REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE MOHAMED KHIDER - BISKRA

الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي
جامعة محمد خيضر بسكرة

Faculté des sciences exactes et des sciences de la nature et de la vie
Département d'informatique

THÈSE

Pour l'obtention de grade de

DOCTEUR EN SCIENCES

Filière : Informatique

Titre

Équilibrage de charge dynamique dans les grilles de calcul : une approche à base d'agents

Par

Ali Wided

Devant le jury composé de

Président :	Professeur Boubetra Abdelhak	Université de Bordj Bou Arreridj
Rapporteur :	Professeur Kazar Okba	Université de Biskra
Examinateur:	Professeur Tari Kamel	Université de Bedjaia
	Professeur Laouar Mohamed Rida	Université de Tebessa
	Professeur Khaled Rezeg	Université de Biskra
	Dr.Slatnia Sihem (MCA)	Université de Biskra

Année universitaire 2019/2020

Remerciements

Tout d'abord, je remercie Dieu le tout puissant pour la force et la volonté qu'il m'a donné pour mener à bien ce modeste travail.

Je tiens à remercier le directeur de ma thèse, Mr. Kazar Okba, Professeur au sein de l'Université Mohamed Khiader Biskra qui m'a fait honneur de m'accueillir au sein de son équipe. Il m'a initié à la recherche et à diriger cette thèse en me faisant profit de sa grande expérience et de son talent dans la recherche scientifique. Je lui adresse l'expression de ma profonde gratitude et de ma reconnaissance.

Je tiens également à remercier les membres qui m'ont fait honneur de participer au Jury de cette thèse :

Mr Boubetra Abdelhak Professeur à l'université de Bordj Bou Arreridj pour avoir accepté de présider le jury.

Mr Tari Kamel Professeur à l'université de Bedjaia qui, dans sa charge a accepté d'être examinateur et pour l'intérêt qu'il a porté à ce travail.

Mr Laouar Mohamed Rida Professeur à l'université Cheikh Larbi Tbessi Tébessa pour l'intérêt qu'il a porté à ce travail en acceptant d'en être l'un des examinateurs de cette thèse.

Dr. Khaled Rezeg maître de conférences A à l'université mohamed khider de Biskra pour le grand honneur qu'il me fait en acceptant de juger ce travail.

Dr.Slatnia Sihem maître de conférences A à l'université mohamed khider de Biskra pour m'avoir honoré par sa participation à mon jury de thèse.

J'adresse mes sincères remerciements au Mr Yahia Slimani , Professeur à l'université de la Manouba Tunisie , pour tout son aide pendant la période de mon stage.

Je tiens à remercier aussi, Mr Belabass yagoubi , Professeur à l'université d'Oran ,et Doyen de la Faculté des sciences de la nature et de la vie ,pour son aide et sa modestie.

Finalement, Je désire remercier très sincèrement Alexandra Elbakyan fondatrice de Sci-Hub, le site qui m'a aidé dans mes recherches.

A ceux qui m'ont encouragé et soutenu

Mon père, Ma mère

Mes sœurs , Wafa et Sabrine

Mes fréres,Zohir ,Saber et Ayoub

Mon Mari

mes enfants, Chahd ,Lina et Abdelhalim

A toute ma famille...

Abstract

In grid computing, if the system load in each of resources is nearly equal, it indicates correct use of resources. In this thesis , an agent based load balancing model is presented. In addition of balancing the load in grid by job migration technique , or by moving an entire process to a underloaded resources. The proposed agent based load balancing model aims to take benefit of the agent's characteristics to generate an autonomous system. It also has a distinct advantage over other agent based load balancing models on resource utilisation. The performance evaluation appears that the proposed algorithm can enhance the overall performance of grid computing.

Keywords: Grid Computing; Job Migration; Load Estimation; Performance Metrics; Resource Utilization; Queue Length; CPU Usage ; Multi-agent system

Résumé

En grille de calcul, si la charge du système dans chacune des ressources est presque égale, cela indique une utilisation correcte des ressources. Dans cette thèse, un modèle d'équilibrage de charge basé sur des agents est présenté. En plus d'équilibrer la charge dans la grille par la technique de migration des processus ou en déplaçant un processus vers des ressources sous-chargées. Le modèle d'équilibrage de charge basé sur l'agent proposé vise à tirer parti des caractéristiques de l'agent pour générer un système autonome. Il a également un avantage distinct sur les autres modèles d'équilibrage de charge basés sur les agents sur l'utilisation des ressources. L'évaluation des performances montre que les algorithmes proposés peuvent améliorer les performances globales du grille de calcul.

Mots-clés: Grille de calcul ; Migration de processus; Estimation de charge; Indicateurs de performance; Utilisation des ressources; Longueur de la file d'attente; L'utilisation du processeur ; Système multi-agents

ملخص

في حوسبة الشبكة ، إذا كان تحميل النظام في كل من الموارد متساوي تقريبًا ، فهذا يشير إلى الاستخدام الصحيح للموارد. في هذه الأطروحة، يتم تقديم نموذج موازنة التحميل على أساس الوكيل. بالإضافة إلى موازنة التحميل في الشبكة بواسطة تقنية ترحيل الوظيفة ، أو عن طريق نقل العملية بأكملها إلى موارد ناقصة . يهدف نموذج موازنة التحميل المقترح على أساس الوكيل إلى الاستفادة من خصائص الوكيل لإنشاء نظام مستقل. كما أن لديها ميزة واضحة على نماذج موازنة التحميل المعتمدة على العوامل الأخرى في استخدام الموارد. يظهر تقييم الأداء أن الخوارزمية المقترحة يمكن أن تعزز الأداء الكلي للحوسبة الشبكية.

كلمات البحث: شبكة الحوسبة; هجرة الوظائف; تقدير الحمل; مقاييس الأداء; استخدام الموارد; طول قائمة الانتظار; استخدام المعالج ; نظام متعدد الوكلاء

Table des matières

Table des figures

Liste des tableaux

Introduction générale

A cours de ces dix dernières années, l'état de l'informatique a connu une suite de plates-formes et des améliorations environnementaux [80]. Ces améliorations incluent le développement évolutifs dans l'exploitation plateforme de système, l'architecture de la machine , la connectivité réseau et la charge de travail de l'application. Donc au lieu d'utiliser un ordinateur central à résoudre des problèmes de calcul ,on utilise un système de calcul parallèle et distribué ,ce système utilise plusieurs ordinateurs pour résoudre des problèmes à grande échelle sur internet. Le système distribué devient adéquat aux calculs intensifs et aux traitements des données massives. Ces applications à grande échelle ont évidemment amélioré la qualité de vie dans tous les aspects de notre civilisation[81].

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

Les grilles de calcul sont de ce fait, des systèmes distribués à grande échelle, de composition hétérogène et dynamique. Ce sont là, leurs principales caractéristiques que nous allons aborder tout au long de cette thèse.

L'intérêt de ce nouveau concept de calcul réside dans le fait qu'il ne demande pas d'infrastructures spécifiques ou particulières. En effet, le déploiement d'une infrastructure de type grille peut se faire avec des machines classiques (PC, calculateurs parallèles, stations de travail , etc.) et en utilisant comme infrastructure réseau, le réseau Internet. Si le déploiement de ce type d'infrastructure est relativement simple, leur exploitation est assez complexe. Cette complexité est dûe principalement à leur attribut fortement hétérogène. En effet, les grilles de calcul présentent quatre niveaux d'hétérogénéité : au niveau système d'exploitation , au niveau matériel, , au niveau environnements de développement des applications ,et au niveau réseau[82].

les grilles de calcul est distribuées, c’est-à-dire, répartie sur un grand nombre de machines, le plus souvent hétérogènes et sujettes aux pannes. Ceci constitue un ensemble de contraintes fortes qui doivent être satisfaites si l’on désire que les grille de calcul soient exploitable par des utilisateurs qui n’auront pas à se préoccuper du fonctionnement interne de ces infrastructures. Il est donc nécessaire de définir et de mettre en œuvre un certain

nombre d'outils , de méthodes, et de techniques pour prendre en charge les caractéristiques des grilles . Aussi les travaux, présentés dans cette thèse, tentent d'arborer quelques solutions à la prise en charge de ces caractéristiques au niveau gestion des ressources de calcul d'une grille.

Problématique et Motivations

Une Grille de Calcul est une infrastructure virtuelle constituée d'un ensemble coordonné de ressources informatiques potentiellement partagées, distribuées, hétérogènes et sans administration centralisée [6]. Cependant la gestion de ressource dans ce type d'infrastructure pose évidemment des problèmes beaucoup plus complexes que ceux posés par les systèmes distribués traditionnels, et ce à cause notamment de leur hétérogénéité et de leur dimension dynamique [76]. Parmi ces problèmes, l'équilibrage de charge où il faut en effet éviter, dans la mesure du possible, les situations où certains nœuds sont surchargés alors que d'autres sont sous chargés ou complétement libres. Pour remédier à ce problème plusieurs algorithmes d'équilibrage de charge ont été développés.

Le problème de l'obtention d'une distribution optimale de tâches à des machines dans un système distribué est très complexe et est bien connu pour être NP-complet .

Toute technique d'équilibrage de charge doit en général atteindre les buts suivants [83, 84] : la minimisation du temps de réponse moyen des tâches, la maximisation du débit moyen du système, la répartition équilibrée de la charge du système et la minimisation du temps d'inactivité des ressources.

Bien que le problème d'équilibrage de charge soit un problème qui a été largement étudié, les stratégies d'équilibrage de charge actuelles ne peuvent pas être utilisées telles quelles dans les grilles de calcul. Par exemple, dans le cas des systèmes distribués classiques, les ressources de calcul sont généralement homogènes, leur dissémination géographique est relativement réduite, les utilisateurs de ces systèmes ainsi que leurs applications sont généralement connus à l'avance. Ces différents facteurs font que les presciences de charge des ressources peuvent être prévues à l'avance, ce qui permettra de définir une stratégie d'équilibrage qui soit plus ou moins efficace. Or ceci n'est pas du tout réalisable dans le cas des grilles, puisque plusieurs facteurs font que toute prescience est très difficile à déterminer : hétérogénéité des ressources, dissémination à l'échelle

mondiale, réseaux d'interconnexion hétérogènes, demandes imprédictibles, dynamicité des ressources, profils des utilisateurs différents, applications hétérogènes, etc.[82].

Étant donné toutes ces caractéristiques et bien d'autres, il est donc très difficile, de définir un modèle d'équilibrage universelle pour les grilles. Ainsi, toute proposition de technique d'équilibrage de charge devra, pour des causes d'efficacité, définir le contexte dans lequel elle s'applique. La définition d'un tel contexte nécessite elle même la définition d'un certain nombre d'hypothèses portant sur le type de grilles, le type de ressources de calcul, le type d'applications et les buts à atteindre. C'est dans ce cadre que se situent les travaux présentés dans cette thèse et qui ont pour but de proposer un modèle d'équilibrage de charge dans les grilles de calcul.

Contributions

Notre contribution consiste en la proposition de deux stratégies d'équilibrage de charge dynamiques pour les grilles de calcul :

- La proposition de deux modèles pour résoudre le problème d'équilibrage de charge dans les grilles de calcul : Modèle d'équilibrage de charge avec migration de processus et Modèle d'équilibrage de charge basé sur des agents .
- Les deux modèles proposées ont été testées sur un simulateur de grille.
- Les deux modèles proposées ont été évaluées et comparées à d'autres modèles et les résultats obtenus démontrent leurs hautes performances.

Organisation de la thèse

Les travaux que nous avons menés dans le cadre de la problématique d'équilibrage de charge dans les grilles de calcul, sont résumés dans ce manuscrit composé de deux grandes parties.

La première présente un état de l'art et la deuxième partie est consacrée à la conception et l'implémentation de nos propositions.

Première partie : La première partie se trouve subdivisée en deux chapitres :

- Le premier chapitre, présente un état de l'art sur les systèmes distribués et les grilles de calcul, puis il décrit les composants et les fonctionnalités d'un système d'équilibrage de charge. Par la suite, il expose des principaux travaux sur l'équilibrage de charge dans les grilles de calcul .

- Le deuxième chapitre présente le domaine SMA autant que contexte de recherche pour notre problématique, il rappelle les concepts marquants de ce paradigme de l'agent aux SMA, les interactions entre les agents. Il expose par la suite les travaux d'équilibrage de charge sur grille de calcul à base d'agents.

Deuxième partie : Cette partie se trouve subdivisée en deux chapitres aussi :

- Le troisième chapitre, aborde la conception en matière des modèles et algorithmes proposés. il présente en détail les modèles d'équilibrage que nous proposons pour les grilles de calcul. Il décrit par la suite les algorithmes développée sur ce dernier. ces algorithmes pour le but de minimiser le temps de réponse moyen des tâches et maximiser l'utilisation des ressources

- Le quatrième chapitre, présente une série d'expérimentations et d'évaluations de nos modèles proposés, nous essayons de montrer que les algorithmes proposés permet d'atteindre les buts que nous avons fixés..

Finalement, nous terminons ce manuscrit par une conclusion qui résume à la fois la problématique que nous avons traitée dans cette thèse, ainsi que les résultats que nous avons obtenus. Par la suite nous présentons les, perspectives de recherche ,et les travaux futures pour poursuivre la réflexion sur le problème d'équilibrage de charge dans les grilles de calcul.

Chapitre 1

Équilibrage de charge dans les grilles de calcul

1.1 Introduction

La Grille est un type de système parallèle et distribué qui permet le partage, la sélection et l'agrégation de ressources autonomes géographiquement réparties de manière dynamique. Chacune de ces ressources a sa propre capacité, sa disponibilité, son coût, ses performances et ses utilisateurs, avec ses propres contraintes de qualité de service [1].

Ce chapitre présente le problème de l'équilibrage de charge dans les grilles de calcul. Tout d'abord, nous donnons un aperçu des systèmes distribués et des grilles de calcul à travers leurs définitions, puis nous montrons pourquoi l'équilibrage de charge dans les systèmes distribués est différent de l'équilibrage de charge dans les grilles de calcul et pourquoi le problème de l'équilibrage de charge dans les grilles de calcul est un nouveau défi pour les développeurs et les chercheurs.

Ensuite, nous décrivons les composants et les fonctionnalités qui devraient être inclus dans tout système d'équilibrage de charge. Enfin, une présentation des principaux travaux sur l'équilibrage de charge dans les grilles de calcul est présentée.

1.2. Les systèmes distribués

1.2.1. Définition des systèmes distribués

Un système distribué est un ensemble d'ordinateurs indépendants, qui apparaît à ses utilisateurs comme un système unique et cohérent [2]. Plus précisément, un système distribué peut être décrit comme un ensemble d'unités de calcul (appelé nœud) avec les caractéristiques suivantes [3] :

- Chaque nœud possède son propre espace d'adressage ;
- Le nombre de nœuds dans le système distribué est arbitraire ;
- La communication entre les différents nœuds se fait au moyen de messages. Il convient, dès lors, de tenir compte du délai de communication entre les différents nœuds ;
- Le système est capable de traiter un nombre quelconque de processus.

Les interactions entre les différents processus se font de manière coopérative, et non sur base du modèle maître/esclave ; En cas de panne d'une (ou plusieurs) unité(s) de calcul, le système doit pouvoir se reconfigurer (tolérance au panne).

1.2.2. Caractéristiques d'un système distribué

Un système distribué doit assurer plusieurs caractéristiques pour être considéré comme performant. Nous ne citerons dans cette section que celles que nous trouvons les plus connexes à notre contexte d'étude : transparence, robustesse et puissance

- **Transparence**

Lorsqu'un service est délivré grâce à un système distribué, la complexité de son fonctionnement doit être complétement cachée aux utilisateurs. L'objectif est de pouvoir faire profiter aux applications une multitude de services sans avoir besoin de connaître exactement les détails techniques des ressources qui les fournissent ou la localisation.

Ceci rend plus simple la maintenance évolutive ou corrective des applications et leur développement. Selon la norme (ISO, 1995) la transparence a plusieurs niveaux :

1. Accès : un système distribué doit cacher à l'utilisateur l'organisation logique des ressources et les moyens d'accès à une ressource ;
1. Localisation : la localisation physique d'une ressource du système ne doit pas être connue ;
2. Migration : une ressource peut changer d'emplacement sans que cela ne soit aperçu ;
3. Re-localisation : il s'agit de pouvoir changer la localisation d'une ressource alors qu'elle est en utilisation ;
4. Réplication : c'est la présence de plusieurs copies d'une ressource mais les utilisateurs n'ont aucune connaissance de cela ;
5. Panne : si un nœud est en panne, l'utilisateur ne doit pas s'en rendre compte et encore moins de sa reprise après panne ;
6. Concurrence : un système distribué doit cacher aux utilisateurs qu'une ressource peut être partagée ou sollicitée simultanément par plusieurs utilisateurs ;
7. Défaillance : l'utilisateur ne doit pas savoir qu'une défaillance a eu lieu ;
8. Une défaillance dans le système : l'utilisateur ne doit pas non plus se rendre compte que le système est en train d'exécuter une procédure de tolérance de panne.

- **Robustesse**

Dans un système distribué, plusieurs machines travaillent ensemble pour fournir un service, ce qui conféré au service une meilleure robustesse, comparé à un système centralisé. Nous pouvons distinguer plusieurs aspects de la caractéristique de robustesse :

1. Passage à l'échelle : le concept de passage à l'échelle désigne la capacité d'un système à continuer à délivrer avec un temps de réponse constant un service même si le nombre de clients ou de données augmente de manière importante .
2. Disponibilité : la disponibilité est la probabilité qu'un service soit disponible à un moment donné.

- **Puissance**

La possibilité d'augmenter la vitesse d'exécution des taches. L'idée est d'associer les puissances de plusieurs machines. Combiner les puissances moyennes de plusieurs machines revient bien moins cher et l'ajout de nouveaux membres (de nœuds) augmente la puissance globale du système.

- **Autonomie**

Un système ou un composant est dit autonome si son intégration ou son fonctionnement dans un système existant ne nécessite aucun changement des composants du système hôte.

Les systèmes distribués offrent une capacité de traitement considérable. Toutefois, afin de réaliser cette capacité et d'en profiter pleinement, un bon schéma d'équilibrage de charge est nécessaire. Les termes suivants sont fréquemment utilisés dans le domaine des systèmes distribués :

- **Tâche** : est une unité d'exécution ou une unité de travail, ordonnancée par un ordonnanceur et assigné à une ressource. Elle a des besoins en termes de ressources système (processeurs, mémoire, etc.) ;
- **Job** : (une méta-tâche, une application distribué, un processus, un consommateur de ressource ou un graphe de tâches) est un ensemble de tâches atomiques qui doivent être exécutées sur un ensemble de ressources. Un job peut avoir une structure récursive, c'est-à-dire qu'un job est composé d'un ensemble de tâches où chaque tâche est un job composé d'un autre ensemble de tâche et ainsi de suite ;

- **Ressource** : est une entité destinée à être utilisée pour la réalisation d'une tâche, par exemple, un processeur de traitement de calculs, un dispositif de stockage de données, ou un lien réseau pour le transport de données ;

- **Nœud** : est une collection de ressources avec un ensemble particulier d'attributs associés.

1.2.3. Systèmes distribuées existants

Cette sous-section présente trois grands types des systèmes distribuées existants : les Clusters, les Clouds et les Grilles de calcul.

1.2.3.1. les clusters

Définition

Le **cluster** est une grappe, un groupe de deux ou plusieurs machines indépendantes fonctionnant comme un seul et même système, et qui accèdent à des données communes.

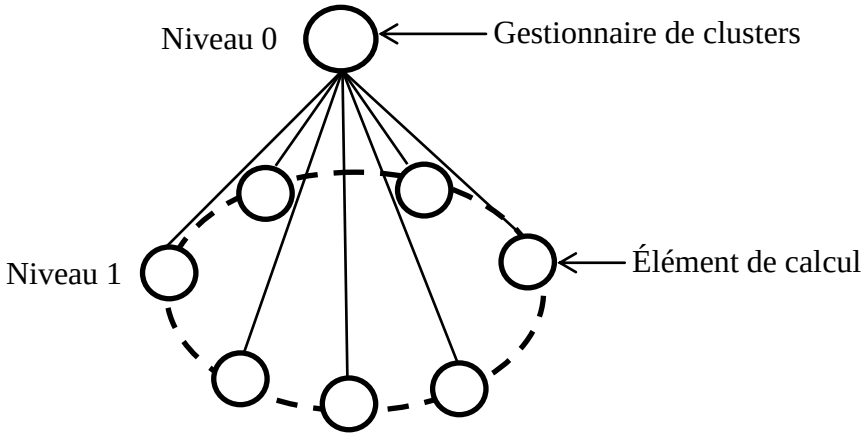


Figure 1.1 : Modèle générique de représentation d'un cluster [4]

La Figure 1.1 présente le modèle générique d'un cluster avec deux niveaux ; le niveau 0 contient un gestionnaire de clusters qui coordonne les éléments de calcul du niveau 1.

Classification des clusters

- Cluster de haute disponibilité (HA) : aussi nommé Failover Cluster, en cas de défaillance d'un serveur (ou d'un ordinateur), toutes les demandes adressées à cet ordinateur sont redirigées vers un autre ordinateur (ou des ordinateurs) du réseau, ce qui permet une indisponibilité nulle ;

- Cluster d'équilibrage de charge : un distributeur est nécessaire pour distribuer les requêtes des utilisateurs à chacun des nœuds ; celui-ci vérifie que chaque nœud possède la même charge de travail. La requête sera envoyée au nœud qui aura le meilleur temps de réponse à celle-ci ;

- Cluster de performance : nommé aussi *High Performance Technical Clustering* (HPC) ; des ordinateurs synergiques qui fonctionnent ensemble pour offrir des vitesses, un stockage, une puissance de traitement et des jeux de données plus importants.

1.2.3.2 Clouds

L'informatique en nuage (*Cloud Computing*) est un paradigme qui propose à l'utilisateur de délocaliser ses ressources de calcul et ses ressources de stockage. Les ressources qui servent de support au *Cloud Computing* sont généralement des clusters car ceux-ci sont plus homogènes et disponibles que les grilles de calcul.

À la maniéré d'un réseau électrique, les entreprises paient la consommation réelle des services proposés par le cloud du fournisseur qui leur garantit une qualité de calcul et de stockage. Avec ce concept, elles n'ont plus besoin de se préoccuper de l'achat de serveurs de calcul et de stockage ou de l'organisation des infrastructures réseaux.

Les applications et les données et ne se trouvent plus sur l'ordinateur local, mais dans un nuage (cloud) composé d'un nombre de serveurs distants reliés au moyen d'une bonne bande passante nécessaire à la commodité du système. L'accès au service se fait par une application standard aisément disponible, la plupart du temps un navigateur Web.

Les services offerts par le Cloud sont nombreux parmi lesquels nous citons :

- **Infrastructure as a service (IaaS) :** les utilisateurs disposent de ressources accessibles et partagées sous forme d'unités de calcul, de supports de stockage, de moyens de communication, etc. Avec ces moyens, les consommateurs pourront dérouler leurs propres logiciels et outils à distance. Ceci veut dire que le client a le contrôle sur le système d'exploitation, l'aspect applicatif et le stockage, mais il n'a pas la gestion des couches basses comme l'infrastructure et le réseau matériel.

- **Platform as a Service (PaaS) :** en plus de fournir une infrastructure virtuelle, il assure également la maintenance de la plateforme permettant d'exécuter les applications de l'utilisateur.

- **Software as a Service (SaaS) :** pouvoir utiliser des applications web s'exécutant sur les serveurs du nuage. Les logiciels sont accessibles par des clients différents, dans le monde entier. Ces derniers n'ont de contrôle que sur les paramètres laissés ouverts explicitement par le fournisseur du service, mais ils n'ont aucun contrôle sur la gestion de l'infrastructure, le système d'exploitation, le réseau ou de l'application elle-même, etc.

Caractéristiques du Cloud

Le *Cloud Computing* a pour but d'offrir des services qui vont au-delà de ces offres classiques et qui peuvent être définis avec les caractéristiques suivantes:

- Il s'agit d'une informatique distribuée ou les échanges sont gérés et centralisés par des serveurs distants, les applications ne sont plus stockées sur le poste de travail, mais sur un « *Cloud* » de serveurs, accédées grâce à une connexion Internet et un navigateur Web.

- Les applications, infrastructures et plateformes nécessaires sont louées en fonction de l'usage souhaité, que ce soit durant le développement de ces applications ou durant leurs utilisations en production.

- Les applications, infrastructures et plateformes sont simplement extensibles.

- Les ressources peuvent être assignées dynamiquement en fonction du besoin.

- Les applications, infrastructures et plateformes continuent disponibles en cas de panne d'une ressource.

1.2.3.3 Grilles de calcul

Le terme anglais ***grid*** s'inspire de la grille d'électricité. Initialement, le concept de grille partait du principe d'un tel système : les ressources d'un ordinateur (processeur, espace disque, mémoire) ont été mis à la disposition d'un utilisateur aussi facilement que le branchement d'un appareil électrique sur une prise électrique.

Une grille de calcul est une infrastructure virtuelle constituée d'un ensemble de ressources de calcul potentiellement partagées, hétérogènes, distribuées, autonomes et délocalisées. Une grille est en effet une infrastructure, c'est-à-dire, des équipements techniques d'ordre matériel et logiciel. Cette infrastructure est qualifiée de virtuelle car les

relations entre les entités qui la composent n'existent pas sur le plan matériel, mais d'un point de vue logique [5].

Définition 1

Ian Foster et Carl Kesselman ont été les premiers a proposé une définition de la grille de calcul:

« …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 » [6].

Le terme **infrastructure matérielle et logicielle** désigne un ensemble de ressources de calcul et de stockage, autonomes et hétérogènes interconnectées par un réseau de communication hétérogène et gérés au moyen d'une couche logicielle (*middleware*) [7].

