MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE
LA RECHERCHE SCIENTIFIQUE
UNIVERSITE ABDELHAMID IBN BADIS MOSTAGANEM

Faculté des Sciences Exactes & de l'Informatique
Département de Mathématiques et d'Informatique
Filière Informatique

MEMOIRE DE FIN D'ETUDES
Pour l'Obtention du Diplôme de Master en Informatique
Option : Ingénierie des Systèmes d'Information

Placement des répliques dans les grilles de données hiérarchiques

Présenté par :

BELATRECHE Mansour

Encadré par:

Mr DJEBBARA M

Année Universitaire 2012/ 2013

Résumé :

Les grilles de données sont des environnements distribués très étendus où les nœuds sont distribués sur le globe, et les données qui y sont partagées sont très volumineuses.

Cependant, la nature dynamique des grilles et la dispersion de ses sites posent des problèmes de disponibilité de données, d'une part, et des performances d'accès d'autre part. La réplication de données constitue une des principales techniques utilisées à cet effet. Pour bénéficier au maximum du gain que peuvent apporter les répliques de données, leur placement stratégique dans le système est critique. Dans notre mémoire, nous nous intéressons au problème de placement des répliques dans un contexte de grille de données hiérarchique. Plusieurs stratégies de gestion de répliques sont étudiées. Ces stratégies essayent de répondre aux questions : quand, où et comment les répliques sont créées et supprimées. L'accès à la copie la plus proche au bon moment augmente les performances du système. Nous proposons une stratégie de placement des répliques basée sur le choix de l'élément de stockage et le meilleur fichier répliqué et supprimé. Son évaluation par simulation sur OptorSim montre une amélioration considérable des temps d'exécution des jobs et une diminution importante du nombre de répliques entrainant moins d'espace de stockage et donc moins de problèmes de mises à jour.

Mots Clés : grilles de données, réplication, disponibilité de données, temps d'exécution, placement de répliques, simulation, OptorSim.

Abstract

The grids of data are very wide distributed environments where the nodes are distributed on the world, and the data which are shared there are very important.

However, the dynamic nature of the grids and the dispersion of its sites pose problems of availability of data, on the one hand, and performances of access on the other hand. The replication of data constitutes one of the principal techniques used for this purpose. To profit to the maximum of the profit that the data replication can bring, their strategic placement in the system is critical. In this paper, we are interested in the problem of replication placement in a hierarchical data grid. Several strategies of replication management are studied. These strategies try to answer the questions: when, where and how the replicas are created and removed. The access to the nearest replica at the good moment increases the performance of the system. We propose a strategy of replication placement based on the choice of the storage element and the best replicated and removed file. Its evaluation by simulation on OptorSim shows a considerable improvement of the mean time jobs and a major reduction in the number of replicas involving less storage space and thus less problems of updating.

Key words: data grids, replication, availability of data, mean time job, replica placement, simulation, OptorSim.

Sommaire

Chapitre III : Etat de l'art des stratégies de placement des répliques dans les grilles

Chapitre IV : Simulateurs de grilles de données

Chapitre V : Implémentation et simulation

INTRODUCTION GENERALE

Au cours de la dernière décennie, les recherches et les évolutions autours des technologies de réseaux,de microprocesseurs et des supports de stockage de données ont fortement contribué à l'émergencede l'informatique distribuée. Les performances de calcul des microprocesseurs et les capacités destockage de données ne cessent de croître, alors que les nouvelles technologies de réseau ouvrent lesnouvelles potentialités pour les communications. Ce contexte a motivé la communauté informatique às'intéresser aux architectures distribuées à large échelle, afin d'offrir des solutions pour le stockage dedonnées et le calcul réparti à un plus grand nombre d'applications et d'utilisateurs.La collecte des grandes bases de données a atteint des tailles assez importantes mesurées en Petabytes.Les communautés des chercheurs qui doivent accéder et analyser ces données sont souvent importantes,de même pour les ressources de calcul et de stockage. Cette combinaison de bases de données de grandetaille, des utilisateurs et des ressources avec un calcul intensif nécessite une nouvelle infrastructure degestion.

L'exécution fiable et efficace de cesrequêtes exige une gestion soigneuse des données, des réseaux étendus de haut débit et d'autres techniquesavancées qui maximisent collectivement l'utilisation du stockage, de la gestion des réseaux etdes ressources informatiques. La réponse à ces changements est de passer à un modèle informatiqueréparti permettant d'exploiter pleinement les ressources et les capacités offertes. Cet environnement offriraun service et un accès uniforme et économiquement aux ressources d'infrastructure. Cetteévolution est connue sous le nom de "grilles informatiques".

Les grilles informatiquessont actuellement des solutions proposées pour répondre aux besoins des systèmes à large échelle. Ellesfournissent un ensemble de ressources variées, géographiquement distribuées dont le but est d'offrir uneimportante capacité de calcul parallèle, d'assurer un accès rapide et efficace aux données, d'améliorer ladisponibilité et de tolérer les pannes.

Les grilles permettront de répondre à ces besoins en découpant les traitements en calculs dispersés et en partageant les données sur de nombreux sites reliés entre eux par de puissants réseaux.

Répartir efficacement les traitements et les données sur différents nœuds, de façon à minimiser les temps de calcul et les temps de transfert des données, est un problème critique. Il est nécessaire de trouver des moyens pour stocker et rendre rapidement accessible de

grandes masses de données. La problématique qui se pose est de trouver quelles sont les stratégies de placement et de répartition de données qui soient efficaces et fiables dans un tel environnement.

La réplication est une approche très utilisée par les systèmes répartis à grande échelle. Cependant, l'environnement des grilles, précisément les grilles de données, présente différents défis, tel que la réduction du temps de calcul et la disponibilité des données. A cet effet, la sollicitation d'une bonne stratégie de placement des répliques est primordiale.

Notre mémoire comprend cinqchapitres outre une introduction générale et une conclusion.

Le premier chapitre présente les grilles de données. Il commence par l'origine du concept « grille », ensuite une définitiondc ce concept, les types de grilles, les motivations d'utilisation des grilles, les différentes architectures des grilles, enfin les caractéristiques des grilles.

Le deuxième chapitre traite le processus de réplication. Ilprésente le principe de la réplication, ses objectifs, ses formes, ses avantages et inconvénients, ses deux principales méthodes et quelques types. Ensuite le processus de chaque stratégie de réplication et dans quelle topologie l'instaurer.

Le troisième chapitre est consacré pour l'état de l'art dequelques stratégies de placement des répliques dans les grilles. En premier, nous avons exposé une évaluation des performances des stratégies en général, ensuite une liste de stratégies de placement des répliques avec les spécificités de chacune.

Le quatrième chapitre décrit les différents simulateurs de grille de données. Il présente une définition d'un simulateur, puis une description de quelques simulateurs, leurs architectures et leurs fonctionnement comme : GridSim, SimGrid, Alea et OptorSim.

Le cinquième et dernier chapitre est consacré à l'implémentation de notre approche proposée. Il présente une description détaillée du simulateur utilisé, puis des détails sur la configuration de la simulation. Ensuite, on a défini l'algorithme de notre stratégie proposée. Enfin, la simulation, la discussion des résultats par rapport à d'autres stratégies de réplication.

Nous terminerons ce manuscrit par une conclusion qui récapitule notre travail et des futures perspectives.

Liste des tableaux :

Tables des Figures :

I-1- Introduction :

La physique expérimentale produit des grands nombres de données scientifiques (dans certains cas, l'échelle de pétabytes/année). Ces données doivent être employées par des milliers de scientifiques autour du monde. Ce volume important de données et l'exigence de puissance de calcul ont posé de nouveaux problèmes d'accès, de traitement et de distribution des données.

La disponibilité de l'Internet, des ordinateurs puissants et des périphériques a mené à l'utilisation des ordinateurs distribués comme une ressource informatique unifiée, menant à ce qui est populairement connu comme grille informatique. Les grilles permettent le partage, la sélection, et la totalisation d'une série de ressources comprenant le calcul et les systèmes de stockage, les sources de données, et les dispositifs spécialisés qui sont géographiquement distribués pour résoudre des problèmes de grande puissance.

I-2- Origine du concept de grille :

Le mot « grille » vient de l'analogie avec le réseau de distribution d'électricité appelé « Electrical Power Grid » en Anglais.

En effet on peut rapprocher la puissance de calcul actuelle avec ce qu'était l'électricité au début du XXème siècle, où chaque personne devait produire sa propre électricité et la consommer sur place. Ce n'est qu'avec le développement du réseau de distribution électrique que la vraie révolution a pu s'opérer en offrant, au plus grand nombre, la possibilité de recevoir et d'utiliser une puissance électrique grâce à une interface simple (la prise de courant) sans se soucier de la provenance de cette énergie et de la manière dont elle a été fabriquée. [1]

Le terme « The Grid » a été introduit pour la première fois aux Etats-Unis durant les années 1990 pour décrire une infrastructure répartie utilisée dans des projets de recherche scientifique et industrielle.

Le terme de « grille » a été répandu en 1998 par l'ouvrage de Ian Foster et Karl Kesselman comme un paradigme de la grille informatique dans leur ouvrage : « The Grid for a New Computing Infrastructure ». L'accès aux services informatiques devrait être

comparable en coût et en facilité de branchement d'appareils sur le réseau électrique. Cet accès devrait être aisé et transparent quel que soit la puissance nécessaire, la complexité des équipements matériels et logiciels mis en œuvre par les fournisseurs du service. [2]

L'idée des grilles informatiques est semblable à cela à la différence que les ressources fournies aux consommateurs ne sont plus de l'électricité mais de la puissance de calcul, de l'espace de stockage informatique, ou encore l'accès à certaines données. Les unités de production sont alors des ressources informatiques placées dans des sites distants et reliés aux consommateurs par des réseaux de transport de données.
Ainsi, au lieu de se brancher dans une prise électrique, l'usager se connecterait à un réseau informatique sur lequel il pourrait alors consommer des calculs et/ou des données pour faire fonctionner ses programmes. [3]

Le concept de grille a été mis au point pour optimiser au maximum l'utilisation des moyens de traitement et de stockage disponibles. Il a pour objet de fournir, de manière transparente et sure, à des communautés d'intérêt (organisation virtuelle), l'accès à des moyens de traitement et de stockage hétérogènes distribués géographiquement, permettant de disposer de capacités difficilement accessibles, individuellement ou incompatibles avec les propres moyens financiers d'une structure telles que des puissances de calcul de plusieurs téraflops ou des capacités de stockage de l'ordre du pétabyte.

I-3- Définition de la grille :

En général, les grilles sont définies comme un ensemble d'institutions/individus qui décident de partager des ressources et des services sous certaines contraintes et politiques. Ce partage peut être un simple accès aux fichiers, une demande de calcul, une allocation de mémoire ou toute autre opération qui paraît parfois nécessaire pour résoudre quelques problèmes de recherche ou d'industrie qui demandent des ressources importantes. Les ressources partagées entre les institutions/individus qui font partie de la grille sont physiquement distribuées et parfois individuellement administrées.

Une grille est une collection de ressources géographiquement dispersées, fournissant un grand système virtuel de calcul aux utilisateurs.

Les grilles permettent le partage, la sélection et la totalisation d'une série de ressources comprenant le calcul et les systèmes de stockage, les sources de données, et les dispositifs spécialisés qui sont géographiquement distribués pour résoudre des problèmes de grande puissance.

FigI-1 : Exemple d'une grille informatique

I-4- Différents types de grilles :

Il existe différents types de grilles qui correspondent aux différents types de problèmes à résoudre. Certaines grilles sont conçues pour tirer avantage de la puissance de calcul des ressources, alors que d'autres sont conçues pour exploiter la capacité de stockage de ces ressources.

Le type de grille est donc sélectionné selon le type d'application à traiter :

a- Grille de calcul :

En 1998, Ian Foster et Carl Kasselman 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 ». [4]

Depuis leur apparition, les grilles de calcul sont de plus en plus utilisées. Elles 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 ? » 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…). [5]

Comme autres caractéristiques importantes des grilles de calcul, nous pouvons mentionner l'hétérogénéité et l'aspect dynamique. L'hétérogénéité peut être une hétérogénéité 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.
Brièvement, les grilles de calcul agrègent les puissances de calcul des ressources du système. Elles correspondent, de ce fait, aux applications gourmandes en calcul.

b- Grille de service ou grille d'information :

Elles sont utilisées pour les systèmes qui offrent des services pouvant être disponibles sur une simple machine. On distingue deux catégories :

- Grille à la demande : elle agrège dynamiquement différentes ressources pour fournir de nouveaux services. On peut citer comme exemple, le problème de simulation qui nécessite des ressources (en nombre et en type), qui dépendent des paramètres d'exécution ;
- Grille de collaboration : elle rassemble les utilisateurs et les applications en groupe de travail collaboratif. Ce type de grille permet une interaction entre utilisateurs et applications en temps réel, au moyen d'un espace de travail virtuel. [6]

c- Grille de données :

En 1999, le concept de la grille de données (data grid) a été introduit comme architecture intégrée qui est une spécialisation et extension de la grille classique. La grille de données a été conçue pour répondre aux exigences de la combinaison de grands ensembles de données ; distribution géographiquement éloignée des utilisateurs et des ressources ; et des calculs intensifs. [7]

Une grille de données connecte une collection d'ordinateurs et de ressources de stockage géographiquement distribués qui peuvent être situées dans différentes parties d'un pays ou même dans différents pays, et permet à des utilisateurs de partager des données et d'autres ressources. Plusieurs projets de recherche visent à établir des grilles de données scientifiques pour permettre les scientifiques de divers universités et laboratoires de recherches à collaborer entre eux et partager l'ensemble de données et puissance de calcul.

Les grilles de données incluent le concept de base de données, elles assurent la gestion d'accès sécurisé aux données distribuées. Ce type de grilles est donc adapté pour les applications nécessitant une importante capacité de stockage.

L'objectif d'un système de grille de données est double. Premièrement, intégrer des grandes quantités de données stockées dans un grand nombre de sites hétérogènes et géographiquement éloignés dans un système virtuel de gestion de données. Deuxièmement, fournir des services divers pour satisfaire les besoins de calcul performant et intensif de données. [8]

I-5- Motivations :

Il est important de connaitre les raisons qui nous amènent à déployer une grille :

- Exploitation des ressources : Les études montrent que les ordinateurs personnels et les stations de travail sont sous-exploités, que ce soit du point de vue processeur ou disque de stockage.

- Exécution parallèle : les hautes performances de calcul résultent de la combinaison de ces puissances de calcul pour traiter des problèmes qui ne peuvent pas être résolus sur un système unique.

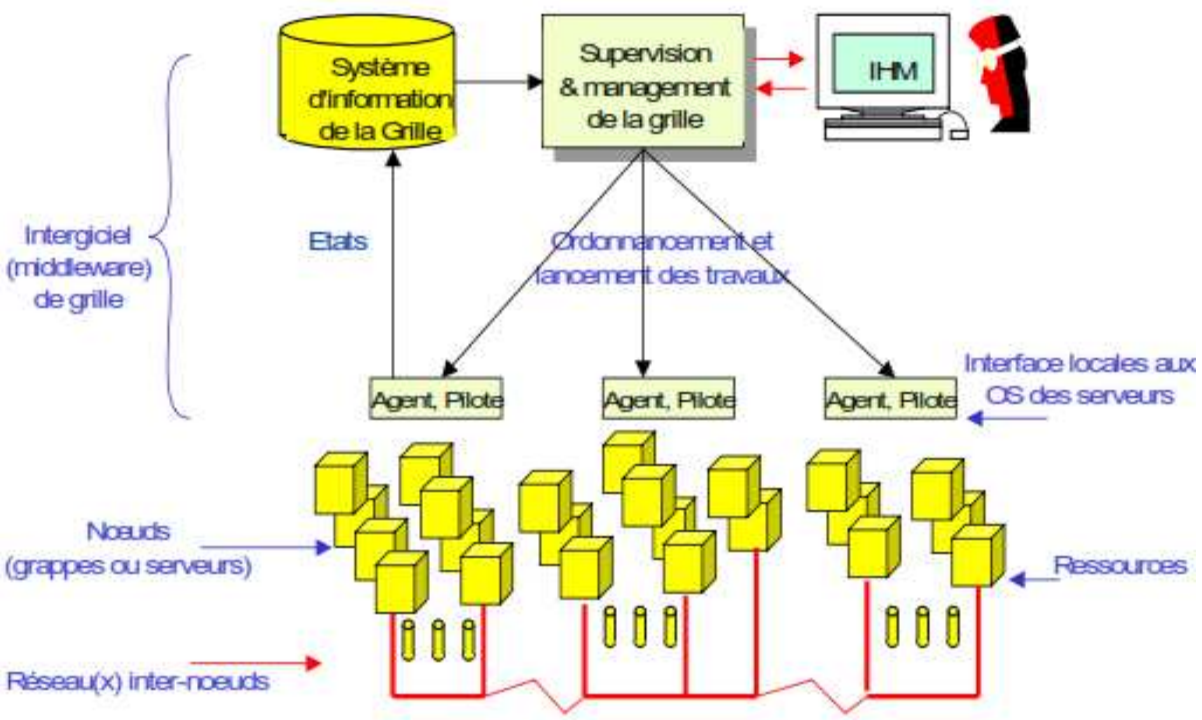
- Organisation virtuelle : cette organisation cache la complexité de la structure de la grille et le partage des ressources hétérogènes, dispersées géographiquement et y accède d'une manière transparente.

- Fiabilité et disponibilité des services : la grille assure la continuité du service même si certaines ressources deviennent inaccessibles en utilisant des logiciels de contrôle et de gestion de la grille qui seront en mesure de soumettre la demande à d'autres ressources.

I-6- Architecture d'une grille :

Une architecture générale des grilles de calcul est proposée. Bien que chaque projet ait sa propre architecture, une architecture générale est importante pour expliquer certains concepts fondamentaux des grilles.

a- La figure ci-dessous illustre une telle architecture :



FigI-2: Architecture simplifiée d'une grille

La grille est physiquement constituée de nœuds (voir figure I.2), qui sont des processeurs avec leurs disques, l'ensemble étant inter connecté via un réseau, dont la qualité peut être primordiale suivant la technologie de grilles retenue. Suivant la voie technologique retenue, ces nœuds sont des serveurs plus ou moins puissants, voire des PC, ou des grappes de serveurs (clusters).

Un logiciel d'interface et de pilotage est installé sur chaque nœud. Il assure le lien entre l'activité locale du nœud, supervisé par les systèmes d'exploitation de chaque serveur, et un outil de supervision et de management global de la grille logiquement unique, mais généralement distribué sur plusieurs machines pour une meilleure fiabilité. L'ensemble des logiciels assurant la gestion de la grille est dénommé l'intergiciel (middleware) de la grille. Celui-ci gère toutes les ressources de la grille, est informé en permanence de leur état: unités de traitement et de stockage constitutives des nœuds, branches de réseau, librairies de programmes… toute entité qui peut être définie et administrée par le gestionnaire de ressources. [9]

b- Une autre architecture de grille proposée qui est organisée en couches empilées où chaque couche est une abstraction représentant un ensemble de services et remplissant un rôle spécifique.

La figure I-3 illustre un tel exemple :

FigI-3 : Architecture en couches d'une grille

- La couche la plus basse correspond à l'infrastructure matérielle. Elle comprend les ressources interconnectées à travers les réseaux LAN ou WAN. Cette infrastructure matérielle comprend notamment des PCs, des stations de travail, des grappes de calcul (clusters), des équipements de stockage, des bases de données, des équipements spéciaux …

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

- La troisième couche fournit les intergiciels (middleware) qui regroupe l'ensemble des services logiciels pour sa mise en œuvre. Comme tout middleware, son principal rôle est d'assurer une interface entre les différentes couches de la grille.

Un intergiciel comprend notamment les fonctions suivantes :

1. Le partage et l'allocation des différentes ressources d'une grille ;
2. L'exécution, l'ordonnancement des travaux ainsi que l'administration d'une grille ;

3. L'ensemble des procédures de sécurisation d'une grille ;
4. Les outils collaboratifs permettant aux divers acteurs de travailler ensemble ;
5. Les outils d'évaluation des performances et de mesure de qualité de service ;
6. Les outils de développement et les interfaces utilisateurs pour le déploiement d'applications.

- La quatrième couche regroupe tous les outils de développement permettant de concevoir des applications optimales pour la grille. On y trouve plus particulièrement des compilateurs, des librairies, des outils de conception d'applications parallèles ainsi que des interfaces de programmation ou API.

- Enfin, la dernière couche regroupe les applications proprement dites qui sont de nature variée : projets scientifiques, médicaux, financiers, d'ingénierie … [10]

I-7- Caractéristiques d'une grille de données :

Les grilles possèdent plusieurs caractéristiques :

- Hétérogénéité
- Transparence
- Autonomie
- Extensibilité et dynamité
- Disponibilité
- Administration distribuée
- Tolérance aux pannes
- Sécurité.

I-8- Conclusion :

Les grilles de calcul et de données constituent à l'heure actuelle des plates-formes pour répondre aux besoins des applications à grande échelle et aux problèmes de gestion des masses de données énormes. Elles fournissent un ensemble de ressources variées, géographiquement distribuées dont le but est de réduire les coûts d'exécution des applications, d'assurer un accès rapide et efficace aux données, d'améliorer la disponibilité des ressources et de tolérer, à une large échelle, les pannes.

II-1- Introduction :

La réplication représente l'une des techniques les plus anciennes et les plus importantes utilisées dans les systèmes parallèles et distribués. C'est une technique basée sur la redondance des données et/ou des processus dans le but d'améliorer les performances d'un système ainsi que sa disponibilité.

La réplique de données est l'un des composants critiques de l'environnement de grille à usage intensif de données. Le besoin de ce composant se fait sentir dans diverses zones d'analyse de données telles que la physique de grande énergie, la bio-informatique, la modélisation de climat et l'astronomie. Par exemple, des térabyte et les pétabytes des données produites par le CERN doivent être partagées et consultées par la communauté de physique de grande énergie autour du monde.

II-2- Principe de réplication :

La réplication est une méthode pratique et efficace pour atteindre une haute performance du réseau dans les environnements distribués, et elle a été appliquée largement dans les zones de la base de données répartie et de l'Internet. La création des répliques peut dérouter des demandes de client à certains sites de réplication et offrir une vitesse d'accès remarquablement plus élevée qu'un serveur simple. En même temps, la charge de travail du serveur initial est distribuée aux serveurs de réplication et la diminue considérablement. Dans la grille de données, la réplication joue également une fonction clé pour améliorer l'exécution du calcul à usage intensif de données. De nouveaux défis sont relevés dans la grille de données, par exemple, les fichiers de données de taille énorme, système de ressources appartenant aux propriétaires multiples, ressources changeantes dynamiquement, et modèle de coût compliqué.

