Résumé :

Mots clés:

grilles'données,'tolérance'aux'fautes,'points'de'reprise,journalisation,'algorithmes'hiérarchiques, 'défaillances' '

La construction des grilles informatiques est un des axes de recherche majeurs sur les systèmes informatiques en réseau. L'objectif principal de la construction d'une grille informatique, c'est de fournir les concepts et composant logiciels système adéquats pour agréger les ressources informatiques (processeurs, mémoires, et aussi réseau) au sein d'une grille de traitements informatiques, pour en faire (à terme) une infrastructure informatique globale de simulations, traitement de données ou contrôle de procédés industriels. Cette infrastructure est potentiellement utilisable dans tous les domaines de recherche scientifique, dans la recherche industrielle et les activités opérationnelles (nouveaux procédés et produits, instrumentation, etc.), dans l'évolution des systèmes d'information, du Web et du multimédia.

Les grilles de qualité production supposent une maitrise des problèmes de fiabilité, de sécurité renforcé par un meilleur contrôle des accès et une meilleur protection contre les attaques, de tolérance aux défaillances ou de prévention des défaillances, toutes ces propriétés devant conduire à des infrastructure de grille informatique sûres de fonctionnement. Dans cette thèse on propose de poursuivre des recherches sur les problèmes de gestion automatisée des défaillances, l'objectif principal étant de masquer le mieux possible ces défaillances, à la limite les rendre transparents aux applications, de façon à ce que, du point de vue des applications, l'infrastructure de grille fonctionne de façon quasi-continue.

Nous avons conçu un nouvel algorithme hiérarchique pour assurer la tolérance aux fautes dans les grilles de données. Ce protocole s'appuie sur l'architecture hiérarchique des grilles informatiques. Dans chaque cluster, nous avons défini un coordonnateur appelé processus leader, dont le rôle consiste à coordonner les échanges intra-cluster et à assurer le rôle d'intermédiaire entre les processus appartenant à des clusters différents.

Pour sauvegarder les états des processus inter-cluster, le protocole adaptatif utilise La combinaison des protocoles de journalisation optimiste et point reprise non bloquant de chandy-lamport.

A l'intérieur du cluster, le protocole exécuté dépend de la fréquence des messages. A partir d'un seuil de fréquence maximale déterminée en fonction de la densité des communications, c'est le protocole de point de reprise coordonné non bloquant qui sera utilisé, tandis que si le nombre de messages dans le cluster est faible, les messages sont sauvegardés avec la journalisation optimiste.

Introduction Générale

Ces dernières années les applications provenant de divers domaines des sciences nécessitent de plus en plus de puissance de calcul, de plus Les évolutions des technologies de réseaux et de communication , de microprocesseurs et leur performance de calcul et des supports de stockage de données ont fortement contribué à l'émergence de l'informatique distribuée .

Ce contexte a motivé la communauté informatique à s'intéresser aux architectures distribuées à large échelle, afin d'offrir des solutions pour le stockage de données et le calcul réparti à un plus grand nombre d'applications et d'utilisateurs.

Cette approche qui est à l'origine des grilles informatique ou **"*Grid*"** dont l'idée centrale est de pouvoir mutualiser les ressources de machines individuelles, pour les mettre à la disposition des utilisateurs en toute transparence.

Une grille est une infrastructure particulièrement complexe, car elle est composée de plusieurs milliers de machines très hétérogènes et réparties géographiquement sur une très large échelle.

Du point de vue utilisation, ces différents niveaux de complexité sont rendus transparents aux utilisateurs grâce à l'existence de middlewares **(intergiciel)** dédiés aux grilles tels que *Globus*[1], *gLite*,[2] Datagrid[3] …etc.

Du point de vue exploitation, la gestion des ressources d'une grille (leur dynamicité et le fort taux d'hétérogénéité), il est très difficile (voir impossible) d'avoir un système qui puisse fonctionner sans présence de pannes matérielles et logicielles .Il est donc nécessaire de

[1]Globus : un projet open source visant à créer des logiciels et des outils nécessaires pour la conception et la mise en ÷uvre de grilles de calcul.
[2] Glite : un intergiciel orionté services web. http ://glite.web.cern.ch.
[3] Datagrid : grille de données qui a rassemblé 21 partenaires scientifiques et industriels européens, regroupant jusqu'à 15000 ordinateurs et plus de 15 Teraoctets de stockage répartis sur 25 sites en Europe.

mettre en place un système de tolérance aux fautes qui puisse offrir et assurer une certaine sûreté de fonctionnement du système, En particulier dans le cas des grilles de données.

La tolérance aux pannes dans les systèmes répartis est un domaine qui a été très largement étudié depuis une trentaine d'années [RAN 75]. De nombreux protocoles de recouvrement arrière ont été développés dans la littérature. Ces protocoles peuvent être classés en deux catégories :

- Les techniques de sauvegarde par points de reprise [10]: chaque processus sauvegarde périodiquement son état et l'enregistre sur un support de stockage stable dans un point de reprise. En cas de panne, tous les processus effectuent un retour arrière vers un point de reprise appartenant à un état global cohérent, limitant ainsi la quantité de calcul perdu.
- Les mécanismes de journalisation des messages : les messages sont sauvegardés pour pouvoir être rejoué dans le même ordre en cas de défaillance.

Notre contribution consiste à :

1.Une étude comparative des protocoles de recouvrement arrière à savoir la sauvegarde coordonnée par point de reprise non bloquant de *Chandy-Lamport* [12] et le mécanisme de journalisation optimiste . Ces algorithmes ont été testés dans deux cas : intra cluster (à l'intérieur d'un même cluster) et inter cluster (à tous les clusters composant la grille) .

2. Une composition des deux algorithmes cités auparavant, cette composition est au niveau inter et intra-cluster.

Le présent mémoire est constitué de quatre chapitres à savoir :

Le premier chapitre est consacré à la notion des grilles informatiques, leurs objectifs, leurs architectures, leurs caractéristiques et domaines d'application.

Le deuxième chapitre est dédié aux concepts de sureté de fonctionnement, nous nous focalisons ensuite sur les systèmes répartis, en présentant les problèmes et les différentes solutions qui permettent d'assurer la tolérance aux fautes dans de tels systèmes à savoir la

tolérance par le recouvrement arrière proposées dans la littérature : les solutions par points de reprise et les solutions par journalisation .

Le troisième chapitre présente l'approche proposée pour tolérer les fautes dans une grille de données hiérarchiques. La description de notre proposition est faite a l'aide des algorithmes et des diagrammes UML.

Le quatrième chapitre est destiné à l'implémentation du protocole proposé. L'étude d'évaluation est décrite à partir des résultats d'expérimentation et leurs interprétations.

Nous terminons notre manuscrit par une conclusion générale qui synthétise notre solution et quelques perspectives à envisager pour la suite de ce travail.

Les applications de grilles provenant de divers domaines des sciences et nécessitant de plus en plus de puissance de calcul et de stockage émergent et se développent continuellement ces dernières années. Les calculateurs modernes se trouvent sous-dimensionnes pour de telles applications au moment où les progrès dans le domaine des télécommunications ont rendu possible le regroupement d'une multitude d'ordinateurs connectes entre eux par le réseau Internet. Une idée est alors apparue : pourquoi ne pas faire coopérer ces ressources géographiquement distribuées pour passer à un modèle informatique reparti permettant d'exploiter pleinement les ressources et les capacités offertes par le réseau. Cet environnement offrira un service avec un accès uniforme et économiquement viable aux ressources de l'infrastructure.

La solution envisagée pour ce problème est connue sous le nom de '**Grid'** ou grilles. Dans ce chapitre nous présentons une taxonomie de cette nouvelle technologie.

I.1 Définition :

Le terme grille ou grille **(Grid)** est un concept qui a été introduit pour la première fois aux Etats-Unis par *Ian. Foster* et *Carl. Kesselman* en 1998, dans le livre « *The Gri : Bueront for a New Computing Infrastructure* » **[1]** : " Une grille de calcul est une infrastructure matérielle et logicielle fournissant un accès fiable (dependable), cohérent (consistent), à taux de pénétration élevé (pervasive) et bon marché (inexpensive) à des capacités de traitement et de calcul ".

L'ensemble des ressources qui composent une grille a un certain nombre de caractéristiques, résumé comme suit **[2]** :(Voir figure I .1).

Figure I.1 : Exemple de grille informatique

- Partagées : Elles sont mises à la disposition des différents utilisateurs de la grille et éventuellement pour différents usages applicatifs.
- Distribuées : Les ressources d'une grille sont localisées dans des sites géographiques séparés.
- Hétérogènes : L'hétérogénéité des ressources d'une grille se retrouve à différents niveaux : Architectures, systèmes d'exploitation, systèmes de gestion de fichiers, systèmes de gestion de bases de données, réseaux, équipements spécifiques, etc.
- Hiérarchie administrative et sécurité : La grille est découpée en plusieurs domaines administratifs (universités, entreprises, etc..). Chaque domaine (ou site) applique une politique de sécurité particulière pour l'utilisation et l'accès aux ressources qu'il partage. Les intergiciels ont la lourde tâche de faire cohabiter ces diverses politiques de sécurité.

Un utilisateur peut se connecter à la grille seulement s'il dispose des droits d'accès : c'est le mécanisme d'authentification. Une fois authentifié il doit pouvoir exécuter une application depuis n'importe quelle machine du site (PC, Portable, PDA, etc..).

- Gestion coordonnée : La gestion des ressources d'une grille n'est pas centralisée. Elle est assurée par plusieurs gestionnaires de ressources qui ont des politiques de gestion différentes. Mais doivent néanmoins coordonner leurs actions pour assurer une politique globale de gestion d'une grille.
- Externalisées : Les ressources d'une grille sont accessibles à la demande chez un fournisseur (serveur) externe.

On y ajoute d'autres caractéristiques qu'une grille de calcul possède **[3]** :

- Passage à l'échelle (scalability) : une grille pourra être constituée de quelques dizaines de ressources à des millions voire des dizaines de millions de ressources.
- Dynamité des ressources : les grilles sont caractérisées par leur aspect dynamique (arrivée de nouveaux membres, départ des membres existants, etc..). Cela pose des contraintes sur les applications telles que l'adaptation au changement dynamique du nombre de ressources, la tolérance aux fautes et aux délais d'allocation, etc..

I.2 Architecture d'une grille :

L'architecture d'une grille est organisée en couches qui sont une abstraction représentant un ensemble de fonctions de la grille **[4]**.

Chaque couche fait appel aux services de toutes les couches inférieures. (Voir figure I.2)

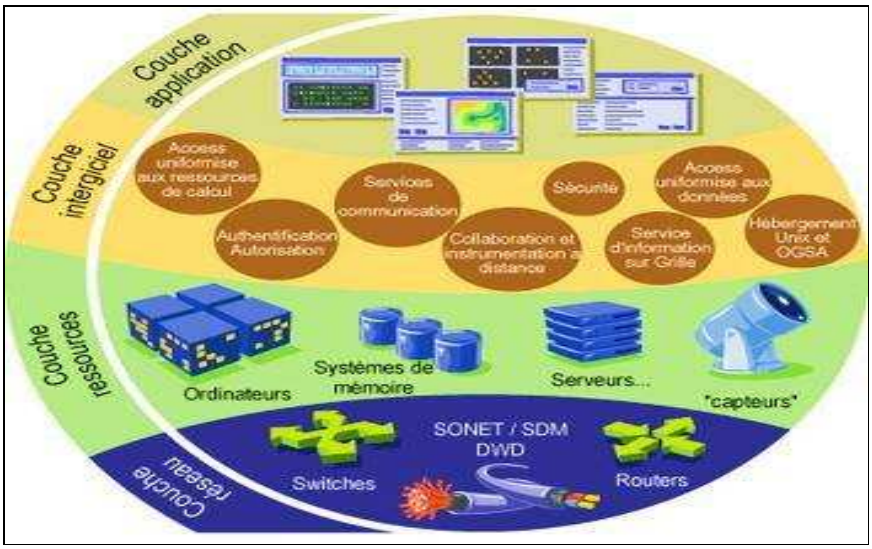

Figure I.2 : Architecture d'une grille en couches

- La couche Fabrique (Fabric) : désigne l'ensemble de l'infrastructure physique d'une grille (ordinateurs, systèmes de stockage, bases de données, etc.). Ces ressources implémentent un certain nombre de mécanismes de demandes d'informations permettant de les découvrir, de connaitre leurs états, leurs capacités et les mécanismes de gestion correspondants.
- La couche Connectivité implémente les principaux protocoles de communication et d'authentifications nécessaires aux transactions sur un réseau de type grille. Ces protocoles permettent l'échange des données à travers les ressources du niveau Fabrique et fournis des mécanismes sécurisés de vérification de l'identité des utilisateurs et des ressources.
- La couche Ressource (intergiciel) : utilise les services des couches *Connectivité* et *Fabrique* pour collecter des informations sur les caractéristiques des ressources, les surveiller et les gérer. Elle s'occupe également de l'aspect facturation (ressources payantes) et fournit les intergiciels nécessaires à la gestion des ressources, la coordination de l'accès, l'ordonnancement des tâches, etc...

- La couche Collective : regroupe tous les outils et les programmes pouvant aider les développeurs à concevoir et à développer des applications pouvant tourner sur une grille. On y trouve plus particulièrement des compilateurs, des librairies, des outils de développement d'applications parallèles ainsi que des interfaces de programmation ou API (découverte et réservation des ressources, des mécanismes de sécurité, stockage, etc...)
- La couche Application : représente l'ensemble des applications des utilisateurs qui seront exécutées sur une grille, afin de stocker et de récupérer des données ou pour y effectuer des calculs (projets scientifiques, médicaux, financiers, ingénierie, etc...).

I.2.1.Notion de Middleware :

Une grille doit intégrer un intergiciel *(middleware)* qui regroupe l'ensemble des services Logiciels pour sa mise en œuvre. Son principal rôle est d'assurer une interface entre les différentes couches de la grille.

Un intergiciel comprend notamment les fonctions suivantes **[4,1]** :

- Le partage et l'allocation des différentes ressources d'une grille.
- L'exécution, l'ordonnancement des travaux ainsi que l'administration d'une grille.
- L'ensemble des procédures de sécurisation d'une grille (d'authentification des utilisateurs, la gestion des restrictions d'accès, confidentialité des données, etc...)

Ces middlewares s'appuient sur des protocoles standards de l'Internet tels que FTP (File Transfer Protocol) **[5],** LDAP (Light Directory Access Protocol) **[6],** http (HyperText Transfert Protocol) **[7]** .Parmi les intergiciels les plus utilisés dans le domaine des grilles, nous pouvons citer *Globus, Legion*et *Unicore***[1]**.

I.3 Les différents types de grilles :

Il existe trois (03) Types majeurs de grilles, chacun correspond à un type de ressources partagées **[8].**

I.3.1 Les grilles d'informations :

Représente le premier concept de grille, permet le partage de l'information (ressources, données, applications) a tous les utilisateurs *(user)* à travers un réseau (réseau local

d'entreprise, Internet, ..) afin de rendre accessible l'information au plus grand nombre (*client)*.

La machine permettant l'accès à ces informations est appelé (*serveur)*.

L'exemple le plus caractéristique de ce type de grille est le Web **[8]** (Voir figure I.3).

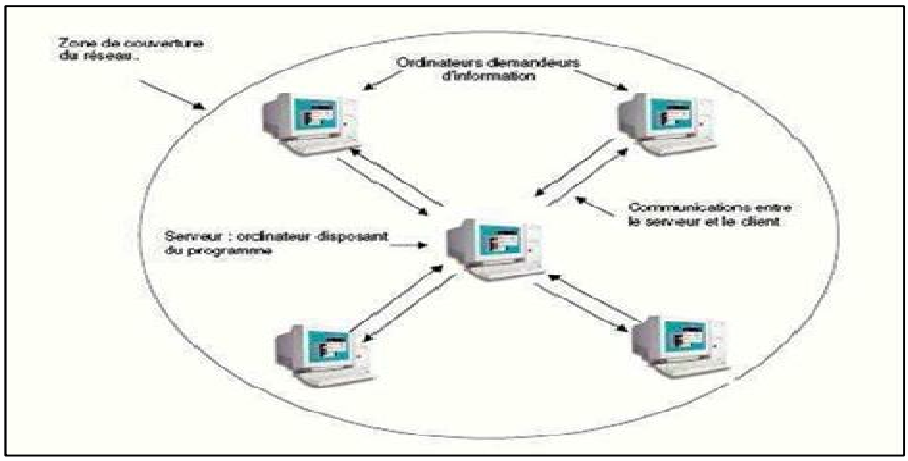

Figure I.3 : La grille informatique

I.3.2 Les grilles de données :

Connu aussi sous le nom grille de données, ce type repose sur la mise à disposition d'une grande quantité de données qui peut atteindre facilement l'échelle du téraoctet*(To)* voire beaucoup plus comme dans certaines applications scientifiques ou dans les entrepôts de données *(Data Waterhouse).*

Ces quantités importantes imposent aux développeurs de mettre en place des mécanismes spéciaux pour la recherche d'informations (moteurs de recherche), l'analyse de données ou le stockage des données, les capacités physiques pour les liens entre les éléments de la grille (disponibilité des serveurs lors de transfert de quantités de données, etc. (Voir figure I.4).

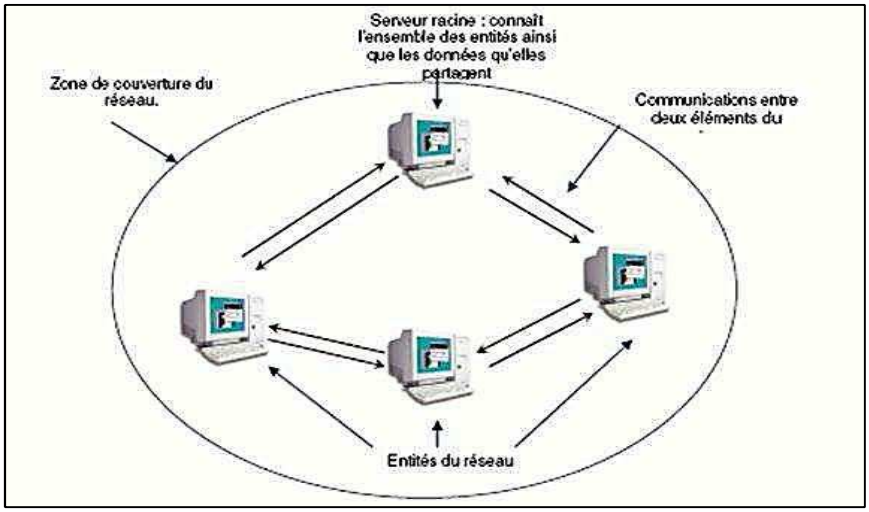

Figure I.4 : La grille de données

Les grilles fonctionnant sur ce modèle : Clients/Serveurs[1]et les grilles reposant sur le concept *Pair à Pair* (P2P)[2], on trouve comme exemple de cette catégorie de grilles, le réseau *GNUTELLA* **[1]** ou encore *KaZaA* **[1,9]**

I.3.3 Les grilles de calcul :

La mise en place de ces grilles a pour objectif de répondre aux exigences de puissance de calcul, Les premières utilisations des grilles de cette catégorie ont consisté à récupérer les heures d'inactivité des processeurs à travers le monde, via le réseau Internet. Cette approche s'est ensuite généralisée pour mettre en place le principe de regroupement des capacités de calcul de milliers, voire de millions d'ordinateurs, pour répondre à un intérêt commun.

Comme exemple pratique de ce principe, la participation d'IBM avec son *World Grid Computing***[1],** qui est un organisme mettant à la disposition des laboratoires de recherche une très grande puissance de calcul de l'ordre du *TeraFlop*[3]. C'est-à-dire qu'il est capable de réaliser 40000milliards d'opération à virgule flottante à la seconde.

