Cette application est basée sur une architecture centralisée, où le serveur se charge uniquement de recenser l'ensemble des morceaux partagés et leurs emplacements.

Les démêlées juridiques relatives au non-respect des droits d'auteur ont abouti au bannissement de Napster.

1.4.2 Deuxième Génération : Architecture Décentralisée

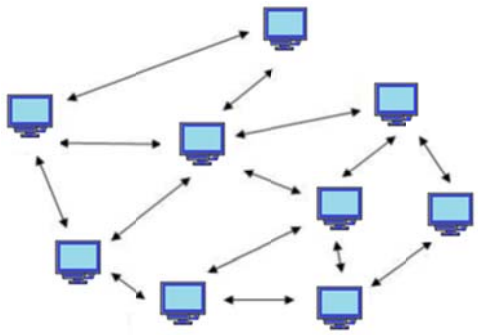


Figure 1.2. Système complètement décentralisé

Plutôt que de faire intervenir un serveur central pour l'indexation des ressources et la synchronisation des Peers, toutes ces informations sont réparties sur l'ensemble des Peers qui forment le parc informatique, ces modèles sont généralement classifiés en deux catégories distinctes :

1.4.2.1 Systèmes décentralisés non-structurés

On parle de réseau P2P non-structuré si le réseau ne prend pas en charge l'indexation des ressources partagées ou bien si la topologie du réseau n'est pas connue.

1.4.2.1.1 Mode de fonctionnement

- Chaque Peer indexe ses propres ressources partagées.
- Un Peer qui est la recherche d'une ressource émet une requête d'index vers les Peers voisins (Broadcast), qui la propagent à leurs tours.
- Ce Processus est répété aussi longtemps qu'aucune réponse n'a été retournée (la ressource n'a pas été trouvée ou que le nombre maximum d'étapes d'inondation n'a pas été atteint)
- Lorsqu'une réponse favorable est retournée au Peer demandeur, ce dernier communique directement avec les peers détenteurs de la ressource (requête de données).

1.4.2.1.2 Avantages

- La taille du réseau est théoriquement infinie puisqu'elle ne dépend plus du nombre de serveur ni de leur puissances.

- Le réseau permet la préservation de l'anonymat de ses utilisateurs et rend la censure des données quasi-impossible.
- Le réseau est tolérant aux fautes notamment grâce à l'absence d'un point de défaillance.
- L'architecture s'adapte bien à la dynamique du réseau du point de vue des connexions et déconnexions des peers.

1.4.2.1.3 Inconvénients

- Consommation accrue de la bande passante à cause des diffusions (*broadcasts)*, donc l'affirmation que la taille du réseau est infinie est une illusion.
- On ne peut pas avoir une estimation du temps de réponse à une requête car son itinéraire dépend à la fois de l'état du réseau et de la situation géographique des peers source et destination.
- Le réseau ne garantit pas la fiabilité des données téléchargées, car il n'y a plus de point central qui effectue des vérifications d'authenticité des fichiers, de la qualité des peers et des données fournies.
- Les personnes qui ne partagent pas de fichiers nuisent sérieusement au réseau, on les appelle les Free-Riders. Leur multiplication est incontrôlable, ce qui altère la diversité et la disponibilité des données.

.

Exemple : Gnutella v0.4 [66]

Il succède à Napster dont l'architecture centralisée est jugée vulnérable. C'est le premier système d'échange de fichiers où la recherche et le transfert de fichiers sont complètement décentralisés. Il a été créé en 2000 par Justin Frankel et Tom Pepper. Dans un réseau Gnutella, chaque nœud possède un ensemble de fichiers qu'il met à disposition des autres et qu'il indexe localement. Il est connecté à un ensemble de voisins (typiquement une dizaine). La recherche se fait par diffusion avec un TTL[2] (Time To Live) initialisé à 7.

1.4.2.2 Systèmes décentralisés structurés

A l'inverse de l'architecture décentralisée non structurée, la topologie du réseau dans ce cas est connue (en général c'est une structure en anneau) et l'emplacement des données est choisi dans le but d'optimiser les recherches, les fichiers sont ainsi placés à des emplacements spécifiques.

2 : paramètre indiquant le nombre de nœuds rencontrés

1.4.2.2.1 Mode de fonctionnement

- L'ensemble des ressources du système sont indexées sous forme de tables de hachage distribuées (DHT), en effet chaque Peer indexe une partie des ressources partagées sur le réseau.
- Chaque Peer dispose d'un identifiant permettant de le localiser en suivant un chemin déterministe parmi les Peers, et nécessitant un minimum de messages.
- Chaque ressource partagée au sein du réseau possède également un identifiant, qui est le résultat d'une fonction de hachage, et permettant de la localiser rapidement.
- L'ensemble des identifiants forment une table de hachage qui est distribuée sur les Peers du réseau pour former une DHT, chaque Peer étant responsable des entrées de la table égales ou proches de son identifiant.

1.4.2.2.2 Avantages

- Le succès d'une recherche de fichier dans le système est garanti.
- Le nombre de nœuds contactés avant d'arriver à destination est optimisé.
- Le passage à l'échelle du système est assurée jusqu'à plusieurs millions de nœuds.

1.4.2.2.3 Inconvénients

- La gestion de l'indexation est très compliquée mais compte tenu des avantages apportés par cette architecture ceci ne constitue pas un inconvénient majeur.

Exemple : Chord [67]

C'est un projet P2P développé au MIT (Massachusetts Institute of Technology). Il utilise une topologie en anneau. Il a pour particularité de disposer d'algorithmes d'une complexité en O(log N) pour trouver une information dans un anneau de N éléments. Il utilise pour cela une table de hachage distribuée (DHT).

Un nœud Chord a la connaissance de son prédécesseur et de son successeur. Une fonction de hachage régulière génère une clé pour chaque nœud à partir de son adresse IP et de chaque fichier à partir de son contenu. Ensuite, chaque nœud ainsi que chaque fichier sont placés dans l'anneau de manière à ordonner les clés par ordre croissant. Ainsi, chaque nœud i est responsable des fichiers dont l'intervalle de clés est [clé (nœud i), clé (nœud i+1)]
Chord est utilisé dans les applications suivantes :

- Coopérative File System (CFS), qui est un système de stockage de fichiers distribué. Ce système répartit la charge de manière équitable sur tous les nœuds du réseau.
- Résolution de Domain Name Service de manière distribuée.

1.4.3 Troisième Génération : Architecture hybride

Cette troisième génération est une hybridation des deux architectures précédentes, Ce type d'architecture utilise plusieurs Peers parmi lesquels on distingue des Super-Peers (SP) appelés aussi super-nœuds qui ont la possibilité d'indexer et de contrôler un ensemble de Peers connectés au système. Un super-Peer est connecté à d'autres super-Peers suivant le modèle de l'architecture décentralisée (en mode P2P), alors que les autres nœuds sont connectés à un super Peer en mode client/serveur pour former un cluster.

On distingue deux types d'architecture hybride : la 1ère dite statique car une machine peut devenir un super-Peer si l'utilisateur le souhaite alors que la 2ème est dite dynamique vu que c'est l'application P2P qui décide automatiquement si la machine cliente fera office de Super-Peer sous réserve de satisfaire certaines conditions (Puissance de calcul, bande passante etc).

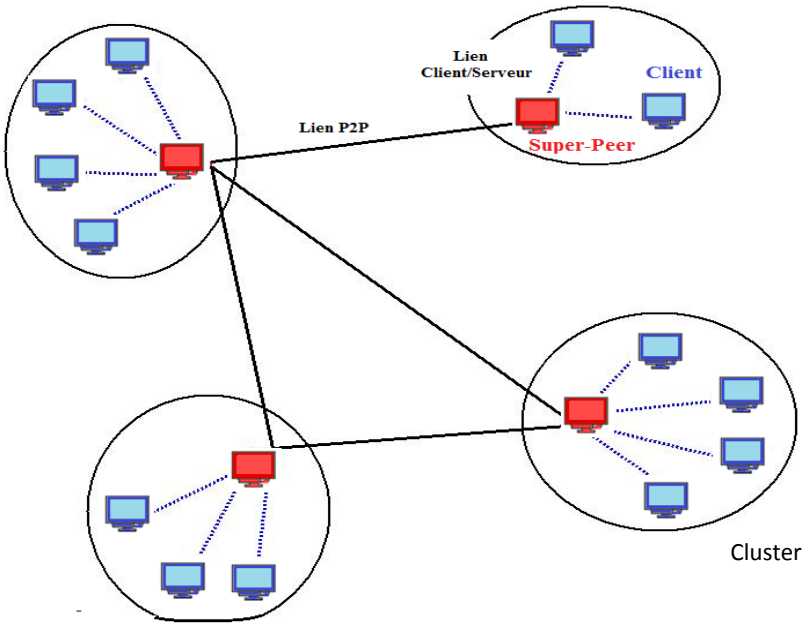


Figure 1.3. Architecture hybride

1.4.3.1 Mode de fonctionnement

- Un Peer qui se connecte au réseau commence par contacter un super Peer pour lui indiquer le nom des fichiers qu'il souhaite partager
- Le super Peer se charge de collecter l'ensemble des informations relatives aux fichiers partagés au sein de son cluster
- Lorsqu'un Peer émet une requête (recherche d'un fichier), son Super Peer se charge de répondre à sa demande en procédant à sa recherche :
  - Si le fichier est disponible au sein du cluster alors une liste des sources est envoyée au client chercheur.
  - Sinon le super Peer contacte d'autres super Peer par flooding (Broadcast) qui répète le même processus de recherche jusqu'à trouver la ressource ou atteindre le seuil limite de recherche matérialisé par un TTL.

1.4.3.2 Avantages

- Le réseau est moins engorgé que dans une architecture centralisée puisque la diffusion des messages ne se fait plus qu'entre les super-peers.

- L'indexation centralisée par cluster permet de veiller à l'intégrité des fichiers partagés et à la fiabilité des nœuds participants.
- L'hétérogénéité des nœuds du point de vue de la bande passante, des capacités de traitement et de la disponibilité est prise en compte dans le choix des super-peers.

1.4.3.3 Inconvénients

- L'existence d'un point de défaillance unique au niveau de chaque cluster.
- L'anonymat n'est plus assuré du fait de la semi-centralisation.
- Le flooding peut être évité en déployant un index structuré, distribué parmi les super-peers mais ceci induirait un coût de maintenance non négligeable.

Exemples : Kazaa [68]

KaZaa est une application P2P basée sur le réseau FastTrack. Bien que les détails de fonctionnement de KaZaa ne soient pas entièrement divulgués, on sait que ce logiciel utilise une technique de super-peer. En outre il s'intéresse aussi au problème de la qualité des nœuds.

Apparu en 2001, ce logiciel a été tout de suite apprécié des utilisateurs. Il alliait la facilité d'utilisation de Napster et la diversité des fichiers disponibles assurée jusqu'ici uniquement par les serveurs FTP. Dans Kazaa le téléchargement se fait à partir de plusieurs sources ce qui permet de pallier au problème de disponibilité temporelle. Un autre atout de Kazaa est la possibilité de reprendre un téléchargement interrompu.

JXTA [69]

JXTA pour juxtapose est un environnement P2P développé en java. Il peut servir de support pour construire différentes applications P2P allant du stockage distribué au calcul réparti. JXTA tire partie de l'hétérogénéité des machines présentes dans le réseau pour construire un réseau hybride, combinant DHT et architecture P2P non structurée hiérarchique. JXTA offre aussi des fonctionnalités pour que les nœuds derrière un pare-feu puissent participer au réseau P2P.

1.5 Taxonomie des systèmes P2P

Le terme peer-to-peer est source de confusion chez la plupart des utilisateurs, Il fait allusion aux systèmes d'échange de fichiers qui ne représentent qu'un cas de figure des types d'applications basés sur un fonctionnement peer-to-peer.

Les architectures peer-to-peer sont utilisées dans plusieurs catégories d'applications qui peuvent être réparties en quatre grandes classes comme le montre la figure suivante : [28,53,54]

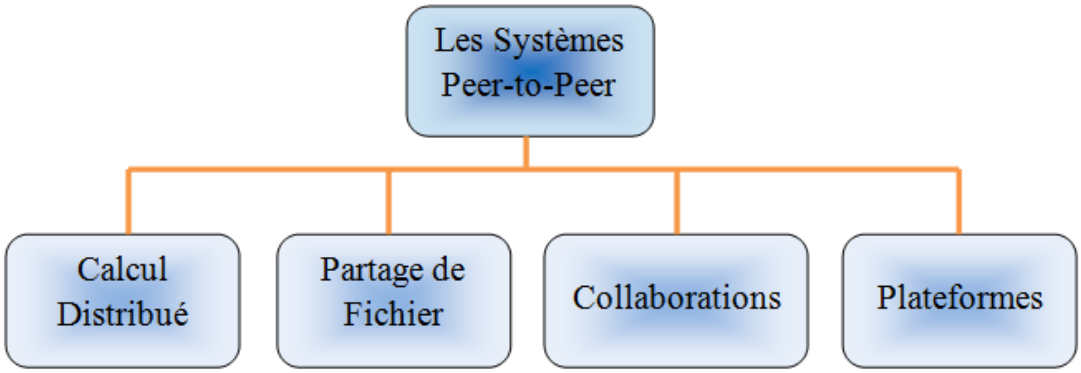


Figure 1.4. Applications du Peer-to-Peer

1.5.1 Calcul distribué

Cette catégorie inclut les systèmes ayant pour but d'utiliser la puissance de calcul (cycles CPU) disponible sur un grand nombre d'ordinateurs. L'idée est d'utiliser les machines inactives d'un réseau pour effectuer des parties d'un calcul découpé en plusieurs unités indépendantes et ré-assemblables. Ces unités sont distribuées sur les différents peers qui en font leurs tâches et retournent les résultats. Un serveur central est souvent nécessaire pour découper et distribuer les tâches puis collecter les résultats des peers afin d'effectuer le calcul final.

1.5.2 Distribution de contenu (partage de fichiers)

Cette catégorie regroupe la plupart des systèmes peer-to-peer actuels. Elle comporte une infrastructure conçue pour le partage de tout type de données. Ces systèmes varient des applications de partage de fichiers relativement simples aux applications plus sophistiquées créant un système de fichiers distribué et permettant un grand nombre d'opérations (stockage et récupération des données, organisation, indexation, recherche, sécurité, disponibilité, mise à jour et gestion des versions).

Nous pouvons distinguer dans cette catégorie deux sous classes suivant les buts et la complexité des applications :

1.5.2.1 Systèmes d'échange de fichiers

Leurs applications principales concernent l'échange de contenu. Ils sont utilisés pour créer un réseau de peers et permettre la recherche et le transfert de fichiers. Ce sont typiquement des applications légères qui ne se préoccupent pas de la sécurité ou de la persistance. Cette classe est fortement connue grâce à Napster, Gnutella, Morpheus, Freenet, Kazaa et BitTorrent.

1.5.2.2 Système de stockage et de publication de contenu

Depuis que les systèmes peer-to-peer sont devenus plus performants, de nouvelles fonctionnalités y sont intégrées. Le but de ces systèmes est de créer un support de stockage distribué qui permettra de publier et stocker des données d'une manière sûre et persistante. Ces données seront utilisées par les peers qui en ont le privilège d'accès.

Les principaux objectifs de ces systèmes sont la sécurité, la fiabilité et la persistance. D'autres fonctionnalités y sont de plus en plus intégrées comme l'anonymat, la gestion de comptes, la résistance à la censure et la gestion du contenu (mise à jour, suppression, gestion des versions…) [54].

1.5.3 Travail collaboratif

Ce type d'application permet à des utilisateurs de collaborer et communiquer en temps réel d'une manière directe sans l'utilisation de serveur central pour la collecte et la transmission d'informations. La messagerie instantanée (MSN, Jabber et ICQ) en est une application populaire.

Groove, Magi, PowerPoint distribué et NextPage sont des applications partagées de travail collaboratif permettant d'interagir et de travailler simultanément sur un même projet distribué. Les jeux en réseaux constituent un autre exemple des applications collaboratives peer-to-peer. Ces jeux sont hébergés sur tous les nœuds et les parties évoluent de manière distribuée sans avoir recours à un serveur central [53].

1.5.4 Plateformes

Les plateformes proposent une infrastructure générique pour développer des applications peer-to-peer en offrant les fonctions de base tel que la gestion de peer, l'attribution d'identifiants, la découverte des ressources, la communication entre Peers et la sécurité. Sun propose sa plateforme JXTA [69] basée sur la technologie Java et Microsoft intègre dans .NET des outils pour développer des applications peer-to-peer.

1.6 Conclusion

Ce chapitre nous a permis de faire un état de l'art des systèmes P2P et de définir certaines notions relatives à ces systèmes. Nous avons aussi exposé les avantages et les inconvénients que présentent les différentes architectures P2P.

Dans le chapitre suivant, nous exposerons un autre type de systèmes à large échelle, à savoir les grilles informatiques.

Chapitre 2

Etat de l'art sur les grilles informatiques

2. Etat de l'art sur les grilles informatiques

2.1 Introduction

Depuis quelques années, l'informatique répartie et les technologies associées sont dans une constante évolution. Les technologies sont de plus en plus abouties et sophistiquées. Les serveurs de calcul et de stockage voient leur rapport qualité prix en constante augmentation en intégrant les nouvelles innovations technologiques. La même observation peut être appliquée aux technologies réseaux et à la bande passante offerte qui tend à devenir illimitée d'un point de vue applicatif. La réponse à ces changements est de passer à un modèle d'informatique répartie permettant d'exploiter pleinement les ressources et les capacités offertes. C'est cette opportunité qui a permis aux grilles de calcul et de données d'émerger comme un important domaine de calcul parallèle et distribué se distinguant par son orientation vers le partage de ressources hétérogènes à grande échelle [22]

En tant que technologie, les grilles informatiques tentent de répondre aux besoins des applications scientifiques caractérisées par un calcul intensif et des volumes de données de l'ordre du PétaOctet. L'idée principale de cette technologie est la distribution des calculs et/ou des données sur des ressources dispersées géographiquement à une large échelle et sous utilisées.

Les grilles informatiques trouvent leurs applications dans divers domaines de la science telles que la biologie et la physique. Elles ont aussi envahi le monde industriel, le travail collaboratif et le e-business. Une telle émergence nécessite des standards offrant une interopérabilité entre systèmes. Un modèle en couche protocolaire a été décrit et des middlewares ont été développés [1,22,49].

2.2 Origine

Le concept de la grille a émergé à la fin des années 90 [1,22,35]. Son inspiration(Grid) vient de la prédominance, la fiabilité et la facilité d'utilisation du réseau de distribution électrique, d'ailleurs le terme Grid trouve son origine dans cette analogie. La transposition de ce concept au monde de l'informatique a bousculé les conventions mais a aussi permis d'envisager une consolidation à grande échelle des ressources informatiques.

Historiquement, les grilles ont commencé par exploiter la puissance de calcul des nœuds présents sur Internet, ce sont les grilles de calcul. L'avènement du monde de l'ordinateur personnel (PC) et le développement d'Internet ont rendu accessible un nombre sans cesse croissant de machines suréquipées en terme de puissance de processeur, capacité de la mémoire et espace de stockage. Or, la plus grande proportion de ces machines sont exploitées par des applications tels que : la bureautique, les jeux ... qui sont peu exigeantes en terme de besoins en performance au vue de la puissance de calcul et de la capacité de stockage dont sont dotées ces machines. Ce constat a eu pour conséquence l'apparition de nombreux projets opérant selon le principe du vol de cycles c'est-à-dire l'exploitation du processeur lorsque celui-ci n'est pas ou peu utilisé (gridscavenging). L'un des projets le plus connu est le projet

Seti@Home [43], qui a pour ambition la recherche de formes de vies extra-terrestre à partir de l'analyse de signaux captés par des radiotélescopes.

En parallèle à l'apparition et au développement des grilles de calcul sont apparues les grilles de données. L'objectif premier de ces dernières est l'agrégation des ressources de stockage et la mise à disposition d'espaces de stockage virtuels pour le partage de données. Parmi les grilles de données, on peut notamment citer le projet BIRN (BiomedicalInformaticsResearch Network) [43], qui relie une multitude de bases de données médicales pour des traitements et diagnostics médicaux avancés.

2.3 Définition

Les grilles informatiques ou GRID (Globalisation des Ressources Informatique et des Données) consistent en un regroupement de machines, connectées entre-elles par un réseau large-échelle (WAN : Wide Area Network), le plus souvent Internet (Figure 2.1).Tous les nœuds connectés à la grille partagent leurs ressources : puissance de calcul (CPU), espace de stockage (disques durs), bande passante réseau. Les grilles s'avèrent d'un intérêt majeur pour l'accès à des ressources distribuées hétérogènes, qu'il s'agisse de ressources de calcul ou de données.

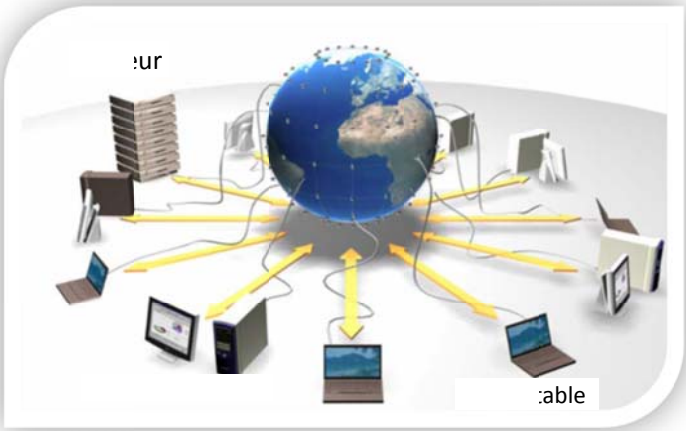


Figure 2.1. Grille informatique

L'objectif est donc de pouvoir mutualiser des ressources, souvent réparties géographiquement sur plusieurs sites, ainsi un utilisateur dispose de la puissance de calcul, des données et de l'espace de stockage dont il a besoin pour lancer des applications, sans se préoccuper de savoir quelles machines sont utilisées.

Le concept de grille informatique a été introduit par I. Foster et Kesselman [1] comme une infrastructure massivement distribuée pour le calcul scientifique. Cette architecture devait permettre de centraliser différentes ressources distribuées géographiquement et reliées entre elles par des réseaux haut débit. Le principe des clusters a donc été repris mais élargi. Les différents composants de la grille sont variés ; on retrouve des clusters, des supercalculateurs

ou même des stations de travail permettant de répondre encore une fois à un besoin de calcul énorme ou de manipulation de grands volumes de données. Une grille devient donc une organisation virtuelle, regroupant plusieurs acteurs mettant en commun une partie de leurs ressources.

En 1998, Ian Foster et Carl Kasselman [35] ont défini les grilles de calcul comme suit : "*Une grille de calcul est une infrastructure matérielle et logicielle qui fournit un accès fiable, uniforme, répandu, à coût réduit à des capacités de calcul importantes*". Les auteurs ont ensuite affiné cette définition en mettant l'accent sur l'aspect " *partage de ressources et organisation virtuelle* ". Ils considèrent notamment que le partage n'est pas un simple échange de fichiers, mais plutôt un accès direct aux ordinateurs, applications, données et autres ressources. Le partage est hautement contrôlé par les fournisseurs de ressources et les consommateurs, en définissant clairement ce qui est partageable, qui a accès aux ressources partagées et dans quelles conditions se fait ce partage. Les individus et/ou les institutions définis par la politique de partage constituent ce que l'on appelle une Organisation Virtuelle(VO) [22,1]. Les utilisateurs pourront être groupés, selon leurs différents intérêts, dans de telles organisations dont chacune possède sa propre politique. Chaque organisation virtuelle mettra à la disposition de ces utilisateurs un ensemble de ressources sous la forme d'une grille.

Les grilles sont passées du cadre académique au cadre industriel, cependant, des confusions apparaissent et on a tendance à parler de grille à chaque fois qu'on dispose d'un réseau dans lequel on partage des ressources. Pour éviter ce type de confusions, Ian Foster, dans son article "What is the Grid ?" [22] manifeste la nécessité d'une définition claire. Il présente trois critères de base pour distinguer une grille d'un autre système distribué. Il définit une grille comme un système qui :

1. Coordonne des ressources sans contrôle centralisé (ressources autonomes).
2. Se base sur des standards.
3. Délivre une Qualité de Service non négligeable (temps de réponse, disponibilité, sécurité, etc.).

Les systèmes qui ne respectent pas les trois points précédents ne sont pas considérés comme grilles.

Comme autres caractéristiques importantes des grilles, nous pouvons mentionner l'hétérogénéité et l'aspect dynamique. L'hétérogénéité peut être matérielle (processeurs, mémoires, réseaux de communication) ou logicielle (systèmes d'exploitation, systèmes de fichiers, environnements de développement, etc.). Les ressources sont dynamiques du fait qu'elles se connectent et se déconnectent de manière imprévisible. La définition suivante résume les points clés cités précédemment : " *Une grille de calcul est une infrastructure constituée de ressources matérielles/ logicielles hétérogènes, distribuées, autonomes, partagées offrant un accès fiable, dynamique, hautement sécurisé à des capacités de calcul et de stockage importantes* "[35].

2.4 Caractéristiques des grilles

Les grilles de calcul possèdent quatre principales caractéristiques [48]:

- **Existence de plusieurs domaines administratifs** : les ressources sont géographiquement distribuées et appartiennent à différentes organisations chacune ayant ses propres politiques de gestion et de sécurité. Ainsi il est indispensable de respecter les politiques de chacune de ces organisations.
- **Hétérogénéité des ressources** : les ressources dans une grille sont de nature hétérogène en termes de matériels et de logiciels.
- **Passage à l'échelle (scalability)** : une grille pourra consister de quelques dizaines de ressources à des millions voire des dizaines de millions. Cela pose de nouvelles contraintes sur les applications et les algorithmes de gestion des ressources.
- **Nature dynamique des ressources** : dans les grilles ce caractère dynamique est la règle et non pas l'exception. Cela pose des contraintes sur les applications telles que l'adaptation au changement dynamique du nombre de ressources, la tolérance aux pannes et aux délais …

Une grille est une collection de ressources dont pourra bénéficier l'utilisateur. Les ressources sont en général réparties en plusieurs types [35] :

- **Cycles processeur** : il existe plusieurs façons d'exploiter les cycles processeurs non utilisés des machines participant à la grille. La première est de faire tourner une application sur une machine distante au lieu du processeur local. La seconde est d'utiliser une application conçue pour tourner ses différentes parties en parallèle sur différents processeurs. La troisième est de faire tourner plusieurs occurrences d'une même application sur différentes machines afin de traiter différentes données plus rapidement.
- **Capacité de stockage :** le second type de ressources disponible dans une grille est la capacité de stockage. 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.
- **Equipements spéciaux et logiciels à licence élevée :** certains logiciels dont les prix de licence sont élevés seront présents en quelques exemplaires seulement dans la grille. Cette grille permettra donc en les exposants à beaucoup d'utilisateurs, une meilleure utilisation de ces logiciels. Il en va de même pour certains équipements spéciaux comme les microscopes électroniques et les appareils médicaux.

2.5 L'évolution des grilles informatiques

Depuis leur apparition, les grilles ne cessent d'évoluer et le monde industriel tout comme le monde scientifique est attiré par leur apport. Cette évolution est décrite sur trois générations :

2.5.1 Première génération : interconnexion de supercalculateurs

Les premiers projets de grilles de calcul s'intéressaient à relier des super calculateurs pour répondre aux besoins de calcul de haute performance, qu'on appelle Métacomputing. Cette première génération est caractérisée par deux projets de référence, à savoir FAFNER et I-

WAY.[49] qui ont, chacun à sa façon, influencé l'évolution de certains des projets technologiques clés, en cours, afférents à la grille :

Le projet FAFNER (Factoring via Network-EnabledRecursion) traite le problème de factorisation de grands nombres qui est très couteux en temps de calcul. Pour cela, des algorithmes de factorisation parallèle ont été développés. En 1995, un consortium mis en place par le laboratoire Bellcore de l'université Syracuse (New York) a initié le projet de factorisation parallèle via le web.

Le projet I-WAY (Information Wide Area Year) fut également conçu en 1995. Il intègre plusieurs réseaux situés sur 17 sites différents des Etats-Unis. Une des innovations de ce projet fut la mise au point d'un système de courtage des ressources de calcul. I-WAY a fortement influencé la conception du projet Globus [71], qui est à la base de la plupart des activités relatives à la grille En effet, les outils développés pour monter ce projet ont été intégrés au projet Globus.

2.5.2 Seconde génération : utilisation de middleware

Les années 2000 ont connu une véritable émergence des grilles de calcul grâce à leur maturité. Une grille ne se réduit plus à l'interconnexion des supercalculateurs mais relie également de simples machines géographiquement dispersées pour supporter diverses applications nécessitant des capacités de calcul et/ou d'espace de stockage qui ne peuvent pas être supportées par des machines classiques. De plus les projets développés dans la première génération ont montré la nécessité de concevoir un middleware de grille pour prendre en charge et masquer à l'utilisateur l'hétérogénéité des environnements informatiques de chaque nœud du système. Ainsi la deuxième génération des grilles a apporté un outil essentiel d'automatisation dans l'usage de grilles, le gestionnaire ou le courtier de ressources (resources broker). Ce courtier dispose de l'état de l'ensemble des ressources offertes par la grille, ce qui lui permet d'allouer celles demandées par les scripts de lancement des travaux, voire de les réserver préalablement, avant ordonnancement puis exécution. Cette génération fut par ailleurs, particulièrement marquée par l'apparition de middlewares spécifiques aux grilles. Beaucoup de middlewares ont été proposés et le projet Globus [71] est actuellement la norme de base pour les middlewares des grilles de calcul.

2.5.3 Troisième génération : approche orientée service

Le développement de nouvelles applications pour les grilles a suscité le besoin de définir des composants qui pourraient ensuite être utilisés comme briques de base dans un processus de développement. C'est ainsi qu'il a été proposé d'organiser les composants selon leurs fonctionnalités pour constituer un service. On parle d'architecture orientée service ou OGSA (Open Grid Service Architecture) [49]. Pour de gros besoins en matière de stockage, cette génération a aussi connu une extension vers le monde du e-business.

2.6 Taxonomie des grilles

On peut classer les grilles informatiques selon deux aspects : un aspect orienté topologie ou un aspect orienté service, ainsi les grilles sont classées d'un point de vue topologique en

trois types par ordre croissant d'étendue géographique et de complexité : Intragrilles (Intragrids), Extragrilles (Extragrids) et Intergrilles (Intergrids). [48]

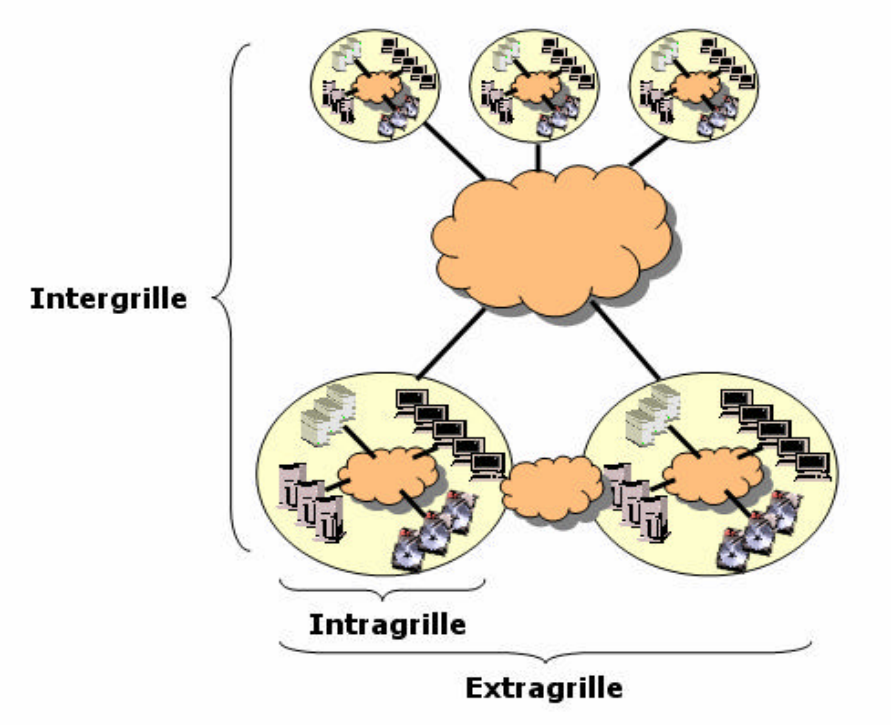


Figure 2.2. Différentes topologies des grilles

2.6.1 Intragrille (par analogie avec Intranet) :

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

2.6.2 Extragrille (par analogie avec Extranet) :

Une extragrille étend le modèle en agrégeant plusieurs intragrilles. Les principales caractéristiques d'une telle grille sont la présence d'un réseau d'interconnexion hétérogène haut et bas débit (LAN / WAN), de plusieurs domaines de sécurité distincts, et d'un ensemble plus ou moins dynamique de ressources. Un exemple d'utilisation de ce type de topologie est le modèle "Business-to-Business" (B2B) entre entreprises partenaires [48].

2.6.3 Intergrille (par analogie avec Internet) :

Une intergrille consiste à agréger les grilles de multiples organisations, en une seule grille. Les principales caractéristiques d'une telle grille sont la présence d'un réseau

d'interconnexion très hétérogène haut et bas débit (LAN / WAN), de plusieurs domaines de sécurité distincts et ayant parfois des politiques de sécurité différentes et même contradictoires, et d'un ensemble très dynamique de ressources. Les intergrilles seront souvent mises en œuvre lors de grands projets industriels (conception d'un avion par un consortium aéronautique par exemple) ou scientifiques (modélisation de protéines) où plusieurs organisations seront amenées à participer.

D'un point de vu service une grille peut être classée selon le service qu'elle offre, ainsi nous pouvons distinguer trois types de grilles :

2.6.4 Grilles de calcul :

Elles permettent de distribuer des calculs sur des ressources réparties pour bénéficier d'une plus grande puissance de calcul. Ces grilles sont en général formées de clusters et de serveurs, mais parfois des PC de bureau sont utilisés. Ces grilles deviennent de plus en plus nécessaires pour faire face à l'augmentation constante des besoins en puissance de calcul. Ainsi, la complexité des systèmes étudiés en recherche scientifique et dans l'industrie (thermique, structure, fluides, biologie, etc.) induit des besoins de puissance de calcul pouvant atteindre plusieurs téraflops. Les projets nécessitant de telles puissances de calcul sont par exemple les modèles météo et les études sur le changement climatique global, les simulations de matériaux, les simulations et outils de conceptions en aéronautique, automobile, chimie ou nucléaire, certains calculs de risques dans le domaine de la finance

2.6.5 Grilles d'information :

Elle permet le partage de l'information à travers un réseau. L'exemple le plus caractéristique de ce type de grille est le Web, il offre la possibilité d'accès à de grandes masses de données disséminées à travers le monde.