II-3- Définitions et objectifs :

La réplication de données est une technique qui consiste à créer des copies identiques d'ensembles de données (fichiers, bases de données, etc.) sur des sites géographiquement distribués. Chaque copie est appelée 'réplique'.

La création des répliques peut dérouter des demandes de client à certains sites de réplication et offrir une vitesse d'accès remarquablement plus élevée qu'un serveur simple. En même temps, la charge de travail du serveur initial est distribuée aux serveurs de réplication qui fait qu'elle est diminuée considérablement. Dans la grille de données, la réplication joue également une fonction clé pour améliorer l'exécution du calcul à usage intensif de données.

II-4- Formes de réplication :

La réplication peut prendre plusieurs formes :

- Aucune réplication : les données ne sont pas dupliquées entre les sources du système (centralisation du système).
- Réplication partielle : les données sont répliquées sur quelques sources du système.
- Réplication totale : les données sont répliquées sur toutes les sources du système**.**

II-5- Avantages et inconvénients de la réplication :

L'utilisation de la technique de réplication procure un certain nombre d'avantages que nous pouvons résumer comme suit :

- Amélioration de la fiabilité (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 :
  - Diviser la charge de travail entre plusieurs serveurs ;
  - Extensibilité géographique en rapprochant les serveurs des clients.
- Une 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 n'est pas coupé de ses données en cas de défaillance d'une connexion réseau longue distance.

- Temps de réponse :

La réplication améliore les temps de réponse des requêtes d'interrogation pour deux raisons :

- Les requêtes sont traitées sur un serveur local sans accès à un réseau étendu, ce qui accélère le débit.
- Par ailleurs, le traitement local allège la charge du serveur de bases de données central, ce qui permet de moins solliciter le processeur.

Néanmoins, la réplication peut présenter quelques problèmes tels que :

- Le coût :

  La mise à jour ou la modification de la copie originale engendre la mise à jour de l'ensemble des copies, ce genre de coût dépend du nombre de copies.
- Maintien de la cohérence des copies par rapport à la donnée de référence.

- Gestion de la tolérance aux pannes. [11]

II-6- Méthodes de réplication :

Les méthodes de réplication peuvent être classées comme statiques et dynamiques.

a- Réplication statique : après qu'une réplique soit créée, elle existera dans le même lieu jusqu'à ce qu'elle soit effacée manuellement par des utilisateurs ou sa durée est expirée. L'inconvénient de la réplication statique est évident, quand les configurations d'accès du client changent considérablement dans la grille de données, les avantages apportés par la réplication diminueront brusquement.

b- Réplication dynamique : elle prend en compte les modifications des environnements de grille et crée automatiquement de nouvelles réplications pour les fichiers de données populaires ou déplacer les réplications vers d'autres sites si nécessaire pour améliorer la performance.

Le but principal de la réplication dynamique est de diminuer le temps de réponse moyen qui est remarqué par l'utilisateur final. En même temps, de la vue du système entier, la métrique de performance de la consommation de largeur de bande et la fréquence de réplication doivent être estimées pour s'assurer que les réplications dynamiques n'entraînent pas la lourde charge sur le système. [12]

II-7- Types de réplication :

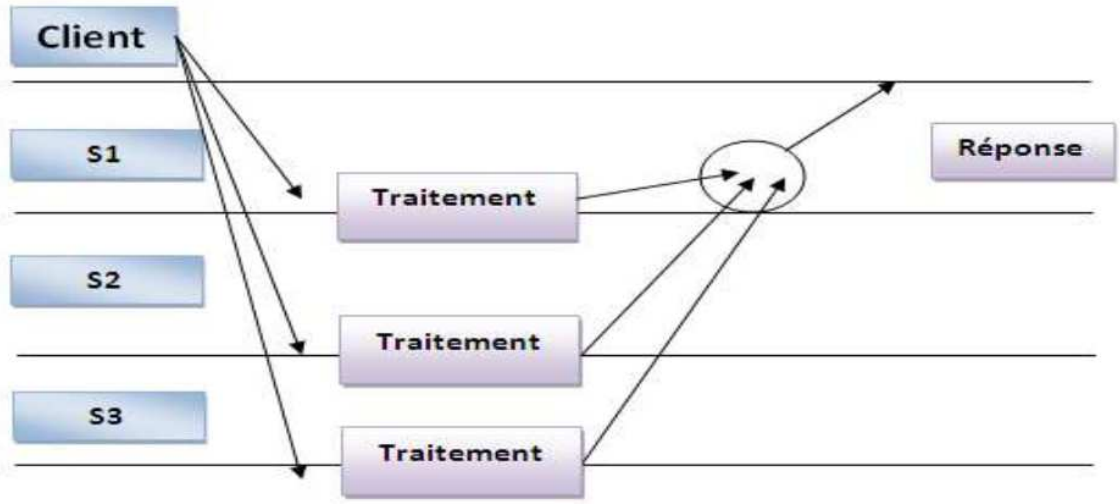
On peut distinguer plusieurs types de réplications qu'on peut en citer quelques-unes :

a- Réplication active, passive, semi-active :

Avec la réplication active, chaque copie joue un rôle identique à celui des autres. Le principe de cette réplication peut être défini en trois phases [13]:

- Réception des requêtes : toutes les copies reçoivent la même séquence (ensemble totalement ordonné de requêtes).
- Traitement des requêtes : toutes les copies traitent les requêtes de manière déterministe.
- Emission des réponses : toutes les copies émettent la même séquence de réponses.

La figure II-1 illustre ce principe :



FigII-1 : Principe de la réplication active

La réplication passive distingue deux comportements d'un composant répliqué :

La copie primaire (primary copy) et les copies secondaires (backups). Dans ce type de réplication, la copie primaire est la seule à effectuer tous les traitements. Les copies secondaires, passivent, surveillent la copie primaire.

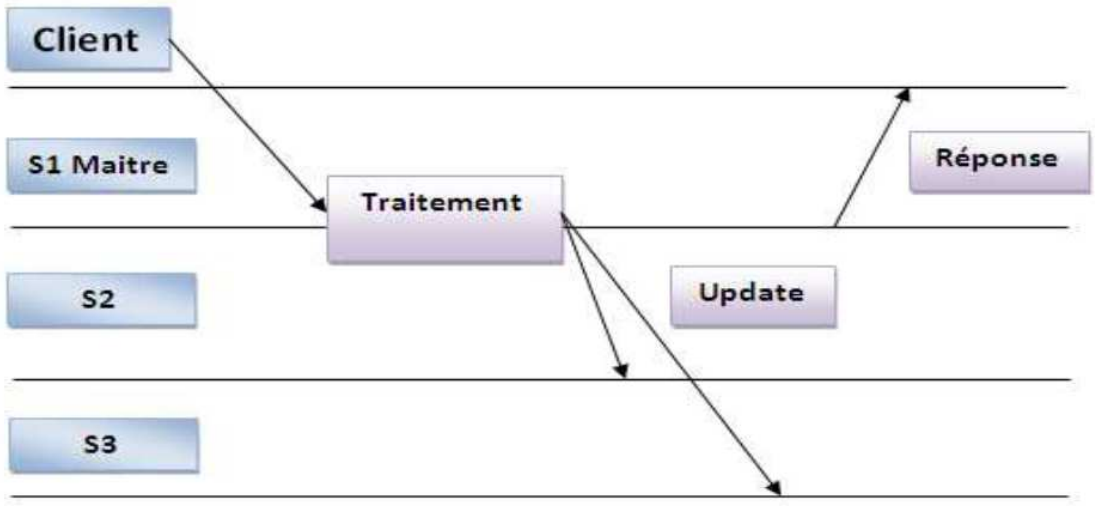
La copie primaire diffuse régulièrement son nouvel état aux copies secondaires ce qu'on appelle un point de reprise.

En cas de défaillance de la copie primaire, une copie secondaire devient la nouvelle copie primaire.

Le principe de la réplication passive est défini comme suit :

- Réception des requêtes : la copie primaire est la seule à recevoir les requêtes.
- Traitement des requêtes : la copie primaire est la seule à traiter les requêtes.
- Emission des réponses : la copie primaire est la seule à émettre les réponses.

Comme le montre La figure II-2, Le client envoie la requête uniquement à la copie primaire S1. Celle-ci traite la requête, construit un point de reprise et l'envoie à l'aide d'un multicast fiable assurant l'ordre FIFO, aux copies secondaires S2 et S3 en précisant le changement de son état (message update). Après la mise à jour de leurs états, les copies secondaires envoient un Ack à la copie primaire. La copie primaire envoie la réponse au client après réception des Ack de toutes les copies secondaires. Le point de reprise permet de synchroniser l'état des copies secondaires avec celui de la copie primaire puisque celle-ci est la seule qui communique avec le reste du système.

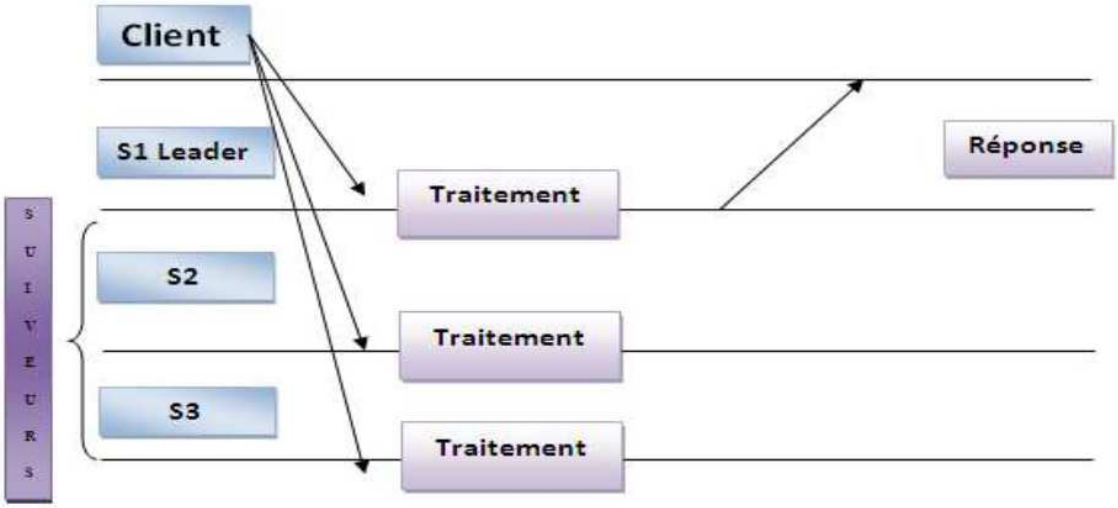


FigII-2 : Principe de la réplication passive

La réplication semi-active (semi-active réplication) se situe à mi-chemin entre la réplication active et la réplication passive. Contrairement à la réplication passive, les copies secondaires ne sont pas oisives. La copie primaire est appelée leader et les copies secondaires sont appelées suiveurs.

La réplication semi-active se définit comme suit (figure II-3): [14]

- Réplication des requêtes : toutes les copies reçoivent les requêtes.
- Traitement des requêtes : toutes les copies traitent toutes les requêtes. Le leader traite une requête dès qu'il la reçoit mais le suiveur doit attendre une notification du leader pour pouvoir traiter une requête.
- Emission des réponses : le leader est le seul à émettre les réponses.



FigII-3 : Principe de la réplication semi-active

b- <u>Réplication à mise à jour synchrone, asynchrone</u> :

La réplication synchrone est aussi appelée "réplication en temps réel". La synchronisation est effectuée en temps réel puisque chaque requête est déployée sur l'ensemble des bases de données avant la validation (commit) de la requête sur le serveur où la requête est exécutée.

Dans le contexte des copies, ce mode est très utile, lorsque les mises à jour effectuées sur un site doivent être prises en compte immédiatement sur les autres sites.

Ce type de réplication assure un haut degré d'intégrité des données mais requiert une disponibilité permanente des serveurs et de la bande passante. Il est fortement dépendant des pannes du système et nécessite de gérer des transactions multi sites coûteuses en ressources. La réplication asynchrone (aussi appelée "Store and Forward" pour « stocker et propager ») stocke les opérations intervenues sur une base de données dans une queue locale pour les propager plus tard à l'aide d'un processus de synchronisation.

Ce type de réplication est plus flexible que la réplication synchrone. Il permet en effet de définir les intervalles de synchronisation, ce qui permet à un site de fonctionner même si un site de réplication n'est pas accessible. L'utilisateur peut ainsi déclarer que la synchronisation sera effectuée la nuit, à une heure de faible affluence. Si le site distant est victime d'une panne, l'absence de synchronisation n'empêche pas la consistance de la base maîtresse. De même une panne de réseau laissera les deux bases, maître et esclave dans des états de consistance. Les opérations sur les données sont plus rapides, puisqu'une requête n'est pas immédiatement déployée. Le trafic sur le réseau est de ce fait plus compact. Par

contre, le planning de réplication est dans ce cas plus complexe, puisqu'il s'agit de gérer les conflits émanant d'un éventuel accès en écriture sur une base esclave entre deux mises à jour.

c- **La réplication symétrique, asymétrique :**

La réplication symétrique ne privilégie aucune copie. Elle permet les mises à jour simultanées de toutes les copies par des transactions différentes. Cette technique permet de gérer la propagation des mises à jour des copies à tout instant.

Cette technique pose le problème de concurrence d'accès risquant de faire diverger les copies. Une technique globale de résolution de conflits doit donc être mise en œuvre.

Par contre, la réplication asymétrique distingue un site maître chargé de centraliser les mises à jour. Appelé aussi site primaire, il est le seul autorisé à mettre à jour les données et est chargé de diffuser les mises à jour aux autres copies dites secondaires (ou esclave).

Dans ce mode de réplication, le site primaire effectue les contrôles et garantit l'ordonnancement correct des mises à jour.

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

II-8- La réplication dans les grilles de données :

Dans les grilles de données, les tailles des données s'élèvent aux Petabytes et la communauté des utilisateurs est dans l'ordre des milliers dans le monde entier. Un tel système a besoin des stratégies de réplication dynamiques (où la création, la suppression et la gestion des répliques sont effectués automatiquement) qui ont la capacité de s'adapter aux changements dans le comportement des utilisateurs.

Les trois questions fondamentales auxquelles n'importe quelle stratégie de réplication doit répondre sont :

- Quand les répliques doivent être créées ?
- Où les répliques doivent être placées ?
- Quelles données doivent être reproduites ?

a- Quand les répliques doivent être créées ?

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

- Vérifier si de nouvelles répliques sont nécessaires.

- Créer effectivement de nouvelles répliques

b- Où les répliques doivent être placées ?

Un autre aspect important dans une stratégie de réplication, est où placer des répliques. Un placement différent d'une réplique a un effet différent sur l'amélioration du système. Généralement, les stratégies de réplication placent la nouvelle réplique suivant des critères choisis au préalable, pour déterminer le site candidat le plus approprié au placement de la réplique. L'ensemble des sites candidats est l'ensemble des sites potentiels qui répondent à ces critères :

- Le site ne contient pas de copie du fichier à répliquer.
- La capacité de stockage du site est suffisante.
- Le temps de transfert entre le site et les utilisateurs potentiels est raisonnable.

c- Quelles données doivent être reproduites ?

Le dernier aspect d'une stratégie de réplication, est le choix des répliques. La sélection des répliques est utilisée pour deux objectifs différents qu'il faut distinguer : d'une part elle est utilisée pour répondre aux besoins d'un utilisateur, et d'autre part, elle est utilisée pour désigner une copie, parmi l'ensemble des copies, à être reproduite.

II-9- Topologies de distribution des répliques :

Dans la structure des grilles de données, on peut distinguer deux topologies les plus utilisées, hiérarchique et anneau. La première ressemble à la structure de gros arbre, plus le nœud est à la racine, plus il a de liens. Cette topologie a été à l'origine présentée par Popek et al en 1981[15] pour améliorer l'exécution des réseaux dans les systèmes informatiques parallèles. La deuxième topologie se caractérise par une connexion circulaire des nœuds. Dans cette topologie, chaque nœud est intermédiaire entre deux autres nœuds.

La distribution hiérarchique est bien adaptée pour des applications à plusieurs niveaux, alors que la topologie d'anneau s'adapte mieux aux applications multiserveurs ou de réplique paire (peer replica). Dans le modèle peer-to-peer, n'importe quelle réplique peut synchroniser avec n'importe quelle autre réplique, et n'importe quelle mise à jour peut être appliquée à n'importe quelle réplique accessible. Dans le modèle hiérarchique, les répliques sont placées à différents niveaux, et communiquent les uns avec les autres. Ce modèle a été mis en application dans beaucoup de systèmes de réplications.

II-10- Conclusion :

La réplique est une bonne zone établie dans l'informatique distribuée, l'Internet, et les communautés de base de données. L'environnement de grille, cependant, introduit de nouveaux et différents défis tels que les temps d'attente élevés et la disponibilité des ressources dynamiques.

Etant donné la taille des ensembles de données, il est impossible qu'un humain prenne des décisions au sujet d'où les données doivent être placées. Clairement une bonne stratégie de réplique est nécessaire pour prévoir et/ou analyser des demandes de données d'utilisateurs et pour placer des sous-ensembles des données (répliques) en conséquence.

III-1- Introduction :

Plusieurs stratégies de placement des répliques ont été proposées dans la littérature, nous présenterons quelques récents travaux relatifs à ce domaine.

III-2- Evaluation des performances des stratégies :

L'étude de différentes stratégies de réplication se base sur plusieurs facteurs tels que :

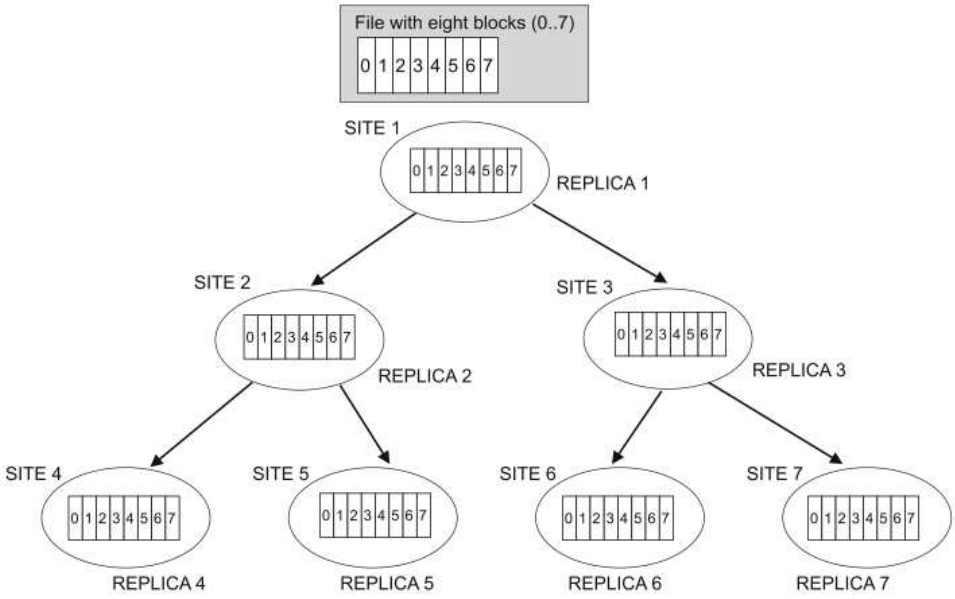
- Temps de réponse : C'est le temps qui s'écoule quand un nœud envoie une demande d'un dossier jusqu'à ce qu'il reçoive le fichier complet. Si une copie locale du dossier existe, le temps de réponse est assumé d'être zéro.
- Spécification de topologies : la topologie d'une grille revient à voir comment elle est arrangée : le nombre de nœuds à chaque rangée, la manière dont les nœuds sont reliées entre eux, la bande passante et le placement des dossiers à travers les divers nœuds.
- Localisation de la réplique la plus proche : diverses méthodes proposées pour localiser la réplique la plus proche. Dans une grille un nœud doit pouvoir identifier la réplique la plus proche, qui est généralement calculé par le « nombre de sauts minimal ». La réplique la plus proche est celle qui est atteinte à travers le nombre minimal d'étapes à partir du nœud. Dans le cas de deux répliques ou plus, l'une d'entre elles est choisie aléatoirement.

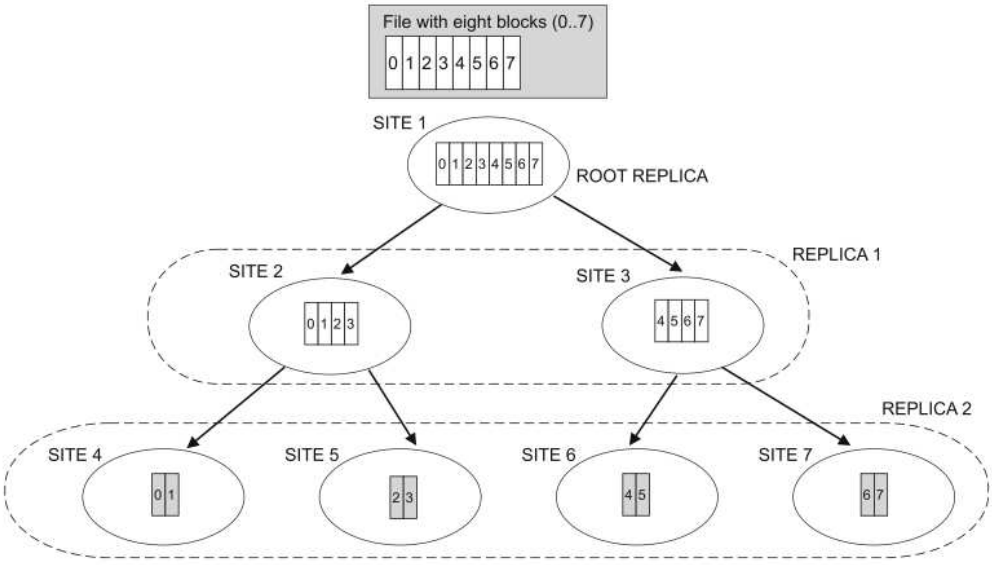
L'exécution des systèmes de réplications dépend du placement de données et du protocole employé pour maintenir la cohérence des répliques. On fait appel à des modèles de coût qui mesurent le coût d'accès aux données d'une grille, le coût de propagation de mises à jour et le coût de stockage employé par les répliques. De là, le système de gestion de la réplication décide quand créer une réplique et où la placée. L'évaluation des coûts est basée sur plusieurs facteurs, tels que les statistiques d'exécution lecture/écriture, la latence d'exécution du réseau, le temps de réponse, la bande passante et la taille de répliques. Ces paramètres changent pendant l'exécution du programme, ainsi ils doivent être mesurés au temps d'exécution et contrôlé par une procédure d'optimisation qui réduit au minimum le coût d'accès des données en changeant dynamiquement le nombre et le placement des répliques.