La **fiabilité** du système de grille est définie comme la probabilité pour tous les programmes de calcul en grille sont exécutés avec succès dans le système informatique en grille. Le besoin d'un accès fiable est fondamental. Les utilisateurs exigent des assurances qu'ils recevront des performances fiables et souvent de haut niveaux à partir des composants constituant la grille.

Le besoin de **cohérence** (*consistency*) du service est un second intérêt fondamental. Une grille doit être construite avec des services standard, des protocoles, des interfaces, et opérants au sein de paramètres standards. Un important défi lors du développement des standards est d'encapsuler l'hétérogénéité sans compromettre la haute performance d'exécution.

L'accès à **taux de pénétration élevé** permettrait la disponibilité des ressources en s'adaptant à un environnement dynamique dans lequel la défaillance des ressources est courante ; cela n'implique pas que les ressources sont partout ou universellement accessibles.

Finalement l'infrastructure doit offrir un accès relativement **bon marché** au regard des bénéfices qu'elle peut apporter. L'aspect bon marché est très important d'un point de vue de la viabilité économique.

Définition 2

La définition 1 a été raffinée par la suite pour s'accommoder à des questions sociales et politiques

« …le concept de grille est **un partage des ressources** et résolution de problèmes de manière coordonnée au sein **d'organisations virtuelles** dynamiques et multi institutionnelles » [9].

Le **partage** ici n'est pas principalement un échange de fichiers, mais plutôt l'accès direct à des ordinateurs, des logiciels, données, ainsi que d'autres ressources**.**

Les **organisations virtuelles** représentent l'ensemble d'individus et/ou d'institutions définis par les règles d'un tel partage.

Définition 3

Les grilles sont passées du cadre académique au cadre industriel, cependant, des confusions apparaissent et nous avons tendance à parler de réseaux chaque fois que nous avons un réseau dans lequel nous partageons des ressources. Pour éviter ce type de confusions, Ian Foster, dans son article « *What is the Grid ?* », présente trois critères de base pour distinguer une grille d'un autre système distribué sous forme de *checklist*:

- **La grille coordonne des ressources qui ne sont pas sous contrôle centralisé :** une grille intègre et coordonne des ressources et utilisateurs qui se trouvent au sein de différents domaines de contrôle, et aborde les questions de sécurité, politique, paiement, appartenance (adhésion), etc., qui se posent dans ce cadre. Sinon, nous avons affaire à un système de gestion local.

- **La grille utilise des protocoles et interfaces standards, ouverts et universels** : la grille est conçue à partir de protocoles et interfaces universels qui adressent des questions fondamentales telle que l'authentification, l'autorisation, la découverte des ressources et l'accès aux ressources. Il est important que ces protocoles et interfaces soient standardisés et ouverts. Sinon, il s'agit d'un système à application spécifique.

- **La grille a pour but de délivrer des qualités de service non triviales :** la grille autorise ses ressources constitutives à être utilisées d'une manière coordonnée afin

de délivrer différentes qualités de service, concernant par exemple le temps de réponse, le débit, la disponibilité, et la sécurité, et/ou la co-allocation de multiples ressources pour satisfaire les demandes complexes de l'utilisateur, afin que l'utilité du système combiné est significativement plus grande que celle de la somme de ses parties.

1.3. Taxonomie des grilles

Il existe différents types de grilles qui correspondent aux différents types de problèmes à résoudre. Quelques grilles sont conçues pour tirer avantage de la puissance de calcul des ressources, alors que d'autres sont conçues pour exploiter la capacité de stockage de ces ressources. Le type de grille est donc, sélectionné suivant le type d'application à traiter. Nous pouvons citer les trois principaux types de grilles :

1.3.1 Grilles de calcul

Une grille de calcul se concentre sur la mise en réserve de ressources spécifiquement pour la puissance de calcul. Dans ce type de réseau, la plupart des machines sont des serveurs hautes performances et sont dédiées aux calculs intensifs.

Ce type de grille peut être à son tour subdivisée en deux catégories :

- **Supercalculateurs distribués :** exécutent une application parallèle sur plusieurs machines dans le but d'améliorer son temps de réponse [10] ;
- **Accélérateurs de calcul :** exécutent une application sur une machine appropriée dans l'objectif d'améliorer le temps de réponse moyen d'un flot d'applications [7].

1.3.2 Grilles de données

Une grille de données est responsable de l'hébergement et de l'accès aux données de plusieurs organisations. Les utilisateurs ne sont pas concernés par l'emplacement de ces données tant qu'ils ont accès aux données. Une grille de données leur permettrait de partager leurs données, de gérer les données et de gérer des problèmes de sécurité tels que l'accès à quelles données.

Ce type de Grille convient aux applications nécessitant une importante capacité de stockage. Elles assurent un accès sécurisé aux données distribuées. Les grilles de données

incluent le concept de bases de données fédérées dans lequel un groupe disponible de ces bases fonctionne comme une seule.

1.3.3 Grilles de services

Les grilles de services sont utilisées pour les systèmes qui offrent des services ne peuvent pas être disponibles sur une simple machine. Elles peuvent être subdivisées en deux catégories [7] :

- **Grille à la demande** : elle agrège dynamiquement différentes ressources pour fournir des nouveaux services. A titre d'exemple, nous pouvons citer le problème de simulation qui nécessite des ressources (en nombre et en type), qui dépendent des paramètres d'exécution. Autre exemple de grille à la demande est la grille « Cloud Computing ». Ces dernières années, les grilles de Cloud Computing ont suscité un intérêt particulier. Ce concept consiste à déporter sur des serveurs distants, des traitements informatiques traditionnellement localisés sur le poste client de l'utilisateur ou les serveurs locaux .

- **Grille de collaboration :** elle regroupe plusieurs utilisateurs et applications en groupes de travail collaboratif. Ce type de grille permet une interaction entre les utilisateurs et les applications en temps réel au moyen d'un espace de travail virtuel.

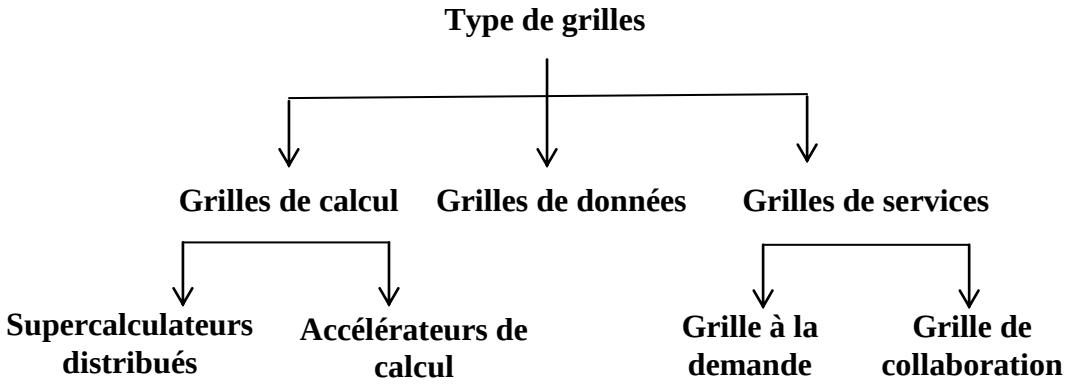


Figure 1.2: Types de Grilles

1.4 Objectifs des Grilles de calcul

La grille de calcul vise à atteindre les objectifs suivants [11] :

- **Le partage de ressources de calcul distribuées et hétérogènes appartenant à différentes organisations :** la grille de calcul est le partage, la sélection et l'agrégation d'un groupe de ressources telles que les superordinateurs, les ordinateurs centraux, les systèmes de stockage, les sources de données et les systèmes de gestion qui se comportent comme des réseaux de calcul. Elle favorise le partage des ressources distribuées pouvant être de nature hétérogènes.

- **L'exploitation des ressources sous-utilisées :** dans la plupart des organisations et des entreprises, les ressources de calcul sous-utilisées sont très nombreuses. La plupart de ces ressources ne sont occupées que moins de 5% du temps. Donc, ces ressources sont relativement inactives. La grille de calcul est conçue pour exploiter ces ressources sous-utilisées et augmenter l'efficacité de l'utilisation des ressources. Les utilisateurs peuvent également louer les ressources qui se trouvent sur la grille pour exécuter leurs applications de calcul intensif, au lieu d'acheter leurs propres ressources (coûteuses) dédiées.

- **L'activation et la simplification de la collaboration entre différentes organisations** : une autre capacité de grille de calcul est la création d'un environnement propice à la collaboration entre organisations. Le Grid Computing permet aux systèmes très hétérogènes et distribués de fonctionner ensemble, ce qui simplifie ainsi la collaboration entre les différentes organisations en fournissant un accès direct aux ordinateurs, logiciel et données.

- **La fourniture d'un service de connexion unique avec un accès sécurisé aux ressources du réseau tout en protégeant la sécurité des utilisateurs et des sites distants :** les grilles fournissent un service de connexion unique à tous les utilisateurs sur toutes les ressources distribuées en utilisant des mécanismes d'authentification de grille. Elles fournissent également un accès sécurisé à toute information n'importe où sur n'importe quel type de réseau. Ceci est réalisé en fournissant mécanismes de contrôle d'accès qui régissent ces ressources.

- **La fourniture de gestion des ressources, des services d'information, surveillance et transport sécurisé des données :** le partage des ressources et des réseaux impliqués dans la grille de calcul sont difficiles à gérer et surveiller, mais la

grille de calcul est en mesure de relever ces défis en raison de son architecture et de ses protocoles.

- **Une solution aux problèmes à grande échelle :** les grilles sont conçues pour exploiter les ressources sous-utilisées, ce qui signifie qu'elles peuvent en employer un grand nombre pour résoudre un problème à grande échelle. C'est une solution prometteuse pour des problèmes tels que le stockage et le traitement de grandes quantités de données que même les ordinateurs centraux ne peuvent pas traiter, comme les prévisions météorologiques. Ces ressources peuvent être des dispositifs de grande capacité tels que le stockage de disque de grande capacité et le calcul de haute performance.

- **La fourniture de résultats rapides et une livraison plus efficace :** la grille de calcul permet le traitement en parallèle. Ce traitement peut également avoir des périphériques à haute capacité. Avec les grilles de calcul, les entreprises peuvent utiliser efficacement les ressources de calcul et de données et les combiner pour des charges de travail de grande capacité.

1.5 Différentes topologies de Grilles

Les auteurs dans [12] ont répertorié les grilles en trois classes d'un point de vue topologique, par ordre croissant de l'étendue géographique et d'un point de vue de la complexité.

Ainsi, nous distinguons les intragrilles, les extragrilles et les intergrilles :

1.5.1 Intragrille

La plus simple des trois topologies est l'*intragrid*, qui consiste simplement d'un ensemble de ressources et de services qui appartiennent à une organisation unique. Les principales caractéristiques d'une telle grille sont l'interconnexion à travers un réseau performant et haut débit, un domaine de sécurité unique et maıtrisé par les administrateurs de l'organisation et un ensemble relativement statique et homogène de ressources.

1.5.2 ExtraGrille

Une extragrille étend le modèle en rassemblant plusieurs intragrilles. Les principales caractéristiques d'une telle grille sont la présence d'un réseau d'interconnexion hétérogène

haut et bas débit (LAN/WAN), de plusieurs domaines de sécurité distincts, et d'un ensemble plus ou moins dynamique de ressources. Un exemple d'utilisation est lors d'alliances et d'échanges « *Business-to-Business* » (B2B) entre les entreprises partenaires.

1.5.3 InterGrille

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

1.6 Équilibrage de charge dans les grilles de calcul

En nous concentrant sur l'étude des approches d'équilibrage de charge dans les grilles de calcul, nous avons trouvé une vaste documentation disponible dans ce domaine.

Dans leur article [13], Casavant et Kuhl ont défini l'ordonnancement (*scheduling*) de façon très simple. Il s'agit d'un mécanisme ou d'une politique d'allocation qui décide où seront allouées les tâches pour accéder aux ressources, utilisée pour gérer efficacement l'accès à une ressource physique (mémoire, périphériques, réseaux, processeurs) et son utilisation par ses variés consommateurs.

Dans [14], le partage de charge (*load sharing*) est le processus de partage de ressources de calcul en répartissant de manière transparente la charge de travail du système. La performance du système peut être améliorée en transférant une partie de la charge de travail du nœud surchargée (*overloaded*) vers le nœud sous-chargé (*underloaded*). La migration de tâches n'est donc envisagé que lorsque la charge locale dépasse un seuil admissible. Par contre dans l'équilibrage de charge (*load balancing*), l'allocation des tâches est envisagée chaque fois que les conditions globales du système changent, c'est à dire à chaque création ou terminaison de processus.

Dans cet travail, nous nous intéressons à l'équilibrage de charges où le but est de répartir la charge entre les nœuds en évitant qu'un nœud ne soit inactif alors que des tâches restent en attente sur d'autres nœuds.

Le problème de l'équilibrage étant un problème relativement ancien, beaucoup des approches ont été proposées pour le résoudre dans différentes plates-formes. Casavant et Kuhl ont proposé une classification très complète dans le cadre de systèmes distribués (Figure 1.3). Les quatre principales classifications suggérées par ces auteurs sont :

- **approche statique vs approche dynamique :** dans une approche statique, l'allocation des tâches s'effectue avant l'exécution de l'application qui les contient. Les informations concernant le temps d'exécution des tâches et les caractéristiques dynamiques des machines sont supposées connues a priori. Cette approche est efficace et simple à mettre en œuvre lorsque la charge de travail est au préalable suffisamment bien caractérisée. En parallélisme, l'approche statique a une longue histoire ; elle est encore très utilisée et étudiée. Des résultats très satisfaisants ont été obtenus d'un point de vue théorique et pratique [16] [17]. Dans une approche dynamique, l'assignation des tâches aux machines se décide pendant la phase d'exécution, en fonction des informations qui sont collectées sur l'état de charge du système. Ceci permet d'améliorer les performances d'exécution des tâches mais au prix d'une complexité dans la mise en œuvre de cette stratégie, notamment en ce qui concerne la définition de l'état de charge du système, qui doit se faire de manière continue. Dans le cadre de ce travail nous nous intéresserons seulement aux approches dynamiques, car nous cherchons à répondre aux besoins des applications irrégulières.

- **approche centralisée vs approche distribuée :** dans une approche centralisée, un nœud désigné comme coordinateur reçoit les informations de charge de tous les autres nœuds qu'il assemble pour obtenir l'état de charge global du système. Quand un nœud décide de transférer une tâche, il envoie une demande au coordinateur, qui choisit alors un nœud cible, en utilisant le vecteur de charge, et informe le nœud source de ce choix. Cette approche réduit les frais généraux du système grâce à la centralisation de l'information de charge. Cependant, le coordinateur peut être l'objet d'un goulot d'étranglement. Une panne au niveau du coordinateur provoquera l'effondrement du système. Pour remédier à cette situation, des mécanismes de réplication sont souvent utilisés [18]. Dans le cas d'une approche distribuée, chaque nœud du système est responsable de collecter les informations de charge sur les autres nœuds et de les rassembler pour obtenir l'état global du système. Chaque nœud construit de manière autonome son propre vecteur de

charge en rassemblant l'information de charge des autres nœuds. Les décisions de placement sont faites localement en utilisant les vecteurs de charge locaux. Les approches distribuées peuvent accélérer de manière significative le processus de prise de décision, mais le coût des communications engendrées peut être élevé.

- **approche source-initiative vs receveur-initiative :** l'approche source-initiative est appliquée lorsqu'un nœud, appelé source, découvre qu'il a une surcharge de travail et qu'il cherche à transférer le surplus vers un nœud sous-chargé. L'approche receveur initiative s'applique lorsque la charge d'un nœud est au-dessous du seuil minimal de charge et il demande à recevoir des processus à partir des nœuds surchargés.

La différence entre les deux types de stratégie, source-initiative et receveur-initiative, réside dans le fait que dans le premier cas, les décisions de distribution de charge sont habituellement prises au moment de l'arrivée d'une tâche, alors que dans le second cas, les décisions sont prises lorsqu'une tâche se termine. La stratégie source-initiative produit de meilleurs temps de réponse quand la charge du système est basse, et la stratégie receveur initiative donne de bons résultats quand la charge du système est haute [19]. Une stratégie hybride consiste à utiliser la stratégie source-initiative quand la charge du système est basse ou moyenne, et la stratégie receveur-initiative quand elle devient excessive. Cette stratégie est souvent appelée stratégie symétrique.

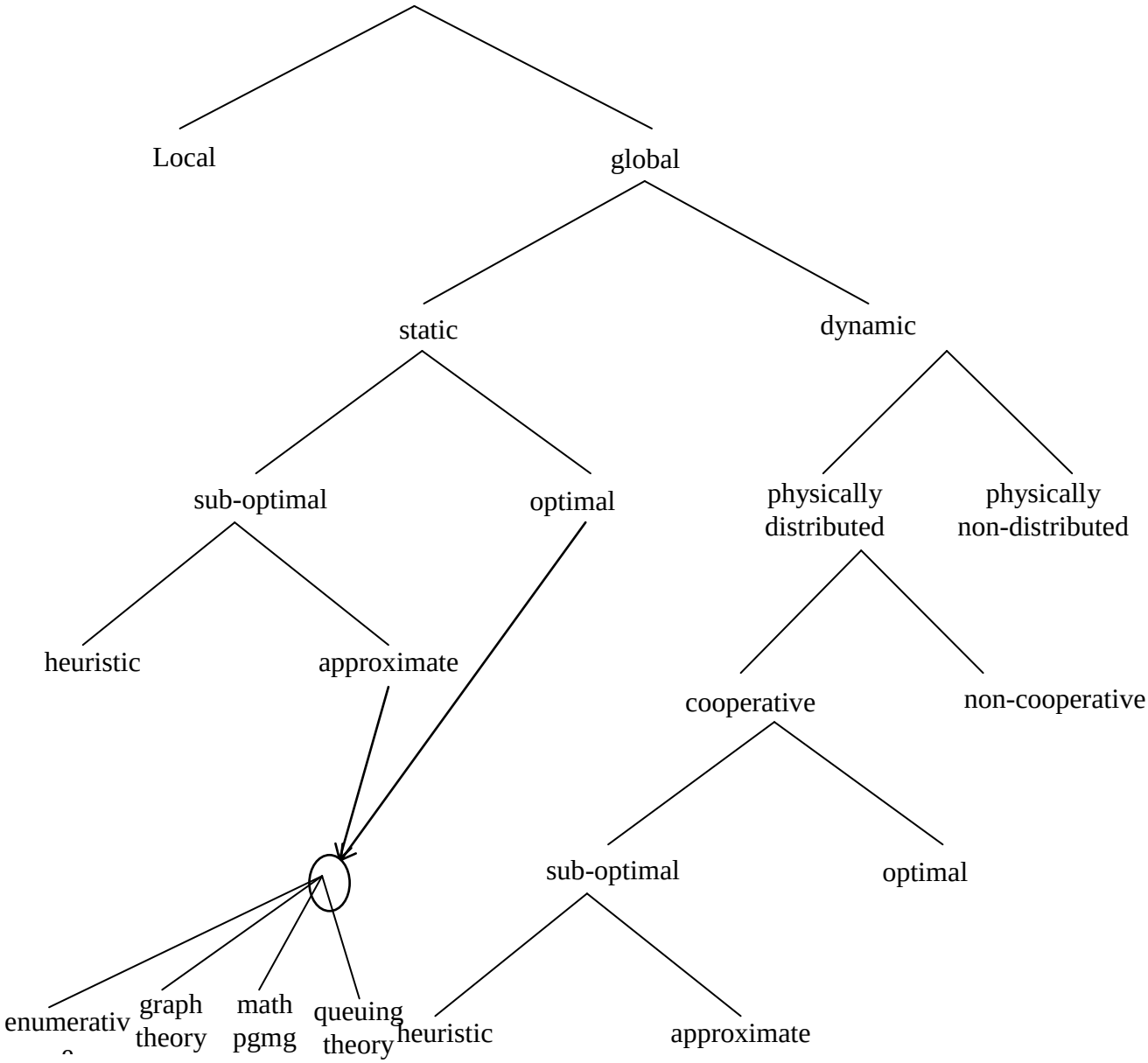


Figure 1.3: Caractéristiques d'ordonnancement des tâches[13]

1.6.1 Problématiques particulières liées aux grilles de calcul

Bien que les clusters et les grilles sont des environnements de calcul parallèle et distribué, ils diffèrent sur plusieurs aspects essentiels que nous allons décrire dans ce qui suit :

1.6.1.1 Caractéristiques d'un système d'équilibrage pour clusters

Les environnements de cluster se distinguent par cinq principales caractéristiques [7] :

- **Homogénéité des ressources et des applications :** les ressources sont homogènes en terme de capacité, ce qui simplifie l'estimation du temps d'exécution des tâches. Un cluster est caractérisé par une relative stabilité sur le nombre de ressources. En plus, un cluster est généralement dédié à un ensemble d'applications prédéfinies, ce qui conduit aux mêmes requêtes en matière de ressources [7].

- **Appropriation des ressources :** en général, toutes les ressources de traitement et de communication appartiennent au même domaine d'administration. En conséquence, leurs comportements sont facilement prévisibles [7].

- **Ordonnancement centralisé :** dans un cluster, l'ordonnancement de tâches est réalisé par une entité centralisée. L'ordonnanceur centralise les informations de charge de chaque ressource du cluster et peut disposer d'un état global sur la charge instantanée du système [7].

- **Réseau de communication homogène et haut débit :** en raison de leur étendue géographique limitée, les réseaux d'interconnexion utilisés par les clusters sont homogènes et offrent une largeur de bande assez suffisante et stable [7].

- **Objectif de performance unique :** l'exécution d'un système d'équilibrage de charge vise un seul et unique but de performance, ce qui simplifie sa conception.

1.6.1.2 Caractéristiques d'un système d'équilibrage pour les grilles

Les grilles de calcul possèdent des caractéristiques qui déterminent leurs principales différences par rapport aux systèmes parallèles et distribués classiques [20] :

- **Existence de plusieurs domaines administratifs :** les ressources de la grille sont géographiquement distribuées et appartiennent à différentes organisations chacune ayant ses propres politiques de gestion et de sécurité. Ainsi, il est indispensable de respecter les politiques de chacune de ces organisations.
- **Hétérogénéité des ressources :** les ressources dans une grille sont de nature hétérogène en termes de matériels et de logiciels.
- **Passage à l'échelle (scalability) :** une grille pourra consister de quelques dizaines de ressources à des millions voire des dizaines de millions. Cela pose le problème de la dégradation potentielle des performances lorsque la taille des réseaux

augmente. Par conséquent, les applications nécessitant un grand nombre de ressources localisées doivent être conçues pour être tolérantes en temps de latence et en bande passante.

- **Nature dynamique des ressources :** dans les grilles, le caractère dynamique est la règle et non pas l'exception. En fait, avec autant de ressources dans une grille, la probabilité de défaillance de certaines ressources est élevée, cela pose des contraintes sur les applications telles que l'adaptation au changement dynamique du nombre de ressources, la tolérance aux pannes et aux délais d'attente.
- **Autonomie :** dans une grille, les nœuds constituent des entités de calcul autonomes. Ils sont géographiquement distribués et appartiennent à différentes organisations, chacune ayant ses propres politiques de gestion et de sécurité. D'une part, il est indispensable de respecter les politiques de chacune de ces organisations pour que les utilisateurs non autorisés ne puissent pas accéder aux ressources appartenant à certains domaines spécifiques [7].
- **Non-appropriation des ressources :** le partage des ressources, par plusieurs utilisateurs à profils différents, conduit à l'utilisation concurrente des ressources. Cette concurrence concerne tous les types de ressources : ressources de calcul, réseaux d'interconnexion, logiciels, etc. Une des conséquences de cette compétition est que le comportement, et donc les performances, varie continuellement avec le temps (comportement dynamique). Sous un tel environnement, concevoir un modèle d'équilibrage précis est extrêmement difficile [7].
- **Diversité des applications :** les applications soumises au système peuvent être très diverses, étant donné qu'elles émanent de déférentes organisations, chacune ayant ses propres exigences [21]. Par exemple, certaines applications nécessitent une exécution séquentielle, d'autres sont composées de modules indépendants qui peuvent s'exécuter en parallèle. Dans ce contexte, il est très complexe de concevoir un système d'équilibrage prenant en compte tous ces paramètres pour supporter une large variété d'applications.
- **Séparation des données et des calculs :** dans les systèmes parallèles traditionnels, les codes exécutables des applications et les données résident en général dans le même nœud. Au cas échéant, les sources d'entrée et les destinations de sortie sont déterminées avant la soumission de l'application. Ainsi, le coût des entrées/sorties

peut être négligé dans le premier cas ou déni avant l'exécution dans le second cas. En conséquence, dans les deux cas, le système d'équilibrage de charge ne considère pas ces coûts. Cependant dans une grille, les données peuvent être distantes des codes, et compte tenu de l'étendue et des performances des réseaux, elles peuvent avoir un coût considérable. Ce coût devient plus significatif quand il s'agit d'applications nécessitants un volume important de données. Ainsi, nous pouvons nous retrouver dans des situations où les gains apportés par un transfert de tâches sont neutralisés par les coûts d'accès aux données nécessaires à l'exécution de ces tâches [7].

Ces caractéristiques posent des problèmes cruciaux de conception d'un système d'équilibrage efficace et effectif pour les environnements de grilles. Certains problèmes sont toujours ouverts et constituent des domaines de recherche d'actualité [22] [23] [24].

1.6.2 Le système d'équilibrage de charge

Un système d'équilibrage de charge est composé de deux éléments essentiels : les politiques et les mécanismes [25]. Les politiques considèrent l'ensemble des choix à réaliser pour distribuer une charge de travail alors que les mécanismes réalisent physiquement la distribution de la charge et fournissent les informations requises par les politiques. La Figure 1.4 illustre la décomposition arborescente d'un système d'équilibrage de charge [26].