2.6.6 Grilles de données (stockage) :

Ce type de grille permet le stockage de très grands volumes d'information. Elles sont souvent, mais pas toujours, combinées avec des grilles de calcul. Beaucoup d'applications en ingénierie scientifique nécessitent l'utilisation de grands volumes de données exprimés en Téra ou péta octets. Nous assistons ces dernières années à une augmentation rapide du volume d'information à stocker et à traiter. Il apparait donc impossible de stocker ces dernières sur une seule et même machine et on a donc souvent recours à une grille de données.

Une grille de données ne permet pas seulement de découper des fichiers et de les ranger dans plusieurs machines. Elle doit mettre en place des mécanismes de recherche, d'indexation, d'intégrité et de sécurité pour assurer un accès fiable et permanent. Il faut aussi que la répartition des données soit transparente pour l'utilisateur final. En effet, il est impossible de savoir précisément où va être stockée chaque donnée. Un fichier peut être découpé en plusieurs morceaux et réparti sur plusieurs machines. Il peut aussi être répliqué sur plusieurs entités du réseau pour permettre une meilleure disponibilité. Il faudra alors avoir un mécanisme qui répartit la charge sur chaque serveur. La combinaison d'utilisateurs de ressources et de données

volumineuses et largement distribuées a nécessité l'élaboration d'infrastructures de gestion des données qui, jusque-là, n'existaient pas. L'objectif initial était d'assurer l'exécution fiable et efficace de requêtes en fournissant la gestion de teraoctets de caches, le transfert de gigaOctets de données à travers des réseaux distants, le co-ordonnancement des calculs et des transferts de données, l'estimation de performance afin de permettre la sélection de réplicas. [45]

Selon Reagan Moore [43], les capacités offertes par les grilles de données sont essentielles pour automatiser les processus de préservation, limiter les risques de perte de données à travers la reproduction de composants numériques, assurer l'association permanente de l'identité et l'intégrité des méta données avec les enregistrements, et en supporter la récupération et l'accès. En même temps, les grilles de données sont définies comme des infrastructures qui permettent de gérer des entités numériques stockées dans n'importe quel type de système de stockage, tout en fournissant l'accès à travers une très grande variété de mécanismes d'accès. Cette possibilité d'interagir avec plusieurs types de systèmes de stockages et d'accès aux données forme le cœur du support de grille de données.

La définition de Reagan Moore exhibe un point fondamental des caractéristiques des grilles de données : la transparence. Ainsi, la grande hétérogénéité des supports de stockage, des méthodes d'accès aux données et la manière dont elles sont identifiées impliquent de définir une organisation qui soit indépendante des architectures de gestion de données mises en place par chaque organisation virtuelle [43].

2.7 Architecture d'une grille :

L'architecture d'une grille est organisée en couches. Une couche est une abstraction représentant un ensemble de services. La figure 2.3 illustre le modèle en couches de l'architecture d'une grille.

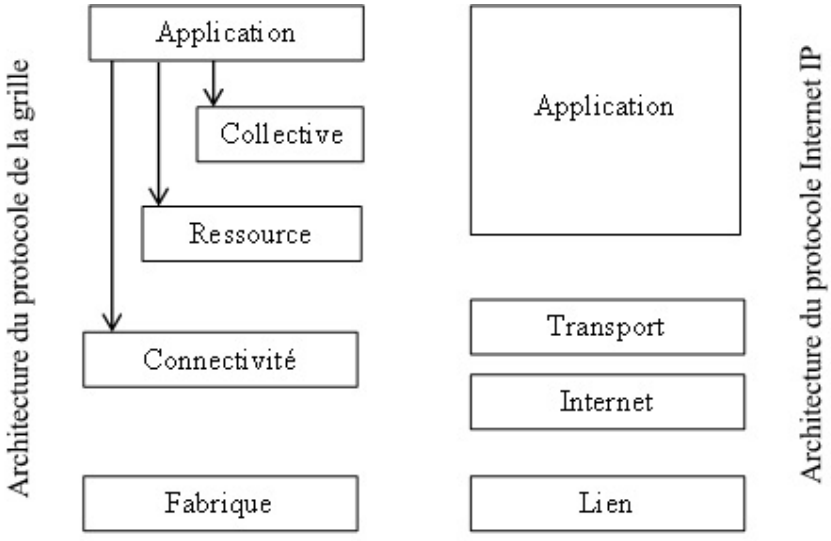


Figure 2.3. Modèle en couches de l'architecture d'une grille

- « Fabrique » : Cette couche fournit les ressources partagées, qui peuvent être des calculateurs, serveurs de stockage, instruments de mesures, bases de données …etc . Ces ressources exécutent des logiciels tels que des systèmes d'exploitation, systèmes de gestion de bases de données, systèmes de soumission de tâches, etc.
- « Connectivité » : Cette couche implémente les protocoles de communication pour l'échange de données ainsi que les protocoles d'authentification nécessaires au transfert sécurisé dans la grille
- « Ressources» : Cette couche définit les protocoles pour la négociation, la supervision le contrôle et la compatibilité pour le partage des ressources individuelle. Elle fournit aussi les mécanismes de sécurité nécessaires à la protection des ressources
- « Collective » : Cette couche est chargée de la coordination des ressources. Les services décrits dans cette couche ne sont pas associés à une ressource spécifique mais à un ensemble de ressources. En effet cette couche est responsable de la Co-allocation, l'ordonnancement et la réplication des données, des services et des protocoles
- « Application » : Cette couche représente l'ensemble des applications utilisateurs qui opèrent dans l'environnement de la grille. Ils sont de nature variée : projets scientifiques, médicaux, financiers, ingénierie, etc.

De très nombreux outils existent implémentant tout ou une partie des différentes couches d'une grille de données. Il est relativement difficile de faire une classification précise de chacun des projets en particulier parce que le nombre de recherches dans le domaine est très grand. Cependant, une donnée doit pouvoir être stockée, accédée, déplacée, répliquée de manière transparente. Une autre approche de la description de l'architecture de la grille existe, elle met l'accent sur les constituants de la grille plutôt que sur les services offerts comme on peut le voir sur la figure 2.4

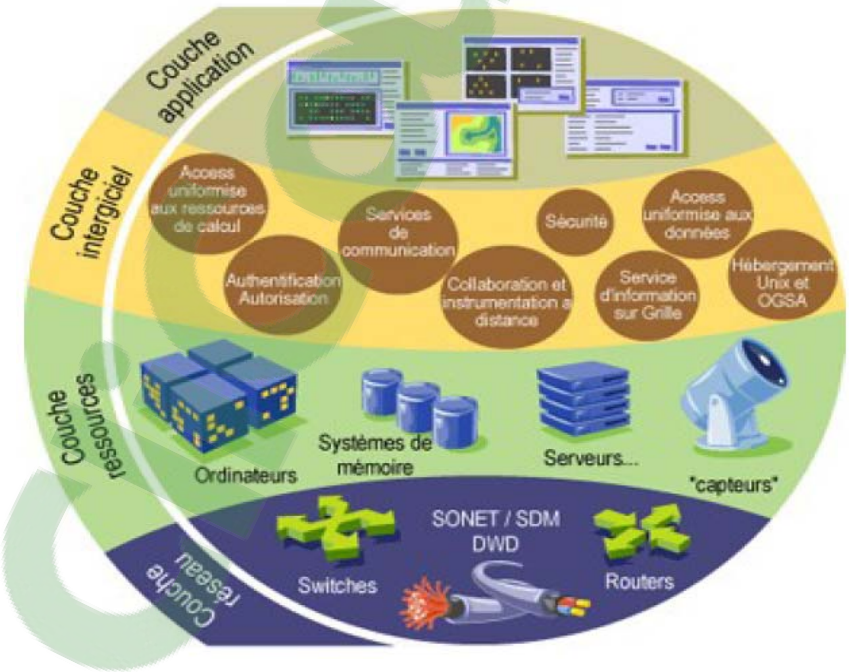


Figure 2.4. Différents constituants d'une grille

2.8 Applications des grilles

Au début des années 2000, Ian Foster et Carl Kesselman ont identifié les domaines d'application des grilles et les ont classés en cinq catégorie : [3,35]

- **Supercalculateur réparti (Distributed Supercomputing)**

Une grille de calcul est l'incarnation des grilles dans le monde informatique. En effet, les premières grilles mises en place avaient pour objectif de répondre à des exigences en puissance de calcul, qui ne pouvaient pas être satisfaites par les ressources appartenant à des individus ou à des institutions. Une grille peut agréger une importante quantité de ressources afin de fournir la puissance de calcul nécessaire pour de nombreuses applications. Ces ressources sont fédérées et présentées comme un système unifiée permettant de résoudre des calculs à grande échelle. Des disciplines aussi diverses que la météorologie, la science des particules ou la biologie sont des exemples d'applications qui requièrent de grandes capacités de calcul.

- **Calcul haut-débit (High-Throughput Computing)**

Une grille de calcul peut être utilisée pour ordonnancer en parallèle une importante quantité de tâches indépendantes les unes des autres. Comme exemples d'applications nous pouvons citer la recherche de clés cryptographiques, les simulations de molécules et l'analyse du génome.

- **Calcul à la demande (On-Demand Computing)**

Une grille de calcul pourra fournir les ressources pour satisfaire les demandes à court terme d'une application et que les ressources locales ne sont pas en mesure d'assurer (cycles processeur, stockage). Le défi principal pour les concepteurs de telles grilles est la nature dynamique et aléatoire des demandes faites par les utilisateurs qui peuvent constituer une large population.

- **Calcul Collaboratif (Collaborative Computing)**

Cette classe d'applications inclut les applications d'interaction entre usagers dans des environnements de simulation en temps-réel, elle permet à des utilisateurs de collaborer et communiquer en temps réel d'une manière directe sans l'utilisation de serveur central pour la collecte et la transmission d'informations (Conception et interaction avec un nouveau moteur d'avion, conception d'un plan urbain, La messagerie instantanée, les applications de travail partagé.

- **Génération, traitement et stockage d'énormes quantités de données (Data-intensive Computing)**

Dans de telles applications, une grille de calcul pourra absorber et stocker une importante quantité d'informations générées. Comme exemple d'applications nous pouvons mentionner la production d'une carte de l'univers, la prévision météorologique à long terme, les simulations quantiques.

2.9 Grille vs système P2P ?

Les termes « grille »s et « systèmes P2P » sont utilisés chez certains auteurs de façon interchangeable. Certains auteurs classent les systèmes P2P comme une sous-classe décentralisée des grilles. Pourtant, il existe une distinction entre les deux systèmes quoique leurs domaines d'application se rejoignent (cf chapitre 1 section 5 et chapitre 2 section 8).

Les systèmes P2P sont des architectures qui facilitent le passage à l'échelle et la disponibilité des ressources avec un faible coût. Cependant il est à noter qu'en premier lieu les nœuds du système sont totalement autonomes et donc ils peuvent choisir de partager ou non leur CPU et leur capacité de stockage. Cette autonomie leur confère aussi le choix de rejoindre ou quitter le système à tout moment. Cela a pour effet de compromettre la capacité de calcul totale et réelle du système mais aussi la disponibilité des ressources (informations, capacité de stockage, etc.). En second lieu, le système de communication utilisé est de faible bande passante (en général Internet), ce qui peut créer une surcharge du système et une latence plus élevée que dans les clusters. En troisième lieu, l'hétérogénéité des ressources et de leur système de sécurité pose un problème de protection des machines interconnectées. Enfin, l'absence d'infrastructures de contrôle sur les systèmes P2P rend ces derniers moins pratiques pour prendre en compte certains types d'applications qui exigent une grande qualité ou des services transactionnels. D'ailleurs, Ian Foster [22] considère que les systèmes basés sur Internet, ne constituent pas une infrastructure de type grille car ils ne coordonnent pas les ressources pour offrir une bonne qualité de service (Qos).

Contrairement aux systèmes P2P, une grille garantit des qualités de service non triviales, c'est à-dire qu'elle répond adéquatement à des exigences (accessibilité, disponibilité, fiabilité) compte tenu de la puissance de calcul ou de stockage qu'elle peut fournir.

Les grilles informatiques sont caractérisées par une forte hétérogénéité et une grande dynamicité. En effet, les machines d'une grappe sont reliées par un réseau gigabits alors que les sites sont liés par un réseau WAN, dont la latence peut aller jusqu'à 100 millisecondes. De là, nous pouvons noter une différence importante de la latence entre deux machines d'un même cluster et celle entre deux machines de deux clusters différents. En outre, chaque cluster est administré de manière autonome et par conséquent les politiques de sécurité varient d'un cluster à l'autre. Un autre exemple d'hétérogénéité relève de la composition interne d'un cluster. Chaque cluster est libre de choisir le type de processeur de ses machines (Intel, AMD, IBM, ...), la capacité de stockage et le réseau d'interconnexion entre machines (Gigabit Ethernet, Infiniband, ...). Enfin l'infrastructure d'une grille est composée d'un nombre important de sites et de machines qui sont susceptibles de tomber en panne à tout moment. A cela s'ajoute le fait que de nouveaux sites peuvent être ajoutés ou retirés de la grille sans trop impacter le fonctionnement de la grille. De par leur hétérogénéité, leur gestion décentralisée et leur taille, les grilles sont des infrastructures très complexes à mettre en œuvre. La transparence n'est assurée qu'à moitié parce qu'il faut avoir une information précise des ressources dans un site pour pouvoir distribuer les tâches convenablement. Cependant à l'intérieur d'un site, la machine qui effectue concrètement la tâche n'est pas connue en général.

Enfin, les grilles et les systèmes P2P ont des racines très différentes qui expliquent leur différence d'organisation et de problématique.

2.10 Conclusion

Les grilles constituent un enjeu fondamental dans l'évolution des architectures des systèmes informatiques. Elles ouvrent de nouvelles voies importantes au développement des technologies de l'information et de la communication.

L'objectif d'une grille est d'optimiser des infrastructures informatiques existantes et de généraliser l'accès et le partage de ressources de calcul et de stockage à grande échelle. En effet, la puissance conjuguée des ressources matérielles et logicielles disponible au sein d'une grille, offre une capacité très importante pour gérer de grandes quantités de données.

Comme les grilles sont des infrastructures dynamiques où les différents nœuds peuvent se connecter et se déconnecter à tout moment, le problème de disponibilité des données est alors discuté. Pour assurer une telle disponibilité, la technique la plus utilisée est alors la réplication qui consiste à multiplier ces données sur plusieurs sites. Nous discuterons cette technique dans le chapitre suivant.

Chapitre 3
La réplication dans les grilles de données

3. Réplication dans les grilles de données

3.1 Introduction

Pour de nombreuses applications scientifiques ayant recours aux systèmes à large échelle, les quantités de données nécessaires aux traitements de ces applications sont très importantes. De telles quantités de données impliqueraient de très grands délais de transfert si seule une copie de ces données était stockée. De là, le temps de réponse et la disponibilité des données deviennent les défis principaux à adresser. Afin de répondre à ces défis, une technique importante est de répliquer les données dans différents sites, de sorte qu'un utilisateur puisse accéder aux données d'un site de sa proximité.

3.2 Technique de réplication des données

3.2.1 Définition

La réplication est aujourd'hui largement utilisée dans les grilles. Elle consiste à créer plusieurs copies d'un même fichier sur des ressources de stockage différentes de la grille [3] en mettant en œuvre un processus de création et de placement des copies d'entités logiciels. La phase de création consiste à reproduire la structure et l'état des entités répliquées, tandis que la phase de placement consiste à choisir, en fonction des objectifs de la réplication, le bon emplacement de cette nouvelle reproduction. Cette technique permet d'améliorer la fiabilité, la tolérance aux pannes, l'accessibilité et d'augmenter la disponibilité des données, la charge étant alors répartie sur les différents nœuds possédant une réplique [5,11,31].

3.2.2 Création des répliques

Ranganathan et Foster définissent les quatre questions auxquelles une stratégie de création de répliques doit répondre [5,31**]**:

- ✓ Quand créer les répliques ? moment de la réplication.
- ✓ Quels fichiers doivent être répliqués ? choix de l'entité à répliquer.
- ✓ Où les répliques doivent-elles être placées ? placement des répliques.
- ✓ Comment une copie est-elle crée ? manière de répliquer une entité.

1. **Moment de la réplication :** Pour répondre à la question *quand* ? deux solutions sont possibles [29] ;
    - **Réplication statique :** les répliques persistent jusqu'à ce qu'elles soient effacées par l'utilisateur du nœud sur lequel elles sont hébergées ou que leurs durées de vie respectives expirent. L'avantage de ce schéma est sa simplicité, son inconvénient est sa non-adaptabilité aux changements de comportement des participants.

- **Réplication dynamique :** contrairement à la réplication statique, la réplication dynamique crée et supprime automatiquement les copies selon l'évolution des demandes des utilisateurs. L'avantage est la réduction des points d'engorgements et l'équilibrage de la charge. L'inconvénient observé est l'induction de coûts supplémentaires causés par l'évaluation en temps-réel du trafic réseau pour prendre les décisions de réplication.
  Selon le moment de la réplication, on distingue :
  - *Réplication à la demande* : la réplique est créée suite à la demande d'un client.
  - *Réplication périodique* : elle est indépendante des requêtes des clients. Son but est de permettre la gestion automatique de répliques avec des stratégies adaptées aux comportements des clients. Le processus de réplication est déclenché à chaque intervalle de temps (période).

2. **Choix de l'entité à répliquer :** Pour répondre à la question *quoi* : les données répliquées sont généralement de deux types : des fichiers ou des objets. Les objets peuvent être composés d'un ensemble de fichiers distribués (on les appelle aussi collection). Selon les stratégies de réplication, les données à répliquer, peuvent être les plus populaires ou encore les plus fréquemment accédées.

3. **Placement des répliques :** Pour répondre à la question *où* : les stratégies de placement de répliques doivent tenir compte du fait que les sites potentiels :
   a) ne possèdent pas déjà de réplique de la donnée ;
   b) possèdent l'espace de stockage suffisant ;
   c) sont à une distance raisonnable en termes de temps de transfert.

4. **Manière de répliquer une entité :** Pour répondre à la question *comment* : Le processus de création de copie dépend de la structure et de l'état de l'entité à répliquer. La structure de l'entité peut être indivisible ou composée, alors que l'état peut être constitué de données, de code et éventuellement d'un état d'exécution.

Les problèmes de coûts sont au centre des stratégies de réplication**.** Un enjeu majeur de la réplication est la réduction de la latence d'accès ainsi que la consommation de bande passante.

3.3 Avantages et inconvénients de la réplication

Le recours à la technique de réplication procure certains avantages que nous pouvons énumérer comme suit :

- *Amélioration de la fiabilité et de la sûreté de fonctionnement*,
  - Si une copie tombe en panne, il est toujours possible d'obtenir les données à partir d'une autre copie.
  - La redondance permet une meilleure protection contre la corruption de fichiers.

- *Amélioration des performances du système*,
  - La charge de travail est divisée entre plusieurs serveurs.
  - L'extensibilité géographique en rapprochant les serveurs des clients.
  - Disponibilité de données, Les données sont disponibles localement et non plus par le biais de connexions des réseaux. Elles sont accessibles localement, même en l'absence de toute connexion à un serveur central, de sorte que l'utilisateur ne soit pas coupé de ses données en cas de défaillance d'une connexion réseau.
  - La réplication améliore les temps de réponse des requêtes d'interrogation. Les requêtes sont traitées sur un serveur local sans accès à un réseau distant, ce qui accélère le débit.
  - Réduction du temps de latence à l'accès et la consommation de la bande passante.

Néanmoins, la réplication peut présenter quelques problèmes tels que :

- Le coût de mise à jour : La modification de la copie originale engendre la mise à jour de l'ensemble des copies. Le coût de mise à jour dépend donc du nombre de copies. De plus, la propagation des mises à jour nécessite généralement un certain temps ce qui entraine une période de repos (pas de services). Une requête qui arrive pendant cette période sera donc retardée.
- Maintenance de la cohérence des copies par rapport à la donnée de référence : Toutes les répliques doivent avoir le même état pour que les services fournis par la grille soient toujours fiables, ce qui implique un travail de gestion énorme.

3.4 Stratégies de réplication

L'objectif principal d'une technique de réplication est d'améliorer la disponibilité des données et les performances du système.
Dans la littérature, les stratégies de réplication sont classifiées selon plusieurs critères, on peut les classer selon le modèle de grille adoptée. Il existe des stratégies dédiées aux grilles hiérarchiques et d'autres dédiées aux grilles P2P ou hybrides. Un autre type possible de classification se base sur la prise de décision, on distingue dans ce cas des stratégies centralisées et décentralisées, la réplication centralisée est déclenchée par un site central qui collecte toutes les informations pour déterminer les fichiers à répliquer et les sites où les placer. En revanche, pour la réplication décentralisée, tous les sites de la grille gardent un historique sur les fichiers demandés pour décider de ceux à répliquer ou à supprimer localement. La périodicité est aussi un critère sur lequel on peut classer les stratégies de réplication. Il définit le moment du déclenchement du processus de réplication. Dans ce sens, on peut observer des stratégies de réplication périodiques où la réplication est déclenchée à chaque intervalle de temps et des stratégies non périodiques où la réplication a lieu à la demande.

Les principales caractéristiques des algorithmes de réplication sont les critères pour la sélection de données appropriées pour la réplication et la sélection de sites appropriés pour accueillir les répliques. Si un nœud décide de répliquer tous ses objets dans d'autres nœuds, il

va augmenter la charge du réseau et en plus les répliques doivent être maintenues dans les sites qui sont à proximité du nœud source afin d'accroître les performances.

Dans la suite de cette section, nous détaillons les techniques de réplication classées selon : i) la politique de sélection des sites, ii) la distribution des répliques, iii) les techniques pour les architectures super-peer. Cependant, certaines techniques, qui appartiennent à une catégorie, peuvent posséder des propriétés d'une autre catégorie. Nous présentons ci-dessous ces différentes catégories :

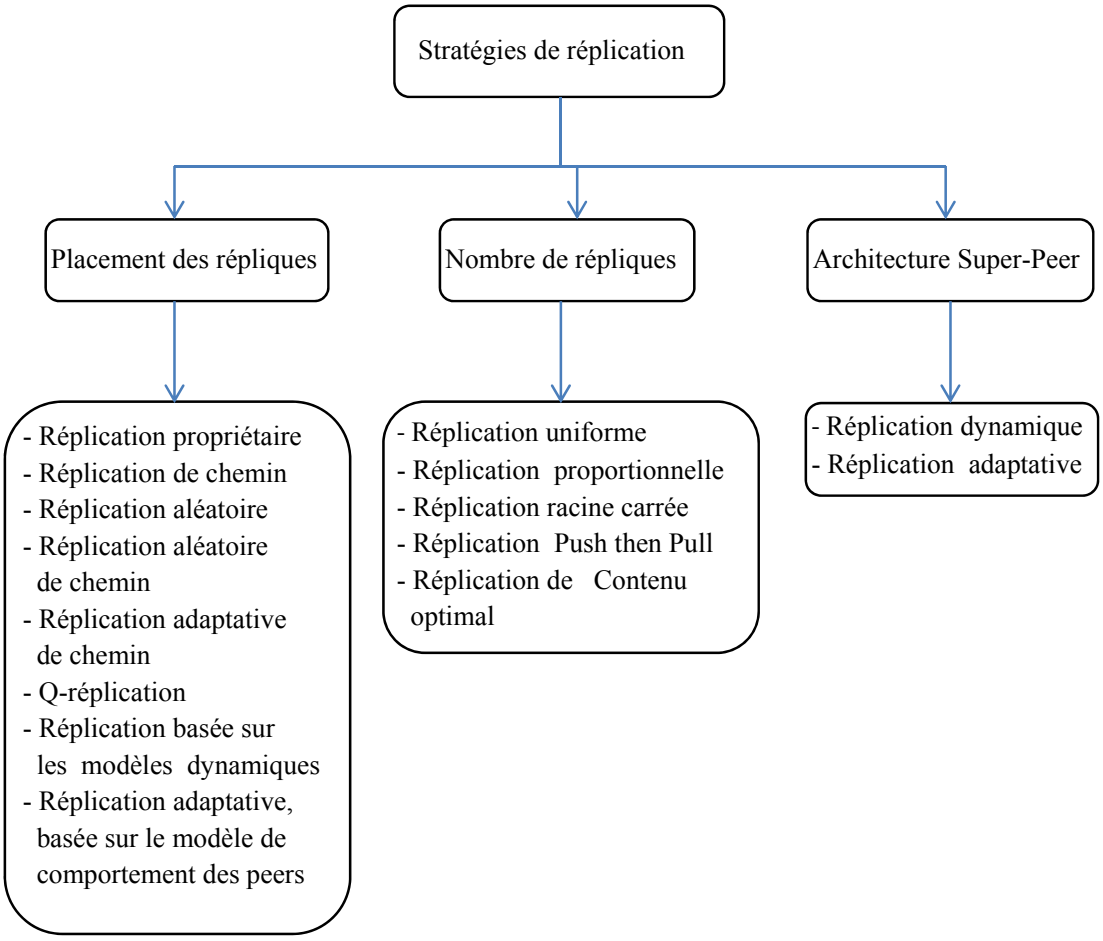


Figure 3.1. Classification des stratégies de réplication

3.4.1 Réplication basée sur le placement des répliques

Toute stratégie de placement doit répondre à trois problématiques principales : Quand faut-il créer des répliques ? Quelles données doivent être répliquées ? Et où doit-on placer les répliques ? Ce qui conduit à différentes stratégies de placement des répliques, qui diffèrent d'un système à un autre.

Mathématiquement le placement des répliques est un problème NP-complet ayant une fonction de coût à minimiser (selon le besoin) et un certain nombre de contraintes (capacité de

stockage, charge maximale supportée, bande passante du nœud, nombre de répliques, délai de réponse, taux de disponibilité minimum,…) [9].

Les approches dans ce domaine reposent sur la construction d'un réseau virtuel sur le réseau physique (typiquement Internet) et qui détermine la notion de voisinage. Plusieurs stratégies peuvent être utilisées dans ce contexte :

3.4.1.1 Réplication du propriétaire [7]

Cette méthode réplique un objet seulement lorsqu'il est sollicité, ainsi le nombre de réplique s'accroit proportionnellement au nombre de requêtes, mais les performances de recherche pour un fichier donné sont limitées à cause du temps de propagation des répliques sur le réseau.

3.4.1.2 Réplication aléatoire [7]

Cette stratégie consiste à placer les répliques aléatoirement sur des nœuds dispersés au lieu de suivre un ordre topologique. Elle est très efficace pour la recherche d'objet, mais un nœud doit avoir une connaissance de tous les nœuds du réseau logique. Ceci s'avère difficile puisque les nœuds n'ont en général aucune connaissance de leurs voisins.

3.4.1.3 Réplication de chemin [74]

La réplication de chemin place les répliques sur tous les nœuds se trouvant sur le chemin de l'objet demandé jusqu'au nœud faisant la requête. Le nombre de répliques créées peut être très grand, ce qui peut poser un problème du point de vue performance du système, surtout en ce qui concerne les capacités de calcul et de stockage des différents nœuds.

3.4.1.4 Placement dans le voisinage topologique [24]

Le nœud qui insère la donnée place les K répliques de la donnée chez ses K plus proches voisins. L'inconvénient est que nous avons une forte localité des répliques qui perd son intérêt dès que les requêtes sur la donnée proviennent de nœuds disséminés dans tout le système. Notons que la connexion/déconnexion d'un nœud affecte ses voisins sur un disque de rayon K. Plus K est élevé plus le surcoût engendré pour la maintenance des tables de routage est aussi élevé.

3.4.1.5 Placement dans un voisinage stationnaire [24,75]

Elle consiste à définir une métrique plus complexe sur la topologie. Cette métrique offre l'avantage de voisinages stationnaires, c'est-à-dire de voisinages qui varient peu au regard de la dynamique du système. Les répliques sont placées dans ce voisinage. L'idée de voisinage stationnaire est intéressante surtout pour l'équilibrage de charges. Elle offre un niveau de granularités auquel on réduit le mouvement des charges.

3.4.1.6 Réplication aléatoire de chemin [16,24]

Elle est basée sur la réplication de chemin et la réplication aléatoire. Chaque nœud intermédiaire détermine aléatoirement, s'il créée une réplique ou non, en se basant sur un ratio prédéterminé de probabilité de réplication. Le ratio de réplication est le ratio des répliques

créées sur tous les nœuds intermédiaires du chemin de l'objet recherché. Donc pour cette approche, les objets demandés sont répliqués sur chaque nœud intermédiaire avec une certaine probabilité, qui est la même pour tout nœud se trouvant dans le réseau, ceci diminue la charge sur les nœuds, qui sont fréquemment localisés sur les chemins de transmission des données.

3.4.1.7 Réplication de chemin adaptative [16]

C'est une alternative à la réplication de chemin aléatoire, qui détermine la création de répliques de façon adaptative, en se basant sur la capacité de stockage. Cette approche détermine la probabilité de réplication sur chaque nœud en utilisant un ratio de réplication prédéterminé et le statut des ressources. Ce qui s'avère être mieux que la réplication de chemin aléatoire en nombre moyen de sauts. La réplication de chemin adaptative peut améliorer aussi l'équilibrage de charge, en utilisant les connaissances locales sur les ressources disponibles de chaque nœud.

Ces deux dernières stratégies, permettent à chaque nœud de choisir entre créer ou non des répliques en fonction de ses connaissances locales, sans aucune connaissance global du réseau. En plus les répliques sont distribuées équitablement sur tous les nœuds, indépendamment de leur degré et ratio de réplication. On remarque aussi que la réplication adaptative donne une meilleure performance de recherche que la réplication aléatoire.

Cependant ces stratégies ne se basent que sur les probabilités. Et dans la réplication adaptative, d'autres attributs comme la bande passante, la connectivité… ne sont pas pris en compte pour la sélection des nœuds cibles pour héberger les répliques.

3.4.1.8 Q-Réplication [13,77]

La Q-Réplication utilise le Q-apprentissage (Q-learning) pour une réplication autonome d'objets qui se fait par un nœud selon ses performances. Ainsi malgré les changements constants des connexions, les objets sont toujours hautement disponibles.

La Q-Réplication maintient une Q-table contenant les IDs des nœuds correspondant à des Q-Valeurs de chaque nœud. Une Q-valeur représente la contribution passée d'un nœud dans la réplication. Une partie de la réplication consiste à envoyer un message du nœud cible au nœud destiné à héberger la réplique. Le message est renvoyé en ajoutant des paramètres tels que, la bande passante, la capacité de stockage… La Q-valeur est mise-à-jour ainsi.

Le processus de Q-réplication sélectionne les objets cibles pour la réplication en se basant sur leur popularité. Un nœud qui se déconnecte fréquemment et qui maintient une faible valeur de bande passante retourne de petites valeurs. Ainsi les nœuds avec de bonnes performances auront des Q-valeurs élevées et ceux avec de faibles performances seront réduits. Une réplique peut être remplacée si un nouvel objet plus populaire nécessite de l'espace, en prenant en compte le temps durant lequel l'objet a été ajouté. La Q-réplication distribue ainsi les objets populaires aux nœuds à performances élevée dans le réseau afin d'améliorer la disponibilité des objets et améliorer la tolérance aux fautes, sans dépendre de mécanismes de recherche. Mais la maintenance des Q-tables peut parfois entraîner une surcharge du réseau.

3.4.1.9 Réplication basée sur des modèles dynamiques

C'est un modèle décentralisée [2], utilisé pour créer dynamiquement des répliques dans un système peu fiable. Chaque nœud du système possède un modèle du système de stockage afin de déterminer le nombre de répliques nécessaires au maintien d'une certaine disponibilité et il applique ce modèle aux connaissances qu'il a de l'état du système et de l'état de la réplication de ses fichiers afin de déterminer quand et où de nouvelles répliques devraient être créées. Chaque nœud du réseau est autorisé à créer des répliques de ses fichiers. Le modèle de performances compare les coûts et les gains des répliques créées d'un fichier particulier dans un certain emplacement. Cette technique se base aussi sur la probabilité en essayant de prédire le nombre de répliques dans le système.

L'inconvénient de cette approche est que le processus de réplication repose toujours sur le service de découverte d'une ressource pour déterminer le nombre de répliques de l'objet dans le système. De plus les répliques peuvent être créées déraisonnablement dans le système si plus d'un nœud réplique le même fichier simultanément.

3.4.1.10 Réplication adaptive basée sur le modèle de comportement des peers

Cette approche [16] utilise la pertinence et l'utilité des peers pour déterminer combien de répliques doivent être créées et leur emplacement. Les nœuds cibles sont déterminés par un groupe de peers et des critères de sélection. Quatre types de groupe sont utilisés : placement sur les voisins (PN), placement sur références inversées (PIR), placement sur les peers pertinents (PRP) et placement aléatoire. Dans la méthode PN, un groupe de peers est définit comme étant l'ensemble des voisins du peer source. Les peers voisins sont classés dans un ordre croissant selon leur utilité. Dans la méthode PIR, un groupe des peers représente l'ensemble des nœuds qui accèdent aux documents du peer source. La méthode PRP, définit un groupe de peers comme un ensemble de peers ayant envoyés des requêtes au peer source.
Le placement aléatoire, définit un groupe de peer comme tout peer rejoignant le système. Le peer cible est choisi aléatoirement parmi tous les peers. Pour la méthode PN, le traitement des requêtes est efficace lorsque le réseau n'a pas beaucoup de peers. Cependant la performance se dégrade, à mesure que le nombre de peers augmente. Donc elle n'est pas très adaptée pour les systèmes évolutifs.
Dans le cas de la méthode aléatoire, un document favori est placé indépendamment de la distance entre un peer faisant la requête et celui qui détient le document. Toutefois le nombre de résultats de la requête diminue si le nombre de peers augmente. Aucune des méthodes PIR et PRP n'est affectée par l'augmentation du nombre de peers comparée aux méthodes aléatoire et PN. Pour l'efficacité de recherche, la technique PRP donne de meilleures performances que les autres techniques. Mais l'utilité d'un peer est définie uniquement par un seul paramètre. Ce qui rend le processus de sélection de peer non précis.

3.4.2 Réplication basé sur le nombre des répliques

3.4.2.1 Approche prédictive [24]

Cette approche repose sur des schémas de réplication prédéfinis dont le principe n'évolue pas au fil du temps. Elle se base sur une modélisation du taux d'utilisation des données. Une étude réalisée dans [7], a évalué les stratégies de l'approche prédictive sur un

réseau P2P décentralisé, les résultats peuvent aussi être étendus aux autres architectures. Ces techniques calculent le nombre optimal de copies pour chaque objet en termes de recherche moyen par requête réussie. Trois stratégies y sont observées :

3.4.2.1.1 Stratégie de réplication uniforme

C'est la stratégie la plus simple. Elle consiste à répliquer les fichiers du système équitablement. Ainsi, tous les fichiers partagés auront le même nombre de copies distribuées sur les nœuds du réseau. Ceci est réalisé au moment de la première publication d'un fichier où un nombre fixe de copies est créé.