III-3- Stratégies de placement des répliques dans les grilles :

a- Arrangement de répliques de branchement : Un nouveau modèle pour la réplique de données dans des grilles de données de large échelle [16]

Dans ce travail, les auteurs proposent un nouvel arrangement de réplique appelé l'arrangement de réplique de branchement. BRS (Branch Replication Scheme) est visé pour fournir plus d'évolutivité que les solutions séquentielles par la réplique parallèle d'un fichier dans plusieurs sites. Dans cette solution, une réplication est divisée en plusieurs sous réplications (subreplicas) qui peuvent résider dans différents nœuds de stockage. La figure III-1 affiche les différences entre un arrangement hiérarchique de réplique qui utilise les réplications entières, et celui proposé en cet article.





FigIII-1: Différences entre la réplique hiérarchique (Top) et la réplique de branchement (Bottom).

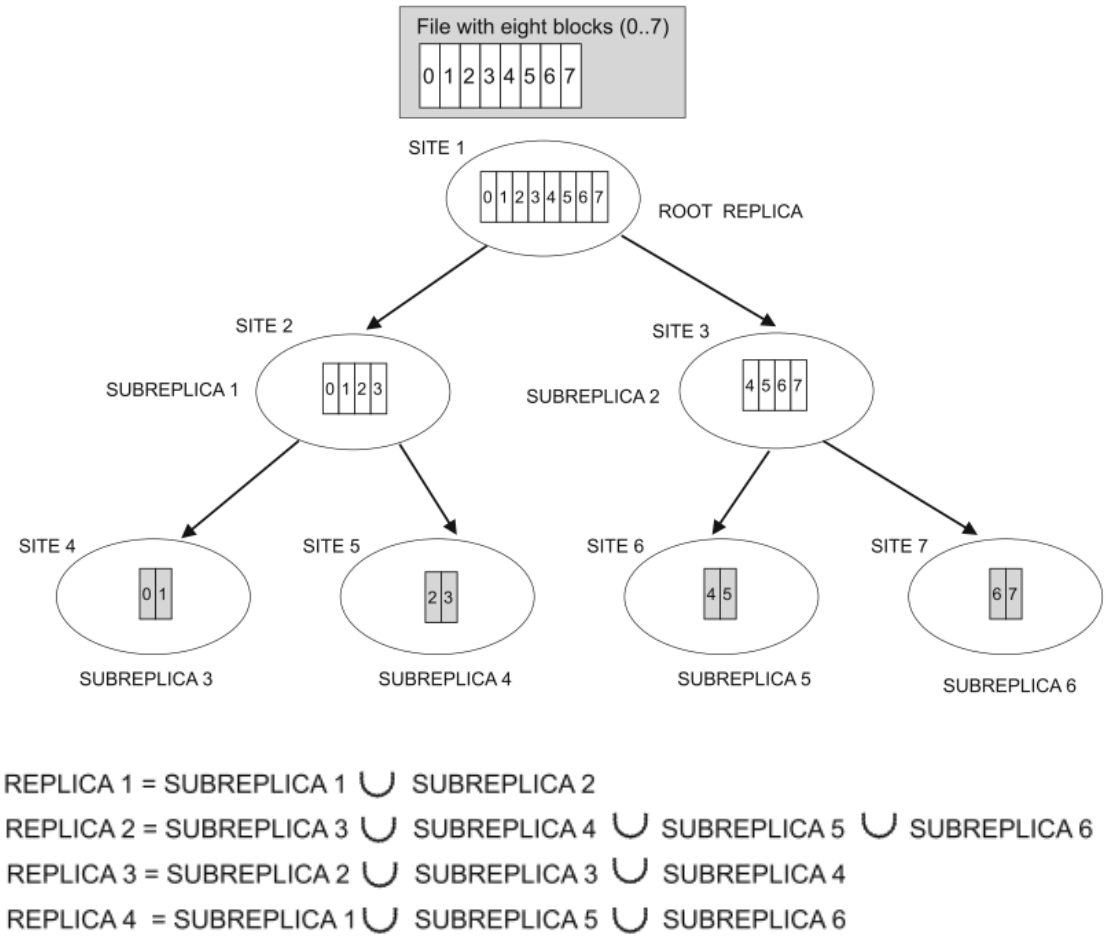
Une réplication R (voir figure III-1) est définie comme ensemble disjoint des subreplicas, R_i (fragments de fichiers) que l'ensemble contient toutes les données du fichier initial (R_R). Formellement, la réplication est définie comme suit : $R = \bigcup_{i=1}^{n} R_i$

Où $\forall i, j \in N : data(R_i) \cap data(R_j) = \emptyset \wedge data(R_R) \cap data(R) = data(R_R)$.

Cette méthode de réplique laisse utiliser cet arrangement dans des environnements de grille avec les ressources de stockage de données qui ont des capacités de stockage très différentes. D'ailleurs, cette approche permet l'accès parallèle aux réplications et elle exige un minimum d'espace dans le nœud où stocker ces réplications. Dans l'approche de réplication des fichiers entiers, chaque fois une réplication d'un fichier de K GB est faite, il y aura besoin du K GB d'espace sur le nœud cible. Pour créer des réplications de n utilisant BRS, ils exigent plus de nœuds cible, qui ne devraient pas être un problème dans de grandes grilles de données. Cependant, chaque nœud cible a besoin de moins d'espace pour enregistrer ses subreplicas.

Pour un fichier initial à K GB et n réplications, on va utiliser 2^{n-1} nœuds cible, chaque nœud cible stocke en moyenne : $\frac{\sum_{i=1}^{N}(\frac{K}{2^{i-1}})}{2^{n-1}}$ GB.

Dans la figure III-2, un arbre avec trois niveaux de réplique est affiché. Le fichier initial (racine) est localisé au SITE 1. Le deuxième niveau de la réplique est constitué par SITE 2 et SITE 3 : la jointure de leurs données est une reproduction entière, et leur intersection est l'ensemble vide. De la même manière, le subreplica dans le SITE 2 est constitué par SITE 4 et SITE 5. Suivant cet arrangement, l'arbre de réplique de données, où ils peuvent voir que dans chaque niveau, une réplication entière est créée. D'ailleurs, une réplication entière peut être obtenue en joignant différentes combinaisons des subreplicas, de sorte qu'une réplication puisse se composer de subreplicas avec différentes profondeurs dans l'arbre de réplique.



FigIII.2 : Exemple de réplique utilisant BRS

b- Un modèle pour prédire la performance optimale de la grille hiérarchique de données [17]

En ce document, les auteurs ont présenté un modèle mathématique pour la grille de données hiérarchique, pour prédire ses performances optimales en termes de temps de réponse

moyen et coût moyen de la largeur de bande. Les paramètres du modèle et quelques hypothèses réalistes sont clairement énoncés. Ils ont également proposé un algorithme optimal de réplication pour la grille de données hiérarchique, qui donne le plus court temps moyen de réponse et le moindre coût moyen de largeur de bande. La performance de l'algorithme proposé de réplication est analysée en détail utilisant le modèle probabiliste construit.

c- La réplique dynamique dans une grille de données utilisant l'algorithme de BHR modifié [18]

Dans une grille de données, le travail de l'utilisateur exige l'accès à un grand nombre de fichiers. Ainsi, pour maximiser la localité de données, la réplique dynamique est exigée. L'algorithme de BHR est basé sur le niveau de localité du réseau. Selon BHR, des sites qui sont situés dans la même région du réseau sont groupés ensemble. BHR essaye de répliquer les fichiers fréquents autant de fois aussi possibles dans une région où la largeur de bande est fournie entre les sites. Si le fichier requit n'est pas dans le même site exécutant le travail, mais disponible dans la même région, il n'y aura aucun retard pour chercher le fichier, de ce fait réduisant au minimum le temps moyen d'exécution d'une tâche, mais seulement dans une certaine mesure. L'algorithme modifié de BHR proposé est également basée sur le niveau de localité du réseau. L'algorithme amélioré essaye de répliquer des fichiers dans une région et stocke la réplication dans un site où le fichier a été consulté fréquemment basé sur l'hypothèse qu'il peut exiger à l'avenir. Le temps moyen d'exécution du travail et l'utilisation du réseau étaient réduits en comparant à BHR.

d- Une étude des stratégies dynamiques de réplication pour améliorer la disponibilité des données dans les grilles de données [19]

A propos de ce travail, ses auteurs ont présenté une étude et une classification des stratégies dynamiques de réplication pour un environnement de grille de données. Ces stratégies ont présenté leurs propres limites pour l'évaluation de leurs méthodes proposées, ainsi que les problèmes qui ont été considérés pour développer ces stratégies, et comment ces problèmes ont été résolus par elles sont également discutés. De cette étude ils ont conclu qu'il reste beaucoup de travail à faire dans le domaine de la réplication de données dans un environnement de grille de données.

En plus, ils ont évoqué quelques problèmes de recherches non résolus. Ils ont observé :

- Il n'existe aucune architecture standard pour un environnement de grille de données. La majorité du travail effectué suit une architecture hiérarchique mais réellement un graphe général est une architecture plus réaliste. Différentes modifications de l'architecture hiérarchique ont été proposées pour aboutir à un vrai environnement de grille.
- Il n'y a aucune stratégie qui aborde tous les problèmes impliqués dans la réplication de données. Par exemple quelques stratégies envisagent de fournir la fiabilité, l'évolutivité, la tolérance aux pannes et l'équilibrage de charge tandis que certaines ignorent totalement ces problèmes. D'autres stratégies considèrent que le fait d'économiser la largeur de bande de réseau est important, alors que certaines ont utilisé plus de largeur de bande que la moyenne.
- La plupart des techniques incluses dans cette étude ont employé la simulation pour évaluer et tester les algorithmes. Alors qu'il fera mieux de les tester dans des scénarios du monde réel pour fournir une évaluation très réaliste des hypothèses qui ont été faites pour ces stratégies.

e- Une stratégie de gestion dynamique de réplication dans la grille de données [20]

Dans ce document, les auteurs ont proposé une nouvelle stratégie de réplication nommée réplication dynamique hiérarchique (DHR : Dynamic Hierarchical Replication) pour un réseau de structure hiérarchisée à 3 niveaux. Le but est de réduire effectivement le temps d'accès au fichier en prenant compte l'espace de stockage limité des sites de grille. DHR efface les fichiers qui existent dans le réseau local LAN (c.-à-d. les fichiers à faible temps de transfert) lorsque l'espace libre n'est pas suffisant pour la nouvelle réplication. Il enregistre les réplications dans le meilleur site où le fichier est consulté fréquemment, au lieu d'enregistrer les fichiers dans plusieurs sites.

Ils ont également présenté une stratégie de sélection de réplication qui choisit le meilleur emplacement de réplication pour l'exécution des tâches des utilisateurs en considérant les requêtes de réplication qu'attendent dans la mémoire et le temps de transfert des données.

f- Combinaison de réplication de données et d'algorithme d'ordonnancement pour améliorer la disponibilité des données dans les grilles de données [21]

La réplication de données est une technique fréquemment utilisée qui peut améliorer la disponibilité des données et la tolérance aux pannes. Puisqu'un environnement de grille est dynamique, la latence du réseau et le comportement d'utilisateurs peuvent changer. Pour aborder ces issues, une stratégie dynamique de réplication bien conçue est nécessaire dans la grille de données. Également le choix de l'algorithme d'ordonnancement a un impact significatif sur les performances du système.

Dans ce travail, les auteurs proposent, en premier, un nouvel algorithme d'ordonnancement des tâches, appelé stratégie d'ordonnancement combiné (CSS : Combined Scheduling Strategy). La stratégie de CSS découvre la meilleure région c.-à-d. la plupart des fichiers demandés existent dans cette région. Ceci diminuera notamment le temps de transfert total, et par conséquent le trafic dans le réseau. Il considère également le nombre de tâches dans la file d'attente, l'emplacement des données requises par la tâche et la capacité de calcul.

En second lieu, ils ont proposé une stratégie dynamique de réplication de données, appelée l'algorithme modifié de réplication dynamique hiérarchique (MDHRA : Modified Dynamic Hierarchical Replication Algorithm). MDHRA substitue les réplications basées sur le dernier moment où la réplication a été demandée, le nombre d'accès, et la dimension de la réplication. Par conséquent, les sites auront leurs les fichiers requis localement au moment du besoin et ceci diminuera le temps de réponse, la latence d'accès, la consommation de largeur de bande et augmente considérablement les performances du système. Il améliore également la latence d'accès en choisissant la meilleure réplication quand divers sites qui détiennent les réplications. La sélection proposée de réplication choisit le meilleur emplacement de réplication parmi d'autres en se basant sur le temps de réponse qui peut être déterminé en considérant le temps de transfert de données, la latence d'accès en mémoire, le temps d'attente dans la file d'attente et la distance entre les nœuds.

g- Réplication dynamique hiérarchique améliorée et stratégie d'ordonnancement pondéré dans la grille de données [22]

Les auteurs, dans ce travail, ont proposé une nouvelle stratégie d'ordonnancement qui considère le nombre de tâches qui attendent dans la file d'attente. La stratégie d'ordonnancement pondérée (WSS : Weighted Scheduling Strategy) découvre la meilleure région c.-à-d. la plupart des fichiers demandés existent dans cette région. Ceci diminuera notamment le temps de transfert total, et par conséquent le trafic dans le réseau.

En second lieu, ils ont proposé une stratégie dynamique de réplication de données, appelée la réplication dynamique hiérarchique améliorée (EDHR : Enhanced Dynamic Hierarchical Replication). EDHR sélectionne le meilleur site pour placer la réplication en considérant la fréquence des requêtes et la dernière fois la réplication a été demandée. Elle choisit également le meilleur emplacement de réplication pour l'exécution des tâches en considérant le nombre de requêtes qui attendent dans la file d'attente de mémoire, temps de transfert des données. S'il n'existe pas assez d'espace pour la nouvelle réplication, EDHR supprime les fichiers en deux étapes. D'abord, il efface ces fichiers avec le délai minimum de transfert (c.-à-d. seulement les fichiers qui existent dans le réseau local LAN et la région locale). En second lieu, si l'espace est encore insuffisant, il utilise le modèle économique.

III-4- Conclusion :

La réplication est un processus primordial dans les grilles de données. Le placement des répliques est une tâche compliquée car elle doit prendre en considération beaucoup de critères et d'exigences. Toutes les stratégies de placement des réplications dans les grilles de données, présentés dans ce chapitre, essayent de trouver des solutions à des problèmes précis dans un environnement choisi, mais vu les comparaisons faites sur ces stratégies, elles ne règlent pas le problème en totalité.

IV-1- INTRODUCTION :

Les systèmes distribués sont naturellement largement étudiés en utilisant la modélisation mathématique, suivie généralement de simulations. Il existe des nombreux simulateurs permettent de simuler une grille de données. Mais ces simulateurs sont en général dédiés à la simulation d'un type d'application précis.

Cependant une recherche étendue a été conduite dans le domaine de la simulation pour modéliser de tels systèmes et comprendre leur comportement.

IV-2- Définitions :

a- Simulation :

La simulation est un moyen utile de test de solutions avant leur validation sous un environnement réel. Elle repose sur la mise en œuvre de modèles théoriques et sert à étudier le fonctionnement et les propriétés d'un système.

b- Simulateur :

Le simulateur est un programme qui est capable d'interpréter des modèles dynamiques, utilisé pour produire les perturbations désirées sur des modèles après l'exécution et pour recueillir les sorties.

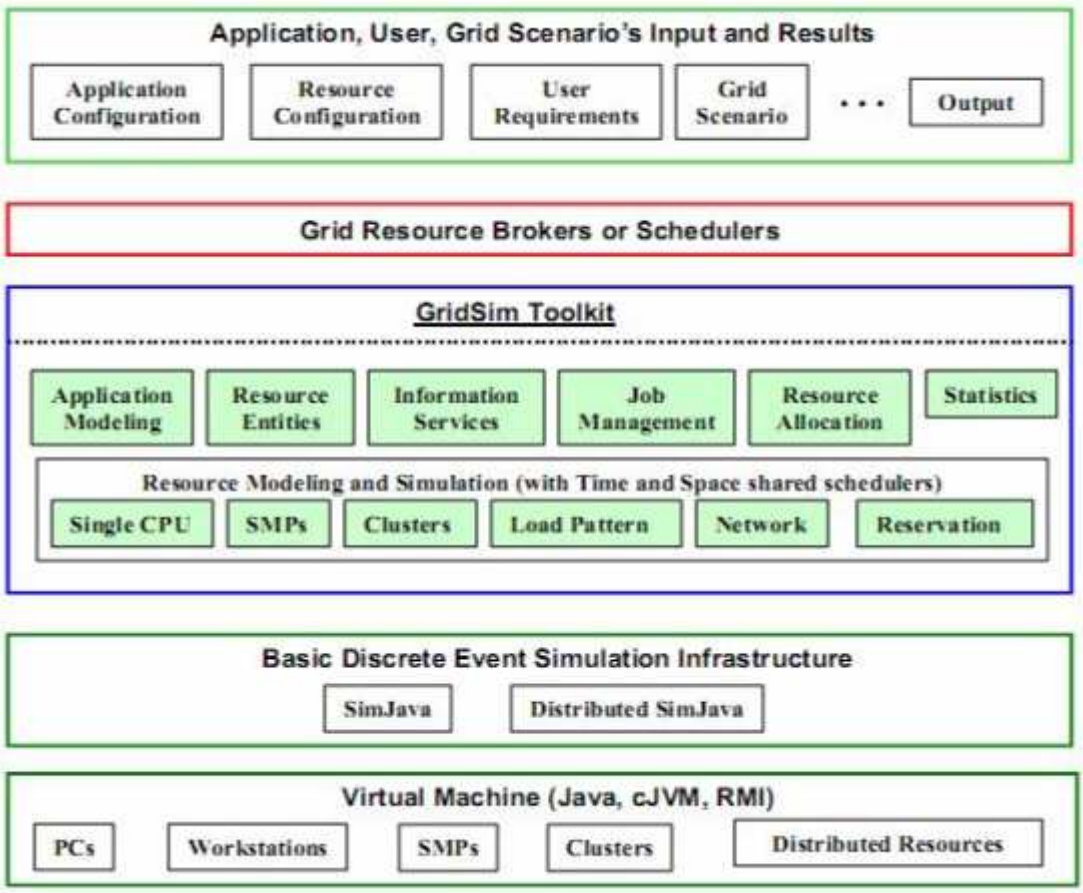
Il représente un outil permettant de simuler des systèmes de grilles informatiques ou des systèmes peer-to-peer, ou bien d'autres. Pour les grilles informatiques, une multitude de simulateurs existent entre autres GridSim, SimGrid, Alea, OptorSim,…

IV-3- Présentation de simulateurs :

a- Le simulateur GridSim :

GridSim [23] est un système de modélisation et de simulation des ressources et des applications d'ordonnancement dans des environnements distribués et parallèles à grande échelle tels que les grilles et les réseaux Peer to Peer. C'est un ensemble d'outils pour la simulation des grilles. Ces outils permettent la modélisation et la simulation des ressources hétérogènes d'une grille, des utilisateurs et des modèles d'applications.

Une architecture et une conception multicouche pour le développement de la plateforme de GridSim et de ses applications est montrée dans la figure IV-1 :



FigIV-1 : Architecture modulaire de la plate-forme et les composants de GridSim

- La première couche est concernée par l'interface extensible et la machine d'exécution de Java, appelée JVM (Java Virtual Machine).
- La deuxième couche est concernée par une infrastructure de base d'évènement discret, tel que SimJava, établie en utilisant les interfaces fournies par la première couche. Une implémentation distribuée de SimJava est aussi disponible.
- La troisième couche est concernée par la modélisation et la simulation des entités de la grille telles que les ressources et les services d'informations.
- La quatrième couche est concernée par la simulation des ressources appelées ressources Broker (intermédiaires) et ressources d'ordonnancements de la grille.
- La dernière couche est consacrée à l'application et la ressource, modélisée avec différents scénarios en utilisant des services fournis par les deux couches inférieurs, pour évaluer l'ordonnancement et la politique de gestion de la ressource.

L'implémentation de GridSim dans Java est une contribution importante puisque Java fournit un ensemble riche d'outils qui augmentent la productivité de programmation, la portabilité d'application, et un environnement d'exécution extensible.

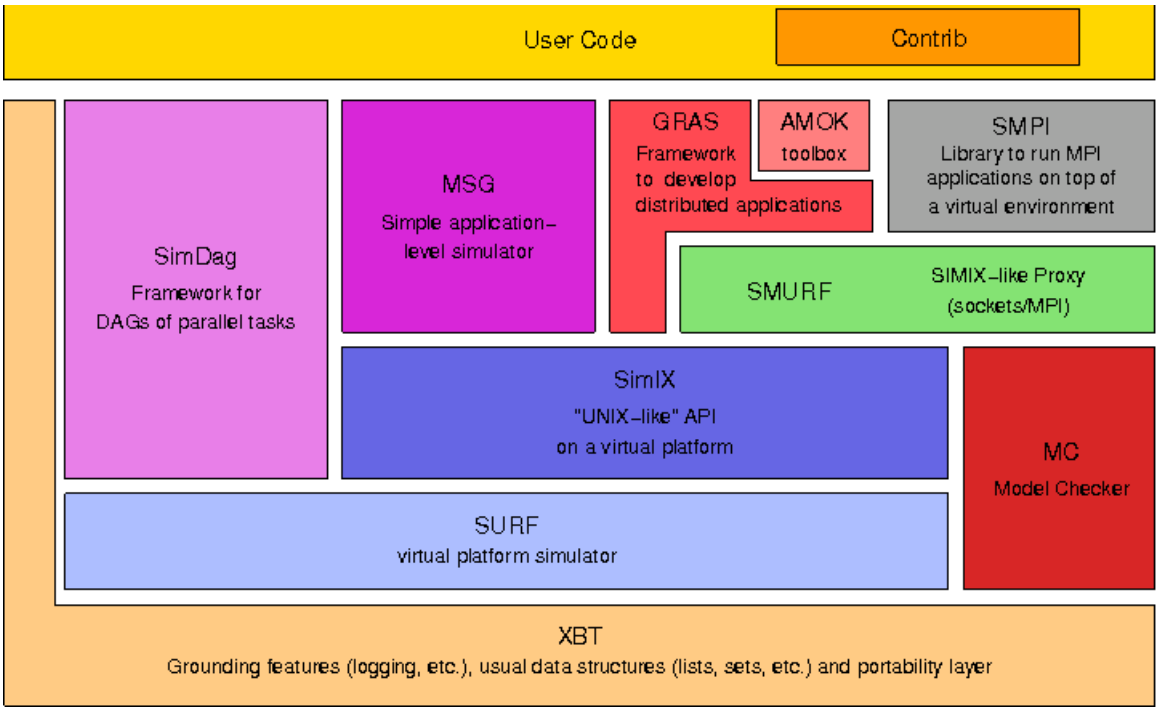
b- Le simulateur SimGrid :

SimGrid [24] est développé à l'université de Californie à San Diego (UCSD) en 2000 par Henri Casanova et Martin Quinson. C'est un système de simulation des applications distribuée dans des environnements informatiques distribués dans le but de développer et d'évaluer des algorithmes d'ordonnancement.