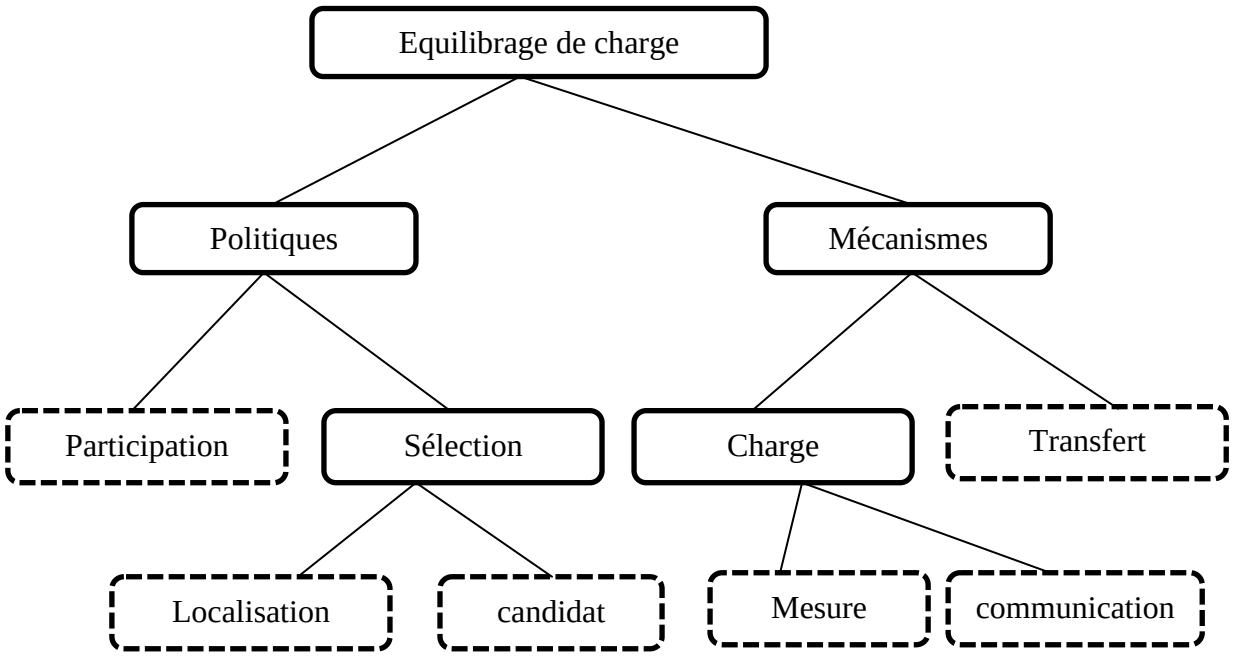


Figure 1.4 : Composants d'un système d'équilibrage de charge[26]

Politique de participation : le but de cette politique consiste à déterminer si un nœud est dans un état approprié pour participer à un transfert de tâches comme source (nœud surchargé) ou comme receveur (nœud sous-chargé) [26].

Politique de sélection de la localisation : cette politique est responsable de trouver, pour un nœud donné, un partenaire (source ou receveur), une fois que la politique de participation a décidé que ce nœud était soit source, soit receveur [26].

Politique de sélection des tâches à transférer : une fois que les politiques de participation et de localisation ont décidé qu'un nœud N_i est source et qu'un autre nœud N_j est receveur, cette politique est responsable du choix des tâches à transférer de N_i vers N_j [26].

Mécanisme de mesure de la charge : dans toute approche d'équilibrage de charge, une des difficultés majeures est celle qui consiste à évaluer la mesure de la charge d'un nœud. Dans la plupart des travaux existants, c'est la longueur de la file d'attente qui détermine la charge d'un nœud. Certains auteurs préconisent comme indicateur de charge, une combinaison entre la longueur de la file d'attente CPU, celle des entrées/sorties et l'occupation mémoire. Dans le cas des grilles de calcul, il est nécessaire de prendre en considération aussi l'hétérogénéité des ressources et des réseaux de communication pour mesurer la charge d'un nœud [26].

Mécanisme de définition de la charge : ce mécanisme essaie de définir la charge globale d'un système en collectant les informations de charge (partielles) sur l'ensemble ou une partie des nœuds du système. Il faudra alors définir les méthodes selon lesquelles l'information de charge est collectée puis diffusée aux nœuds [26].

1.6.2.1 Evaluation de la charge d'une ressource

L'état de charge des différentes ressources est la principale source d'informations pour les techniques d'équilibrage. Le but premier de la mesure de la charge est d'estimer la quantité de traitement attribuée à une CPU. Pour cela un index de charge est associé à la ressource et évolue avec elle. Ferrerai [25] a proposé des critères pour l'indice de charge que nous reprenons ici:

- La capacité d'estimer la charge courante ;

- L’approximation de ce que sera la charge dans un futur proche, car le temps de réponse d’un processus affecté à une machine dépend de la charge future, et non de la charge actuelle ;
- La stabilité, afin de ne pas prendre en compte les fluctuations à court terme ;
- La capacité à prendre en compte la spécificité d’un nœud, l’utilisation d’une ressource particulière, comme la mémoire.

L'indice de charge choisi doit pouvoir être facilement représentable, normalement par un nombre ou un niveau logique de charge ; un nombre s’il représente un état mesuré ; ou un niveau (sous-chargé, surchargé) s’il est comparé avec un seuil. Les valeurs des seuils sont données statiquement ou peuvent changer au cours de l’exécution. Plusieurs indices sont utilisés pour évaluer la charge d’une ressource. Nous citons dans ce qui suit quelques exemples :

- La longueur de la file d'attente du CPU: le nombre de tâches allouées au processeur et pas encore exécutées (plus la file d’attente est longue, plus le temps de réponse sera élevé) ;
- Longueur de la file d’entrée-sortie ;
- Moyenne de la longueur de la file d’attente CPU sur un intervalle de temps ;
- Nombre de messages à traiter ;
- Age des tâches en cours d’exécution ;
- Taux d’occupation de la mémoire de la ressource ;
- Le temps de réponse après une sollicitation multicast ;
- Des modèles statistiques ou stochastiques afin d’établir des prédictions de charge.

Certains auteurs préconisent comme indicateur de charge, une combinaison entre la longueur de la file d’attente CPU, celle des entrées/sorties et l’occupation mémoire.

1.6.2.2 Architecture d'un système d'équilibrage

Cette section décrit une stratégie générale décrivant les différentes étapes à suivre pour réaliser un équilibrage effectif.

Schopf [27] a proposé un schéma général portant sur les différentes étapes à suivre pour concevoir et développer une stratégie d'équilibrage de charge dans les systèmes hétérogènes à large échelle.

Le schéma proposé est composé de trois phases principales : la découverte de ressources, qui génère une liste de ressources potentielles; la collecte d'informations sur ces ressources et la sélection des ressources candidates à participer à un équilibrage; et exécution des tâches, ce qui inclut le stockage et le nettoyage de fichiers.

Phase 1 : Découverte de ressources

La première étape de toute interaction d'ordonnancement consiste à déterminer quelles ressources sont disponibles pour un utilisateur donné.

Étape 1 : Filtrage des autorisations

Il s'agit de disposer d'une liste des ressources auxquelles l'utilisateur est autorisé à y accéder.

Étape 2 : Définition les exigences d'application

L'utilisateur doit pouvoir spécifier les ressources qu'il demande (système d'exploitation, mémoire, vitesse, etc.).

Étape 3: Filtrage sur la base des exigences minimales

La troisième étape de la phase de découverte des ressources consiste à filtrer les ressources qui ne répondent pas aux exigences minimales de la tâche.

Phase 2 : Sélection des ressources

Étape 4 : Collecte d'informations

L'étape de collecte dynamique d'informations comporte deux composants : quelles informations sont disponibles et comment l'utilisateur peut y accéder. Le but est de collecter les informations de charge sur chaque ressource potentielle, afin de réaliser le choix des ressources à utiliser : charge du système, longueur de file d'attente pour chaque CPU, etc.

Étape 5 : Sélection des ressources

Sur la base des informations détaillées recueillies à l'Étape 4, l'étape suivante consiste à choisir la ressource (ou le groupe de ressources) à utiliser.

Phase 3 : Exécution de Tâche

Étape 6: Réservation avancée

Afin de tirer le meilleur parti d'un système donné, il peut être nécessaire de réserver une partie ou la totalité des ressources. Selon la ressource, une réservation préalable peut être facile ou difficile à faire et peut être faite avec des moyens mécaniques ou des moyens humains. De plus, les réservations peuvent ou non expirer avec ou sans coût.

Étape 7 : Soumission de la tâche

Une fois les ressources choisies, l'application peut être soumise aux ressources. La soumission d'une tâche peut être aussi simple que d'exécuter une seule commande ou aussi compliquée que d'exécuter une série de scripts.

Étape 8 : Préparation

L'étape de préparation peut impliquer la configuration, la préparation, la réclamation d'une réservation ou d'autres actions nécessaires pour préparer la ressource à exécuter l'application (transfert de fichier, installation de logiciel, etc.).

Étape 9 : Suivi du progrès

En fonction de l'application et de son temps d'exécution, les utilisateurs peuvent surveiller le progrès de leur application et éventuellement changer d'avis quant à l'emplacement ou la façon dont elle s'exécute.

Étape 10 : Achèvement de la tâche

Lorsque le travail est terminé, l'utilisateur doit être averti. Souvent, les scripts de soumission pour les machines parallèles incluent un paramètre de notification par courrier électronique.

Étape 11 : Tâches de nettoyage

Une fois le travail exécuté, l'utilisateur peut avoir besoin de récupérer des fichiers de cette ressource pour analyser les données des résultats, supprimer les paramètres temporaires, etc.

1.7 les principaux travaux d'équilibrage de charge dans les grilles de calcul

À ce jour, un certain nombre d'efforts ont été déployés pour développer des systèmes d'équilibrage de charge dans les grilles de calcul. Il est difficile de faire une comparaison entre des efforts distincts car chaque approche d'équilibrage de charge est généralement développée pour un environnement système particulier ou une application particulièrement gourmande avec différentes hypothèses et contraintes.

Ainsi, nous avons classé les techniques d'équilibrage de charge selon différentes approches[28] : approche basée sur l'arborescence, approche basée sur l'estimation, techniques de vie artificielle, approche hybride, approche basée sur le partitionnement, approche basée sur les agents et approche basée sur le voisinage.

- **L'approche basée sur l'arborescence :** la méthode d'équilibrage de charge hiérarchique propose le modèle d'arborescence dynamique de grid pour gérer la charge de travail, ce qui diminue la quantité de messages d'échange dans l'environnement grid et entraîne une diminution des couts généraux de communication.
- **L'approche basée sur l'estimation :** l'équilibrage de charge est effectué en estimant le temps d'arrivée prévu d'une tâche sur les processeurs à chaque arrivée de tâche. Les algorithmes d'équilibrage de charge estiment les différents paramètres du système tels que : taux d'arrivée d'une tâche, taux de traitement du processeur et la charge sur le processeur ; puis la charge est équilibrée en migrant les tâches vers les processeurs en tenant compte du coût de transfert des tâches, de l'hétérogénéité des ressources et de l'hétérogénéité du réseau.
- **La méthode hybride d'équilibrage des charges** combine les principes de l'équilibrage de charge statique et dynamique pour résoudre le problème de l'allocation des ressources. Ils utilisent la métrique de mise à jour de l'intervalle

pour réduire le retardement et l'impasse. Il réduit ainsi le temps d'attente des travaux et attribue la priorité.

- **Le partitionnement d'une grille** adaptative pour la distribution sur des processeurs parallèles est considéré dans le contexte de méthodes adaptatives à plusieurs niveaux pour résoudre des équations différentielles partielles. L'exécution parallèle efficace de calculs scientifiques orientés grille nécessite la partition de la grille qui minimise à la fois le déséquilibre de charge et communication entre processeurs.
- **L'approche basée sur les agents :** dans cette approche, une combinaison d'agents intelligents et d'approches multi-agents est appliquée à la fois à l'ordonnancement des ressources locaux du grille et à l'équilibrage de charge global du grille. Chaque agent est un représentant d'une ressource locale de grille et il utilise des données de performance d'application prédictives avec des algorithmes heuristiques itératifs pour concevoir un équilibrage de charge local sur plusieurs hôtes. À un niveau supérieur, les agents coopèrent pour équilibrer la charge de travail à l'aide d'un avertissement de service pair à pair et mécanisme de découverte.
- **L'approche basée sur le voisinage :** une technique d'équilibrage de charge dynamique qui permet aux nœuds de communiquer et de transférer des tâches avec leurs voisins afin que l'ensemble du système soit équilibré après un certain nombre d'itérations. Cette technique ne nécessitant pas de coordinateur global, elle est intrinsèquement locale, tolérant aux pannes et passage à l'échelle.

Le Tableau 1 présente la comparaison entre les approches d'équilibrage de charge précédentes. Dans cette comparaison, les caractéristiques suivantes sont prises en considération :

- **Passage à l'échelle :** il permet la mise à disposition des ressources de la grille pour rejoindre le système de grille sans dégrader le système. Une stratégie d'équilibrage de charge doit prendre en charge un nombre illimité de ressources.
- **Nature dynamique des ressources :** dans un environnement de grille, les ressources de calcul présentent un comportement dynamique dans termes de leur disponibilité. Les ressources de la grille peuvent rejoindre et quitter l'environnement de grille, ou ils peuvent ne pas être disponibles à tout moment en raison d'une défaillance du réseau.

- **Tolérance au panne :** fournit la fonctionnalité qui permet le fonctionnement correct d'un système de grille en présence de défauts. Le besoin de caractéristiques de tolérance des défauts devient également plus évident parce que les ressources peuvent avoir des limites organisationnelles différentes, de sorte qu'il n'y a aucun contrôle sur la disponibilité des ressources.
- **Prise en compte de la capacité de traitement des ressources :** cette fonctionnalité est utilisée pour améliorer la performance des décisions d'équilibrage de charge en examinant la capacité de traitement de chaque ressource du grille.

Tableau 1.1 : Comparaisons des techniques d'équilibrage de charge [28,29,30,31,32,33,34,35,36,37,38,39,40,41,42]

Stratégies d'équilibrage de charge dans les grilles de calcul	**Proposé par**	**Passage à l'échelle**	**Tolérance au panne**	**Amélioration des mesures de performance**	**La capacité de traitement des ressources**	**Gab/Future**
Approche basée sur les arbres	Rathore &Channa[28]	Oui	Oui	Temps de réponse, Efficacité d'allocation des ressources, Temps de communication, Makespan	Non	Réglage de la fonction de seuil d'équilibre
	Nanthiya &Keerthika[29]	Oui	Oui	Makespan, Frais généraux de communication, Taux de succès	Non	Pas donné
	Goswami& Sarkar[30]	Oui	Non	respect de la date limite du tâche soumis	oui	Un nombre variable d'éléments de traitement et une réduction des frais généraux de communication
	Meddeber & Belabbes[31]	Oui	Non	Temps de réponse moyen, Coût de transfert de tâches	oui	Intégrer l'algorithme d'équilibrage de charge dans d'autres simulateurs de grille connus

Approche basée sur l'estimation	Malarvizhi & Uthariaraj[32]	Oui	Non	Temps moyen de réponse, Temps moyen de traitement	oui	La contrainte de priorité entre les différentes tâches et certaines mesures de tolérance aux pannes
	Shah et al.[33]	Oui	Non	Temps d'exécution total, Temps moyen de réponse	oui	Étend en fournissant une tolérance aux pannes
Approche hybride	Yan et al.[34]	Non	Non	Temps de redistribution des tâches, Temps d'achèvement des tâches.	Non	Pas donné
	Li et al[35]	Non	Non	Makespan,Utilisation moyenne des nœuds, L'écart quadratique moyen	Oui	Pas donné
Approche basée sur le partitionnement	Anousha & Ahmadi[36]	Non	Non	Makespan, Taux moyen d'utilisation des ressources	Non	Appliquer d'autre problémes tel que les dates limites des tâches et des ressources
	Penmatsa & Chronopoulos[37]	Oui	Non	Temps de réponse prévu, Temps de communication, Indice d'équité	Oui	Assure l'équité en tenant compte de la charge actuelle du système sur la base de la théorie des jeux dynamiques et d'autres aspects de l'hétérogénéité
Approche basée sur les agents	L.Zhang et al.[38]	Oui	Non	les coûts de communication , la stabilité de la découverte des ressources	Non	temps de requêtes, mises à jour , et enregistrement , mais affecter l'instantanéité de la découverte de l'information

	Herrero & Paletta [39]	Oui	Non	Speedup Surdébit de communication	Non	réduire le temps du processus de négociation
	Cao et al[40]	Oui	Non	Temps d'exécution, Utilisation des ressources, Niveau d'équilibrage de charge,	Non	négociation automatique Qos, coordination auto-organisée, intégration sémantique,le raisonnement basé sur les connaissances, et le courtage de services basé sur l'ontologie.
	Salehi et al.[41]	Oui	Non	Efficacité, Nombre de communications, Vitesse de convergence	Non	l'intelligence de la fourmi, l'ajout de contrats de facturation entre les ressources au fur et à mesure qu'elles échangent les charges, la sécurité
Approche basée sur le voisinage	Balasangameshwara & Raju[42]	Oui	Oui	Temps de réponse, Niveau d'équilibrage de charge, Coût de réplication	Oui	intégration de modèle proposé dans des environnements de grille réels

1.8 Conclusion

Dans ce chapitre nous nous sommes intéressés à quelques notions de bases sur les systèmes distribués à large échelle et l'équilibrage de charge dans les grilles de calcul. Nous avons pu constater que l'équilibrage de charge dans les grilles de calcul représente un défi pour les chercheurs et les développeurs de ce type de systèmes,ce défi est en rapport avec les particularités de ces infrastructures, à savoir, l'hétérogénéité, la dynamicité et le passage à l'échelle.

Nous avons terminé ce chapitre par décrire les principaux travaux de recherche dans le domaine d'équilibrage de charge dans les grilles de calcul, ces travaux consistent en sept directions de recherche : approche basée sur l'arborescence, approche basée sur l'estimation, techniques de vie artificielle, approche hybride, approche basée sur le partitionnement, approche basée sur les agents et approche basée sur le voisinage. Nous avons présenté aussi les travaux existants dans chacune de ces directions . À la fin du chapitre, nous avons présenté une comparaison entre ces travaux. cette comparaiosn a été utile pour trouver la technique appropriée dans un environnement différent pour l'optimisation.

Les architectures multi-agents suscitent un intérêt grandissant en tant qu'outil facilitant la conception, le développement et le déploiement d'applications réparties. L'objectif est de développer un système d'équilibrage basé sur des agents qui soient coopératifs, de telle sorte que chaque agent évolue d'une manière autonome, et coopère avec les autres agents pour se répartir les tâches et les ressources d'une manière efficace. A cet égard, le chapitre suivant sera consacré à cette question.

Chapitre 2

L'équilibrage de charge basé sur des agents dans les grilles de calcul

2.1 Introduction

Les systèmes multi-agents visent à résoudre des problèmes complexes basés sur la répartition des connaissances et des compétences sur un ensemble d'entités autonomes. Dans ce chapitre, nous tenterons de définir le domaine des systèmes multi-agents, nous l'aborderons d'abord sous le concept «agent» en présentant les caractéristiques relatives à cette unité de base du système multi-agents. Ensuite, nous passerons aux interactions entre les agents en présentant ses caractéristiques telles que la coopération, la coordination, la communication et la négociation. Nous terminerons ce chapitre avec une présentation des travaux d'équilibrage de charge dans les grilles de calcul à base d'agents.

2.2 L'intérêt de l'utilisation des systèmes multi-agents

Un système Multi-Agents est défini par Sycara [43] comme l'émergence d'un comportement global produit par un ensemble d'interactions entre agents afin de résoudre des problèmes qui dépassent individuellement leurs capacités de raisonnements.

Les systèmes multi-agents sont particulièrement adaptés pour les systèmes complexes, en effet, ils permettent de reproduire le fonctionnement global d'un système complexe à partir des entités qui le compose et de leurs interactions [44].

Le développement de l'informatique en termes de puissance des machines, le développement des réseaux en particulier, le développement du Web et l'augmentation de la quantité d'informations à traiter et à stocker, ont conduit à des exigences d'application assez complexes. Généralement, ces dernières sont physiquement et fonctionnellement distribuées.

Les systèmes multi-agents répondent à ce genre d'applications, ils permettent la conception modulaire du système. Ces modules peuvent être distribués sur plusieurs machines. Un module est plus simple à concevoir qu'un programme monolithique, de plus, la maintenance du système est plus facile et l'amélioration d'un traitement est localisée en général au niveau d'un agent (s). L'approche par les systèmes multi-agents fournit des solutions robustes et capables de s'adapter dans des environnements qui peuvent être aussi imprévisibles que l'Internet. De surcroit, le problème avec ce système est la coordination et la gestion des interactions entre les agents. C'est la maitrise de ces interactions qui permettra aux agents qui sont autonomes, souvent hétérogènes, mais qui opèrent dans un

environnement public de réaliser des fonctions cohérentes dans un environnement collectif, en formant ainsi un système.

Le défi consiste donc à trouver de nouvelles méthodes adaptatives pour concevoir de nouveaux systèmes informatiques répondant aux difficultés existantes telles que la prise en compte de l'augmentation de la complexité ou la mise en œuvre de systèmes répartis et fiables. Pour découvrir de telles méthodes, il semble approprié de se pencher sur les systèmes collectifs (biologiques, physiques, sociologiques), pour comprendre les mécanismes et processus qui leur permettent de fonctionner.

2.3 Les agents et les systèmes multi-agents

Les systèmes multi-agents possèdent des propriétés que nous allons les introduire dont ils en ont une pluralité de définitions, cependant celle de Jacques Ferber [45] à propos des agents et de Wooldridge [46] à propos des systèmes multi-agents.

2.3.1 Le concept d'agent

Jacques Ferber définit un agent comme

« … **Une entité physique ou virtuelle** qui est **capable d'agir** dans un environnement qui est capable de percevoir son environnement et qui ne dispose que d'**une représentation partielle** de cet environnement, qui montre **un comportement autonome** et qui peut communiquer directement avec d'autres agents qui peut éventuellement se reproduire dont le comportement tend à satisfaire ses buts, en tenant compte des ressources et des compétences dont elle dispose et en fonction de sa perception, de ses représentations et des communications qu'elle reçoit »[45].

- **Une entité physique** est quelque chose qui agit dans le monde réel par exemple : un avion, une voiture ou un robot [45] .
- **Une entité virtuelle** n'existe pas physiquement par exemple : un composant logiciel ou un module informatique [45].
- **Les agents sont capables d'agir** et non pas seulement de raisonner, les agents réalisent des actions qui vont modifier l'environnement des agents et donc leurs prises de décision futures [45].

- **Une représentation partielle** de leur environnement**,** c'est-à dire qu'ils n'ont pas de vue globale de tout ce qui se passe [45].

- **Un comportement autonome** cela signifie que les agents ne sont pas dirigés par des commandes venant de l'utilisateur mais par un ensemble de tendances qui peuvent prendre la forme des objectifs individuels à satisfaire ou des fonctions de satisfaction ou de survie que l'agent cherche à optimiser [45].

2.3.2 Caractéristiques et propriétés d'un agent

Les agents ont de nombreuses propriétés que nous nous rappelons les plus essentielles afin d'avoir une meilleure idée sur un agent, ce dernier, cependant, ne possède pas nécessairement toutes ces caractéristiques :

- **L'autonomie**, autrement dit, l'agent est capable d'agir sans l'intervention d'un tiers (agent ou humain) et contrôler ses propres actions ainsi que son état interne.

- **La perception**, c'est-à-dire, l'agent peut avoir une vue très locale sur son environnement, aussi une représentation plus large de cet environnement et notamment des agents qui l'entourent.

- **La capacité à agir**, en fait, un agent est poussé par un certain nombre de buts qui mènent ses actions, il ne répond pas simplement aux sollicitations de son environnement.

- **La réactivité**, d'une autre façon, l'agent obtient des informations de son environnement et il doit être capable de réagir par suite.

- **La pro-activité**, cela veut dire, un agent ne réagit pas simplement aux changements de l'environnement voire il se dirige vers ses objectifs et il prend des décisions quand il est nécessaire.

- **La communication avec les autres agents** est plus ou moins étendue.

- **L'anticipation** veut dire l'agent détient plus ou moins les capacités de prévenir les événements futurs.

- **La rationalité**, autrement dit, les agents rationnels rangent des critères d'évaluation de leurs actions et ils choisissent des meilleures pour atteindre l'objectif.

- **L'adaptabilité ou la flexibilité**, en effet, un agent adaptable est un agent capable de moduler ses aptitudes selon l'état de l'environnement, autrement dit, il est capable d'apprendre et il sait résoudre un nouveau problème à partir de son expérience.

- **La sociabilité**, c'est-à-dire, un agent interagit avec les autres agents (logiciels et humains) quand la situation l'exige afin d'accomplir ses tâches et aide les autres à rejoindre leurs objectifs.

2.3.3 Typologie des agents

Ferber classe les agents en fonction de leur caractère cognitif ou réactif [45], d'une part, les agents cognitifs dont la plupart sont intentionnels, chacun dispose d'une base de connaissances contenant l'ensemble d'informations et de savoir-faire pour leur permettre de réaliser leur tâche et de gérer les interactions avec les autres agents ainsi avec leur environnement. Bref, ils sont capables d'anticiper et peuvent planifier leur comportement.

Par contre, les agents réactifs sont incapables d'anticiper ou de planifier leur comportement, pourtant il les classe aussi en fonction de leurs comportements téléonomiques dirigés vers des buts explicites ou bien des comportements réflexes régis par les perceptions.

Quant à Nwan, il propose sept catégories d'agents [48] :

- **Les agents réactifs** : Les agents à capacités réactives ne disposent que d'un protocole et d'un langage de communication réduit, ils n'ont pas une représentation précise de leur environnement, ils ne sont pas capables de prise en compte de leurs actions passées et ils ne possèdent pas de procédé de mémorisation ainsi ils ne peuvent répondre qu'à la loi de stimulus/action.