Cette stratégie de réplication est mise en œuvre dans PAST, P-GRID [7].

3.4.2.1.2 Stratégie de réplication proportionnelle

Consiste à répliquer les fichiers partagés selon leur popularité. Plus un fichier est demandé plus il est répliqué : une réplication explicite à destination d'un nombre fixe de nœuds est déclenchée à chaque fois qu'un objet est demandé.

Donc pour cette approche, le nombre de copies pour chaque objet est proportionnel aux distributions des requêtes. Plus le taux de demande d'un objet est élevé plus le nombre de répliques augmente. Mais même si les requêtes sur les données populaires sont traitées de manière efficace, la recherche de données non populaires peut prendre un certain temps.

3.4.2.1.3 Stratégie de réplication racine carrée

Pour la réplication racine carrée (Square Root, SR), le nombre de répliques d'un fichier i est proportionnel à la racine carrée du taux d'utilisation. Les travaux montrent que cette réplication permet de minimiser le critère ESS. D'après les simulations réalisées dans l'étude [7,60], cette stratégie est la plus performante du point de vue du coût des recherches.

La difficulté majeure de l'approche prédictive est de disposer d'un modèle convenable.

3.4.2.2 La réplication Pull-then-Push (PtP) [24,74]

Le système se compose de deux phases. La phase Pull, qui consiste en la recherche d'un objet. Après une recherche fructueuse le peer demandant l'objet entre dans une phase Push, par laquelle il transmet les copies de l'élément désiré selon une réplication racine carrée. Pour cette réplication le nombre de répliques à créer est égal au nombre de peers explorés. Puisque les répliques sont placées sur tous les peers explorés, les peers à faible performances peuvent recevoir des répliques inutilement. La mise à jour des répliques peut être considérablement améliorée par une phase « update-push », où le peer qui a créé les copies propage les mises à jour qu'il a reçu avec des paramètres similaires à ceux de la phase « push de réplique », ce qui permet d'atteindre un bon placement des répliques avec de petits messages en général. Mais pour cette stratégie les répliques sont copiées seulement dans le voisinage d'un peer sans prendre en considération leur comportement passé. Ainsi tous les voisins auront une copie de la réplique, ce qui augmente les coûts dans le réseau.

3.4.2.3 Réplication optimale de contenu [24]

C'est une technique adaptative, entièrement distribuée qui réplique dynamiquement le contenu d'une manière quasi-optimale. Elle comprend une règle d'affectation logarithmique

qui fournit une solution optimale pour l'approximation continue du problème. Deux algorithmes sont proposés : l'algorithme « Top-K LRU » et l'algorithme « Top-K MFR » (Most Frequently requested). L'algorithme Top-K LRU réplique le contenu à la volée sans aucune connaissance à priori des requête d'objets ou des probabilités de connexion des nœuds.

L'algorithme LRU ne réplique pas les objets non populaires. Un tel objet est stocké dans l'un des nœuds et y reste jusqu'à ce qu'il soit supprimé avec LRU. Mais si l'objet est très impopulaire il ne sera probablement jamais demandé, ce qui entrainera une perte de l'espace de stockage inutilement.

L'algorithme MFR est une alternative à l'algorithme Top-K LRU qui gère le stockage de manière efficace. L'algorithme MFR suit sa propre recherche et politique de remplacement et rend les donnée avec des taux de réussite élevé de manière très optimale.

3.4.3 Réplication pour les architectures Super-Peer

3.4.3.1 Réplication dynamique [24]

La réplication dynamique est utilisée dans l'architecture super-peer. Elle prend le coût de recherche d'un élément et réplique avec succès les données les plus fréquemment consultés en se basant sur les probabilités d'accès. Deux techniques d'équilibrage de charge par la réplication sont proposées : une réplication périodique basée sur le mode «push» (PPR : *Periodic Push based Replication*) et une réplication à la demande (ODR : On Demand Replication) [25].

Dans la réplication PPR, le super-peer hébergeant les données, envoie périodiquement des répliques des fichiers les plus fréquemment consultés aux super-peers distants sur la base d'une fréquence d'accès globale.

L'algorithme ODR effectue la réplication en se basant sur les fréquences d'accès local. Une demande pour la réplication est initiée par un super-peer si la fréquence d'accès d'un fichier particulier atteint un seuil prédéfini. Cette technique permet aux super-peers de s'adapter dynamiquement aux changements de comportements d'accès. Cependant, il devient gourmand quand chaque super-peer tente d'effectuer la réplication selon ses propres besoins plutôt que de répliquer selon une perspective globale comme c'est le cas pour l'algorithme PPR.

On remarque que la réplication PPR réduit le nombre de sauts pour trouver des fichiers et que la réplication ODR fournit un changement adaptatif en fonction du comportement d'accès. Mais ces stratégies ne fournissent pas de critères appropriés pour la sélection de nœuds cibles lors de la réplication.

3.5 Conclusion

A travers ce chapitre nous avons présenté une synthèse sur le problème de la réplication dans les systèmes à large échelle. Nous avons particulièrement détaillé les techniques de réplication et les différentes classes auxquelles elles appartiennent.

Il est important de souligner qu'il est difficile de trouver un consensus pour une technique particulière car d'une part, les objectifs de la réplication sont conflictuels et d'autre part certains paramètres sont à prendre en charge selon le type et la topologie du réseau. Par conséquent il est impossible de présenter une stratégie de réplication universelle. Une stratégie de réplication telle qu'elle soit doit tenir compte de la déconnexion des nœuds, leurs disponibilités, leurs capacités de stockage, leurs performances en terme de traitement mais aussi du type du réseau sur lequel ils sont implémentés. Idéalement les répliques devraient être placées à proximité des nœuds susceptibles de les demander.

Dans ce chapitre, nous avons mis en évidence les problèmes posés par l'utilisation de la réplication dans les environnements à large échelle. Trois problèmes particuliers suscitent l'intérêt des chercheurs : le problème de gestion de la cohérence des répliques, le problème du nombre de répliques à créer et le problème du placement efficace de ces répliques. Nous nous intéressons à ces deux derniers problèmes, pour lesquels nous proposerons des solutions dans la deuxième partie.

Partie 2 : Stratégies de réplication proposées

Introduction :

L'objectif principal de cette thèse est une contribution à la gestion du problème de réplication dans les environnements à grande échelle. Nous adressons plus particulièrement le problème du nombre de répliques à créer ainsi que leur placement stratégique.

Dans la littérature, la plus part des travaux de recherches se sont focalisés sur les structures hiérarchisées et ont mentionné l'extension de leurs recherches aux topologies sous forme de graphe. Dans le cadre de cette thèse, nous considérons qu'une grille est un environnement totalement distribué, de ce fait nous adoptons une architecture décentralisée. La grille en entrée est représentée par un graphe général où les nœuds sont organisés indépendamment les uns des autres.

Dans les stratégies que nous proposons, deux facteurs sont importants et ont un impact direct sur les performances du système :

- La sélection du fichier à répliquer
- La sélection du site devant héberger la nouvelle réplique.

Cette deuxième partie de la thèse, expose les stratégies de réplication que nous avons proposées et implémentées. Chaque stratégie est synthétisée dans un chapitre dans lequel nous présentons le modèle proposé, les algorithmes implémentés ainsi que les expérimentations menées suivies de l'interprétation de chaque résultat obtenu.

Notons que les plateformes sur lesquelles porte notre étude sont des environnements largement distribués. Nous avons donc choisi de faire usage d'un simulateur afin de tester nos propositions. Notre choix s'est porté sur OptorSim[42,46], un simulateur développé dans le cadre du projet européen DataGrid [44]. Il a été conçu pour tester les différents algorithmes de réplication et de gestion des données au sein de la grille DataGrid. L'objectif initial de sa conception s'avère très proche du nôtre, de plus il intègre de nombreuses fonctionnalités qui nous sont nécessaires.

L'environnement de développement que nous avons choisi est NetBeansIDE [83] en sa version 7.0, toutes nos implémentations ont été réalisées avec le langage JAVA sur une machine fonctionnant sous le système Windows 7 avec les caractéristiques suivantes :

- Processeur : Intel Core I5-480 M 2.7GHz.
- RAM : 4 Go.

Chapitre 4

Stratégie1 :

Réplication des données dans un environnement P2P non structuré

4. Stratégie 1 : Réplication des données dans un environnement P2P non structuré

4.1 Problématique

Nous nous intéressons dans cette stratégie de réplication aux systèmes p2p et plus particulièrement aux réseaux non structurés hybrides. Ces systèmes sont les plus utilisés aujourd'hui et représentent la base du partage de données sur Internet. De plus, les réseaux p2p non structurés présentent plusieurs avantages tels que :(1) l'adaptabilité : en effet, les connexions et déconnexions des peers sont sans conséquence contrairement à un réseau structuré. (2) La tolérance aux pannes : un grand nombre de nœuds peuvent répliquer les mêmes données. (3) l'hétérogénéité : un système structuré suppose que les machines sont homogènes or, cela n'est pas vrai en pratique. Dans un système non structuré, les peers ont des disparités en ce qui concerne la bande passante, la capacité du disque ou la puissance du processeur. (5) la taille du réseau non structuré est théoriquement infinie : il n'y a pas de contraintes sur les ressources d'un peer. Toutefois, les réseaux non structurés présentent un inconvénient majeur qui est la consommation de la bande passante. Pour remédier à ce problème, nous avons hybridé entre le modèle client/serveur et le modèle p2p en faisant intervenir les super peers. Ces derniers sont élus et organisés en réseaux p2p. Les peers ayant une faible bande passante sont reliés en mode client/serveur à un super peer réalisant ainsi une fédération de clusters. Ce modèle permet d'une part de réduire la quantité des messages présents dans le réseau car seuls les super-peers échangent des messages. D'autre part, ceci permet d'accélérer le traitement des requêtes.

Les systèmes peer-to-peer constituent une solution efficace pour le partage d'un grand volume de données. Comme la donnée est un élément essentiel, son indisponibilité conduirait nécessairement au délaissement du réseau par ses participants. Pour remédier à cela, des techniques de réplication sont souvent utilisées. Les protocoles de réplications doivent permettre de gérer au mieux les données échangées et de trouver des compromis afin d'éviter de nuire aux performances du système en le surchargeant de transfert inutile et de réplications superflues. De ce fait une bonne stratégie de réplication doit contrôler le nombre de répliques créées, il est bien évident qu'il ne s'agit pas de répliquer à l'infini car ceci nuirait aux performances générales du système. Dans ce chapitre, nous présentons une stratégie de réplication adaptative qui s'appuie sur des stratégies [4, 9,10] déjà abouties et consiste en une combinaison judicieuse de ces travaux. La stratégie proposée dans [9,10] calcule le nombre de répliques d'un objet sans la prise en compte des répliques déjà existantes pour cet objet, ce qui a pour conséquences de favoriser les données les plus populaires de certains clusters au détriment des données des autres clusters. Nous proposons une stratégie qui prend en compte le taux de sollicitation [4] d'un objet afin de générer le nombre optimal de répliques. De plus, l'algorithme de réplication proposé assure le placement dynamique des répliques sur les sites les plus appropriés afin de s'adapter aux accès des utilisateurs et donc rendre les données plus disponibles. Une politique de suppression de répliques est utilisée si nécessaire. L'approche proposée tente de répondre aux questions suivantes :

- *Que répliquer ?* Sélection de répliques

La sélection de répliques cibles dépend de la popularité et de l'importance du contenu qu'on peut connaître à travers les accès utilisateurs. Pour construire un modèle d'accès, la distribution uniforme est utilisée. Pour l'accès au contenu on suppose un accès en lecture seulement ce qui est le cas en général pour les systèmes P2P de partage de fichiers comme Gnutella et Kazaa. [66,68.]

- *Combien répliquer ?* Nombre de répliques

En plus de la popularité et l'importance du contenu, la capacité de stockage et la bande passante des peers affectent fortement le choix du nombre de répliques. Pour fixer le nombre de répliques nécessaires et pouvoir les placer, on utilise des distributions dynamiques proportionnelles aux taux de sollicitation des ressources.

- *Où placer les répliques ?* Placement de répliques

Pour trouver un bon placement des répliques, il ne faut pas prendre en compte seulement la popularité du contenu et la capacité de stockage mais aussi la disponibilité et la volatilité des peers, c'est-à-dire le nombre de peers connectés à un certain moment et dont le contenu peut être accessible. Ainsi les répliques seront placées sur les peers les plus disponibles et ayant les plus grandes capacités de stockage.

4.1.1 Pourquoi un réseau non structuré

Du fait de leur nature complètement distribuée et de l'utilisation de techniques adaptatives, les architectures P2P sont bien adaptées aux environnements dynamiques.
Suivant la nature de cet environnement (peu ou très dynamique, homogène ou hétérogène…), certaines architectures sont plus adaptées que d'autres.
Dans le cadre de ce travail nous avons opté pour une architecture totalement décentralisée où tous les peers sont égaux ce qui les rend indépendants les uns des autres. Cette indépendance a pour effet une haute tolérance aux pannes et aux connexions et déconnexions des peer. Cependant, du fait de la consommation accrue de la bande passante, dû à l'échange de grand nombre de messages, en plus de l'absence de point central nous avons choisi d'utiliser des super-peer.
Nous détaillons ci-dessous les raisons qui nous ont poussé à nous orienter vers une architecture P2P non-structurée plutôt que structurée

- **Localisation des informations**

La localisation des informations est ce qui différencie le plus les architectures structurées des architectures non-structurées. Alors que les premières font correspondre l'emplacement des informations à des propriétés topologiques et permettent ainsi d'acheminer très rapidement les messages vers leur destination, les secondes n'imposent aucune contrainte à ce niveau.

- **Hétérogénéité entre les peers**

La plupart des architectures P2P structurées proposées considèrent un réseau contenant un ensemble de machines homogènes.

Nous souhaitons que le système fonctionne sur des réseaux hétérogènes de machines, du point de vue des capacités de traitement, stockage, ou bande passante. Donc tous les nœuds de l'architecture P2P non structurée ne sont pas égaux les uns par rapport aux autres.

- **Capacité de stockage et de bande passante**

L'insertion d'une information et le traitement de requêtes dans un réseau P2P se fait sans la connaissance de la charge du peer censé héberger cette information. Ainsi, les peers ayant des capacités de stockage et de bande passante en dessous de la moyenne seront en surcharge, alors que les peers ayant des capacités de stockage et de bande passante au-dessus de la moyenne seront en sous-charge. Il est important de prendre en compte cette contrainte pour le choix du modèle.

- **Acheminement des requêtes**

Dans les réseaux P2P utilisés pour le partage de données, les peers ne restent, en moyenne, pas très longtemps dans le réseau (la durée d'activité est de l'ordre d'une heure). Dans les réseaux de grande taille cela entraîne une fréquence globale de connexion et déconnexion très importante. Cela ne pose quasiment aucun problème aux réseaux P2P non structurés, l'essentiel pour un peer étant de rester connecté à au moins un autre nœud du réseau. Si ce peer se déconnecte complètement, il peut simplement recommencer la procédure d'entrée dans le réseau. Cependant il faut ajouter des opérations pour détecter les pannes et faire des copies des données perdues.

4.2 Modèle proposé

Le choix d'une topologie représente un aspect crucial de la modélisation des systèmes P2P. Jusqu'à récemment, la plupart des applications P2P étaient caractérisées par l'absence de mécanisme spécifiant la topologie, les conséquences étant des schémas de communication inefficaces, tel que le flooding [72], Comme c'est le cas pour le système Freenet, ou encore le projet Gnutella V0.4). Un autre problème étant la considération des peers comme étant égaux, mais en réalité les différents peers sont très hétérogènes [6,17]. La rapidité dans les recherches ou les transferts de contenu dans les réseaux Peer-to-Peer va nécessairement dépendre de la topologie du réseau. Pour qu'un réseau Peer-to-Peer soit efficace, il faut que sa structure prenne en considération la topologie au niveau overlay mais également la topologie « réelle » du réseau au niveau sous-jacent IP.

La topologie Overlay d'un système P2P est la vue globale du système. C'est un graphe non orienté dont les sommets sont les peers. Donc un réseau P2P est représenté par un graphe non orienté G = (A, S), avec n arêtes où A : Arêtes, S : Sommets. Ainsi la topologie au niveau overlay reflète la disposition des liens entre les peers et doit être prise en compte pour une meilleure efficacité. La topologie au niveau sous-jacent, quant à elle considère plutôt les distances « physiques » entre les peers au niveau IP.

Ces deux topologies peuvent être très différentes pour un même réseau. En particulier le réseau Peer-to-Peer étant un réseau overlay, un saut entre deux peers voisins au niveau overlay ne signifie pas nécessairement que les deux peers soient proches. Si on regarde au

niveau du réseau Internet le réseau overlay ne prend pas en compte les caractéristiques du réseau sous-jacent.

4.2.1 Modèle du système Peer-to-Peer

On suppose un modèle Peer-to-Peer pur sans aucune hiérarchie imposée sur l'ensemble des peers. Tous les peers peuvent faire une requête pour un objet donné et répondre à d'autres. Chaque peer maintient localement sa propre liste de fichiers ainsi qu'une vue local du système. Les peers ignorent la connectivité physique et la topologie du système, mais ont des connaissances de leurs voisins et d'autres informations (table de routage, fichiers existants…).

Le système est caractérisé par un comportement dynamique avec des peers qui se connectent et se déconnectent à tout instant et ajoutent ou suppriment des fichiers de leurs listes. Bien que l'environnement soit symétrique, il est clair que les peers présentent différentes capacités de partage. Une variété de paramètres notamment la capacité de stockage, de calcul CPU, la popularité des fichiers stockés, la charge de travail du système, la connectivité, …, contribuent à cet état de fait. Certains de ces facteurs restent plus ou moins statiques dans le temps (par exemple, la puissance de traitement ou la taille maximale disponible de bande passante d'un peer), tandis que d'autres changent de façon dynamique.

Notre objectif est de concevoir et mettre en œuvre une stratégie de réplication qui permettra un partage efficace des fichiers (fournir un nombre minimal de répliques), l'évolutivité et la tolérance aux pannes.

4.2.2 Architecture du modèle logique

Tout d'abord, l'architecture du réseau physique est organisé en sous réseaux sous diverses topologies. Les réseaux peuvent être connectés entre eux via l'Internet qui est moins rapide et moins fiable que le réseau local. De plus les machines sont hétérogènes, de bande passante et de système d'exploitation différents.

Afin d'optimiser la communication entre peers, nous organisons la topologie logique [54] en structure à deux couches :

- La couche supérieure se compose de clusters : Un cluster est constitué par les peers dans un même sous-réseau. Chaque cluster a un Identifiant unique. Les clusters sont organisés par un graphe (S, A), où S= {C1, C2,…, CN} est l'ensemble de tous les clusters et A est un ensemble donné d'arêtes virtuelles entre les clusters dans S : Les arêtes dans A sont bidirectionnelles.
- La couche inférieure est l'ensemble des peers, chaque peer a aussi un identifiant unique.

Il y a deux types de lien :

- Les liens locaux (intra-cluster) : qui relient des peers dans un cluster.
- Les liens globaux (inter-clusters) : qui relient des super-peers de différents clusters.

Deux peers sont des voisins locaux s'ils se relient par un lien local. Ils sont des voisins globaux s'ils se relient par un lien global.

Chaque peer maintient k liens globaux avec un cluster voisin.

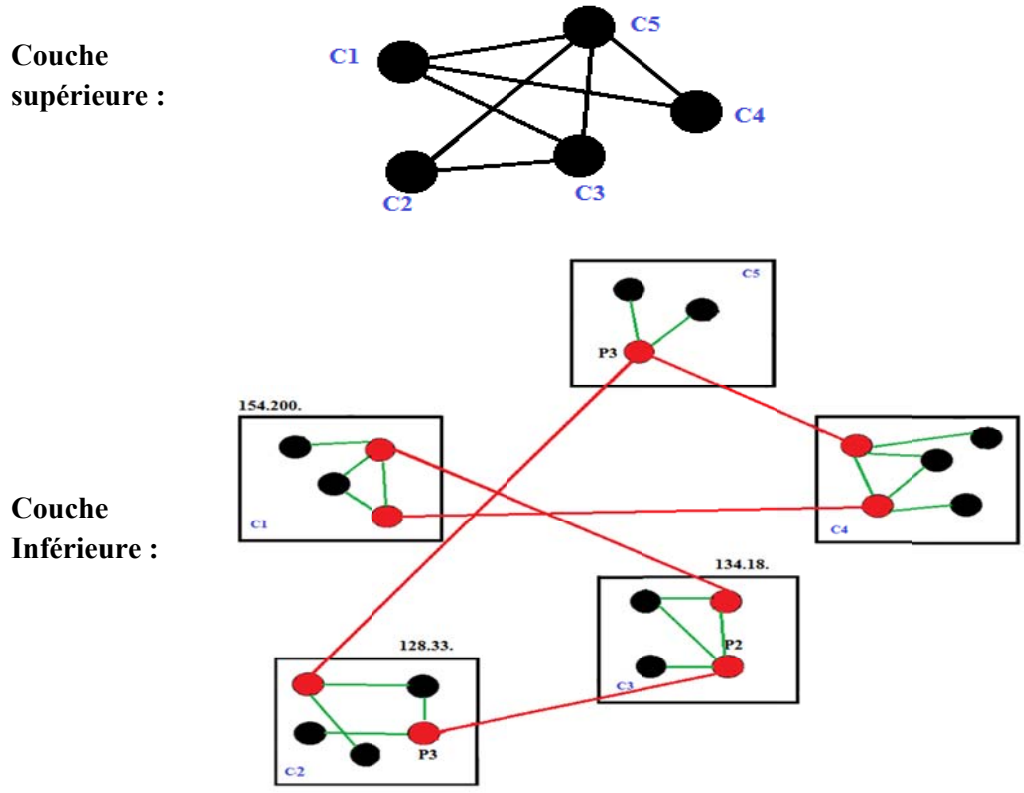


Figure 4.1. Architecture à deux couches

La figure 4.1 montre un exemple de l'architecture à deux couches. Il y a cinq clusters C1, C2, C3, C4, C5. Les lignes en vert sont des liens locaux et les lignes en rouge représentent les liens globaux.

4.3 Architecture générale du modèle proposé

Le modèle proposé pour la gestion de la réplication des données est présenté dans la figure 4.2.

- Une étape de prétraitement précède le processus de réplication. Elle intègre les services suivants :
  - Construction du modèle : cette phase définit la topologie logique du réseau.
  - Maintenance du modèle : cette phase permet de gérer la défaillance afin de maintenir la topologie.
- La deuxième étape, celle du traitement définit la stratégie de réplication adoptée.

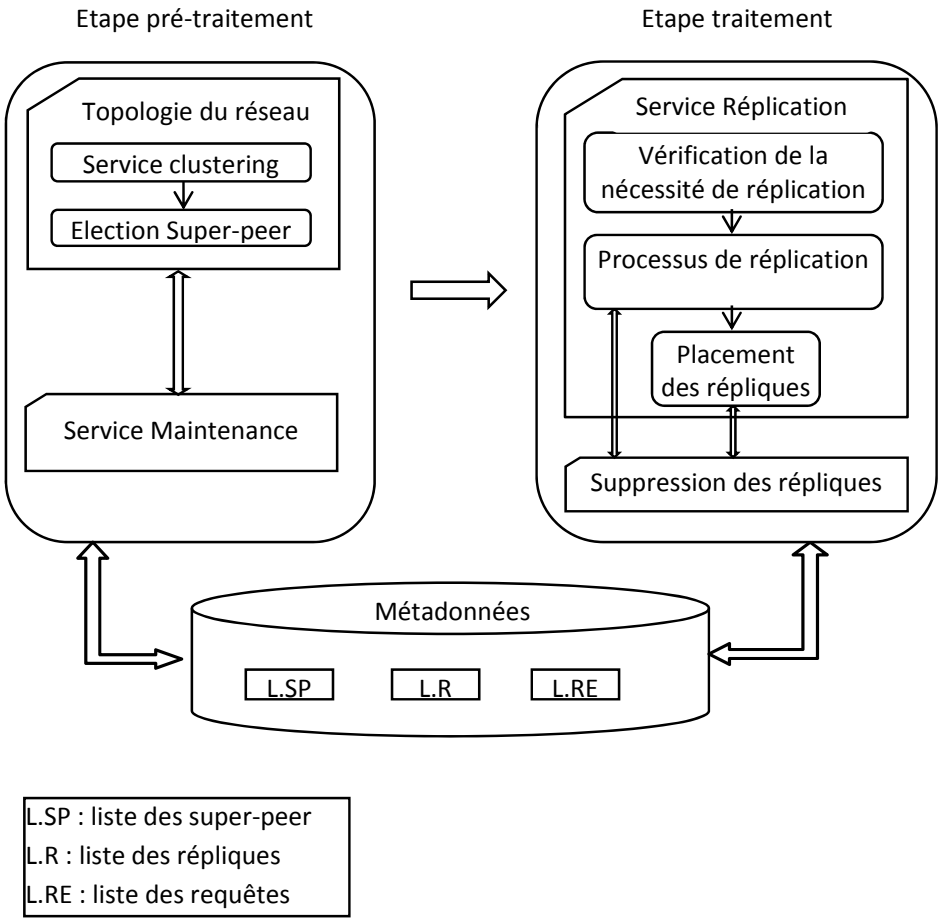


Figure 4.2. Architecture générale du modèle de réplication proposé

4.3.1 Etape de prétraitement

4.3.1.1 Construction de la topologie

- **Construction de clusters**

Le clustering est une technique qui consiste à agréger et à organiser les différents peers d'un système en un ensemble de groupes, chaque groupe étant contrôlé par un chef de groupe, appelé 'super peer'. L'objectif est d'obtenir un ensemble de clusters couvrant l'ensemble de la population des peers , de manière à ce que les peers sélectionnés forment une infrastructure virtuelle permettant de surmonter les problèmes de passage à l'échelle et d'assurer une utilisation meilleure des ressources du réseau.

- **Création des identifiants**

Nous utilisons la fonction de hashage SHA-1[73] et l'adresse IP pour créer les identifiants des peers : SHA-1 (Secure Hash Algorithm) est une fonction de hachage cryptographique qui produit une chaîne de 160 bit (appelé condensé d'une chaîne) à partir d'un message.

L'adresse IP d'un peer dans un sous-réseau peut être divisée en deux parties : le préfixe de réseau identifie le sous-réseau qui est commun pour tous les hôtes dans ce sous-réseau ; le numéro d'hôte qui distingue les hôtes dans un même sous-réseau. Le masque de sous-réseau est utilisé pour déterminer ces deux parties. (Par exemple, l'adresse IP d'un peer est 132.46.216.56 et le masque de son sous-réseau est 255.255.0.0).

- L'ID du cluster est le résultat de la fonction hachage du préfixe de sous-réseau (par exemple si SHA-1(132.46.) = 113, alors IDC = 113).
- L'ID d'un peer est la concaténation de l'ID de son cluster et le résultat de la fonction de hachage de son numéro d'hôte (Par l'exemple, si SHA-1(216.56) = 412 alors IDP = 113412).

Algorithme 1: Procédure Clustering ()

Begin
Créer le 1er cluster; //*un cluster représente un sous réseau*
AdresseCluster$_1$ = adressePeer$_1$ AND Masque;
Ajouter peer$_1$ au cluster$_1$;
For (i = 2 jusqu'à NbPeers) **do**
Créer = true;
For (j = cluster$_1$ jusqu'à Nbclusters) **do**
  If (adressePeer$_i$ AND Masque = adresseCluster$_j$) **then**
    Ajouter peer$_i$ au cluster$_j$;
    Créer = false;
  End if;
  If (Créer = true) **then**
    Créer cluster$_{j+1}$;
    AdresseCluster$_{j+1}$ = adressePeer$_i$ AND Masque;
    Ajouter peer$_1$ au cluster$_{j+1}$;
  End if;
End for;
End for;
End.

Algorithme 4.1. Procédure de Clustering

- **Election des Super-Peer**

Dans ce travail nous nous intéressons aux systèmes P2P de partage de fichiers où la garantie de la disponibilité des données est nécessaire à leur existence. Nous avons donc prévu de garantir cette disponibilité même en cas de pannes ou de défaillances d'un peer en prenant en compte la redondance des super-peers et en détectant les pannes par échange de messages de vie entre les différents peers du réseau.

Afin de bien choisir les super peers, nous avons utilisé une fonction multicritère qui détermine le poids de chaque peer. Les critères retenus pour un peer i sont :

- Capacité de stockage s_i

- Vitesse CPU v_i
- Bande passante b_i
- Disponibilité d'un peer (Temps connexion / (Temps connexion + temps déconnexion) d_i

Nous calculons le poids de chaque peer après normalisation de ses valeurs par la formule suivante.

$$P_i = f(s_i, v_i, b_i, d_i) = s_i + v_i + b_i + d_i.$$

Afin de bien pondérer les quatre paramètres intervenant dans la fonction multicritères "f ", nous avons mené plusieurs expérimentations en faisant varier les coefficients s_i, v_i, b_i et d_i. Les meilleurs résultats ont été obtenus avec des coefficients tous égaux à 1.

Un vecteur de poids de tous les peers est formé et ordonné par ordre décroissant. Ainsi les k premiers peers seront élus super-peers et diffuseront leurs adresses à tous les peers du cluster. De ce fait la présence d'une entité faisant office de serveur local fait émerger le problème du point de défaillance unique. Si celui-ci est hors service, tout le sous réseau se voit coupé du reste du réseau. La redondance des super-peers aura pour effet de rendre le système plus robuste et plus tolérant aux pannes ce qui améliore la disponibilité des données. Les peers étant fortement dépendants de leur super peer, l'ajout de la redondance fiabilise plus le système : Un peer peut alors avoir plusieurs super peers.

Chaque super-peer maintient une table contenant le nombre de requêtes total ainsi que le nombre de requêtes reçu pour chaque ressource.

Algorithme 2: ProcédureChoisir_SP ()

```
Begin
For (chaque cluster) do
For (chaque peer) do
Définir (si, vi, di, bi)
Normaliser (si, vi, di, bi)
        Pi = f (si, vi, di, bi) = si + vi + di + bi;
End for;
VP (Idi, Pi); //Former vecteur de poids
Diffuser vecteur de poids // ordonné selon un ordre croissant
Choisir K premier peer comme SP ;
Diffuser (SP, IP_SP);
End for;
End.
```

Algorithme 4.2. Procédure Election Super-Peer

- **L'ajout d'un Peer**

Quand un nouveau peer se connecte au réseau, il obtient une adresse IP et il peut récupérer le masque de son sous-réseau. A partir de ces données, il engendre son ID. Après, il diffuse un message ***(ID, NEW)*** dans son sous-réseau.
Le peer reçoit des réponses ***ACK*** des supers-Peer qui fournissent leurs ID et leurs adresses IP. Le nouveau peer choisit les super-peers voisins avec lesquels il veut se connecter parmi ceux qui ont répondu et les ajoute à sa liste de super-peer.
Si le nouveau peer ne reçoit aucune réponse, il considère qu'il est le premier peer dans son sous-réseau. Donc, il forme un nouveau cluster et se considère comme super-peer.

- **Formation des liens globaux entre les Super-Peer**

Quand un nouveau peer désigné comme super-peer rejoint le système, il cherche à établir des liens globaux à travers ses voisins locaux.

Par exemple, dans la figure 4.3. P1 est un nouveau super-peer qui a établi des liens locaux avec P2 et P3 et il cherche un voisin global, alors il envoie un message de demande de voisin global à P2 qui va retransmettre la demande à ses voisins globaux. P4, qui est un voisin global de P2, retransmet la demande à ses voisins locaux. Si P5 accepte, il va répondre à P2 et les deux peers vont établir un lien global entre eux.

Nous supposons que, initialement, chaque super-peer connait une liste des peers appartenant aux autres clusters dit peer de contact. Cette liste s'enrichira au cours de son fonctionnement. Le nouveau peer envoie les messages de demande de cluster voisin aux peers de contact. Le peer de contact cherche les clusters qui sont les plus proches du nouveau cluster et retransmet la demande aux peers de contact de ces clusters. Ces peers répondent au nouveau peer avec l'ID de son cluster et son adresse IP. Enfin, le nouveau peer établit les liens globaux avec ces peers.

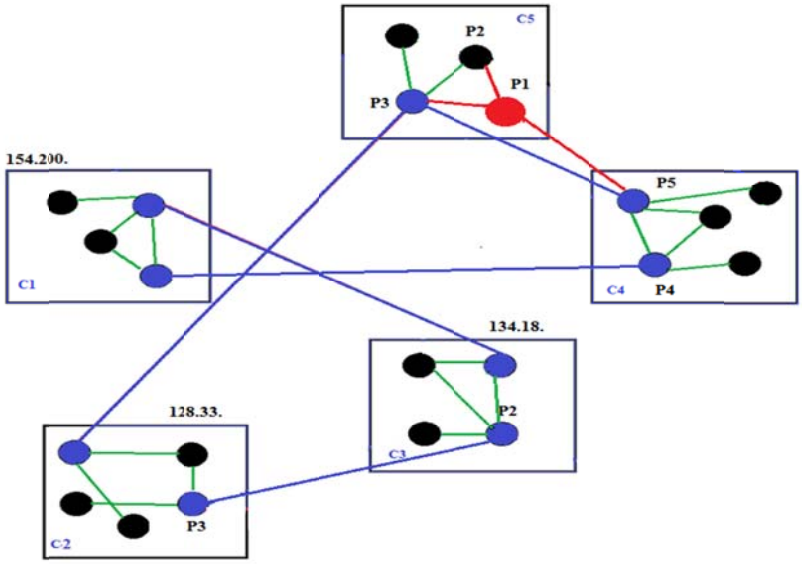


Figure 4.3. Arrivé d'un nouveau peer

- **Choix des clusters voisins**

Le critère pour choisir les clusters voisins est la proximité.

Il y a deux approches pour estimer la proximité :

- La première approche est basée sur le temps de réponse (RTT - Round Trip Time).
- La deuxième approche est basée sur l'adresse IP.

Notre système fonctionne sur le réseau IP, nous choisissons donc l'approche basée sur l'adresse IP et nous utilisons la métrique XOR pour calculer la proximité [52].