I.4 Principaux enjeux des grilles :

Les grilles tendent à s'imposer dans la technologie de besoins de type Business-to-Business **[1]** et regroupe les personnes ayant un intérêt commun par l'agrégation et l'unification de leurs ressources. Ce regroupement impose un certain nombre d'enjeux que les grilles doivent satisfaire dont les cinq critères suivants **[9]** :

I.4.1 Partage de ressources :

Ce critère permet d'accéder à une grille pour partager l'utilisation de ressources distantes.

Il consiste plus qu'un simple transfert de fichiers, dans la mesure où elle requiert un accès direct à différents logiciels, ordinateurs, données et équipements et requiert des politiques de sécurité et de contrôle d'accès appropriées.

---

[1]Le modèle client/serveur désigne un mode de comportement et de communication synchrone entre un client et un serveur, où une fois que le client a soumis une requête au serveur, il doit attendre une réponse de ce dernier pour reprendre son exécution.

[2]Le premier système Pair à Pair ou Peer-To-Peer(P2P) est apparu en 1999.Il s'agissait du logiciel Napster , permettant la diffusion de fichiers musicaux (principalement au format mp3). Ce système possédait une structure centralisée, à savoir que les pairs (ou nœuds) du réseau envoyaient à un serveur central la liste des fichiers qu'ils mettaient à disposition des utilisateurs.

[3]FLOPS (Floating Point Operations Per Second) : La vitesse de traitement de la partie virgule flottante. Tera Flops = 10^{12} Flops.

I.4.2 Accès sécurisé:

Une conséquence du partage des ressources est le contrôle des accès à ces ressources pour garantir un certain nombre de règles de sécurité:

- Politique concernant l'accès : ce qui est partagé, autorisé ainsi que les conditions dans lesquelles aura lieu ce partage.
- Authentification : afin d'éviter toute utilisation abusive et non autorisée de ressources partagées, il faut préciser l'identité de l'utilisateur d'une ressource.
- Autorisation : déterminer si une opération est conforme aux règles de partage qui ont été prédéfinies.

I.4.3 Utilisation efficace de ressources :

Ce critère permet d'optimiser l'utilisation des ressources de la grille, même si leur nombre et leurs capacités dépassent les besoins des utilisateurs. Pour cela, il est indispensable d'assurer que les ressources sont utilisées de manière efficace (exploitation de leurs potentialités) et que la charge de travail est bien équilibrée afin d'éviter que des ressources soient surchargées alors que d'autres sont sous-utilisée ou complètement libres.

I.4.4 Abolition de la distance :

Les connexions à haut débit entre ordinateurs rendent possible la mise en place d'une grille à l'échelle mondiale. Il s'agit en fait de donner l'impression que la distance qui sépare un utilisateur des ressources utilisées pratiquement nulle.

I.4.5 Mise en place de normes :

Il est important de standardiser une architecture pour les grilles. Elle permettra de contribuer de manière significative à la résolution des problèmes d'hétérogénéité qui se posent au niveau des grilles.

Il existe un organisme de normalisation, appelé *GGF (Global Grid Forum),* cet organisme représente une force significative en matière d'édition de normes et d'élaboration d'éléments permettant le travail en commun. Actuellement, une norme, connue sous le nom *OGSA (Open Grid Services Architecture*) a été mise en place et considérée comme référence clé pour les projets d'élaboration de grilles.

I.5 Cadres d'utilisation de grille :

Une grille offre un éventail de possibilités pour tous les domaines pouvant bénéficier de ses capacités de traitement et de stockage. Cela est représenté en cinq grandes classes d'applications **[9] :**

I.5.1 Calcul distribué :

Les applications de calcul distribué sont évidemment d'excellentes candidates pour être utilisées sur une grille. Elles bénéficient ainsi d'un nombre beaucoup plus important de ressources de calcul leur permettant

- L'ordonnancement à grande échelle des processus.
- La souplesse des algorithmes et des protocoles qui doivent être capables de gérer un nombre de nœuds (de la dizaine à des centaines voire des milliers de machines).
- La tolérance des algorithmes aux temps de latence inhérents à la taille delà grille.
- La possibilité d'atteindre et de maintenir un haut niveau de performances dans un système très hétérogène.

I.5.2 Calcul haut débit :

Contrairement aux calculs distribues, les applications de calcul a haut débit nécessitent la résolution d'un très grand nombre de calculs de manière indépendante et parfois coordonnée. Les ordinateurs, lorsqu'ils sont inutilisés, peuvent ainsi contribuer à agréger une puissance de calcul pour résoudre des applications très exigeantes en matière de puissance de calcul.

Le projet SETI@home [27] est un exemple d'infrastructure qui a permis d'agréger une puissance de calcul très importante uniquement a partir des périodes d'inactivité des processeurs de simples ordinateurs de type PC.

I.5.3 Calcul à la demande :

Ce type d'applications utilise une grille afin de satisfaire des besoins à court terme en ressources pour des raisons pratiques ou de rentabilité. Ces ressources peuvent être du temps de calcul, des logiciels, des données.

I.5.4 Traitement massif de données :

Dans ce type d'applications, le but est d'extraire de nouvelles informations à partir de grandes bases de données géographiquement distribuées. Généralement, ces types de

traitement sont de grands consommateurs de puissance de calcul et de bande passante. Nous pouvons citer les systèmes de prévisions météorologiques.

I.5.5 Informatique collaborative :

Les grilles de calcul permettent non seulement de regrouper (virtuellement) des ressources informatiques hétérogènes, mais aussi de fédérer des milliers de personnes réparties sur des entreprises et institutions différentes dans le but d'un travail collaboratif. Dans ce contexte, les grilles de par leurs caractéristiques, constituent une véritable plate-forme de travail collaboratif moindre cout.

1.6 Organisation virtuelle:

Une organisation virtuelle ou Virtual Organization (VO) est un groupe dynamique d'entités qui décide de partager les ressources et de définir les conditions et les rôles de partage de celles-ci. Elle représente un élément clé d'une grille, permet ainsi de :

- définir de façon précise qui fait quoi à quel moment et avec quelles ressources.
- une classification et présentation des possibilités basées sur les besoins plutôt que sur les contraintes physiques.
- Une utilisation plus efficace des ressources mises en commun.
- un équilibrage de charge de la couche applicative.
- un accès commun aux ressources rares telles que des applications et du hardware spécialisés.

Dans l'exemple présenté (voir Figure I.5) deux organisations virtuelles (VO) ont été créées afin de répondre à deux besoins spécifiques : la prévision météorologique et la modélisation de séries temporelles. Chacune de ces organisations virtuelles utilise des ressources software et hardware durant la période de temps nécessaire à la résolution du problème. Imaginons que pour résoudre le problème de prévision météorologique, nous ayons besoin du software spécialisé *Weather Research and Fore-casting Model*, de différents capteurs aériens, d'une puissance informatique équivalente à 200.000 SPECint2000[4] répartis sur 200 PC équipés de processeurs Intel, d'une capacité de stockage de 200 Téra bytes3 et d'une

[4] Le SPECint2000 est un benchmark du Standard Performance Evaluation Corp. mesurant les performances des processeurs et des mémoires pour les applications intensives en calculs sur les entiers.

bande passante de 10 Gbits entre les nœuds. Le second problème de la modélisation aura besoin du logiciel spécialisé *Time Series Expert Grid* pour la modélisation des séries temporelles et d'une puissance de calcul équivalente à 100.000 SPECint2000 répartis sur 50 machines fonctionnant sur Unix. Grace aux organisations virtuelles, les utilisateurs se partagent les ressources communes. Lorsque les problèmes auront été résolus, les organisations virtuelles seront dissolues et les ressources seront à nouveau libérées pour d'autres utilisateurs **[10].**

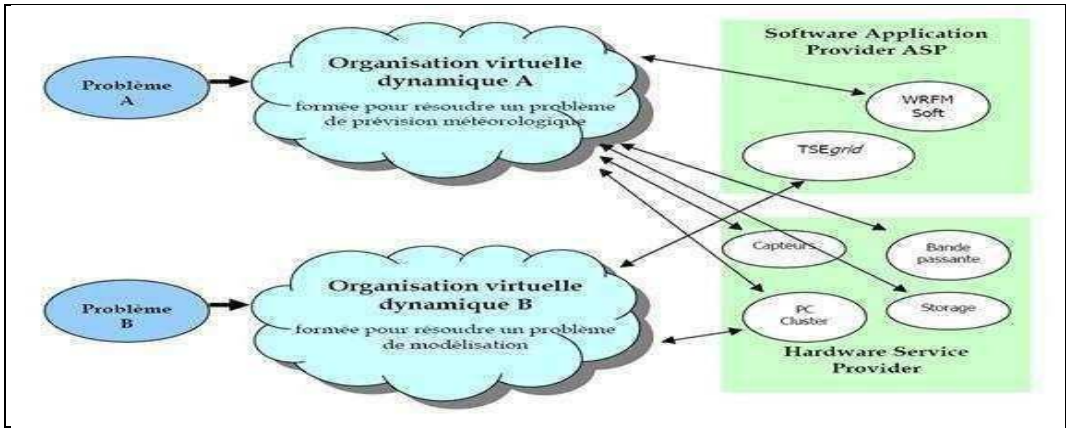

Figure I.5 Organisation Virtuelle

1.7 Fonctionnement de la grille :

Le fonctionnement d'une grille nécessite la coopération de plusieurs éléments :

Resource Broker (RB): ou Workload Manager (WM) reçoit les requêtes provenant des clients. Celles-ci sont écrites en RSL (Resource Spécification Language) et qualifiées à ce stade de RSL de haut niveau. Le broker est alors responsable de la phase de découverte des ressources. Il utilise pour cela *le Meta Directory Service.*

Le processus de spécialisation de la requête RSL peut alors commencer : il s'agit de la transformation de la requête originale en une nouvelle requête RSL pour laquelle l'endroit où se trouvent les ressources adéquates est spécifie.

Elément de calcul (CE) : il représente le point d'accès unifie à des ressources de calcul, des *Worker Nodes* qui seront utilisés par la grille pour l'exécution des jobs. Le CE se charge de la gestion des jobs qui lui sont attribués. Il gère une liste de jobs à soumettre (batch queue) au client.

Elément de stockage (SE) : c'est l'organe de gestion du stockage de l'information.

Worker Nodes (WN) : c'est un groupe de machines sur lesquelles les jobs vont être

exécutes. C'est également sur les *Worker nodes* que sont stockées les données transmises par le SE. Il s'agit généralement d'un cluster de plusieurs PC ou serveurs.

Catalogue des répliques : il fait la correspondance entre un fichier et ses répliques, on parle ainsi d'un fichier logique pour designer proprement le fichier et des noms de fichiers physique pour désigner ses répliques.

1.8 Schéma de prise en charge d'un job :

La Figure I.6 présente le cheminement d'un job exécuté sur une grille.

- L'utilisateur soumet le job au Broker (RB) via l'interface de la grille. Le Broker recherche le/les CE(s) pouvant prendre en charge l'exécution du job en consultant le système d'information. L'utilisateur transmet ses fichiers d'entrée dans l'Input SandBox.
- Le job ainsi que l'Input SandBox sont transfères au CE qui prend en charge le job dans la file des jobs.
- L'élément de calcul (CE) envoie le job sur un ou plusieurs Worker Node (WN) disponibles.
- Lorsque le job est terminé, les fichiers produits par celui-ci sont disponibles sur le LRMS (Local Resource Management System). Le Broker est informe que le job s'est terminé.
- Le Broker récupère les fichiers de sortie dans l'Output SandBox.
- Le Broker envoie les résultats (l'Output SandBox) a l'utilisateur via l'interface.
- L'utilisateur peut interroger à tout moment l''état de son job par l'intermédiaire du Logging and Bookkeeping Service (LBS). Le logging and Bookkeeping conservent une trace de l'exécution des jobs. **[11]**

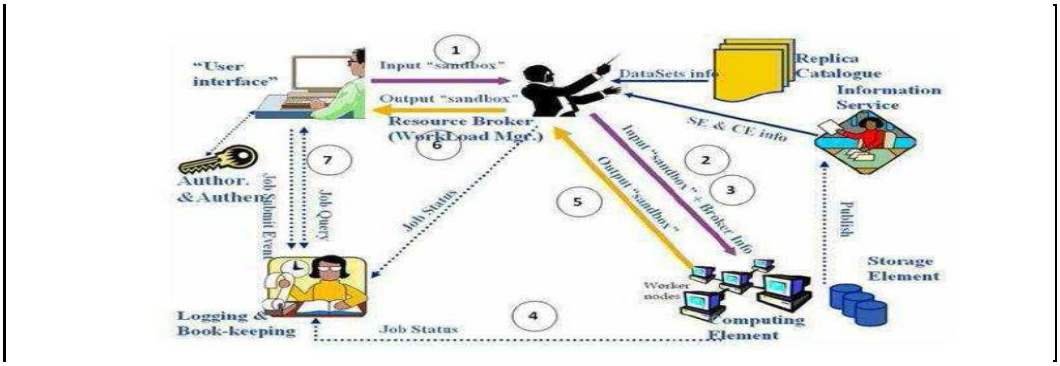

Figure I.6 Cycle de vie d'un job dans une grille de données

Conclusion :

Dans ce chapitre nous avons introduit la notion des grilles informatiques ainsi que leurs caractéristiques et leurs types. Puis nous avons détaillé la technologie des grilles de données. La grille de données possède les mêmes caractéristiques qu'une grille informatiques mais elle est concernée plus par la gestion des données (réplication, manipulation, placement, disponibilité et sécurité des données).

Les principales caractéristiques d'une grille par rapport à un système distribué résident dans la dynamicité de ses ressources et de sa taille, par conséquent les pannes deviennent des événements naturels. Delà des problèmes de fiabilité s'imposent et nous incitent à se poser les questions suivantes : Quelle sont les types de pannes dans la grille et comment garantir la disponibilité des données et la continuité des services en cas de pannes ?

Toutes ces préoccupations feront l'objet du prochain chapitre.

Dans les grilles informatiques, les applications manipulent de grandes quantités de données qui sont distribuées sur les sites d'exécution. Le nombre de sites et leur distribution géographique rendent le système dynamique à tout instant. Les sites peuvent se connecter ou se déconnecter du système suite à un départe ou une défaillance. La tolérance aux pannes est un des objectifs principaux pour la mise en œuvre de systèmes repartis et les grilles. Il existe plusieurs études qui sont développées dans ce domaine avec des propositions de mécanismes et de stratégies de contrôle pour gérer la dynamicité et la défaillance des composants de la grille.

Dans ce chapitre, nous présentons les différents concepts liés à la tolérance aux fautes dans les systèmes répartis particulièrement dans les grilles informatiques et terminer avec des références de protocoles hiérarchiques basés sur ces techniques ainsi une analyse comparatives de ces derniers.

II.1. Sureté de fonctionnement et tolérance aux fautes :

La sureté de fonctionnement est définie comme la capacité de fournir un service dans lequel un utilisateur peut raisonnablement placer sa confiance.

Elle permet de décider si un système est capable d'assurer que la fréquence de défaillance du service et la gravité de ces d´défaillances restent inferieure sa un minimum considère comme acceptable. **[12].**

La sureté de fonctionnement peut être représentée par trois paramètres décrits **[13]** : les attributs, les entraves et les moyens. (Voir figure II.1)

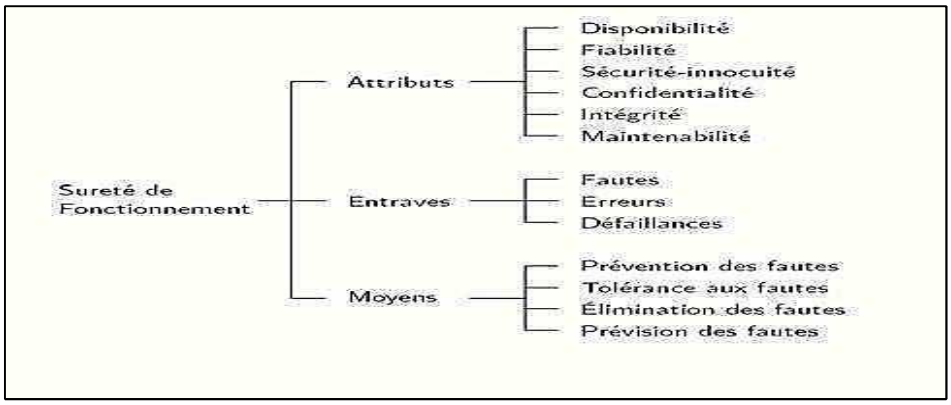

Figure II.1 : Les paramètres de sureté

II.1.1 Les attributs :

- La disponibilité : la capacité à rendre un service à tout instant.
- La fiabilité : la continuité du service.
- La sécurité-innocuité : l'absence de conséquences catastrophiques sur l'utilisateur ou son environnement.
- L'intégrité : l'absence de données corrompues dans le système.
- La confidentialité : l'absence de divulgation de données non-autorisées.
- La maintenabilité : l'aptitude du système à être réparé ou amélioré.

II.1.2 Les entraves à la sureté de fonctionnement :

- Défaillance (failure) : un élément est dit défaillant lorsqu'il ne se comporte pas conformément à sa spécification. Une défaillance est souvent appelée *panne.*
- Erreur (errer) : une erreur est un état anormal d'un élément, susceptible de causer une défaillance, mais cette dernière n'est pas encore déclarée.
- Faute (fault) : une faute peut être une action, un évènement ou une circonstance qui peut entrainer une erreur.

La figure suivante représente la relation entre ces entraves (Voir figure II.2)

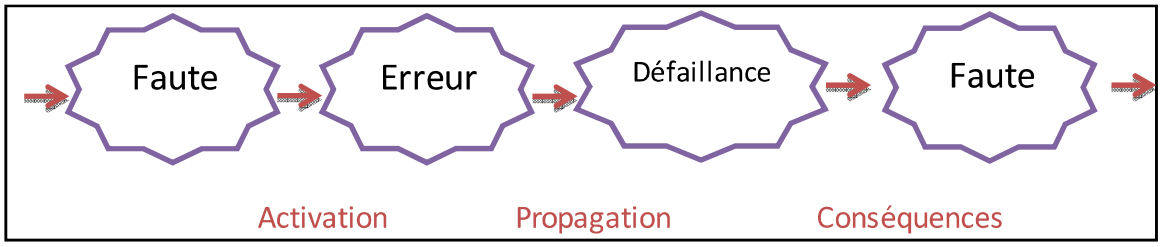

Figure II.2 : relation entre les entraves

II.2 Approches pour assurer la sureté de fonctionnement :

la sureté de fonctionnement est assurée par les moyens suivants :

II.2.1La prévention des fautes *(Fault prevention)* : Cette méthode vise à empêcher l'occurrence ou l'apparition de fautes par le développement des systèmes informatiques de manière à éviter l'introduction de fautes de conception ou de fabrication et à empêcher que des fautes ne surviennent en phase opérationnelle.

II.2.2L'élimination des fautes *(Fault removal)* : Elle se focalise sur les techniques permettant de réduire la présence de fautes ou leurs impacts. Cela est réalisé par des méthodes statiques, de la validité du système simulation, preuves analytiques, tests, ..).

II.2.3 La prévision des fautes *(Fault forecasting)*:Consiste à estimer le nombre de fautes (physiques, de conception ou malveillantes) courantes et futures ainsi que leur conséquences.

II.2.4 La tolérance aux pannes *(Fault tolerance)* : Qui essaye de fonctionner en dépit des fautes. Le degré de tolérance aux pannes se mesure par la capacité du système à continuer à délivrer son service en présence des fautes.