  En effet, ils sont toujours en état de veille sur les changements de leur environnement dès qu'ils aperçoivent une modification de leur environnement, ils répondent par une action programmée, également leurs actions rapides et non réfléchies sont similaires à des réflexes. Ainsi, les agents réactifs ne s'intéressent pas par les individus, mais il se focalise sur la population et les

capacités d'adaptation et d'évolution qui font sortir des interactions entre ses membres [46]. En somme, ces agents réactifs ont des capacités de raisonnement très limitées, mais leurs interactions permettent l'apparition d'une intelligence collective.

- **Les agents collaboratifs** sont des agents autonomes qui coopèrent et négocient avec les autres agents afin d'atteindre des ententes lors de la résolution distribuée de problèmes.

- **Les agents d'interface** : ces agents fournissent une assistance à l'utilisateur, autrement dit, un agent d'interface est capable de s'adapter aux habitudes de l'utilisateur et à ses préférences.

- **Les agents mobiles** sont des agents qui peuvent se déplacer d'un site à un autre en cours d'exécution pour se rapprocher des données ou des ressources [48].

- **Les agents d'information ou d'Internet** : ces agents ont pour rôle de gérer, manipuler ou collecter les informations à partir de plusieurs sources d'informations distribuées.

- **Les agents hybrides** : Un agent hybride repose sur la combinaison de plusieurs caractéristiques des autres agents au sein d'un même agent, à titre d'exemple : la mobilité, la collaboration, l'autonomie, la capacité à apprendre, etc.

- **Les agents intelligents** : Nwana affirme que les agents intelligents (logiciels) n'existent réellement pas encore [48], par contre, d'après Jennings et Wooldridge, un agent intelligent se caractérise par son autonomie, sa réactivité et sa capacité à agir ainsi que sa sociabilité [46].

2.4 Les systèmes multi-agents

Wooldridge définit les systèmes multi agents ainsi :

« … Un ensemble d'agents en interaction afin de réaliser leurs objectifs ou d'exécuter leurs tâches. Les interactions peuvent être directes par l'intermédiaires des communications, comme elles peuvent être indirectes à travers l'action et la perception de l'environnement » [46].

Quant à Ferber, il les définit comme:

« … un système composé d'un environnement E, d'un ensemble d'objets O, d'un ensemble A d'agents, d'un ensemble de relation R qui unissent des objets (et donc des agents) entre eux, d'un ensemble d'opérations OP permettant aux agents de A de percevoir, produire, consommer, transformer et manipuler les objets de O et des opérateurs chargés de représenter l'application de ces opérations et la réaction du monde à cette tentative de modification, que l'on appellera les lois de l'univers » [45].

2.4.1 L'environnement

Dans un système multi-agents, « environnement » veut dire l'espace commun des agents du système, il peut être [47] :

- **Accessible/inaccessible (observable/non observable)**, en effet, il est accessible si un agent, à l'aide des primitives de perception, peut déterminer son état et ainsi effectuer une action, de plus, une fois que l'environnement est inaccessible, il faut que l'agent ait des moyens de mémorisation en vue d'enregistrer les modifications qui ont survenues.

- **Déterministe/non Déterministe,** cela dépend de l'état futur de l'environnement qui est ou n'est pas fixé par son état courant et les actions de l'agent, de plus, dans un environnement déterministe, une action à un effet unique garanti.

- **Discret/continu** : il est discret si le nombre des actions faisables et des états de l'environnement est fini.

2.4.2 Les caractéristiques d'un système multi-agents

Un SMA est généralement caractérisé par

- Premièrement, Chaque agent possède des informations et/ou des capacités de résolution de problèmes limités également chaque agent a une vision partielle.

- Deuxièmement, Il n'y a pas de contrôle global du système multi-agents sauf dans le cas du contrôle centralisé, il peut avoir une entité qui a une vue globale sur l'activité du système et qui prend en charge le contrôle de tout le système.

- Troisièmement, les connaissances et les données sont décentralisées.

- Dernièrement, les agents d'un système multi-agents n'ont pas un point de vue global sur le système, ils n'ont qu'une vision limitée de leur environnement et des autres agents. Chacun ne renferme qu'une partie de la connaissance et qu'une partie des compétences du système, ils doivent posséder des mécanismes tels que : la communication, la coopération, la coordination et une certaine organisation dans le but d'arriver à une résolution collective du problème.

2.4.3 Les interactions

Les interactions proviennent de la mise en relation dynamique de deux ou plusieurs agents par le biais d'un ensemble d'actions mutuelles, en effet, il y a plusieurs types des interactions en fonction de trois paramètres : les objectifs, les ressources et les compétences comme l'indique le tableau 2 suivant :

Tableau 2.1: Types des interactions [45]

But	Ressources	Compétences	Type de situation	Remarques
Compatible	Suffisantes	Suffisantes	Indépendante	Situation d'indifférence
Compatible	Suffisantes	Insuffisantes	Collaboration simple	Situation de coopération
Compatible	Insuffisantes	Suffisantes	Encombrement	Situation de coopération
Compatible	Insuffisantes	Insuffisantes	Collaboration coordonnée	Situation de coopération
Incompatible	Suffisantes	Suffisantes	Compétition individuelle	Situation d'antagonisme
Incompatible	Suffisantes	Insuffisantes	Compétition collective	Situation d'antagonisme
Incompatible	Insuffisantes	Suffisantes	Conflits individuels	Situation d'antagonisme
Incompatible	Insuffisantes	Insuffisantes	Conflits collectifs	Situation d'antagonisme

Il existe différentes situations d'interactions déterminées entre les agents, à savoir, la communication, la collaboration, la coopération, la négociation et la coordination.

1. La communication

La communication est l'un des éléments importants du système multi-agents, c'est un élément essentiel de toute interaction, elle permet l'échange des informations entre deux

agents, c'est grâce à la communication que les agents peuvent échanger des informations et coordonner leurs activités. Egalement, pour des raisons de communication, différents protocoles d'interaction sont implémentés, les agents peuvent interagir en envoyant des messages et même par l'établissement des conversations structurées ou en communiquant directement entre eux en exerçant des opérations sur leur environnement, en somme, la communication entre agents peut être directe ou indirecte.

2. La collaboration

Selon Ferber, la collaboration est considérée comme le processus de création et maintenir une conception partagée d'un problème ainsi qu'un espace commun pour stocker et partager les informations, elle s'appuie sur un engagement mutuel des participants par un effort coordonné pour résoudre conjointement le problème. [45]

3. La coopération

D'après Ferber, la coopération est accomplie par la division du travail entre les participants en tant qu'activité où chaque personne est responsable d'une partie de la résolution du problème, autrement dit, plusieurs agents se coopèrent ou encore ils sont en situation de coopération, si l'une de ces deux conditions est vérifiée: D'une part, l'ajout d'un nouvel agent permet d'accroitre différemment les performances du groupe, d'autre part, l'action des agents sert à éviter ou à résoudre des conflits potentiels ou actuels.

La coopération et **la collaboration** ne se diffèrent pas selon la tâche si elle est répartie ou non mais en raison de la manière dont elle est divisée: dans la coopération, la tâche est divisée (d'une manière hiérarchique) en sous-tâches indépendantes, néanmoins, en collaboration, les processus cognitifs peuvent être (d'une manière hétérarchique) divisés en couches imbriquées. Ainsi, dans la coopération, la coordination n'est nécessaire que lors de l'assemblage de résultats partiels tandis que la collaboration est une activité coordonnée et synchrone qui en résulte.

4. La coordination

La coordination est définie comme le fait de gérer les liaisons des différentes activités exécutées pendant la réalisation d'un objectif, à savoir, les interdépendances regroupent les pré-requis (résultat d'une activité nécessaire à une autre activité), le partage des ressources et la coïncidence (il existe une synchronisation lors de l'exécution des activités) [49].

La coordination est un aspect important pour assurer un travail collectif efficace entre les différents agents, par conséquence, on distingue plusieurs méthodes de coordination [46]:

- **Coordination par planification**: Le principe est l'échange d'informations entre les agents coopérants pour parvenir des conclusions communes sur le processus de résolution du problème, en effet, chaque agent produit un plan local pour résoudre le problème, par suite, ces plans seront échangés entre les différents agents pour le bien comprendre.

- **Coordination par modélisation mutuelle**: Le principe est que chaque agent se met à la place de l'autre et essaye de comprendre ses attentes et ses intensions pour prévoir l'action à entreprendre.

- **Coordination basée sur des lois sociales**: Pour assurer une coordination efficace et cohérente, les agents utilisent des normes comparables à nos normes sociales de coordination, ces dernières peuvent être prédéfinies ou définies tout au long de la vie du système.

- **Coordination à travers les buts communs** : Lorsqu'un groupe d'agent est engagé dans une activité de coopération, les agents ont un objectif commun en plus de leurs objectifs personnels.

La coordination est nécessaire pour améliorer et garder cohérent le fonctionnement global du système, entre autres, dans le cas de la coopération, les agents font collectivement à la résolution d'un problème, ils peuvent utiliser les mêmes ressources et/ou contribuer dans la résolution d'une partie du problème c'est pourquoi ils doivent accomplir les tâches liées au problème à résoudre et coordonner leurs actions.

En fait, les tâches de coordination ne sont pas directement liées à la résolution du problème mais ils permettent au système multi-agents de fonctionner d'une manière efficace ce qui permet au système de résoudre le problème collectivement, de gagner du temps d'exécution, d'éviter les conflits entre agents et de diminuer autant que possible les interactions entre les agents cela pour augmenter les performances du système.

5. La négociation

Elle désigne la stratégie de résolution qui utilise le dialogue pour parvenir à un accord afin de résoudre des conflits des attentes ou d'objectifs, en effet, les conflits des attentes sont produits par l'existence de contradictions entre les attentes des différents agents, ils sont dus au fait que les agents possèdent des connaissances incomplètes [46].

La négociation est basée sur des protocoles qui assignent des rôles aux agents. Chaque agent impliqué dans la négociation exécute le protocole avec le rôle qui lui est assigné. Dans les SMA, il existe trois types de négociation:

- **La négociation par affectation de tâches** : Ce type de négociation fait appel au protocole « Contract-Net » qui est un protocole de négociation entre deux types d'agents : le contractant et le gestionnaire. Le contractant annonce d'abord les sous-tâches aux agents qui doivent faire des recommandations en fonction de leurs capacités à exécuter ces sous-tâches, puis le gestionnaire recueille ensuite toutes les offres qu'il a reçues par suite, attribue la tâche à l'agent ayant fait la meilleure offre.

- **La négociation heuristique** : elle permet aux agents de dépasser l'étape d'acceptation et de rejet en fournissant des commentaires utiles, ces réactions peuvent prendre deux formes:

    - la critique sur l'acceptation ou le refus d'exécuter une tâche.
    - Ou bien, la contre-proposition qui est une proposition alternative donnée par l'agent en répondant à une proposition.

- **La négociation par argumentation** : Une négociation commence toujours par une proposition qui peut être une offre ou une demande, cette étape est suivie par un échange d'illocutions qui peuvent être l'envoi de contre-propositions ou d'arguments de certitude, finalement, une illocution de fin de processus est produite, à titre d'exemple : acceptation ou refus.

2.5 SMA et les grilles de calcul

Bien que les agents et les grilles de calcul sont des termes potentiels entre eux, et ils peuvent être combinés pour produire des techniques innovantes, très peu de recherches ont pris en charge ces technologies, en effet, une novelle méthode appelée **grille de calcul** à

base d'agents est apparait pour offrir des solutions à base d'agents afin d'améliorer la gestion des ressources de grille et ses performances globales.

Quoique les agents ont été utilisés pour l'équilibrage de charge dans les grilles de calcul depuis de nombreuses années en essayant d'appliquer des agents intelligents dans la réalisation de vision de grille a été faite par des chercheurs au cours des dernières années. Une série des travaux sur grille de calcul à base d'agents et clusters de calcul ont été lancés en 2001 [50] dans le cadre du Conférence internationale IEEE / ACM sur les clusters de calcul et les grilles.

2.5.1 La simulation multi-agents de systèmes à large échelle

Les systèmes à large échelle impliquent un très grand nombre d'agents, appartenant à des familles variées et liées par une grande diversité d'interactions, un système à large échelle peut être soumis à une évolution macroscopique déterministe et formulable de façon simple. En fait, les systèmes à large échelle ne peuvent plus passer par des méthodes statistiques ou équationnelles dès lors que les interactions entre les entités qui les composent ne sont plus triviales, en revanche, réciproquement, les approches plus qualitatives qui ont aidé à comprendre les mécanismes à l'œuvre dans les systèmes complexes doivent se doter de nouvelles armes pour faire face à l'effet de masse du passage à large échelle.

Pour ce qui est des systèmes multi-agents, originellement appliqués à la simulation ou à la production de systèmes complexes, le principal problème serait donc de passer au large-échelle mais il ne s'agit pas seulement d'un changement de volume dont viendrait aisément à bout l'accroissement régulier des capacités de calcul des machines, il faut également revoir la façon de représenter les connaissances pour rendre celles-ci plus lisibles, permettre facilement la révision de modèles dont l'expression est de plus en plus compliquée et de façon générale faciliter et simplifier l'accès du thématicien au modèle conceptuel.

2.5.2 Les avantages du paradigme multi-agents

Les avantages du paradigme multi-agents dans le développement d'applications sont [51] :

- **Exécution asynchrone et autonome**: Le déploiement de multi-agents avec des tâches intégrées qui nécessitent des connexions ouvertes continues entraîne une faible économie de latence, effectivement, les MA peuvent être invoqués dans des réseaux où ils fonctionnent indépendamment et de manière synchrone sans surveillance constante.

- **Tolérance aux pannes et robustesse**: La réactivité de l'agent permet la construction d'applications distribuées et résistantes aux pannes et robustes.

- **Consommation de bande passante**: L'utilisation de la bande passante réseau est minimisée car les agents déplacent le code de calcul vers les données, ce qui réduit le passage des résultats intermédiaires.

- **Hétérogénéité** : Les MA offrent des conditions optimales pour une intégration transparente du système car ils sont indépendants des couches de transport et de matériel.

- **Adaptabilité dynamique**: Grâce aux fonctionnalités intégrées, les agents peuvent percevoir les changements dans leur environnement d'exécution et réagir de manière autonome à ces changements.

2.5.3 Défis du système multi-agents

Malgré que les principales caractéristiques du MAS accroissent son applicabilité dans plusieurs disciplines, un défis des recherches importantes doit être abordé notamment: la coordination entre agents, apprentissage, détection des défauts, répartition des tâches, localisation et organisation et sécurité. Autrement, les défis MAS sont généralement spécifiques à l'application, c'est pour cela, dans cette section, nous décrivons les principaux défis qui s'appliquent dans la grande majorité des applications [52].

2.5.3.1 La coordination entre agents

Le contrôle de la coordination fait référence à la gestion des agents pour atteindre en collaboration leurs objectifs, de nombreux défis découlent de la coordination, notamment le consensus, la contrôlabilité, la synchronisation, la connectivité et la formation qui sont décrits ci-dessous:

1. **Consensus:** Au MAS, le consensus se réfère à la conclusion d'un accord mondial sur une caractéristique particulière d'intérêt, parvenir à un consensus comporte deux sous-défis principaux fondés sur les caractéristiques de la MAS sous-jacente, à savoir :

   - **Le suivi** : Dans le MAS suivi par le leader avec un leader, les agents devraient parvenir à un consensus sur la position du chef, cela garantit que les agents maintiennent leur connexion avec le leader, ce défi est appelé suivi. [53]

   - **Confinement:** Il est similaire au suivi à l'exception que MAS a plus d'un leader [54], les dirigeants peuvent limiter la position géographique d'agents pour contrôler les frontières de leur groupe. Par addition, les dirigeants peuvent se connecter les uns aux autres pour échanger les données de contrôle ou celles produites par des agents comme les données captées de l'environnement.

2. **Contrôlabilité :** elle fait référence à la situation où le MAS peut être dirigé d'un état initial à un état spécifique en utilisant certains règlements, la contrôlabilité du MAS est affectée par deux mesures clés qui sont le degré de dynamisme dans la topologie et le degré de déterminisme dans l'environnement. D'une part, dans un MAS dynamique, la topologie change périodiquement, affectant les liens entre les agents et leur collaboration. D'autre part, dans les environnements non déterministes, les résultats des actions ne sont pas prévisibles, les agents devraient donc décider de nouvelles actions après en observant le résultat de leurs actions antérieures qui encourent retarder et limiter la proactivité des agents.

3. **Synchronisation** : elle signifie que les actions exécutées par chaque agent sont alignées dans le temps avec d'autres agents, ainsi une hétérogénéité accrue entre les agents complique la synchronisation, la raison en est que les agents homogènes peuvent être synchronisés dans une caractéristique commune, tandis que pour les agents hétérogènes, la synchronisation doit être réalisée dans plusieurs caractéristiques distinctes, en outre, la synchronisation hétérogène est

également connue sous le nom de synchronisation d'état partiel ou de sortie dans la littérature [55].

4. **Connectivité:** Dans certaines circonstances, les agents exigent une connexion permanente les uns aux autres, par exemple les agents dans leader-follow doivent toujours être connectés au leader, cette exigence introduit le défi de la connectivité. Les défis suivants contribuent à la complexité de la connectivité:

   - **Mobilité:** la mobilité des agents entraîne des déconnexions fréquentes entre les agents et le rétablissement ultérieur de nouvelles connexions.
   - **Environnements bruyants**: Toute interférence dans l'environnement peut interrompre la connexion entre deux agents et donc nécessiter le rétablissement de ces connexions.
   - **Vue limitée de la topologie MAS**: Vu que les agents ont une vue limitée, le positionnement de l'agent pour atteindre une connectivité maximale est difficile.

5. **Formation:** Cela veut dire que MAS à organiser dans une structure particulière et que cette dernière est maintenue pendant une période de temps spécifique (qui pourrait même être toute la durée de vie du MAS). La formation se déroule dans trois étapes principales :

   - Trouver la structure la plus efficace à appliquer par tous les agents.
   - Organiser les agents sur la base de la structure déterminée.
   - Maintenir la structure déterminée pendant un temps spécifique.

De surcroît, hétérogénéité des agents, vue limitée de l'environnement et dynamicité du MAS ou de l'environnement rendent les étapes décrites très difficiles [56].

2.5.3.2 Apprentissage

Dans MAS, chaque agent décide de manière autonome de l'action appropriée pour atteindre son objectif en fonction de plusieurs métriques, les défis suivants augmentent la complexité de l'adoption de systèmes d'apprentissage pour MAS [57], [58], [59]:

- Frais généraux de traitement et de communication des méthodes d'apprentissage qui consomment les ressources des agents.
- L'environnement MAS peut être dynamique, ainsi les agents doivent fréquemment détecter des informations mises à jour à utiliser par la machine d'apprentissage qui à son tour consomme une quantité importante de ressources d'agent.
- La topologie du MAS peut se changer, ce qui nécessite de se reconnecter avec les agents voisins.
- Protection des agents contre les agents malveillants qui injectent des informations fausses.
- Passage à l'échelle de la méthode d'apprentissage pour les MAS à grande échelle.

2.5.3.3 Détection de fautes

Détecter et isoler les agents défectueux est une tâche fondamentale car un agent défectueux peut infecter d'autres agents qu'il se collabore avec [60]. Les études actuelles sur la détection et l'isolement des défauts souffrent des limitations suivantes [61], [62], [63]:

- L'accent est principalement mis sur les agents homogènes alors que dans la plupart des applications, les agents sont hétérogènes.
- La plupart des méthodes existantes exigent un traitement des ressources et / ou des données élevé qui pourrait ne pas être abordable pour tous les agents.
- Seuls quelques travaux discutent de l'isolement des agents défectueux. Cependant, un agent défectueux détecté mais non isolé est capable d'envoyer des informations à d'autres agents, consommant ainsi des ressources d'autres agents.
- La plupart des solutions proposées sont centralisées et donc pas nécessairement utile pour MAS

2.5.3.4 Répartition des tâches

Elle fait référence à l'allocation des tâches aux agents en tenant compte du coût, du temps et des frais généraux (de communication et de traitement) associés, en effet, il est important de présenter ces deux mesures fondamentales pour allouer des tâches aux agents :

- **Talent d'agent**: il s'agit du nombre total de ressources de chaque agent, les agents attribuent des tâches proportionnelles à leurs ressources, cependant, il est important de tenir compte de la charge actuelle d'un agent avant d'allouer une nouvelle tâche, en cas où les tâches allouées à un agent dépassent les ressources de l'agent (c'est-à-dire que l'agent est surchargé), puis le retard dans la réception d'une réponse de l'agent s'augmentera considérablement. Pour éviter une surcharge, chargez l'équilibrage est utilisé pour répartir également la charge entre agents.

- **Position d'agent**: la position d'un agent a un impact sur le retard de communication ainsi que les frais généraux (par exemple, le nombre de paquets doit être transmis pour communiquer avec d'autres agents) [64], [65], ainsi, l'agent devrait être pris en considération les frais généraux, lors de l'attribution des tâches réduire.

2.5.3.5 Localisation

Chaque agent a une vue limitée (uniquement ses voisins) de la topologie MAS avec cette vue limitée, la localisation d'un agent particulier, cela veut dire que la localisation peut être difficile, un agent peut être localisé sur la base de [66]:

- Avoir des ressources particulières.
- Qui est connu comme la localisation des ressources.
- Exécuter des services spécifiques.
- Posséder une identité spécifique.

2.5.3.6 Organisation

L'organisation fait référence à la façon dont les communications des agents et les connexions sont définies, dans certaines approches qui sont décrit ci-dessous, les agents peuvent être regroupés en fonction des caractéristiques telles que les objectifs, les

ressources et les services [67], en fait, les approches les plus répandues pour organiser le MAS sont [68] [69]:

- **Plat:** c'est la structure organisationnelle la plus élémentaire où dans tous les agents sont considérés comme égaux, par conséquent, il n'y a pas chef désigné voire chaque agent communique avec ses voisins.

- **Hiérarchique:** Dans une organisation hiérarchique, les agents ont relations arborescentes, au niveau le plus élevé, il existe un agent appelé agent racine, l'organisation hiérarchique peut entraîner un retard ou créer un goulot d'étranglement, en particulier au niveau de l'agent racine (ou des pères) car il est responsable sur le traitement des communications de tous les agents feuilles.

  Basé sur le nombre d'agents autorisés, autrement dit, ceux qui ont le contrôle sur d'autres agents, l'organisation hiérarchique peut être divisée en deux types : simple et uniforme. Dans l'approche simple, l'agent racine a une autorité exclusive et il contrôle toutes les communications. Dans l'approche uniforme, il y a plus d'une autorité d'agents dans la hiérarchie, ce qui signifie qu'en plus à la racine, tous ou certains pères peuvent également contrôler leur fils.

- **Holonique**: Dans l'organisation holonique, les agents sont organisés en plusieurs groupes appelés holons sur des caractéristiques particulières, par exemple l'hétérogénéité ou la détection capacité des agents à holon. Les holons sont, ensuite, superposés en plusieurs couches. Egalement dans cette organisation holonique, un agent peut être membre de plusieurs holons dans la même couche. Pour les communications de couche supérieure, un agent principal qui est sélectionné parmi les agents les plus disponibles dans le holon, est utilisé.

2.5.3.7 Sécurité

La sécurité est très difficile dans MAS à cause de la décentralisation, sociabilité et mobilité [70], en effet, les effets de ces fonctionnalités sont discutés ci-dessous:

- **Sociabilité**: les agents utilisent les informations ou les connaissances qu'ils acquièrent auprès d'agents voisins ou l'environnement du processus décisionnel, cette sociabilité rend un agent vulnérable contre les entités

malveillantes qui peuvent partager des données falsifiées pour influer sur la décision d'agent.

- **Décentralisation**: En l'absence d'une autorité centrale de confiance, la vérification de l'identité des agents et la création de relations de confiance entre eux deviennent très difficiles.

- **Mobilité**: Un agent mobile peut être affecté par des agents malveillants, si le cas se présente, il diffuse de fausses informations à un nombre croissant d'agents qu'il rencontre en se déplaçant, en revanche, l'autre menace pour la sécurité des agents mobiles est qu'ils pourraient attaquer l'agent sur lequel ils ont migré et consommé leurs ressources ou lire leurs données.

2.6 Travaux connexes

Nous présentons dans cette section un nombre de travaux de recherche portant sur l'équilibrage de charge basé sur des agents dans les grilles de calcul.

Les auteurs [71] ont proposé un modèle d'équilibrage de charge dans les grilles de calcul basé sur des agents (AGLBM) en analysant la charge des nœuds et la migration subséquente des machines virtuelles des nœuds surchargés, vers les nœuds sous-chargés.

Le système proposé implique plusieurs nœuds qui interagissent pour implémenter des travaux MapReduce, le système multi-agents se compose d'un groupe d'agents: agent capteur de nœud, agent capteur de modèle de simulation, agent d'analyse, agent de migration et agent de distribution.

Les agents d'analyse et de distribution sont définis comme des agents de raisonnement ainsi les capteurs et l'agent de migration sont des agents réactifs.

L'agent capteur collecte les informations nécessaires sur l'état du nœud, ces informations contiennent la charge du nœud et la charge de la ligne de communication, ainsi lors de la collecte d'informations, utiliser des compteurs de performances. L'agent capteur observe l'évolution de l'état des objets du modèle simulé situés sur le nœud, en tant qu'information que l'agent capteur envoie comme information de sortie, la fréquence de l'événement, la fréquence de réception et d'envoi des messages à partir des pôles et la

fréquence du changement d'état. L'agent d'analyse interroge les agents d'analyse à certains intervalles, il décide, à l'aide des règles, si un équilibrage est nécessaire, si tel est le cas, l'agent de distribution est communiqué.