Logiciel développé en C, SimGrid V2 (deuxième version) [25] permet d'étudier des modèles et des topologies plus réalistes que la première version. Cette version intègre les modules suivants :

- Agent
- Endroit
- Tâche
- Chemin
- Conduite

Avec ces concepts, des algorithmes d'ordonnancement avec SimGrid devraient toujours être décrits en termes d'agents qui s'exécutent à des endroits et réagissent en envoyant, recevant et traitant des tâches de l'application simulée.



FigIV-2 : Architecture modulaire de SimGrid

Le SimGrid regorge en lui quelques modules importants (voir la figure IV-2) dont nous pouvons citer:

- **SURF:** noyau de simulation de simGrid (produit les fonctionnalités nécessaire pour la plate-forme virtuelle)
- **MSG:** est l'API (interface) utilisée pour la simulation des applications distribuées
- **SMPI:** consiste à utiliser les applications MPI en faisant une recompilation de celle-ci
- **Simdag:** exprimer la simulation sous forme d'un graphe de tâche parallèle
- **XBT:** utilisé par les autres modules, implémentation des tâches de la gestion des mémoires et des structures très utiles comme des tableaux dynamiques ou des graphes

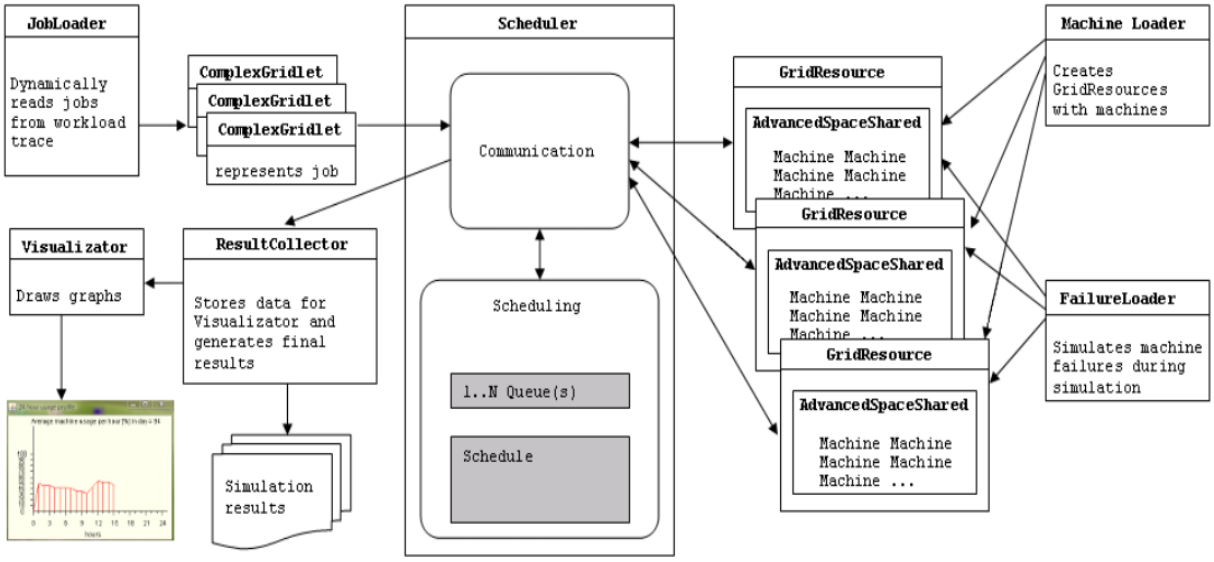
Le programme de SimGrid suit toujours les étapes suivantes :

- Définition du code de chaque agent
- Création des ressources
- Création et attribution des agents aux endroits
- La simulation peut être commencée par la fonction MSG_main.

SimGrid est limité à des systèmes d'ordonnancement et de temps partagé simples, il est difficile de simuler des applications et des programmes d'ordonnancements multiples, surtout dans des environnements tels que les grilles où beaucoup de ressources à grande échelle sont des machines à espace partagé et elles doivent être simulées.

c- Le simulateur Alea:

Basé [26] sur les caractéristiques des algorithmes de règles de priorité, un ensemble d'algorithmes ont été conçus pour les mettre en œuvre dans l'environnement de la grille. Architecture générale de ces algorithmes est illustrée à la figure VI-3, qui détermine la sélection des ressources et la répartition du tâches.



FigIV-3 : Principales parties du simulateur Alea

Alea, version étendue de de GridSim, est utilisé pour simuler le processus de planification dans un environnement informatique en grille. Ce dernier est constitué de plusieurs machines avec différents CPU avec différentes vitesses d'exécution et un nombre important de tâches à ordonnancer. Chaque tâche contient des fichiers différents avec un temps d'arrivée.

Le simulateur Alea est modulaire, composé d'entités indépendantes qui correspondent à la réalité. Il se compose de l'ordonnanceur centralisé, le travail sous-système de la mission, et les ressources de la grille. Ces entités communiquent entre eux par passage de messages. Actuellement, les utilisateurs de la grille ne sont pas directement simulés, mais un générateur de tâches attaché au système de soumission de tâches est utilisé pour simuler les arrivées de tâches.

Le système de soumission stocke les tâches avant et après leur exécution et communique avec l'ordonnanceur pour obtenir une stratégie d'ordonnancement, dont il se sert en outre de sélectionner une ressource pour exécuter une tâche.

Le générateur de tâches est relié au système de soumission et il est utilisé pour simuler les arrivées de tâches. Il génère de nouvelles tâches synthétiques qui apparaissent pendant l'exécution de la simulation. Les temps d'arrivée des tâches correspondent à la distribution statistique sélectionnée.

Alea est capable de simuler des emplois séquentiels et parallèles, des grappes de calcul, les exigences spécifiques du travail ainsi que des systèmes dynamiques tels que les pannes de la machine ou de l'activité des utilisateurs. Il prend également en charge les critères

d'optimisation différents, y compris l'utilisation, ralentissement, temps de réponse, ainsi le temps d'attente normalisé d'utilisateur.

Alea est un outil pour concevoir et tester les algorithmes d'ordonnancement pour certains scénarios typiques de la grille.

d- Le simulateur OptorSim :

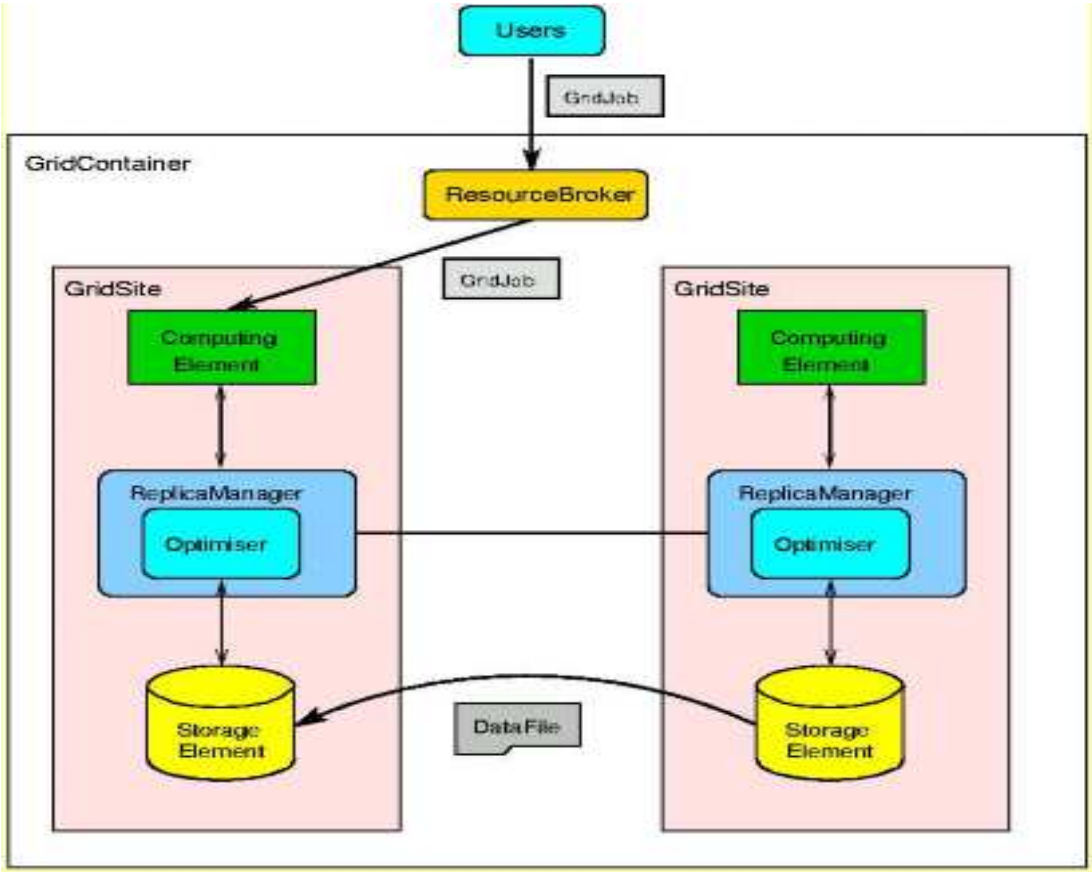
OptorSim [27] est un simulateur de grille de données en open source, écrit en Java, a été développé dans le cadre du projet Européen Data Grid (EDG).

Pour la réplication de données sur les grilles, OptorSim s'avère être le plus adéquat. Il est spécifique à la gestion des répliques sur les grilles de données, et explore le comportement de différents algorithmes de réplication dans plusieurs scénarios. Pour cela, Plusieurs stratégies de réplication sont déjà implémentées et l'intégration de nouvelles stratégies est aisée grâce aux packages spécifiques offerts.

OptorSim reproduit les composants d'une grille réelle. Il est donc constitué de (voir figure IV-4) :

- Une interface utilisateur pour accéder à la grille
- Un Resource Broker qui est un agent responsable de l'ordonnancement des requêtes soumises via l'interface
- Les sites de la grille. Chaque site est en général un cluster de machines constitué de :
    - Une ou plusieurs ressources de calcul (CE : Computing Element). Sur la version actuelle d'OptorSim, on accepte un seul CE seulement. Un CE peut contenir un ou plusieurs processeurs ou « worker nodes ».
    - Une ou plusieurs ressources de stockage (SE : Storage Element)
    - Un gestionnaire de répliques, responsable de la création, suppression, placement, cohérence des répliques

Les gestionnaires de répliques des différents sites interagissent entre eux.



FigIV-4 : Structure du simulateur OptorSim

Fonctionnement :

En entrée, OptorSim a besoin de la configuration de la grille. Il utilise principalement les fichiers de configuration suivant :

- Paramètres généraux (nom du fichier par défaut : parameters.conf) : dans ce fichier, on spécifie :

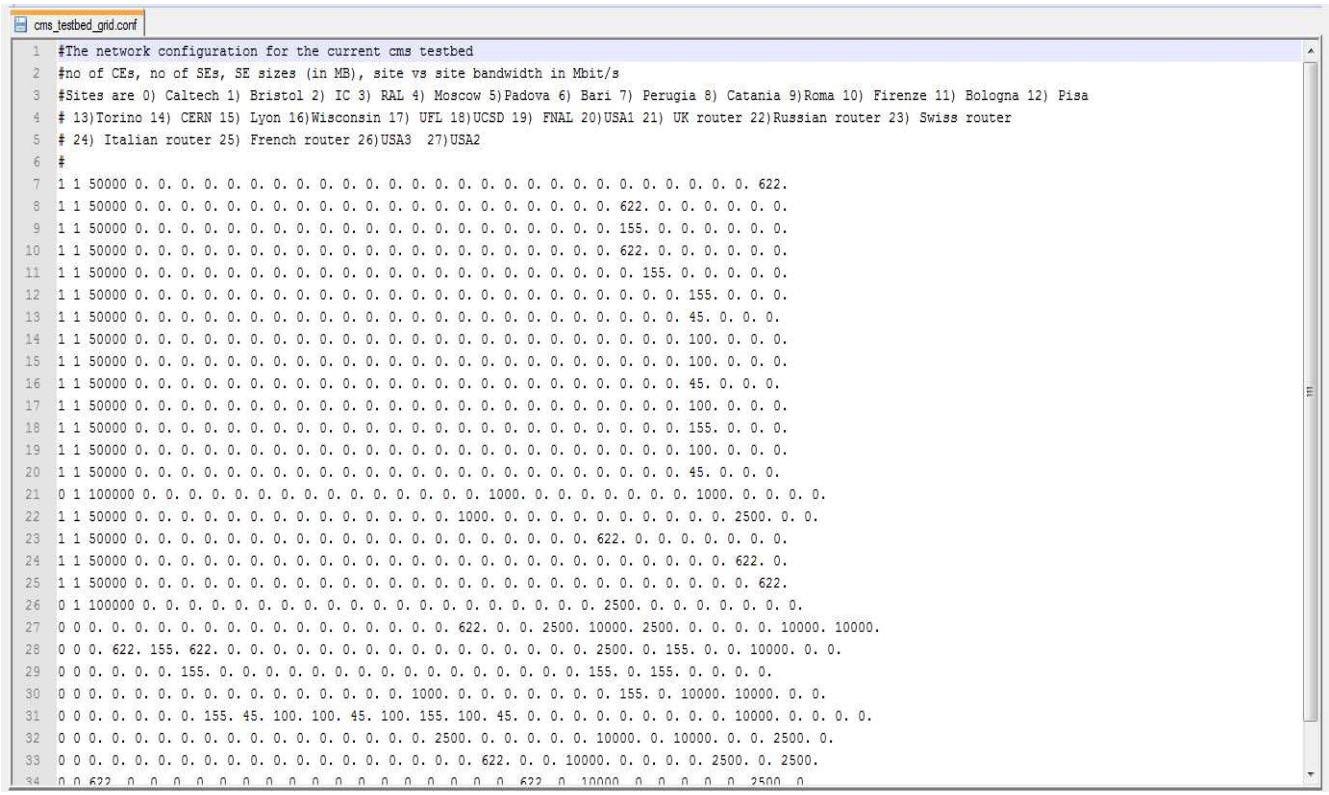
parameters.conf

```
#
# This file contains all the parameters required for OptorSim
# using the Properties class to store this information.
#
grid.configuration.file = examples/edg_testbed_grid.conf
job.configuration.file = examples/edg_testbed_jobs.conf
bandwidth.configuration.file = examples/edg_testbed_bandwidths.conf
number.jobs = 500
#
# The choice of the scheduling algorithms for the RB is:
# (1) random
# (2) queue length
# (3) access cost for current job
# (4) access cost for current job + all queued jobs
scheduler = 4
#
# The categories of users available are:
# (1) Simple - wait job delay between submitting jobs
# (2) Random - wait uniform random time between 0 and 2 * job delay
# (3) CMS DC04 Users
users = 1
#
# The choice of optimisers is:
# (1) SimpleOptimiser - no replication.
# (2) LruOptimiser - always replicates, deleting least recently
#       created file.
# (3) LfuOptimiser - always replicates, deleting least frequently
#       accessed file.
# (4) EcoModelOptimiser - replicates when eco-model says yes, deleting
#       least valuable file.
# (5) EcoModelOptimizer Zipf-like distribution
# (6) MaStrategieOptimizer - Ma strategie Algorithm.
optimiser = 5
#
```

FigIV-5 : contenu du fichier parameters.conf

- les noms des fichiers de configurations et leur chemin d'accès (autres fichiers que parameters.conf).
- le nombre de jobs (requêtes) à exécuter sur la grille.
- l'algorithme d'ordonnancement des jobs (utilisé par le Resource Broker).
- l'algorithme de réplication des données (utilisé par le gestionnaire de répliques).
- la distribution initiale des fichiers originaux (données originales non répliquées).

- Topologie de la grille (exemple : cms_testbed_grid.conf) : dans ce fichier, on décrit la topologie de la grille par une matrice où chaque ligne correspond à un site de la grille et où :



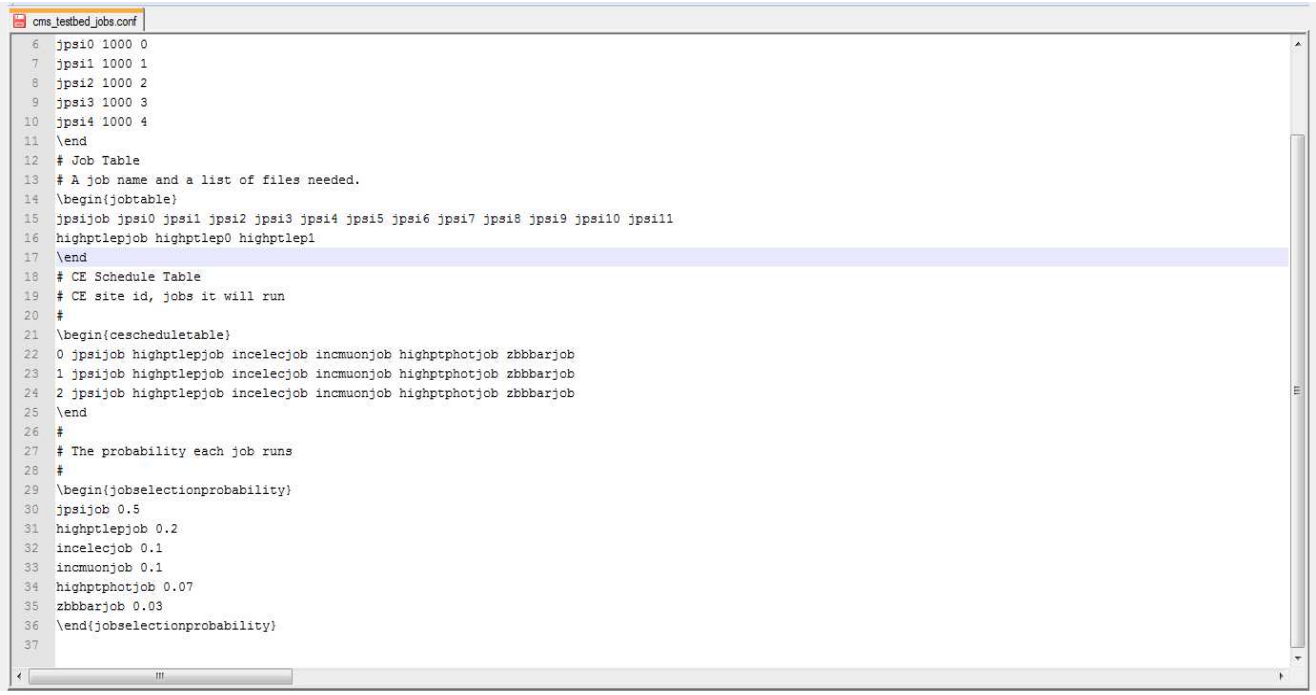
cms_testbed_grid.conf

```
#The network configuration for the current cms testbed
#no of CEs, no of SEs, SE sizes (in MB), site vs site bandwidth in Mbit/s
#Sites are 0) Caltech 1) Bristol 2) IC 3) RAL 4) Moscow 5)Padova 6) Bari 7) Perugia 8) Catania 9)Roma 10) Firenze 11) Bologna 12) Pisa
# 13)Torino 14) CERN 15) Lyon 16)Wisconsin 17) UFL 18)UCSD 19) FNAL 20)USA1 21) UK router 22)Russian router 23) Swiss router
# 24) Italian router 25) French router 26)USA3  27)USA2
#
1 1 50000 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 622.
1 1 50000 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 622. 0. 0. 0. 0. 0.
1 1 50000 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 155. 0. 0. 0. 0.
1 1 50000 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 622. 0. 0. 0.
1 1 50000 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 155. 0. 0.
1 1 50000 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 155. 0.
1 1 50000 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 45.
1 1 50000 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 100.
1 1 50000 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 100.
1 1 50000 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 45.
1 1 50000 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 100.
1 1 50000 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 155.
1 1 50000 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 100.
1 1 50000 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 45.
0 1 100000 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1000. 0. 0. 0. 0. 0. 0. 0. 1000. 0. 0. 0.
1 1 50000 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1000. 0. 0. 0. 0. 0. 0. 0. 0. 0. 2500. 0. 0.
1 1 50000 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 622. 0. 0. 0. 0. 0. 0.
1 1 50000 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 622. 0.
1 1 50000 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 622.
0 1 100000 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 2500. 0. 0. 0. 0. 0. 0. 0.
0 0 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 622. 0. 0. 2500. 10000. 2500. 0. 0. 0. 0. 10000. 10000.
0 0 0. 622. 155. 622. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 2500. 0. 155. 0. 0. 10000. 0. 0.
0 0 0. 0. 0. 0. 155. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 155. 0. 155. 0. 0. 0. 0.
0 0 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1000. 0. 0. 0. 0. 0. 0. 0. 155. 0. 10000. 10000. 0. 0.
0 0 0. 0. 0. 0. 0. 155. 45. 100. 100. 45. 100. 155. 100. 45. 0. 0. 0. 0. 0. 0. 0. 0. 0. 10000. 0. 0. 0. 0.
0 0 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 2500. 0. 0. 0. 0. 0. 10000. 0. 10000. 0. 0. 2500. 0.
0 0 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 622. 0. 0. 10000. 0. 0. 0. 0. 2500. 0. 2500.
0 0 622. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 622. 0. 10000. 0. 0. 0. 0. 0. 2500. 0.
```

FigIV-6 : contenu du fichier cms_testbed_grid.conf

- la première colonne désigne le nombre de worker nodes dans le site correspondant.
- la deuxième colonne désigne le nombre de ressources de stockage (SE) dans le site correspondant.
- la troisième colonne désigne la taille des ressources de stockage en Mo.
- le reste des colonnes de la matrice est une matrice carrée représentant la bande passante en Mb/s entre deux sites.