II.3. Tolérance aux fautes dans les systèmes distribués :

Il existe plusieurs phases successives décrivant le processus de tolérance aux fautes :

- Détection : Découvrir l'existence d'une faute (état incorrect) ou d'une défaillance (comportement incorrect).
- Localisation : Identifier le point précis (dans l'espace et le temps) où l'erreur (ou la défaillance) est apparue.
- Isolation : Confirmer l'erreur pour éviter sa propagation à d'autres parties du système.
- Réparation : Remettre le système en état de fournir un service correct. Le composant défectueux est identifié et le système fonctionne comme si les composants défectueux ne sont pas utilisés ou sont utilisés d'une façon telle que la faute ne cause pas désormais une défaillance.

II.4 Tolérance aux pannes dans les grilles :

La tolérance aux fautes est l'un des moyens de la sûreté de fonctionnement dont le principe consiste à réagir *"à chaud "* aux états erronés d'un système et à empêcher que ces erreurs ne conduisent à un dysfonctionnement visible pour l'utilisateur du système considéré **[14].**

La nature dynamique des grilles est à la fois un atout et un défi car elle permet d'ajouter des ressources au fur et à mesure de leur disponibilité ou d'en retirer.

Par ailleurs, pour le concepteur du système, cette nature dynamique soulève de nombreux problèmes, Le premier étant liée à une caractéristique des grilles, la **grande échelle :** plus le

nombre de nœuds de la grille est grand, plus la probabilité d'occurrence d'une faute est importante.

Lorsqu'un nœud s'arrête (à cause d'une défaillance ou d'un arrêt volontaire), les données qu'il stockait ne sont plus accessibles, les chemins du réseau risque d'être coupées, et les nœuds qui communiquaient avec lui risquent d'être bloqués, etc. Il est donc nécessaire de prendre des précautions afin que le système puisse continuer d'évoluer malgré la faute. **[15]**

II.4.1 Types de pannes dans les grilles :

Les types de défaillances qui peuvent survenir dans les grilles de calcul sont classés en cinq catégories **[12]** :

- Les pannes franches *(crash, fail-stop) :* appelé aussi arrêt sur défaillance, c'est le cas le plus simple. On considère qu'un processus peut être dans deux états, soit il fonctionne et donne un résultat correct, soit il ne fait plus rien. Dans ce cas, le processus est considère comme définitivement défaillant.
- Les pannes par omission *(transient, omission failures).* Dans ce cas, on considère que le système peut perdre des messages. Ce modèle peut servir à représenter des défaillances du réseau plutôt que du processus.
- Les pannes de temporisation *(timing, performance failures)* : Ce sont les comportements anormaux par rapport à un temps (l'expiration d'un délai d'attente).
- Les pannes arbitraires, ou byzantines *(malicious, byzantine failures)* : Cette classe représente toutes les autres pannes : le processus peut alors faire « n'importe quoi », y compris avoir un comportement malveillant.
- Les pannes des middlewares :

Un middleware permet la communication entre les clients et les serveurs et l'échange d'informations. Il doit fournir un moyen aux clients pour trouver leurs serveurs et les serveurs pour trouver les clients. Les principaux types de défaillances sont liés à faute de configuration dû au manque de contrôle des ressources sur la grille. La Figure II.3 montre la relation entre les types de pannes.

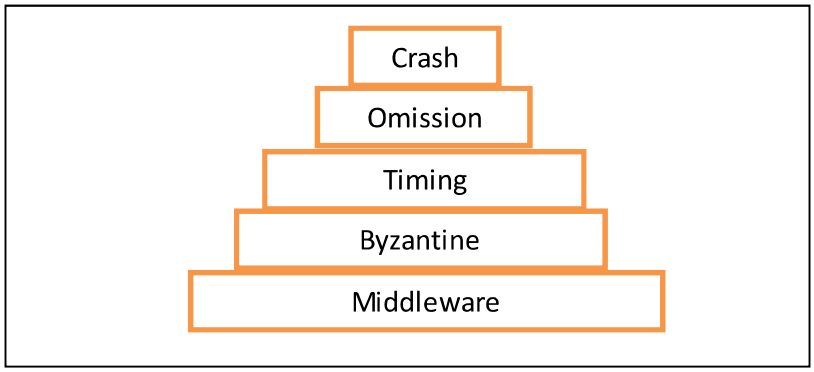

Figure II.3 : Relation entre les types de pannes

II.5 Techniques de tolérance aux fautes dans les grilles de données :

La tolérance aux pannes dans les grilles est un problème qui mobilise beaucoup de travaux de recherche. On distingue deux principales techniques de tolérance : La réplication et Le recouvrement arrière.

II.5.1 La Réplication

II.5.1.1Principe :

L'idée d'utiliser la redondance dans les systèmes informatiques pour masquer les défaillances des composants a été introduite par Von Neumann en 1956**[15]**.

La réplication de données consiste à stocker des données sur plusieurs nœuds géographiquement distribués .L'intérêt premier est que, si une donnée n'est plus disponible, le système peut continuer à assurer ses fonctionnalités en utilisant une donnée répliquée, ce qui permet d'augmenter la disponibilité des données et la tolérance aux pannes. (Voir figure II.4)

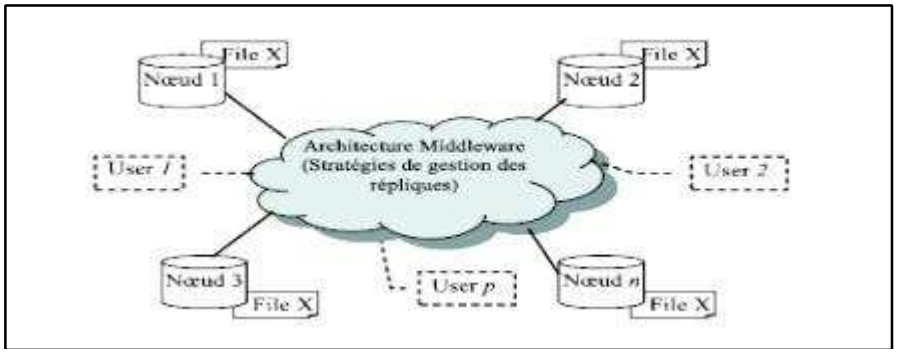

Figure II.4 : Environnement d'utilisation de la technique de réplication

II.5.1.2Techniques de la réplication :

La création des copies ou répliques d'une entité consiste à reproduire la structure et l'état des entités répliquées (copie d'un fichier est un autre fichier de même contenu, copie d'un

programme est un autre programme exécutant le même code, etc.)

Le placement des copies ou des répliques consiste à choisir, en fonction des objectifs de réplication, un environnement de stockage et/ou d'exécution pour les copies. Par exemple, le placement dans un environnement accessible localement assure le travail en mode déconnecté alors que le placement de répliques sur plusieurs machines permet une distribution de charge **[16].**

II.5.2 Le Recouvrement :

II.5.2.1 Notion d'État global cohérent & incohérent:

- L'état global d'un système est défini comme la collection des états locaux des processus et des canaux de communication qui les relient. L'état local d'un processus *p* est défini par son état initial et l'ensemble des événements ayant eu lieu sur *p.*
- Un état global *cohérent* est un état global intervenu lors d'une exécution correcte. Cet état doit garantir la causalité des événements.
- L'ensemble des processus de la grille communiquant par passage de messages, l'état global *cohérent* doit assurer que l'événement d'émission d'un message *m* précède sa réception.
- Un ensemble de points de reprise représentant un état global cohérent est dite *une ligne de recouvrement.*
- L'effet domino : est caractérisé par une cascade de retours arrière. Lors de la reprise du système après une panne.
- Un message orphelin *:* est un message qui a été reçu sans être envoyé. Lors du calcul de la ligne de recouvrement, si la réception d'un message appartient à un des états locaux et son émission n'appartient pas à un de ces états, l'état global devient *incohérent.* La figure II.5&II.6 illustre ce principe.

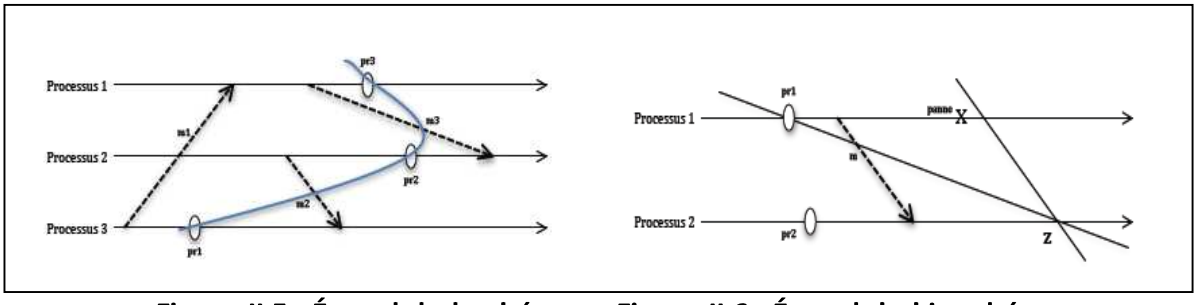

Figure II.5 : État global cohérent Figure II.6 : État global incohérent

II.5.2.2 Le recouvrement d'erreur :

Le recouvrement ou recouvrement d'erreur consiste à remplacer un état erroné par un état stable garantissant un système fonctionnel. Il existe deux techniques de recouvrement :

- Le recouvrement par poursuite: le système est ramené à un nouvel état *reconstitué*, sans effectuer de retour arrière. La reconstitution n'est souvent que partielle. C'est une technique exige spécifique parce que exige une analyse préalable des différents types d'erreurs possibles pour que la reconstitution d'état soit correct.
- Le recouvrement par reprise: C'est une technique générale qui ramène le système à un « état correct sans erreur » qu'il occupait (*Retour arrière*). Elle nécessite la sauvegarde périodique de l'état du système sur support stable**. [17]** (voir figure II.7)

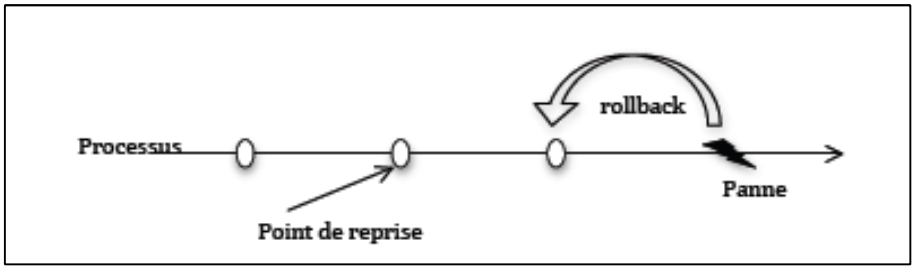

Figure II.7 : Recouvrement arrière après panne (rollback-recovery)

- Un état stable, sauvegardé au cours del 'exécution des applications est appelé *point de reprise* (*checkpoint*).
- Périodiquement, un point de reprise est effectué et stocké sur un support stable (disque ou un serveur). En cas de défaillance, le système redémarre au point de reprise le plus récent.
- Il effectue ainsi un *retour arrière* depuis un point de reprise antérieur à la panne : ce qui constitue un gain en terme de temps d'exécution, puisse que les applications ne reprennent pas leur calcul à l'instant zéro.

II.5.2.3. Analyse des mécanismes de recouvrement :

Les deux techniques vues en section (5.2) présentent des avantages et des inconvénients :

- La reprise, est la plus utilisée dans la plupart des architectures informatiques, mais peut être couteuse en temps et en espace de stockage. Etant donnée les phases de sauvegarde des points de reprise augmentent le temps de réponse des applications.

- En plus, les points de reprise occupent de la place selon le protocole exécuté dans le système.
- La poursuite présente un champ d'application limité, efficace si les types d'erreurs possibles ont été listés antérieurement.

II.6 Protocoles de recouvrement arrière :

II.6.1 Les protocoles de points de reprise :

Par définition, un point de reprise (*checkpoint*) est la sauvegarde de l'état d'un processus à un instant donné sur un support de stockage stable (plus souvent un serveur). Les protocoles basés sur les points de reprise permettent de faire des sauvegardes périodiques de l'ensemble des états des processus.

La tolérance aux fautes par recouvrement arrière se base sur les protocoles de points de Reprise pour éviter la perte de temps de calcul dans les environnements de calcul haute performance comme les grilles. Il existe différentes catégories de protocoles de points de reprise classés selon la méthode utilisée pour effectuer la sauvegarde : **[18]**

II.6.1.1Points de reprise coordonnés :

Ce protocole exige aux processus de coordonner l'établissement de leurs points de reprise pour former un état global cohérent. L'algorithme bloquant se déroule comme suit (voir figure II.8) :

- Les communications sont bloquées pendant l'établissement des points de reprise.
- Un coordinateur fait un point de reprise et diffuse un message '*checkpoint-request*' à tous les processus pour initier une phase de sauvegarde.
- Quand un processus reçoit ce message, il arrête son exécution et vide ses canaux de communication, fait une tentative de point de reprise et envoie un accusé de réception au coordinateur.
- Après avoir reçu l'accusé de réception de tous les processus, le coordinateur diffuse un message *commit* à tous les processus pour leur ordonner de faire un point de reprise.
- Chaque processus supprime ainsi l'ancien point de reprise, et sauvegarde son état. Le

- processus est maintenant libre de terminer son exécution et échanger des messages avec les autres processus du système.

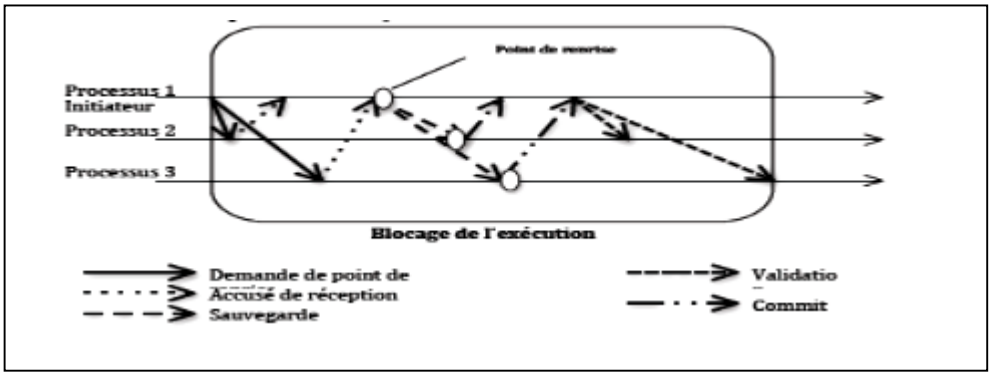

Figure II.8 : Point de reprise coordonné

Cette approche simplifie le recouvrement et évite l'effet domino, puisque chaque processus redémarre toujours au point de reprise le plus récent. Aussi, le protocole exige à chaque processus de maintenir un seul point de reprise permanent en mémoire stable, ce qui réduit le surcoût dû au stockage.

II.6.1.2 Points de reprise non-coordonné (indépendante):

Ce protocole évite la phase de synchronisation en laissant à chaque processus l'autonomie de sauvegarder son état local au moment opportun, dans le cas par exemple où la quantité d'information est minimale, cela réduit le surcoût du stockage.

Les étapes de recouvrement dans ce protocole sont (voir figure II .9) **[19]** :

- Le processus de récupération entame un retour arrière par la diffusion d'un message de *dependancy-request* pour rassembler les informations de dépendance maintenues par chaque processus.
- Quand un processus reçoit ce message, il arrête son exécution et en voie les informations de dépendances enregistrées sur la mémoire stable.
- L'initiateur (processus initie) calcule la ligne de recouvrement basée sur les 'informations de dépendance' globales et diffuse un message de *rollback request* contenant la ligne de recouvrement.
- Sur réception de ce message, chaque processus appartenant à la ligne de recouvrement reprend son exécution, sinon il revient au point de reprise précédent, celui indiqué par la ligne de recouvrement.

La dernière étape de l'algorithme peut causer l'effet domino, qui peut engendrer une perte du travail réalisé avant la panne et la sauvegarde inutile de points de reprise qui ne feront jamais partie d'un état global cohérent.

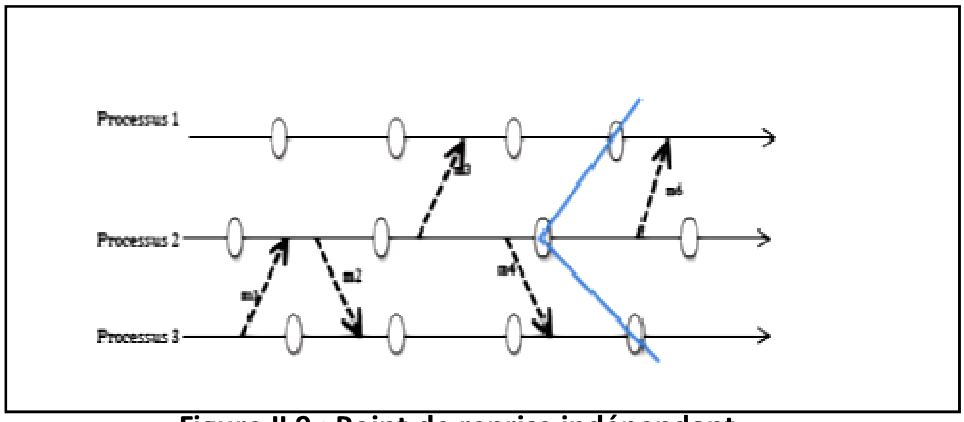

Figure II.9 : Point de reprise indépendant

II.6.1.3 Points de reprise induit par les communications :

Ce protocole (CIC: Communication Induced Checkpointing) définit deux types de points de reprise **[20][21]**:

- *Des points de reprise locaux* pris par les processus de manière indépendante, pour éviter la synchronisation de la sauvegarde coordonnée.
- *Des points de reprise forcés* en fonction des messages reçus et envoyés et des informations de dépendance estampillées sur ces messages, pour ainsi éviter l'effet domino de la sauvegarde non coordonnée, garantir l'avancement de la ligne de recouvrement.

A l'opposé des protocoles de point de reprise coordonné, ce protocole ne nécessite aucun échange de message pour forcer un point de reprise. Par ailleurs il présente des inconvénients liés à la fréquence de sauvegarde des points de reprise forcés. Comme les points de reprise augmentent le temps d'exécution des applications dans les grilles, il serait mieux de réduire leurs nombres pour éviter la dégradation des performances du système.

II.6.2 Les protocoles de journalisation :

La journalisation des messages est une technique très utilisée pour assurer la tolérance aux pannes dans les systèmes répartis.

En cas de panne d'un processus, il est relancé à partir du dernier point de reprise, et tous les messages reçus après ce dernier lui sont renvoyés dans l'ordre où ils ont été initialement

reçus.

Tous les protocoles de journalisation des messages exigent que l'état d'un processus récupérable soit toujours compatible avec l'état des autres processus. Ils doivent ainsi garantir l'absence du processus *orphelin*, qui rend le calcul de l'état global incohérent.

On distingue trois catégories de journalisation de messages : optimiste, pessimiste, causale.

II.6.2.1 Journalisation optimiste :

- Au cours de l'exécution des processus, le contenu et identifiants des massages sont conservés en mémoire volatile avant d'être périodiquement vidés sur support stable.
- Cependant une panne peut survenir avant que les messages ne soient enregistrés dans ce cas, les informations sauvegardées en mémoire volatile du processus en panne sont alors perdues .Le recouvrement devient compliqué puisque tous les processus ayant reçu un message provenant du processus défaillant deviennent *orphelin*s. Ce qui peut produire l'effet domino. En plus, le protocole est obligé de maintenir plusieurs points de reprise en mémoire. La récupération des points de reprise devient alors plus complexe.
- si le processus 3 tombe en panne avant la sauvegarde des messages m3 et m5, il va reprendre à partir du point de reprise pr3. Les messages m2 et m5 devenant orphelin, le processus 2 va reprendre à partir du point de reprise pr2. (voir figure II.10).