L'agent de distribution est la source de "connaissances" sur l'environnement (nœuds voisins et données statistiques sur les nœuds voisins) pour l'agent d'analyse, cette connaissance est proposée pour clarifier les règles selon lesquelles l'agent décide de la nécessité d'un équilibrage.

L'agent courtier contrôle, avec tous les agents, le fonctionnement des nœuds, lorsque le nœud a terminé sa partie du travail, il prend une partie du travail de l'autre nœud et la mélange avec un nœud libre, cela garantit que le nœud entier est toujours en cours de traitement.

L'agent de distribution, basé sur des règles, sélectionne les objets de modélisation à transférer et choisit les nœuds de réseau cible, lors de la sélection d'objets à transférer vers d'autres nœuds, il fait référence aux agents de distribution voisins. Ensuite, les agents de distribution sont synchronisés, ce qui permet de déterminer l'agent principal, après l'exécution de l'algorithme de synchronisation, l'agent principal achève le processus de simulation en envoyant un message correspondant au système de simulation, les agents demandent alors les objets nécessaires au système.

L'agent de migration déplace les objets vers d'autres nœuds et il envoie les objets du modèle de simulation aux agents de distribution des nœuds cibles, une fois la migration terminée, les agents de distribution sont à nouveau synchronisés et le système de simulation est lancé. Le processus de modélisation est en cours, cependant, dans ces études, les performances du système n'ont pas été étudiées, la plate-forme Hadoop implémentée est également simple et l'installation de son expérimentale peut ne pas être optimale et les résultats peuvent être trompeurs dans une certaine mesure.

Dans les études de [72], un modèle de protocole d'allocation de ressources distribuées (dRAPM) est proposé pour allouer et planifier des tâches sur une grille distribuée, en utilisant les propriétés des systèmes multi-agents, le protocole d'allocation de ressources réparties (dRAP) proposé est décrit comme suit: Un agent dans le système est simplement un nœud, et chaque agent dispose d'un vecteur comprenant le nombre de CPU

dans son cluster et le temps résiduel pour terminer l'exécution de son processus actuel. Chaque agent est assuré d'être dans un cas sur quatre lors de la simulation:

État 1 : Un agent n'a pas des tâches assignés et ne fait pas partie d'une cluster, dans ce cas, l'agent analyse la file d'attente, prend en compte les besoins en ressources Cpu_{req} des tâches non allouées et gère la tâche qui minimise l'équation suivante : Cpu_{req} -1

État 2: Un agent a été affecté à des tâches et ne fait pas partie d'une cluster Dans ce cas:

- L'agent continue d'exécuter la tâche et met à jour son vecteur d'informations.
- Si les exigences du tâche ne sont pas totalement satisfaites, l'agent interrogera ses voisins et tentera de former une cluster tel que: $CPU_{req} = CPU_{cluster}$
- Lorsque l'agent met fin à l'exécution du tâche, il revient à l'état 1.

État 3: un agent fait actuellement partie d'une cluster et n'a aucun tâche affecté.

- Dans ce cas, l'agent analyse la file d'attente, considère les travaux non alloués et assume le travail qui minimise l'équation: $CPU_{req} - CPU_{cluster}$

État 4: un agent a été affecté à un tâche et il fait actuellement partie d'une cluster. Dans ce cas:

- L'agent poursuit l'exécution du tâche et met à jour son vecteur d'informations.
- Une fois la tâche est terminé, l'agent se déconnecte de cluster et revient à l'état 1.

Une caractéristique principale de cet algorithme est que les nœuds demandent à leurs voisins de former des clusters, cela réduit le temps d'attente et les coûts de communication. Une optimisation à envisager serait de retarder la déconnexion de cluster dans l'état 4, ce qui guiderait l'apprentissage ou la mémoire dans le système où l'ordonnaceur serait capable de se souvenir des exigences du processus passé, mais le

problème avec cet algorithme est sa nature décentralisée, ce n'est ni un contrôle centralisé ni une synchronisation précise sur des nœuds (agents).

La principale contribution de cette étude [73] est le développement d'un modèle basé sur agent pour la gestion des ressources avec des opérations définies afin que l'utilisateur puisse effectuer des tâches de manière efficace, efficiente et ainsi améliorer considérablement la gestion par un middleware gLite Grid.

Le modèle de gestion des ressources basé sur les agents (ABRMM) fournit une plate-forme basée sur une collection d'agents dans une organisation virtuelle, les principaux aspects de cette architecture sont: le suivi des ressources, l'équilibrage de charge et la hiérarchie des agents.

Les agents qui composent cette plate-forme sont définis comme l'interface utilisateur de l'agent (AUI), l'agent de planification (SA), l'agent de recherche de ressources (RSA) et l'agent de surveillance (MA). AUI collecte des informations de base sur le type d'exigences en matière de ressources ou de demandes spéciales de ressources à l'aide d'une interface Web pour communiquer avec l'utilisateur.

Le SA est responsable de la planification de l'exécution des tâches avec les ressources appropriées, il a contacté RSA pour une liste des ressources disponibles, il trouve la meilleure ressource en fonction des demandes et de la disponibilité des machines. RSA donne une liste de ressources, RSA fait le lien entre les utilisations et les ressources. Ensuite, la fonctionnalité RSA bascule vers WM gLite, l'utilisateur peut choisir la ressource ou permettre son exécution de manière transparente, tout en respectant les critères établis.

L'agent de surveillance (MA) est responsable de la surveillance de l'état des ressources pour fournir des informations en ligne sur la disponibilité et, en même temps, il contrôle les tâches effectuées par chaque travail. Le système multi-agents MA interagit via une interface de communication, maintenant avec succès la découverte des services d'information actuels et la mise à jour du middleware gLite.

Dans cet article [74], les auteurs ont proposé une nouvelle structure d'équilibrage de charge basée sur l'agent mobile et l'approche d'optimisation par colonies de fourmis. Dans le modèle proposé (ACOM), un agent répartiteur est impliqué dans la distribution des

tâches reçues aux agents travailleurs selon les bonnes décisions afin de minimiser le temps d'exécution global (makespan).

Le cadre proposé est construit en utilisant trois couches qui sont le producteur des tâches utilisateur, la couche d'équilibrage de la charge et la couche des travailleurs. Cette étude doit être complétée en comparant leurs résultats avec d'autres méthodes, en minimisant les mouvements de tâches et en entraînant des coûts supplémentaires dans le processus de migration.

Une autre étude [75] a présenté la conception et la mise en œuvre d'un modèle d'équilibrage de charge floue basé sur les priorités (PBFLBM) dans une grille de calcul. Dans ce modèle, l'utilisateur envoie ses tâches à l'agent de grille, une fois que l'ordonnanceur de grille a utilisé l'algorithme d'ordonnancement basé sur la priorité pour planifier des tâches de l'agent de grille vers la ressource disponible. L'équilibrage de charge est effectué à l'aide de la technique de logique floue, dans laquelle un ensemble de règles floues est produit à l'aide de la ressource et du paramètre de tâche. Comme les règles de contrôle flou sont collectées à l'aide de variables linguistiques, la connaissance perceptuelle et l'inspection sont facilement intégrées dans le mécanisme de contrôle.

2.7 Conclusion

Dans ce chapitre nous avons donné un aperçu général sur les systèmes multi-agents. Egalement, nous avons montré l'intérêt de l'utilisation des systèmes multi-agents dans les grilles de calcul et la combinaison entre les deux techniques et les difficultés et les avantages qui entourent la tâche d'intégration des systèmes multi-agents dans les grilles de calcul . Nous avons aussi cité dans ce chapitre les travaux connexes relatifs à l'équilibrage de charge basé sur des agents dans les grilles de calcul.

Chapitre 3

Modèles et Algorithmes proposés

3.1 Introduction

Dans les deux premiers chapitres, nous présentons l'essentiel des notions de base sur l'équilibrage de charge dans les grilles de calcul, ainsi le système multi agents.

En s'appuyant sur quelques travaux de la littérature qui traitent le problème d'équilibrage de charge, différentes approches sont suivies en se basant sur l'arborescence, l'estimation, le voisinage, le partitionnement, les techniques de vie artificielle, l'approche hybride, et l'approche basée sur les agents. Afin de traiter ce problème, nous proposons deux modèles d'équilibrage de charge dans un environnement de grille, dont, le premier sert à traiter le problème par la migration de processus, alors que le deuxième est procédé par utilisation du système multi agents. Les systèmes multi-agents offrent de nombreux avantages en termes de fonctionnement coopératif et émergence. En effet, nous proposons une architecture multi-agents avec implémentation dans une plateforme multi-agents qui nous permettent d'étudier la contribution de l'utilisation de cette technologie dans le but de faciliter le déploiement des applications avec un comportement dynamique. En outre, ce modèle d'équilibrage de charge proposé vise à tirer parti des caractéristiques de l'agent pour générer un système autonome. Il traite également des inconvénients des systèmes similaires tels que l'instabilité, la non évolutivité ou la non adaptabilité.

Nous commençons ce chapitre par une description détaillée du modèle d'équilibrage de charge avec migration de processus, ainsi que le modèle de grille proposé. Ensuite, nous décrivons les algorithmes associés.

La deuxième partie du chapitre est consacrée à l'explication de notre principale contribution et à la modélisation de notre système reposant sur des agents. Suite à la présentation de notre contribution, nous détaillerons les éléments de conception de l'approche proposée en définissant les rôles de chacun des agents et en précisant ainsi le protocole d'interaction. Les algorithmes implémentés par des agents sont également discutées. Enfin, l'approche UML est employée pour présenter les fonctionnalités du notre système.

3.2 Modèle d'équilibrage de charge avec migration de processus dans une grille de calcul

Pour une gestion efficace des ressources dans la grille, la ressource surchargée doit être prohibée, ce qui peut être réalisé par la technique de migration des processus. Dans la migration des processus, un processus exécuté sur une ressource est redéployé sur une autre de manière à ce que la migration n'entraîne aucune modification dans l'exécution du processus, cela signifie que le processus est suspendu et repris plus tard sur une nouvelle ressource [28].

3.2.1 Topologie de la grille

La topologie de la grille utilisée comme support du modèle proposé, est décrite ci dessous dans la figure 3.1, et inspirée de l'architecture de grille de Belabass yagoubi [76].

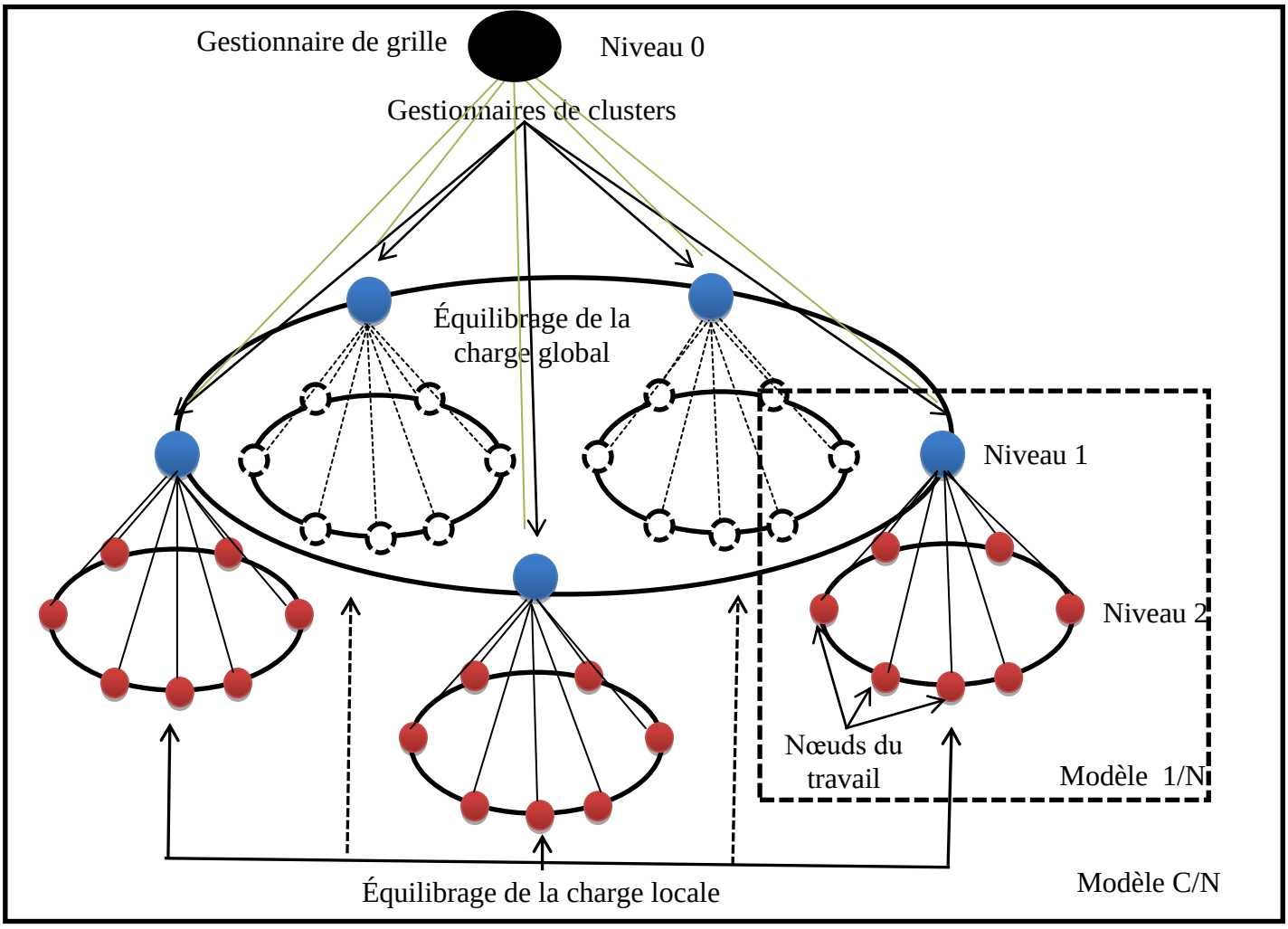


Figure 3.1. Représentation arborescente d'une grille [76].

Une grille de calcul a été modélisé comme un ensemble de clusters. Chaque cluster était composée de nœuds appartenaient à un domaine local LAN (Local Area Network). Chaque cluster était connectée au réseau mondial WAN (World Area Network) par un commutateur [76]. La grille est composée de 3 niveaux : la racine (niveau 0), un niveau intermédiaire contenant des gestionnaires de clusters (niveau 1) et le dernier qui s'occupe des feuilles (niveau 2). Ce dernier contenant les principaux nœuds du travail.

- **Niveau 0:** Dans ce premier niveau, nous trouvons un nœud virtuel appelé gestionnaire de grille qui correspond à la racine d'arbre. Ce nœud sert à réaliser les fonctions suivantes:
  - Recevoir les tâches soumises par les utilisateurs de grille,
  - Envoyer les tâches aux gestionnaires de clusters,
  - En cas d'échec, le gestionnaire de la grille supprime tous les tâches en cours d'exécution dans la ressource en échec en les faisant passer à l'état de redémarrage. La machine en panne sera également redémarrée.
- **Niveau 1**: Une fois qu'une tâche est allouée pour être exécutée sur un nœud particulier, ce dernier doit ensuite être géré par le gestionnaire de cluster, où, chaque gestionnaire de cluster à ce niveau est lié à une cluster physique. Le gestionnaire est responsable de :
  - Recevoir les tâches émises par le gestionnaire de grille,
  - Envoyer les tâches aux nœuds du travail,
  - Maintenir les informations de charge associées à chacune de ses ressources,
  - Estimer la charge de la cluster associée et la diffuser aux autres gestionnaires de cluster,
  - Décider de lancer l'équilibrage de charge locale que nous appellerons équilibrage de charge intra-cluster,
  - Envoyer les décisions de migration de processus aux ressources qu'ils gèrent pour l'exécution,
  - Lancer un équilibrage de charge globale que nous appellerons équilibrage de charge inter-cluster.

- **Niveau 2:** A ce niveau, nous trouvons les nœuds de travail liée à leurs clusters respectifs. Chaque nœud à ce niveau est responsable de:
  - Maintenir ses informations de charge telles que l'utilisation du processeur, la longueur de la file d'attente du processeur et la consommation de mémoire,
  - Envoyer les informations à son gestionnaire de cluster,
  - Exécuter la migration du processus décidée par son gestionnaire de cluster.

3.2.2 Caractéristiques du modèle proposé

Le modèle d'équilibrage de charge proposé est un modèle d'équilibrage hiérarchique associée à la migration de processus. Deux défis majeurs apparaissent pour les grilles de calcul qui sont l'hétérogénéité et le passage à l'échelle. Nous avons proposé une architecture à trois niveaux pour résoudre le problème de passage à l'échelle. Les caractéristiques du modèle proposé peuvent être résumées comme suit:

- **Hiérarchie:** cette caractéristique facilite la circulation de l'information à travers l'arbre, dont, trois types de mouvements d'information de charge sont différenciés :
  - **Mouvement ascendant:** ce mouvement concerne le mouvement d'information de charge qui vise à obtenir l'état actuel de la charge en passant du niveau 2 (les nœuds) vers le niveau 1 (gestionnaires de clusters). Avec ce mouvement, le gestionnaire de cluster peut avoir une vue globale de la charge de ce cluster aussi pour celle de d'autres.
  - **Mouvement horizontal :** il concerne les paramètres utiles lors de l'exécution des opérations d'équilibrage de charge. Ce mouvement concerne l'assignation de tâches intra-cluster au niveau 2. En plus, ce flux véhicule les informations de charge entre les gestionnaires de clusters sur le niveau 1.
  - **mouvement descendant**: ce mouvement permet de prendre des décisions pour l'assignation de tâches ou la migration de processus, dont, les décisions prises par les gestionnaires de clusters sont orientées du niveau 1 vers les nœuds du niveau 2.

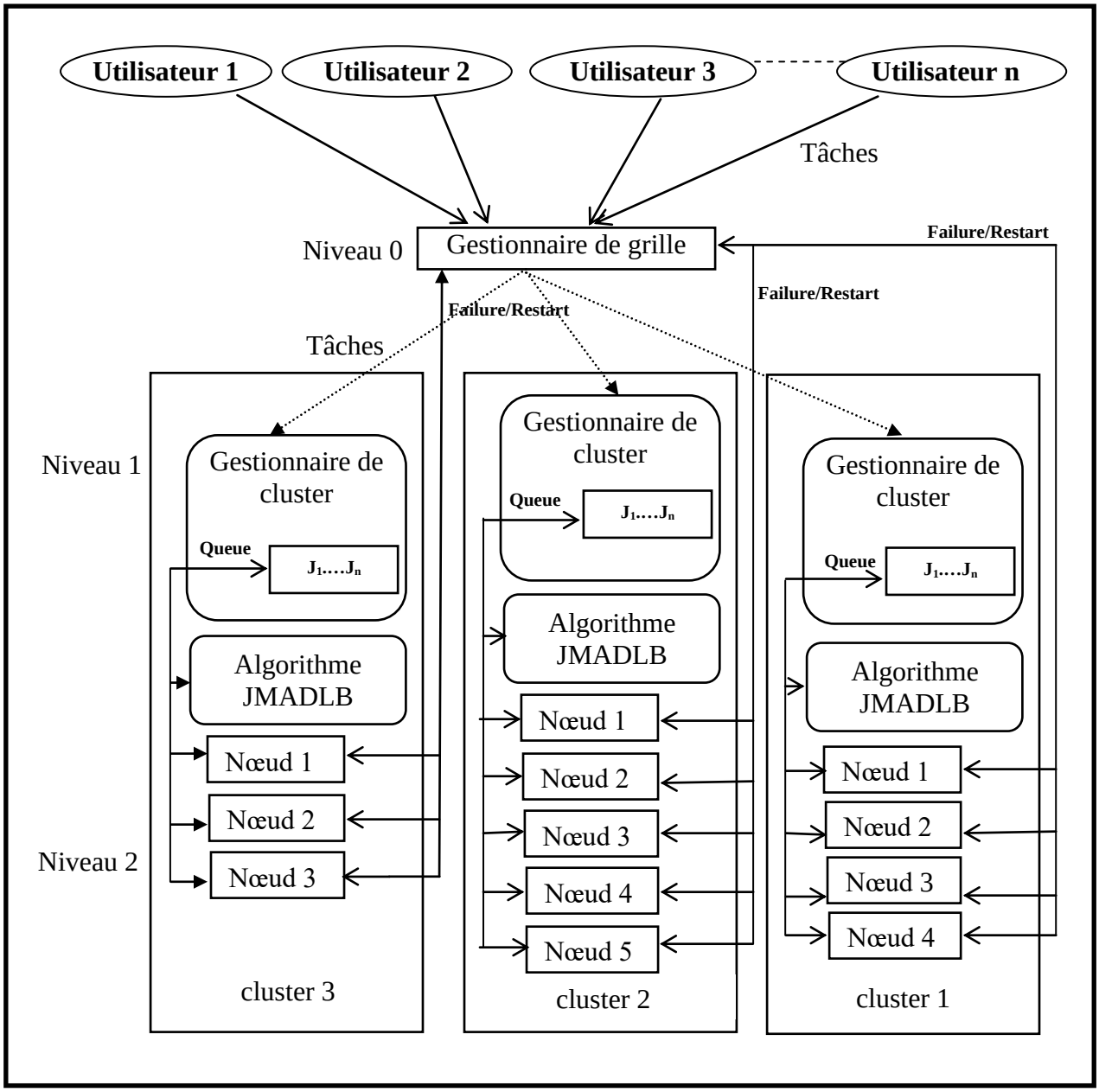


Figure 3.2 : l'architecture du système de modèle proposé

- **Passage à l'échelle et hétérogénéité:** l'insertion ou l'élimination d'entités (nœuds ou clusters) sont des opérations très simples dans le modèle proposé (insertion ou élimination de nœuds ou de sous-arbres).

- **Indépendance:** de toute structure physique de grille, la conversion d'une grille en arbre est une opération unique. Chaque grille correspond à un seul arbre.

- **Dynamicité:** La connexion ou la déconnexion des ressources (nœuds de travail ou clusters) correspondent à des opérations simples effectuées dans un arbre (ajout ou suppression de feuilles ou de sous-arbres).

3.2.3 Représentation structurelle du modèle proposé

La représentation structurelle du modèle est schématisée sous un diagramme UML (Figure 3.3).

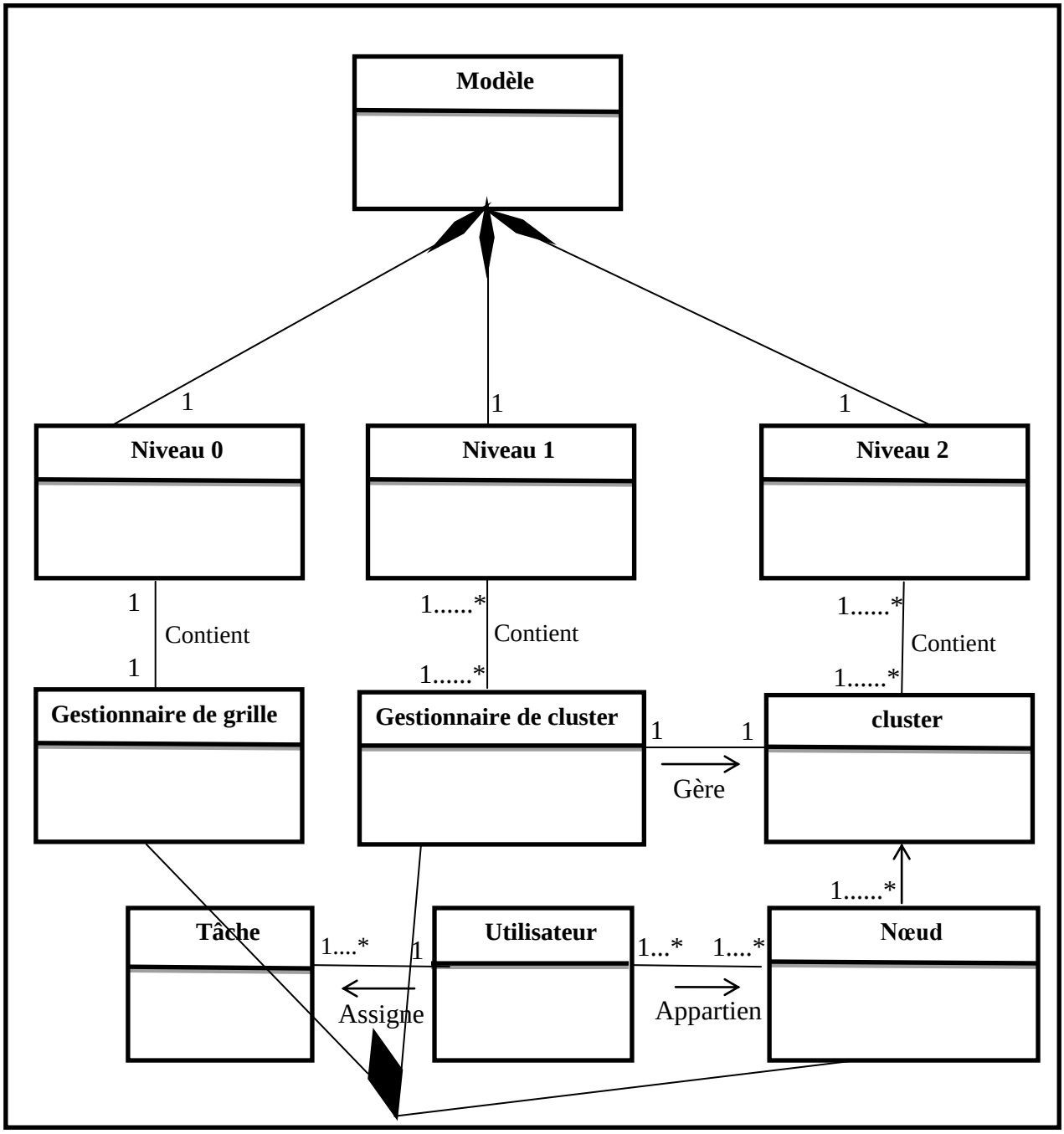


Figure 3.3 : Diagramme de classes associées au modèle proposé

3.2.4 Système d'équilibrage de charge

Le système d'équilibrage de charge proposé met en œuvre trois types de politiques: politique d'information, politique de sélection de la localisation et la politique de sélection

du candidat. Pour la mise en œuvre de politique d'information, nous utilisons une approche basée sur les événements. La stratégie FIFO est appliquée pour la mise en œuvre de la politique de sélection due candidat. Alors que pour la mise en œuvre de la politique sélection de localisation nous utilisons comme index de charge, la longueur de la file d'attente, l'utilisation du processeur et l'usage de la mémoire.