Ainsi la distance entre deux clusters Cj ayant le préfixe {pi31, pi30 … pi0} (présenté sous forme binaire de 32 bits) et Ck ayant le préfixe {pj31, pj30 … pj0} est définie comme suit, avec la métrique XOR [52] :

$$d(C_j, C_k) = \sum_{v=0}^{31} |j_v - k_v| \cdot 2^v$$

La métrique ***d*(Cj, *Ck*)** a les propriétés suivantes :

- Si ***d*(Ci,*Ck*)** = ***d*(Cj,*Ck*)** pour n'importe quelle *Ck*, alors *Ci*= *Cj*.
- Max***d*(*Cj, Ck*)**≤ $2^{32} - 1$.
- Si***c*(Cj,*Ck*)**est le nombre de bits commun dans le préfixe entre Cj *etCk*, et si ***c*(Cj,*Ck*)** = *m*, alors ***d*(Cj,*Ck*)**= 2^{32}- m –1.
- Si ***d*(Ci,*Ck*)**≤***d*(Cj,*Ck*)**, alors ***c*(Ci,*Ck*)**≥***c*(Cj,*Ck*)**.

On peut trouver que d(Cj,Ck) correspond à l'assortiment de préfixe le plus long, si Ci est l'unique préfixe le plus long correspondant à Cj, alors Ci est le plus proche de Cj en termes de cette métrique. Donc, un cluster va choisir le cluster voisin tant que la métrique entre deux clusters est minimum.

Par exemple, dans la figure 4.4. P1 est un nouveau peer, qui va former un nouveau cluster C6 ayant le préfixe 128.77. P1 contacte P2 du cluster C3 ayant le préfixe 134.18. Pour chercher un cluster voisin. P2 calcule la distance entre son cluster ainsi que ses clusters voisins C1, C2 avec C6 :

- **d(C3, C6)** = 524423145, **d(C2, C6)** = 235489, **d(C1, C6)** = 908853211.

Donc, P2 trouve que C2 est le plus proche de C6, il transmet la demande à P3 de C2. P3 répond à P1 avec son ID et son adresse IP, puis P3 et P1 établissent un lien global entre eux. C2 devient alors un cluster voisin de C6.

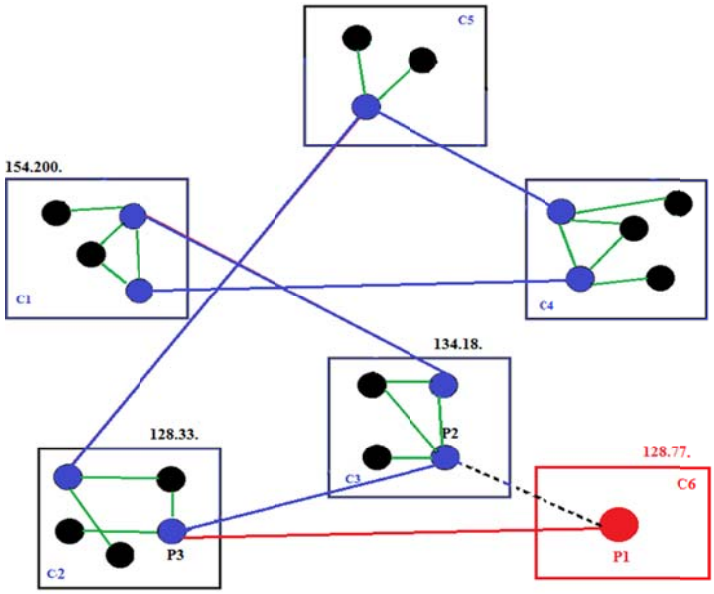


Figure 4.4. Choix du cluster voisin

Chaque super-peer maintient une table de voisins au niveau local, une table de voisins au niveau global, une table de clusters voisins et une table de correspondance (qui contient l'adresse des peers avec lesquels il doit échanger les résultats des itérations de calcul) et chaque peer contient une liste de ses super-peers.

4.3.1.2 Maintenance du modèle

Nous employons des messages de vie AYA() (Are You Alive) pour maintenir la topologie. Chaque super-peer envoie périodiquement ce message à ses voisins. Lors de la réception de ce message, les peers répondent en accusant réception par le message ACK(). Chaque fois qu'un super-peer envoie un message AYA(), il fixe un délai de temps t , s'il ne reçoit pas de message d'acquittement d'un voisin au bout de deux tentatives, le super peer considère qu'il a une défaillance. Il vérifie alors auprès des autres super-peers s'il s'agit d'une défaillance du peer lui-même ou uniquement du lien le reliant à celui-ci. En fonction des réponses, des mises à jour sont effectuées dans les catalogues (métadonnées). .

4.3.2 Etape de traitement

- **Notation**

M : Nombre de données dans le système,
$\boldsymbol{T_j}$: Taille d'une donnée j,
$\boldsymbol{R_{ij}}$: Nombre de répliques d'une donnée j stockée dans un peer i,
$\boldsymbol{TR_j}$: Nombre de réplique totale d'une donnée j ($TR_j = \sum_i Rij$),
$\boldsymbol{TR_{jc}}$: Nombre de réplique pour une ressource j existante dans le cluster c,
$\boldsymbol{B_i}$: Bande passante moyenne d'un peer i,
$\boldsymbol{C_i}$: Capacité de stockage d'un peer i,
T : Taille du réseau (en nombre de peers),

λ : Nombre de requêtes total,
λ_j : Nombre de requêtes total pour une donnée j,
R_i : Nombre de répliques stockées dans un peer i,
$S2$: Seuil maximal de capacité de stockage d'un peer,
NR_j : Nombre de répliques requis pour une donnée j,
NR_{jc} : Nombre de répliques d'une donnée j pour un cluster c,
T_c : Taille du cluster c (en nombre de peers),
$S1$: Seuil maximal de nombre de requêtes.

4.3.2.1 Stratégie de réplication

La stratégie proposée se base sur le nombre de requêtes pour déclencher le processus de réplication. Une vérification de la nécessité de réplication est entamée en comparant le nombre de répliques déjà existantes pour une ressource par rapport au nombre de demande pour cette ressource. S'il est inférieur, le processus de réplication est déclenché sur chaque cluster et est exécuté par un super-peer qui se charge du calcul du nombre de réplique à créer pour chaque cluster, en prenant en compte le nombre de répliques déjà existantes. Ensuite les répliques sont placées sur les peers, les plus disponibles et ayant les plus grandes capacités de stockage. On définit ensuite une approche de suppression des répliques pour chaque peer de façon autonome s'il atteint le seuil de stockage maximal.

4.3.2.1.1 Hypothèses

- Un système P2P non structuré, où les peers partagent des fichiers et font des requêtes,
- Les fichiers échangés sont en mode lecture seulement,
- Un peer peut tomber en panne et un lien peut être défaillant,
- Tout peer connait son adresse et une liste de super-peer du cluster,
- Un super-peer peut être considéré comme un peer ordinaire,
- Les super-peers s'échangent périodiquement des métadonnées (nombre de peers, taille du cluster, nombre de répliques…) avec leurs voisins locaux et voisins globaux).
- Chaque peer a un identifiant unique.

4.3.2.1.2 Algorithme de réplication

Pour calculer le nombre optimal de répliques à créer pour un objet, nous nous sommes basés sur une modélisation du taux d'utilisation de cet objet. Nous avons étudié les stratégies existantes et opté pour la stratégie de réplication racine carré [4]. Dans cette stratégie, le nombre de répliques d'un objet est proportionnel à la racine carré du taux d'utilisation de cet objet. Des travaux de recherche [4,13] ont montré que cette stratégie permet de réduire le coût de recherche moyen et de minimiser le critère ESS (ExpectedSearch Size)[21]. Ce dernier définit le nombre moyen de peers explorés avant que le système ne réponde à une requête soluble. Initialement, On place dans chaque cluster un nombre aléatoire k de répliques pour chaque ressource. Chaque super-peer teste régulièrement si le nombre de requêtes total reçues est supérieur à un seuil[3] [9,10] maximal du nombre de requêtes.

3 : la valeur du seuil est définie par calibrage en se basant sur plusieurs expérimentations

Dans le cas où cette condition est vérifiée, l'algorithme lance le processus de vérification de la nécessité de réplication : il calcule pour chaque donnée la racine carrée du nombre de requêtes sollicitant cette ressource selon l'algorithme 4.4. Si cette valeur est inférieure au nombre total de répliques de cette ressource, le message (RepG, NRj) sera diffusé aux super-peers voisins globaux les informant du début du processus de réplication.
Ainsi pour chaque cluster c, le message (Repliquer, NR_j) est envoyé à un super peer i pour le calcul du nombre de réplique requis pour chaque cluster :

$$\mathbf{NR_{jc}} = \left(\boldsymbol{NR_j} - \boldsymbol{TR_{jc}}\right) . \frac{\mathbf{Tc}}{\boldsymbol{T}}.$$

Une fois cette étape terminée, on procède au placement des répliques.

RepG : Message de réplication pour les voisins globaux.

Algorithme 3: ProcédureRéplication ()

Begin
Placement _Init (k, Cluster);
For (chaque super-peer) **do**
If(λ > S1) **then**
Verifier_Necessite_Replication();
End if;
End for;
End.

Algorithme 4.3. Procédure de Réplication

Algorithme 4: ProcédureVerifier_Necessite_Replication ()

Begin
For (Chaque donnée j) **do**
NRj = $\sqrt{\lambda}$j;
If(TR_J< NR_J) **then**
Diffuser aux SP_Voisins le message (RepG, NR_j);
For (chaque cluster) **do**
Choisir aléatoirement un SPi;
Envoyer le message (Repliquer, NR_j) par le SPi
Calculer NR_{jc} = (NR_j - TR_{jc}). Tc/T ;
End for;
End if;
End for;
End.

Algorithme 4.4. Procédure Vérifier la nécessité de Réplication

4.3.2.1.3 Placement des répliques

En fonction du nombre de répliques calculé, les données seront placées sur les peers les plus disponibles, et ayant les plus grandes capacités de stockage. Un poids de placement est associé à chaque peer. Ce poids est calculé selon les paramètres :

- ✓ La capacité de stockage
- ✓ Le churn des peers

Les peers seront ordonnés dans un vecteur selon un ordre décroissant du poids, les répliques sont alors placées dans les NRjc premiers peers jusqu'à recevoir tous les acquittements pour un placement réussi.

Algorithme 5: ProcédurePlacement_Réplique (Réplique R)

Begin
VP (Si, di); // construire vecteur de poids
Ordonner vecteur de poids selon un ordre croissant
Choisir les NR_{jc} premiers peer;
For (i=1 jusqu'à NR_{jc}) **do**
$Peer_i$ = R // Placer les répliques jusqu'à recevoir tous les
// acquittements pour un placement réussi
End for;
Diffuser le message (RépSuccess, NR_{jc}) aux voisins locaux et globaux
End.

Algorithme 4.5. Procédure de placement

4.3.2.1.4 Suppression des répliques

Il est évident que nous ne pouvons pas répliquer et stocker les répliques à l'infini, Pour cela, Notre stratégie propose un service de suppression des répliques qui utilise la politique LRU (LastRecentlyUsed [18]). Chaque peer vérifie sa capacité de stockage libre, si celle-ci est inférieure à un seuil minimal S2, le processus de suppression des répliques est lancé.

Le peer demande des informations concernant le nombre de requêtes pour chaque donnnée. Le super-Peer répond en lui envoyant une liste ordonnée utilisant la stratégie LRU. Le peer décide alors d'effacer les données les moins populaires jusqu'à atteindre l'espace libre requis.

Algorithme 6: Procédure Suppression_Réplique ()

Begin
For (chaque $peer_i$) **do**
If(C_i< S2) **then**
Choisir un super-peer j ;
Demander une liste du nombre de requêtes pour chaque donnée au SPj;

Recevoir la liste des données les moins populaires (LRU);
While (espace libre insuffisant) **do**
Supprimer la donnée la moins populaire
End while;
Informer le super-peer j;
End if;
End for;
End.

Algorithme 4.6. Procédure de suppression

4.4 Etude Expérimentale et Evaluation

Afin de valider et évaluer l'approche proposée, nous avons développé un environnement de simulation peer-to-peer, implémenté avec le langage de programmation java. Les requêtes et les accès aux fichiers sont uniformément distribués dans tout le système. Nous avons effectué une série d'expérimentations dont les résultats et les interprétations font l'objet de cette section.

4.4.1 Paramètres de simulation

Pour tester les différentes expérimentations nous avons utilisé les paramètres de simulation définis dans le tableau 4.1.

Ces paramètres sont les mêmes pour l'ensemble des expérimentations :

Paramètre	**Intervalle**
Nombre de nœuds	[10..10000]
Nombre de super Peer	[1..500]
Nombre de ressources	[100..100000]
Nombre de répliques	[100…]
Taille des ressources	[10Mo..1Go]
Capacité de stockage	[10Mo..2To]
Bande Passante	[1Mhz..5Ghz]
Cycle Processeur	[10Kbps..2Gbps]
Nombre de cycles	[1...]
Nombre de requêtes	[1..]
Temps de simulation	[30s...]

Tableau 4.1 Paramètres de simulation

4.4.2 Expérimentation 1 :

La première évaluation consiste à comparer la stratégie proposée par rapport à une stratégie avec réplication implicite (réplication à la demande).

- **Impact du nombre de répliques créées (la disponibilité)**

La figure 4.5 montre les résultats obtenus selon le nombre de répliques créées pour chaque stratégie. On peut constater l'écart du nombre de répliques créées par rapport à la réplication implicite. La stratégie proposée minimise le nombre de répliques créées car elle ne réplique qu'après avoir vérifié la nécessité de la réplication et non à chaque demande. De ce fait, la création de répliques est contrôlée grâce à la modélisation du taux d'utilisation d'un objet.

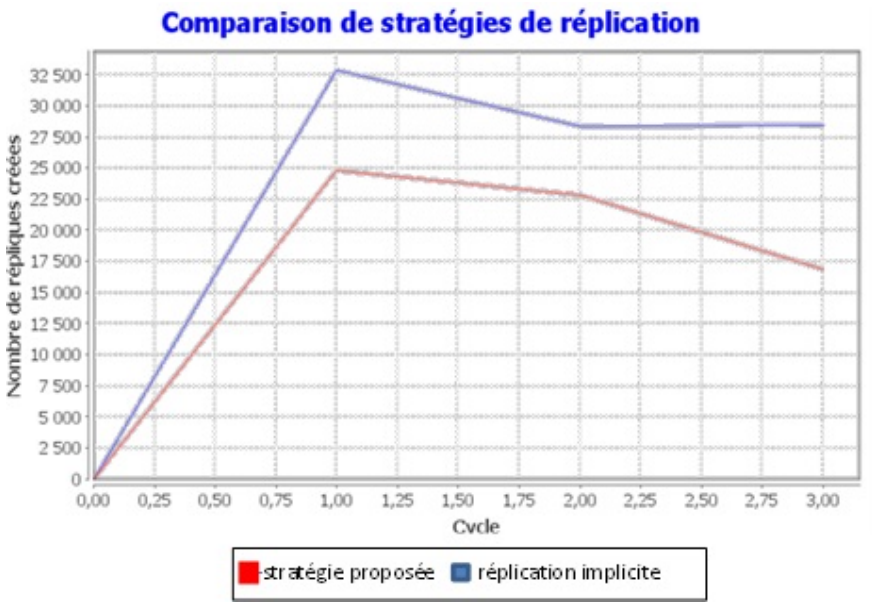


Figure 4.5. Nombre de répliques créées.
Stratégie proposée et réplication implicite

4.4.3 Expérimentation 2 : Temps de réponse moyen

Dans cette partie, nous étudions l'apport du service de clustering implémenté. Nous comparons pour cela, le temps de recherche des données dans un système P2P totalement décentralisé par rapport au modèle proposé qui utilise les super-Peers. D'après la figure 4.6, nous constatons qu'avec le clustering (réseau hybride) le temps de recherche diminue significativement car la donnée est recherchée seulement dans le cluster où la requête doit être exécutée. Par contre dans l'approche totalement décentralisée, le temps de recherche est nettement supérieur car le nombre de peers visités pour trouver la donnée peut augmenter de façon aléatoire.

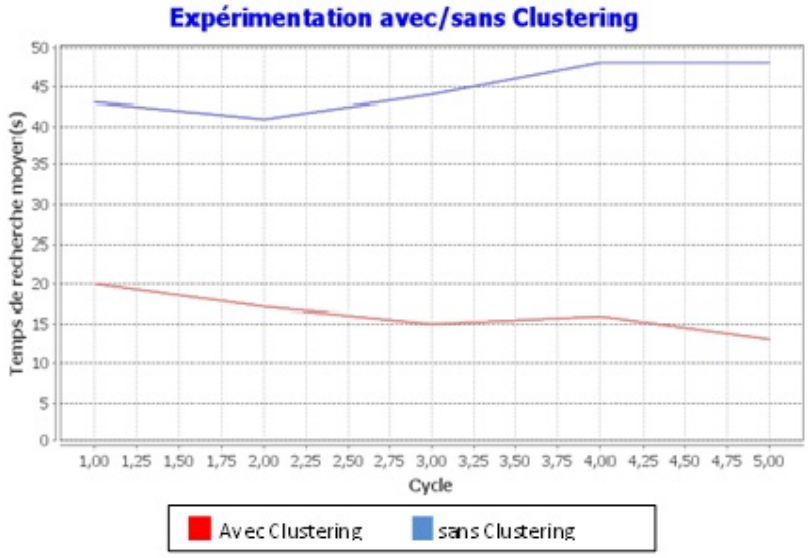


Figure 4.6. Temps de réponse moyen (Avec/Sans Clustering)

4.4.4 Expérimentation 3 : Impact de la suppression des données

Dans cette évaluation, nous avons testé l'apport de la politique de suppression pour le modèle proposé.

- **Nombre de requêtes perdues**

Pour une stratégie sans suppression, le nombre de requêtes perdues augmente par rapport à la stratégie avec suppression. Ceci peut s'expliquer par l'indisponibilité des données quand la réplication n'est plus possible sur les sites concernés et ceci par manque d'espace de stockage.

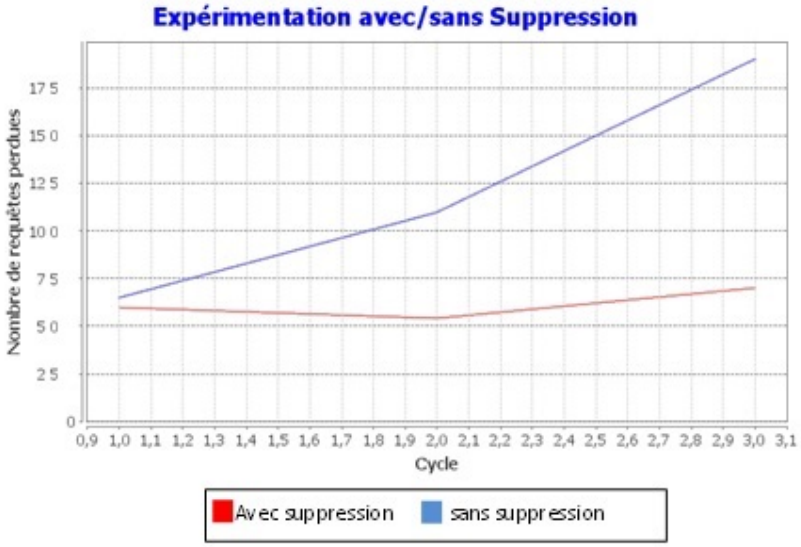


Figure 4.7. Nombre de requêtes perdues (Avec/Sans suppression)

- **Taux de stockage disponible**

Les résultats de la figure 4.8 montrent la perte de l'espace de stockage progressive pour la stratégie sans suppression, vu que les répliques sont placées sur les peers et n'y sont plus supprimées. Par contre pour la stratégie avec suppression, l'espace de stockage reste à peu près stable grâce à la stratégie LRU (Least RecentlyUsed) utilisée tout en gardant une certaine disponibilité des données afin de maintenir un temps de recherche raisonnable.

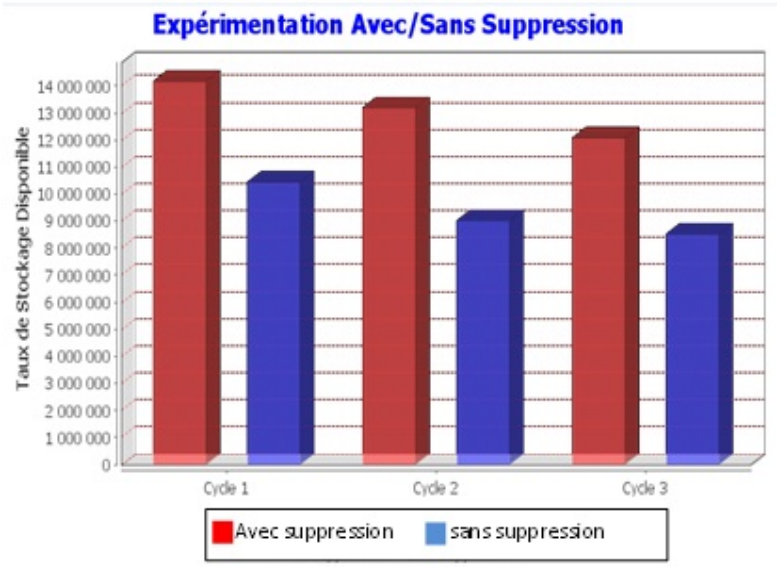


Figure 4.8. Taux de stockage disponible (Avec/Sans suppression)

4.4.5 Expérimentation 4 : Impact de la gestion des défaillances

Dans cette dernière expérimentation, nous avons mesuré le nombre de requêtes perdues avec le service de gestion de la défaillance et sans l'utilisation de ce service.

La figure 4.9 montre l'apport de notre approche avec tolérance aux pannes dans chaque cycle. Nous constatons ainsi une diminution progressive du nombre de requêtes perdues, qui peut s'expliquer par le fait que si un super-peer tombe en panne, un autre le remplace, ce qui réduit le nombre de requêtes perdues.

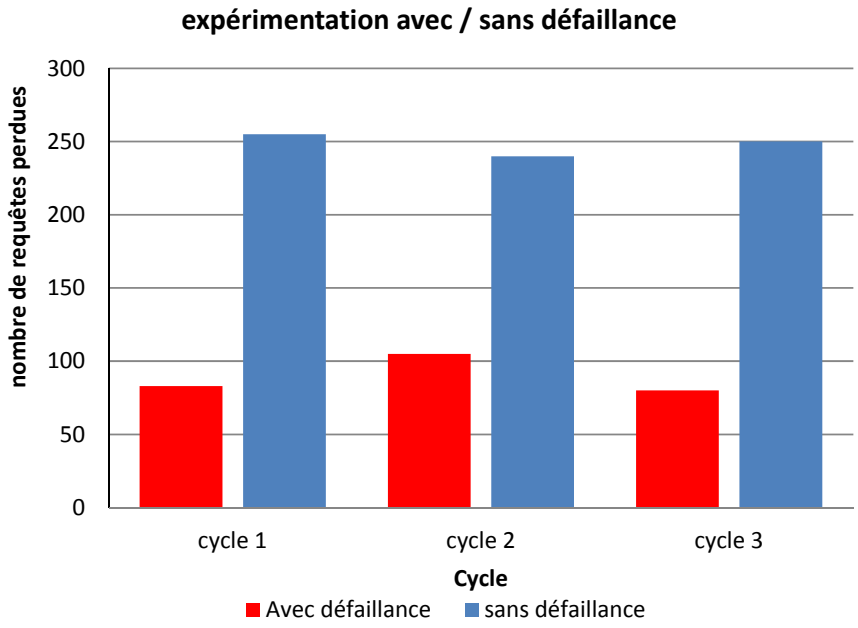


Figure 4.9. Nombre de requêtes perdues (Avec/Sans défaillance)

4.5 Conclusion

Ce chapitre a permis de présenter, d'analyser et de mettre en évidence le fonctionnement de l'approche de réplication proposée.

La stratégie de réplication détecte la nécessité de réplication, calcule le nombre minimal de répliques, tout en assurant la disponibilité des données, leur placement équitable et la libération d'espace de stockage en utilisant la politique LRU. Elle est basée sur la notion de popularité et du taux de sollicitation des données. La capacité de stockage de chaque peer est prise en compte pendant le processus de réplication.

Comme nous nous intéressons aux systèmes non structurés et hybrides, Nous utilisons la notion de super peer pour regrouper les peers en un ensemble de clusters en fonction de leur proximité. Nous procédons à l'élection de plusieurs super-peers dans un même cluster et ceci pour renforcer la robustesse et la fiabilité du réseau. Ces supers peers assurent des fonctions essentielles pendant le processus de réplication.

Enfin, nous soulignons aussi que l'évolutivité du système n'altère en rien le fonctionnement de la stratégie proposée, qui s'adapte selon le nombre de peers dans le système.

Chapitre 5

Stratégie2 :

Réplication de données basée sur un modèle de coût

5. Stratégie 2 : Réplication de données basée sur un modèle de coût

5.1 Problématique

Les grilles de données exigent la gestion d'un nombre massif de données afin de les rendre accessibles aux clients qui les réclament. Les stratégies de réplication des données représentent un facteur important vis-à-vis des performances des services proposés par une grille. Ces stratégies doivent répondre à des questions essentielles telles que : 1) comment équilibrer le nombre de répliques dans les sites de la grille ? En effet, l'augmentation du nombre de répliques conduit à accroître la disponibilité des données et la fiabilité, mais l'espace de stockage sera également augmenté et peut devenir insuffisant. Par conséquent, un bon équilibre entre le nombre de répliques est nécessaire. 2) Où la réplique doit être placée ? Placer les nouvelles répliques sur le site de l'emplacement approprié peut favoriser la réduction de la consommation de bande passante du réseau et réduit donc le temps de réponse. L'idée principale est de conserver les données à proximité de l'utilisateur afin de rendre l'accès efficace et rapide. Mais le comportement dynamique des utilisateurs du réseau rend plus difficile la prise de décisions en ce qui concerne les réplications de données pour atteindre l'objectif de disponibilité maximale [19,31]. La stratégie de réplication est guidée par des facteurs tels que la demande des données, les conditions du réseau, le coût de transfert et le coût de stockage. Afin de maximiser le potentiel des gains de la réplication de fichiers, une stratégie de placement des répliques est très importante. Un service de placement des répliques est un élément de l'architecture des grilles de données qui décide où une réplique de fichier doit être placée dans le système.

Dans ce chapitre, nous proposons une stratégie de réplication de données distribuée pour le placement dynamique de répliques dans la grille de données. La décision de réplication et de placement est déterminée en fonction d'un modèle de coût qui dépend de plusieurs paramètres à savoir : la puissance des sites, le coût de transfert, la distribution des répliques entre les sites, la charge des sites, la bande passante entre les sites et la taille des répliques. Dans cette approche, les données sont répliquées uniquement si leur coût d'accès est supérieur au coût de leur réplication.

La grande majorité des stratégies proposées [2,33,60,78,79,81] dans ce cadre néglige la capacité de stockage des machines qui constituent les sites, or c'est un facteur très important qu'il faut prendre en compte car on ne peut pas répliquer et stocker les répliques à l'infini. De plus nous proposons un service d'entretien qui redistribue les répliques en les transférant vers d'autres nœuds évitant ainsi de les supprimer faute d'espace mémoire.

5.2 Architecture du modèle proposé

Dans le cadre de ce travail, nous proposons une topologie logique formée de clusters. Un cluster représente un ensemble de ressources informatiques homogènes reliées par un réseau et localisées géographiquement dans une même organisation (Campus universitaire,

Centre de calcul, Entreprise ou Personne individuelle). Cet ensemble forme un domaine d'administration autonome, uniforme et coordonné [48, 49].

Chaque nœud de cette structure comporte au moins deux éléments de base : l'élément de stockage (SE) et l'élément de calcul (CE). La premier doit recevoir et traiter la demande du client en utilisant la réplique stockée dans SE. Le modèle proposé reflète l'architecture réelle de la grille et permet de minimiser les accès inter sites (inter-clusters) à plus forte latence. Il convient de noter que la largeur de bande entre les clusters (inter-clusters) est inférieure à la largeur de bande à l'intérieur d'un cluster (intra-cluster). Le clustering est un moyen efficace de réduire les délais de communication.

5.2.1 Topologie de la grille

La grille de donnée est vue comme une fédération de clusters ou chaque cluster comporte un ensemble de nœuds et est géré par un nœud particulier appelé « leader ».

Comme le montre la figure 5.1, le modèle distribué de la grille que nous proposons, comporte deux couches : une couche supérieure et une autre inférieure.

Cette modélisation nous permet de travailler sur un modèle logique moins complexe composé d'un ensemble de clusters, au lieu de travailler sur un modèle physique où le nombre de nœuds est très grand.

- **La couche supérieure :** cette couche est constituée d'un ensemble de clusters. Les clusters sont organisés par un graphe (X, U), où X= {C1, C2,…, CN} est l'ensemble de tous les clusters et U est un ensemble donné d'arêtes virtuelles entre les nœuds (c'est-à-dire, clusters) dans X. Les arêtes dans U sont bidirectionnelles.
- **La couche inférieure :** c'est l'ensemble des nœuds. Chaque nœud à son propre identifiant $\mathbf{ID_{nœud}}$, une unité de stockage **US**, et une unité de calcul **UC**.

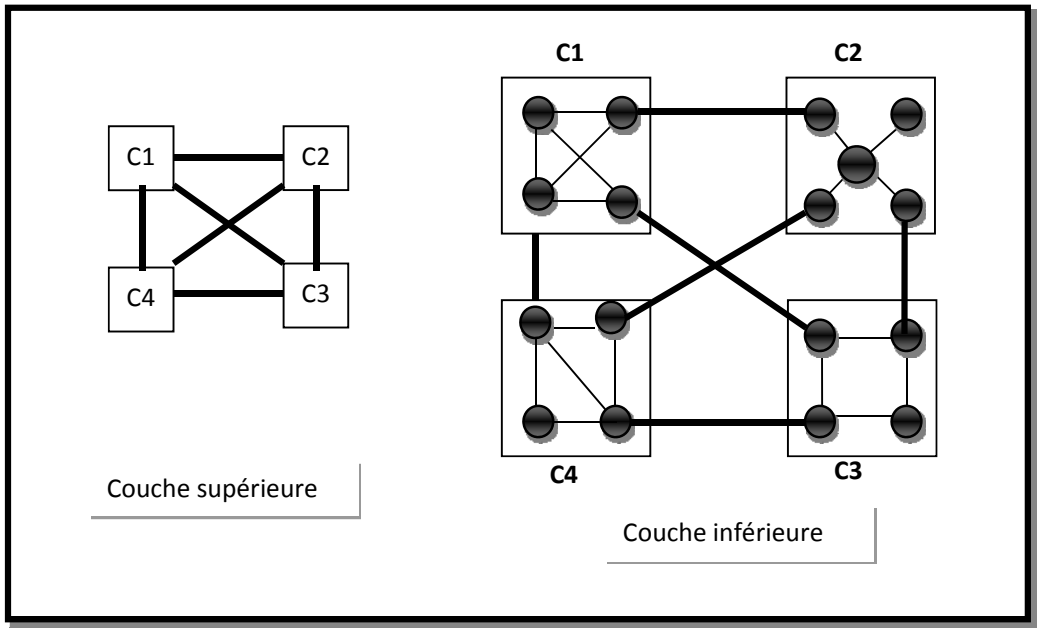


Figure 5.1. Topologie à deux couches.

La figure 5.1 montre un exemple de la topologie à deux couches comportant quatre clusters C1, C2, C3, C4. Les lignes minces représentent les liens intra-cluster et les lignes épaisses les liens inter-clusters.

5.3 Modèle de réplication

5.3.1 Hypothèses

- La grille est organisée sous forme de clusters.
- Chaque cluster a son propre identifiant et un espace de stockage limité.
- Chaque cluster est composé de plusieurs nœuds dont un seul qui le gère appelé « *FirstLeader*».
- Chaque nœud a son propre identifiant et un espace de stockage limité et une table qui contient les meilleurs chemins vers les autres nœuds.
- Un nœud peut tomber en panne.
- Le *FirstLeader* et le *SecondLeader* sont des nœuds ordinaires (le *SecondLeader* représente le remplaçant du *FirstLeader* en cas de panne).

5.3.2 Notations

- **BP (i,j)** : le débit de la bande passante entre le nœud *i* et le nœud *j*.
- **Way(i,j)** : l'ensemble des nœuds rencontrés le long du chemin du nœud *i* jusqu'au nœud *j*.
- **File(i)** : l'ensemble des requêtes présentes dans la file d'attente du nœud *i.*
- **T_replic(i) :** la taille de la réplique *i*.
- **T_req(i,j) :** la taille de la requête *i* envoyée par un nœud *j*.
- **T_rep(i,j) :** la taille de la réponse *i* envoyée par un nœud *j*.
- **V_proc(i) :** la vitesse du processeur du nœud *i.*
- **Mid_line :** 14bits
- **C_replic(a,i,j) :** le coût de réplication d'une donnée *a* du nœud *i* vers le nœud *j.*
- **C_req(i,a,b) :** le coût de transmission de la requête *i* d'un nœud *a* vers un nœud *b*.
- **C_rep(i,a,b) :** le coût de transmission de la réponse *i* d'un nœud *a* vers un nœud *b*.
- **C_att(i,j) :** le coût d'attente et traitement d'une requête *i* dans une file d'attente du nœud *j*.
- **C_trait(i,j) :** Coût de traitement de l'opération de lecture sur la donnée *i* située au nœud *j.*
- **C_acc(i,j) :** le coût d'accès à une donnée *i* du nœud *j*.
- **ES_total(i) :** l'espace de stockage total d'un nœud *i.*
- **ES_libre(i) :** l'espace de stockage libre d'un nœud *i.*
- **ES_cluster(i) :** l'espace de stockage total du cluster *i.*
- **ES_cluster_libre(i) :** l'espace de stockage libre du cluster *i.*
- **Alpha :** un paramètre ajustable qui sert à vérifier si l'espace de stockage libre est suffisant pour la réplication d'une donnée ou non.
- **P_choix(i) :** la permission de répliquer une donnée dans le site *i.*

5.3.3 Architecture fonctionnelle du modèle de réplication

La figure 5.2 montre les différents services conçus dans le cadre de la stratégie proposée. D'abord, une étape préliminaire précède l'étape de la réplication. Il s'agit d'un pré

traitement qui consiste en l'implémentation d'un algorithme d'élection des gestionnaires des différents clusters. L'étape de traitement comprend le processus de réplications avec les différents services implémentés.