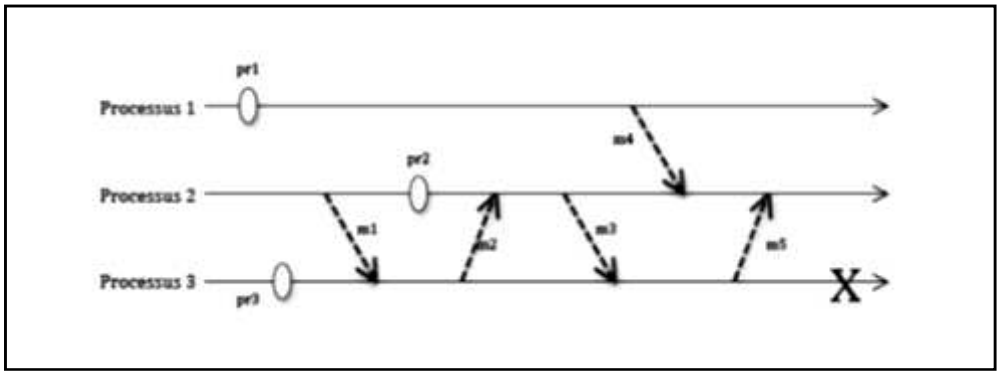

Figure II.10: Enregistrement de messages optimiste

II.6.2.2 Journalisation pessimiste :

- Ce protocole a été conçu dans l'hypothèse qu'une panne peut se produire après n'importe que l'événement non déterministe[1]. Il enregistre en mémoire stable le

[1] Un événement est déterministe si et seulement si plusieurs exécutions de cet événement à partir du même état initial donnent le même état final.

déterminant de chaque message avant que ce dernier ne soit autorisé à interagir avec le système.

- Il est souvent référence à la journalisation synchronisée car l'orque un processus enregistre un événement non déterministe sur mémoire stable, il attend de recevoir un acquittement pour continuer son exécution.
- Dans un système à journalisation pessimiste, l'état de chaque processus est toujours récupérable cela rend le recouvrement simple car les effets d'une panne se limitent seulement aux processus en panne. (voir figure II.11**). [22]**

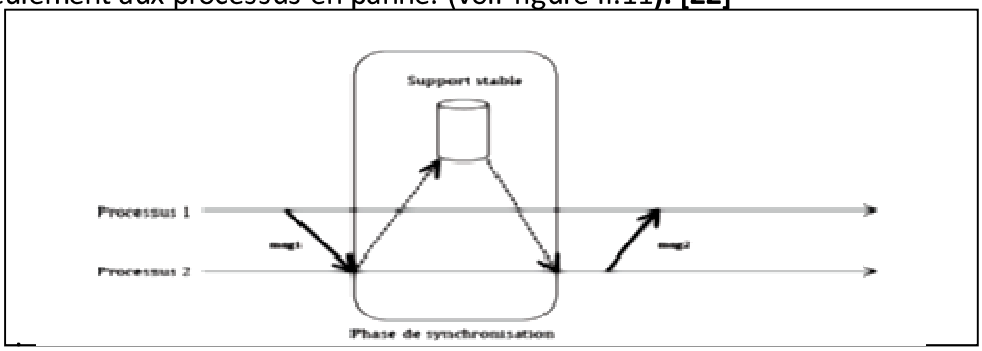

Figure II.11 Processus de sauvegarde avec la journalisation pessimiste

II.6.2.3 Journalisation causale :

Permet aux processus d'effectuer des interactions avec l'extérieur de manière indépendante.

La procédure de journalisation s'effectue de la manière suivante:

- les messages sont enregistrés sur support stable, mais le processus n'attend pas un accusé réception pour continuer son exécution.
- Chaque processus détient une table de dépendances causales de messages. Tant que l'accusé de réception des messages n'est pas arrivé, les déterminants des messages sont ajoutés aux informations de dépendance de ces messages, en sachant que la table de dépendance causale de chaque message est attachée aux messages échangés.
- En cas de panne, le processus défaillant effectue un retour arrière au point de reprise le plus récent et rejoue les messages enregistrés sur support stable ou utilisent les tables de dépendances des processus. (voir figure II.12).
- Le processus *2* subit une défaillance. Il va redémarrer à partir de son dernier point de reprise et rejouer tous les messages reçus, comme par exemple le message *m4* dont le déterminant est contenu dans le message *m5* grâce au protocole de journalisation

causale qui attache les déterminants des messages sur les autres messages circulant dans le système.

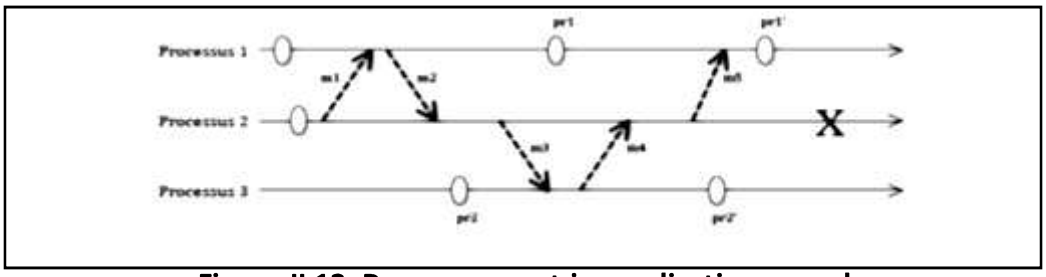

Figure II.12: Recouvrement journalisation causale

II.7. Analyse comparative :

Les mécanismes de recouvrement arrière sont les techniques de base pour assurer la tolérance aux fautes dans les grilles informatiques. Les protocoles à base de point de reprise et de journalisation permettent de sauvegarder les états locaux du système qui seront utilisés durant le recouvrement pour éviter que les applications ne recommencent leur exécution à partir du début.

II.7.1 Comparaison des protocoles de points de reprise :

Les mécanismes à base de points de reprise permettent de sauvegarder les états globaux cohérents de la grille à un instant donné.

Ces protocoles se différencient par les critères suivants :

- Le surcoût engendré durant l'exécution sans faute.
- Le surcoût du recouvrement.
- L'utilisation du support de stockage stable.
- **La résistance aux effets domino et messages orphelins.**

II.7.1.1 Le surcoût engendré durant l'exécution sans faute :

L'approche indépendante est la technique la plus simple dans sa manière de sauvegarder les états des processus. En effet, les processus sauvegardent leur état sans aucune coordination, tandis que pour la sauvegarde coordonnée, l'exécution est bloquée durant toute la procédure d'établissement des points de reprise. La synchronisation des processus engendre un surcoût assez élevé si la latence du réseau est importante, surtout quand il s'agit d'une grille informatique où les clusters sont reliés par des liens à longue distance.

Les protocoles induits par les communications permettent d'éviter le coût des synchronisations tout en maintenant des ensembles de points de reprise représentant un état global cohérent. Cependant, leur principal surcoût est dû à la pose de nombreux points de reprise forcée. Ces derniers sont effectués en utilisant des données supplémentaires ajoutées aux messages. Ces données peuvent ralentir les communications du réseau. Des points de reprise sont forcés aussi après chaque interaction avec l'extérieur. Dans le cas d'applications fortement communicantes avec plusieurs interactions avec l'extérieur, les temps d'exécution vont s'allonger à cause de la sauvegarde de ces points de reprise forcés. **[23]**

II.7.1.2 Le surcoût du recouvrement :

Même si le protocole indépendant simplifie la manière d'établir des points de reprise, le recouvrement est complexe. Les derniers états locaux des processus ne font pas toujours partie de la ligne de recouvrement. Pour calculer un état global cohérent, le protocole est obligé, dans la plupart des cas, de faire plusieurs retours arrière.

Pour la technique coordonnée, tous les points de reprise sont utiles. Le système redémarre à partir du dernier point de reprise. Le protocole induit par les communications peut aussi sauvegarder des points de reprise qui ne sont pas utiles. Ces derniers peuvent faire partie de chemin zigzag [42] qui entraine des cycles qui rendent la procédure de recouvrement assez longue.

II.7.1.3 L'utilisation du support de stockage stable :

Les points de reprise sauvegardés par les protocoles sont enregistrés sur un support de stockage stable qui peut être par exemple un serveur NFS répliqué ou un disque de stockage redondant. Les protocoles indépendants effectuent plusieurs points de reprise au cours de l'exécution des applications. Comme les processus sauvegardent leurs états de manière totalement indépendante, le protocole maintient plusieurs points de reprise en support stable pour garantir le calcul d'un état global cohérent lors de la réexécution.

De même, le protocole induit par les communications préserve plusieurs points de reprise. Le maintien de plusieurs points de reprise sur support stable engendre un large surcoût durant la procédure de libération (*garbage collection*).

À la différence de la technique indépendante, un protocole coordonné ne garde que le

dernier point de reprise sur le support de stockage car elle garantit la ligne de recouvrement. Dès qu'un point de reprise est sauvegardé, le précédent est supprimé.

II.7.1.4. La résistance aux effets domino et messages orphelins :

Un état global cohérent est un état ne contenant pas de messages orphelins. Il faut que le protocole garantisse une ligne de recouvrement ne contenant aucun processus orphelin. La sauvegarde de points de reprise coordonnée garantit l'absence d'orphelin. Cependant, en cas de défaillance, le protocole indépendant à tendance à produire des messages orphelins qui vont entrainer un effet domino. Ce dernier peut faire redémarrer l'exécution à partir du début.

II.7.2 Comparaison des protocoles de journalisation :

L'objectif principal des protocoles de journalisation est l'enregistrement des messages qui transitent entre les processus sur un support stable, afin d'être utilisé ultérieurement en cas de défaillance. Durant le recouvrement, l'exécution redémarre au dernier point de reprise, et les messages sauvegardés sont rejoués. Nous allons comparer les mécanismes de journalisation selon les critères suivants :

- Le surcoût durant l'exécution.
- Les performances durant le recouvrement.
- L'utilisation du support de stockage stable.

II.7.2.1. Le surcoût durant l'exécution :

La journalisation pessimiste a un impact significatif sur le temps de communication. En effet, pour chaque envoi, le protocole enregistre le message sur mémoire stable avant que le processus ne soit autorisé à interagir avec le système. Cette synchronisation introduit un large surcoût sur le temps d'exécution des applications.

À la différence du protocole pessimiste, le surcoût de l'approche optimiste est réduit en diminuant la fréquence d'enregistrement des messages sur le support stable.

Dans la journalisation optimiste les messages sont sauvegardés par vague. Chaque message envoyé est enregistré en mémoire volatile ; à un instant donné les messages en mémoire

sont déversés sur le support stable. Ainsi pour éviter le ralentissement des communications, il suffit de réduire les fréquences d'accès au support stable. Cependant, ce réglage peut allonger le temps de recouvrement, si la fréquence d'enregistrement est trop longue.

Si les mécanismes d'enregistrement pessimiste et optimiste ont un surcoût lié aux sauvegardes fréquentes sur le support stable, la journalisation causale quant à elle induit d'importantes quantités de données transportées à travers le réseau.

L'approche causale ajoute des déterminants dont la taille peut être très grande dans les messages échangés entre les processus. Les communications à travers le réseau peuvent être alors surchargées, ce qui augmente le temps d'exécution des applications**.[24]**

II.7.2.2 L'utilisation du support de stockage stable :

La comparaison des temps d'exécution des protocoles de journalisation à montrer que les accès fréquents au support stable introduisent un surcoût important. En effet, la journalisation est basée sur la sauvegarde des messages. Le support de stockage joue donc un rôle prépondérant dans la mesure où il doit pouvoir contenir tous les messages transitant dans le réseau.

Le protocole d'enregistrement causal augmente la taille des messages à cause des données supplémentaires ajoutées aux messages.

Le protocole de journalisation causale utilise ainsi partiellement les supports de stockage. Mais tel n'est pas le cas pour les deux autres mécanismes de journalisation.

Les protocoles à enregistrement pessimiste et optimiste utilisent le support stable pour sauvegarder les messages. La première approche ne modifie pas la taille des messages, alors que la deuxième enregistre les informations de dépendance dans un vecteur ou graphe de dépendance dont la taille peut atteindre le nombre total de processus du système**. [25][26]**

II.7.2.3. Les performances durant le recouvrement :

Les performances durant le recouvrement dépendent du nombre de processus à reprendre. Quel que soit le protocole de journalisation utilisé, le processus fautif redémarre au point de reprise le plus récent et rejoue les messages du journal. Dans le cas de la journalisation optimiste, plusieurs processus peuvent être amenés à reprendre leur exécution.

En effet si une panne survient avant l'enregistrement des messages sur support stable, une partie de l'exécution sera perdue, il faut alors régénérer les messages en ré-exécutant les processus émetteurs à partir des données antérieures sauvegardées. Ceci rend le recouvrement long et complexe à cause du nombre potentiellement important de processus qui reprennent et ainsi que le nombre de retour arrière.

II.7.3. Synthèse :

Le tableau 1**[27]** est un récapitulatif des différences entre les protocoles de tolérances aux fautes. À travers les comparaisons faites précédemment, les deux classes de protocoles de recouvrement arrière se différencient par les critères comme le surcoût durant l'exécution sans recouvrement, les performances durant le recouvrement et les accès au support de stockage.

Une des principales différences entre les protocoles à base de point de reprise et de journalisation est la complexité du recouvrement. En effet, pour la journalisation, seul le processus fautif reprend tandis que pour les techniques de point de reprise tous les processus reprennent leur exécution.

Protocole	Hypothèse PWD	Processus orphelin	Effet Domino	Nombre Sauvgardes
Journalisation Péssimiste	Oui	Non	Non	Une
Journalisation Optimiste	Oui	Possible	Non	Possible
Journalisation Causale	Oui	Non	Non	Une
Sauvgarde Coordonnée	Non	Non	Non	Une
Sauvgarde Non Coordonnée	Non	Possible	Possible	Toutes
Sauvgarde induite par les communications	Non	Possible	Non	Plusieurs

Tableau 1 : Tableau comparatif des mécanismes de recouvrement arrière

Conclusion

Dans ce chapitre, nous avons présenté essentiellement les techniques utilisées pour assurer la tolérance aux fautes dans les grilles informatiques. Pour comprendre l'origine des pannes dans les grilles, nous avons exposé le vocabulaire relatif à la sûreté de fonctionnement, pour ensuite centrer notre étude sur les mécanismes de recouvrement arrière par reprise. Ces

mécanismes se heurtent à plusieurs problèmes liés à la nature répartie des grilles qui rend complexe le calcul de l'état global cohérent du système, et au déterminisme de l'exécution des applications.

Ces problèmes liés à la reprise permettent de classifier les protocoles de recouvrement arrière en deux groupes : les protocoles de sauvegarde par point de reprise et les protocoles de Journalisation des messages. Ces protocoles se différencient particulièrement de par leur manière de sauvegarder des points de reprise, d'effectuer le recouvrement en cas de panne. Cependant, ces techniques ne sont pas totalement adaptées à tous les types de grilles.

En effet, il n'existe pas de solutions génériques de tolérances aux fautes dans les grilles informatiques.

Dans le chapitre suivant, nous allons faire une étude détaillée des protocoles de tolérance aux fautes et établir une combinaison hiérarchique adaptée aux architectures de grilles.

Les environnements à grande échelle telles que les grilles informatiques permettent l'exécution de grandes applications scientifiques qui nécessitent des temps d'exécution qui peuvent aller sur plusieurs heures voire plusieurs jours. Le principal objectif de ces Infrastructures est d'utiliser au mieux les puissances de calcul et de stockage disponibles au sein des ressources connectées à la grille informatique. Pour assurer la sûreté de fonctionnement au sein de la grille, il faudra mettre en place des mécanismes de tolérance aux fautes adaptés à leur architecture. C'est dans ce contexte notre approche est basée sur les protocoles de points de reprise et de journalisation aux grilles informatiques. Nous allons faire une étude approfondie de ces deux protocoles avec les différentes combinaisons possibles.

III.1 Modèle du système :

On considère qu'une grille informatique est un ensemble de ressources regroupé dans différents clusters. On pourra définir ainsi une grille informatique comme une fédération de clusters reliés par un réseau de type WAN (Wide Area Network).

En général dans les systèmes répartis, communiquent **par passage de messages**. Dans ce type de communication, l'acteur principal est le **processus**. Les processus communiquent en s'échangeant des messages en utilisant au moins deux procédures :

- Une procédure d'envoi de messages
- Une procédure de réception de messages

Dont le but l'échange de données ou la synchronisation entre les processus. Chaque envoi ou réception de messages est un événement. Les échanges entre les processus produisent des dépendances entre eux.

Notre approche inspirée des solutions des auteurs de **[25] [27],** est basée sur la structure hiérarchique des grilles informatiques. Nous avons adapté les protocoles de point de reprise et de journalisation à l'architecture hiérarchique de la grille. Le cluster est l'unité représentative dans laquelle va s'exécuter un des protocoles de sauvegarde d'états.

Au sein de chaque cluster, un **processus leader** assure le rôle de coordonnateur (voir Figure III.13).

Il est relié aux autres processus appartenant au même cluster, mais aussi aux leaders des autres clusters appartenant à la même grille informatique. Quel que soit le protocole utilisé, le leader assure toujours le rôle d'intermédiaire dans les communications inter-cluster.

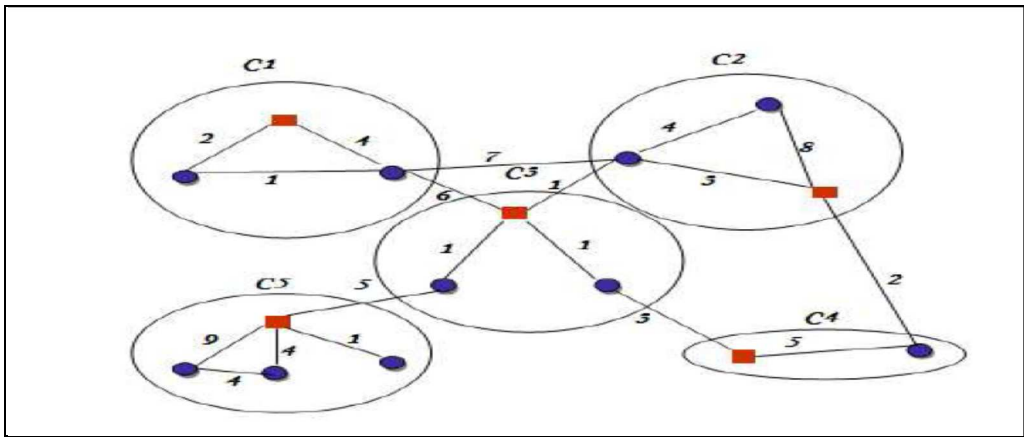

Figure III.13 : Architecture hiérarchique d'une grille

III.2. Protocoles hiérarchiques :

Dans cette section, nous décrivons deux protocoles hiérarchiques représentatifs qui illustrent notre approche à base de leader.

III.2.1 Sauvegarde coordonnée :

La procédure de sauvegarde de l'état global cohérent de la grille peut être divisée en quatre grandes phases:

- *Phase d'initialisation* : un processus initiateur envoie une demande d'établissement de point de reprise à son leader.
- p*hase de coordination des leaders* : Durant cette phase les leaders constituent les uniques acteurs. Après la réception de la demande de sauvegarde, le leader de l'initiateur envoie la même demande de point de reprise aux autres leaders de la grille.
- *Phase de sauvegarde de point de reprise local* : après la réception de la demande, chaque cluster sauvegarde son état local en utilisant le mécanisme de point de reprise coordonné décrit dans le chapitre précédent.
- *Phase de clôture* : c'est la dernière étape où chaque leader envoie un accusé de réception au leader de l'initiateur pour lui signifier la fin de sauvegarde de l'état global de son cluster.