3.2.4.1 Politique d'information

Nous considérons certains événements spécifiques qui modifient la configuration de charge dans l'environnement de grille qui peuvent être classés comme suit:

- Une nouvelle tâche est arrivée,
- Réalisation de l'exécution de toute tâche,
- Une nouvelle ressource est arrivée,
- Toute ressource existante est un retrait,
- Une machine en panne à n'importe quelle ressource,
- Le nœud est surchargé.

Lorsqu'un de ces événements se produit, la valeur de charge locale est modifiée et l'information de charge de nœud est mise à jour.

La charge du nœud à un moment donné est décrite simplement par la longueur de la file d'attente du processeur. Il indique le nombre de processus qui attendent d'être exécutés. Dans notre proposition, nous avons considéré chacun de la CPU-U (utilisation du processeur), la longueur Q (longueur de la file d'attente) et la Mem (utilisation de la mémoire) comme paramètres d'information de charge pour mesurer la charge d'un nœud. Ces paramètres sont calculés comme suit :

Load (CPU-U)= $(U_1+U_2+\ldots\ldots+U_T)/T$ où: U_1, U_2,……, U_T est la valeur de CPU-U dans un intervalle d'une seconde précédent et T est le nombre d'intervalles de temps,

Load (Qlength) = $(Q_1+Q_2+\ldots\ldots...+Q_T)/T$ Où: Q_1, Q_2,……..., Q_T est la valeur de Qlength dans un intervalle d'une seconde précédent et T est le nombre d'intervalles de temps,

Load (Mem) = $(M_1+M_2+\ldots\ldots...+M_T)/T$ Où : M_1, M_2,……..., M_T est la valeur de Mem dans un intervalle d'une seconde précédent et T est le nombre d'intervalles de temps.

Les informations moyennes de CPU-U, Qlength et Mem sont les paramètres de charge utilisés pour décrire la charge de nœud.

3.2.4.2 Politique de sélection de localisation

Cette politique classe les nœuds en fonction de leurs paramètres de charge. Elle utilise trois états pour la classification: surchargé, sous-chargé et équilibré. Dans un premier temps, nous devons calculer deux valeurs de seuil pour chaque paramètre de charge (CPU-U et Qlength). Le calcul de ces seuils se fait comme suit:

$Load_{avg}$ (Qlength) = ($load_1$+$load_2$+….load $_{NBRN}$) / NBR_N; Où : $Load_{avg}$ (Qlength) représente la charge moyenne de Qlength sur touts les nœuds. $load_1$, $load_2$,….load $_{NBRN}$ est la valeur actuelle de Qlength de chaque nœud

$Load_{avg}$ (CPU-U) = ($load_1$+$load_2$+….$load_{NBRN}$) / NBR_N; Où : $Load_{avg}$ (CPU-U) représente la charge moyenne de CPU-U sur touts les nœuds. $load_1$, $load_2$,….$load_{NBRN}$ est la charge actuelle de CPU-U de chaque nœud

Calculer les valeurs de seuils

Les valeurs de seuils inférieures et supérieures des paramètres de charge sont calculées en multipliant la charge moyenne de chaque paramètre par une valeur constante.

TH_H=H*$Load_{avg}$
TH_L=L* $Load_{avg}$

Où TH_H est le seuil supérieur et TH_L est le seuil inférieur H et L sont des constantes.

L'étape suivante consiste à partitionner les nœuds à des nœuds équilibrés, surchargés ou sous-chargés en utilisant les valeurs de seuil comme suit:

Load= { Equilibré,
Surchargé (CPU-U est supérieur) ou (Qlength est supérieur) ou (Mem>85%),
Sous-chargé (CPU-U est inférieur) ou (Qlength est inférieur),
Idle CPU-U<=1%.

- **Surchargé:** le nœud sera ajouté pour la liste surchargée, si l'une de ces conditions est vraie:
  - La valeur CPU-U est élevée, si l'utilisation du processeur est élevée, l'état du nœud est surchargé
  - La valeur Qlength est élevée, si le nombre de tâches dans la file d'attente de nœud est élevé, alors le nœud est classé comme nœud surchargée.

- L'utilisation de la mémoire est supérieure à 85%, le nœud avec moins de 15% de mémoire libre risque de paginer la mémoire et la pagination ralentira le traitement sur ce nœud.

- **Sous-chargé:** le nœud sera ajouté pour la liste sous-chargée si l'une de ces conditions est vraie:
  - La valeur CPU-U est faible; si l'utilisation du processeur est faible, le nœud est ajouté pour la liste sous-chargée.
  - La valeur Qlength est faible; si le nombre des travaux dans la file d'attente de nœud est faible, le nœud est classé à l'état sous-chargé.

- **Équilibré:** les nœuds ne sont pas dans la liste surchargée et ni à la liste sous-chargée sont des nœuds dans un état de charge équilibrée. Ils sont considérés comme plus chargés que l'état sous chargé et moins chargés que l'état surchargé.

3.2.4.3 Politique de sélection de candidat

Suivant notre politique de sélection et afin de déterminer les processus à migrer d'un nœud surchargé vers un nœud sous-chargé, nous utilisons le critère FIFO (First In First Out), où, une sélection de la tâche introduite en premier dans la file d'attente est procédée.

3.2.5 Algorithmes proposés

Conformément au modèle proposé, nous distinguons deux niveaux d'équilibrage de charge: l'algorithme d'équilibrage de charge intra-cluster et l'algorithme d'équilibrage de charge inter-cluster.

Les notations utilisées dans nos algorithmes sont récapitulés dans le tableau 3.1 :

Tableau 3.1 : Notation utilisées

Paramètre	Description
N	Noeud
$Load_N$	Charge de Noeud
Qlength	La longueur de la file d'attente de CPU
TH_H	Seuil supérieur
TH_L	Seuil inférieur
OLD-list	Liste surchargeé
ULD-list	List sous chargeé
BLD-list	Liste équilibreé
$Load_{avg}$	Charge moyenne
NBR_N	Nombre de noeuds de cluster c
C	cluster

3.2.5.1 Algorithme d'équilibrage de charge intra-cluster

En fonction de sa charge actuelle, chaque gestionnaire de cluster décide de démarrer une opération de migration de processus. Dans ce cas, le gestionnaire de cluster essaie, en priorité, d'équilibrer sa charge entre ses nœuds. Pour implémenter l'équilibrage de charge local, nous proposons les algorithmes suivants:

```
Gathering information algorithm
Begin
T←5 seconds ;  Waiting for jobs;
Create jobs queue for each node;
For every Node N and in each one second of T intervals do
        Calculate (CPU-U);
        Calculate (Qlength);
        Calculate (Mem);
End For ;
Load (CPU-U) = (U0+U1+…..UT)/T;  Load (Qlength) = (Q0+Q1+…..QT)/T;
Load (Mem) = (M0+M1+…..MT)/T;
//According to proposed events //  Cluster manager receives Load informations from all nodes ;
Cluster manager compute load of cluster C associated;
Cluster manager Sends Load information of C to other clusters managers ;
Loop
wait for load change // happening of any of defined events
if (events_happens ()=1 or  events_happens ()=4) then   begin
Remove terminated or migrated job from the waiting queue;
Subtract their load value from the total local load of node;
Send new load to its cluster manager associated ;
End
if (events_happens ()=2 or  events_happens ()=3) then
begin
Add the newly created or incoming job for the waiting queue;
Add their load value for the total local load of node;
Send new load to its cluster manager associated;
End
If ((events_happens () =6) and (events_happens () =7)) then begin
ask the slowest resource to send a portion of its load to the idle resource;
End
If ((events_happens () =8) then  Inter-cluster Load Balancing Algorithm;
End Loop
```

```
Function events_happens ()
output Type: integer
begin
If (Job.state=Termination) then events_happens () =1;
If (Job.state=Start) then events_happens () =2;
If (Job.state=Incoming Migrating ) then events_happens ()=3;
If (Job.state = migrated) then events_happens ()=4;
If (Arrival of any new resource) then events_happens ()=5;
If (resouce.state= idle) then    events_happens () =6;
If (resource.state= slowest) then events_happens ()=7;
If (cluster.state=saturated then events_happens ()=8;
If (cluster.state=unbalanced) then events_happens ()=9;  End.
```

L'algorithme suivant classe les nœuds en fonction de leur charge. Il a utilisé trois états pour la classification: surchargé, sous-chargé et équilibré.

Location policy algorithm
Begin
If (events_happens ()=8) then Inter-cluster load balancing algorithm
somme ←0; somme1←0;
For every Node N of cluster C do
Somme ←somme+ $Load_N$(CPU-U) ;
Somme1← Somme1+ $Load_N$(Qlength) ;
End For
$Load_{avg}$(CPU-U)= somme/NBR_N;
$Load_{avg}$(Qlength)= somme1/NBR_N;
TH_H(CPU-U)= $Load_{avg}$(CPU-U)*H; TH_L(CPU-U)= $Load_{avg}$(CPU-U)*L;
TH_H(Qlength)= $Load_{avg}$(Qlength)*H; TH_L(Qlength)= $Load_{avg}$(Qlength)*L;
Partionning Nodes into overloaded list OLD-list , underloaded list ULD-list and balanced list BLD-list
OLD-list← ∅; ULD-list←∅; BLD-list←∅;
For every Node N of cluster C **do**
If ($Load_N$(Qlength) >TH_H(Qlength) or $Load_N$(Qlength) >TH_H(Qlength) or Load (Mem)>85%) **then**
OLD-list ←OLD-list ∪ N;
Else If ($Load_N$(Qlength) < TH_L(Qlength) or $Load_N$(Qlength) < TH_L(Qlength)) **then**
ULD-list← ULD-list ∪ N
Else BLD-list← BLD-list ∪ N;
End If
End For
Sort OLD_list by descending order relative to their $Load_N$.
Sort ULD_list by ascending order relative to Their $Load_N$.
End.

Au cours de l'étape qui suit la classification des nœuds, le gestionnaire de cluster décide de migrer les processus des nœuds surchargés vers les nœuds sous-chargés par application de l'algorithme suivant:

Job Migration Decision algorithm
begin
While (OLD-list ≠ ∅ .AND. ULD-list ≠ ∅) **do**
For i = 1 To ULD-list.Size() **do**
Select job from queue of first node
belonging to OLD-List by FCFS algorithm
Migrate the selected job from first
Sender node of OLD-List to i^{th} receiver
node of ULD-list;
Update the current $Load_N$ of receiver
and sender nodes;
Update OLD-list, ULD-list and BLD-list;
Sort OLD-list by descending order of their
$Load_N$;
End For
End

3.2.5.2 Algorithme d'équilibrage de charge inter-cluster

Cet algorithme applique un équilibrage de charge global entre tous les clusters de la grille. L'équilibrage de charge inter-cluster à ce niveau est effectué dans le cas où un gestionnaire de cluster n'arrive pas à équilibrer sa charge entre ses nœuds associés. Lorsque la charge courante d'une cluster surpasse sa capacité maximale, il est évident qu'il ne sert à rien de l'équilibrer puisque tous ses nœuds sont saturés. Nous définissons un autre seuil, qui prend la nomination de seuil de saturation, dont, dans ce cas, le gestionnaire de cluster transfère des processus vers des clusters sous-chargés en fonction des informations de charge reçues par le gestionnaire. Nous proposons l'algorithme suivant:

```
Inter-cluster Load Balancing Algorithm
Begin
if (events_happens ()=6) then  // cluster is saturated
   Create underloaded_clusters_table;
   While (underloaded _clusters_table ≠ Φ) Do
   For j = 1 To size(underloaded _clusters_table) Do
        Sort the clusters Cr of underloaded _clusters_table by ascending order of inter
        clusters(Ci-Cr)   WAN bandwidth sizes;
        Sort nodes of saturated cluster by descending order of their load;
        Sort Jobs of first node of saturated cluster by  FCFS algorithm and communication cost;
        Migrate the selected job from the first node of saturated cluster to j^th cluster of
        underloaded_clusters_table;
        Update its load;
        Update ULD_clusters_table, and its load ;
 End For
End.
```

3.3 Modèle d'équilibrage de charge basé sur des agents dans une grille de calcul

Le système multi-agents offre des fonctionnalités prometteuses aux gestionnaires de ressources. La réactivité, la proactivité, l'évolutivité, la coopération, la robustesse, la flexibilité et l'autonomie, en tant que propriétés des agents, peuvent aider dans la gestion des ressources dans des environnements dynamiques et changeants. Dans cette contribution nous proposons un modèle d'équilibrage de charge hiérarchique basé sur un système multi-agents. Une architecture hiérarchique avec coordination est conçue pour améliorer le passage à l'échelle et une certaine efficacité, également. De plus, une approche multi-agents est appliquée pour améliorer l'adaptabilité. Le modèle proposé vise à tirer parti des caractéristiques de l'agent pour créer un système autonome.

3.3.1 Modèle proposé

Dans le modèle proposé, nous utilisons la topologie de grille expliquée dans la section 3.2.1. Nous proposons une architecture à trois niveaux pour résoudre le problème de passage à l'échelle dans la grille.

- **Niveau 0**: à ce niveau, nous trouvons l'Agent Grille, où, les utilisateurs de la Grille envoient leurs tâches à cet agent. Ce dernier est responsable de:
  - Recevoir les tâches des utilisateurs de la grille,
  - Envoyer les tâches aux agents nœuds,
  - Démarrer tous les Agents clusters.
- **Niveau 1**: à ce niveau, l'Agent Cluster est associé à une cluster physique; cet agent est responsable de:
  - Sauvegarder les informations de charge relatives à chacun de ses nœuds,
  - Estimer la charge du cluster associé et la distribuer au autres Agents Cluster,
  - Lancer un équilibrage de charge locale,
  - Envoyer les décisions d'équilibrage de charge à l'Agent Migration,
  - Lancer un équilibrage de charge globale,
  - Agent Migration est démarré par son Agent Cluster associé,
  - Tous les Agents Noeud sont démarrés par leur Agent Cluster correspondant.

  L'Agent Migration est également présent à ce niveau, dont son rôle est de:
  - Démarrer le processus de migration,
  - Envoyer les décisions de migration aux Agents Nœuds,
  - Attendre un accusé de réception du nœud récepteur et s'assurer que les processus migrés sont reçus et repris avec succès au nœud de destination.
- **Niveau 2:** à ce niveau, nous trouvons les Agents Nœud et les Agents LBC; il est nécessaire d'avoir un Agent Nœud sur chaque nœud; chaque Agent Nœud à ce niveau est responsable de:
  - Maintenir ses informations de charge,
  - Envoyer ces informations à son Agent Cluster associé,
  - Travailler en coopération avec Agent Migration pour exécuter le processus de migration,
  - Recevoir les tâches envoyées par l'Agent Grille,

- Tous les Agents LBC sont démarrés par Agents Nœud.

À ce niveau, nous trouvons aussi l'Agent LBC, qui réside à chaque nœud et son rôle est de:

- Collecter des informations sur les tâches (nombre de tâches mis en file d'attente sur le nœud, heure d'arrivée, temps d'attente, heure de soumission, heure de début, heure de traitement et heure de fin de chaque tâche sur le nœud local),
- Supprimer les tâches terminés, sortants ou migrés de la file d'attente des tâches,
- Calculer la charge totale du nœud,
- Envoyer des informations de charge à l'Agent Node associé.

3.3.2 Caractéristiques du modèle

Le modèle proposé a quelques caractéristiques uniques, dont, les contributions les plus importantes sont:

- **Hiérarchie**: cette caractéristique facilite la circulation de l'information à travers l'arbre et définit le flux de messages dans le modèle proposé.

- **Tolérance aux pannes**: le modèle offre une tolérance aux pannes grâce à son auto-configurabilité, où les nœuds quittant le système seront remplacés par de nouveaux nœuds, ainsi, les tâches seront également réaffectées à d'autres nœuds lorsqu'un nœud quitte sans terminer sa tâche actuelle.

- **Passage à l'échelle**: il prend en charge l'évolutivité des grilles: l'insertion ou l'élimination des entités (nœuds ou clusters) sont des opérations très simples dans le modèle proposé.

- **Hétérogénéité**: le modèle proposé prend en compte l'hétérogénéité des ressources afin qu'elle soit complètement indépendante de toute architecture de réseau physique.

- **Flexibilité et extensibilité** : il prend en charge la flexibilité et l'extensibilité de divers agents intelligents qui ont été déployés pour réduire la complexité du système par modularisation. Il est facile de modifier ses composants et d'y ajouter plus de fonctions et de fonctionnalités.

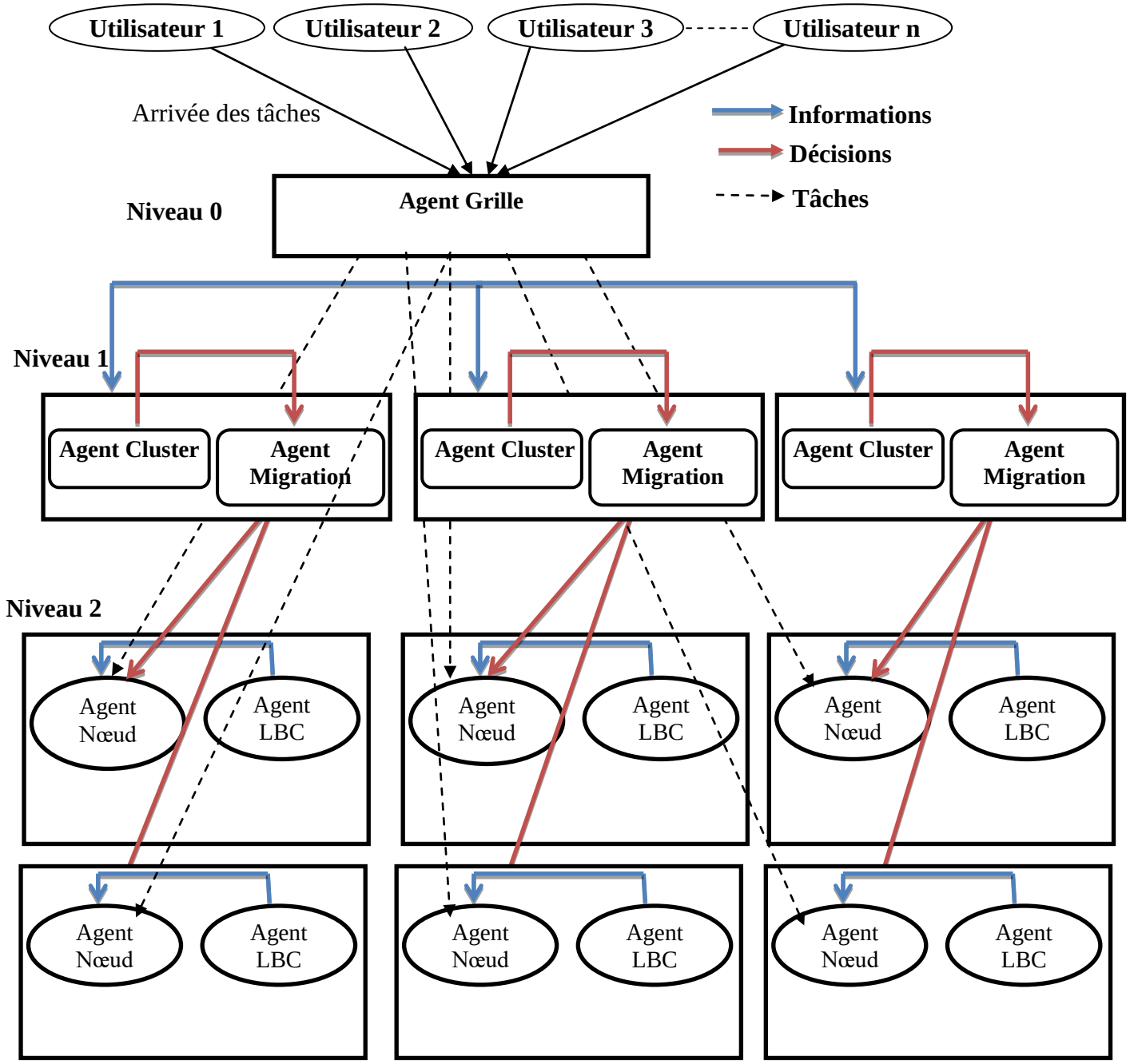


Figure 3.4: Architecture générale du modèle proposé

- **Échange de connaissances** : un avantage important du modèle de l'agent réside dans la possibilité d'échanger des informations entre les agents de la grille.

3.3.3 La communication entre les agents

Le système proposé comprend cinq types d'agents: Agent Grille, Agent Cluster, Agent Migration, Agent Nœud et Agent LBC.

Agent LBC

L'Agent LBC résidant à chaque nœud de calcul sert à calculer la charge du nœud local, aussi il crée la file d'attente des tâches et la met à jour si nécessaire. Par la suite, il envoie les informations de charge pour l'Agent Nœud en fonction des événements définis.

Agent Nœud

L'Agent Nœud envoie la mise à jour de valeur de charge locale à l'Agent Cluster qui met à jour ses informations de charge

Agent Cluster

En cas de détection d'un déséquilibre de charge entre les nœuds sous le contrôle de l'Agent Cluster, ce dernier analyse la charge actuelle du cluster sur la base des informations de charge reçues des Agents Nœud associés,

Agent Migration

L'Agent Migration est responsable de la migration des processus vers le nœud sous-chargé sélectionné. Il garantit aussi que la tâche est correctement reçu et repris ou démarré sur le nœud de destination. L'Agent Migration attend un accusé de réception du nœud récepteur une fois recevoir la tâche migré. Par la suite, l'accusé est renvoyé à l'Agent Cluster associé.

Agent Grille

Le rôle de l'Agent Grille est la distribution des tâches entre les nœuds, dont tous les Agents Cluster sont démarrés par ce type d'agents.

Afin de représenter les interactions entre les agents, nous optons pour la modélisation UML, souvent utilisée pour décrire l'interaction des différentes agents.

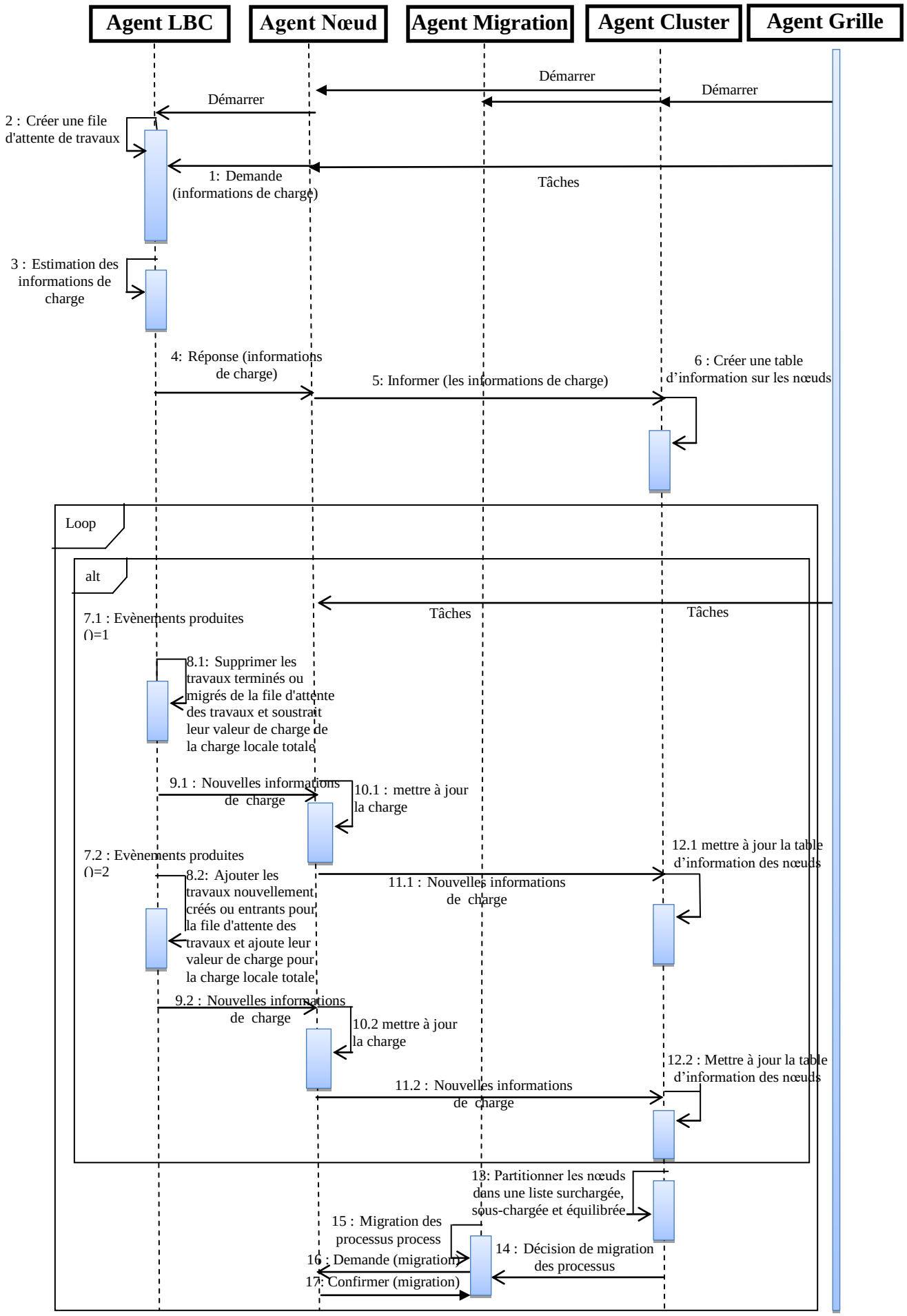

Agent LBC
Agent Nœud
Agent Migration
Agent Cluster
Agent Grille
Démarrer
Démarrer
Démarrer
2 : Créer une file d'attente de travaux
1: Demande (informations de charge)
Tâches
3 : Estimation des informations de charge
4: Réponse (informations de charge)
5: Informer (les informations de charge)
6 : Créer une table d'information sur les nœuds
Loop
alt
Tâches
Tâches
7.1 : Evènements produites ()=1
8.1: Supprimer les travaux terminés ou migrés de la file d'attente des travaux et soustrait leur valeur de charge de la charge locale totale
9.1 : Nouvelles informations de charge
10.1 : mettre à jour la charge
7.2 : Evènements produites ()=2
12.1 mettre à jour la table d'information des nœuds
8.2: Ajouter les travaux nouvellement créés ou entrants pour la file d'attente des travaux et ajoute leur valeur de charge pour la charge locale totale
11.1 : Nouvelles informations de charge
9.2 : Nouvelles informations de charge
10.2 mettre à jour la charge
12.2 : Mettre à jour la table d'information des nœuds
11.2 : Nouvelles informations de charge
13: Partitionner les nœuds dans une liste surchargée, sous-chargée et équilibrée
15 : Migration des processus process
14 : Décision de migration des processus
16 : Demande (migration)
17: Confirmer (migration)

Figure 3.5: Le diagramme de séquence UML décrit les interactions des agents dans le processus d'équilibrage de charge intra-cluster

La figure 3.5 représente le diagramme de la séquence UML associée au processus de négociation proposé.