- Données de la grille et les requêtes soumises (exemple : cms_testbed_job.conf) : Ce fichier décrit :



cms_testbed_jobs.conf

```
jpsi0 1000 0
jpsi1 1000 1
jpsi2 1000 2
jpsi3 1000 3
jpsi4 1000 4
\end
# Job Table
# A job name and a list of files needed.
\begin{jobtable}
jpsijob jpsi0 jpsi1 jpsi2 jpsi3 jpsi4 jpsi5 jpsi6 jpsi7 jpsi8 jpsi9 jpsi10 jpsi11
highptlepjob highptlep0 highptlep1
\end
# CE Schedule Table
# CE site id, jobs it will run
#
\begin{cescheduletable}
0 jpsijob highptlepjob incelecjob incmuonjob highptphotjob zbbbarjob
1 jpsijob highptlepjob incelecjob incmuonjob highptphotjob zbbbarjob
2 jpsijob highptlepjob incelecjob incmuonjob highptphotjob zbbbarjob
\end
#
# The probability each job runs
#
\begin{jobselectionprobability}
jpsijob 0.5
highptlepjob 0.2
incelecjob 0.1
incmuonjob 0.1
highptphotjob 0.07
zbbbarjob 0.03
\end{jobselectionprobability}
```

FigIV-7 : contenu du fichier cms_testbed_jobs.conf

- l'ensemble des fichiers distribués dans la grille. Chaque fichier est défini par son nom, son identifiant et sa taille en Mo.
- l'ensemble des requêtes ou jobs lancés dans la grille. Pour chaque job, on définit l'ensemble des fichiers dont il a besoin pour son exécution, son temps d'exécution, sa probabilité d'exécution.

Pour exécuter les jobs spécifiés dans le fichier de configuration, le Resource Broker utilise l'algorithme d'ordonnancement spécifié dans le fichier des paramètres. Une fois un job affecté à un site, il a besoin de retrouver les données nécessaire à son exécution. Trois cas se présentent :

- Données présentent sur le site d'exécution : le job est alors exécuté immédiatement.
- Réplication des données sur le site d'exécution : Si les données nécessaires ou une partie d'elles n'existent pas sur le site du job, alors le gestionnaire de répliques utilise l'algorithme de réplication spécifié dans le fichier de paramètres pour créer et placer les répliques nécessaires.
- Exécution à distance : dans certain algorithmes de réplication, le gestionnaire des répliques peut décider de faire une exécution à distance au lieu de répliquer les données (ex : modèle économique).

En sortie, OptorSim donne l'évaluation des performances de la grille pour la configuration spécifiée. Les paramètres d'évaluation sont les suivants :

- Le temps d'exécution des jobs : C'est le temps moyen d'exécution d'un job, c'est à dire le temps total d'exécution de tous les jobs divisé par le nombre de jobs. 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.
- Le nombre de répliques : C'est le nombre de répliques créé durant l'exécution des jobs. Plus ce nombre est important plus il influe négativement sur le temps d'exécution des jobs (+temps de réplication et transfert de données). Ce nombre peut être amélioré par un placement stratégique des répliques.
- Le nombre d'accès locaux : C'est le nombre de fois où les données nécessaires se trouvent sur le site d'exécution du job soit à l'origine soit après réplication des données vers ce site. Plus ce nombre est important plus le temps d'exécutions des jobs est meilleur (-temps de transfert).
- Le nombre d'accès distants : C'est le nombre de fois où le job fait un accès distant pour retrouver les données nécessaires.
- La consommation en ressources de stockage : Ce paramètre est défini par le pourcentage de l'espace de stockage occupé par rapport à l'espace disponible. Ce paramètre est affecté par le nombre de répliques créées. Il est meilleur quand le nombre de réplications est minimal.

OptorSim est un bon simulateur de grilles qui permet de tester des stratégies de réplication et de mesurer les performances de la grille de données en donnant un ensemble d'information très intéressantes à la fin de la simulation et qui permet également de faire cette dernière en un temps minime.

IV- 4- Conclusion :

Nous avons essayé dans ce chapitre de citer quelques simulateurs de grille, ainsi que la spécificité de chacun. Mais vu notre thème, on a donné plus de détail sur le simulateur OptorSim qui est conçu pour tester les différentes stratégies de réplication.

V-1- Introduction :

La plupart des stratégies de réplication, dans le domaine de grilles informatiques, sont implémentées et évaluées par simulation. Nous aussi, nous avons choisi de concrétiser et d'implémenter notre approche par simulation.

En effet, dans ce chapitre, nous nous intéressons à la mise en œuvre et à l'évaluation de notre stratégie de réplication dynamique dans une grille de données hiérarchique à l'aide du simulateur OptorSim que nous avons déjà cité dans le 4ème chapitre. Une comparaison est effectuée entre les résultats obtenus des expérimentations réalisées par l'application de notre approche et celles implémentées sur le simulateur.

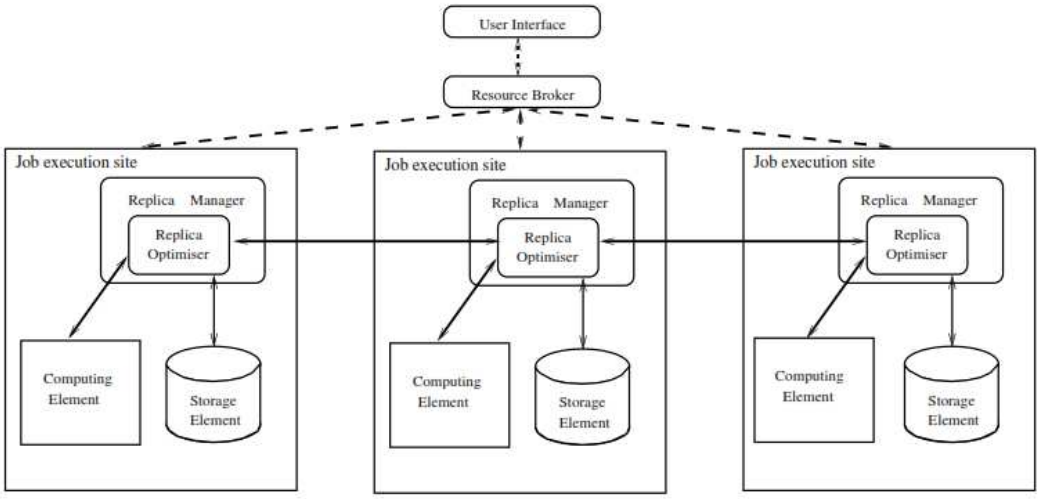
V-2- Conception de simulation :

OptorSim est un module de simulation écrit en Java™. Il a été développé pour imiter la structure d'une vraie grille de données et pour étudier l'efficacité des algorithmes d'optimisation de reproduction dans un tel environnement.

a- Architecture :

Une des considérations principales de conception d'OptorSim est de modéliser les interactions entre les différents composants d'une grille de données aussi normalement que possible.

Par conséquent, la simulation est basée sur l'architecture du projet d'European DataGrid [28] comme illustré sur la figure suivante :



FigV-1 : Architecture de DataGrid simulée

La simulation a été construite supposant que la grille se compose de plusieurs sites, qui peuvent fournir des ressources de calcul et de stockage de données pour les travaux soumis.

Chaque site se compose de zéro élément de calcul ou plus et zéro élément de mémoire ou plus. Les éléments de calcul exécutent les jobs, qui utilisent les données dans les fichiers enregistrés sur des éléments de mémoire.

Un courtier de ressource (Resource Broker) contrôle l'ordonnancement des travaux aux éléments de calcul. Les sites en dehors des éléments de mémoire ou de calcul agissent en tant que nœuds ou routeurs de réseau.

Puisque la grille de données est un environnement fortement dynamique, l'intergiciel devrait pouvoir faire face aux changements du statut pour performer l'allocation des ressources.

En particulier, le service d'optimisation de grille devrait éviter de prendre des décisions irrévocables à priori au temps d'ordonnancement du travail au sujet de quelles réplications de fichier seront employées pour accéder à des données pour un travail particulier.

Par conséquent, nous voyons l'optimisation comme une activité continue, qui est accomplie à deux moments pendant la vie d'un travail [29] :

1. La première phase d'optimisation aura lieu quand l'élément de calcul où le travail devrait fonctionner est choisi.

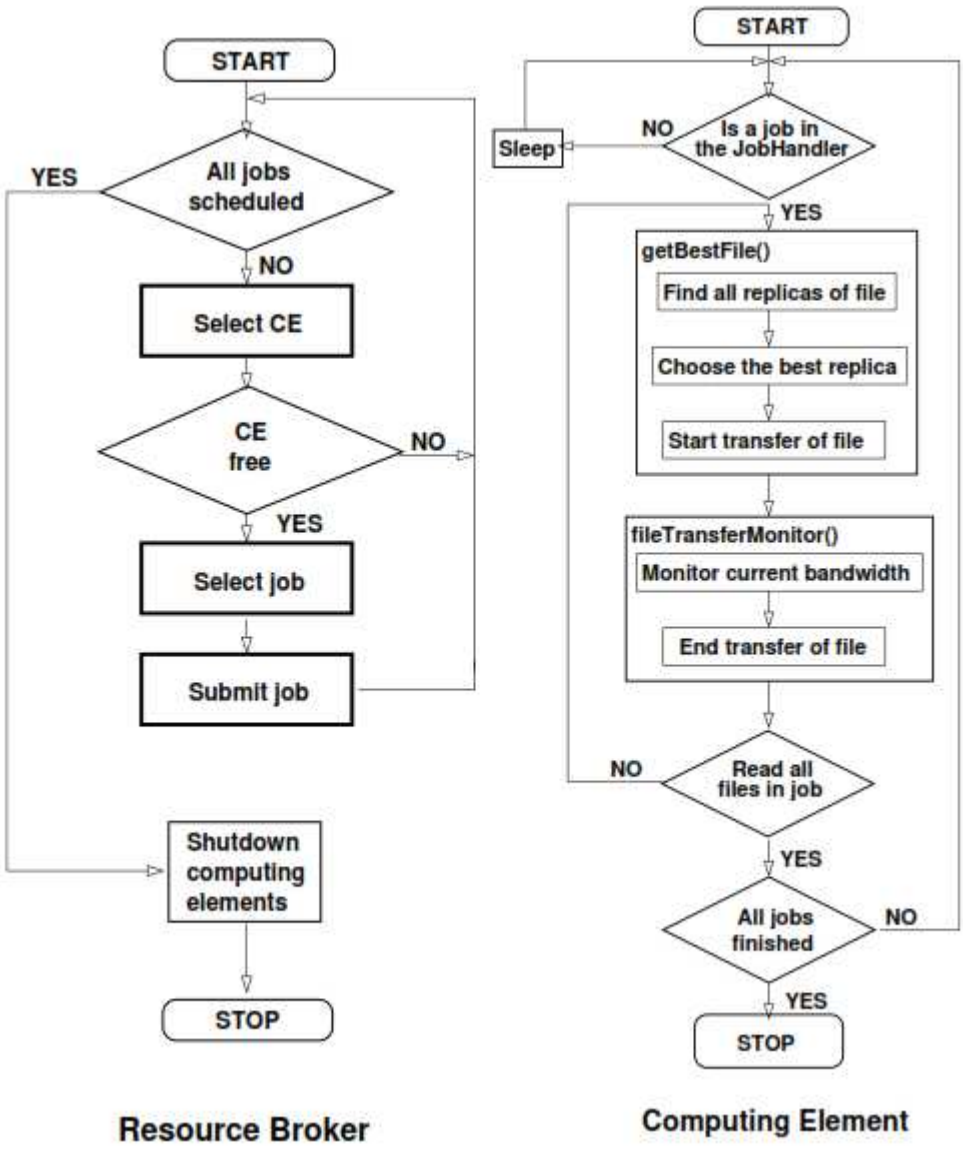
2. La deuxième phase aura lieu quand la sélection d'une réplique dynamique optimale est réalisée pendant le délai d'exécution d'un travail ; dans cette phase la création des répliques peut être déclenchée par l'algorithme d'optimisation.

Dans le projet d'EU DataGrid, le service d'optimisation de grille s'appelle l'optimiseur de réplication (Replica Optimiser) et il est inclus dans un composant appelé le gestionnaire de réplication (Replica Manager) [30].

L'optimiseur de réplication prend des décisions au sujet du transfert de données liées aux travaux entre les sites et la création ou la suppression des répliques.

Dans la simulation chaque élément de calcul est représenté par un amorçage (thread). La soumission de travaux aux éléments de calcul est contrôlée par un autre amorçage appelé : le courtier de ressource (Resource Broker).

Le flux d'exécution de ces amorçages est affiché sur la figure qui suit :



FigV-2 : Exécution du courtier de ressource et des éléments de calcul

Le courtier de ressource s'assure que chaque élément de calcul est continuellement fonctionnel en essayant fréquemment de distribuer les travaux à tous les éléments de calcul.

Quand le courtier de ressource trouve un élément de calcul en veille, il choisit un travail pour qu'il l'exécute selon la politique de l'élément de calcul, c.-à-d. quel type de travaux à exécuter et combien de fois il va exécuter chaque travail.

À tout moment, un élément de calcul exécutera tout au plus un job. Dès que le travail terminé, un autre est assigné par le courtier de ressource.

b- Les outils de simulation :

Le choix des politiques d'ordonnancement pour le courtier de ressource comprennent ce qui suit :

1- L'ordonnanceur aléatoire (Random scheduler) choisit aléatoirement un nœud de traitement pour exécuter un travail spécifique,

2- L'ordonnanceur de la file d'attente la plus courte (Shortest Queue scheduler) calcule la longueur de toutes les files d'attente des nœuds de traitement et choisit un qui a le moindre nombre de travaux qui attendent dans la file d'attente.

3- L'ordonnanceur de coût d'accès (Access Cost scheduler) assigne le travail à l'élément de calcul où le coût d'accès au fichier est le plus bas (coût pour obtenir tous les fichiers nécessaires pour exécuter le travail).

4- L'ordonnanceur de coût d'accès de file d'attente (Queue Access Cost scheduler) choisit l'élément de calcul qui a la plus petite somme du coût d'accès pour le travail et les coûts d'accès de tous les travaux attendant dans la file d'attente.

c- Les entrées (input) de simulation :

Les outils d'OptorSim sont basés sur plusieurs fichiers de configuration :

- Fichier de configuration de paramètre : Les paramètres de simulation de base sont placés dans ce fichier tel que le nombre de travaux total à exécuter, les retards entre chaque soumission de travaux, la taille maximale de la file d'attente, le choix des stratégies de réplique, les modèles d'accès pour le travail, etc.
- Fichier de configuration de grille : contient la topologie du réseau, c.-à-d., les liens entre les sites de grille, la largeur de bande du réseau disponible entre les sites, et le numéro de CEs et de SEs, aussi bien que leurs tailles.
- Fichier de configuration du travail : décrit des informations sur les travaux simulés, les fichiers requis par chaque travail, la probabilité que chaque travail soit exécuté, etc.
- Fichier de configuration de largeur de bande : spécifie le fond du trafic du réseau.

d- Les configurations d'accès :

Comme mentionné ci-dessus, les travaux exige l'accès aux fichiers pendant l'exécution. L'ordre dans lequel ces fichiers sont demandés est déterminé par la configuration d'accès. Quatre configurations importantes d'accès ont été considérées :

- Séquentielle (Sequentiel) : l'ensemble des fichiers est ordonnée, formant une liste de demandes successives,
- Aléatoire (Random) : des fichiers sont choisis aléatoirement à partir d'un positionnement avec une distribution de probabilité,
- Marche aléatoire unitaire (Unitary Random Walk) : l'ensemble est ordonné et les requêtes successives de fichier sont exactement un élément à partir de la demande précédente de fichier, la direction est aléatoire,
- Marche aléatoire gaussienne (Gaussian Random Walk) : comme avec la marche aléatoire unitaire, mais les fichiers sont choisis par une distribution gaussienne portée sur la demande précédente de fichier.

e- Les algorithmes d'optimisation :

Les algorithmes d'optimisation de réplication sont le noyau de l'optimiseur de réplication. La stratégie employée pour répliquer et quel fichier devrait être effacé, différencie les algorithmes d'optimisation. Nous présentons brièvement les algorithmes simples qui ont été mis en application dans OptorSim.

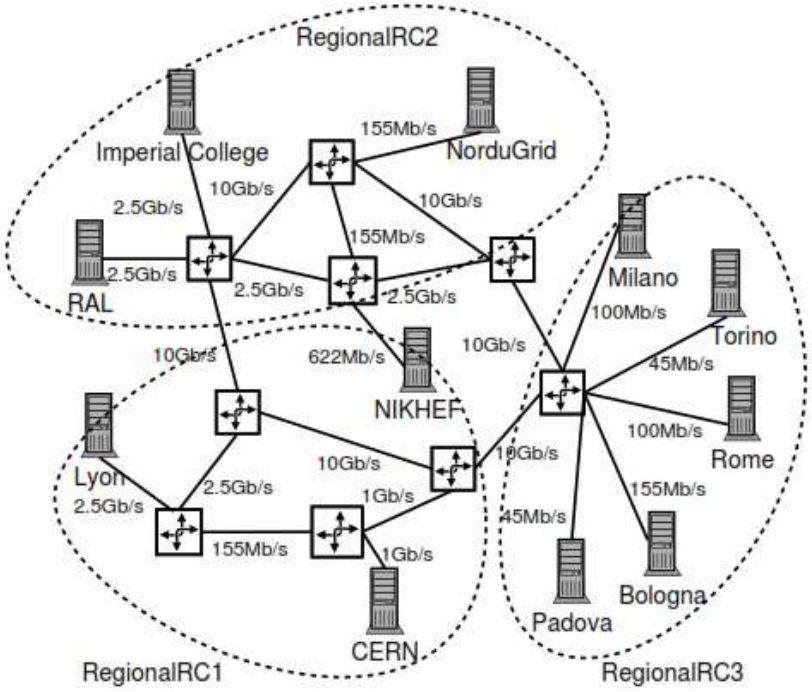
- Aucune réplique :
  Cet algorithme ne réplique jamais un fichier. La distribution des réplications initiales de fichier est décidée au début de la simulation et ne change pas pendant son exécution. Cet algorithme renvoie le nom du fichier avec le temps d'accès le plus rapide.
- Least Recently Used (LRU) : lorsqu'un job a besoin d'une donnée qui n'existe pas sur le site sur lequel il s'exécute, on la réplique automatiquement depuis le site où elle se trouve. En cas de manque d'espace de stockage pour l'héberger, il supprime la réplique la moins récemment utilisée.
- Least Frequently Used (LFU) : lorsqu'un job a besoin d'une donnée qui n'existe pas sur le site sur lequel il s'exécute, on la réplique automatiquement depuis le site où elle se trouve. En cas de manque d'espace de stockage pour l'héberger, il supprime la réplique la moins fréquemment utilisée.

- Le modèle économique : lorsqu'un job a besoin d'une donnée qui n'existe pas sur le site sur lequel il s'exécute, on estime deux coûts :
    - Le cout d'une exécution à distance du job. L'exécution se fait sur le site contenant la donnée puis transfert des résultats au site lançant le job.
    - le coût d'une exécution locale après réplication de la donnée sur le site lançant le job.

Le modèle économique choisit la solution qui donne le meilleur coût. Sur OptorSim, il en existe deux implémentations de ce modèle : Zipf-based economic model (utilisant une fonction de prédiction Zipf) et EconomicBinomial (basé sur le modèle binomial).

V-3- Configuration de la simulation :

a- Topologie de la grille [31] :



FigV-3 : Topologie simulée d'EU DataGrid Testbed1

Comme cité précédemment, il existe plusieurs types grilles de données. Bien que notre étude porte sur l'utilisation d'OptorSim pour modeler et simuler des grilles hiérarchiques de données (avec l'EU DataGrid comme exemple).

Nous avons expérimenté notre stratégie sur le banc d'essai 1[32] d'EU DataGrid. La topologie du réseau du banc d'essai est affichée sur le figure V-3. Dans l'expérience de LHC (Large Hadron Collider), pour laquelle l'UE DataGrid a été construite, la majeure partie des données est en lecture seule. En outre, l'atomicité des fichiers est assurée, c.-à-d. un fichier est une unité non-fractionnée d'information, pour simuler les données brutes déjà traitées de l'expérience de LHC.

b- Configuration de la grille :

L'étude des algorithmes d'optimisation a été réalisée en utilisant un modèle des sites du banc d'essai d'EU Data Grid. Dans ce modèle, chaque site lui est attribué des ressources de stockage proportionnelles à ces allocations réelles de matériel. Chaque site de banc d'essai, à l'exclusion de CERN, a été assigné un élément de calcul et de stockage. CERN a été assigné un élément de stockage pour retenir tous les fichiers maîtres mais n'a pas été assigné un élément de calcul. Des routeurs ont été décrits en étant un site sans éléments de calcul ou de stockage. La taille des éléments de mémoire pour chaque site de banc d'essai sont indiquées dans le tableau 1 (décrit dans le fichier « edg_testbed_grid.conf») :

Site Name	Bologna	Catania	CERN	Imperial College	Lyon
Size (GB)	30	30	10000	80	50

Site Name	Milano	NIKHEF	NorduGrid	Padova	RAL	Torino
Size (GB)	50	70	63	50	50	50

Tableau 1 : Liste des ressources de Stockage assignées aux sites d'EU Datagrid testbed 1, desquelles les résultats ont été produits. La taille des éléments de mémoire sont en giga-octets.

c- Configuration des travaux (jobs) : (décrit dans le fichier « edg_testbed_jobs.conf »)

Au commencement, tous les fichiers ont été placés dans l'élément de stockage de CERN. Il y avait six types de tâches. On a supposé que chaque ensemble de fichiers se compose de 1GByte fichiers. Il y aura une certaine distribution des travaux que chaque site effectue.

Data Sample	Total Size (GB)
Central J/ψ	10
High p_t leptons	2
Inclusive electrons	50
Inclusive muons	14
High E_t photons	58
$Z^0 \rightarrow b\overline{b}$	6

Tableau 2 : Tailles prévues des ensembles de données.

V-4- Implémentation :

a- Algorithme de l'approche proposée :

Le principe du LRU est simple. Lorsqu'un job a besoin d'une donnée qui n'existe pas sur le site sur lequel il s'exécute, on la réplique automatiquement depuis le site où elle se trouve. En cas de manque d'espace de stockage pour l'héberger, il supprime la réplique la moins récemment utilisée. De même pour notre algorithme qui a pour nom « Modified LRU » (LRU modifié) fournit une implémentation de quelques méthodes de la classe «Modified Lru Optimiser ». Durant l'exécution des jobs, LRU crée une réplique à chaque fois qu'un fichier est demandé. A la différence de notre approche, la création des répliques est optimisée de façon à chercher le meilleur élément de stockage (CloseSE), tester sa validité aux autres fichiers stockés au sein de cet élément, et puis le répliquer et s'assurer que la réplique est fiable.

Algorithme :

Chercher le CloseSE ;

Si trouver alors SE = CloseSE ;

Si le fichier recherché se trouve sur SE alors continue ;

Copie la réplique ;

Si espace insuffisant sur SE alors continue ;

Faire

Trouver l'espace sur SE en essayant de supprimer quelques fichiers ;

Si réplication marche alors stocker et sortir

Sinon chercher fichier dépensable

Si non-trouver sortir ;

Tant que fichier répliqué égale à nul ;

Tester la validité du fichier répliqué par rapport au fichier à supprimer ;

Choisir les fichiers à supprimer parmi les fichiers déjà sélectionnés par l'algorithme LRU.

b- Résultats de simulation :

Notre approche a été comparée avec les stratégies implémentées sur OptorSim, qui sont :

1- Simple,
2- LRU,
3- LFU,
4- EconomicBinomial,
5- Zipf-based economic model.