III.2.1. Point de reprise coordonné non bloquant de Chandy et Lamport :

La solution de sauvegarde non bloquante de Chandy et Lamport est une variante des protocoles coordonnés à base de points de reprise. Le principal problème des systèmes répartis est l'absence d'horloge commune. Pour résoudre ce problème, il faudra concevoir un algorithme qui puisse enregistrer les états de processus et de leurs canaux de communication, de telle sorte que l'ensemble de ces états appartient à un état global cohérent [12]. En même temps, le calcul de l'état global du système ne doit pas empêcher l'exécution des processus et vice et versa l'exécution continue du système ne doit pas modifier le calcul de l'état global. C'est ainsi qu'ils ont élaboré *l'algorithme du snapshot* communément appelé p*rotocole de points de reprise non bloquant de Chandy et Lamport*. L'objectif du protocole est de sauvegarder (capter) un état global passé du système qui soit cohérent (instantané). Pour ce faire, les auteurs définissent l'état global cohérent du système (cliché global instantané) comme étant la somme des états locaux des processus (clichés locaux) et des canaux de Communication (Figure III.14).

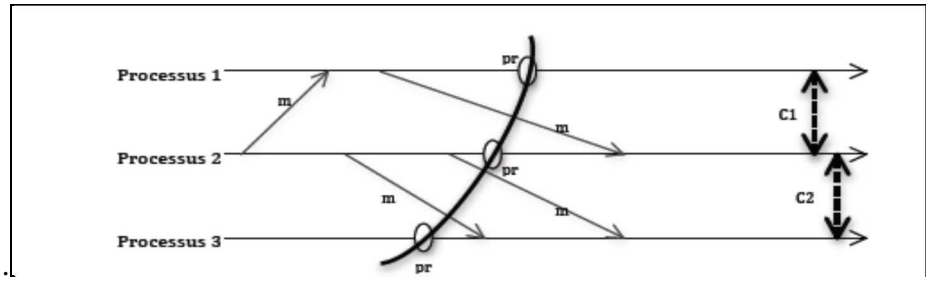

Figure III.14 **Point de reprise coordonné non bloquant de Chandy et Lamport**

- pr1, pr2 et p3 sont les états locaux respectifs des processus P1, P2, P3.
- C1 et C2 représentent les états des canaux de communication entre [P1, P2] et [P2, P3] C1= m3 ; C2=m2+m4.

Le protocole fonctionne sous l'hypothèse des canaux FIFO, c'est-à-dire que sur un même canal tous les messages sont reçus dans l'ordre où ils ont été émis. Pour coordonner la capture des états locaux des processus, l'algorithme utilise des messages de contrôle ou **marqueurs**. Au cours de l'exécution du système, n'importe quel processus peut initier le calcul de l'état global du système. L'objectif est d'enregistrer les états locaux de tous les Messages en transit. Le principe est le suivant :

Soit *P1* l'initiateur du point de reprise :

a) *P1* envoie le marqueur « CtrlMsg » à tous les processus et sauvegarde son état local. (Figure III.15).

b) À la réception de « CtrlMsg », deux cas peuvent se présenter :

b.1) le processus reçoit le marqueur pour la première fois, alors il enregistre son état local et transfère le marqueur sur tous ses canaux sortants. L'état du canal relatif au processus émetteur est vide car, comme les canaux sont FIFO, le processus a reçu tous les messages émis avant la réception du marqueur, et les messages reçus après le marqueur ne font pas partie de l'état global.

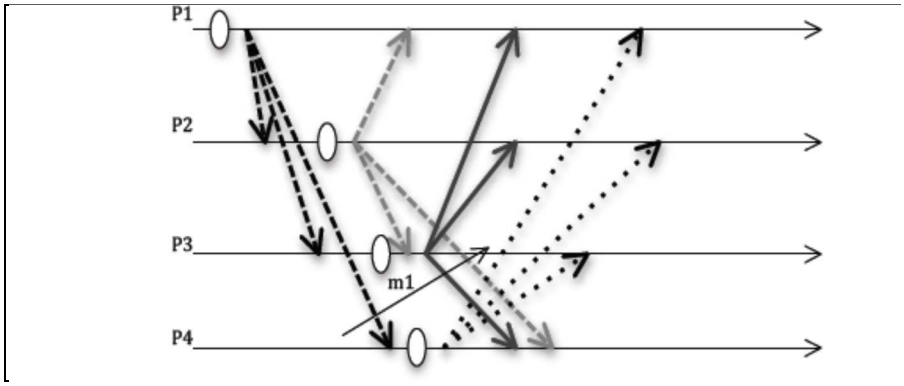

Figure III.15 Transfert de marqueurs vers tous les processus

b.2) le processus a déjà enregistré son état local, suite à une réception antérieure de marqueur, soit de sa propre initiative. L'état du canal entrant relatif au processus émetteur de ce marqueur est égal aux messages reçus entre l'enregistrement du dernier état local et la réception du marqueur. Par exemple, dans la précédente le message m1sera enregistré par P3 car il est reçu avant le marqueur envoyé par P4.

III.2.1.1 Version Algorithme :

L'algorithme 1 présente le traitement des messages pour le protocole coordonné hiérarchique. Cinq types de messages peuvent circuler dans le réseau :

- Les messages de l'application
- Les messages de demande de point de reprise 'ckptRqtMsg'
- Les accusés de réception 'ackRMsg', 'ackFMsg'
- Les messages de validation 'commitMsg'
- Les fautes 'FaultMsg'

➢ Initialement, chaque processus exécute la fonction *initialize* qui permet de fixer la fréquence des points de reprise. Cette fonction, comme son nom l'indique, permet aussi d'initialiser les canaux entrants et sortants des processus (ligne 15), ainsi que

les différents des états des processus pris au cours de l'exécution des applications (lignes 6 et 10).

- Ensuite le processus récepteur fait appel à la fonction *handleMessage*. Cette fonction exécutée à chaque réception de messages permet d'enregistrer les messages reçus par chaque processus dans une structure appelée '*canalMsg*' (lignes 2 et 13). Pour démarrer une session de sauvegarde, un processus initiateur envoie la demande à son leader (ligne 12). À la réception de la demande, le processus leader transfère la demande aux autres leaders (ligne 26) et la diffuse le message dans son cluster (ligne 27).
- Un processus qui reçoit une demande de sauvegarde effectue un point de reprise (ligne 34) si ce dernier ne l'a pas encore fait et envoie un accusé de réception à son leader (ligne 35). Dès que tous les accusés de réception sont reçus, le leader du cluster envoie un message de validation au leader initiateur de la sauvegarde pour clôturer la procédure de sauvegarde de l'état de la grille (ligne 41).
- En cas de faute, un message '*FaultMsg*' est diffusé à tous les noeuds dans la grille pour exécuter la procédure de recouvrement (lignes 50 et 52) détaillée dans **l'algorithme 2**.

Variables
1.etatNoeud.chkpt (* indique si le processus a effectué un point de reprise *)
2.canalMsg (* messages contenus dans les canaux d'entrée et de sortie *)
3.inFault (* indique si le processus fait un recouvrement *)
4.senderMsg (* émetteur du message *)
5.freqCkpt (* indique la fréquence de sauvegarde *)
initialize ()
6.etatNoeud.chkpt=false
10.inFault=false
11.canalMsg=vide
12.envoyer ckptRqtMsg toutes les freqCkpt unités de temps
handleMessage (cMessage *msg)
13.canalMsg=canalMsg+msg
14.si ((ckptRqtMsg ∈canalMsg) ou (FaultMsg ∈canalMsg))
15.alors
16.bloquer les communications
17.sinon
18.délivrer msg à l'application
** *A la réception d'une demande d'un point de reprise* **
19.si (msg= ckptRqtMsg)
20.alors si (processus=leader)
21.alors

```
22.si (senderMsg=initiateur)
25.alors
26.envoyer ckptRqtMsg aux leaders
27.diffuser ckptRqtMsg dans le cluster
28.sinon
29.diffuser ckptRqtMsg dans le cluster
30.sinon
31.si etatNoeud.chkpt=false
32.alors
33.etatNoeud.chkpt=true
34.makeCheckpoint()
35.envoyer (ackRMsg)
36.si (msg= ackRMsg)
37.alors si (nombre(ackRMsg)=(nombre(processus du cluster)))
40.alors
41.envoyer commitMsg à leader(initiateur)
42.sinon
43.nombre(ackRMsg)= nombre(ackRMsg)+1
** A la réception d'une demande de recouvrement **
44.si (msg= FaultMsg)
45.alors si (processus=leader)
46.alors
47.si (senderMsg=initiateur)
48.alors
49.envoyer FaultMsg aux leaders
50.diffuser FaultMsg dans le cluster
51.sinon
52.diffuser FaultMsg dans le cluster
53.sinon
54.inFault= true
55.recovery()
56.envoyer (ackFMsg)
57.si (msg= ackFMsg)
58.alors si (nombre(ackFMsg)=(nombre(processus du cluster)))
59.alors
60.envoyer commitMsg à leader(initiateur)
61.sinon
62.nombre(ackFMsg)= nombre(ackFMsg)+1
```

Algorithme 1: Protocole coordonné non bloquant (chandy lamport)

L'algorithme 2 représente les procédures exécutées respectivement lors de l'établissement d'un point de reprise d'un processus et durant le recouvrement.

```
Variables
1.etatProci
2.inFault (* indique si le processus a fait un recouvrement *)
makeCheckpoint ()
3.vider canaux d'entrée et canaux de sortie
4.etatProci = etat processus i
5.débloquer communication
```

6.etatNoeud.chkpt=false
recovery()
7.etatProci = copie etat processus i
8.inFault =false
9.débloquer communication

Algorithme 2 : Établissement de point de reprise et de recouvrement

III.2.2 Recouvrement hiérarchique :

Le recouvrement obéit aux mêmes règles que le calcul de l'état global avec les mêmes quatre phases:

- *Initialisation du recouvrement* : Le processus fautif envoie un message de demande d'exécution de la procédure de recouvrement à son leader.
- *Coordinations des leaders* : le leader du cluster du processus fautif transfère la demande de recouvrement aux leaders (Algorithme 1 : ligne 49).
- *Recouvrement au sein du cluster* : tous les processus exécutent la procédure de recouvrement après avoir reçu la demande de leur leader (Algorithme 1 : ligne 55).
- *Clôture du recouvrement du système* : chaque leader envoie un message au leader du processus fautif (Algorithme 1 : ligne 56).

III.2.3. Journalisation optimiste :

Comme décrit dans le chapitre précédent, le protocole établit des points de reprise indépendants et sauvegarde les messages sur support stable en utilisant le mécanisme de journalisation optimiste. Nous avons ajouté un nœud supplémentaire au réseau. Ce nœud représente le serveur de stockage stable.

Au cours de l'exécution d'une application, le protocole enregistre les messages dans la mémoire du processus (**Algorithme 3**). Un compteur (ligne 6) permet d'enregistrer le nombre de messages reçus par le processus. Dès qu'un processus reçoit un nombre de messages égal à un certain seuil (ligne 7), la procédure de sauvegarde en mémoire stable est enclenchée. Elle se traduit par l'envoi du message 'stockMsg' à tous les processus de la grille. À la réception du message (ligne 12), tous les messages stockés dans les mémoires des processus commencent à être sauvegardés vers le serveur de stockage.

En respectant la structure de la grille informatique, tous les messages transitent par le leader du cluster avant d'être transférés vers le serveur de stockage.

Variables
1.booléen inFault (* indique si le processus est en cours de recouvrement *)
2.nbMsg=0 (* nombre de messages reçus par un processus *)
3.etatNoeud.chkpt (* indique si le processus a effectué un point de reprise *)
4.logThreshold (* nombre de message à recevoir avant la journalisation *)
5.handleMessage (cMessage *msg)
6.nbMsg=nbMsg+1 ;
7.si (nbMsgVolatilMem = logThreshold)
8.alors
9.envoyer (stockMsg) au leader
10.sinon
11.saveOnVolatilMem ()
12.si ((processus=leader) et (msg=stockMsg))
13.alors
14.diffuser stockMsg dans le cluster
15.sinon
17.saveOnServer ()
*19.** A la réception d'une demande d'un point de reprise **
20.si (msg= ckptRqtMsg)
21.alors si (processus=leader)
22.alors
23.si (senderMsg=initiateur)
24.alors
25.envoyer ckptRqtMsg aux leaders
26.diffuser ckptRqtMsg dans le cluster
27.sinon
28.diffuser ckptRqtMsg dans le cluster
29.sinon
30.si etatNoeud.chkpt=false
31.alors
32.etatNoeud.chkpt=true
33.makeCheckpoint()
34.envoyer (ackRMsg)
35.si (msg= ackRMsg)
36.alors si (nombre(ackRMsg)=(nombre(processus du cluster)))
37.alors
38.envoyer commitMsg à leader(initiateur)
39.sinon
40.nombre(ackRMsg)= nombre(ackRMsg)+1
*41** A la réception d'une demande de recouvrement **
42.si (msg= FaultMsg)
43.alors si (processus=leader)
44.alors
45.si (senderMsg=initiateur)
46.alors
47.envoyer FaultMsg aux leaders
48.diffuser FaultMsg dans le cluster
49.sinon
50.diffuser FaultMsg dans le cluster
51.sinon
52.inFault= true

```
53.recovery()
54.envoyer (ackFMsg)
55.si (msg= ackFMsg)
56.alors si (nombre(ackFMsg)=(nombre(processus du cluster)))
57.alors
58.envoyer commitMsg à leader(initiateur)
59.sinon
60.nombre(ackFMsg)= nombre(ackFMsg)+1
```

Algorithme 3: Journalisation optimiste 'hiérarchique'

III.3 Diagrammes de classes :

La modélisation statique du service proposé est constituée de plusieurs entités, dont l'interaction est présentée à l'aide d'un diagramme de classes d'UML. Un diagramme de classes décrit le type des objets ou données du système ainsi que les différentes formes de relation statiques qui les relient. L'ensemble des classes utilisées sont représentées sur la figure III.16..

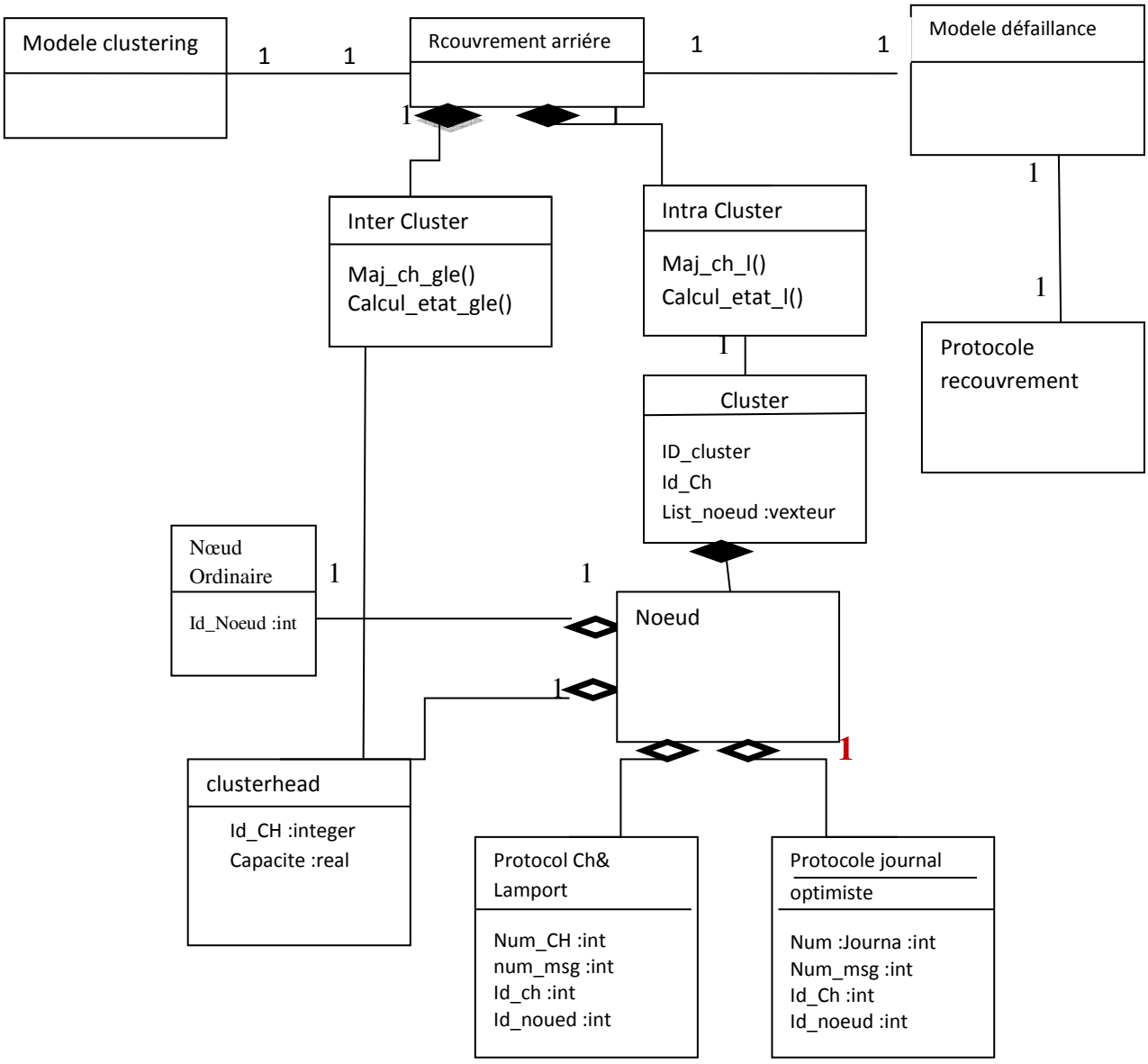

Figure III.16 Diagramme de classe

III.4. Description des combinaisons de protocoles :

Notre approche s'inspire des solutions proposées par les auteurs de **[25][26][27].** Nous allons faire une composition hiérarchique des deux protocoles de recouvrement arrière.

Nous considérons une architecture hiérarchique composée de clusters interconnectés par des liaisons longues distances. Nous distinguons deux types de messages :

- les messages inter-cluster qui circulent entre les clusters de la grille,
- les messages intra-cluster échangés entre les processus d'un même cluster.

Nous étudions, toutes les combinaisons des deux protocoles dont la configuration est résumée dans le Tableau III.5.

Nom Protocole	inter-cluster	Protocole intra-cluster
CH.L&JO : Chandy-Lamport & Journalisation Optimiste»	Chandy.lamport	Journalisation Optimiste
JO&CH Journalisation Optimiste& Chandy-Lamport	Journalisation Optimiste	Chandy.lamport

Tableau III.5 : Combinaisons entre la journalisation optimisste et le protocole de Chandy-Lamport

III.4.1 Chandy-Lamport & Journalisation Optimiste :

Dans cette composition hiérarchique [31], tous les messages échangés au sein de chaque cluster seront sauvegardés en utilisant le protocole de journalisation optimiste **[32]** **(Algorithme 4).**

Chaque processus est identifié par son identifiant unique, l'identifiant de son cluster, le numéro de séquence des messages (lignes 4-5-6).

Pour sauvegarder l'état global cohérent de la grille, c'est le protocole à base de point de reprise non-bloquant de Chandy-Lamport est utilisé.

Les leaders jouent le rôle de coordonnateur au sein de chaque cluster. Ainsi pour sauvegarder l'état global cohérent de la grille, voici les différentes étapes (Figure III.17):

Étape 1 : un processus initiateur envoie un marqueur à son leader et sauvegarde son état local

Étape 2 : cette étape ne fait intervenir que les leaders des clusters. À la réception du marqueur, le leader le transfère aux autres leaders de la grille informatique.