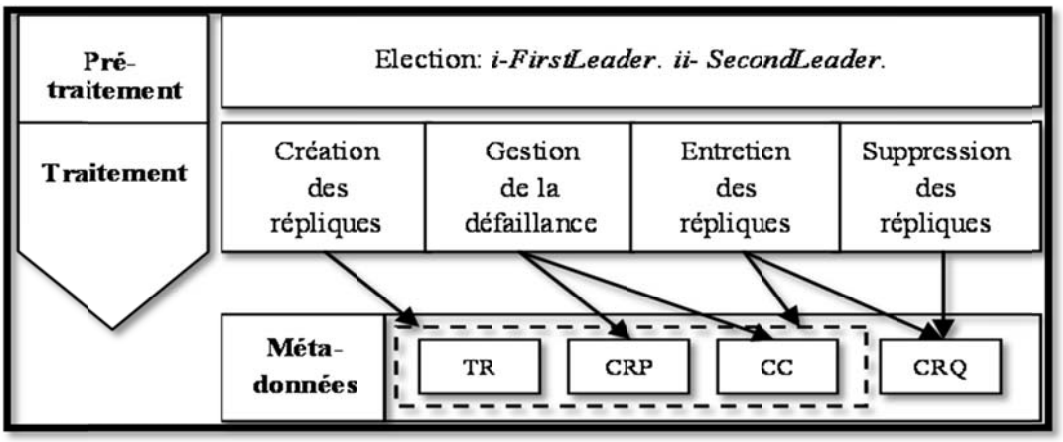


Figure 5.2. Architecture générale du modèle de réplication

5.3.3.1 Métadonnées

Dans un environnement à grande échelle, l'utilisation de métadonnées est indispensable pour exploiter pleinement les ressources qui composent le système. Les métadonnées sont des informations collectées par chaque gestionnaire de cluster. Ces informations sont nécessaires pour le processus de réplication. Elles sont représentées sous forme de tables et catalogues comme le montre le tableau ci-dessous.

Table / Catalogue	Contenu
TR (Table de routage)	1. Identificateurs de clusters, 2. La bande passante inter et intra-cluster 3. Meilleur chemin entre deux nœuds
CRP (Catalogue des répliques)	1. Identificateurs, emplacements et tailles des répliques.
CC (Catalogue du cluster)	1. Identificateurs des nœuds d'un cluster, 2. Espace de stockage total 3. Liste des requêtes en attente
CRQ (Catalogue des requêtes)	1. Tailles des requêtes, 2. Répliques nécessaires à l'exécution des requêtes

Tableau 5.1 Métadonnées

5.3.3.2 Election

Le but de cette phase de prétraitement est de désigner un gestionnaire primaire du cluster (appelé « *FirstLeader* ») et un autre secondaire (appelé « *SecondLeader* »). Le rôle du *FirstLeader* consiste à :

- Diriger les requêtes vers les nœuds qui ont les répliques nécessaires pour leurs traitements.
- Créer, et supprimer des répliques.
- Prendre en charge l'entretien des répliques.
- Dialoguer avec les autres clusters.
- Elire un *SecondLeader* et le remplacer par un autre en cas de pannes.

Ces fonctions sont assurées à l'aide des quatre catalogues qu'il stocke. Cependant, le *SecondLeader* doit vérifier périodiquement l'état du first leader pour voir s'il est opérationnel ou pas. Le *SecondLeader* détient une copie des quatre catalogues. L'avantage principal du *SecondLeader* est d'éviter le temps d'élection d'un leader en cas de pannes (absence du temps d'inactivité du système) et de conserver les informations nécessaires en cas de défaillance du *FirstLeader*

Chaque nœud de la grille a un identifiant unique. Le processus de d'élection est lancé par le nœud de plus grand identifiant dans chaque cluster. L'algorithme calcule alors la distance entre chaque nœud et tous les autres nœuds en termes de débit de la bande passante en suivant le plus court chemin stocké dans les tables de routage comme le montre l'exemple ci-dessous. Le résultat de l'élection fixe comme « First leader » le nœud qui obtient la plus petite distance et comme « second leader » le suivant. Cette étape consiste à choisir le nœud le plus proche des autres nœuds d'un même cluster en terme de bande passante. Les calculs sont basés sur les equations (1) et (2). Cette stratégie d'élection permet un acheminement plus rapide des requêtes.

$$\mathbf{Somme_A(i)}=\sum_{j=1} \text{Dist}(i,j) \;/\; p\text{-}1 \qquad (1)$$

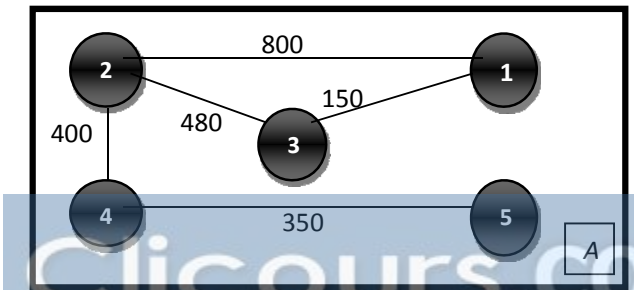
Tel que P: nombre total des nœuds du cluster A, j≠i.

$$\mathbf{Dist(i,\ j)} = \sum_{k=1}^{n-1} 1/BP\ (k,k+1) \qquad (2)$$

Tel que k ϵ Way(i, j) et n: card(Way(i,j))

Exemple :

Soit un cluster « A » qui contient 5 nœuds :



On calcule la distance entre chaque nœud et tous les autres en termes de débit de bande passante en suivant le plus court chemin qui est déjà stocké dans la table de routage de celui-ci.

- **Le nœud 1 :**

La table routage du nœud 1 est remplie comme suit :

Nœud source	Nœud arrivée	Le plus court chemin (WAY)
1	2	1, 2
1	3	1, 2, 3
1	4	1, 2, 4
1	5	1, 2, 4, 5

On calcule la somme

Somme(1) = (Dist(1,2) + Dist(1,3) + Dist(1,4) + Dist(1,5)) / (P-1). Tel que *P* est le nombre des nœuds du cluster A

Somme(1) = (0,0012+ 0,003 + 0,0033 +0,006) / 4 = 0,00375.

De la même façon, on calcule les sommes des autres nœuds :

Somme(2) = (0,0012+ 0,002+ 0,0025+ 0,0053)/4 =**0,00275**

Somme(3) = (0,003+ 0,002+ 0,0045+ 0,0074)/4 =0,00422

Somme(4) = (0,0033+ 0,0025+ 0,0045+ 0,0028)/4 =**0 ,00327**

Somme(5) = (0,006+ 0,0053+ 0,0074+ 0,0028)/4 =0,00537

D'après les résultats, le nœud 2 sera proclamé «FirstLeader », et le le nœud *4*« SecondLeader »

Algorithme 1: Election des leaders

Begin
For Tout cluster (i) ϵ grille **do**
For Tout nœud (j) ϵ $cluster_i$ **do**
Somme (j) ← 0 ;
For Tout nœud (k) ϵ $cluster_i$ **do**
If j≠k then

$$Dist(j,k) = \sum_{l=1}^{n-1} 1/BP(l, l+1) \quad // \; l \in Way(j,k) \text{ et } n: card(Way(j,k)).$$

Somme (j) ← Somme (j) + Dist (j, k);
End if;
End for;
Election. Ajouter ($nœud_j$, Somme (j));
End for;
Ordre Ascendant (Election. Somme);

$Cluster_i$.FirstLeader ← Election.nœud[0] ;
$Cluster_i$.SecondLeader ← Election.nœud[1] ;
End for;
End.

Algorithme 5.1 Election des leaders.

5.3.3.3 Création des répliques

Initialement, les requêtes sont soumises à la grille, le gestionnaire de chaque cluster soumet les requêtes aux différents nœuds. Si un nœud réclame une donnée qui n'existe pas dans son cluster, le processus de réplication commence : Dans la stratégie proposée, une réplique est créée dans deux cas :

- Si un nœud demande une réplique qui n'existe pas dans son cluster, le « *FirstLeader*» de ce cluster lance une requête de recherche vers les autres clusters en commençant par le plus proche en terme de débit de bande passante, et crée une réplique de la donnée dans le nœud le moins chargé.
- Si un nœud demande une donnée qui existe dans son cluster, le processus de décision est entamé pour décider si une réplication locale de cette donnée doit se faire ou le nœud doit accéder à distance. Un algorithme calcule le coût d'accès à la donnée (ou une réplique) ainsi que le coût de sa réplication, le « *FirstLeader*» crée une réplique seulement si le premier coût dépasse le deuxième.

Avant la réplication, une vérification de l'espace de stockage du nœud recevant la réplique est nécessaire. Pour cela, on utilise la formule suivante [20]:

$$P\_choix(i) = \begin{cases} 0, & ES\_libre(i) * Alpha < T\_replic(j) \\ 1, & ES\_libre(i) * Alpha \geq T\_replic(j) \end{cases} \quad (3)$$

Où « Alpha » est un paramètre ajustable utilisé pour s'assurer qu'il y a un espace suffisant pour stocker les fichiers temporaires ou les résultats des requêtes après ou en cours de traitement. i est le nœud concerné par la réplication de la donnée j.
Dans le deuxième cas (citer ci-dessus), la décision de réplication se fait en comparant deux coût :

- Le coût d'accès à une donnée *i.*
- Le coût de réplication de la donnée *i.*

Le coût d'accès à une donnée *i* d'un nœud *j* est calculé par la formule suivante :
On suppose qu'une requête *a* utilisant la donnée *i* est formulée par le nœud *k*. Cette requête sera traitée par le nœud *j* en rendant une réponse *c*.

C_acc(i,j) = C_req(a,k,j) + C_att(a,j)+ C_rep(c,j,k). (4)

Tel que :

C_req(a,i,j) = T_req(a) * $\sum_{K=1}^{n-1}$1/BP (k,k+1).Tel que k ϵWay(i,j) et n: card(Way(i,j)). (5)

C_rep(c,i,j) = T_rep(c) * $\sum_{K=1}^{n-1}$1/BP (k,k+1). Tel que k ϵWay(j,i)et n: card(Way(i,j)). (6)

C_att(a,j) = $\sum_{K=1}^{n}$C_trait(k,j). Tel que k ϵ file(j) et n: card(file(j)).Tel que : (7)

C_trait(k,j) = [(1 / (V_proc(j)*10^9))]*[(T_replic(k) / Mid_line)]**[41]**. (8)

Soit :

« j » : le nœud qui contient la réplique *i.*

« l » : le nœud le moins chargé.

Le coût de réplication d'une donnée i du nœud j vers le nœud l est calculé comme suit :

C_replic (i,j,l) = C_acc(i,l) + [T_replic(i) * $\sum_{K=1}^{n-1}$ 1/BP (k,k+1)]. (9)

Tel que:k ϵWay(j,l) et n: card(Way(j,l)).

Si le coût de d'accès à une donnée est supérieur au coût de sa réplication, la donnée sera répliquée dans le nœud le moins chargé.

Algorithme2 : Création des répliques

Begin
Arrivée d'une requête (a,i) ;
Réplique$_T$ ← requête (a,i).réplique.
Chercher le nœud$_j$ qui contient la réplique « réplique$_T$ ».
Chercher le nœud$_k$ qui est le nœud le moins chargé du cluster « nœud$_i$.cluster ».
If (réplique$_T$ ϵ CRP (nœud$_i$.cluster)) **then**

C_req(a,i,j) ← T_req(a) * $\sum_{m=1}^{n-1}$ 1/BP (m,m+1); // m ϵ Way(i,j) et n: card(Way(i,j)).

C_rep(c,i,j) ← T_rep(c) * $\sum_{m=1}^{n-1}$ 1/BP (m,m+1); // m ϵ Way(i,j) et n: card(Way(i,j)).

C_att (a,j) ← $\sum_{m=1}^{n}$ C_trait(m,j); // m ϵ file(j) et n: card(file(j)).

C_acc (rélique$_T$, j) ← C_req (a,i,j) + C_rep(c,j,i) + C_att(a,j).
Calculer (C_req(a,i,k));
Calculer(C_rep(c,k,i));
Calculer(C_att(a,k));

C_replic (réliqueT,j,k) ← C_acc(réliqueT, k) + [T_replic(réliqueT)* $\sum_{m=1}^{n-1}$ 1/BP (m,m+1)].
// m ϵ Way(j,k) et n: card(Way(j,k)).
If (C_replic(réliqueT,j,k) < C_acc(réliqueT,j)) **then**
noeudk.CréerRéplique (répliqueT) ;
End if;
Else
Chercher le clusterP qui contient la réplique « répliqueT ».
Chercher dans clusterP le nœudPJ qui contient cette réplique.
noeudk.CréerRéplique (répliqueT) ;
End if;
End.

Algorithme 5.2. Création des répliques.

5.3.3.4 Suppression des répliques

Une suppression est nécessaire lorsqu'une création de réplique est requise alors que l'espace libre de stockage est insuffisant. Dans notre approche, nous avons opté pour une combinaison de deux politiques de suppression : la réplique la moins populaire et la réplique la plus ancienne. Pour cela, on attribue à chaque réplique un score (poids) qui correspond à son ordre d'utilisation, ensuite on additionne les scores de chaque réplique, et on supprime celle qui a le score le plus faible.

NB : les répliques concernées par la suppression sont celles qui sont stockés dans au moins deux nœuds : Une donnée unique dans la grille ne sera jamais supprimée.

Exemple :

Soit A, B, C, D des répliques qui ont été utilisées dans l'ordre suivant et stocké dans un vecteur T :

T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
A	B	B	C	A	C	C	A	A	D
1	2	3	4	5	6	7	8	9	10

La requête «A» est apparue dans la 1ère, la 5ème, la 8ème, et la 9ème position, donc son score est :

Score (*A*) = 1+5+8+9 = **23**.Score (*B*) = 2+3 = **5.**Score (*C*) = 4+6+7 = **17.**Score (*D*) = **10.**

La réplique qui sera supprimée par notre politique de gestion est la réplique B. Si nous avions utilisé la politique « la réplique la plus ancienne » alors la réplique A serait éliminée bien qu'elle soit la plus populaire, ou récemment utilisée, par contre, si nous avions utilisé la politique « la réplique la plus populaire » c'est la réplique D qui serait supprimée bien qu'elle soit la dernière à être utilisée et la probabilité qu'elle soit à nouveau demandée est grande.

Algorithme3 : Suppression des répliques

Begin
K ← 1;
For toute donnée D do
Somme ← 0 ;
For (i=1, i=T.long, i++) do
If (T[i] = D) then
Somme→ Somme + T[i];
End if;
End for;
Score [k]→ Somme;
K ← K+1;
End for;
Score.Ordre Ascendant;
//Supprimer les données relatives aux plus petist scores jusqu'à avoir l'espace requis
While ((nœud.espace_libre*Alpha ≤ réplique_à_crée.taille) **do**
If (Donnée not unique) then
Supprimer donnée ;
End if;
End while;
End.

Algorithme 5.3. Suppression des répliques.

5.3.3.5 Gestion de la défaillance

La grille est un environnement dynamique où les nœuds peuvent se déconnecter fréquemment, ceci affecte donc la disponibilité des données. Pour cela une gestion des défaillances des nœuds est requise. Pour la maintenance du système, nous utilisons les messages AYA() (Are You Alive). La défaillance d'un nœud ordinaire ou du gestionnaire du cluster est traitée différemment car ils n'ont pas les mêmes rôles. Le *firstLeader* envoie périodiquement des messages de vie aux nœuds du cluster et fixe un délai, s'il ne reçoit pas un ACK() après l'expiration du délai, il envoie un autre message, si aucune réponse n'est reçue après le troisième message, le *FirstLeader* considère ce nœud comme défaillant.

Le *SecondLeader* est le seul nœud du cluster qui envoie des messages aya au FirstLeader, s'il n'acquitte pas réception après la troisième message, il le considère comme défaillant.

- **Défaillance du FirstLeader :** En cas de défaillance du *first leader*, le *SecondLeader* prend sa place et lance une opération d'élection pour choisir un autre nœud comme *SecondLeader*. Il lui envoie alors les catalogues nécessaires à la gestion du cluster, puis envoie aux nœuds de son cluster et aux *FirstLeader*des autres clusters un message « NewLeader » pour les informer du changement.

- **Défaillance du SecondLeader :** le *firstLeader* choisira, parmi les autres nœuds, un*SecondLeader* et lui envoie les catalogues nécessaires.
- **Défaillance d'un nœud :** dans ce cas, ce nœud ainsi que toutes ces informations le concernant seront supprimées des catalogues du *FirstLeader*.

5.3.3.6 Entretien des répliques

Ce service contribue à la maintenance des données dans les clusters et permet de libérer de l'espace de stockage nécessaire pour répliquer. Il consiste à redistribuer les répliques si les performances du système se dégradent d'une manière significative. Dans notre cas, la diminution de l'espace de stockage d'un cluster jusqu'à 10% de l'espace globale au moins est la condition de dégradation du système. L'objectif est d'éviter au maximum la suppression des répliques en les transférant vers d'autres clusters.
L'algorithme sélectionne les répliques à transférer en utilisant le même principe que celui de la suppression (compromis entre deux modèles). Le cluster le plus proche en termes de débit de bande passante recevra ces répliques à condition que le transfert ne mène pas à une dégradation du système de ce cluster. Ce service contribue à une amélioration significative du temps de réponse aux requêtes quand une réplication est nécessaire.

La condition de dégradation est :

$ES\_cluster\_libre(i) \leq 10\% (ES\_cluster(i))$

5.4 Expérimentation et Evaluation

Afin d'évaluer le comportement de l'approche proposée et valider les résultats obtenus, nous avons effectué une série d'expérimentations en utilisant le simulateur OptorSim [42,45,46] qui constitue un bon environnement pour le test des stratégies dynamiques de réplication.

Dans ces expérimentations, nous nous concentrons sur l'importance des services que nous avons présentés dans la section précédente ainsi que sur les gains qu'ils apportent selon les métriques suivantes :

- les temps de réponse aux requêtes
- le nombre des messages échangés entre nœuds du système
- les espaces de stockages,
- le nombre de requêtes créées /perdues.

Dans la plupart des expériences, nous comparons les résultats en utilisant quatre stratégies implémentées

- La stratégie proposée
- La stratégie proposée sans entretien
- La réplication implicite
- La réplication statique

Le tableau ci-dessous montre les paramètres pris en compte lors des simulations :

Paramètres	Valeurs
Nombre de clusters	[2….]
Nombre de noeuds	[10…1000]
Bande passante inter-clusters	[1..99] Mb/S
Bande passante intra-cluster	[100..1000] Mb/S
Capacité de stockage des nœuds	[3.0..45] Gb
Vitesse processeur	[3.0..4.0] GHz
Nombre de données	100
Nombre de répliques	100
Taille des répliques	[1..4] Gb
Nombre de requêtes	45000
Taille des requêtes	350 Ko
Durée d'un cycle	5 Sec

Tableau 5.2 Paramètres de simulation

5.4.1 Temps de réponse moyen

C'est le temps moyen d'exécution d'une requête, c'est à dire le temps total d'exécution de toutes les requêtes divisé par le nombre de requêtes. Il inclut le temps propre à l'exécution plus le temps de communication dans le réseau (transfert de données) et le temps passé dans les files d'attentes. La minimisation de ce paramètre est importante si on vise à améliorer les performances de la grille

Les figures 5.3 et 5.4 montrent l'impact de la réplication et de l'entretien sur la moyenne des temps de réponse. On peut noter la grande différence entre la réplication statique et la stratégie proposée. Ces résultats montrent aussi que l'intégration du service entretien des répliques améliore le temps de réponse des requêtes.

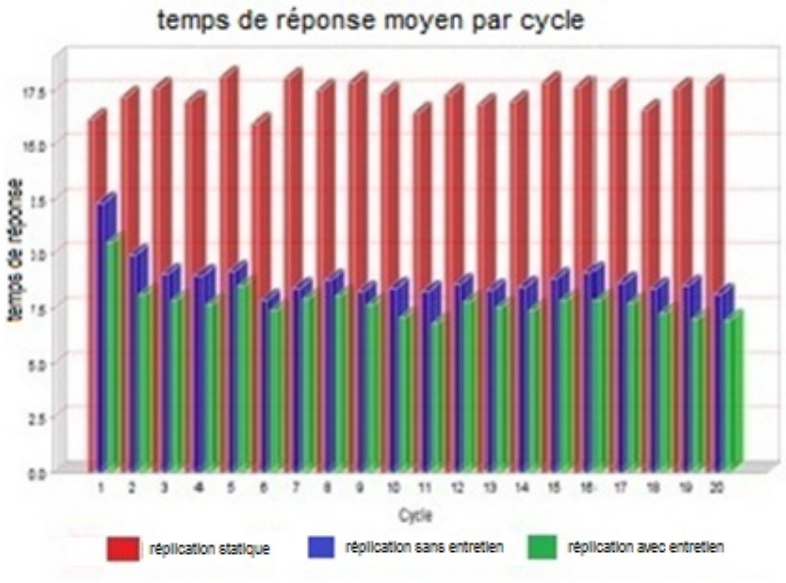


Figure 5.3. Temps de réponse moyen par cycle

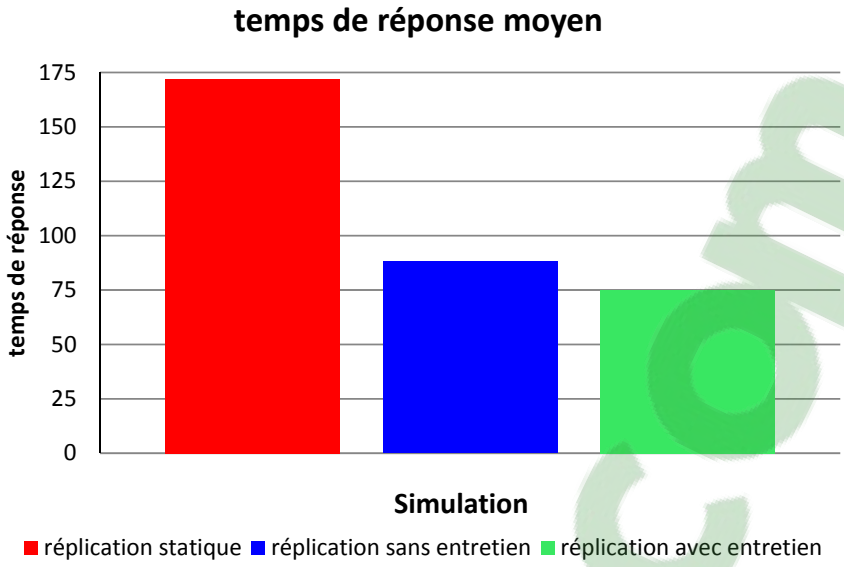


Figure 5.4. Moyenne du temps de réponse global

5.4.2 Nombre de répliques créées :

Ce paramètre désigne le nombre de répliques créées durant l'exécution des requêtes. Si ce nombre est important, il augmentera le temps d'exécution des requêtes (temps de réplication + transfert de données). Ce nombre est amélioré par un placement stratégique des répliques.

Cette deuxième évaluation compare la stratégie proposée à une stratégie avec réplication implicite (ie une stratégie qui réplique à la demande).

La figure 5.5 montre les résultats obtenus selon le nombre de répliques crées pour les deux stratégies. En effet, la stratégie proposée réduit de façon significative le nombre de répliques tout en assurant la disponibilité des données.

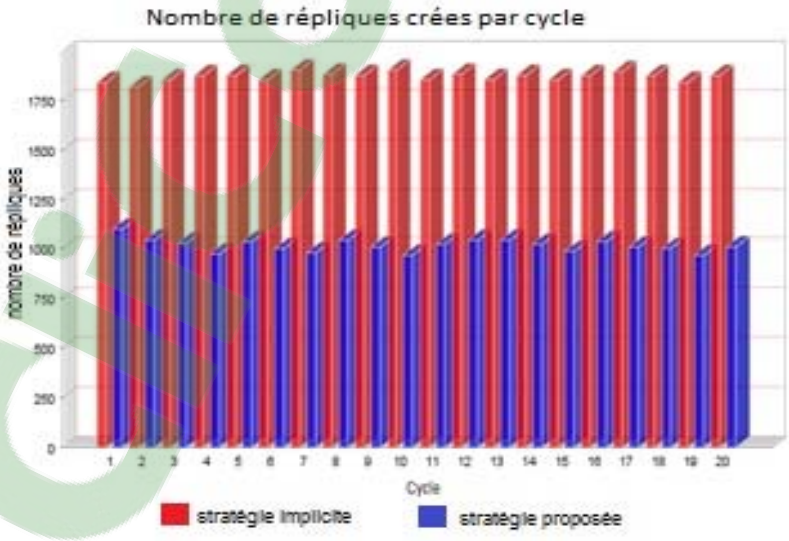


Figure 5.5. Nombre de répliques créées

5.4.3 Consommation des ressources réseaux :

La consommation des ressources réseaux est un paramètre qui indique l'occupation et l'usage du réseau. Il est défini par la formule suivante :
Usage réseau = (nombre d'accès distants + nombre de réplications) / nombre d'accès locaux[42,45].
Le nombre d'accès locaux représente le nombre de fois où la donnée est présente dans l'unité de stockage local du site initialement ou après réplication. Le temps de réponse est meilleur quand ce nombre est élevé. Le nombre d'accès distants représente le nombre de fois où on accède aux données nécessaires à distance. Plus le nombre d'accès distants et le nombre de réplications sont élevés, plus le mouvement dans le réseau augmente et vice versa.
La figure 5.6 montre l'impact de la stratégie proposée sur l'usage du réseau.

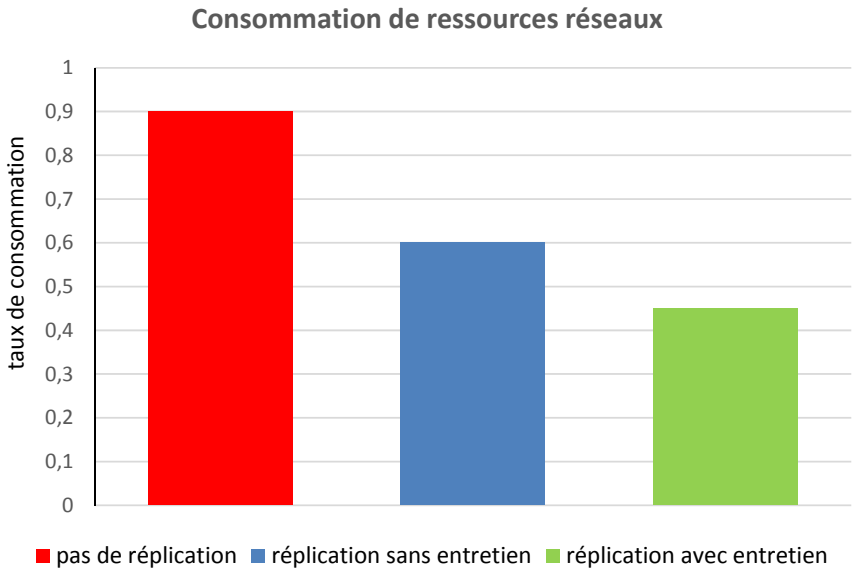


Figure 5.6. Consommation de ressources réseaux

5.4.4 Nombre de messages échangés :

Dans cette section, nous étudions l'apport d'une architecture basée sur les clusters. La figure 5.7 montre que le nombre de messages intra-cluster est beaucoup plus important que celui entre les clusters pour les deux stratégies. Nous pouvons conclure que le clustering minimise l'utilisation du réseau WAN. Nous constatons également que le service d'entretien (maintenance) réduit le nombre de messages inter-clusters car il redistribue régulièrement les répliques et contribue ainsi à augmenter la disponibilité des données dans les clusters. Finalement les courbes de la figure 5.8 montrent qu'en général, le clustering favorise la communication intra-cluster où la bande passante est plus élevée que celle entre les clusters.

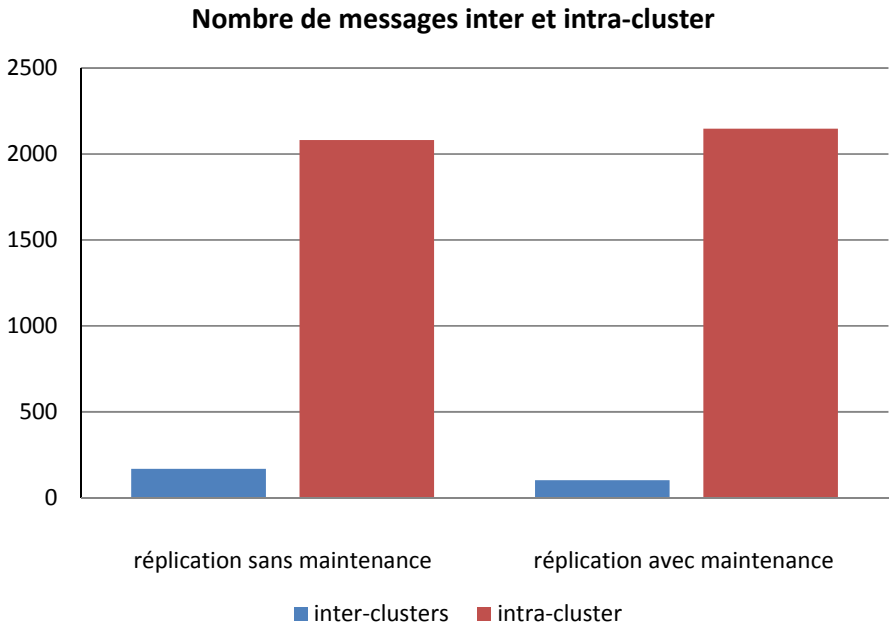


Figure 5.7. Nombre de messages inter et intra-cluster

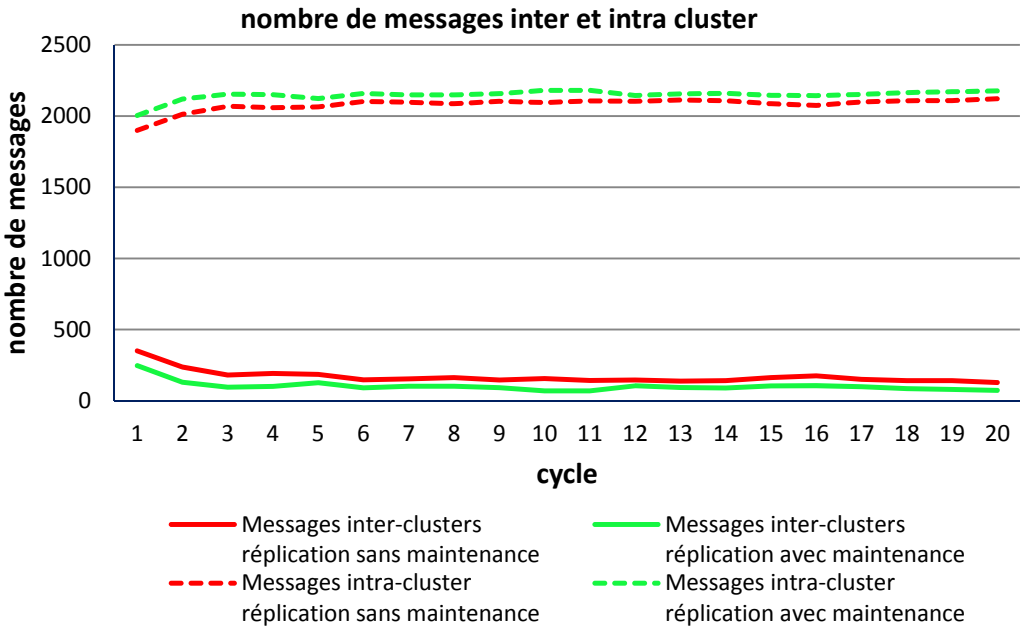


Figure 5.8. Nombre de messages dans la grille

5.4.5 Nombre de requêtes perdues :

La dernière évaluation compare le nombre de requêtes perdues selon la stratégie proposée avec ou sans service de gestion des pannes. La figure 5.9 montre le gain apporté, il réduit de façon significative le nombre de requête perdues. En effet, la stratégie implémentée gère la défaillance des nœuds et particulièrement celle du *FirstLeader*. Si le gestionnaire d'un cluster tombe en panne, le *SecondLeader* le remplace épargnant ainsi le temps d'élection d'un nouveau leader d'une part et la paralysie du cluster d'autre part. Ainsi les requêtes dirigées vers ce cluster ne seront pas perdues car le second leader détient les catalogues nécessaires à la gestion du cluster.

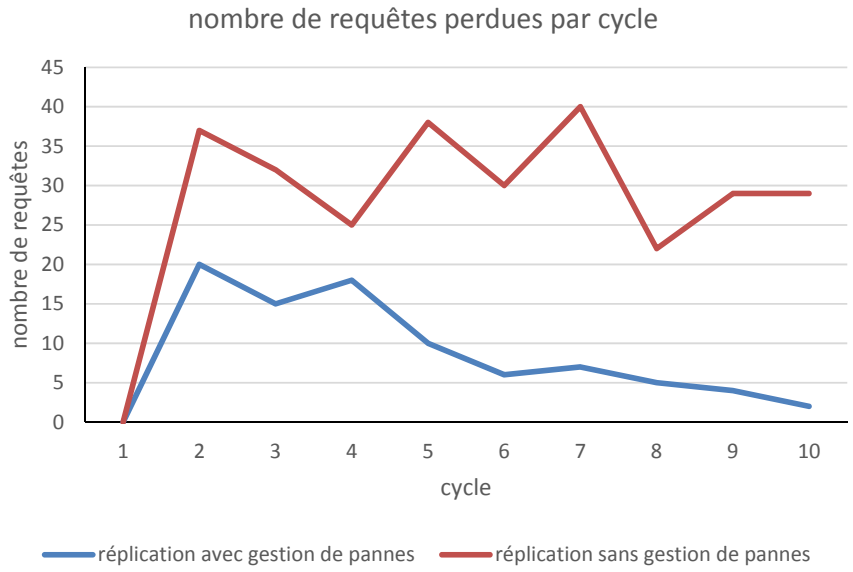


Figure 5.9 Nombre de requêtes perdues (Avec/Sans gestion de panne)

5.5 Conclusion

Nous avons présenté dans ce chapitre les résultats de simulation obtenues par l'approche implémentée. Ces résultats s'avèrent encourageant : une amélioration est observée dans les temps de réponse. De plus l'intégration des services entretien et maintenance a contribué à améliorer la disponibilité, l'usage du réseau ainsi que le temps de réponse aux requêtes. Ces performances résultent du fait que la stratégie proposée privilégie une réplication intra-cluster et diminue la communication inter-clusters à plus forte latence, ce qui influe positivement sur le temps de réponse. De plus, la réplication d'un objet n'est possible que si son coût d'accès distant est plus important que le coût de sa réplication. Ainsi le nombre de répliques est diminué évitant le transfert superflu de données qui encombrent le réseau. Par ailleurs, le service d'entretien intégré à la stratégie redistribue les répliques pour garantir leur disponibilité dans les différents clusters d'une part et pour éviter la réplication à chaque demande d'autre part.

Chapitre 6

Stratégie3 :

Réplication dynamique des données basée sur la méthode Fast spread

6. Stratégie 3 : Réplication dynamique des données basée sur la méthode Fast spread

6.1 Problématique

La topologie d'une grille affecte énormément la technique de réplication utilisée. Une grille de donnée est généralement représentée par une topologie sous forme de graphe, où chaque nœud peut être connecté à plusieurs nœuds sans restriction. Cette représentation est dépourvue de nœud central et les nœuds sont organisés indépendamment les uns des autres, ce qui permet de pallier aux inconvénients de l'architecture hiérarchiques multi-tiers (arbre) et offre une flexibilité dans la communication entre nœuds.