Nous avons simulé notre approche (Modified LRU (6)) avec les différentes configurations d'accès (Sequentiel, Random, Unitary Random Walk, Gaussian Random Walk, Random Zipf), en choisissant différentes politiques d'ordonnancement (1-Random scheduler, 2-Shortest Queue scheduler, 3-Access Cost scheduler, 4-Queue Access Cost scheduler), et en introduisant différents nombres de travaux (jobs).

La configuration d'accès choisie dans notre simulation est : séquentielle (Sequentiel) et la politique d'ordonnancement maintenue est : Queue Access Cost scheduler.

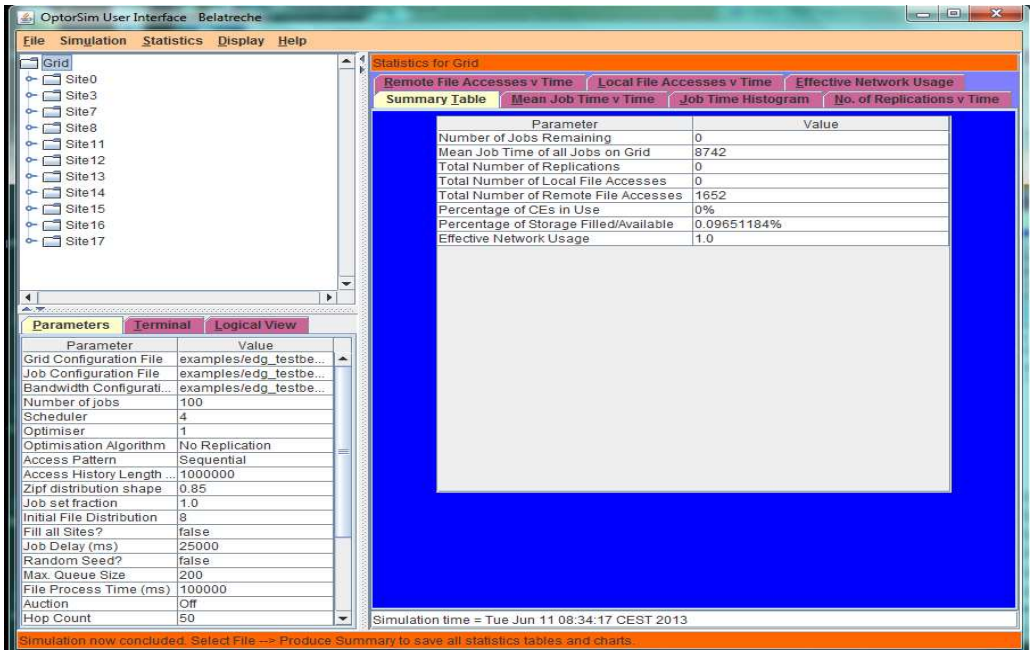
Les paramètres de simulation comparés sont :

- le nombre de répliques : ce paramètre est très important dans la simulation (moins de répliques entraîne moins d'espace de stockage et donc moins de problèmes de mises à jours).

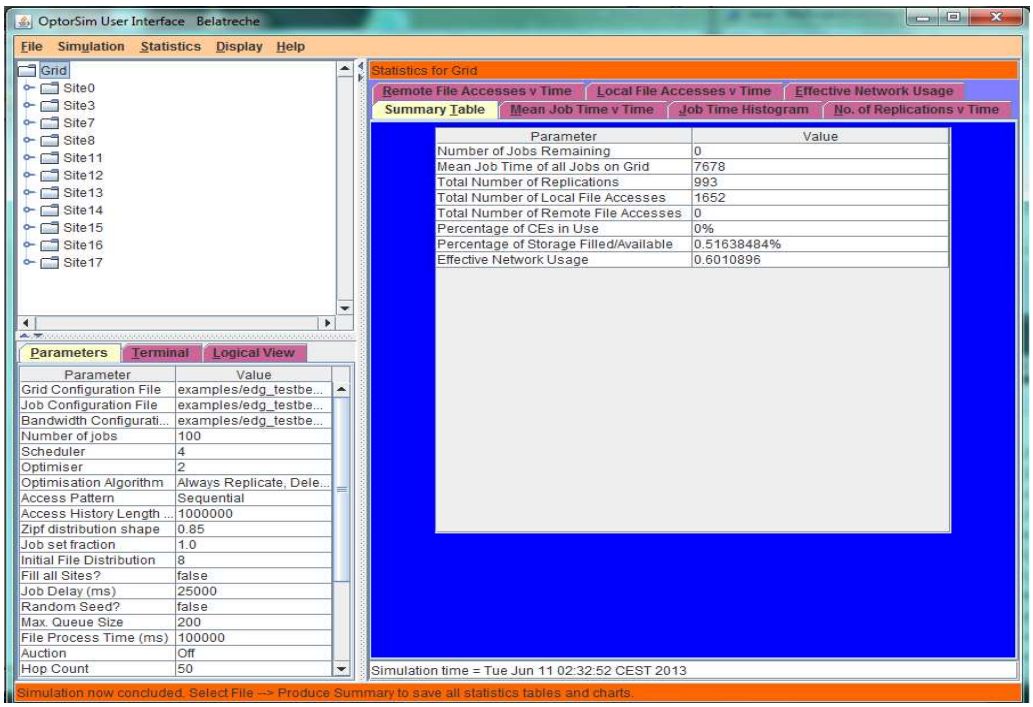
- le temps d'exécution des Jobs : le premier but de toutes les stratégies est de réduire le temps d'exécution.

En voici les interfaces des simulations :

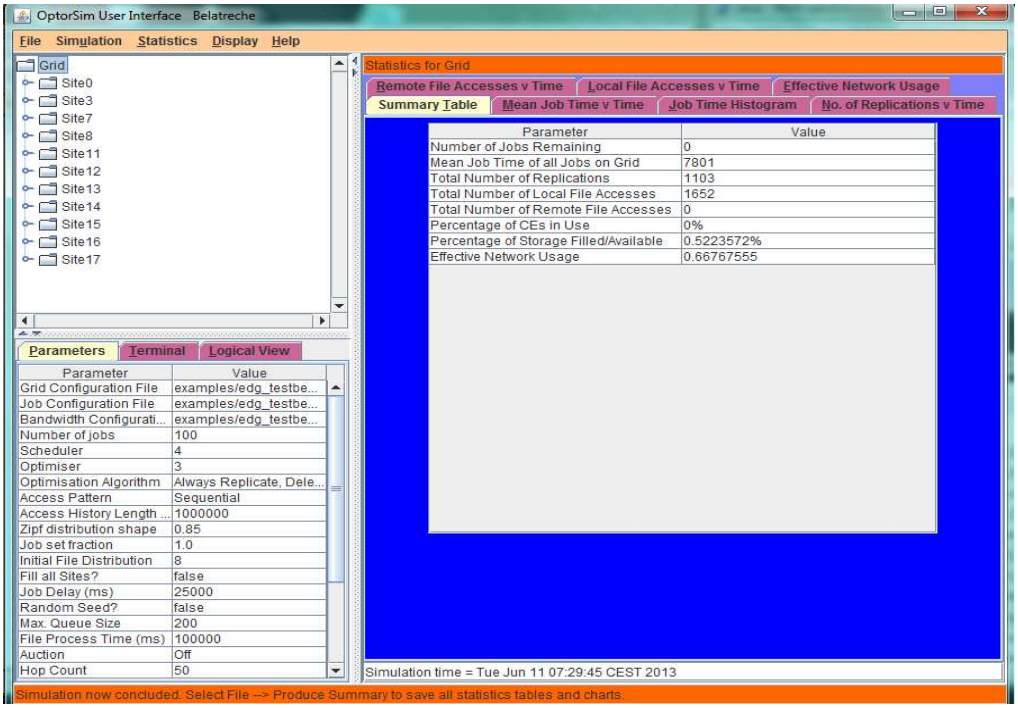
1- Simulation de 100 jobs :



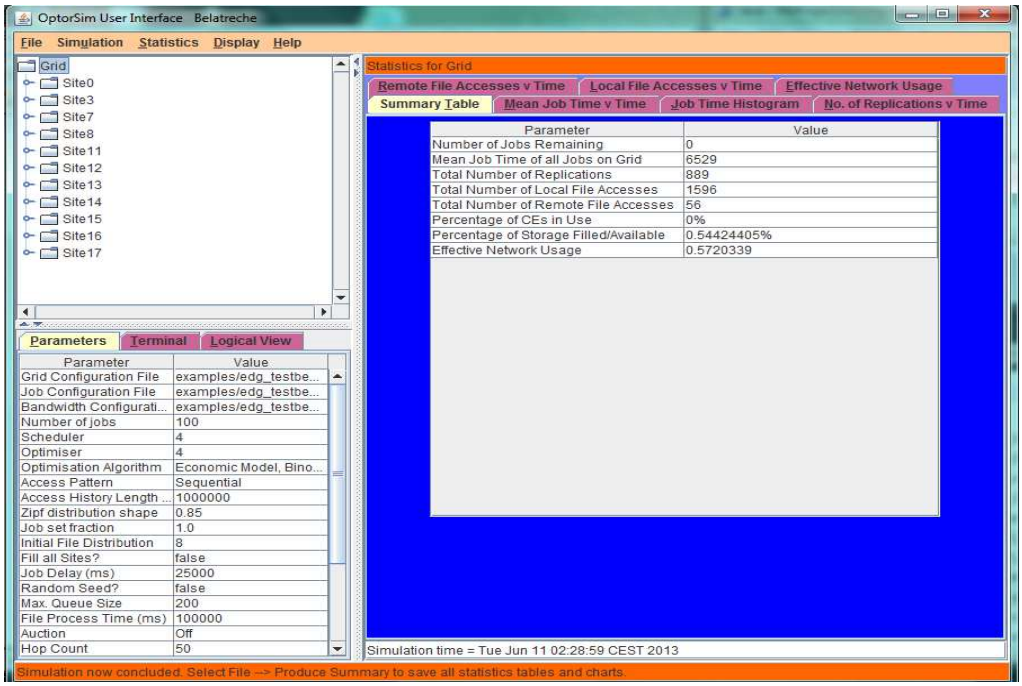
FigV-4 : Simulation de l'approche Simple (1)



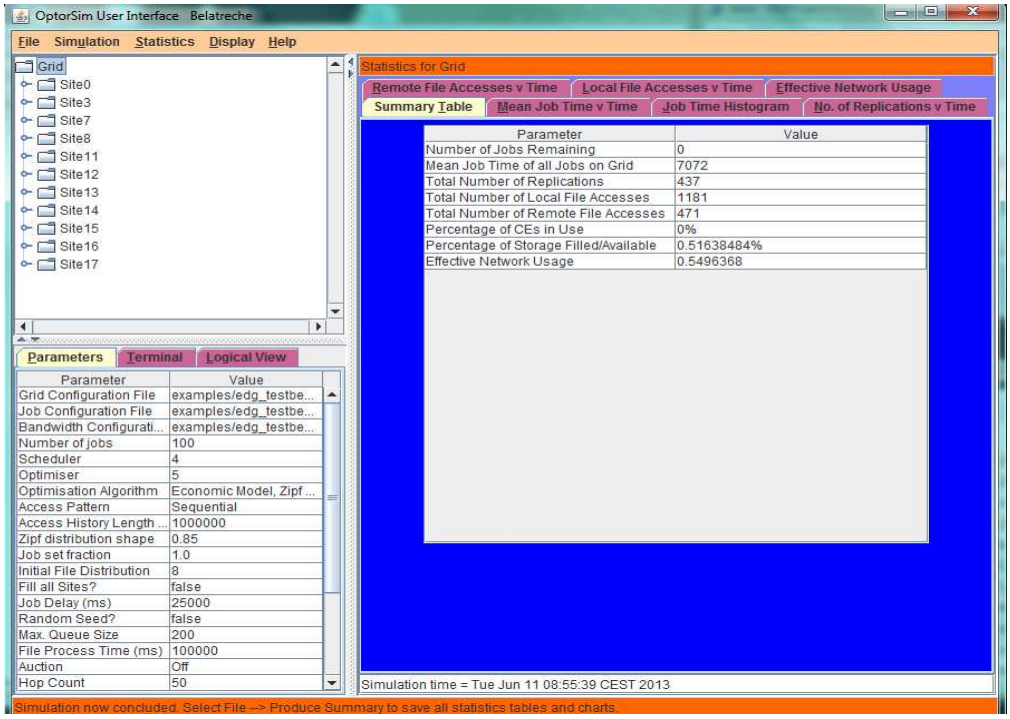
FigV-5 : Simulation de l'approche LRU (2)



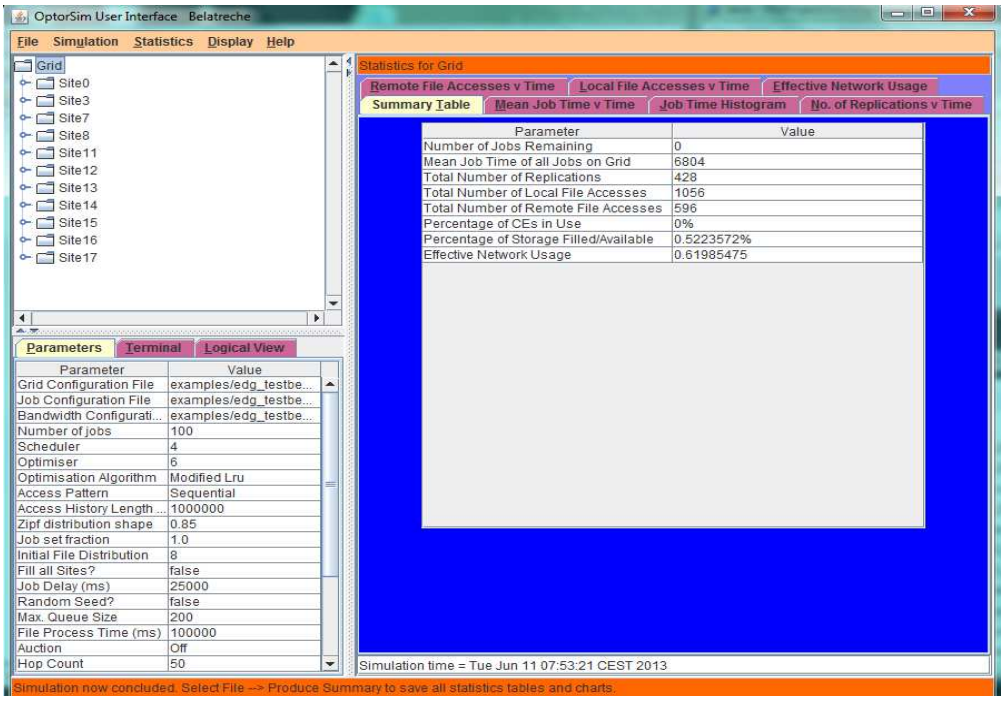
FigV-6 : Simulation de l'approche LFU (3)



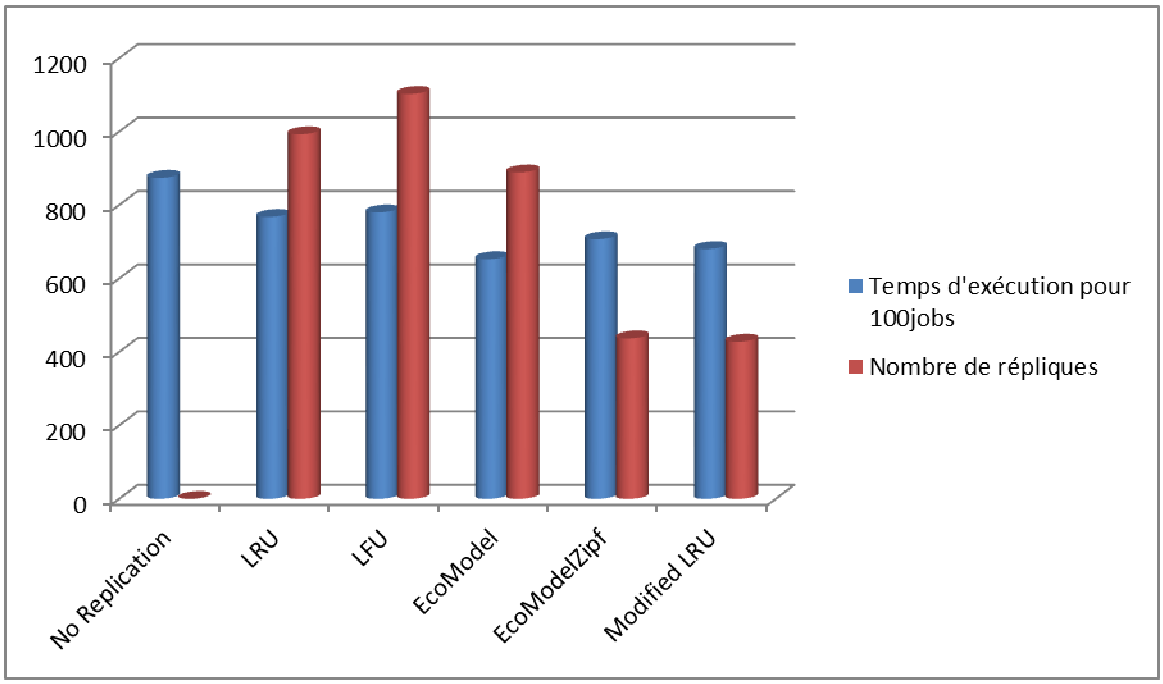
FigV-7 : Simulation de l'approche EconomicBinomial (4)



FigV-8 : Simulation de l'approche Zipf-based economic model (5)

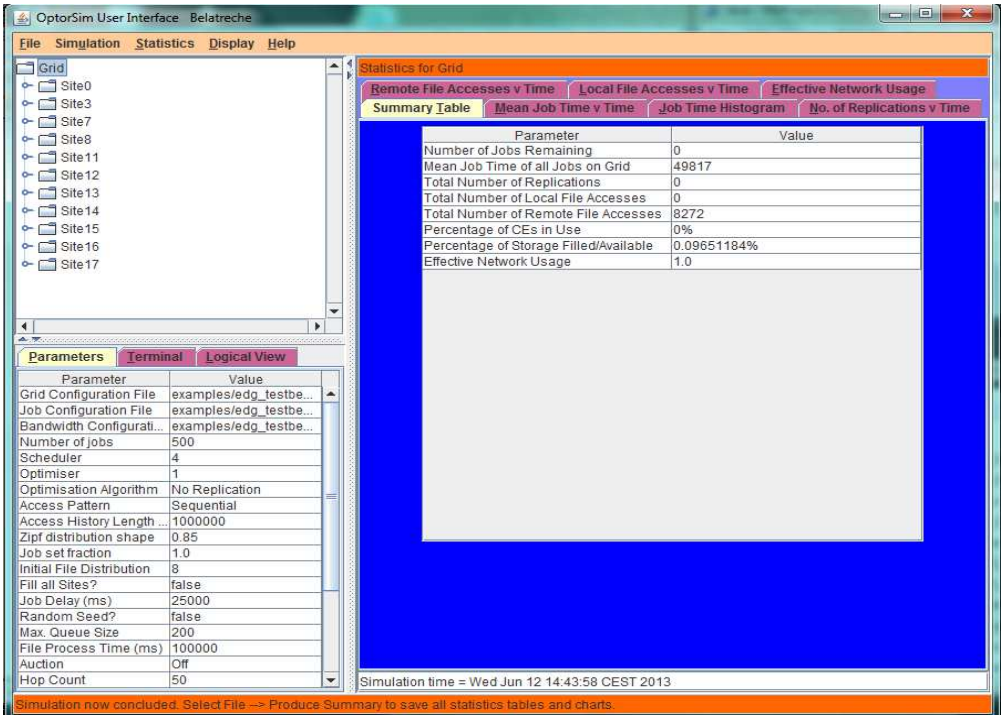


FigV-9 : Simulation de l'approche Modified LRU (6)

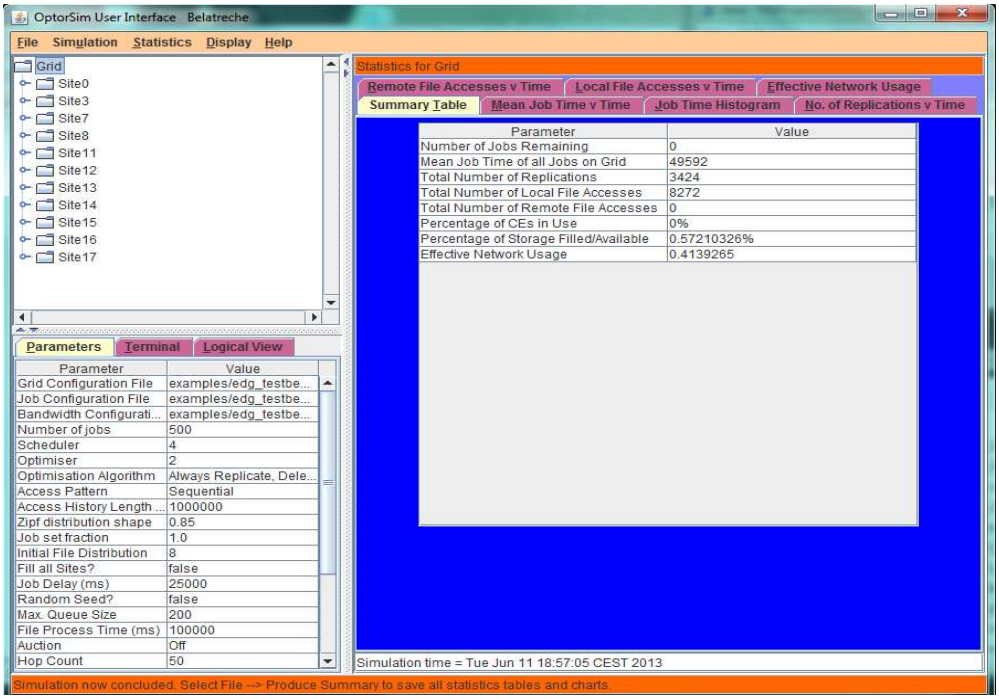


FigV-10 : Récapitulation des scénarios de simulation de 100 jobs

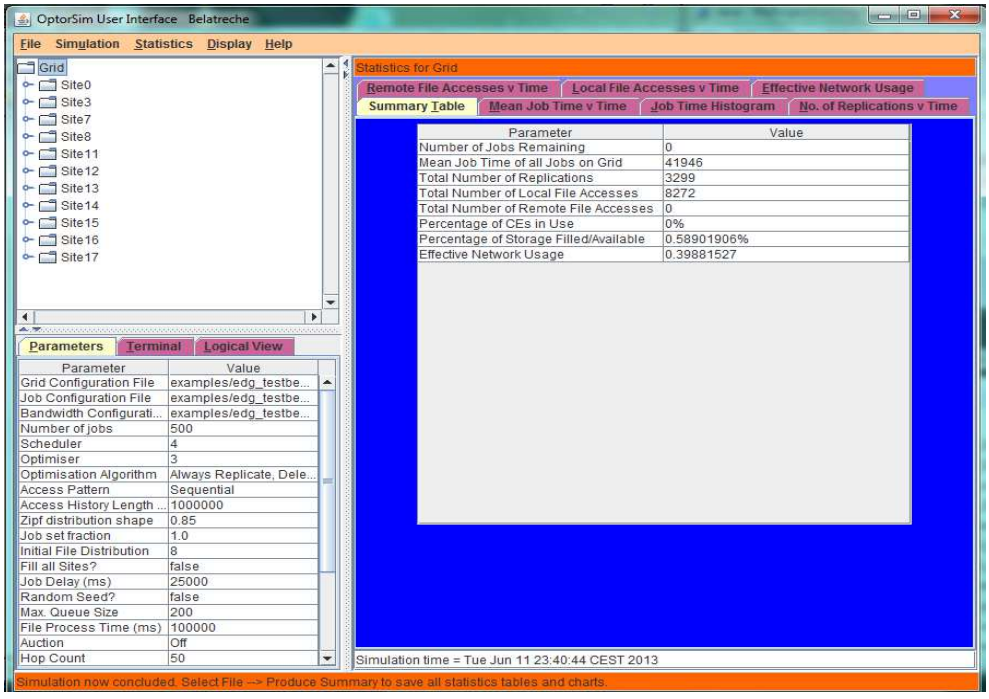
2- Simulation de 500 jobs :