Étape 3 : cette étape est identique à celle de l'exécution du protocole de Chandy-Lamport dans une architecture plate. Chaque leader envoie le marqueur à tous les processus de son cluster. Les processus sauvegardent leur état selon les règles définies dans l'algorithme.

Étape 4 : chaque leader, après avoir sauvegardé son cluster envoie au leader un message de fin de sauvegarde. La fin du calcul de l'état global cohérent de la grille est marquée par la réception de tous les messages de fin par chaque leader.

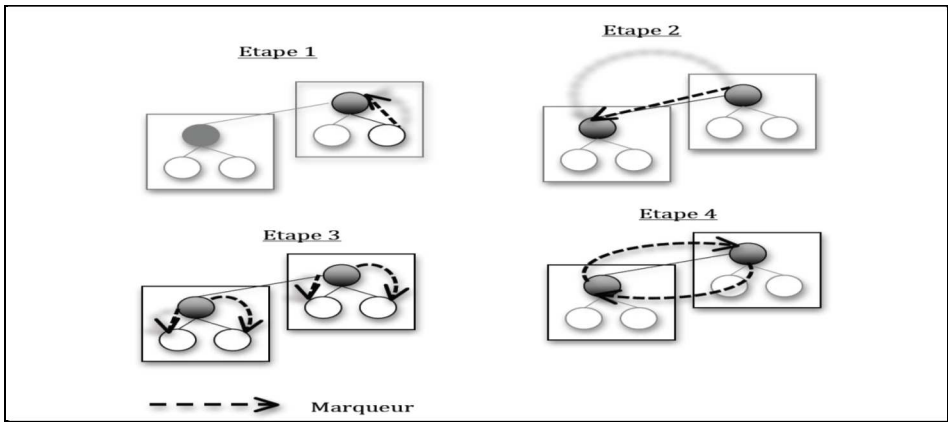

Figure III.17: Protocole de point de reprise non bloquant hiérarchique

III.4.2 Le Recouvrement :

En cas de panne, le processus fautif ainsi que tous les autres processus appartenant à ce cluster redémarrent à partir du dernier point de reprise et rejouent les messages reçus entre le dernier calcul d'état global et l'apparition de la faute.

Un problème se pose lors du recouvrement. Ce dernier doit garantir que tous les messages soient rejoués dans le même ordre où ils ont été reçus.

La Figure III.18 montre un exemple d'exécution dans un système comportant deux clusters (Cluster 1 et Cluster 2).

Le Cluster 1 contient deux processus *P1* et *P2* et les processus *P3* et *P4* appartiennent au Cluster 2. *A, B, C, D, E* et *F* représentent les messages échangés par les processus et qui peuvent être classés selon les deux catégories précitées :

- *A, B, D* et *F* sont des messages intra-cluster
- *C* et *E* sont des messages inter-cluster

Selon les règles de la composition hiérarchique des deux protocoles, seuls les messages *A, B, D* et *F* sont sauvegardés vers le serveur de stockage selon le déclenchement de la procédure de sauvegarde en mémoire stable.

Les messages *C* et *E* ne sont pas sauvegardés. En cas de panne du processus *P2, P1* et *P2* doivent reprendre leur exécution à partir de leur dernier point de reprise, et les messages *A, D,*

C et *E* doivent être rejoués. Le recouvrement ne sera pas correct puisque les messages inter-cluster *C* et *E* seront perdus.

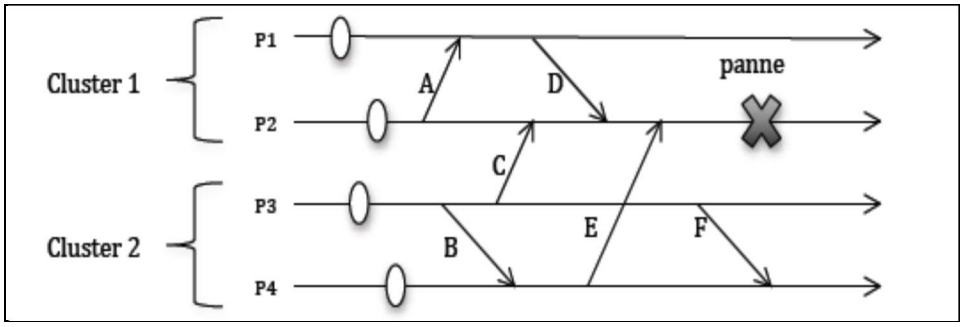

Figure III.18 Recouvrement avec protocole Chandy.Lamport & Journalisation Optimiste

Variables

```
1.etatProci (* etat processus i*)
2.ecIni (* etat canal d'entrée processus i*)
3 .ecOuti (* etat canal de sortie processus i*)
4.rsn (* numéro de séquence *)
5.idProc (* identifiant du processus *)
6.idTeam (* identifiant du cluster *)
7.etatNoeud.chkpt (* indique si le processus a effectué un point de reprise *)
8.nbMsg=0 (* nombre de messages reçus par un processus *)
9.logThreshold (* nombre de message à recevoir avant la journalisation *)
handleMessage (cMessage *msg)
9.si (msg=CtrlMsg)
10.alors
11.si ((processus=initiateur) et (etatNoeud.chkpt=false))
12.alors
13.makeCheckpoint()
14.envoyer msg au leader
15.sinon
16.si ((processus<>leader) et (etatNoeud.chkpt=false))
17.alors
18.makeCheckpoint()
19.envoyer msg vers les canaux de sortie
20.si ((processus=leader) et (etatNoeud.chkpt=false))
21.alors
22.makeCheckpoint()
23.diffuser msg dans le cluster
24.si ((msg<>CtrlMsg) et si (processus=leader)
 25.alors
 26.si (senderMsg=initiateur)
27.bloquer l'exécution
28.rsn=rsn+1
29.envoyer (ack, rsn, initiateur)
30.si ((msg=ack) et (processus=initiaateur)
31.alors
32.ajouter rsn au message
```

```
33.envoyer (acquittement, initiateur)
34.débloquer communication
35.si ((msg<>CtrlMsg) et  si (nbMsgVolatilMem = logThreshold)
36.alors
37.envoyer (stockMsg) au leader
38.sinon
39.saveOnVolatilMem ()
40.si ((processus=leader) et (msg=stockMsg))
41.alors
42.diffuser stockMsg dans le cluster
43.sinon
44.saveOnServer
```

Algorithme 4 : Protocole Chandy.Lamport & Journalisation Optimiste

III.4.3 Journalisation Optimiste &Chandy Lamport:

Il s'agit ici d'utiliser la journalisation optimiste entre les clusters, et le protocole de point de reprise coordonné non bloquant de Chandy-Lamport au sein des clusters (Figure III.19) **[28][31].**

Chaque message échangé entre deux processus appartenant à deux clusters différents est sauvegardé dans la mémoire du serveur de stockage en respectant la technique de sauvegarde. Pour garantir le déterminisme d'exécution, les déterminants des messages feront partie des éléments stockés au niveau du serveur. Ce déterminant contient la date d'émission du message, sa date de réception, et son numéro de séquence. Ainsi durant le recouvrement, ces informations pourront être utilisées pour rejouer les messages dans l'ordre où ils ont été émis au cours de l'exécution normale.

Selon l'architecture hiérarchique définie au précédent, tous les échanges inter-cluster passent par l'intermédiaire des leaders qui représentent les coordonnateurs des clusters.

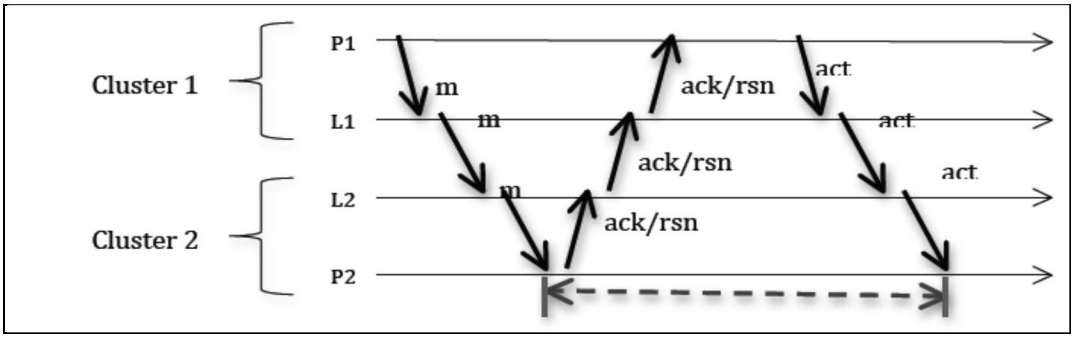

Figure III.19 Journalisation optimiste

La Figure III.32 montre la procédure de sauvegarde du message ***m*** échangé entre les processus *P1* et *P2* situés sur deux clusters différents. Les processus *P1* et *P2* appartiennent

respectivement au Cluster 1 et Cluster 2, *L1* et *L2* étant leur leader. Voici les différentes étapes pour envoyer le message *m* de *P1* vers *P2*:

- *P1* envoie m à son leader *L1*
- *L1* transfère le message *m* à *L2*, leader du processus *P2*
- *L2* envoie *m* au destinataire du message *P2*
- À la réception de *m*, *P2* met à jour le numéro de séquence du message, l'attache à l'accusé de réception, et envoie ce dernier son leader *L2*
- Le leader *L2* transfère l'accusé de réception à son homologue *L1*
- Enfin *L1* envoie l'accusé de réception à l'émetteur *P1*
- Le processus émetteur *P1* ajoute le numéro de séquence au message *m* sauvegardé dans la mémoire
- *m* étant sauvegardé en mémoire avec son numéro, le *P1* peut maintenant envoyer un acquittement au récepteur pour qu'il continue à recevoir des messages de tel sorte que le nombre de messages reçus ne dépasse pas un certain seuil ainsi la sauvegarde en mémoire stable est enclenchée.

Au sein des clusters, les états des processus seront sauvegardés en utilisant la technique non bloquante de Chandy-Lamport. L'intérêt de l'utilisation de cette solution non bloquante en intra-cluster est que pendant son exécution le protocole de journalisation s'exécute en même temps sur les messages inter-cluster. Ces messages sauvegardés peuvent ne pas faire partie du calcul de l'état global du système, puisqu'ils seront rejoués en cas de panne.(Figure III.20).

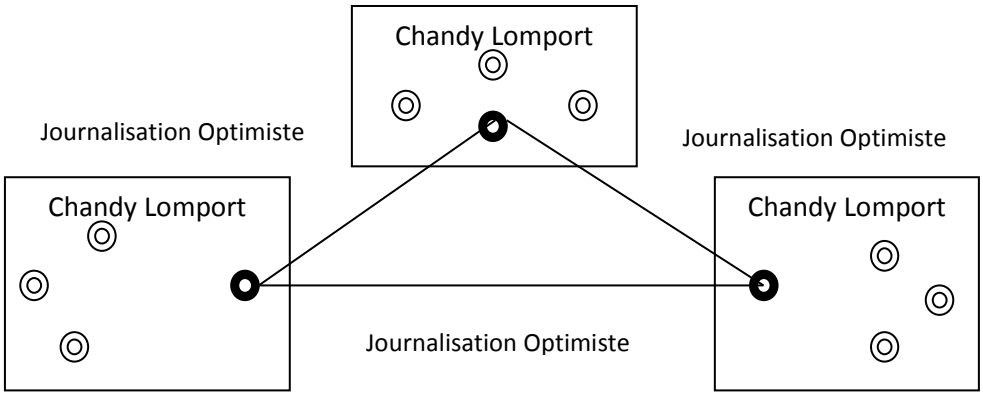

Figure III.20 : Composition hiérarchique

En cas de défaillance, seuls les processus du cluster contenant le processus fautif redémarrent à partir du dernier point de reprise et rejouent les messages échangés après la dernière sauvegarde. Dans **[28]** les auteurs montrent que le recouvrement ne sera pas correct à cause

des messages intra-cluster qui ne sont pas sauvegardés. En effet, si des messages sont échangés entre processus d'un même cluster avant une panne, ces messages seront tout simplement perdus. C'est pour cette raison, qu'il propose la sauvegarde de ces messages dans une structure appelée *TeamTable* stockée aussi dans la mémoire des processus et contenant les déterminants des messages. Les auteurs de **[30]** proposent aussi la sauvegarde des déterminants de tous les messages, c'est-à-dire aussi bien les messages inter-cluster et intra cluster.

Conclusion :

Dans ce chapitre, nous avons fait une étude approfondie du protocole de journalisation Optimiste et de la sauvegarde à base de point de reprise coordonné non bloquant de Chandy et Lamport (CH.L).

Afin de les utiliser dans notre architecture hiérarchique en grille, nous avons déterminé la combinaison possible à savoir :

- CH.L&JO : avec l'utilisation du protocole coordonné de Chandy-lamport au niveau de chaque cluster et une journalisation optimiste pour les messages inter-cluster
- JO&CH.L : avec une journalisation optimiste pour les messages inter-cluster et l'utilisation du protocole coordonné de Chandy-Lamport au niveau de chaque cluster.

Le prochain chapitre sera consacré à la partie implémentation et évaluation des algorithmes et de tolérance aux fautes.

L'objectif de toute conception est de produire un logiciel pour confirmer ou infirmer les déclarations théoriques. De ce fait, pour valider et évaluer notre proposition, nous avons conçu un simulateur reflétant le fonctionnement de l'approche proposée. Nous avons effectué une série d'expérimentations dont les résultats et les interprétations font l'objet du présent chapitre.

Nous commencerons d'abord par décrire l'environnement dans lequel nous avons réalisé notre simulateur, puis nous discuterons et analyserons les résultats obtenus.

VI.1 Environnement d'expérimentation :

L'environnement dans lequel nous avons réalisé notre simulateur et obtenu les différentes expérimentations est défini par les éléments suivants :

1. Environnement matériel et logiciel : Nous avons développé notre application sur une machine fonctionnant sous le système Windows (Windows XP), avec les caractéristiques suivantes :

- Intel Core I3,
- 4.0Ghz
- Capacité de mémoire 4Go de RAM.

2. Environnement de développement : L'application est développée par le langage de programmation NetBeans 7.4 et simulateur Optorsim

VI.2.Optorsim :

OptorSim [**31**], un simulateur développé dans le cadre du projet européen DataGrid [32]. Il s'agit d'un simulateur conçu pour tester différents algorithmes de réplication et de gestion de données au sein de la grille DataGrid.

VI.2.1 Fonctionnement global :

La figure VI.21 montre les principaux éléments qui constitue la grille simulée. Cette grille est composée de sites. Un site contient une unité de calcul appelé Computing Element (CE) , un gestionnaire de réplicats, le Replica Manager (RM) ainsi qu'une unité de stockage de données, le Storage Element(SE) . En dehors de ces sites se trouvent deux autres entités : l'Utilisateur, qui génère les requêtes et les soumet à l'ordonnanceur de la grille, le Resource

Broker(RB).(voir Figure VI.1)

- Computing Element (CE) : il simule une machine seule ou une grappe de machines.
- Il a pour rôle de simuler l'exécution des requêtes qui lui sont soumises. Lorsqu'un CE s'apprête à exécuter une requête il s'adresse au Replica Manager afin de récupérer les différentes données nécessaires à l'exécution. Un CE est défini par le nombre de nœuds qui le compose. La durée d'exécution des requêtes est identique pour toutes les requêtes quel que soit leur type, les données auxquelles elle accède ou le CE sur lequel elle est exécutée. Cette durée est fixée dans les fichiers de configuration du simulateur. Lors de la simulation de l'exécution d'une requête, ce temps fixe est divisé par le nombre de nœuds du CE.

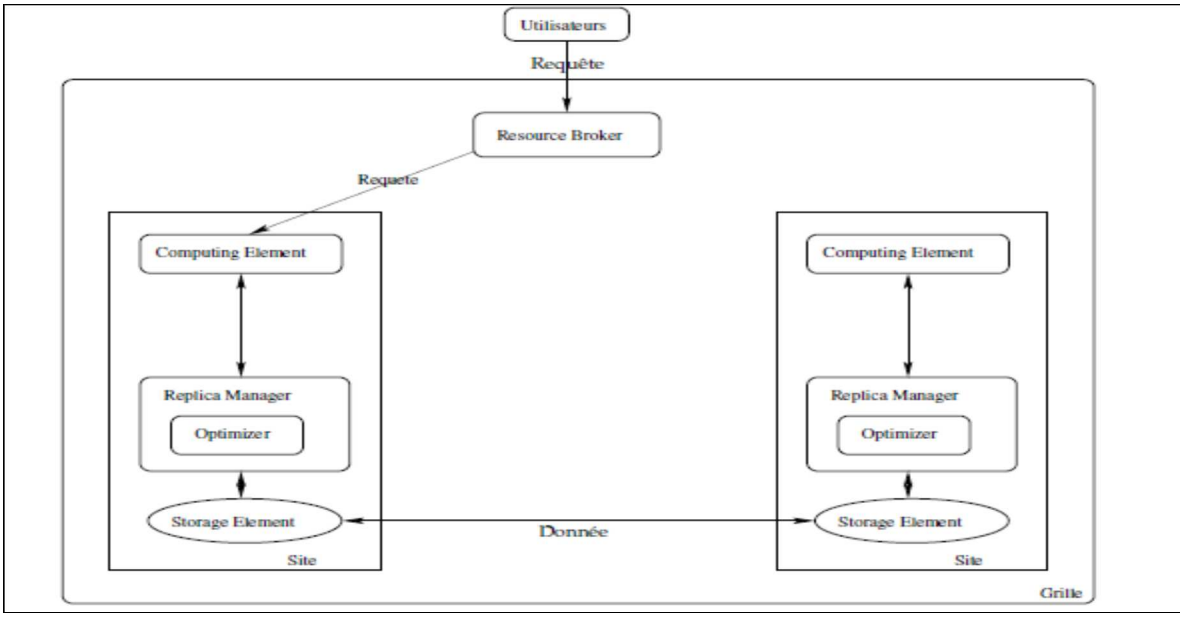

Figure VI.1 : Architecture fonctionnel d'optorsim

- Replica Manager (RM) : il y en a un par site et il a en charge la localisation et la gestion des mouvements des données. Le RM contient un Optimizer , il s'agit d'un objet dans lequel sont définies les stratégies de réplication et de suppression de données. Ce que les auteurs d'OptorSim appellent stratégies de réplication sont en fait la façon dont sont calculés les coûts de transfert d'une donnée, la façon dont est choisie la source d'un transfert ainsi que le choix des données à supprimer sur le SE lorsque cela est nécessaire.
- Storage Element (SE) : il a la charge du stockage et est caractérisé par sa capacité de stockage (en Mégaotets). Un même fichier peut être stocké sur différents SE au même

moment, et chaque SE a la charge de décider quel fichier doit être supprimé si de l'espace est requis pour un nouveau fichier alors que le SE est déjà plein. Pour chaque fichier, il est possible de définir au moins une copie maître qui ne peut être supprimée du serveur où elle se trouve. La façon dont les copies maîtres sont distribuées est définie par une option de configuration.

- Utilisateur : il est la source des requêtes. C'est lui qui crée les requêtes selon un schéma bien précis. Là encore, divers schémas sont fournis et le choix de celui qui sera utilisé est fait dans la configuration.
- Resource Broker (RB) : c'est l'organe central de décision. Il est capable de communiquer avec les différents CEs et RMs pour récupérer toutes les informations nécessaires à l'ordonnancement. Plusieurs algorithmes sont disponibles pour effectuer
- l'ordonnancement, le choix de celui qui doit être employé est défini dans les fichiers de configuration du système

VI.3 Pourquoi JAVA?