3.3.4 Algorithmes proposés

Conformément au modèle proposé, nous distinguons deux niveaux d'équilibrage de charge: l'algorithme d'équilibrage de charge intra-cluster et l'algorithme d'équilibrage de charge inter-cluster.

3.3.4.1 Algorithmes Intra-cluster Agent based load balancing

Les algorithmes de divers agents déployés dans le modèle proposé sont présentés ci-dessous dans les figures, respectivement.

Algorithme Agent LBC

Begin
T←5 seconds
Waiting for jobs;
Create jobs queue for related node;
In each one second of T intervals **do**
Calculate (CPU-U);
Calculate (Qlength);
Calculate (Mem);
End do
Load (CPU-U) = (U0+U1+…..UT)/T;
Load (Qlength) = (Q0+Q1+…..QT)/T;
Load (Mem) = (M0+M1+…..MT)/T;
Wait for load change // happening of any of defined events
if (events_happens ()=1 or events_happens ()=4) **then** // Termination or migration of job
Remove terminated or migrated jobs from the queue
Subtract their load value from the total local load of node.
Send new load to its Agent Nœud associated;
End if
if (events_happens ()=2 or events_happens ()=3) **then** // new or incoming job
Add the newly created or incoming job for the waiting queue
Add their load value for the total local load of node
Send new load to its Agent Nœud associated;
End if
End

Function events_happens ()
output Type: integer
begin
If (Job.state=Termination) **then** events_happens () =1;

If (Job.state=Start) **then** events_happens () =2;
If (Job.state=Incoming Migrating) **then** events_happens ()=3;
If (Job.state = migrated) **then** events_happens ()=4;
If (Arrival of any new resource) **then** events_happens ()=5;
If (Node.state= idle) **then** events_happens () =6;
If (Node.state= slowest) **then** events_happens ()=7;
If (cluster.state=saturated **then** events_happens ()=8;
If (cluster.state=unbalanced) **then** events_happens ()=9;
End.

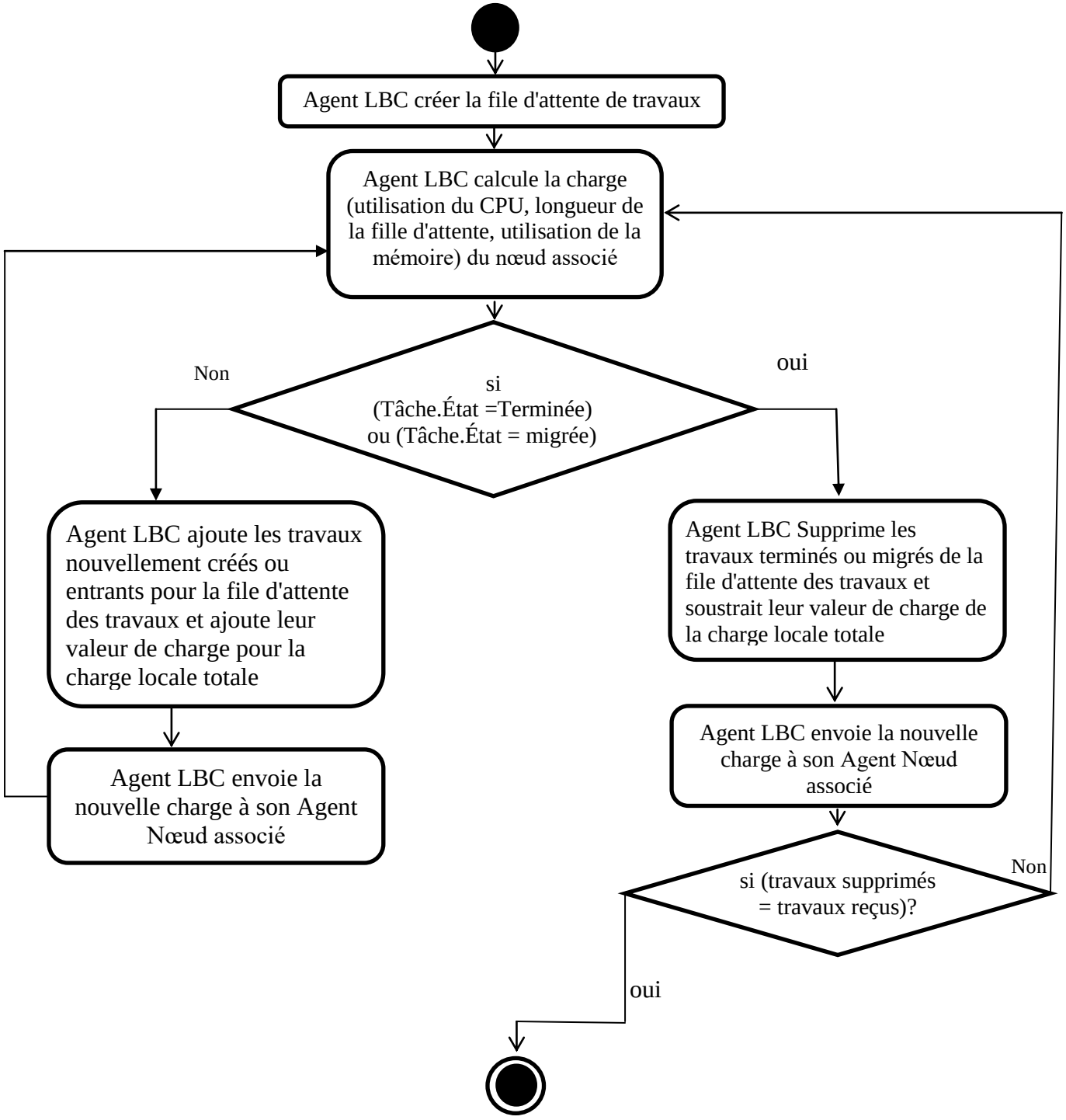


Figure 3.6: Organigramme pour l'algorithme Agent LBC

Algorithme Agent Noeud
Begin
Receive jobs from Agent Grille
Startup its related Agent LBC
Receive load information from its Agent LBC
Send load information for related Agent Cluster
Perform the job migration process
End.

Algorithme Agent Cluster

Begin
Startup its related Agents Nœud
Startup its related Agent Migration
Receive load information ($Load_N$(Qlength), $Load_N$(CPU-U)) from its related Agents Nœud
Calculate and send its load information for other Agents Cluster.
Receive and save load information from other clusters
somme←0; somme1←0;
For every Node N of cluster C **do**
Somme← Somme+ $Load_N$(Qlength);
Somme1← Somme1+ $Load_N$(CPU-U);
End For
$Load_{avg}$(Qlength) = somme1/NBR_N;
$Load_{avg}$(CPU-U) = somme/NBR_N;
TH_H(Qlength) = $Load_{avg}$(Qlength)*H;
TH_L(Qlength) = $Load_{avg}$(Qlength)*L;
TH_H(CPU-U) = $Load_{avg}$(CPU-U)*H;
TH_L(CPU-U) = $Load_{avg}$(CPU-U)*L;
Partition Nodes into overloaded list OLD-list, underloaded list ULD-list and balanced list BLD-list
OLD-list←∅; ULD-list←∅; BLD-list←∅;
For every Node N of cluster C **do**
If (($Load_N$(Qlength)>TH_H(Qlength)) **or** ($Load_N$(CPU-U)> TH_H(CPU-U))**or** (Load(Mem)>85%)) **then**
OLD-list ←OLD-list ∪ N;
End If
Else If (($Load_N$(Qlength))< TH_L(Qlength))**or** ($Load_N$(CPU-U)<TH_L(CPU-U))) **then**
ULD-list← ULD-list ∪ N;
Else BLD-list← BLD-list ∪ N;
End If . End For.
Sort OLD_list by descending order relative to their $Load_N$(Qlength).
Sort ULD_list by ascending order relative to Their $Load_N$(Qlength).
If (events_happens ()=7) **then** //cluster is unbalanced
While (OLD-list ≠ ∅.AND. ULD-list ≠ ∅) **do**
For i = 1 To ULD-list.Size() **do**
Send the decision of migration for AgentMigration (with address of first sender node of OLD List and i^{th} receiver node of ULD-list); **End For**
If an Acknowledgment received from AgentMigration **then**
Update the current $Load_N$ of receiver and sender nodes
Update OLD-list, ULD-list and BLD-list;
Sort OLD-list by descending order of their $Load_N$(Qlength).;
End For

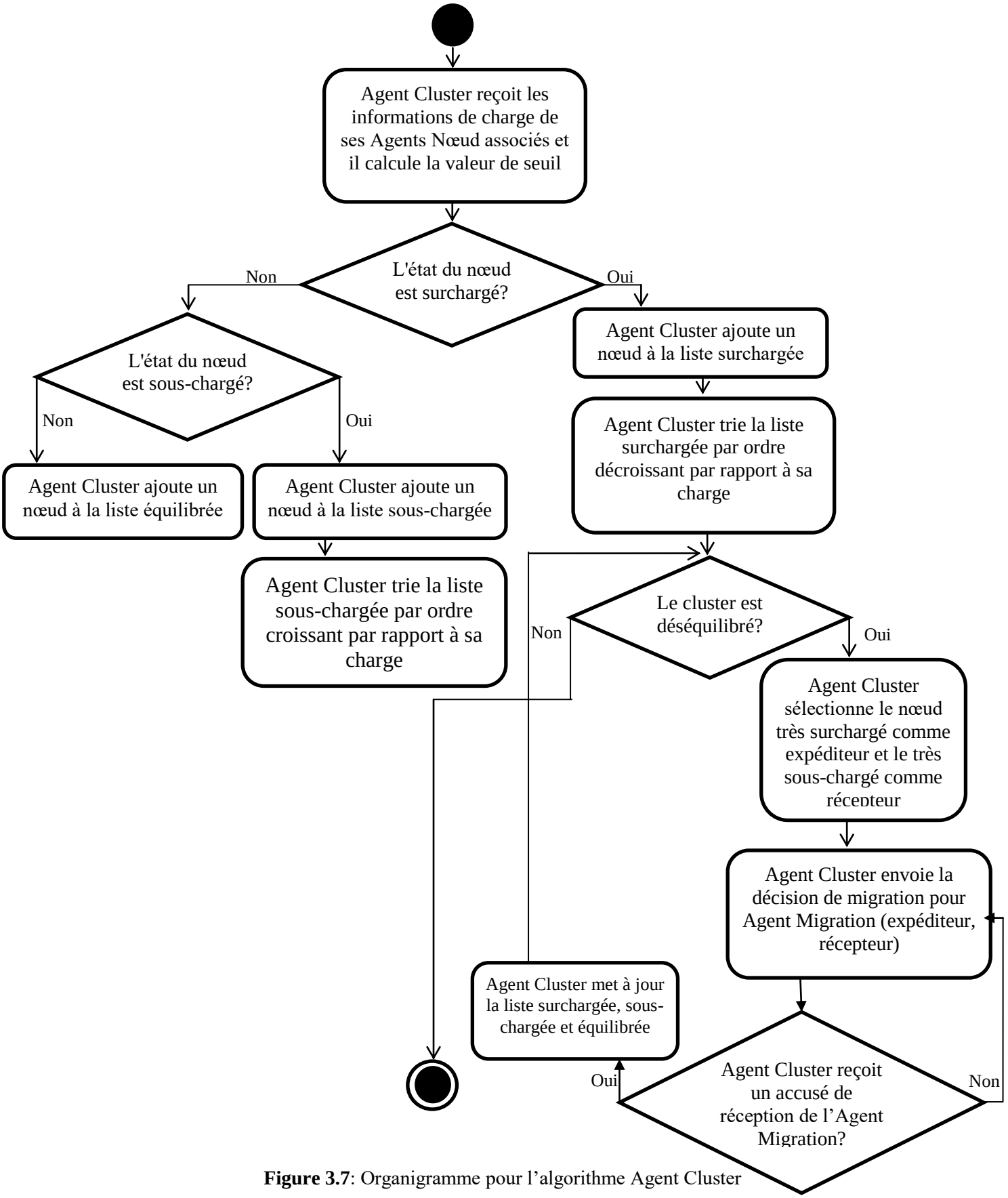


Figure 3.7: Organigramme pour l'algorithme Agent Cluster

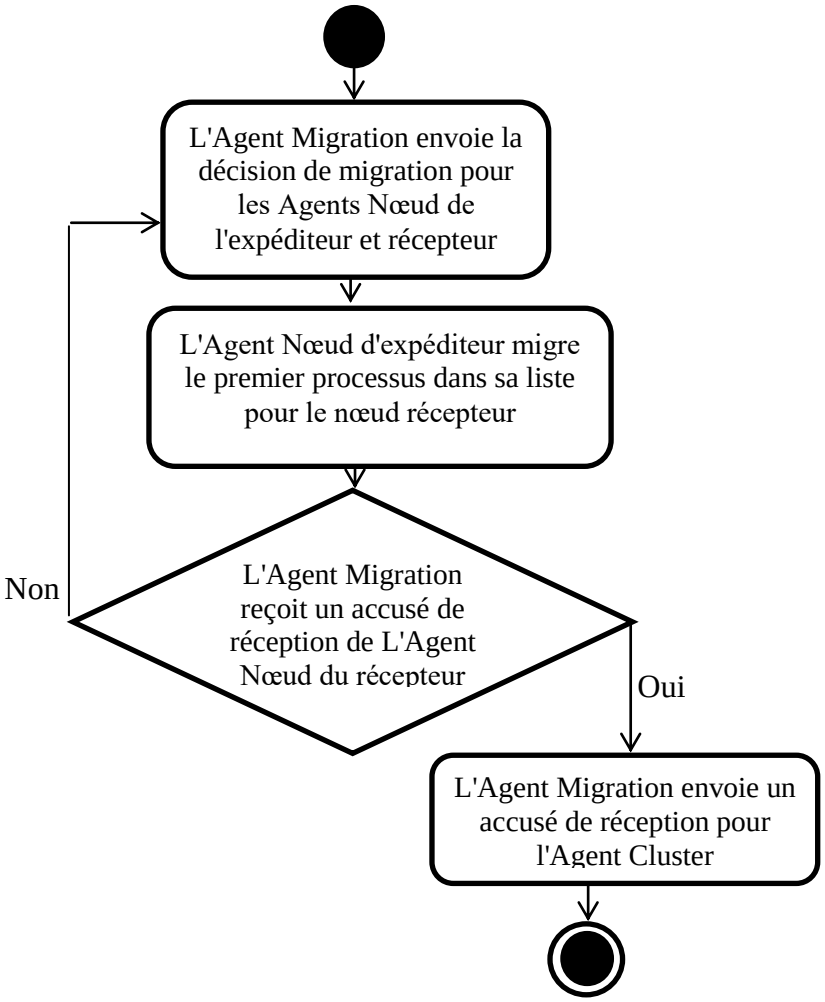


Figure 3.8: Organigramme pour l'algorithme Agent Migration

Suite à la classification des nœuds, l'étape suivante consiste à que l'Agent Cluster décide de transférer les travaux des nœuds surchargés vers les nœuds sous-chargés. Il envoie cette décision pour Agent Migration.

Algorithme Agent Migration
Begin
Receive decision of migration from its related Agent Cluster.
Sending the migration decisions to the Agents Nœud of sender and receiver node.
Wait for an Acknowledgment from the Agent Nœud of receiver node.
Send an Acknowledgment for its related Agent Cluster
End.

3.3.4.2 Algorithmes Inter-cluster Agent based load balancing

Si l'Agent Cluster ne parvient pas à équilibrer sa charge entre ses nœuds associés. Dans ce cas, c'est l'Agents Cluster qui va transférer des travaux vers des clusters sous-chargés en fonction des informations de charge reçues par d'autres Agents Cluster. Les algorithmes suivants sont proposés:

```
Algorithme AgentCluster
Begin
if (events_happens ()=6) then // cluster is saturated
    Create underloaded_clusters_table;
    While (underloaded clusters_table ≠ Φ) Do
  For j = 1 To size(underloaded _clusters_table) Do
    Begin
    Sort the clusters Cr of underloaded clusters_table by ascending order of inter- Clusters (Ci-Cr) WAN bandwidth sizes.
    Sort nodes of saturated cluster by descending order of their load
    Sort Jobs of first node of saturated cluster by FCFS algorithm and communication cost
    Migrate the selected job from the first node of saturated cluster to jth cluster of underloaded_clusters_table
    Update ULD_clusters_table, and its load .
    End For
End
End
```

Le dernier algorithme est implémenté dans l'Agent Cluster qui détermine la façon dont un cluster récepteur est sélectionné pour un processus migré à partir d'un cluster surchargé. L'Agent Cluster du cluster saturé calcule le coût de communication minimal de l'envoi des travaux aux clusters sous-chargés en fonction des informations collectées au cours du dernier intervalle d'échange. L'Agent Cluster sélectionne le cluster qui donne le coût global minimal.

```
Algorithm Agent Grille
Begin
Startup all Agents Cluster
Receive jobs from grid user
Send jobs for Agents Noeud
End
```

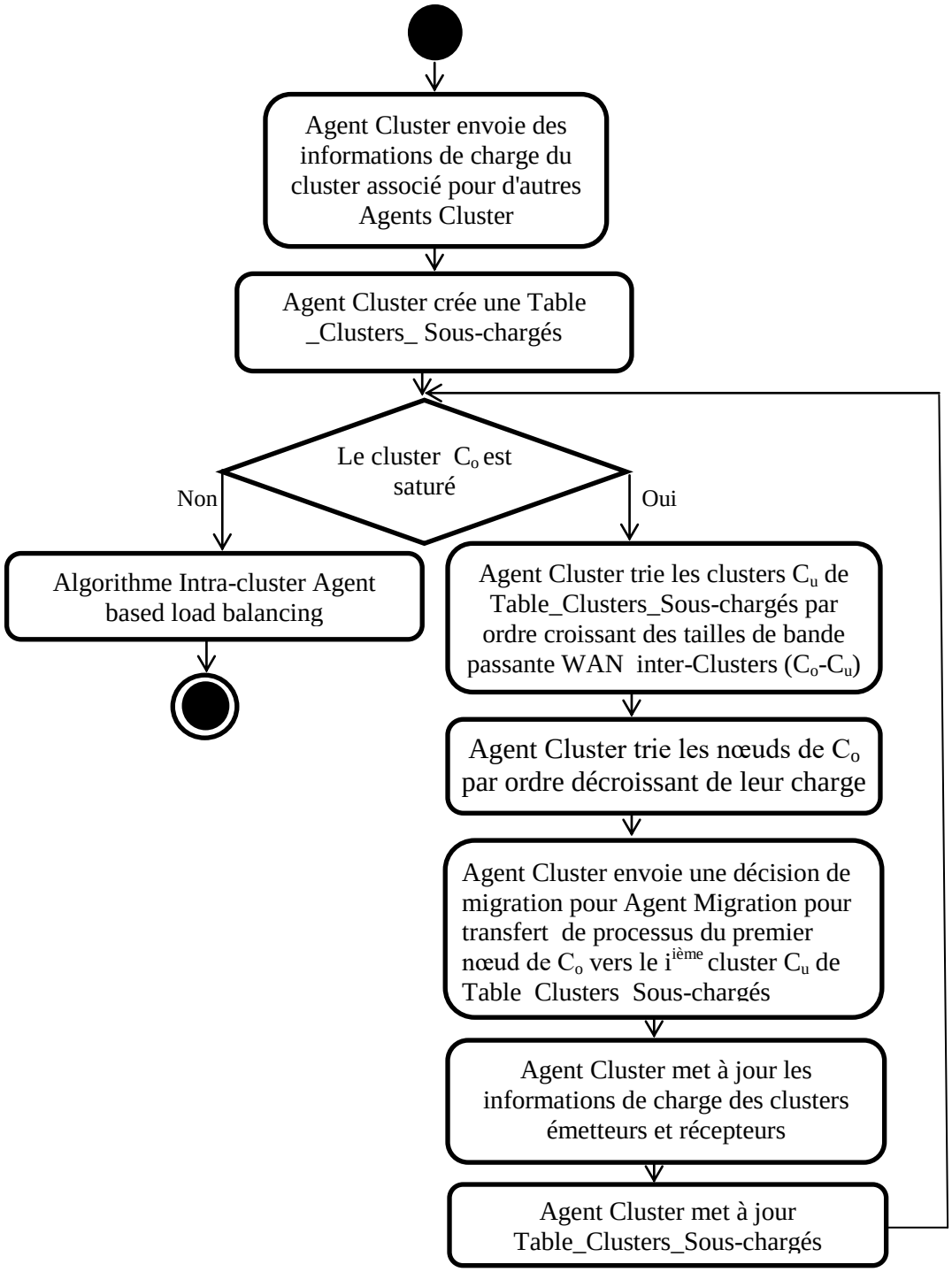


Figure 3.9 : Organigramme pour l'algorithme Inter-cluster Agent based load balancing

3.4 Conclusion

Dans ce chapitre, nous avons présenté en détails notre contribution. Dans la première contribution nous avons proposé un nouvel algorithme de migration de processus pour l'équilibrage de charge dynamique (JMADLB), dans lequel des paramètres tels que la charge du processeur et la longueur de la file d'attente ont été pris en compte et ont été utilisés pour la sélection des ressources surchargées (ou sous-chargées) dans la grille. Ici, les ressources surchargées n'acceptent aucune nouvelle tâche ; mais, les nouvelles tâches sont migrées vers des ressources sous-chargées, même si ce mécanisme conduit à migrer des tâches supplémentaires pour obtenir l'équilibrage de charge. Dans la deuxième contribution, un modèle d'équilibrage de charge basé sur un agent est présenté. Le modèle d'équilibrage de charge basé sur l'agent proposé vise à tirer parti des caractéristiques de l'agent pour générer un système autonome. Il a également un avantage distinct sur les autres modèles d'équilibrage de charge basés sur les agents sur l'utilisation des ressources.

Dans le chapitre suivant, nous entamerons l'implémentation des algorithmes présentés dans ce chapitre et nous accomplirons au lancement de plusieurs simulations avec des explications portant sur les résultats obtenus. Au cours du même chapitre, nous présenterons quelques interfaces ainsi que des résultats graphiques obtenus par notre application. Nous montrons, à partir des résultats obtenus, que notre algorithme proposé améliore les performances globales de la grille de calcul.

Chapitre 4

Implémentation et expérimentations

4.1 Introduction

Après avoir présenté les différentes fonctionnalités de notre système, nous allons passer à l'implémentation du système proposé. Afin de valider et évaluer nos modèles d'équilibrage de charge, nous avons réalisé une série d'expérimentations dont les résultats et les explications font l'objet de ce chapitre.

Tout d'abord, nous commençons par exposition des outils de développement avec lesquels notre système a été réalisé, ensuite, nous expliquons le fonctionnement de la plateforme développée et définissons les métriques et les notations utilisées. Enfin nous discutons et analysons les résultats obtenus lors des différentes simulations.

4.2 Implémentation du modèle d'équilibrage de charge avec migration de processus

Pour tester ou valider les algorithmes utilisés dans des systèmes à large échelle ou dans les grilles de calcul, trois approches sont couramment utilisées : l'analyse mathématique, les expérimentations et la simulation. Ces trois approches permettent la reproductibilité des résultats, chose importante afin de donner la possibilité d'être validés par la communauté des chercheurs.

La première approche est la méthode analytique qui se base sur la modélisation mathématique et l'utilisation des démonstrations pour démontrer qu'une configuration est meilleure par rapport à une autre,. Cependant, avec des systèmes complexes, la méthode analytique devient très difficile à être achevée voir impossible.

La deuxième méthode employée est celle qui est basée sur l'expérimentation. Cette approche consiste à l'implémentation directe de différentes solutions afin de trouver la meilleure d'entre elles. L'expérimentation peut à son tour, devenir contraignante si nous voulons expérimenter des systèmes nécessitant des moyens importants, dont le coût peut être très lourd ou même des systèmes qui font intervenir plusieurs paramètres où les conditions expérimentales ne soient pas les mêmes.