Dans la littérature, la plus part des chercheurs ont travaillé sur les structures hiérarchisées [5,8,9,10,13,14,18,23,25] et ont mentionné l'extension de leurs recherches aux topologies sous forme de graphe. Dans ce travail, nous considérons qu'une grille est représentée en entrée par un graphe général où il n'existe pas un nœud désigné comme racine, ceci reflète la vraie représentation d'une grille.

Comme mentionné précédemment, le placement efficace des répliques représente l'un des défis importants pour améliorer les performances d'un système, et de bonnes stratégies de placement peuvent avoir comme conséquence des gains significatifs dans le temps de réponse aux requêtes. Ce but peut être atteint, quand la disponibilité des données augmente au sein du réseau. Pour garantir cette disponibilité nous nous sommes intéressés à la stratégie de réplication Fast Spread [5] qui est une des meilleures stratégies de réplication particulièrement pour les modèles d'accès aléatoires. Dans cette stratégie une réplique du fichier demandé est stockée à chaque nœud le long de son chemin au demandeur. Si l'espace de stockage d'un de ces nœuds est insuffisant, un groupe de répliques existantes (contenant une ou plusieurs répliques) doit être remplacé par la nouvelle réplique, le problème qui se pose est que lors de la phase de suppression, des répliques plus importantes que la nouvelle réplique peuvent être supprimées. Pour remédier à ce problème nous utilisons un seuil dynamique qui détermine l'importance d'une réplique pour un nœud donné [36]. La réplication n'est alors permise que si la nouvelle réplique est plus importante. Le calcul de ce seuil dynamique se base sur quatre facteurs pour déterminer la notion d'importance, il s'agit du nombre d'accès à la réplique, la fréquence d'accès à cette réplique, la taille de la réplique ainsi que la dernière fois où la réplique a été accédée. De plus la solution proposée, est implémentée sur une architecture décentralisée à l'inverse de la méthode Fast spread qui considère la grille avec une topologie en arbre. Nous introduisons alors une nouvelle notion « la réplique bloquée « qui définit l'importance d'une réplique au sein d'une région. Une région est définit par l'ensemble des nœuds interconnectés d'un même niveau. Une réplique bloquée est à la fois importante et fréquemment accédée dans sa région. De ce fait, la stratégie proposée ne supprime jamais ce type de répliques ce qui constitue une optimisation dans le processus de réplication.

6.2 Stratégie de réplication proposée

6.2.1 Hypothèses

- La grille est représentée sous forme de graphe.
- Chaque nœud a son propre identifiant, un élément de stockage (ES) et un élément de calcul (EC).
- Un nœud ne peut pas tomber en panne.
- Les requêtes sont en lecture.

6.2.2 Architecture Fonctionnelle du modèle de réplication

La figure 6.1 montre les différents services conçus.

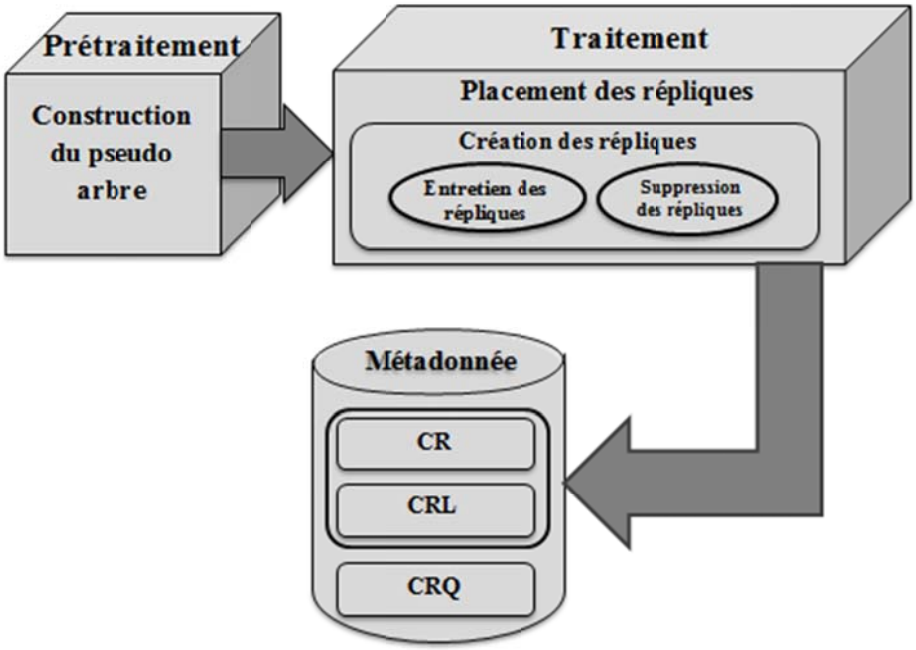


Figure 6.1. Architecture fonctionnelle du modèle de réplication

6.2.2.1 Métadonnées

Les métadonnées sont les données qui servent à décrire l'ensemble des informations contenues dans la grille.

- CR (Catalogue des répliques) : ce catalogue comporte les informations qui décrivent les répliques existantes dans la sous arborescence du nœud contenant ce catalogue tel que leurs identificateurs et où elles se trouvent.
- CRL (Catalogue des répliques local) : ce catalogue comporte les informations qui décrivent les répliques existantes dans un nœud.
- CRQ (Catalogue des requêtes) : les informations de ce catalogue représentent les requêtes provenant des autres nœuds qui requièrent une réplique. Ces informations sont la taille des requêtes et la réplique concernée.

6.2.2.2 Construction de l'arbre

Notations :

- Sommet : racine du pseudo arbre.
- Marqué : liste qui permet de marquer les nœuds parcourus.
- Voisin : liste qui contient les voisins d'un nœud i.

En entrée une grille est représentée par un graphe. L'algorithme implémenté convertit cette structure en pseudo arbre pour une meilleure gestion des nœuds. Le transfert de données entre nœuds frères devient possible, contrairement à la structure d'arbre connue, où la communication entre deux nœuds de même niveau n'est pas envisageable. Pour le choix de la racine, nous nous sommes basés sur un seul critère qui nous semble le plus important : c'est la connectivité. En effet, la racine est le nœud qui est le plus relié à un plus grand nombre d'autres noeuds, ceci limite les niveaux hiérarchiques et facilite le parcours du pseudo arbre. La figure 6.2 montre un exemple de transformation d'un graphe en pseudo arbre. Pour la construction du pseudo arbre, nous parcourons les nœuds en profondeur en commençant par la racine et en marquant les nœuds qui n'ont pas encore été visités selon l'algorithme 6.1

Algorithme 1: Construction de l'arbre

Begin
Marqué.ajouter(sommet);
For Tout noeud(i) ∈ Marqué **do**
For Tout noeud(j) ∈ $Voisin_i$(noeud(i)) **do**
If (Marqué.NeContientPas(noeud(j)) **then**
Marquer.ajouté(noeud(i));
End if;
End for;
End for;
End.

Algorithme 6.1 Construction de l'arbre

Exemple du choix de la racine : soit le graphe de la figure 6.2 (a)

$Voisin_0$:{1, 3}.$Voisin_1$:{0, 2}.$Voisin_2$:{1}.$Voisin_3$:{0, 4, 7, 8}.$Voisin_4$:{3, 5, 6}.
$Voisin_5$:{4, 6}.$Voisin_6$:{4, 5}.$Voisin_7$:{3, 9}.$Voisin_8$:{3, 7, 9}.$Voisin_9$:{7, 8}.
Le nœud qui a le plus de connectivité est le nœud 3 car il a quatre voisins donc on le considère comme racine (Figure 6.1).

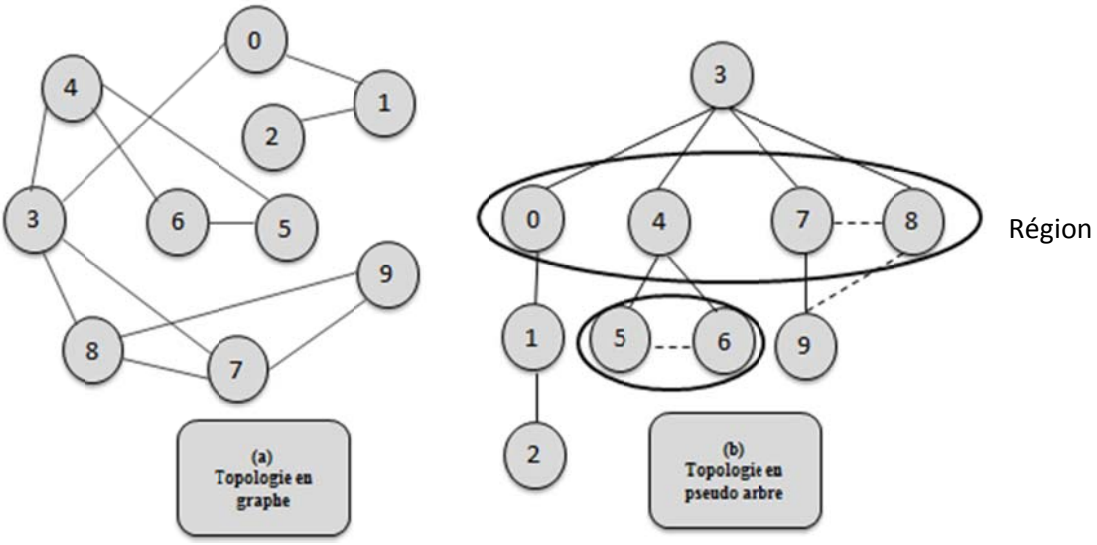


Figure 6.2. Topologie de la grille

6.2.2.2.1 Placement des répliques

- **Création des répliques**

A l'arrivée d'une requête, si le nœud recevant cette requête contient la réplique demandée, la requête est alors exécutée, sinon la réplique sera cherchée dans la sous arborescence du nœud. Si la réplique n'existe pas, on passe au père et on refait le même processus jusqu'à ce qu'on trouve la réplique demandée tout en marquant les nœuds visités. La réplique sera créée dans les nœuds tout au long du chemin vers le nœud demandeur.

Notations :

- **ES(i) :** l'espace de stockage total d'un nœud i**.**
- **EL(i) :** l'espace de stockage libre d'un nœud i**.**
- **N :** le nombre de réplique dans le groupe.
- **NAR(i) :** nombre d'accès à la réplique i dans le groupe.
- **NA(i) :** nombre d'accès à la réplique i
- **T(i) :** la taille de la réplique i.
- **ITSF :** intervalle de temps spécifique pour le calcul de la fréquence.
- **TC :** le temps courant.
- **TDA(i) :** le temps du dernier accès à la réplique i dans le groupe.
- **NARR(i) :** le nombre d'accès à la réplique i dans la région.
- **NR(i) :** nombre de nœud dans la région i.
- **LR :** liste des répliques.
- **$NARITSF_i$:** nombre d'accès à la réplique i dans le groupe dans ITSF.
- **$NAITSF_i$:** nombre d'accès à la réplique i dans ITSF.
- **RD :** réplique demandée.
- **Selectionné :** liste qui contient le groupe de répliques à supprimer ou à déplacer.
- **Région :** ensemble des nœuds qui ont le même père.

Algorithme 2: Création des répliques

Begin
Arrivée d'une requête (i) ;
Réplique$_T$ ← requête (i).réplique ;
If (Réplique$_T$ existe sur CRL du nœud demandeur) **then**
Exécuter la requête;
Else
While (Réplique$_T$ n'est pas trouvé sur CRL du nœud visité) **do**
Chemin.ajouter(nœud) ;
Chercher dans CR de ce nœud ;
Passer au père ;
End While
End If;
For tout nœudj ∈ chemin **do**
If (EL(nœudj) > taille de Réplique$_T$) **then**
noeudj.CréerRéplique (répliqueT) ;
End If;
End for;
End;

Algorithme 6.2 Création des répliques.

- **Entretien et suppression des répliques**

Lorsqu'une création de réplique est requise sur un nœud alors que l'espace libre de stockage est insuffisant, une suppression est nécessaire. Un groupe de répliques est alors sélectionné pour être supprimé. Ce groupe doit avoir une taille supérieure ou égale à la nouvelle réplique. Comme mentionné précédemment, la stratégie ***Fast Spread*** supprime ce groupe sans conditions alors qu'il peut être plus important pour le nœud que la nouvelle réplique. Pour remédier à ce problème, la stratégie proposée calcule l'importance d'une réplique (VRi : valeur de la réplique i) pour un nœud par rapport aux répliques devant être supprimées (VG : valeur du groupe) selon les formules (1) et (2).
L'importance d'une réplique R est définie par quatre facteurs qui sont :

- Le nombre d'accès à la réplique R.
- La fréquence d'accès à la réplique R.
- La taille de la réplique R.
- La dernière fois où la réplique R a été accédée.

$$VG = \frac{\sum_{i=1}^{N} NAR_i}{\sum_{i=1}^{N} T_i} + \frac{\sum_{i=1}^{N} NARITSF_i}{ITSF} + \frac{1}{TC - \frac{\sum_{i=1}^{N} TDA_i}{N}} \qquad (1) \qquad [22]$$

$$VRi = \frac{NAi}{Ti} + \frac{NRITSFi}{ITSF} + \frac{1}{TC - TDAi} \qquad (2) \qquad [22]$$

Pour un nœud donné, la réplique qui a la plus petite valeur de VR est considérée comme étant la moins importante, alors que celle détenant une valeur de VR supérieure est plus importante. Le calcul des coefficients VR et VG est basé sur quatre paramètres importants : en effet, le nombre d'accès à une réplique montre combien de fois la réplique a été utilisée dans un nœud alors que la fréquence d'accès donne la même information mais se rapportant à un intervalle de temps spécifique. En combinant ces deux paramètres avec la dernière fois où la réplique a été accédée, une indication sur la probabilité qu'une réplique soit à nouveau sollicitée est alors possible. Enfin, la taille de la réplique est aussi un paramètre important car la politique de suppression est basée sur la possibilité de stocker ou non une réplique d'une certaine taille.

De plus, dans la stratégie proposée, certain types de répliques ne peuvent pas être supprimés. Il s'agit des répliques bloquées. Nous définissons une réplique bloquée comme une donnée à la fois importante et fréquemment accédée. Pour tester si une réplique est bloquée, nous avons proposé un seuil dynamique pour déterminer si une réplique est importante au sein d'une même région, les répliques qui ont un nombre d'accès supérieur à ce seuil sont dites bloquées et ne peuvent pas être supprimées.

$$S = \frac{NARRi}{NR}$$

Notons que dans cette stratégie, la sélection du groupe à supprimer se fait d'abord parmi les répliques non bloquées. Si toutefois l'espace reste toujours insuffisant, on sélectionne des répliques bloquées. Vu, leur importance, ces répliques ne seront jamais supprimées mais transférées dans un autre nœud de la même région. Enfin, la stratégie proposée ne réplique pas de données dans la même région.

Algorithme 3: Entretien et suppression des répliques

Begin
E ← 0;
For tout replique$_i$ ∈ LR **do**
If (replique$_i$ non bloquée) **then**
If (E < RD.taille) **then**
E $<-$ E + replique$_i$ .taille;

selectionné.ajouter(replique$_i$);
End if;
End if;
End for;
If (E < RD.Size) **then**
For tout replique$_i$ ∈ LR **do**
If (replique$_i$ bloquée) **then**
If (E < RD.taille) **then**
E ← E + replique$_i$ .taille;

selectionné. ajouter (replique$_i$);
End if;
End if;
End for;
End if;
If (E > RD.Size) **then**
calculer VR et VG ;
If (VR > VG) **then**
For tout réplique$_i$ ∈ selectionné
If (réplique bloquée) **then**
chercher dans la même région le nœud qui a $EL > \sum_{i=1}^{n} replique_i.taille$
déplacer le groupe de réplique vers ce nœud
else
supprimer replique$_i$
End if;
End for;
End if;
End if;
End;

Algorithme 6.3 Entretien et Suppression des répliques.

6.3 Etude expérimentale

6.3.1 Métriques utilisées

Pour valider et évaluer le comportement de l'approche proposée, nous avons utilisé les métriques suivantes :

- Temps de réponse moyen
- Espace de stockage libre :
- La consommation des ressources réseau
- Nombre de répliques créées
- Nombre de répliques transférées : Cette métrique représente le nombre de répliques transférées d'un nœud vers un autre au lieu d'être supprimées. Elle mesure donc l'impact du service d'entretien proposé dans la stratégie

Ces métriques ont été mesurées sur la base de plusieurs expérimentations.

6.3.2 Expérimentation et évaluation

Nous présentons dans cette partie les différents résultats obtenus à partir des expérimentations que nous avons effectuées. La comparaison des résultats est réalisée principalement entre cinq stratégies :

- Sans réplication : les répliques sont distribuées de façon statique et sont accédées à distance.
- Fast Spread : cette stratégie réplique tout au long du chemin de retour au demandeur en supprimant sans conditions s'il n'y a pas d'espace de stockage suffisant.
- Fast Spread avec LRU : cette stratégie réplique tout au long du chemin de retour au demandeur en supprimant les répliques les moins fréquemment utilisées.
- Enhanced Fast Spread : cette stratégie réplique tout au long du chemin de retour au demandeur en tenant compte de l'importance des répliques lors de la suppression.
- Strategie proposée (FSGB: Fast Spread Based Graph).

6.3.2.1 Paramètres de simulation

Afin d'effectuer les expérimentations, nous avons utilisé un certain nombre de paramètres dont les valeurs sont définies dans le tableau ci-dessous :

Paramètre	Valeur
Nombre de nœuds	[10..1100]
Débit de la bande passante	[40..1000] Mb/S
Capacités de stockage des nœuds	[10..180] Gb
Vitesse des processeurs	[2.0..4.0] GHz
Nombre de données	[10..3000]
Nombre de répliques	[4..20]
Tailles des répliques	[1..4] Gb
Nombre de requêtes	[50..1200]

Tableau 6.1 Paramètres de simulation

6.3.2.2 Expérience 1 : Temps moyen de réponse

Dans cette première expérience, nous mesurons le temps de réponse moyen obtenu avec les stratégies implémentées. La figure 6.3 illustre le temps de réponse moyen obtenu pendant toute la simulation. Ces résultats montrent que : Le temps de réponse pour la stratégie sans réplication est effectivement élevé car les nœuds accèdent toujours aux répliques à distance. Par contre Fast Spread augmente la disponibilité des répliques, ce qui améliore le temps de réponse mais cette stratégie ne tient compte d'aucun critère lors de la suppression, ce qui peut amener à perdre des répliques nécessaires. La stratégie proposée est largement meilleure que les autres car elle réduit le taux de réplication du même fichier dans une même région et transfère les fichiers importants, ce qui a comme impact de réduire le temps de réponse.

Stratégie	Temps moyen de réponse (ms)
Sans Réplication	166.327
Fast Spread	151.206
Fast Spread avec LRU	116.977
Enhenced Fast Spread	93.316
Stratégie Proposée	71.987

Tableau 6.2 Temps de réponse moyen selon les stratégies

D'après le tableau ci-dessus, le gain apporté par notre proposition concernant le temps de réponse moyen aux requêtes est de 56.73% par rapport à la stratégie sans Réplication et de 22.85% par rapport à la méthode Enhenced Fast Spread, ce qui représente un résultat significatif.

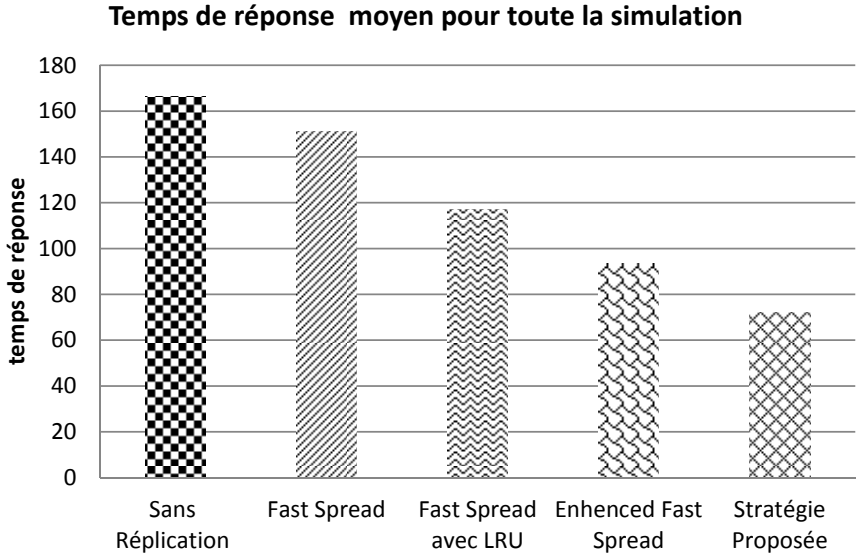


Figure 6.3. Temps de réponse de toute la simulation

6.3.2.3 Expérience 2 : Nombre de messages échangés

Dans cette expérience, nous étudions l'apport de la stratégie implémentée concernant le nombre de messages échangés entre sites afin d'exécuter les requêtes. La courbe de la figure 6.4 montre que la réplication permet de réduire le nombre de messages. Bien que la stratégie proposée donne de meilleurs résultats par rapport aux autres stratégies (sans réplication, Fast Spread, Fast Spread avec LRU), nous constatons que le schéma s'inverse par rapport à Enhanced Fast Spread et ceci s'explique très simplement par le fait que dans notre stratégie, on accède à distance dans la même région ce qui augmente le nombre de messages mais permet également :

- d'éviter de trop répliquer et ceci a l'avantage de diminuer le temps de réponse d'exécution des requêtes comme nous l'avons constaté dans l'expérience précédente.
- De réduire le nombre de requêtes créées. (cf section 6.3.3.4)

Cette expérience confirme que les objectifs qu'on voudrait atteindre en utilisant la réplication sont conflictuels.

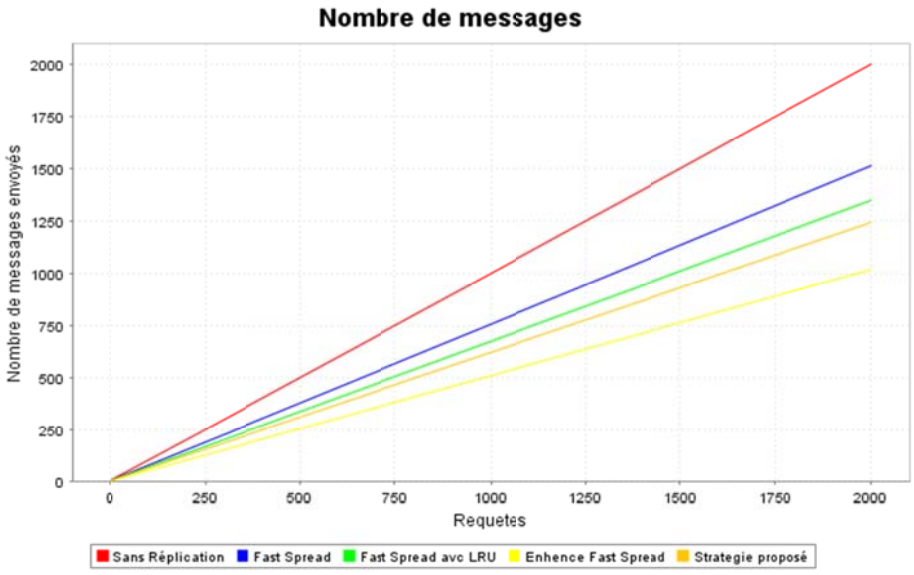


Figure 6.4. Nombre de messages échangés

6.3.2.4 Expérience 3 : Nombre de répliques créées

Dans cette expérimentation, nous mesurons le nombre de répliques créées. La figure 6.5 montre que la stratégie proposée crée moins de répliques par rapport aux autres stratégies car elle prend en considération l'importance des répliques et ne réplique pas dans la même région.

Le tableau ci-dessous donne le nombre de répliques créés pour chaque stratégie, Notons que notre stratégie réduit ce nombre de 37,58%, 33,9%, 32.88% par rapport aux stratégies : Fast Spread, Fast Spread avec LRU et Enhenced Fast Spread

Stratégie	Répliques créées
Fast Spread	556
Fast Spread avec LRU	525
Enhenced Fast Spread	517
Stratégie Proposée	347

Tableau 6.3 Nombre de répliques créées selon les stratégies

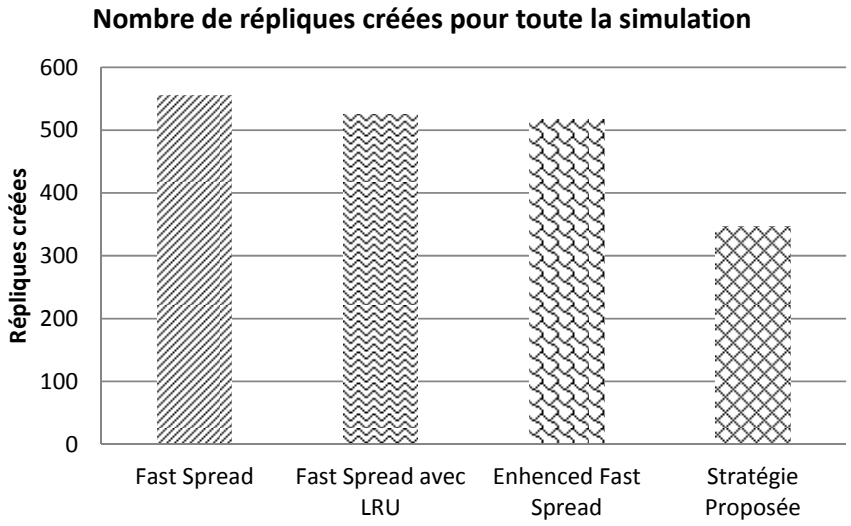


Figure 6.5. Nombre de répliques créées

6.3.2.5 Expérience 4 : Nombre de répliques supprimées

Dans cette expérience, nous mesurons le nombre de répliques supprimées. La figure 6.6 montre que le taux de suppression de la stratégie proposée (FSBG) est inferieur par rapport aux autres stratégies et ceci car c'est la seule stratégie qui déplace les répliques en intégrant la notion de « répliques bloquées » au lieu de les supprimer. Le nombre de répliques supprimées est diminué de moitié par rapport à la stratégie Fast Spread, et de 39.66% par rapport à la stratégie Enhenced Fast Spread (voir tableau 6.4)

Stratégie	Répliques supprimées
Fast Spread	522
Fast Spread avec LRU	425
Enhenced Fast Spread	421
Stratégie Proposée	254

Tableau 6.4 Nombre de répliques supprimées selon les stratégies

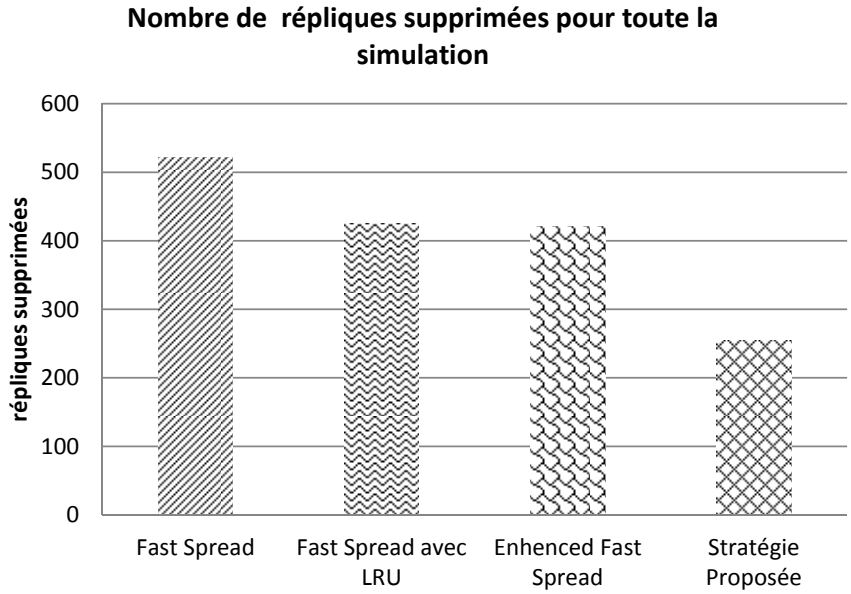


Figure 6.6. Nombre de répliques supprimées

6.3.2.6 Expérience 5 : Nombre de répliques déplacées

Stratégie	Répliques déplacées
Fast Spread	0
Fast Spread avec LRU	0
Enhenced Fast Spread	0
Stratégie Proposée	22

Tableau 6.5 Nombre de répliques déplacées selon les stratégies

Dans cette expérience nous mesurons le nombre de répliques déplacées dans chaque stratégie. Nous remarquons que seule la stratégie proposée (FSBG) transfert les répliques faute d'espace de stockage. Il s'agit-là d'une gestion des répliques déjà existantes dans la grille pour conserver les répliques les plus importantes d'une part et pour économiser le temps de création des répliques d'autre part. Ce temps influe directement sur le temps de réponse des requêtes.

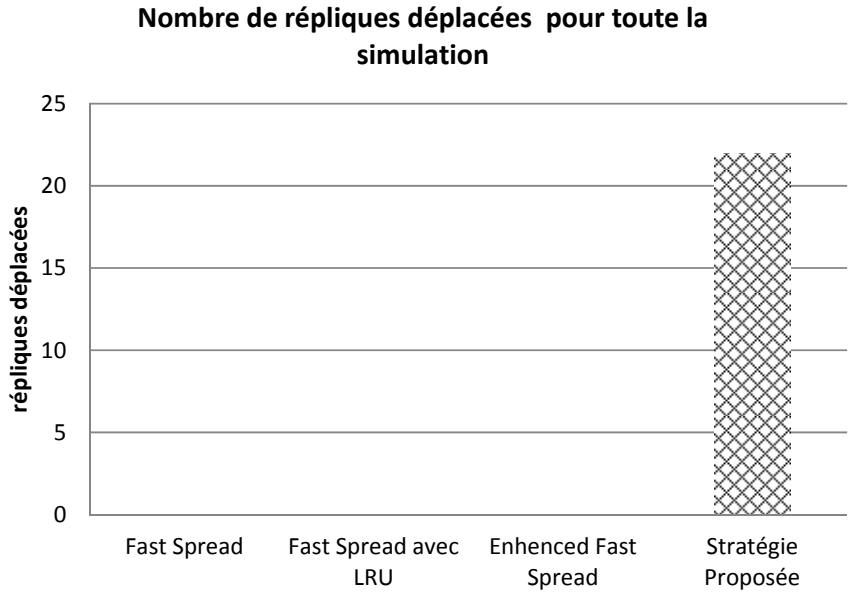


Figure 6.7. Nombre de répliques déplacées

6.3.2.7 Expérience 6 : Espaces de stockage

Stratégie	Espace libre	Espace utilisé
Fast Spread	25.656 %	74.344 %
Fast Spread avec LRU	25.31 %	74.69 %
Enhenced Fast Spread	25.499 %	74.501 %
Stratégie Proposée	29.145 %	70.885 %

Tableau 6.6 Espace de stockage dans la grille selon les stratégies

Le but de cette expérience, est de voir la variation de l'espace de stockage libre de la grille dans chaque stratégie. Ce résultat est directement lié au nombre de répliques créées ainsi que leurs tailles.

Nous pouvons voir dans la figure 6.8 que l'espace de stockage libre de la grille est le plus important dans la stratégie (FSBG) car cette stratégie ne réplique pas les données qui existent dans la même région. Bien que la stratégie proposée transfert quelques répliques au lieu de les supprimer, l'espace de stockage libre reste plus élevé que pour les autres stratégies.

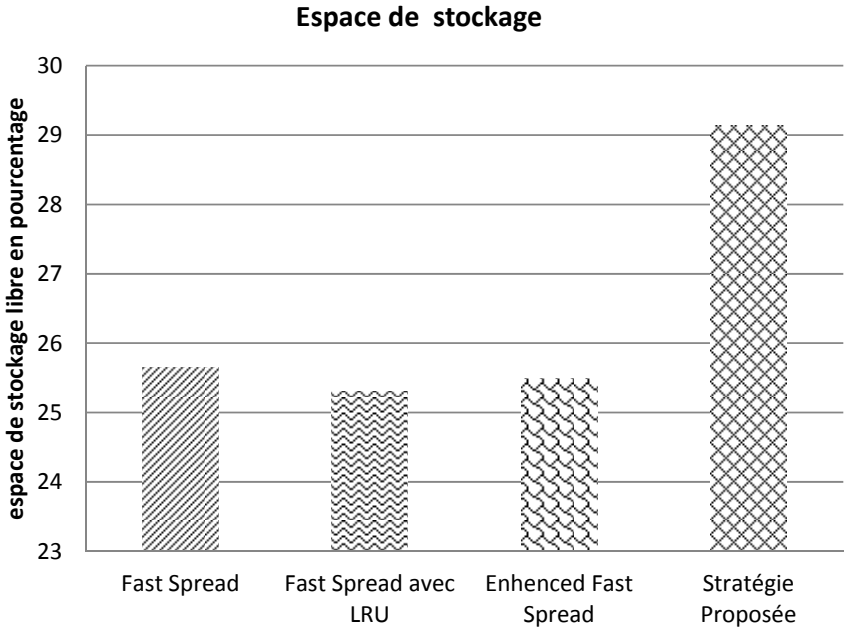


Figure 6.8. Espaces de stockage

6.3.2.8 Expérience 7: Temps de réponse par requête

Cette expérience étudie l'impact de la variation du nombre de requêtes sur le temps de réponse moyen. Nous avons fait varier le nombre de requêtes de 200 à 1000 par pas de 50. Nous remarquons d'après la figure 6.9 que la courbe de l'approche proposée a une tendance linéaire tout comme les autres courbes. De plus la stratégie proposée, réduit le temps de réponse par rapport à toutes les autres stratégies. Le graphique de la figure 6.9 montre un résultat intéressant : la stratégie Enhence Fast Spread donne de meilleurs résultats au début de la simulation mais la tendance s'inverse au profit de la stratégie proposée quand le nombre de requêtes augmente (500 requêtes), ceci est dû aux contributions apportées à la méthode.