Java est un langage de programmation orienté objet et un environnement d'exécution récent, développé par Sun Microsystems en 1991, il était présenté officiellement en 19954. Java est devenu aujourd'hui un langage incontournable dans le monde de la programmation, parmi les différentes caractéristiques attribuées à son succès :

- L'indépendance de toute plate-forme : le code reste indépendant de la machine sur laquelle il s'exécute.
- Il est possible d'exécuter des programmes Java sur tous les environnements qui possèdent une Java Virtual Machine.
- Java est également portable, permettant à la simulation d'être distribuée facilement sans avoir à recompiler le code pour les différents systèmes.
- Le code est structuré en plusieurs classes, dont chacune traite une partie différente de la simulation.
- Java est multitâche : il permet l'utilisation de threads qui sont des unités d'exécution isolées.

VI.4 Métriques utilisées :

Nous avons utilisé un ensemble de métriques pour mesurer les performances de notre approche de tolérance aux fautes.

VI.4.1 Temps de réponse (temps d'exécution) :

Temps de réponses = *T*1+*T*2+*T*3 (4.1)

- *T*1 : est le temps de transfert de la donnée. (le message)
- T2 : est le temps d'accès au support de stockage local.
- *T*3 : est le temps d'attente dans la file d'attente.

Cette formule est appliqué dans le cas de protocole coordonné non bloquant et journalisation optimiste avec et sans fautes

VI.4.2. Nombre de processus à redémarrer :

$Nbr_{\_redémarrer\_grille} = processus_{\_grille} - processus_{\_défaillant}$ (intra cluster)

$Nbr_{\_redémarrer\_cluster} = processus_{\_cluster} - processus_{\_défaillant}$ (inter cluster)

VI.4.3. Performance durant le recouvrement :

$A_{\_Perform\_grille} = Nbre_{\_redémarrer\_grille} / Nbre_{\_retour\_arriére}$ (intra cluster)

$A_{\_Perform\_cluster} = Nbre_{\_redémarrer\_clustere} / Nbre_{\_retour\_arriére}$ (intra cluster)

VI.5 Diagramme de cas d'utilisation :

Le diagramme de cas d'utilisation est une modélisation des fonctionnalités du système. Une fonctionnalité se détermine en observant et en précisant acteur par acteur les séquences d'interaction avec le système. Les cas d'utilisation décrivent les fonctions du système selon le point de vue des utilisateurs.

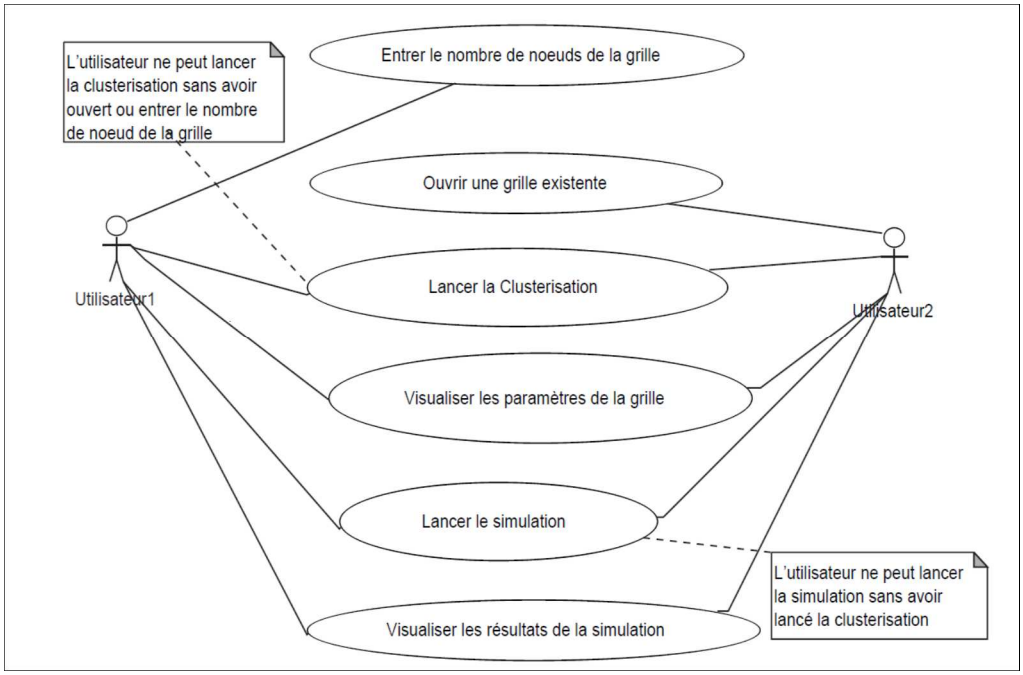

Figure VI.2 Diagramme de cas d'utilisation du simulateur

VI.6 Accès à l'application :

La figure VI.3 illustre la première interface de notre simulateur qui apparaît à l'utilisateur, elle symbolise la représentation de la grille qui est formée d'un ensemble de nœuds indépendants et hétérogènes.

La barre de menu contient deux menus (*Fichier* et *Aide*). Chaque menu est composé d'items.

VI.6.1 Nouvelle configuration de la grille :

Dans le menu *Fichier*, l'utilisateur accède à une nouvelle configuration de la grille par l'item *Nouveau* ce qui donne un nouveau cadre pour introduire le nombre de nœuds pour une simulation donnée (Figure VI.3).

Figure VI.3 – Nouvelle configuration de la grille

VI.6.2 Ouverture d'une configuration existante :

Si un utilisateur veut choisir une grille qui existe déjà (Figure VI.4) en cliquant sur le menu *Fichier* le sous menu *Ouvrir*, une fenêtre de dialogue s'ouvre pour choisir une grille existante.

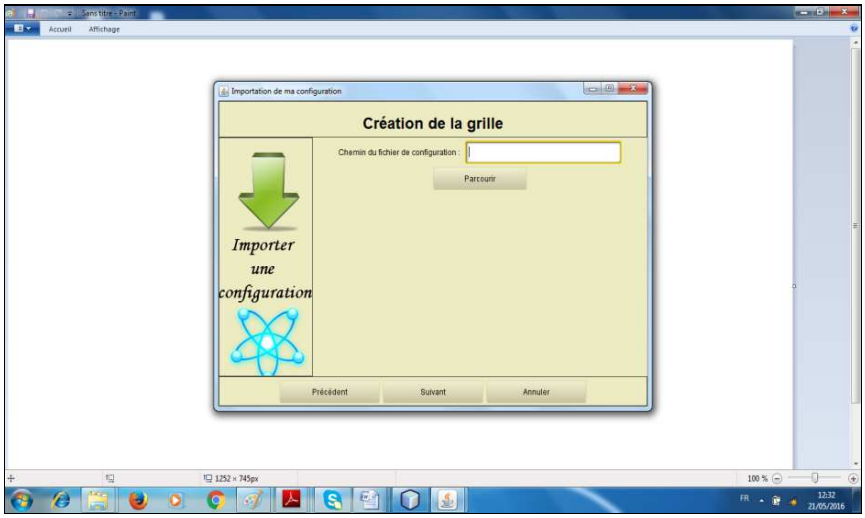

Figure VI.4 Ouverture d'une grille existante

VI.6.3 Création de la grille :

Avant de lancer la simulation, il faut appliquer l'algorithme de Clustering. Lorsqu'on clique sur le bouton *suivant* dans la fenêtre création de la grille, on introduit le nombre de cluster ainsi les nœuds composant et le reste des paramètres tels que débit bande passante, la vitesse de processus et la capacité de stockage des nœuds.(figure VI.5).

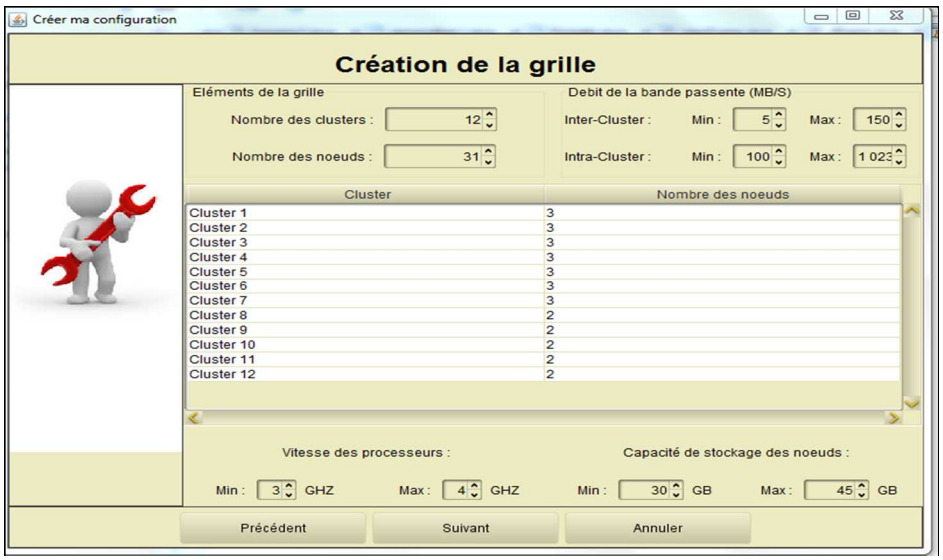

Figure VI.5 Configuration de la grille après le processus de clustering

VI.6.4 Affichage des différentes configurations :

Si on veut visualiser les différentes configurations obtenues après le processus de clustering, on clique sur le bouton *couche inférieur , supérieur* après avoir cliqué sur le bouton *Suivant* de la fenêtre précédente(figure VI.6 & VI.7& VI.8).

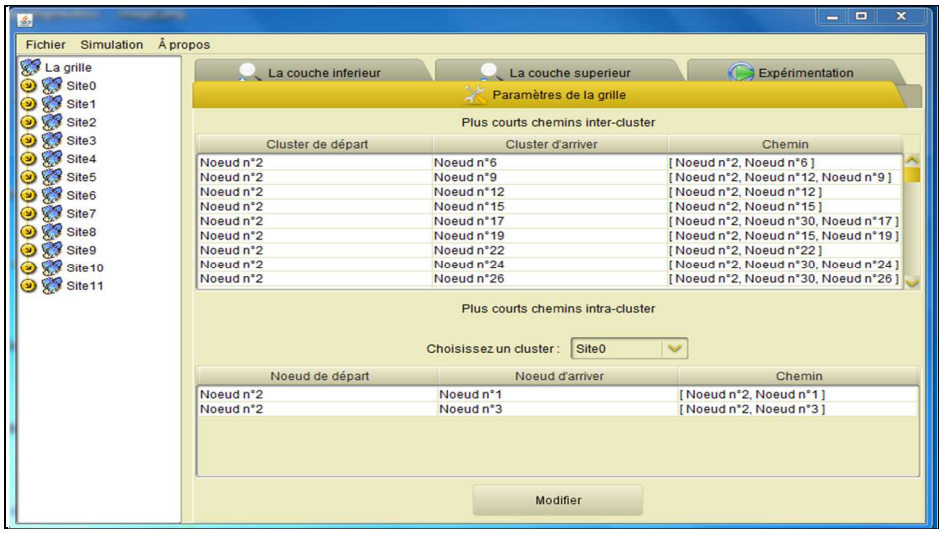

Figure VI.6 différents sites de la grille

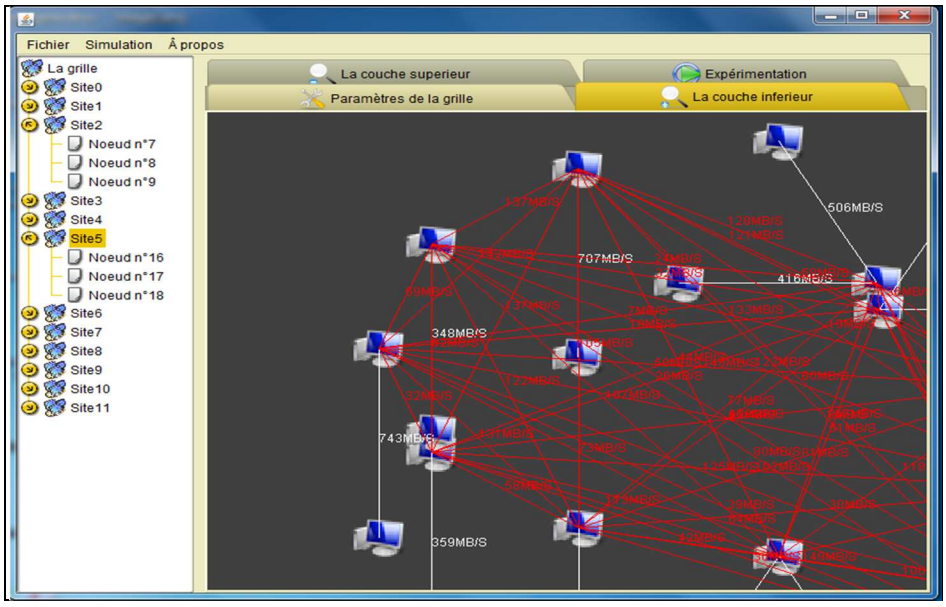

Figure VI.7 : Couche inférieur de la grille

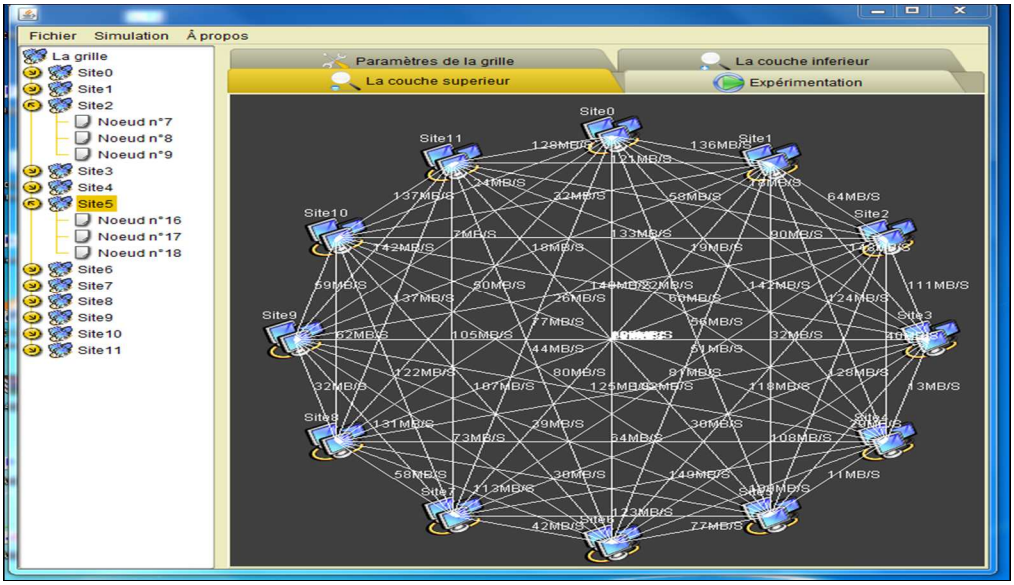

Figure VI.8 : Couche Supérieur de la grille

VI 4.6.5 Simulation

Pour commencer la simulation, l'utilisateur appuie sur le bouton *Lancer simulation* figure VI.9 & figure VI.10 après l'introduction les différents paramètres de la grille, on clique sur le bouton *expérimentation* et on aura la fenêtre suivante qui contient les différents paramètres de l'algorithme recouvrement arrière suivant :

- Sélectionner le cluster dont le nœud initiateur pour effectuer le point de reprise non bloquant
- Introduire le seuil de messages pour la sauvegarde de la journalisation optimiste
- Sélectionner le cas sans fautes
- Choisir le scenario approprié

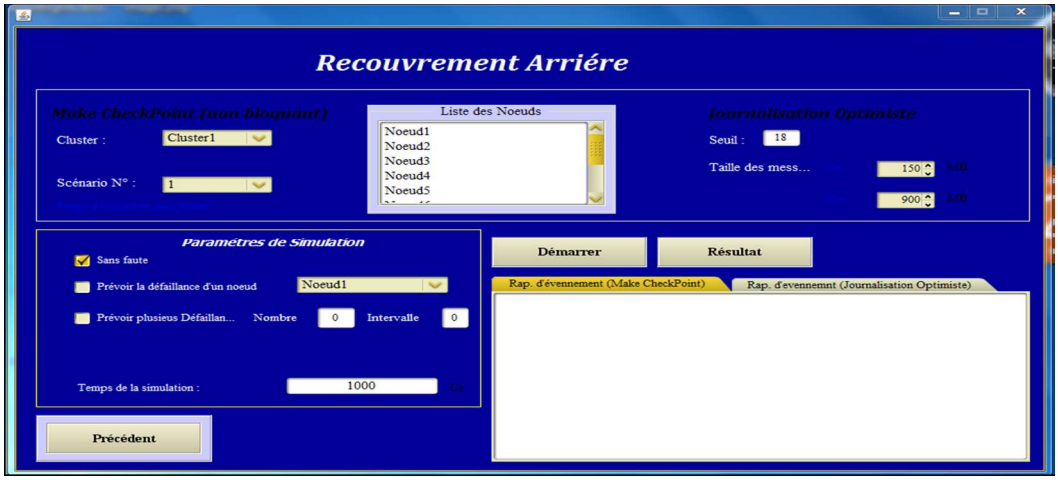

Figure VI.9 : recouvrement arrière sans fautes

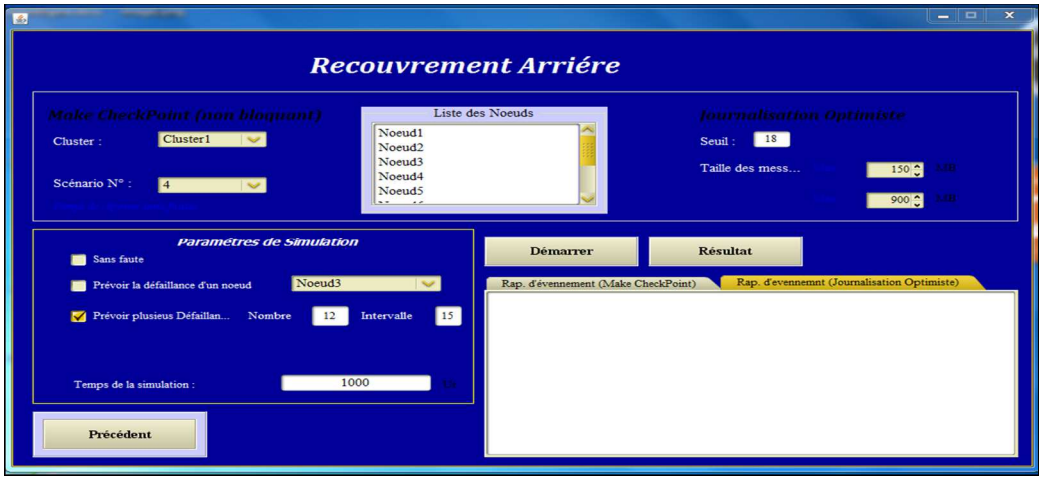

Figure VI.10 : recouvrement arrière avec fautes

Paramètre de la simulation :

Paramètres de simulation	Valeur
Nombre de clusters	[1.. 50]
Nombre de noeuds	[1.. 50]
Nombre de messages	[1 ..100]
Bande passante_inter cluster	[5M..300M]
Bande passante_intra cluster	[1M..2000M]

Scenario 1 : Exécution sans faute :

La simulation est effectuée sur une grille composée de cluster et 34 nœuds répartis sur 12 clusters, le temps de la simulation est 1000 secondes.

La Figure VI.11 présente le temps d'exécution dans les deux types d'architecture. En exécution normale, La technique de Chandy-Lamport présente de meilleures performances parce que même si la sauvegarde est coordonnée, l'exécution n''est pas bloquée durant la procédure.