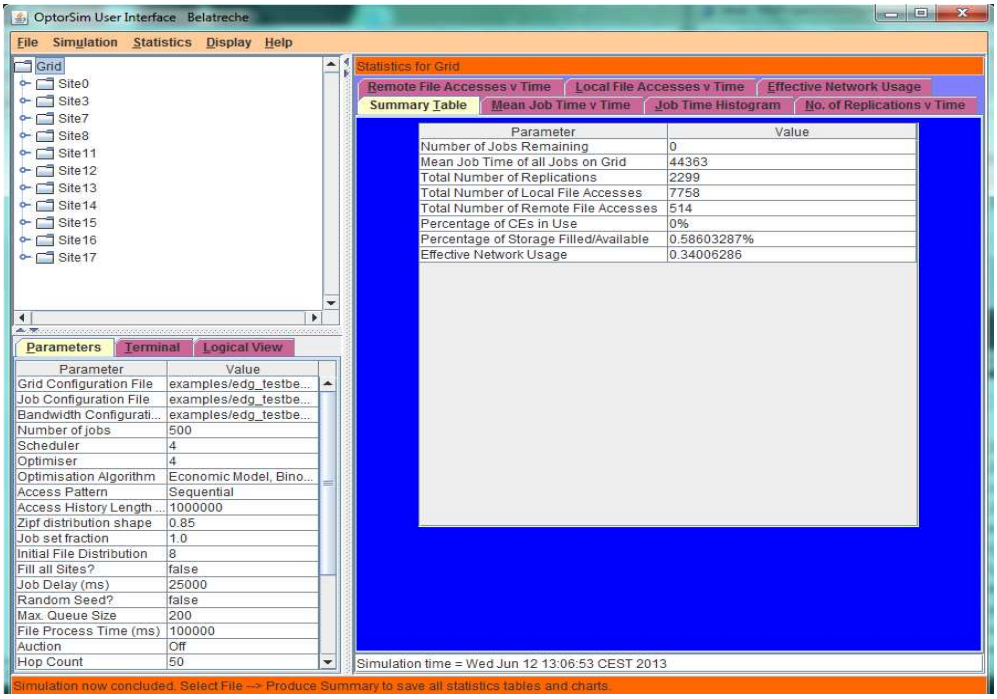
FigV-11 : Simulation de l'approche Simple (1)



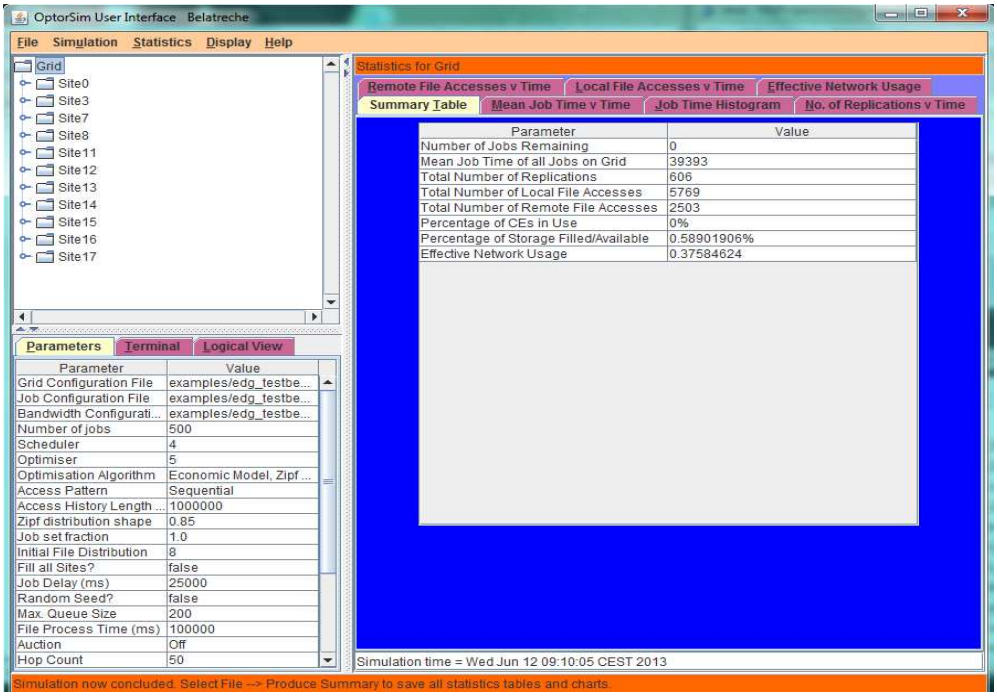
FigV-12 : Simulation de l'approche LRU (2)



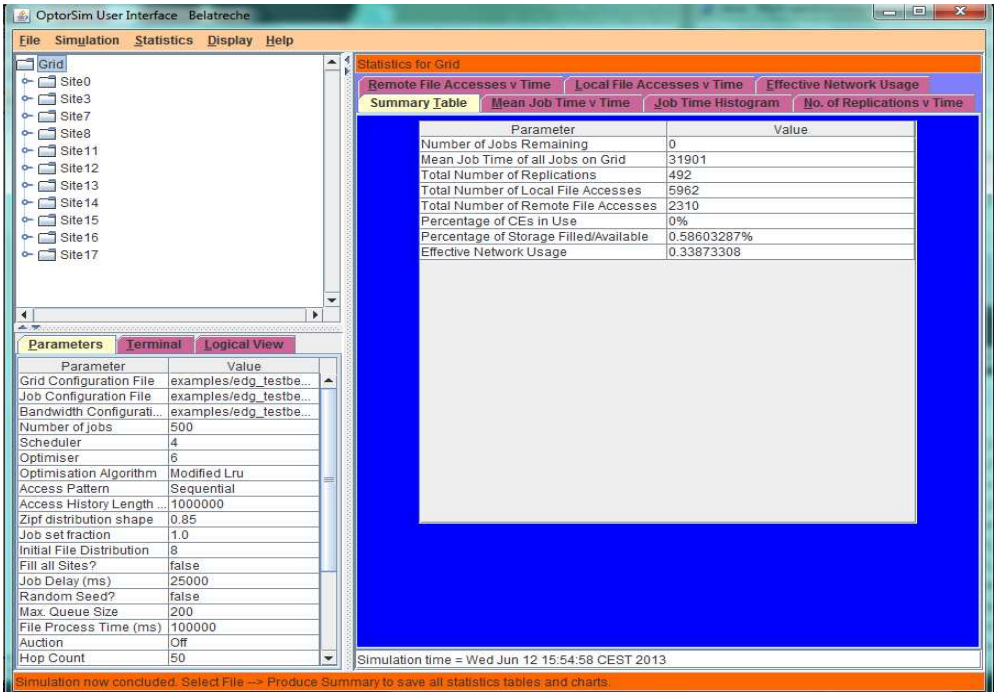
FigV-13 : Simulation de l'approche LFU (3)



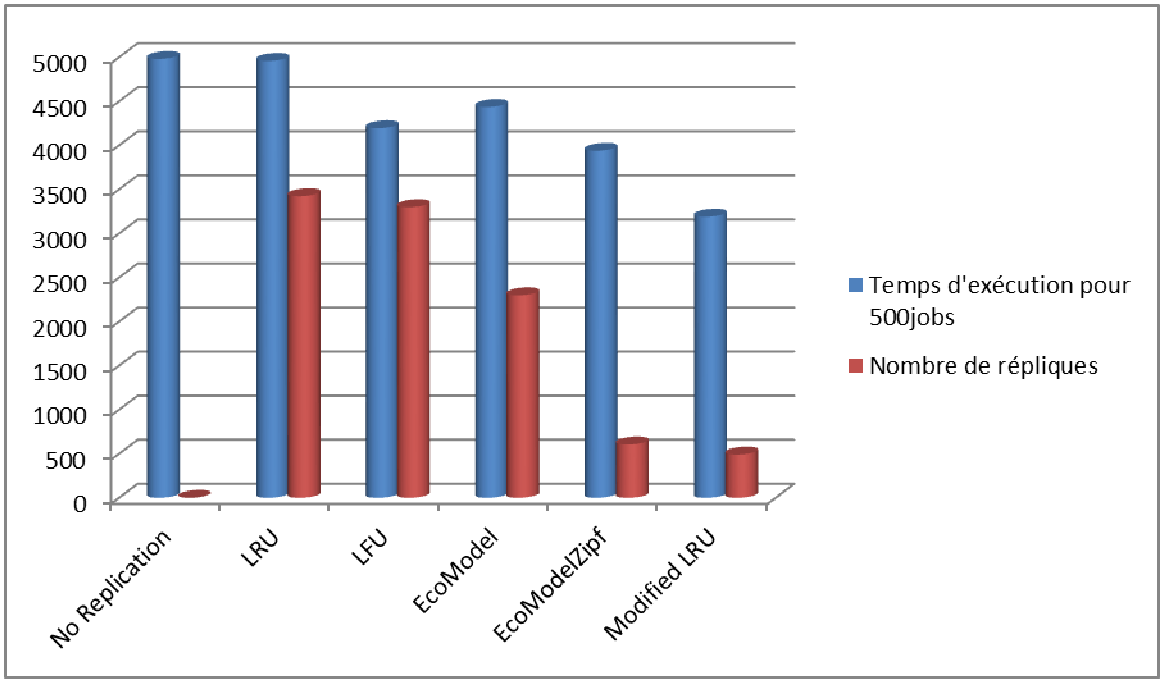
FigV-14 : Simulation de l'approche EconomicBinomial (4)



FigV-15 : Simulation de l'approche Zipf-based economic model (5)

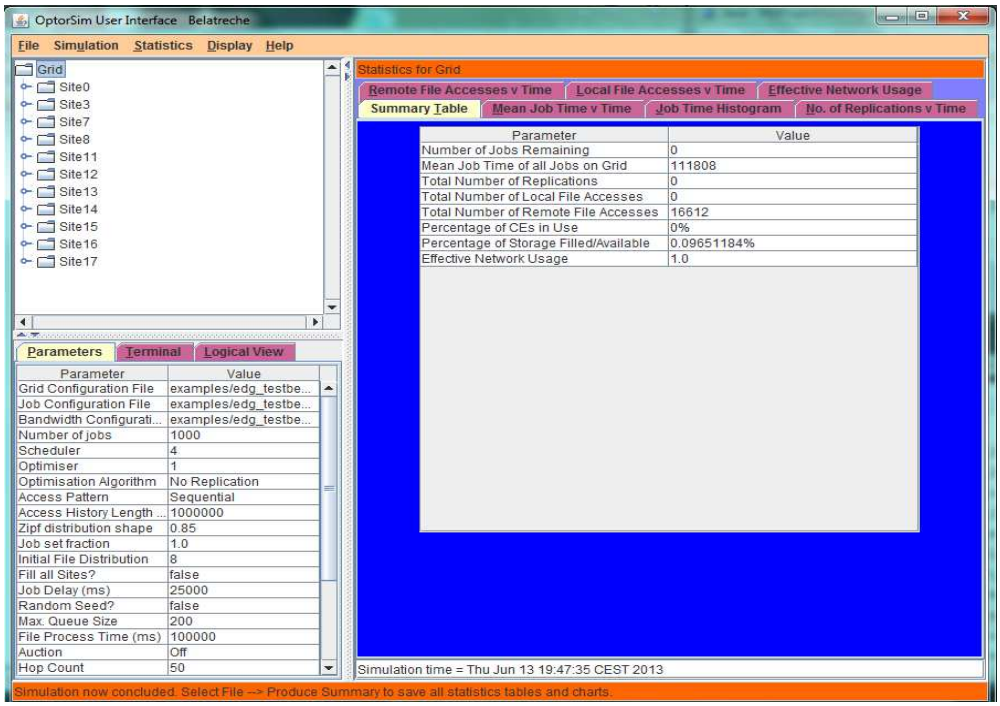


FigV-16 : Simulation de l'approche Modified LRU (6)

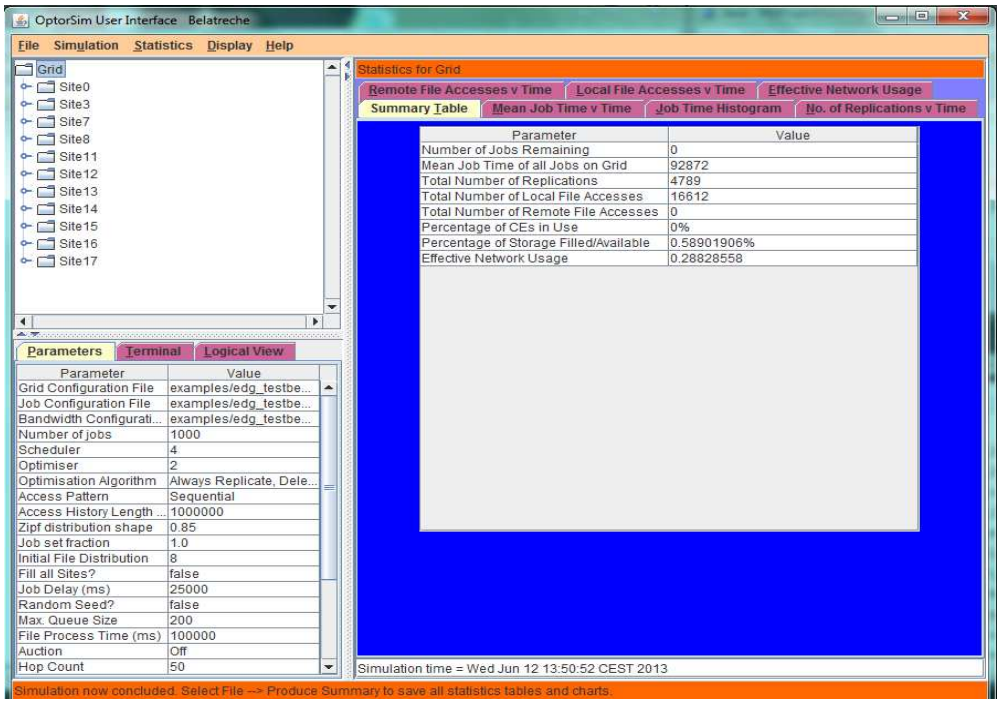


FigV-17 : Récapitulation des scénarios de simulation de 500 jobs

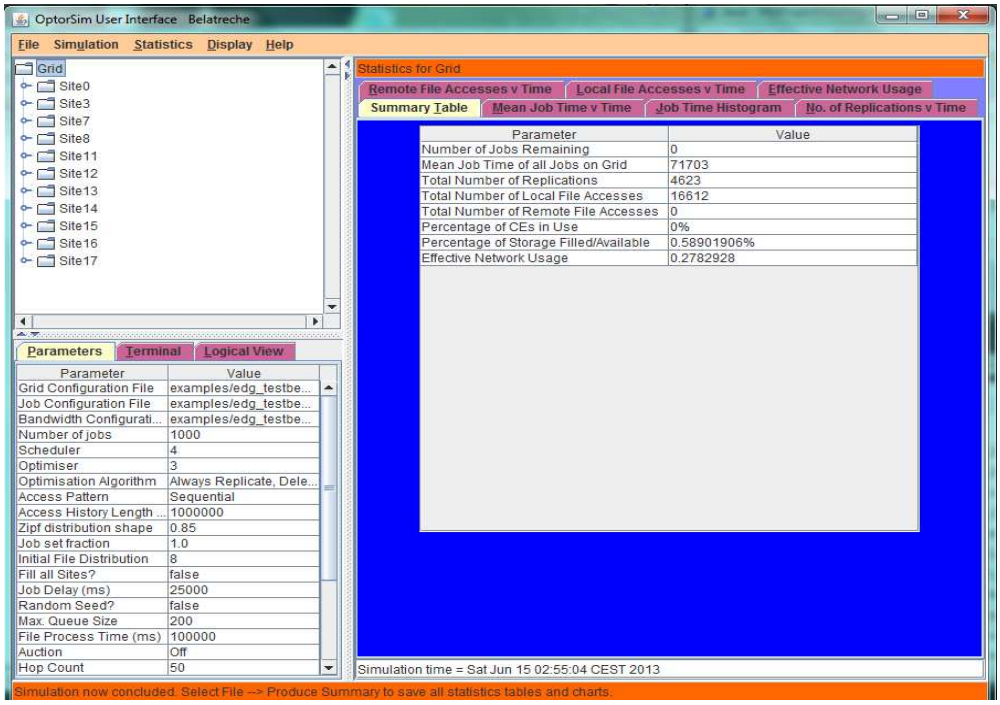
3- Simulation de 1000 jobs :



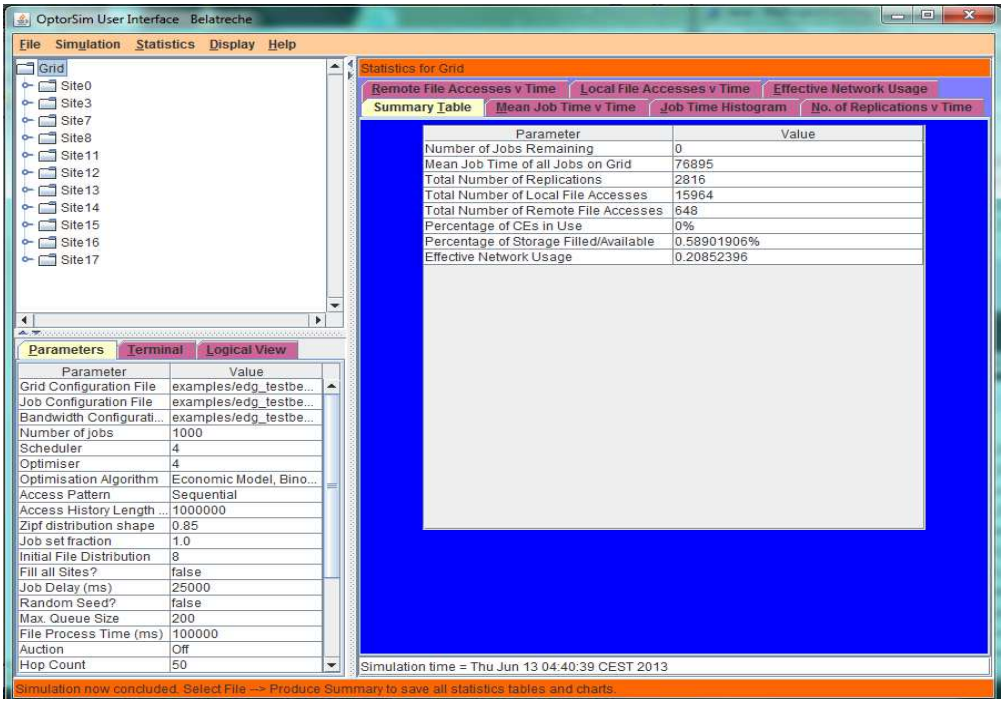
FigV-18 : Simulation de l'approche Simple (1)



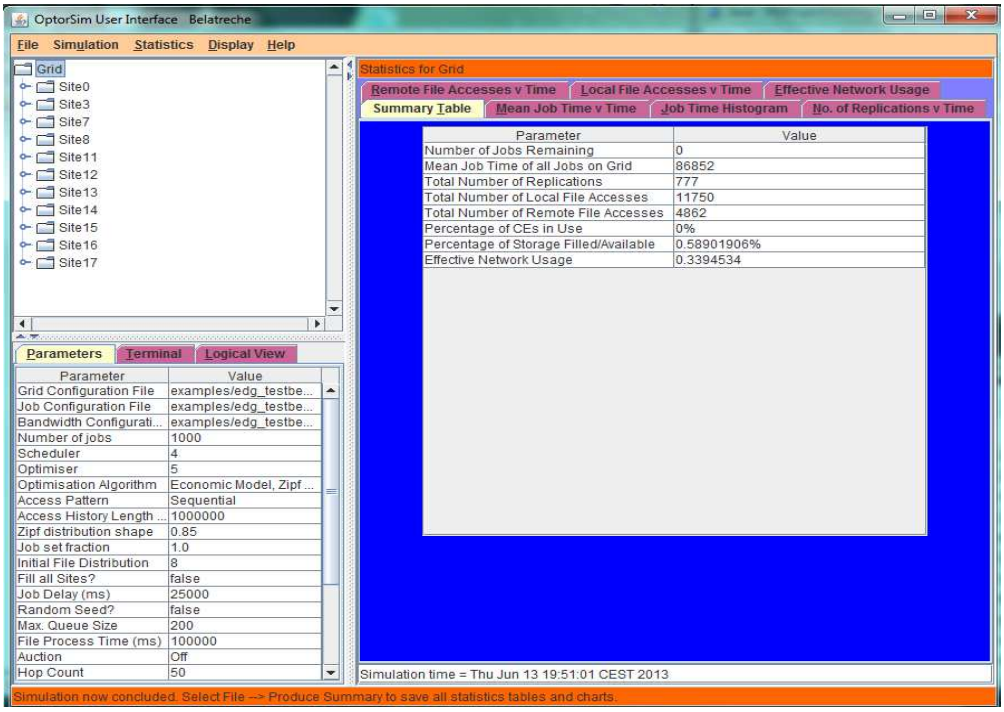
FigV-19 : Simulation de l'approche LRU (2)