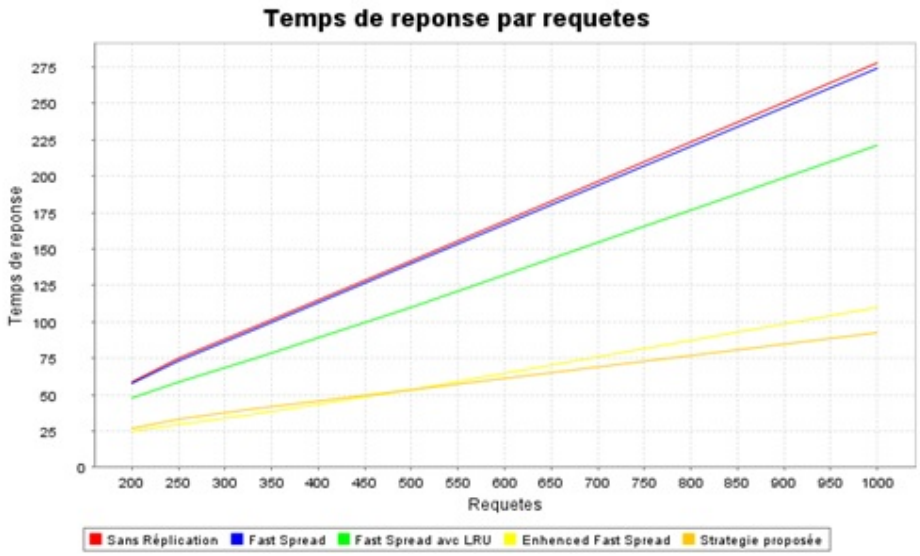


Figure 6.9. Temps de réponse moyen selon le nombre de requêtes

6.4 Conclusion

Dans ce chapitre nous avons exposé la troisième stratégie proposée. Cette stratégie est basée sur la méthode *Fast Spread* et a été intégrée à une grille de topologie non hiérarchique. L'analyse des résultats expérimentaux avec le simulateur Optorsim a révélé une amélioration dans les temps d'exécution des requêtes grâce au rapprochement des données nécessaires à leur exécution. Aussi le nombre de requêtes créées s'avère le plus bas comparé aux autres stratégies. Ceci s'explique très simplement par le fait que la stratégie proposée ne réplique pas dans la même région et déplace les répliques les plus importantes lorsqu'un problème d'espace de stockage est rencontré. Cet entretien de répliques vise à garantir la disponibilité et influe positivement sur le temps de réponse aux requêtes.

Chapitre 7

Stratégie4 :

Réplication dynamique des données basée sur l'algorithme BHR

7. Stratégie 4 : Réplication dynamique des données basée sur l'algorithme BHR

7.1 Introduction

Notre contribution consiste à améliorer la stratégie de recherche présentée dans l'algorithme BHR (Bandwidth Hierarchy Replication). Cet algorithme a été d'abord présenté en 2004 [8] et n'a cessé d'être amélioré en prenant en compte à chaque fois la spécificité des grilles et leurs évolutions.[26,30]

Le principe de base de l'algorithme est d'utiliser la notion de localité pour réduire le temps d'accès aux données et éviter la congestion du réseau. Au départ cette notion de localité a été définie comme suit : la réplique demandée doit être hébergée dans le nœud qui a une large bande passante avec le site (nœud) où la requête doit être exécutée. De là, l'idée de regrouper un certain nombre de nœud en une région est née. Une région peut être un réseau local par exemple (LAN). Le profit tiré de cette organisation est que la bande passante entre les sites d'une même région est supérieure à celle entre nœuds de deux régions distinctes, d'où si une réplique demandée existe dans la même région, le temps de réponse serait beaucoup plus petit. La région définit donc la notion de cluster que nous connaissons.

Il a été prouvé que l'algorithme BHR réduit le temps de réponse en maximisant la localité dans la grille, mais l'inconvénient majeur de cet algorithme est qu'il est implémenté sur une architecture centralisée. En effet, toutes les régions sont rattachées à un nœud central appelé « Master Site » qui détient l'ensemble des fichiers produit par la grille. Ce site a la tâche de distribuer les répliques sur toutes les régions. De ce fait, les répliques d'un fichier existent nécessairement dans toutes les régions, ce qui est une hypothèse très lourde et ne reflète pas la réalité car le site master doit avoir une capacité de stockage théoriquement infinie. De plus il représente un maillon faible, si le nœud master tombe en panne toute la grille est paralysée.

La stratégie que nous proposons tente de remédier aux inconvénients de l'algorithme BHR citées ci-dessus. D'abord, l'architecture a été modifiée en une topologie complètement distribuée où il n'y a pas de site central entre les régions. De plus, nous levons l'hypothèse qui consiste à avoir nécessairement tous les fichiers dans chaque région, ce qui a comme effet, l'introduction d'un nouveau module « Réplication inter Région » : nous prenons en compte la recherche et la réplication entre les régions en utilisant la méthode « Fast Spread »[5]. Enfin, quand une réplication est requise et il n'existe pas l'espace suffisant pour la nouvelle réplique, l'algorithme BHR supprime les fichiers les moins fréquemment utilisés. Dans notre cas, nous calculons une valeur de prédiction pour chaque fichier pour déterminer ceux qui seront supprimés.

7.2 Topologie de la grille

Le modèle logique de la grille est constitué par un ensemble de nœuds (sites Si) (figure 7.1) qui sont organisés en régions (clusters), chaque région est gérée par un site particulier

appelé header (Hi). Les headers sont interconnectés de façon distribuée formant ainsi un graphe général. De ce fait, la représentation obtenue est dépourvue de nœud central. De plus, si un header tombe en panne, seule la région qui possède le nœud est paralysée et non pas toute la grille. Le modèle est illustré dans la figure suivante :

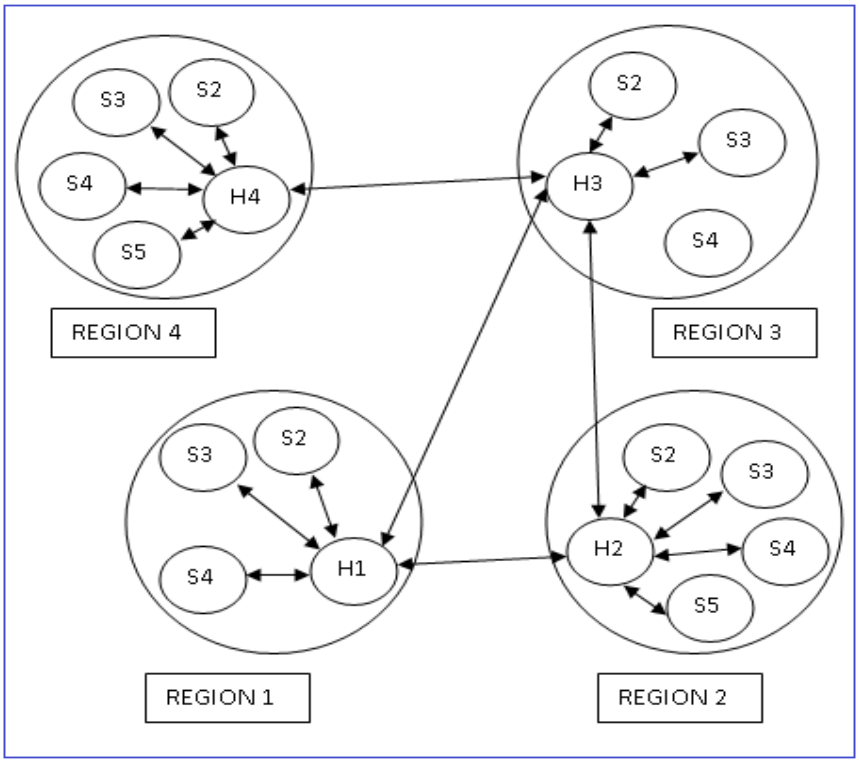


Figure 7.1. Topologie logique de la grille.

7.3 Modèle de réplication

7.3.1 Hypothèses

- Les communications inter-régions se font via les headers.
- Chaque site contient des éléments de calcul (CE) et des éléments de stockage (SE).
- Chaque site header contient toutes les informations sur chaque site dans sa région : capacité de stockage, liste des répliques qu'il contient, historique des demandes de fichiers, …etc.
- Chaque site header maintient une table de routage qui contient les plus courts chemins vers tous les autre headers (Algorithme de Dijkstra [82]).
- Les demandes de fichiers sont propagées vers le header qui prend en charge la recherche de la réplique.
- Le processus de réplication est déclenché à chaque demande de fichier.
- Les répliques sont en lecture seulement.

7.3.2 Architecture fonctionnelle du modèle de réplication

Le modèle de la stratégie est composé de cinq modules :

- Le module de recherche des fichiers (Rech) qui est responsable de la recherche des fichiers nécessaires aux requêtes d'abord dans la région concernée (intra région) puis dans les autres régions (inter régions).
- Le module de la réplication intra-région (RepIntra) responsable de la réplication des fichiers dans une même région.
- Le module de la réplication inter-régions (RepInter) responsable de la réplication des fichiers se trouvant dans d'autres régions.
- Le module de suppression des fichiers (Sup) responsable de la suppression des fichiers si une réplication est requise alors que l'espace de stockage est insuffisant.
- Le module de calcul de la valeur de prédiction (Pred) qui calcule une estimation du taux d'accès d'un fichier donné.

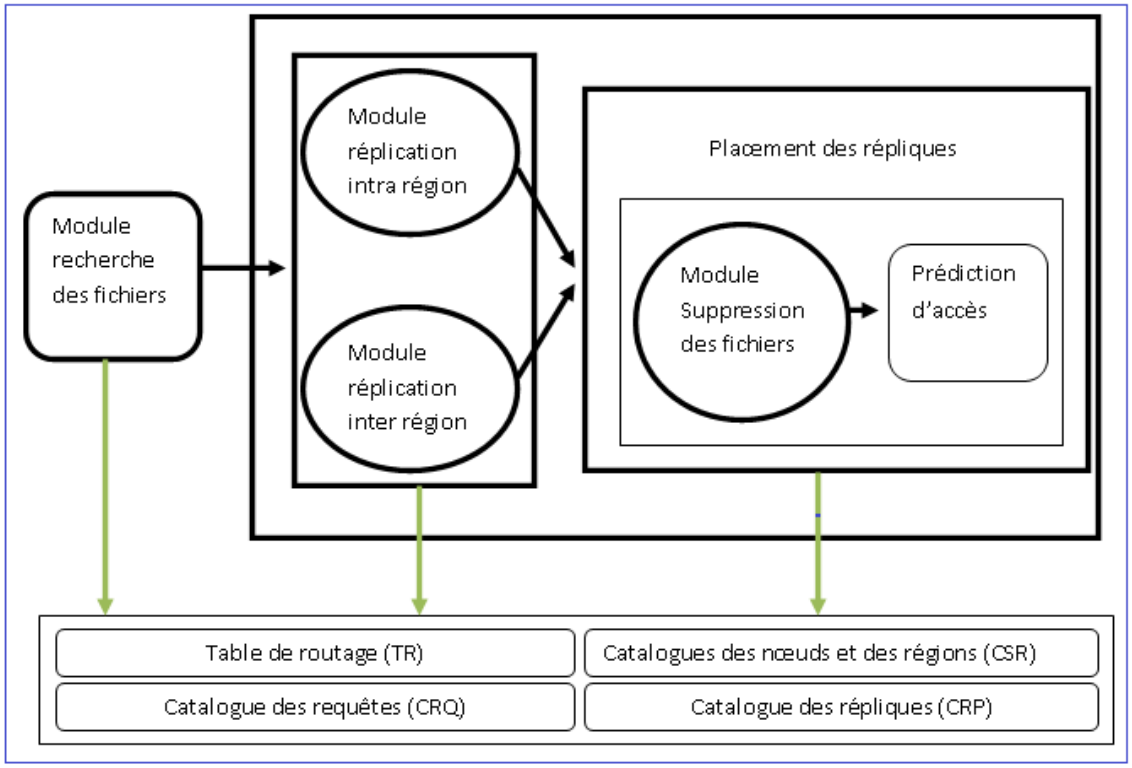


Figure 7.2. Architecture fonctionnelle du modèle de réplication

A l'arrivée d'une requête, le gestionnaire de la région déclenche une recherche pour vérifier si le fichier demandé existe dans sa région. Dans le cas négatif, il lance une demande de réplication vers les autres régions.

7.3.2.1 Recherche intra région

Ce composant est responsable de la recherche des fichiers dans la grille. Dans notre stratégie nous privilégions une recherche intra région dans le but de minimiser les messages

entre sites et de rendre la transmission plus rapide. La recherche du fichier demandé est activée d'abord dans la région où se trouve le site qui le demande, s'il n'existe pas de répliques de ce fichier dans toute la région, on initie alors une recherche inter-régions. La fonction **RechercheFichier()** suivante illustre ce principe:

Algorithme 1: Procédure Recherche Fichier ()

```
Begin
For toute réplique r dans CRP do
If ( r  est le fichier recherché) then
return r ;
End if;
End for;
For tout header h de CSR do
Envoyer_demande_fihier(h);
recevoir _reponse(h);
End for;
// Choisir le fichier qui se trouve dans le nœud et la région
//    la plus proche en consultant TR ;
End.
```

Algorithme 7.1 : Fonction de recherche de fichiers.

7.3.2.2 Recherche inter région

Dans le cas où le fichier recherché n'existe pas dans la région, le header envoie une demande au header de chaque région voisine. Si cette dernière dispose d'une réplique, elle l'envoie, sinon elle envoie une demande aux régions voisines et ainsi de suite. Une fois le fichier trouvé, le module « Rech » déclenche le module « RepIntra » dans le cas où le fichier est trouvé dans la même région, il déclenche le module « RepInter » sinon.

7.3.2.3 Réplication intra région (RepIntra)

L'algorithme implémenté dans ce module réplique le fichier demandé dans la même région si et seulement si le coût de la réplication du fichier est inférieur au coût de son accès à distance. Pour cela nous utilisons le modèle de coûts proposé dans la stratégie 2 définie au chapitre 5 précédent.

Coût d'accès à distance à un fichier : **Coût_accès_distant**

Cout de la réplication d'un fichier : **Coût_Réplication**

L'algorithme de décision de la réplication est donné ci-dessous :

Algorithme 2: Procédure Décision Réplication ()

```
Begin
Calcul (coût_acces_distant)
Calcul (coût_Replication)
If (coût_acces_distant < coût_Replication ) then
Executer_requête_à_distance() ;
Else
Replication ();
End if;
End.
```

Algorithme 7.2 : Procédure Décision Réplication ().

7.3.2.4 Réplication inter régions (RepInter)

A l'arrivé d'une requête, si le fichier demandé n'existe pas dans la région du site demandeur et que le module « Rech » a retrouvé le fichier dans une région voisine, le module « RepInter » sera déclenché. Il répliquera le fichier dans la région du site de demandeur et dans toutes les régions rencontrées tout au long du chemin de retour.

Comment répliquer ?

Pour toute région R appartenant au chemin de retour défini depuis la région disposant du fichier demandé jusqu'à la région demandant le fichier, la réplication aura lieu. Le fichier sera répliqué dans le site de la région R ayant le maximum d'espace libre de stockage. Si toutefois cet espace est insuffisant, le module « sup » sera déclenché afin de supprimer quelque répliques jusqu'à avoir l'espace requis, sinon un autre site sera choisi.

Remarque : pour la région qui demande le fichier en question, la réplication se fait en priorité dans le nœud demandeur.

Algorithme 3: Procédure Replication_Inter-Region ()

Begin
For toute région R dans le chemin de retour **do**
Liste = lister les nœuds de R selon la capacité de stockage disponible (CSR)
If (région demandeur) **then**
Ajouter le nœud demandeur au sommet de liste ;
End if;
For chaque nœud de la liste **do**
If (espace suffisant pour la réplique) **then**
Répliquer() ;
Else
While (suppression possible) **AND** (pas de réplication) **Do**
Supprimer();
End While
End if;
If replication_effectuée() **then**
Break();
End if;
End for;
End for;
End.

Algorithme 7.3 : Réplication_Inter_région().

7.3.2.5 Suppression des fichiers

Ce module gère la suppression des fichiers dans la grille. Une suppression est nécessaire quand une réplication est requise alors qu'il n'y a pas suffisamment d'espace de stockage sur le nœud. Notons que dans cette stratégie proposée, un fichier unique (nombre de son occurrence dans la région est égal un) ne sera jamais supprimé et ceci dans un souci de disponibilité de données.

Quels fichiers seront supprimés ?

Dans la stratégie proposée, les répliques qui seront supprimées sont des répliques non uniques qui ne seront probablement pas ou peu accédées dans le futur. Pour déterminer ces répliques, nous calculons pour chaque fichier une valeur appelée « valeur de prédiction » [70], cette valeur est une estimation du nombre d'accès à un fichier à un instant futur en se basant sur l'historique d'accès à ce fichier.

Le module « Sup » liste les fichiers non uniques selon la valeur de prédiction croissante et supprime les fichiers ayant les plus petites valeurs jusqu'à déterminer l'espace de stockage requis.

Algorithme 4: Procédure Suppression_Fichier ()

Begin
For chaque fichier non unique f du nœud (CSR) **do**
calculer la valeur de prédiction pour le fichier f ;
End for
Liste = lister les fichiers non unique selon la valeur de prédiction croissement
While (espace insuffisant) **AND** (**liste** non vide) **Do**
supprimer le sommet de **liste;**
End While
End.

Algorithme 7.4 : ProcedureSuppression_fichier().

Calcul de la valeur de prédiction

Le calcul de la valeur de prédiction pour un fichier se base sur l'historique des accès à ce fichier sur une période donnée. Le principe consiste à diviser le temps en plusieurs périodes et de conserver les accès aux fichiers.

Soit l'exemple suivant :

- **n(t,f)** : le nombre d'accès d'un nœud au fichier f à l'instant t.
- **Est(t,f) :** l'estimation du nombre d'accès au fichier f à l'instant t.
- **T** nombre de période.

Alors

Est(t+1, f) = n(t,f) * $e^{a(t)}$ avec **: a(t) =** $[\sum_{i=0}^{T-1} a\ i]/T$ **et ai =** $ln(\frac{[n(i+1,f)]}{n(i,f)})$**.** **[70]**

Exemple

Soit la séquence d'accès du fichier f par le site i pour quatre intervalles de temps :

- n (0, f) = 23
- n (1, f) = 20
- n (2, f) = 12
- n (3, f) = 10

Nous voulons estimer le nombre d'accès dans le prochaine intervalle : **Est (4, f) = ?**

Est (4, f) = n (3, f) * $e^{a(3)}$

Calcul de a(3) :

a(3) = [ln[n(1,f)/n(0,f) **] + ln[** n(2,f)/n(1,f) **] + ln[** n(3,f)/n(2,f) **] / T**

= [ln (20/23**) + ln (**12/20**) + ln (**10/12**)] / 3 = -0.27**

D'où :

Est (4, f) = 10 * $exp^{-0.27}$ = 7.6

L'estimation du nombre d'accès au fichier f pour le site i à l'instant 4 = 8 accès.

7.4 Expérimentation et évaluation

Afin de tester la stratégie proposée, nous avons effectué une série d'expérimentations dont les résultats et les interprétations font l'objet de cette section.

7.4.1 Paramètres d'évaluation

Pour évaluer le comportement de la stratégie proposée, nous avons utilisé quatre métriques :

- Le temps de réponse moyen.
- Le nombre de répliques créées.
- La consommation des ressources réseaux.
- Le nombre de messages échangé intra et inter régions.

Dans les différentes simulations nous évaluons les stratégies selon deux modes :

- Le mode centralisé : dans ce mode, la grille possède un seul ordonnanceur de tâches. la soumission des taches est séquentielle par rapport aux régions de la grille.

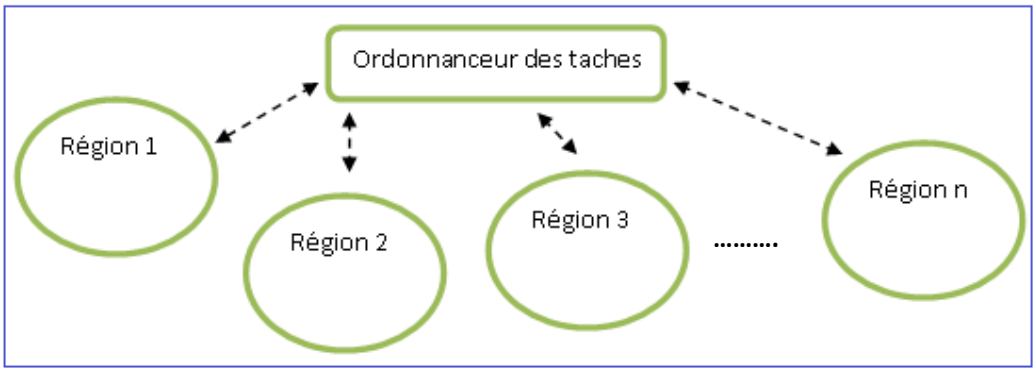


Figure 7.3. Mode centralisé.

- Le mode décentralisé : dans ce mode, chaque région de la grille possède son propre ordonnanceur de tâches.

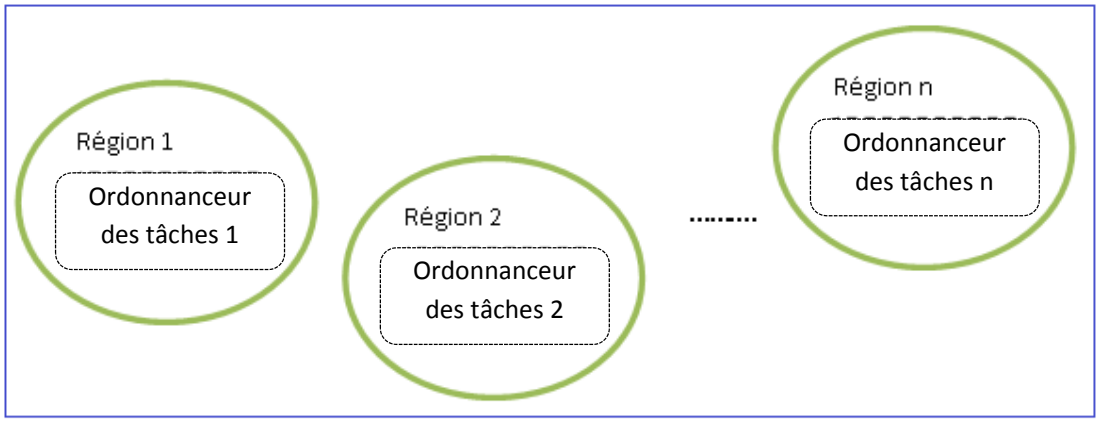


Figure 7.4. Mode décentralisé.

7.4.2 Scénario de configuration de la grille

Les répliques et les requêtes ont été soumises à la grille selon quatre scénarios différents, ceci a pour effet d'étudier l'impact de la distribution sur les performances de la grille.

- **Scénario1** (distribution aléatoire) : dans ce mode, la distribution des nœuds, des répliques ainsi que la distribution des requêtes sur les régions se fait d'une manière aléatoire.
- **Scénario2** : (distribution équitable) : dans ce mode, la distribution est équitable pour toutes les régions.
- **Scénario3** : (distribution selon le nombre de nœuds par région) : la distribution des nœuds dans les régions se fait d'une manière aléatoire alors que celle des répliques et des requêtes sera proportionnelle au nombre de nœuds par région.
- **Scénario4** : (distribution selon les capacités des régions) : les nœuds sont distribués aléatoirement dans les régions alors que les répliques et les requêtes seront distribuées selon la capacité des régions. Cette capacité est mesurée en termes de puissance de calcul et capacité de stockage.

7.4.3 Résultats et discussion

Nous présentons dans cette partie les différents résultats obtenus à partir des expérimentations que nous avons effectuées. La comparaison des résultats se fait principalement entre sept stratégies :

1) BHR + NO réplication (en centralisé).
2) BHR + NO réplication (en décentralisé).
3) BHR + LFU (suppression des fichiers les moins fréquemment utilisés) (en centralisé).
4) BHR + LFU (suppression des fichiers les moins fréquemment utilisés) (en décentralisé).

5) BHR + LRU (suppression des fichiers les moins récemment utilisés) (en centralisé).
6) BHR + LRU (suppression des fichiers les moins récemment utilisés) (en décentralisé).
7) Stratégie proposée (suppression de fichiers selon l'estimation calculée) (en décentralisé)

7.4.3.1 Paramètres de simulation

Afin d'effectuer les expérimentations, nous avons utilisé un certain nombre de paramètres dont les valeurs sont définies dans le tableau ci-dessous :

Paramètre	**Valeur**
Nombre de clusters	[3..29]
Nombre de nœuds	[60..2500]
Débit de bande passante intra-cluster	[1..99] Mb/S
Débit de bande passante inter-clusters	[100..1000] Mb/S
Capacités de stockage des nœuds	[130..2000] Gb
Vitesse des processeurs	[2.0..4.0] GHz
Nombre de données	[10..1000]
Nombre de répliques	[100..200]
Tailles des répliques	[1..4] Gb
Nombre de requêtes	[100..4000]
Taille des requêtes	[200..1000] Ko

Tableau 7.1 Paramètres de simulation

7.4.3.2 Expérimentation 1 : Temps de réponse moyen selon les stratégies.

La figure suivante illustre le temps de réponse obtenu pour les sept stratégies citées ci-dessus :

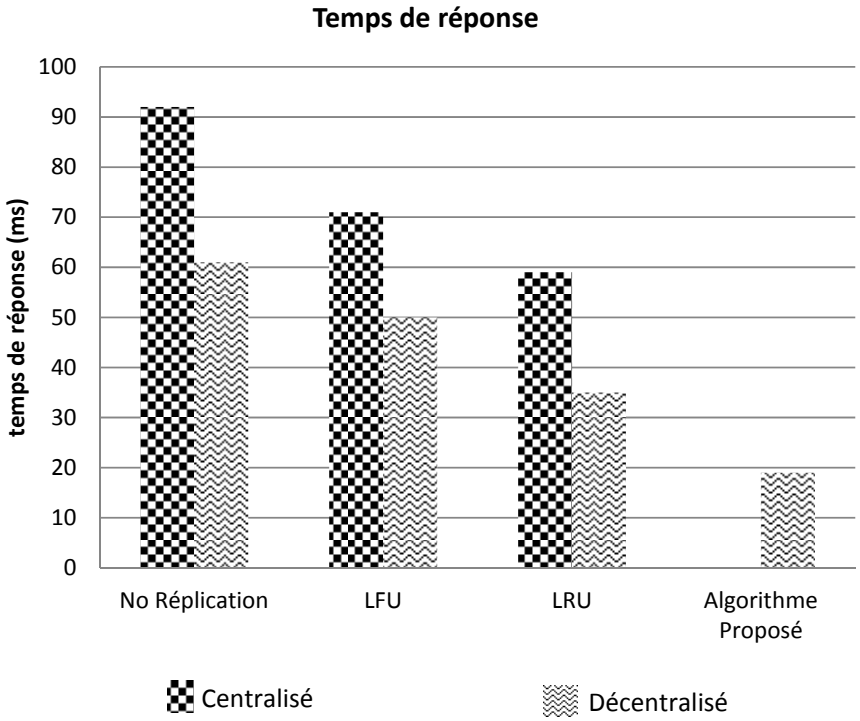


Figure 7.5. Temps de réponse des différents stratégies en centralisé et en décentralisé.

Stratégies	Temps de réponse en centralisé (ms)	Temps de réponse en décentralisé (ms)
No Réplication	92	61
LFU	71	50
LRU	59	35
Algorithme Proposé	/	19

Tableau 7.2 Temps de réponse en centralisé et en décentralisé

Nous remarquons que la stratégie proposée donne de meilleurs temps de réponse par rapport aux autres stratégies. Ceci s'explique par le fait qu'elle utilise une fonction de prédiction dans la suppression des fichiers et donc, les fichiers qui seront probablement utilisés dans le futur proche ne seront pas supprimés. Nous évitons ainsi leurs réplications en cas de besoin ce qui influe positivement sur le temps de réponse. D'un autre côté, notre stratégie réplique le fichier dans chaque région de retour, ce qui augmente la disponibilité des données au fur et à mesure de l'arrivée des requêtes.

La figure 7.5 montre que les temps de réponse obtenus dans le modèle décentralisé sont meilleurs que ceux obtenus dans le modèle centralisé en général grâce à la soumission simultanée des requêtes.

D'après les données du tableau 7.2, on constate que le gain apporté par la stratégie proposée est de 68.85%, 62% et 45.71% (resp) par rapport aux stratégies No Réplication, LFU, LRU.

7.4.3.3 Expérimentation 2 : Temps de réponse moyen selon les scénarios.

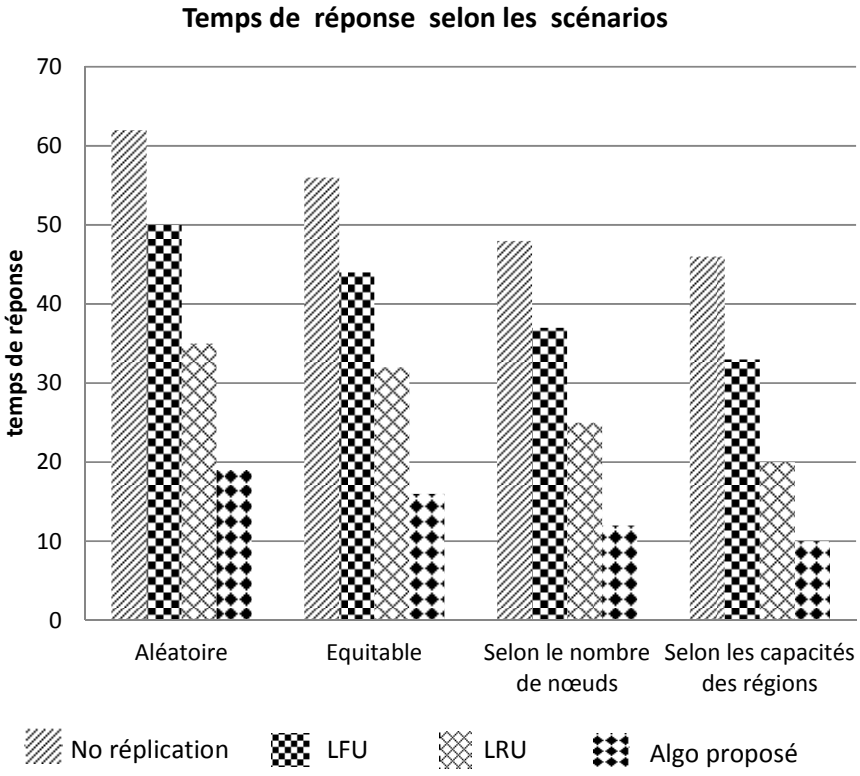


Figure 7.6. Temps de réponse selon les scénarios.

Stratégies\scénarios	Aléatoire	Equitable	Selon le nombre de nœuds	Selon les capacités des régions
No réplication	62	56	48	46
LFU	50	44	37	33
LRU	35	32	25	20
Algorithme proposé	19	16	12	10

Tableau 7.3 Temps de réponse selon les scénarios

Dans cette expérimentation, nous étudions l'impact de la distribution des requêtes sur le temps de réponse moyen. Nous remarquons d'abord que selon tous les scenarios, le temps de réponse est réduit dans l'approche proposée par rapport aux autres stratégies.

La figure 7.6 montre que le scenario 4 donne les meilleurs résultats en termes de temps de réponse aux requêtes. Rappelons que ce scénario distribue les répliques et les requêtes selon les capacités de stockage et d'exécution des nœuds. Donc c'est comme si un équilibrage de charge est fait, ce qui influe positivement sur le temps de réponse.

7.4.3.4 Expérimentation 3 : Nombre de répliques créées selon les scénarios.

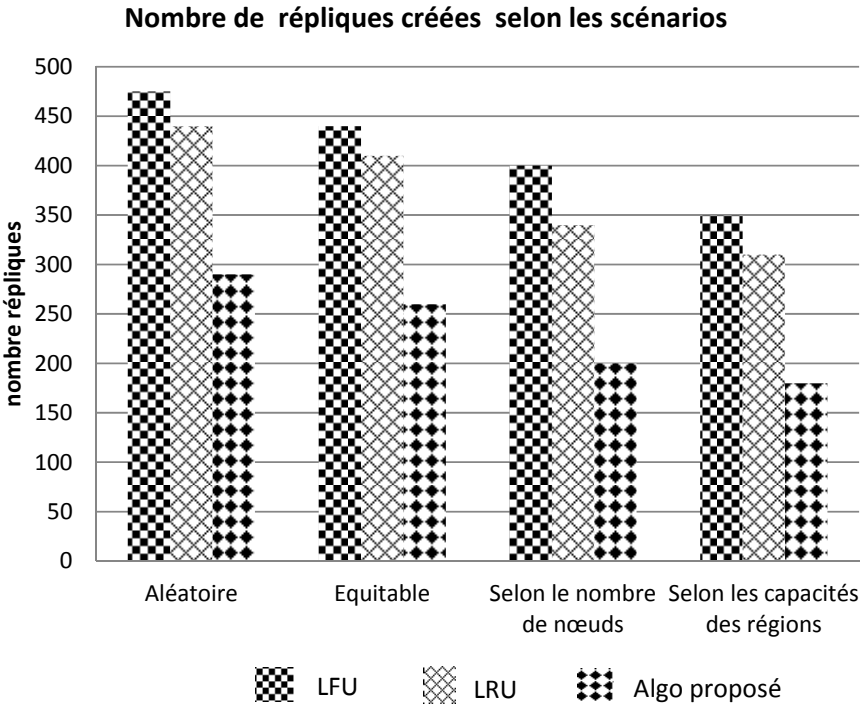


Figure 7.7. Nombre de répliques crées selon les scénarios de configuration.

Stratégies\scénarios	Aléatoire	Equitable	Selon le nombre de nœuds	Selon les capacités des régions
LFU	475	440	400	350
LRU	440	410	340	310
Algorithme proposé	290	260	200	180

Tableau 7.4 Nombre de répliques crées selon les scénarios

Dans cette expérience, nous ne comparons que trois stratégies car la stratégie No-Réplication ne réplique pas de données de façon dynamique. Il est à noter que pour une stratégie efficace, il ne s'agit pas de répliquer à l'infini, et qu'une bonne stratégie de réplication doit minimiser le nombre de répliques créées. Notre stratégie donne un nombre de répliques créées réduit par rapport à la stratégie LRU et LFU car elle supprime les fichiers en utilisant une valeur de prédiction, donc si un fichier pourra être probablement accédé prochainement, il ne sera pas supprimé et donc nous évitons la réplication de ce fichier lors des prochaines requêtes. De plus, si un nœud demande un fichier qui n'existe pas sur son ES, le module de décision de réplication pourra décider d'un accès distant et non pas toujours d'une réplication : la réplication n'est pas automatique.

Le tableau 7.4 donne le nombre de répliques créées pour chaque stratégie, Notons que la stratégie proposée réduit ce nombre de 50% par rapport à LFU et de 58% par rapport à LRU.