Cependant, le temps d'exécution est plus élevé en mode cluster qu'en mode inter cluster . En effet, comme le protocole utilise des marqueurs pour coordonner la sauvegarde, ces derniers sont envoyés à tous les processus, qui à leur tour envoient des acquittements à tous les autres processus. Dans ce cas, c'est des messages supplémentaires qui seront envoyés par chaque processus, tandis qu'en mode inter cluster les processus envoient des acquittements uniquement à ceux appartenant au même cluster.

D'après les résultats obtenus sur les deux architectures, l'utilisation des protocoles

coordonnés présente un avantage certain dans les réseaux inter cluster, car la coordination se limite aux processus appartenant à un même cluster, ceci avant que les leaders finalisent la procédure. Tandis que dans l'architecture intra cluster, si le nombre de processus augmente, les messages supplémentaires envoyés pour assurer la coordination globale augmentent considérablement et ralentissent l'application.

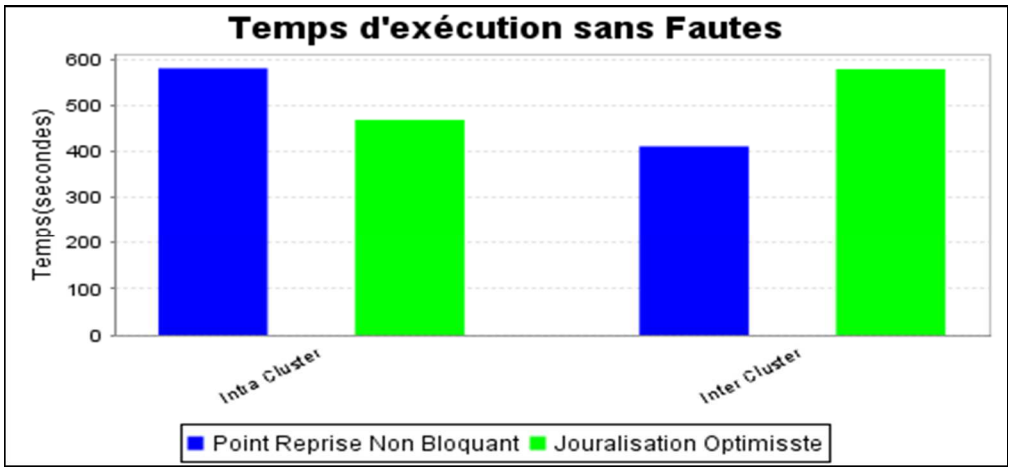

Figure VI.11 : Exécution sans faute

Scenario 2 : Nombre de processus à redémarrer :

Nous avons injecté deux fautes après chaque 150s de simulation, en intra et inter cluster. Les processus reprennent alors leur exécution pour les cas de protocoles coordonnés. Les versions hiérarchiques présentent un avantage certain au niveau inter cluster, puisque seuls les processus du cluster contenant le processus reprennent leur exécution.

Le mécanisme de journalisation d'autres processus différents du processus fautif peuvent reprendre leur exécution à cause des dépendances durant l'enregistrement de leurs états locaux. Voir figure VI.12.

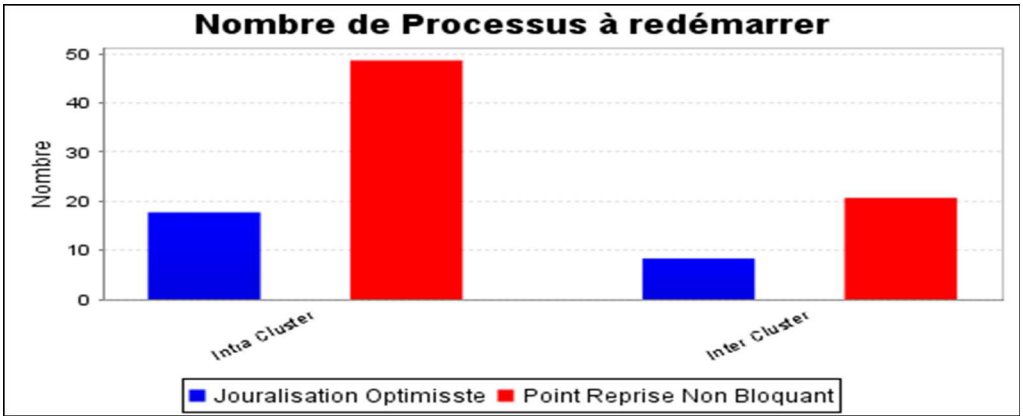

Figure VI.12 : nombre de processus à redémarrer

Scenario 3. Performance durant le recouvrement :

La durée du recouvrement dépend du nombre de processus à reprendre et du nombre de retours arrière. Dans les cas de la sauvegarde à base de points reprise coordonnés non bloquant ('Chandy-Lamport') et de la journalisation optimiste.

Le recouvrement est simplifié parce que le système effectue un retour arrière uniquement depuis le dernier point de reprise maintenu par le protocole. En mode inter cluster, Cependant, si les communications inter-cluster sont intensives, ce surcoût augmentera dans le cas de journalisation optimiste, à cause de la latence inter-cluster élevé (100ms).(voir figure VI.13).

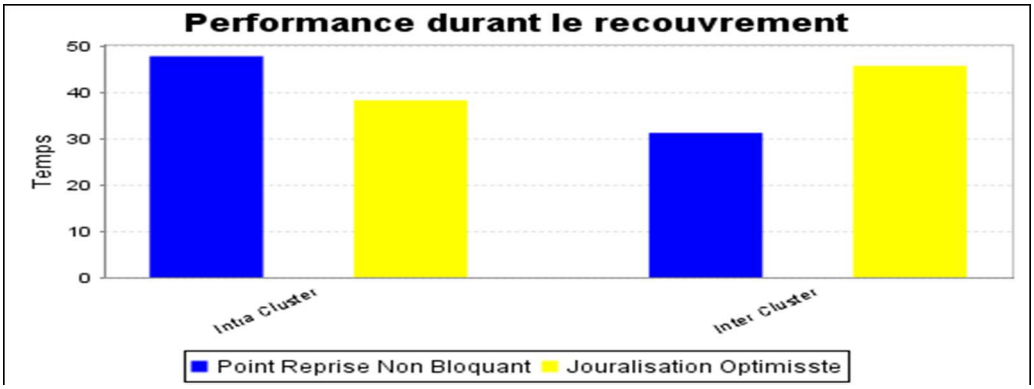

Figure VI.13 : performance durant le recouvrement

Scenario 4 Exécution sans faute :

Les points de reprise sont effectués sans injection de fautes toutes les 150 secondes au cours de l'exécution normale.

On remarque un temps de réponse assez élevé pour le protocole Point reprise non bloquant & Journalisation optimiste. Car les messages intra-cluster sont plus nombreux que les messages inter-cluster, la sauvegarde synchronisée de tous les messages intra-cluster introduit un surcoût sur le temps de réponse, quelle que soit l'application en cours d'exécution.

Le temps de réponse est faible pour la combinaison de protocoles « Journalisation optimiste & Point Reprise Non bloquant » à cause du nombre important de message circulant au niveau de chaque cluster, le nombre de message inter-cluster se limitant uniquement aux messages échangés entre les leaders. (Figure VI.14)

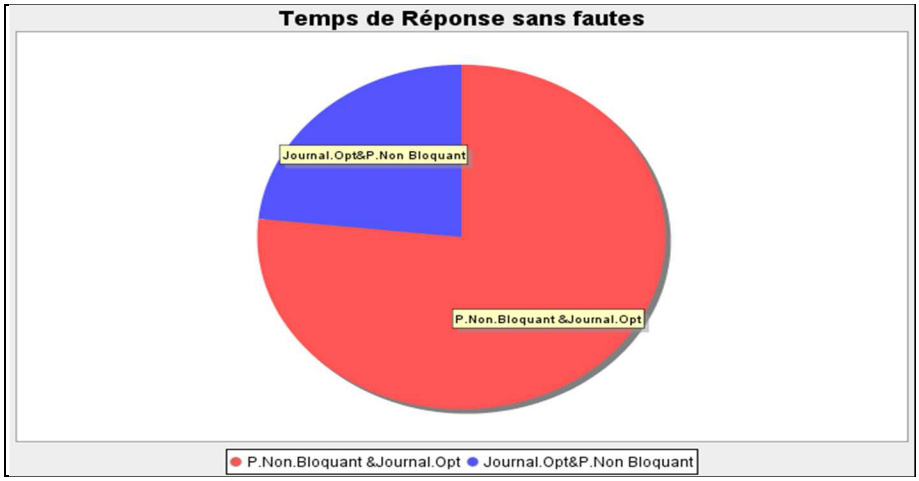

Figure VI.14 Temps de Réponse sans fautes dans la combinaison

Scenario 5 Exécution avec injection de fautes cas de combinaison de protocoles:

Nous étudions dans cette section l'impact des fautes sur le temps de réponses. Dans cette étude, nous avons injecté une faute toutes les 100 secondes avec un temps 1000secondes

la Figure VI.15 montre que Le protocole « Journalisation optimiste &Point reprise non bloquant» est mieux adapté aux mode cluster par apport à Point reprise non bloquant&Journalisation optimiste .Le surcoût induit dans le cas de combinaison de protocoles « JO&Point Reprise non bloquant » n'est pas élevé à cause de la clusterisation des techniques de recouvrement arrière.

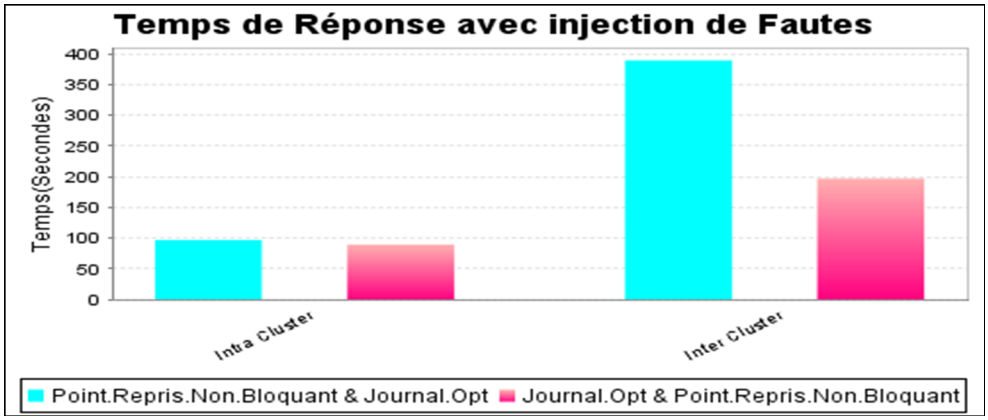

Figure VI.15 Temps de réponse avec fautes dans la combinaison

Conclusion :

Dans ce chapitre, nous avons implémenté les différentes combinaisons possibles de ces protocoles sur une architecture hiérarchique de grille.

La combinaison CH.L& JO et la combinaison JO & CHL (CH.L :chandy lamport, JO :journalisation optimiste).

Les résultats ont montré que dans le cas (intra cluster), le protocole de sauvegarde coordonné non bloquant introduit un faible surcoût car il profite alors pleinement de la clusterisation en diminuant le nombre de messages de contrôle, chaque cluster effectuant sa sauvegarde en parallèle.

Le protocole de journalisation présente un avantage dans le cas de communication d'envoi de messages en inter cluster.

Les combinaison appropriés sont Journalisation Optimiste & Chandy lamport en appliquant pour la sauvegarde des messages inter cluster la journalisation optimiste et l'utilisation du protocole coordonné non bloquant (Chandy-Lamport) au niveau de chaque cluster.

La combinaison Journalisation optimiste & Chandy Lamport présente à avantage par apport à la combinaison Chandy lmport & Journalisation optimiste.

Les grilles informatiques sont devenues des plateformes de partage de ressources très utilisées par l'industrie et le monde académique. On trouve des grilles opérationnelles de quelques dizaines à plusieurs centaines de milliers de serveurs, fonctionnant comme support privilégié en calcul intensif, le traitement de grandes masses de données distribuées.

Une grille est une infrastructure matérielle et logicielle qui fournit un accès normalisé et sécurisé à ses ressources. Composée de grappes nœuds (des clusters) de calcul ou de stockage, reliées par des réseaux de type WAN. Cependant, le nombre élevé de nœuds entraine une augmentation importante des défaillances.

La tolérance aux fautes devient alors un élément indispensable pour assurer la continuité du service. Le recouvrement arrière et la réplication sont les deux solutions pour assurer la tolérance aux fautes dans les grilles informatiques. Dans le cadre d'applications de calcul intensif, la duplication des traitements est trop coûteuse.

Dans cette thèse, nous nous sommes donc concentrés sur les techniques de recouvrement arrière. Il existe deux grandes familles de mécanismes de recouvrement arrière : la sauvegarde synchronisée par point de reprise et les protocoles de journalisation. Ces algorithmes sont utilisés dans des architectures inter et intra cluster.

Les résultats ont montré que les mécanismes de journalisation présentent un avantage pour les environnements à large échelle parce qu'en cas de faute seul le processus fautif reprend son exécution.
Cependant, ils sont particulièrement coûteux lorsque les applications échangent beaucoup de messages. Les algorithmes de sauvegarde coordonnée supportent des applications fortement communicantes.
En revanche, ils passent mal à l'échelle car en cas de défaillance, tous les processus peuvent amenées à redémarrer.
Suite à cette étude, nous avons sélectionné la journalisation optimiste et la solution non bloquante à base de point de reprise de Chandy-Lamport

En perspectives de recherche :

- L'impact de fréquence des messages sur les protocoles.
- Etudier la tolérance aux pannes dans les systèmes répartis tels que les Cloud qui sont l'évolutions des grilles informatiques.

Merci pour la lecture

Bibliographie :

[1]:Foster and C. Kesselman. The Grid 2: Blueprint for a New ComputingInfrastructure. The Morgan Kaufmann Series in Computer Architecture andDesign Series. Elsevier Science, 2003.

[2]:K. Budati, J.D. Sonnek, A. Chandra, and J.B. Weissman. Ridge : combiningreliability and performance in open grid platforms. In HPDC, pages 55_64,2007.

[3]:R. Buyya. Economic-based Distributed Resource Management and Scheduling for Grid Computing. PhD thesis, Monash University, Melbourne, Australia, April 2002.

[4]:I. Foster. What is the grid ? a three point checklist. GRIDtoday, 1(6), 22 July 2002.http ://www.gridtoday.com/02/0722/100136.html.

[5]:J. Postel and J. Reynolds. File transfer protocol (ftp), 1985.

[6]:S. Shukla and A. Deshpande. Ldap directory services- just another databaseapplication (tutorial session). SIGMOD Rec., 29(2) :580, 2000.

[7]:T. Berners-Lee, R. T. Fielding, and H. Frystyk Nielsen. Hypertext transferprotocol - http/1.0. Technical report, 1994.

[8]: F. Berman, A. J. C. Hey, and G. C. Fox. Grid Computing : Making the GlobalInfrastructurea Reality. Wiley Sons, 2003.

[9]:I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtualorganizations. 15(3):200{222, August 2001

[10]:A. Avizienis, J. C. Laprie, and B. Randell. Dependability and its threats – A taxonomy. In Building the Information Society, IFIP 18th World Computer Congress, Toulouse, France, pages 91,120(2004).

[11]:J. Arlat, Y. Crouzet, and Y. Deswarte. Encyclopédie edie de l'Informatiqueet des Systèmes d'Information, chapitre : Tolérance aux fautes. Vuibert, 2006

[12]:X. Besseron. Tolérance aux fautes et reconfiguration dynamique pourles applications distribuées à grande échelle. PhD thesis, Université de Grenoble, France, 2010. (Cite pages 21, 23 et 30.)

[13] : J. C. Laprie. Dependable computing : Concepts, challenges, directions. In COMPSAC, 2004.

[14]:I. Gupta, T. D. Chandra, and G. S. Goldszmidt. On scalable and efficient distributed failure detectors. PODC '01, pages 170_179, USA, 2001. ACM.

[15]:O. Peres. Construction de topologies auto stabilisante dans les systémes à grande échelle. PhD thesis, Université Paris-sud, France, 2008. (Cité page 33)

[16] : E. Badidi. Architectures et Services pour la Distribution de Charge dans les Systèmes distribués Objet. PhD thèses, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Canada, Mai 2000.

: J. Garmany and R. Freeman. Multi-master replication conict avoidance and resolution.Select Journal, independent Oracle Users Group, 11(4):9{15, December 2004)

[17][29]: Ndeye Massata Ndiaye : thèse doctorat « Techniques de gestion des défaillances dans les grilles informatiques tolérantes aux fautes » Université Pierre et Marie Curie Paris VI, 2013.

[18][19]: E. N. (Mootaz) Elnozahy, Lorenzo Alvisi, Yi-Min Wang, David B. Johnson, « A Survey of Rollback-Recovery Protocols in Message- Passing Systems », ACM Computing Surveys, Vol. 34, No. 3, September 2002.

[20]: Briatico D., Ciuffoletti A., Simoncini L., « A distributed domino-effect free recovery algorithm », *IEEE International Symposium on Reliability, Distributed Software, and Databases*, 1984

[21]: Robert H. B. Netzer and Jian Xu, « Necessary and sufficient conditions for consistent global snapshots », IEEE Transactions on Parallel and Distributed systems. VOL.6 NO. 2, February 1995

[22]: Johnson, D. B. and Zwaenepoel, W. 1987. « Sender based message logging », In Digest of Papers, FTCS-17, The Seventeenth Annual International Symposium on Fault-Tolerant Computing, 14–19.

[23]:MEROUFEL Bakhta : thèse magister : «Tolérance aux pannes dans les grilles de données » Université Es-Senia juin 2011.

[24] : REBBAH Mohammed : thèse doctorat «Tolérance aux fautes dans les grilles de calcul /Universite des scineces &technologie –mohammed boudiaf-ORAN-juin2014

[25]: Sebastien Monnet :thése doctorat « Gestion des données dans les grilles de calcul : supportpour la tolérance aux fautes et la cohérence des données. » Université de Rennes 1 Novembre 2006

[26] : Himadri S. Paul, Arobinda Gupta R. Badrinath, « Hierarchical Coordinated Checkpointing Protocol », In International Conference on Parallel and Distributed Computing Systems, pages 240-245, November 2002.

[27] :K. Bhatia, K. Marzullo, and L. Alvisi. « Scalable causal Message Logging for Wide-Area Environments ». Concurrency and Computation: Practice and Experience, 15(3), pp.873-889, Aug. 2003.

[28]: Esteban Meneses, Celso L. Mendes, and Laxmikant V. Kale. « Team-basedMessageLogging : Preliminary Results ». In 3rd Workshop on Resiliency in High PerformanceComputing (Resilience) in Clusters, Clouds, and Grids (CCGRID 2010).

[29]:Sébastien Monnet, Christine Morin, and Ramamurthy Badrinath. A hierarchicalcheckpointing protocol for parallel applications in cluster federations. In *9thIEEE Workshop on Fault-Tolerant Parallel Distributed and Network-Centric Systems*,page 211, Santa Fe, New Mexico, April 2004. Held.

[30]:Sébastien Monnet, Christine Morin, and Ramamurthy Badrinath. Hybrid checkpointing for parallel applications in cluster federations. In *4th IEEE/ACM InternationalSymposium on Cluster Computing and the Grid (CCGrid 2004)*, Chicago, IL,USA, April 2004. Poster, electronic version.

[31] W. Bell, D. Cameron, L. Capozza, A. Millar, K. Stockinger et F. Zini. « OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies ». International Journal of High Performance Computing Applications **17(4)** (2003).

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