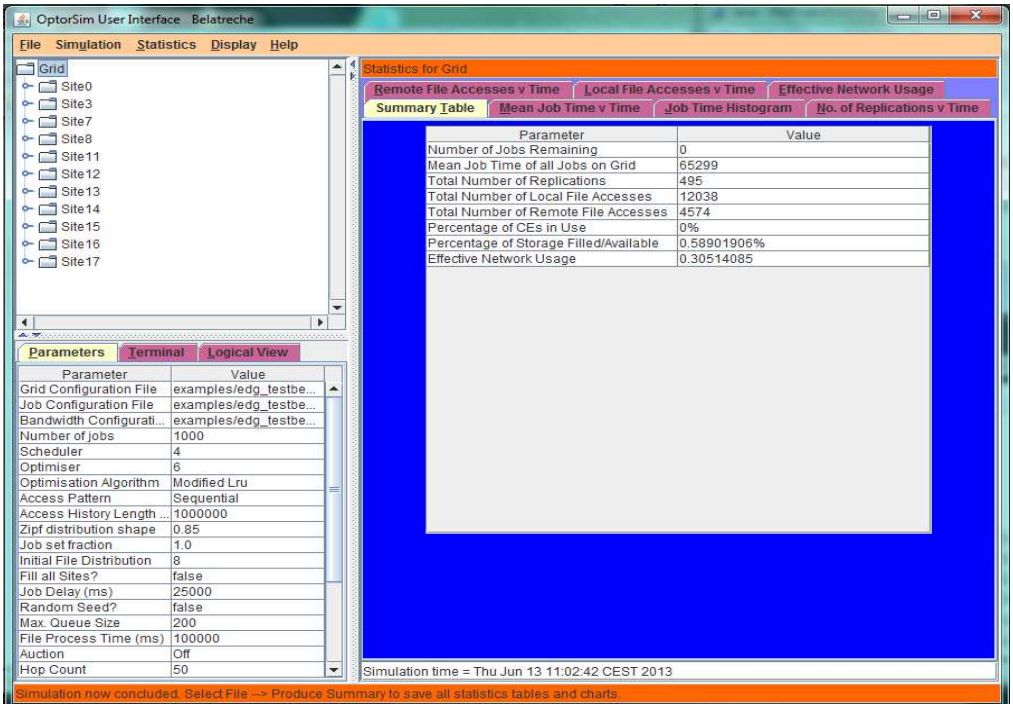
FigV-20 : Simulation de l'approche LFU (3)



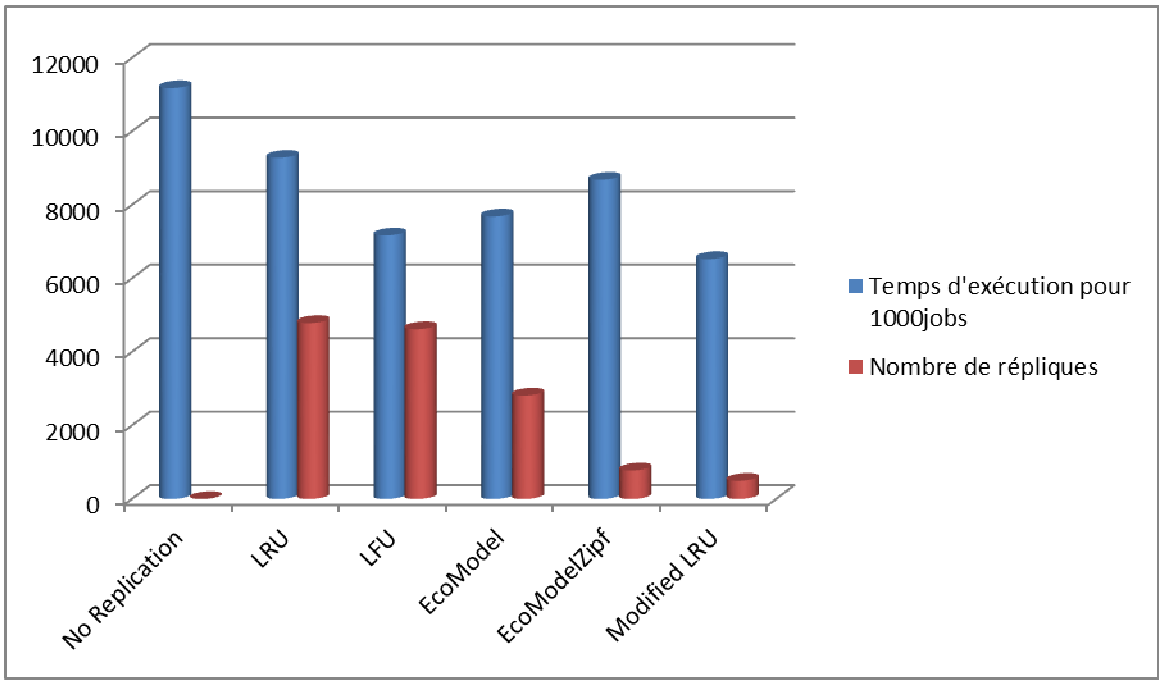
FigV-21 : Simulation de l'approche EconomicBinomial (4)



FigV-22 : Simulation de l'approche Zipf-based economic model (5)

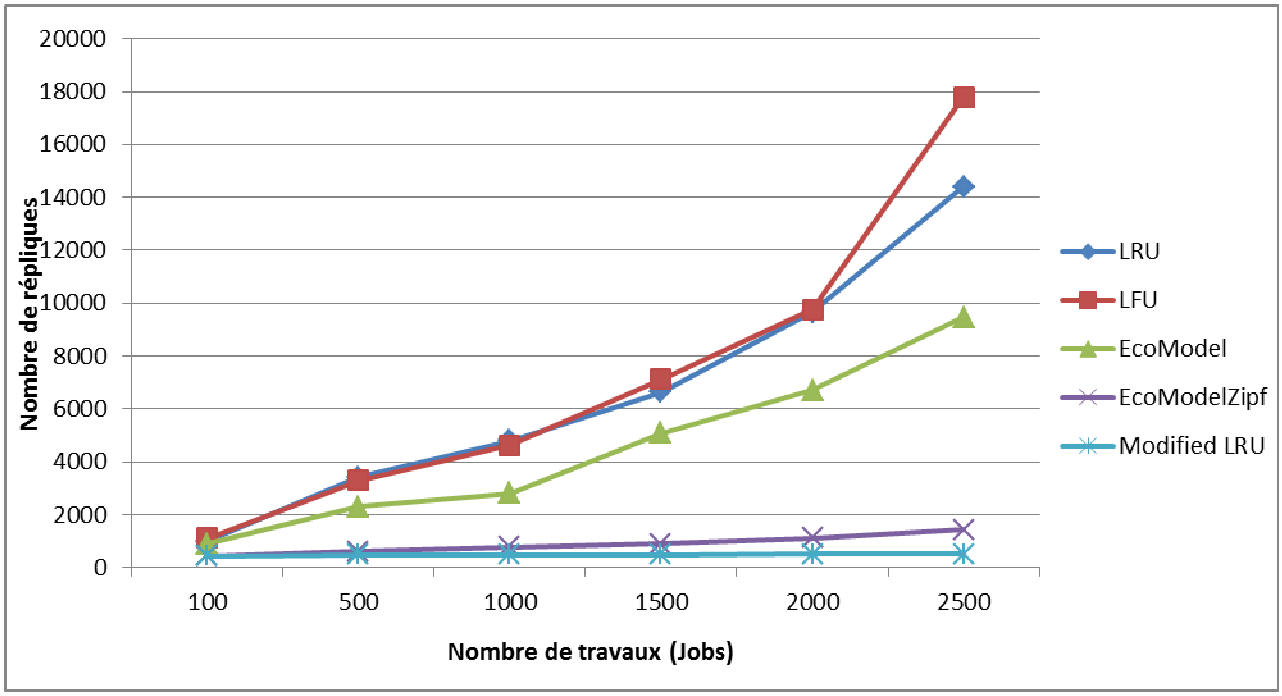


FigV-23 : Simulation de l'approche Modified LRU (6)

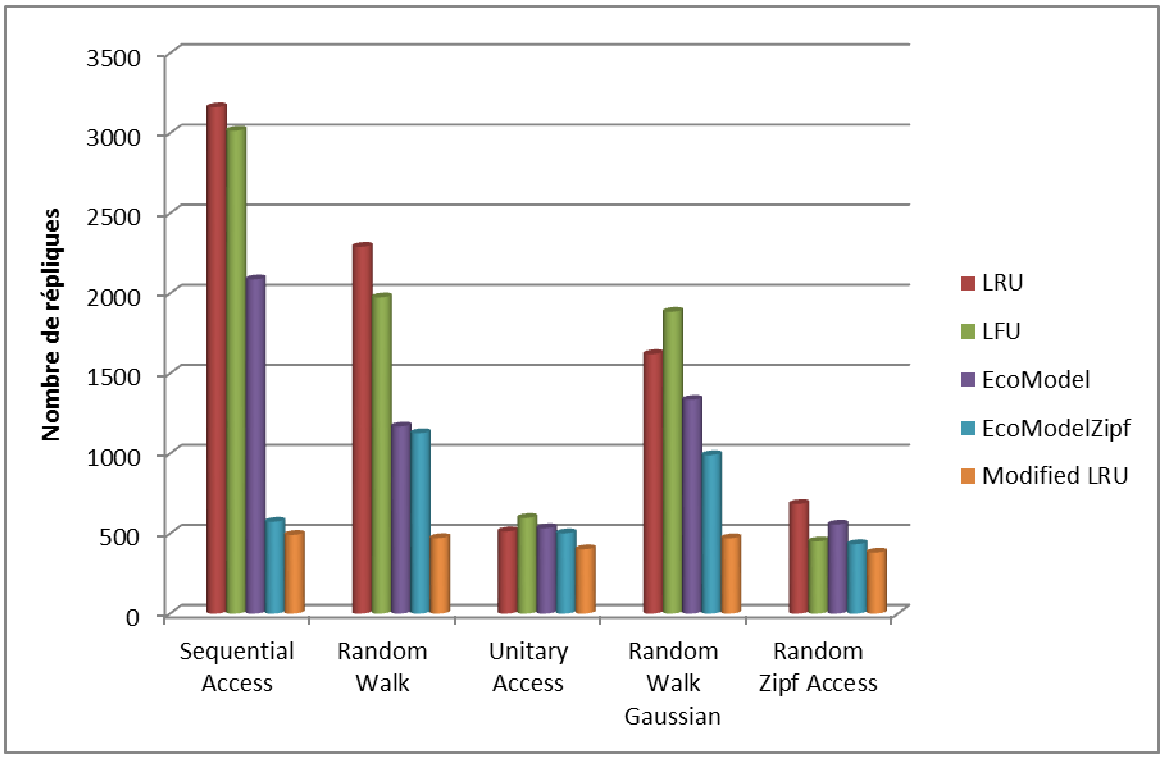


FigV-24 : Récapitulation des scénarios de simulation de 1000 jobs

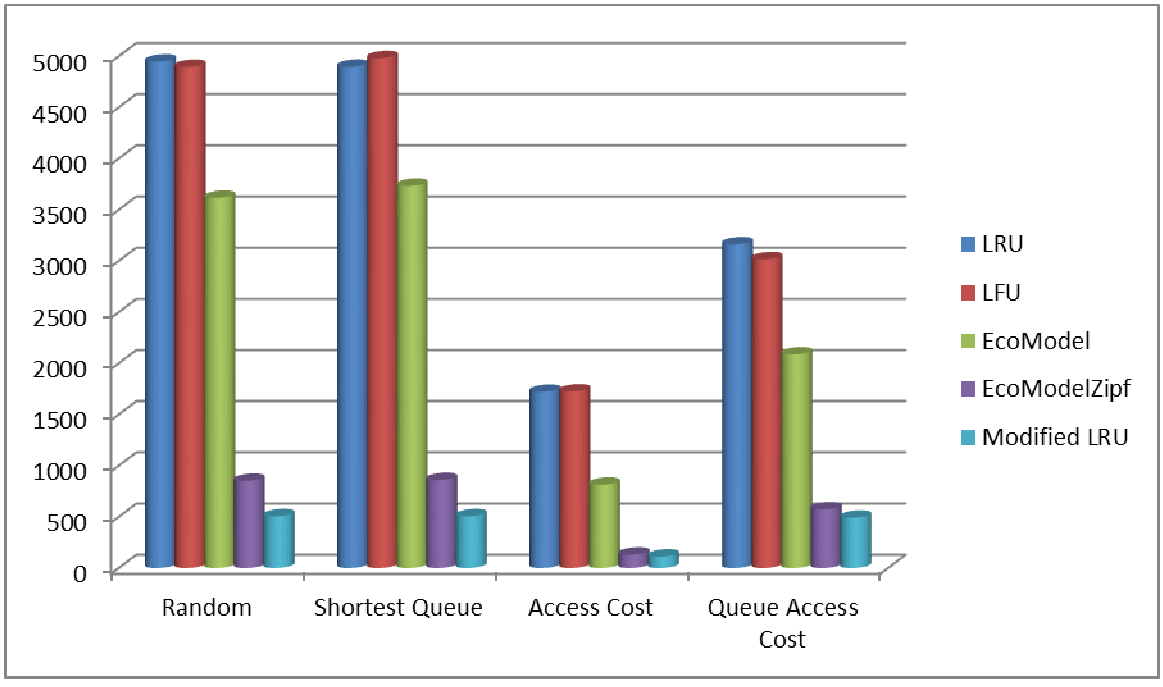
4- D'autres simulations :



FigV-25 : Nombre de répliques basé sur la variation du nombre de travaux (jobs)



FigV-26 : Nombre de répliques basé sur les différentes configurations d'accès (nombre de travaux=500, l'ordonnanceur de coût d'accès de file d'attente)



FigV-27 : Nombre de répliques basé sur les différentes politiques d'ordonnancement (nombre de travaux=500, configuration d'accès séquentielle)

c- Discussion des résultats :

Selon les résultats de simulation, on remarque clairement que notre stratégie (Modified LRU) réduit considérablement le nombre de répliques par rapport à toutes les autres stratégies dans tous les cas de configurations d'accès et en utilisant toutes les politiques d'ordonnancement, ainsi que le temps d'exécution des travaux (jobs).

V-5- Conclusion :

Dans ce chapitre nous avons décrit dans les détails notre simulateur qui est consacré à l'évaluation des différentes stratégies de placement des répliques dans les grilles de données. Plusieurs tests ont été effectués selon plusieurs paramètres de configuration. Les résultats obtenus des différentes expérimentations, ont démontré que les temps de réponses et surtout le nombre de répliques obtenus par l'approche de « Modified LRU » sont nettement meilleurs que les résultats obtenus par les autres approches.

Conclusion générale

Ce mémoire a pour objectif l'étude du problème de gestion de répliques dans les grilles de données, notamment, le problème du placement de répliques. Pour cette étude, nous avons consacré une partie de notre travail à donner un bref aperçu sur les grilles informatiques. Puis, nous avons étudié le processus de réplication. Ensuite, nous avons présenté un état de l'art, dans lequel, plusieurs stratégies de placement de répliques sont étudiées. Dans ce mémoire, nous avons proposé une approche de placement de répliques qui optimise le nombre de répliques et le temps d'exécution dans une grille de données à topologie hiérarchique. Cette approche a été comparée avec d'autres stratégies implémentées sur le simulateur spécifique à notre étude. Le travail que nous avons réalisé dans ce mémoire a montré que la façon de répliquer les fichiers et de supprimer d'autres dépend de plusieurs critères tels que : le choix du meilleur élément de stockage, le meilleur fichier et le plus valable. La stratégie que nous avons proposée a amélioré énormément les performances de la grille vue les résultats prometteurs obtenus lors de la simulation.

Comme perspectives, il est très intéressant d'améliorer le temps d'exécution des jobs de notre approche et la comparer avec les autres stratégies proposées jusqu'à maintenant.

BIBLIOGRAPHIE :

[1] M. Meddeber. *Algorithme d'équilibrage de charge distribuée pour les grilles de calcul*. Master's thesis, Département d'informatique, Faculté des sciences, Centre universitaire de Mascara-Mustapha Estambouli, Algérie, Septembre 2008.

[2] I. Foster and C. Kesselman, eds., *The Grid : Blueprint for a Future Computing Infrastructure,* Morgan Kaufmann, San Francisco, 1999.

[3] A. Vernois, *Ordonnancement et réplication de données bio-informatiques dans un contexte de grille de calcul*. Thèse PHD Octobre 2006.

[4] I. Foster and C. Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers Inc., 2003.

[5] I. Foster. *What is the grid ? a three point checklis*t. June 2002.

[6] B. Yagoubi. *Equilibrage de charge dans les grilles de calcul*. PhD thesis, Département d'informatique, Université d'Oran, Algérie, Novembre 2007.

[7] Henry Lin (2005) *Economy Based Data Replication Broker Policies in Data Grids.*

[8] Jamal Eddine Ghaffour. *Sécurité dans les grilles de calcul*, 2004.

[9] M. Soberman. *Les grilles informatiques - état de l'art et déploiement,* CNRS / STIC, 2005.

[10] Cantone Laurent et Unal Résoul, *Le Grid Computing et son utilisation dans les entreprises et les industries*. 2005

[11] G. Belalem. *Contribution à la gestion de la cohérence de répliques de fichiers dans les systèmes à large échelle*. PhD thesis, Département d'informatique, Faculté des sciences, Université d'Oran, Algérie, Novembre 2007.

[12] Ming Tang, Bu-Sung Lee, Chai-Kiat Yeo, Xueyan Tang, *Dynamic replication algorithms for the multi-tier Data Grid*, 0167-739X/$ – see front matter 2004 Elsevier B.V. All rights reserved.

[13] F. Pedonne, M. Wiesman and A. Shipper, *A Systematic Classification of Replited Database Protocols based on Atomic Broadcast*, In proceedings of the 3th European Research Seminar on Advances in distributed systems (ERSADS99). Madeira Island, Portugal, 1999.

[14] M.Wiesmann, F.Pedone, A. Schiper, B.Kemme , and G. Alonso. *Database replication techniques: a threee parameter classification*. In Proceedings of 19th IEEE Symposium on Reliable Distributed systems (SRDS2000), Nüenberg, Germany, October 2000. IEEE Computer Society.

[15] G. Popek, B. Walker, J. Chow, D. Edwards, C. Kline, G. Rudisin, and G.Thiel, *"Locus: A Network Transparent High Reliability Distributed System,"* Proceedings of the Eighth Symposium on Operating Systems Principles, pp 169-177 ACM, December 1981.

[16] José M. Pérez, Félix García-Carballeira, Jesús Carretero, Alejandro Calderón, Javier Fernández (2009), *Branch replication scheme: A new model for data replication in large scale data grids*. 0167-739X/$ see front matter ' 2009 Elsevier B.V. All rights reserved.

[17] Junwei Zhang, Bu-Sung Lee, Xueyan Tang, Chai-Kiat Yeo (2009), *A model to predict the optimal performance of the Hierarchical Data Grid*. 0167-739X/$ see front matter ' 2009 Elsevier B.V. All rights reserved.

[18] K. Sashi, Antony Selvadoss Thanamani (2010), *Dynamic replication in a data grid using a Modified BHR Region Based Algorithm.* 0167-739X/$ – see front matter © 2010 Elsevier B.V. All rights reserved.

[19] Tehmina Amjad, Muhammad Sher, Ali Daud (2011), *A survey of dynamic replication strategies for improving data availability in data grids*. 0167-739X/$ – see front matter © 2011 Elsevier B.V. All rights reserved.

[20] Najme Mansouri, Gholam Hosein Dastghaibyfard (2012), *A dynamic replica management strategy in data grid.* 1084-8045/$ - see front matter & 2012 Elsevier Ltd. All rights reserved.

[21] Najme Mansouri, Gholam Hosein Dastghaibyfard, Ehsan Mansouri (2012), *Combination of data replication and scheduling algorithm for improving data availability in Data Grids*. 1084-8045/$ - see front matter & 2012 Elsevier Ltd. All rights reserved.

[22] N. Mansouri, G.H. Dastghaibyfard, *Enhanced dynamic hierarchical scheduling strategy in data grid*, J. Parallel Distrib. Comput. (2013),

[23] R. Buyya and M. Murshed. *Gridsim : A toolkit for the modeling andsimulation of distributed resource management and scheduling for gridcomputing.* The Journal of Concurrency and Computation: Practice and Experience (CCPE), 14(13-15), 2002.

[24] H. Casanova. *Simgrid: a toolkit for the simulation of application scheduling*. In Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid'01), Brisbane, Australia, pages 430-437, may 2001.

[25] H. Casanova, A. Legrand, and L. Marchal. *Scheduling distributed applications: the simgrid simulation framework*. In Proceedings of the third Bibliographie IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03), may 2003.

[26] Dalibor Kluscek, Hana Rudov. *Alea 2 – Job Scheduling Simulator*. SIMUTools 2010 March 15–19, Torremolinos, Malaga, Spain. Copyright 2010 ICST, ISBN 78-963-9799-87-5.

[27] W. Bell, D. Cameron, R. Carvajal-Schiaffino, A. Millar, K. Stockinger et F. Zini. *«Evaluation of an Economy-Based File Replication Strategy in DataGrids »*. Dans

Third International Symposium on Cluster Computing and the Grid (CC-GRID) (2003).

[28] EU DataGrid Project. *The DataGrid Architecture*. Technical Report DataGrid-12-D12.4-333671-3-0, CERN, Geneva, Switzerland, 2001.

[29] W. H. Bell, D. G. Cameron, L. Capozza, P. Millar, K. Stockinger, and F. Zini. *Design of a Replica Optimisation Framework.* Technical Report DataGrid02-TED-021215, CERN, Geneva, Switzerland, December 2002. EU DataGrid Project.

[30] L. Guy, P. Kunszt, E. Laure, H. Stockinger, and K. Stockinger. *Replica Management in Data Grids.* Technical report, GGF5 Working Draft, July 2002.

[31] Anthony Sulistio, Uros Cibej, Srikumar Venugopal, Borut Robic, and Rajkumar Buyya, *A toolkit for modelling and simulating Data Grids: An extension to GridSim*. Copyright 0123 John Wiley & Sons, Ltd.

[32] W. Hoschek, F. Janez, A. Samar, H. Stockinger and K. Stockinger. *Data Management in an International Data Grid Project.* In Proc. of the 1st International Workshop on Grid Computing (GRID), Bangalore, India, December 17, 2000.