7.4.3.5 Expérimentation 4 : Consommation des ressources réseaux selon les scénarios.

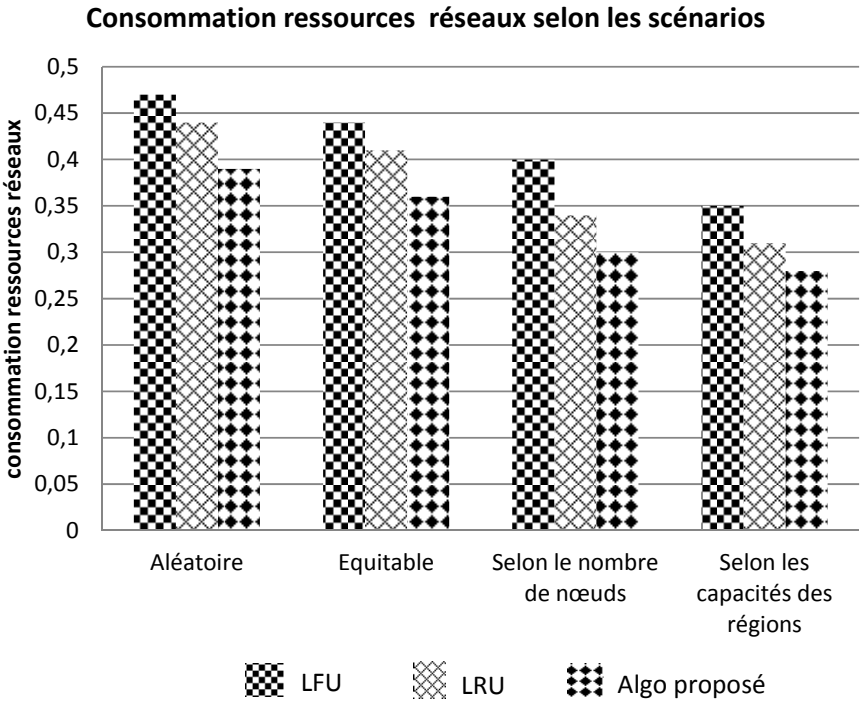


Figure 7.8. Consommation du réseau selon les modes de configurations.

Stratégies\scénarios	Aléatoire	Equitable	Selon le nombre de nœuds	Selon la capacité des régions
LFU	0.47	0.44	0.40	0.35
LRU	0.44	0.41	0.34	0.31
Algorithme Proposé	0.39	0.36	0.3	0.28

Tableau 7.5 Consommation des ressources réseaux selon les stratégies

Cette expérience montre que les ressources réseaux sont mieux consommées dans le dernier scénario où les requêtes et les répliques sont distribuées selon la capacité des régions.

7.4.3.6 Expérimentation 5 : Nombre de message intra/inter régions selon les scénarios.

Nombre de message intra région

Le nombre de messages échangés entre les nœuds de chaque région selon les différents scénarios est illustré dans la figure suivante :

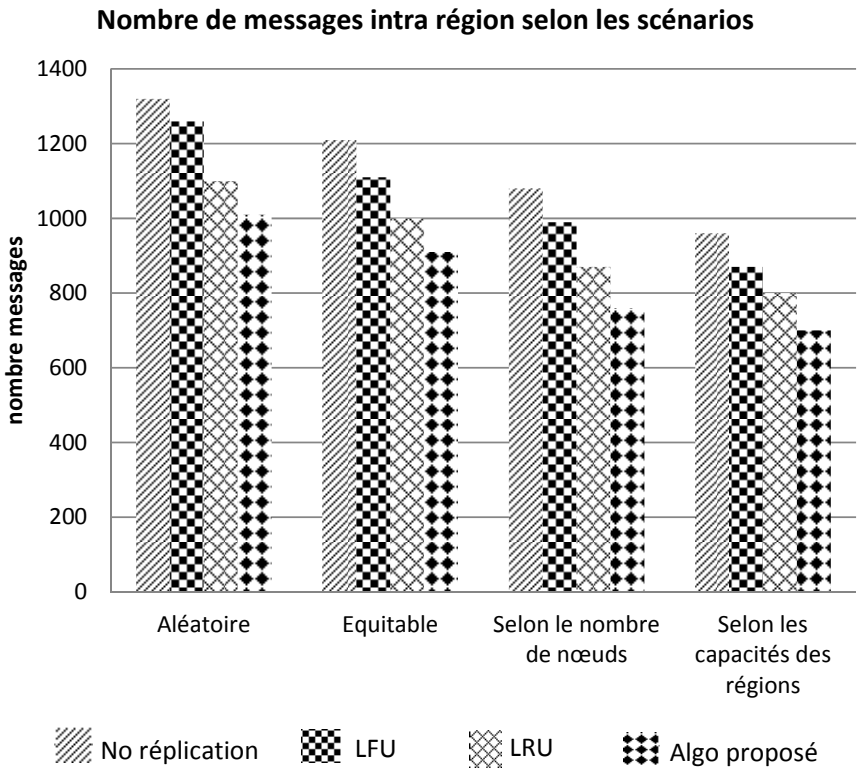


Figure 7.9. Nombre de messages intra régions.

Stratégies\Scénarios	Aléatoire	Equitable	Selon le nombre de nœuds	Selon les capacités des régions
No Réplication	1320	1210	1080	960
LFU	1260	1110	990	870
LRU	1100	1000	870	800
Algorithme Proposé	1010	910	760	700

Tableau 7.6 Nombre de messages intra région

Nombre de message inter régions

Le nombre de message inter régions échangés selon les différents scénarios de configuration de la grille est illustré dans la figure suivante :

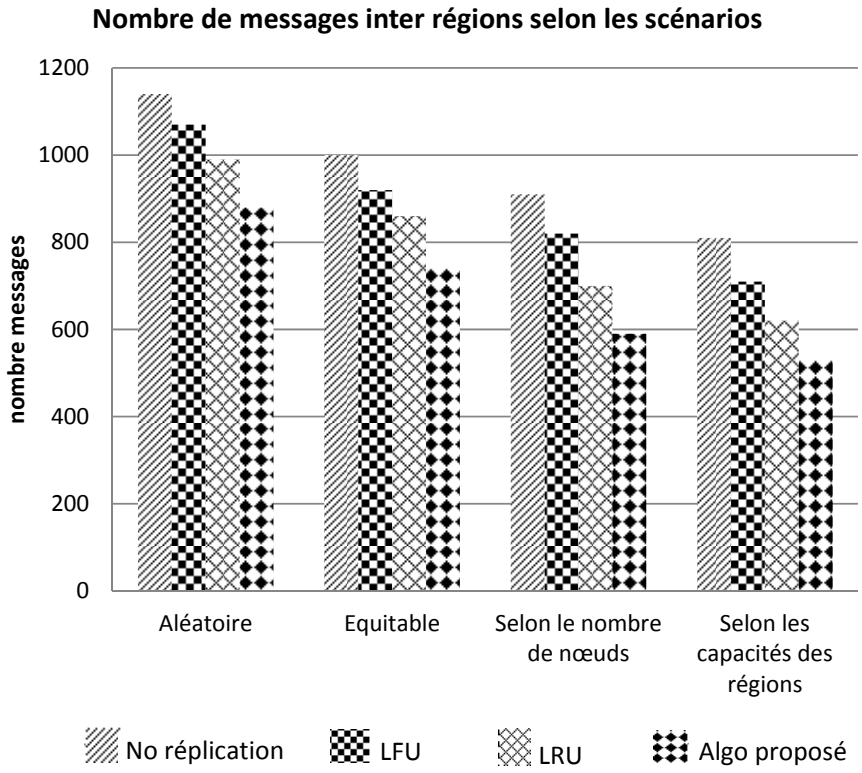


Figure 7.10. Nombre de messages inter régions selon les scénarios.

.

Strétégies\Scénarios	Aléatoire	Equitable	Selon le nombre de nœuds	Selon les capacités des régions
No Replication	1140	1000	910	810
LFU	1070	920	820	710
LRU	990	860	700	620
Algorithme proposé	880	740	590	530

Tableau 7.7 Nombre de messages inter régions selon les scénarios

Les figures 7.9 et 7.10 montrent que la stratégie proposée réduit le nombre de messages intra et inter régions par rapport aux autres stratégies, ce qui confirme les résultats obtenus dans l'expérimentation 4 (consommation des ressources réseaux).

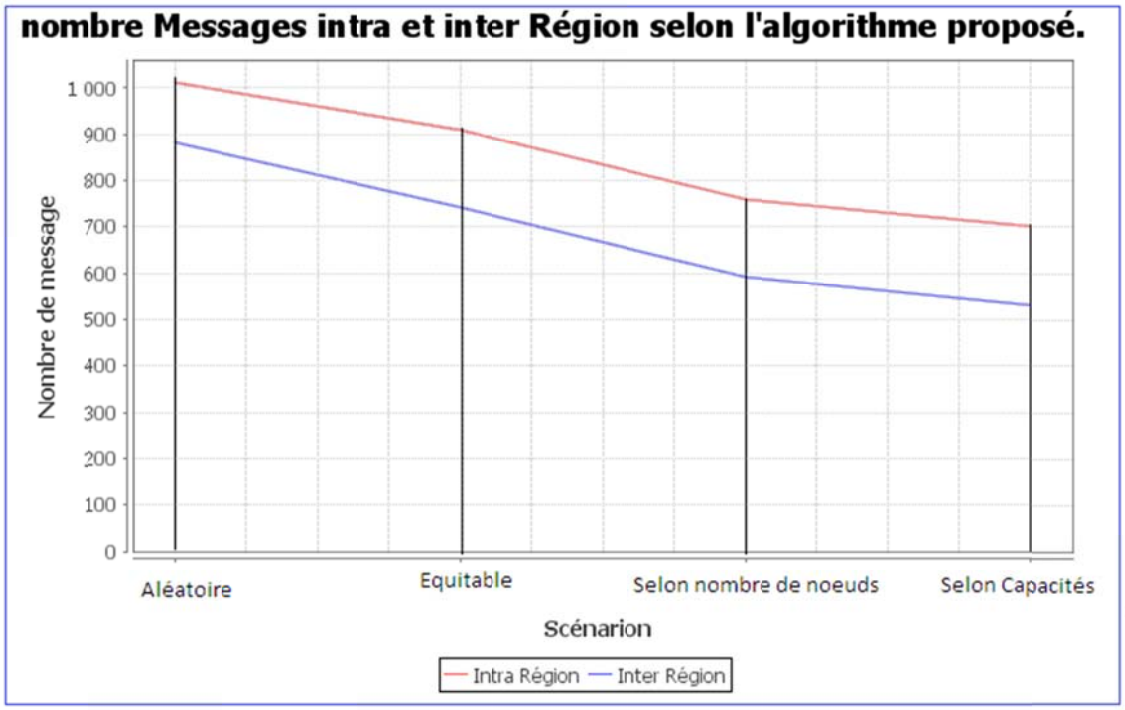


Figure 7.11. Nombre de messages inter et intra région selon les scénarios de l'algorithme proposé.

La figure 7.11 présente un résultat à la fois important et intéressant : Nous remarquons que le nombre de message échangés en intra région est supérieur à celui en inter régions pour tous les scénarios. Ceci s'explique par le fait que nous privilégions la réplication intra région et nous limitons la communication inter régions à plus forte latence.

7.4.3.7 Expérimentation 6 : Temps d'exécution selon le nombre de requêtes.

Cette expérience étudie l'impact de la variation du nombre de requêtes sur le temps de réponse moyen. Nous avons fait varier le nombre de requêtes de 500 à 3500 par pas de 250. Nous remarquons d'après la figure 7.12 que la courbe de l'approche proposée a une tendance linéaire. Le temps moyen d'exécution des requêtes augmente linéairement avec le nombre de requêtes soumises à la grille. Plus le nombre de requêtes augmente plus le temps d'exécution moyen augmente.

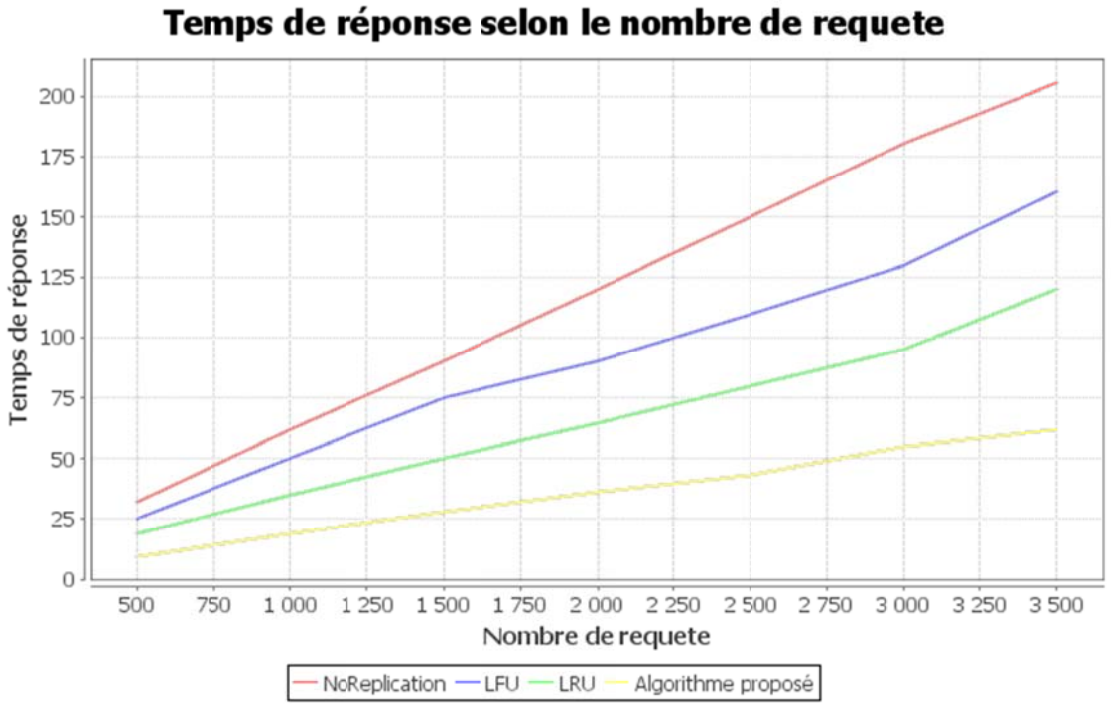


Figure 7.12. Temps de réponse selon le nombre de requêtes.

7.5 Conclusion

Ce dernier chapitre décrit la stratégie de réplication de données proposée pour améliorer l'algorithme BHR. Nous rappelons que cette stratégie pallie à l'inconvénient majeur de l'algorithme en l'intégrant sur une architecture décentralisée. De plus, nous avons levé l'hypothèse de l'existence du nœud master qui constitue le maillon faible de la grille. Nous avons aussi introduit une méthode de suppression de répliques basée sur le taux prédit d'utilisation. Finalement, nous améliorons l'algorithme en intégrant un module de réplication inter-régions. Les résultats des expérimentations obtenus ont montré que la stratégie proposée a permis d'obtenir effectivement des résultats très prometteurs au niveau : des temps de réponses, du nombre de répliques créées, la consommation des ressources réseaux ainsi que le nombre de messages échangés**.**

Conclusion Générale

A travers cette thèse, nous avons étudié le problème de réplication de données dans les systèmes à grande échelle. Plusieurs stratégies de réplication ont été proposées dans la littérature : elles visent principalement à créer et à placer de manière efficace les répliques afin qu'un site puisse trouver les données nécessaires à l'exécution de sa requête. Les approches sur lesquelles se basent ces stratégies diffèrent pour pouvoir offrir un compromis entre les objectifs conflictuels de la réplication, à savoir, la réduction du temps d'accès aux données, l'augmentation du niveau de la disponibilité, la tolérance aux pannes et l'utilisation rationnelle des ressources réseau.

Le problème de réplication de données dans les systèmes à grande échelle est assez complexe, il présente de nouveaux défis, tels que la prise en charge de l'aspect large échelle, l'hétérogénéité et la dynamicité des sites. Une stratégie efficace de réplication doit s'adapter aux changements de la grille et au comportement dynamique des utilisateurs en évitant de nuire aux performances du système en le surchargeant de transfert de données superflus.

D'autre part, le choix de la topologie de la grille est un facteur crucial dont dépendent fortement les performances attendues du système. La stratégie de réplication implémentée est donc affectée par la topologie sous-jacente de la grille.

Dans ce contexte, nous avons proposé des stratégies de réplication de données basées sur un modèle non hiérarchique qui reflète l'état de la grille et qui permet le passage à l'échelle.

Les stratégies décrites dans cette thèse proposent des approches pour contribuer au problème de réplication à grande échelle en s'appuyant sur des modèles différents :

- La stratégie1 discute le nombre optimal de répliques à créer pour un objet donné. Elle est basée sur la notion de popularité et du taux de sollicitation des données. Nous avons opté pour la stratégie de réplication racine carré où, le nombre de répliques d'un objet est proportionnel à la racine carré du taux d'utilisation de cet objet. Les résultats obtenus ont montré une diminution du nombre de répliques créées ainsi que celle du temps de réponse moyen aux requêtes. De plus, grâce au service de défaillance implémenté, le nombre de requêtes perdues a sensiblement baissé.
- La stratégie 2 utilise un modèle de coût pour décider si une réplication est nécessaire. Ce modèle dépend de plusieurs facteurs tels que la vitesse des processeurs, le débit de la bande passante, la taille des répliques ainsi que la capacité de stockage des sites. L'un des plus importants résultats obtenus avec cette stratégie est qu'elle diminue le nombre de répliques créées car une donnée n'est répliquée que si le coût de sa réplication est inférieur au coût de son accès à distance.

Ces deux approches favorisent les réplications intra-cluster et limitent l'utilisation des liens inter-clusters à plus forte latence tout en intégrant un service de gestion des défaillances pour tolérer les pannes.

Enfin, les deux dernières stratégies sont des propositions d'amélioration de stratégies déjà existantes. Il s'agit de la stratégie Fast Spread et BHR. Les modifications apportés aux deux stratégies s'articulent essentiellement sur :

- la topologie : en effet des modifications ont été apportés pour que les deux algorithmes puissent supporter une topologie non hiérarchique
- la politique de suppression : des améliorations ont été apportés dans le choix des répliques devant être supprimés.
- la soumission des requêtes : L'algorithme BHR a été implémentée en décentralisé avec différents scénarios.

Les résultats de simulations obtenus montrent qu'en général les stratégies ont permis d'avoir des résultats satisfaisants pour les différentes métriques mesurées.

Perspectives

Ce travail a permis d'ouvrir quelques perspectives intéressantes que nous récapitulons dans les points suivants :

- Il serait judicieux de combiner les techniques de réplication avec les techniques de placement et d'équilibrage de charge pour pouvoir améliorer les performances des systèmes à large échelle. L'ordonnancement des tâches et la réplication sont deux problèmes qui doivent être étudiés conjointement. Une requête ne peut être satisfaite dans un délai raisonnable que si les données nécessaires à son exécution sont disponibles d'une part et qu'il existe un processus libre pour son exécution d'autre part.
- Dans les grilles de données, la réplication est importante pour garantir la disponibilité des données, mais la gestion de la cohérence des répliques est essentielle. Dans ce but, nous proposons l'extension des stratégies proposées pour pouvoir prendre en charge les requêtes d'écriture en intégrant un service de gestion de la cohérence des données.
- Dans une technique de réplication, non seulement il est essentiel d'avoir des données disponibles à tout moment mais il faut aussi rapprocher ces données des sites qui les demandent pour réduire le temps d'accès. Il serait souhaitable d'expérimenter des techniques pour prédire le besoin des sites formant la grille comme par exemple les techniques de datamining.
- Sur le plan évaluation, les stratégies développées dans le cadre de ce travail, ont été expérimentées en utilisant le simulateur OptorSim. Dans le but de mieux évaluer les apports de ces stratégies, il serait pertinent de les simuler sur un autre simulateur tel que GangSim ou GridNet ou encore en grandeur nature sur des grilles réelles et avec des applications réelles.
- La bande passante joue un rôle essentiel dans le transfert de donnée au cours de la réplication, il serait intéressant de concevoir un outil logiciel de surveillance de réseaux permettant de fournir en temps réel la bande passante à chaque fois qu'il est interrogé.

Bibiliographie

[1] I. Foster et Al. "**The anatomy of the grid: enabling scalable virtual organizations**", *International Journal of High Performance Computing* vol.15, No.3, 2001, pp 200-222.

[2] I. Foster et Al. "**Improving data availability through dynamic model driven replication in large peer-to- peer communities**", *Proceedings 2nd IEEE/ACM international Symposium on Cluster Computing and the grid,* Washington , USA, 2002, pp 376

[3] I. Foster K.Ranganathan, "**Design and evaluation of replication strategies for a high performance data grid**", *Proceedings International Conference on Computing and High Energy and Nuclear Physics,* Beijing, China, 2001.

[4] E. Cohen, S. Shenker, "**Replication Strategies in Unstructured peer-to-peer Networks**", *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications,* 2002, Pittsburgh, PA, USA, pp 177-190.

[5] I. Foster K.Ranganathan, " **Identifying dynamic replication strategies for high performances data grids**", *Proceedings. 3 rd IEEE/ACM International Workshop of grid computing,* London, UK, 2001, pp 75-76.

[6] W.K. Lin et Al. "**Decentralized Replication Algorithms for Improving File Availability in P2P Networks**", *Proceedings of Fifteenth IEEE International Workshop on Quality of Service,* 2007, Chicago, Illinois, pp 29-37.

[7] E. Cohen et Al. "**Search and replication in unstructured peer-to-peer networks**", *Proceedings of the 16th ACM International Conference on Supercomputing (ICS 2002).* New York: ACM *Press, 2002*. pp 84-95.

[8] Won-Sik Yoon et Al. "**Dynamic Data Replication Strategy Based on Internet Hierarchy BHR**", *Computer Science journal* , vol. 3033 , 2004, pp 838–846.

[9] R. Steinmetz et Al. "**The Effectiveness of Realistic Replication Strategies on Quality of Availability for Peer-to-Peer Systems**", *3rd International Conference on Peer-to-Peer Computing*, 2003, Linköping, Sweden, pp 57-64.

[10] S. Shenker et Al. "**A scalable content-addressable network**", *Proceedings ACM SIGCOMM*, 2001, San Diego, California, USA

[11] I. Foster and C. Kesselman. "**The Grid : Blueprint for a New Computing Infrastructure**", *Morgan Kaufmann*, edition, 1998, San Francisco, USA.

[12] D. Bernholdt et Al. "**The earth system grid : Supporting the next generation of climate modeling research**", *Proceedings of the IEEE*, volume 93, 2005, pp 485-495.

[13] M. Sabu et Al. "**Autonomous Data Replication Using Q-Learning for Unstructured P2P Networks**", *Sixth IEEE International Symposium on Network Computing and Applications NCA*, 2007, pp 311-317.

[14] D. Schoder, K. Fischbach, "**Peer-to-Peer Prospects**", *Communications of the ACM*, Vol. 46, N° 2, 2003, pp 27-29.

[15] Q. Rasool et Al. "**Replica placement in multi-tier data grid**", *Proceedings 8th IEEE International Conference on Dependable Autonomic and Secure Computing*, China, 2009, pp 103–108.

[16] T. Yamada et Al. "**Adaptive Replication Method Based on Peer Behavior Pattern in Unstructured Peer-to-Peer Systems**", *Proceedings of 21st International Conference on Data Engineering Workshop*, 2005, pp 1239-1239.

[17] B. Yang, and H .Garcia-Molina, "**Designing a Super-peer Network**", *19th International Conference on Data Engineering (ICDE'03)*, 2003, Bangalore, India pp 49-60.

[18] T. Watanabe et Al. "**A Replica Relocation Method for Improving Search Efficiency in P2P Networks**", *Proceedings of the 1st Conference on Advances in P2P Systems (AP2PS 2009)*, Sliema, Malta, pp 13-18.

[19] N. Shishvan et Al. "**PHFS: A dynamic replication method, to decrease access latency in the multi-tier data grid**", *Future Generation Computer Systems journal* , Vol.27, No.3, 2011, pp 233-244.

[20] Chao-Tung Yang et Al. "**A Dynamic File Replication Strategy in Data Grids**", *IEEE TENCON, International Technical Conference*, Taiwan, 2007, pp 1 – 5.

[21] S. Tewari and L. Kleinrock , "**Proportional Replication in peer to peer Network**", *Proceedings of the 25th IEEE international Conference on Computer communication INFOCOM* 2006, Barcelona, Spain, pp1-12.

[22] I. Foster. "**What is the grid? a three point checklist**." *GRIDtoday*, Vol.1, N°.6, July2002. http ://www.gridtoday.com/02/0722/100136.html.

[23] M. Sozio et Al. "**Near-Optimal Dynamic Replication in Unstructured Peer-to-Peer Networks**," P*roceeding of the conference on principles of database systeme PODS 2008*, Vancouver Canada, pp 281-290.

[24] M. SabuThampi, K. Chandra Sekaran, "**Survey of search and replication schemes in unstructured P2P networks**", *Network Protocols and Algorithms*, Vol. 2, No. 1, 2010, pp 93-131.

[25] B. Rong et Al. "**Load Sharing in Peer-to-Peer Networks using Dynamic Replication**", *Proceedings of the 20th International Conference on Advanced Information Networking and Applications* - vol. 1,2006, Vienna, Austria, pp 1011-1016.

[26] A. Horri et Al. " **A hierarchical scheduling and replication strategy**", *International Journal of Computer Science and Network Security* , vol.8,No.8, 2008.pp 30-35

[27] N. Bonnel et Al. "**Information Replication Strategy in unstructured Peer-to-Peer Networks Using Thematic Agents**", *Proceedings of the 7th Conference on Intelligent Systems Design and Applications*, Rio de Janeiro, Brazil, 2007, pp 697–702.

[28] Steven Lim et Al. "**A Survey and Comparison of Peer-to-Peer Overlay Network Schemes**", *IEEE Communications Survey and Tutorial*, March 2004.

[29] Y. Nemati et Al "**A novel data replication policy in data grid**", *Australian Journal of Basics and Applied Sciences*, Vol.6, N°.7, 2012, pp 339-344.

[30] K. Sashi, A.S. Thanamani, "**Dynamic replication in a data grid using a modified BHR region based algorithm**", *Future Generation Computer Systems Elsevier journal*, vol. 27, No. 2, 2011, pp 202-210.

[31] D.Yang et Al. "**A Comparative study of Replicas Placement Strategies in Data Grids**", P*roceedings of Advances in Web and Network Technologies, and Information Management*, Computer Sciences vol 4537,2007, pp 135-143.

[32] Z. Fadaie, AM. Rahmani, "**A new Placement algorithm in Data Grid**", *International journal of Computer Sciences*,Vol.9, N°.3, 2012, pp 491-507.

[33] R Chang et Al. "**A dynamic weighted data replication strategy in datagrids**", *Proceedings of the IEEE/ACS international conference on computer systems and applications*, Doha, 2008, pp 414–421.

[34] A. Dogan, "**A study on performance of dynamic file replication algorithms for real time file access in Data Grids**", *Elsevier, Future Generation Computer Systems journal* , Vol.25, No.8, 2009 , pp829-839.

[35] M. Tang et Al. "**Dynamic replication algorithms for the multi-tier data grid**", *Future Generation Computer Systems journal*, Vol. 21, No.5, 2005, pp 775- 790.

[36] M. Basoul et Al. "**Enhance Fast Spread replication Strategy for Data Grid**", *Journal of Network and computer Applications* Vol.34, No.2, 2011, pp 575-580.

[37] M.Shorzzaman P.Graham "**Adaptive popularity –driven Replication Placement in Hierarchical Data Grids**", *Journal Supercomput* , Vol. 51, No. 3, 2010, pp 374-392.

[38] A. Horri et Al. "**A Novel Job Scheduling Algorithm for Improving Data Grid's Performance**", *Proceedings 3th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, Barcelona, 2011, pp 142-147.

[39] B.yagoubi et Al. "**Contribution au maintien de la cohérence des répliques dans une grille de données**" , *Proceedings 1st International Conference. on Information Systems and Technologies* ,Algeria, 2011, pp222-230.

[40] "**GriPhyNproject**," http://www.usatlas.bnl.gov/computing/grid/ griphyn/

[41] B. GLACOMONI , http://www.atlantic-83.fr/SiteAtlantic/Cours/

[42] D.G. Cameron et Al "**OptorSim- a grid simulator for studying dynamic data replication strategies**", *International Journal of High Performance Computing Applications*, Vol. 17, No. 4, 2003,pp 403-416.

[43] I. Foster, C.Kesselman "**The Grid 2: Blueprint for a new computing infrastructure**", *second edition, Elsevier* 2004

[44] The European DataGrid Project, http://www.eu-datagrid.org

[45] D.G. Cameron et Al "**OptorSim: a simulation tool for scheduling and replica optimization in data grids**", *Proceedings Computing in High Energy and Nuclear Physics* ,Paysinterlaken,Switzerland , 2004,

[46] D.G. Cameron et Al. "**OptorSim v2.1 Installation and User Guide**", 2006.

[47] L. Randerson et Al. "**Grids for experimental science: The virtual control room**", *Proceedings of the Challenges of Large Applications in Distributed Environments* (CLADE), Honolulu, HI, June 2004, pp 4-11.

[48] T.Soueid "**Grille de calcul : Etat de l'art**", projet de recherche, Ecole supérieure des télécommunications, France, juin 2003

[49] J. Nabrzyski et Al. "**Grid Resource Management**", *Kluwer Publishing*, edition, 2003.

[50] I. Foster et Al. "**The astrophysics simulation collaboratory : A science portal enabling community software development**", *Journal of Cluster Computing*, Vol.5 N°.3, 2002, pp 297-304.

[51] N. Bonnel. "**Stratégies auto-adaptatives pour la gestion de données distribuées sur un réseau P2P**", thèse de doctorat, Université de Bretagne du sud, laboratoire Valoria, 5 décembre 2008. 132p.

[52] W. Ross et Al. "**Topology-Centric Look-Up Service**", *Proceedings of COST264 Fifth International Workshop on Networked Group Communications*, 2003.

[53] R. Schollmeier. "**A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications**", *Proceedings Peer-to-Peer Computing, 1st International Conference*, 2002, pp 101–102.

[54] N. guyen The Tung. "**Environnement pour le calcul pair à pair**", Master informatique et télécommunications, parcours systèmes répartis et géni-logiciels, LAAS-CNRS Toulouse, Juin 2008, 39p.

[55] S. Dejan et Al. "**Peer-to-Peer Computing**" , *HP Laboratories Palo Alto*, In HPL-2002-57, Mars 2002.

[56] D. Barkai, "**An Introduction to Peer-to-Peer Computing**" , *Developer Update Magazine. Peer-To-Peer Architecture Group Microcomputer Reach Lab Intel Corporation*, Février 2000, 7p.

[57] J. Bourgeois, "**Simulation de réseaux pairs à pairs**", *Présentation en vue de la journée RGE sur les simulations et les simulateurs de réseaux distribués* à Nancy, 2010.

[58] C. Shirky: "**What is p2p... and what isn't**" http://openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html, 2000.

[59] D. Caron, D. Poure. "**L'empire du peer-to-peer**", Rapport scientifique, Université Paris Dauphine, 14p, février 2007.

[60] F. Ben Charrada et Al. "An efficient replication strategy for dynamic data grids", Proceedings of the 5th international conference on p2p, parallel, grid, cloud and internet computing, Fukuoka , Japon, 2010 , pp 50–54.

[61] R. Subramanian, Brian D. Goodman, "**Peer to Peer Computing: The Evolution of a Disruptive Technology**", *Idea Group Publishing*, edition December 2004, 300 pages.

[62] I. Foster et Al "**The Open Grid Services Architecture, Version 1.5**"; 2006
http://forge.gridforum.org/projects/ogsa-wg.

[63] F.Z.Bellounar, B.Yagoubi, "**Dynamic data replication in data grid environnement**", *Journal of Parallel & cloud computing pcc*, Vol 2., 2013, pp 65-73

[64] F.Z.Bellounar, B.Yagoubi "**A cluster based architecture using a cost model in data grid**" , *Mediterranean journal of computers and networks*, vol9, N° 4 2013 pp 576-586.

[65] F.Z.Bellounar et Al. "**Dynamic data grid Replication with Storage constraint based on cost model**", *2nd International Symposium on Modelling and Implementation of Complex Systems,* MISC 2012, Constantine, Algérie

[66] E. Adar, B. Huberman. "**Free Riding on Gnutella**", Technical Report, Xerox PARC, septembre 2000

[67] R. Morris et Al. "**Chord: A scalable peer-tp-peer lookup service for internet applications**" , *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, ACM Press, 2001.

[68] KaZaa »: http://fr.wikipedia.org/wiki/Kazaa [on line]

[69] L. Gong. "**JXTA: A Network Programming Environment**", *IEEE Internet Computing*, 5(3), mai-juin 2001

[70] N.Mansouri, "**An Effective Weighted Data Replication Strategy for Data Grid**", *Australian Journal of Basic and applied Sciences*, vol. 6, No. 10, 2012, pp 336-346

[71] Globus alliance.
http : //www.globus.org/

[72] P.Ducrot, "**Sécurité Informatique**",
http : //www.ducrot.org/securite.pdf

[73] "**Sécurité Informatique**",https://www.securiteinfo.com/cryptographie/hash.shtml

[74] E. Leontiadis et Al. "**Creating and maintaining replicas in unstructured peer-to-peer systems**" *Lecture Notes in Computer Science*, Springer, vol. 4128, Berlin / Heidelberg 2006, pp 1015-1025.

[75] L. Gao, "**SAR: Semantic-Aware Replication**". *Thèse de Doctorat*, Université de Texas , Austin, TX, USA, 2005. 175p

[76] G. Belalem, "**Contribution à la gestion de la cohérence de répliques de fichiers dans les systèmes à large échelle**", *Thèse de Doctorat*, département d'informatique, université d'Oran, 2007.

[77] F. Matias et Al. **"Autonomous Replication for High Availability in Unstructured P2P Systems**", *Proceedings of 22nd International Symposium Reliable Distributed Systems*, 2003, Florence, Italy, p. 99- 108.

[78] R. Rahman et Al. "**Replica placement in data grid : Considering utility and risk**", *Proceedings of the international conference on information technology : Coding and computing*, vol. 1, Las Vegas, Nevada, USA, 2005 pp 354–359.

[79] R. Rahman et Al. "**Replica placement design with static optimality and dynamic maintainability**", *Proceedings of the Sixth IEEE international symposium on cluster computing and grid*, Vol 1, Singapore, 2006, pp 434–437.

[80] B.Yagoubi, "**Equilibrage de charge dans les grilles de calcul**", *Thèse de Doctorat d'état en informatique*. Université d'Oran, 2007.

[81] H. Al Mistarihi, C.Yong. "**Replica management in data grid**", *International Journal of Computer Science and Network Security*, 2008, Vol. 8, pp 22–32.

[82] http://www.normalesup.org/~dconduche/TermES/cours/Dijkstra.pdf

[83] https://netbeans.org/index_fr.html

[84] I. Foster et A. Iamnitchi : "**On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing**", Lecture Notes in Computer Science, volume 2735, Springer, pp 118–128.

[85] Napster, 2000, website: http://www.napster.com/