La troisième méthode employant la simulation représente un bon outil. La simulation ne demande pas de grands moyens en permettant d'itérer d'autant que nécessaire les expériences tout en ayant les mêmes conditions expérimentales. La simulation est une imitation d'un système par un autre. En tant que telle, plusieurs aspects de la simulation

sont des simplifications du système réel ou bien ceux qui proviennent des déductions faites à partir d'études sur des systèmes similaires. Bien entendu, ces abstractions peuvent biaiser les résultats observés par la simulation.

Le nombre des paramètres utilisés dans notre système et la problématique étudiée font révéler que l'approche mathématique très difficile dans notre cas. L'acquisition de ressources nécessaires au déploiement de nos algorithmes dans le monde réel semble une tâche impossible. De ce fait, nous avons adopté pour l'utilisation de la simulation, un des simulateurs déjà disponibles et qui sont pour la plupart en Open Source.

Afin d'étudier le comportement de nos algorithmes d'équilibrage de charge, nous les avons intégré dans le simulateur Alea 2 (Job Scheduling Simulator) [77]. Ce simulateur est basé sur le GridSim Toolkit [78] et représente une extension incluant des outils compétents de visualisation de la mise en œuvre d'un algorithme d'ordonnancement et avec une vitesse supérieure des simulations. Alea 2 est un simulateur basé sur des événements utilisés pour tester le fonctionnement des algorithmes d'ordonnancement sous différentes conditions (différents types de ressources et modifications dynamiques des tâches dans l'environnement, etc.).

4.2.1 Paramètres de simulation

Afin d'évaluer la faisabilité et les performances de nos algorithmes, nous les avons testés dans le simulateur Alea 2 par utilisation des paramètres suivants:

- **Paramètres des ressources :** il donne des informations sur les nœuds, les clusters et les réseaux : nombre de clusters, nombre de ressource dans chaque cluster, taille de la mémoire (RAM), vitesse de traitement, les informations des machines étaient en maintenance (panne / redémarrage) et la liste des files d'attente contenant leurs délais et priorités.

- **Paramètres des tâches** : les informations associées à chaque tâche sont : la taille de tâche en MI (Million d'instructions), la durée de tâche (seconde), le temps d'arrivé (UTC timestamp), le temps de début d'exécution (UTC timestamp), le temps de fin (UTC timestamp), le nombre de processeurs utilisés, le nombre des nœuds utilisés et la mémoire utilisée.

- **Paramètres du réseau**: réglage des différentes tailles de bande passante WAN et LAN.

- **Index de charge:** nous avons utilisé trois paramètres, dont la longueur de la file d'attente du processeur, l'utilisation du processeur et l'utilisation de la mémoire de la ressource:
  - La longueur de la file d'attente CPU indique le nombre de tâches en attente dans la file d'attente de la ressource.
  - L'utilisation du processeur est le nombre de CPU utilisés divisé par le nombre total de CPU de ressource.
  - L'utilisation de mémoire est la taille de la mémoire utilisée divisée par la mémoire totale de la ressource.

4.2.2 Métriques utilisées

Nous sommes concentrés sur les métriques suivantes:

- **Temps de réponse moyen:** le temps de réponse est le temps nécessaire pour qu'une tâche passe dans le système, c'est-à-dire le temps entre son arrivée et sa fin. Ensuite, nous avons calculé le temps de réponse moyen pour toutes les ressources, dont T représente le nombre des tâches.

  Temps de réponse moyen=$\sum_{i=1}^{Max\ tâches}(fin_{Temps} - arrivée_{Temps})$/T

- **Temps d'attente moyen:** il indique le temps pris pour que toutes les tâches passent en état d'attente avant de commencer leur exécution.

Temps d'attente moyen = max (Temps de réponse - Temps de traitement) de tous les tâches.

- **Ralentissement :** désigne le rapport entre le temps de réponse et le temps de traitement

  Ralentissement = Max (temps de réponse / temps de traitement) de tous les tâches.

- **Taux moyen d'utilisation des ressources:**

  U=$\sum_{R=1}^{N} U_R$ / où N le nombre de ressources et U_R est le taux d'utilisation d'une ressource.

4.2.3 Environnement expérimental

Pour évaluer l'efficacité de nos algorithmes proposés, nous avons mis en place un environnement expérimental en utilisant l'Alea 2 comme simulateur de grille et le JADE (Java Agent Development Framework) pour l'implémentation d'agents. Dans l'infrastructure proposée, les agents peuvent communiquer au sein d'un environnement de grille à l'aide de la plateforme d'agent Jade.

4.2.3.1 Simulation de l'environnement du grille

Nous avons utilisé le simulateur GridSim pour simuler un environnement de la grille. Le diagramme d'activité d'un exemple de GridSim est donné à la figure 4.1.

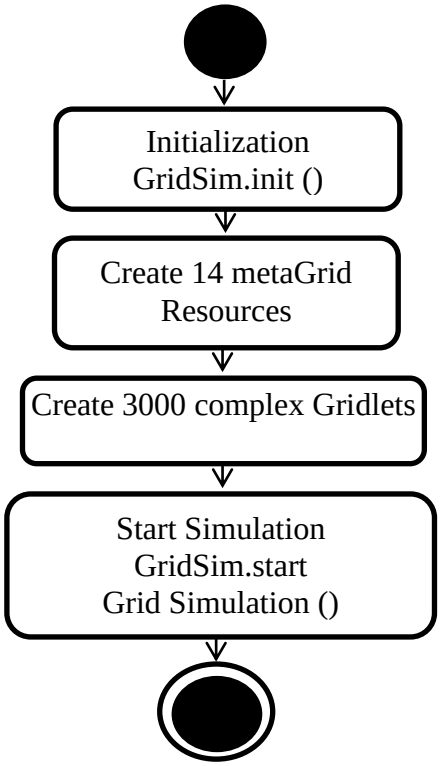


Figure 4.1 : Diagramme d'activité d'un exemple GridSim.

Création d'une ressource de grille

Une ressource de grille simulée dans GridSim contient un ou plusieurs machines. De même, une machine contient un ou plusieurs PE (Processing Elements ou CPU). On présente sur la figure 4.2 le résultat obtenu dans le simulateur GridSim.

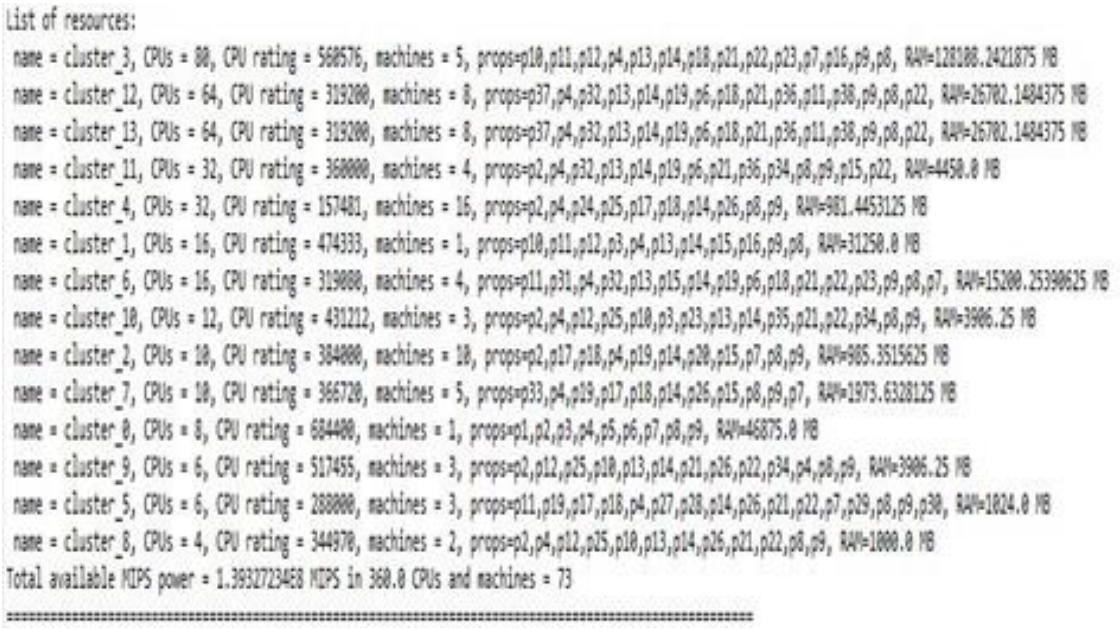
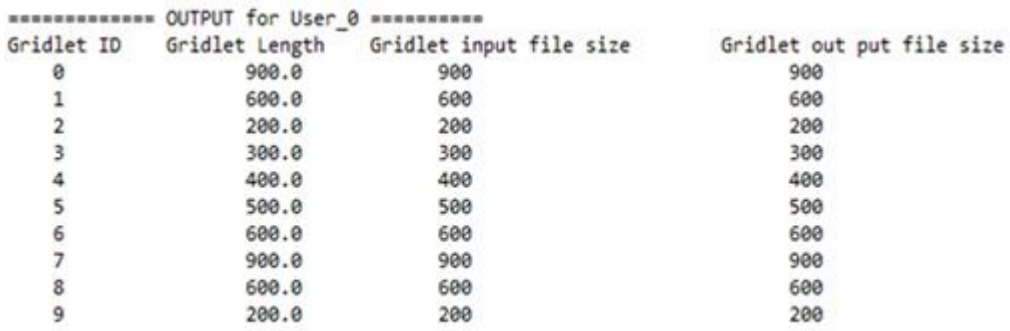


Figure 4.2 : Création de ressources de grille dans le simulateur GridSim.

Création de Gridlets (Tâches)

Dans la terminologie de GridSim, une tâche qui peut s'exécuter séquentiellement et indépendamment sur une ressource de grille est appelée Gridlet. Après la création de ressource, on crée des Gridlets dans le GridSim puis les soumis. Chaque Gridlet a un identificateur unique. L'utilisateur fournit le nombre de Gridlets qui doivent être créés, puis pour chaque Gridlet, nous devons spécifier la longueur de Gridlet ; sa taille de fichier de sortie, sa taille de fichier d'entrée et son identificateur unique pour la simulation. Le résultat obtenu est illustré dans la figure 4.3.



========== OUTPUT for User_0 ==========

Gridlet ID	Gridlet Length	Gridlet input file size	Gridlet out put file size
0	900.0	900	900
1	600.0	600	600
2	200.0	200	200
3	300.0	300	300
4	400.0	400	400
5	500.0	500	500
6	600.0	600	600
7	900.0	900	900
8	600.0	600	600
9	200.0	200	200

Figure 4.3 : Création de Gridlets dans le simulateur GridSim.

Soumission de Gridlets aux ressources de grille

Les étapes à suivre pour soumettre les Gridlets aux ressources sont présentées dans la figure 4.4.

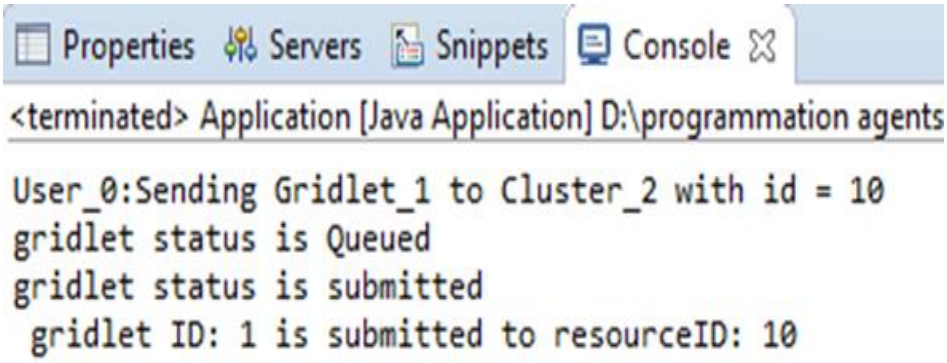


Figure 4.4 : Soumission de Gridlets aux ressources.

Migration du processus dans le simulateur GridSim

Les Gridlets sont déplacés des ressources surchargées vers d'autres ressources sous-chargées. Les étapes de migration sont illustrées dans la figure 4.5.

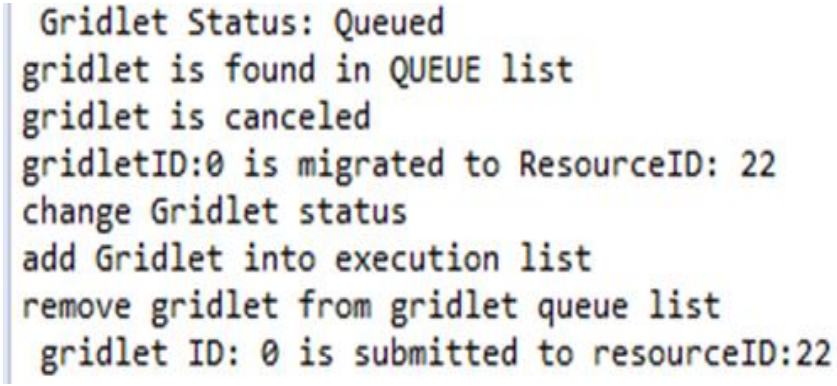


Figure 4.5 : Migration de processus

4.2.3.2 Base de données utilisée

Nous avons modélisé l'ensemble de données complexes de la Grille nationale de la République tchèque "MetaCentrum", ces données nous permettent de mettre en œuvre des simulations très réalistes. En outre, Elles offrent des informations sur les pannes de machines et les exigences spécifiques des tâches. Ces informations influencent la qualité des solutions générées par les algorithmes d'ordonnancement. La description de la tâche comprend : ID du tâche, l'utilisateur, la file d'attente, les processeurs utilisés, etc. La description des clusters comprend également des informations détaillées, notamment la

taille de la RAM, la vitesse du processeur, l'architecture du processeur, le système d'exploitation et la liste des propriétés prises en charge (file d'attente autorisée (s), emplacement du cluster, interface réseau, etc.). De même, les machines qui étaient en maintenance (panne / redémarrage) font aussi partie des détails fournis. Enfin, la liste des files d'attente contenant leurs délais et priorités est fournie. Plus de détails sur le fichier de trace utilisé peuvent être trouvés dans le site web [79].

MetaCentrum est composé de 14 clusters Linux, avec différentes configurations, présentés comme suit:

Tableau 4.1 : L'ensemble de données MetaCentrum: description des clusters

Cluster	Processor	Nodes	CPUs total	CPU speed	Ram
0	Itanium2 1.5GHz	1	8	1500	48000000
1	Opteron2.2GHz	1	16	2200	32000000
2	Xeon 3.2GHz	10	10	3200	1009000
3	Opteron 2.6GHz	5	80	2600	131182840
4	AthlonMP 1.6GHz	16	32	1600	1005000
5	Xeon 2.4GHz	3	64	2400	1048576
6	Xeon 2.7GHz	4	14	2659	15565060
7	Xeon 3.1GHz	5	70	3056	2021000
8	Opteron 1.6GHz	2	20	1600	1024000
9	Opteron 2.4GHz	3	6	2400	4000000
10	Opteron 2.0GHz	3	92	2000	4556800
11	Xeon 3.0 GHz	4	15	3000	27343000
12	Xeon 2.7 GHz	8	64	2660	37343000
13	Xeon 2.3 GHz	1	44	2330	10240000

Tableau 4.2 : Exemple de l'ensemble de données MetaCentrum: description des tâches

Job ID	User	Queue	Number of processors used	Memory used	Arrival time	Duration
0	user_33	q2	1	1009884	1228910204	2592042
1	user_33	q2	1	1010132	1228910204	2592046
2	user_33	q2	1	1010108	1228910223	1893368
3	user_44	q1	2	4940	1229426624	2592030
4	user_43	q1	1	14088	1230194569	2012962
5	user_43	q1	1	13556	1230194569	2290213
6	user_43	q1	1	14124	1230194570	2058925
7	user_8	q1	1	351604	1230387996	1573008
8	user_8	q1	1	351028	1230390265	1572997
9	user_1	q6	8	2989520	1230675195	92810
10	user_0	q3	4	6992	1230716510	84658
11	user_0	q3	4	6972	1230716631	85018
12	user_0	q3	4	5284	1230716750	84720

Tableau 4.3 : Exemple de l'ensemble de données MetaCentrum: description des files d'attente

Queue	Priority	Time limit(Hr)
q1	62	720
q2	70	720
q3	50	24
q4	60	2
q5	80	24
q6	65	720
q7	70	720
q8	70	4
q9	70	720
q10	99	720
q11	65	720

4.2.4 Expérimentations et interprétation des résultats

Pour obtenir des résultats qui soient les plus fiables possibles, nous avons réitéré les mêmes expériences plus de dix fois. L'ensemble de ces expériences a été réalisé sur un PC Intel I3 Duo de 2.00 GHz doté d'une mémoire 4GB de RAM fonctionnant sous un système d'exploitation de Windows 2010.

L'algorithme proposé fournit un meilleur mécanisme de migration de processus avec équilibrage de charge dynamique. Les facteurs de performance importants dans l'estimation de notre algorithme proposé sont la diminution du temps de réponse, la réduction du nombre de travaux en attente dans la file d'attente et la maximisation de l'utilisation des ressources. Nous avons effectué quelques expérimentations pour évaluer l'efficacité et les performances de l'algorithme proposé.

Expérimentations 1

Dans la première expérimentation, nous avons concentré sur le temps de réponse moyen (en seconde), le temps d'attente moyen (en seconde) selon différents nombres de tâches et de clusters. Nous avons supposé différents nombres de clusters en considérant que chaque cluster est composé de différents nombres de ressources. Les tableaux 4.4 et 4.5 montrent les résultats de la comparaison entre le temps de réponse moyen et le temps d'attente moyen de l'algorithme proposé (JMADLB) en comparaison avec l'algorithme FCFS (first come, first served) et l'EDF (Earliest deadline first). Les résultats ont montré que l'algorithme proposé surpassait les autres algorithmes en diminuant le temps de réponse moyen et en réduisant le temps moyen d'attente.

Dans l'algorithme FCFS, si la ressource demandée par la première tâche de la file d'attente n'est pas disponible, les tâches restantes ne peuvent pas être planifiées même si les ressources demandées sont disponibles. Dans l'algorithme proposé, il fonctionne comme le FCFS lors de la sélection des tâches, mais lorsque la première tâche dans la file d'attente ne peut pas être planifiée directement, l'algorithme proposé estime la première heure de démarrage probable pour la première tâche en utilisant le temps de traitement calculé à partir des tâches en cours d'exécution. Ensuite, il fait une réservation pour exécuter la tâche à cette heure pré-estimée. Par la suite, il examine la file d'attente des tâches en attente et planifie directement chaque tâche n'intervenant pas avec la réservation de la première tâche. Cela permet de réduire le temps d'attente moyen des tâches, comme indiqué dans le tableau 4.5.

D'après les figures 4.6, 4.7 et 4.8, nous pouvons percevoir que l'algorithme proposé fonctionne beaucoup mieux que le FCFS et l'EDF, car les tâches sont également répartis sur les clusters disponibles.

Tableau 4.4 : Temps de réponse moyen (en seconde) par rapport à différents nombres de Taches et de clusters

Jobs \ Clusters		14	20	30	40
100	FCFS	717269.16	717271.54	717276.13	717280.43
	EDF	736509.73	736512.20	736516.98	736521.45
	JMADLB	637271.36	700912.49	698531.38	698080.46
500	FCFS	530716.63	530718.29	530721.74	530724.96
	EDF	591561.10	591743.39	591747.36	591751.077
	JMADLB	440809.53	434139	445092.81	449563.46
1000	FCFS	438615.00	434460.11	434461.78	434463.34
	EDF	537610.93	536181.01	535057.38	535059.01
	JMADLB	331294.50	378577.93	373628.36	353522.78
1500	FCFS	399480.80	410087.71	409660.10	409661.21
	EDF	567050.98	562347.11	561211.74	561213.04
	JMADLB	364341.22	362851.65	364643.87	346922.03
2000	FCFS	369632.99	375405.07	373870.80	373782.58
	EDF	552022.05	552652.35	551188.48	551189.61
	JMADLB	330353.53	323158.54	325187.02	319165.92
2500	FCFS	399492.28	382659.48	392318.16	392301.85
	EDF	578191.08	553857.99	559943.26	559944.21
	JMADLB	347443.27	328675.99	343730.72	348397.71
3000	FCFS	474139.31	452357.27	466811.40	463275.92
	EDF	639816.67	627373.21	639639.79	639640.58
	JMADLB	415538.12	405063.64	410469.78	404221.75

Le temps de réponse moyen, le temps d'attente moyen et le ralentissement moyen sont légèrement meilleurs pour l'algorithme proposé. D'un autre côté, dès que des exigences de tâches spécifiques sont prises en compte, l'algorithme proposé rencontre quelques difficultés pour certaines expérimentations, car le modèle des exigences de tâche spécifiques mappé dans Alea 2 a construit une liste de clusters où les tâches ayant le même identifiant d'application ont été exécutés. Ensuite, pendant l'exécution de l'identifiant d'application détecté pour chaque tâche ainsi que le cluster correspondant dans la liste sont sélectionnés pour être les seuls sites d'exécution appropriés pour la tâche.

Tableau 4.5 : Temps d'attente moyen (en secondes) par rapport à divers nombres de Taches et de clusters

Clusters / Jobs		20	30	40
100	FCFS	22306.28	22309.45	22312.41
	EDF	58405.46	58409.96	58414.18
	JMADLB	19258.45	19664.86	19372.73
500	FCFS	137967.61	137969.71	137971.68
	EDF	302655.43	302659.11	302662.55
	JMADLB	78715.0	84158.6	82029.63
1000	FCFS	162299.14	162299.97	162300.75
	EDF	323706.82	322631.32	322632.13
	JMADLB	81357.24	80710.37	81642.68
1500	FCFS	128063.37	128023.5	128024.02
	EDF	346174.1	344616.87	344617.41
	JMADLB	58285.76	57226.62	56832.25
2000	FCFS	107723.99	106842.64	106842.72
	EDF	356689.09	355514.91	355515.32
	JMADLB	45392.07	42603.41	46035.96
2500	FCFS	93796.16	93869.37	93863.96
	EDF	344555.89	339015.8	339016.15
	JMADLB	34909.71	35336.33	34526.03

Le problème ici est que tous les algorithmes exécutés à l'aide de l'ensemble de données MetaCentrum permettront l'exécution de tâches sur certains clusters si et seulement si les clusters sont d'accord avec toutes les exigences spécifiques des tâches. Dans le cas où le cluster approprié est saturé, la tâche correspondante attend que le cluster approprié soit équilibré. Dans l'algorithme proposé, nous avons essayé de résoudre ce problème en empêchant les ressources surchargées et nous ne permettons pas de recevoir plus de tâches. De toute évidence, une solution simple ne suffit pas pour des problèmes plus complexes.

La comparaison du temps de réponse moyen, du temps d'attente moyen et du ralentissement moyen entre le FCFS, l'EDF et le JMADLB avec 20 clusters sont présentés dans les figures 4.6, 4.7 et 4.8.

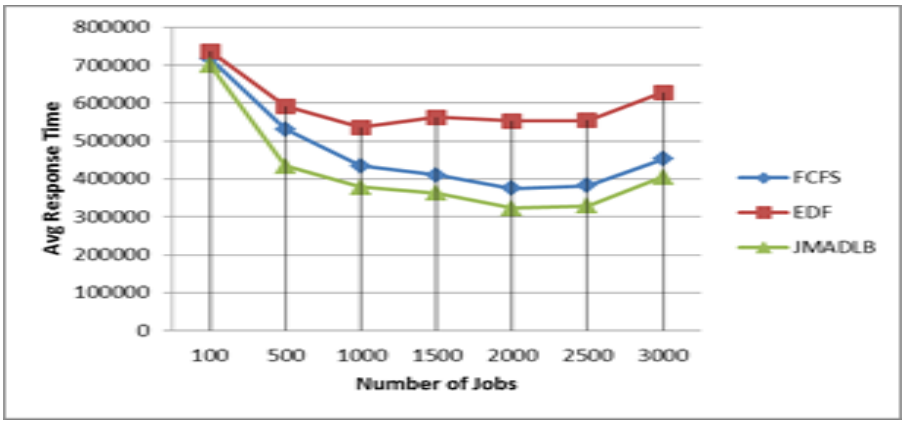


Figure 4.6: Comparaison du temps de réponse moyen (en seconde) entre le FCFS, l'EDF et le JMADLB avec 20 clusters.

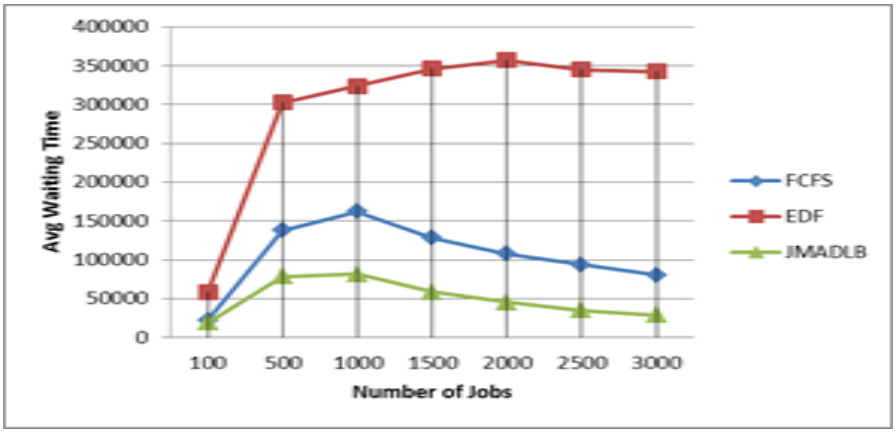


Figure 4.7: Comparaison du temps d'attente moyen (en seconde) entre le FCFS, l'EDF et le JMADLB avec 20 clusters

Figure 4.8: Comparaison du ralentissement moyen (en seconde) entre le FCFS, l'EDF et le JMADLB avec 20 clusters

Expérimentations 2

Dans la deuxième expérimentation, nous avons concentré sur l'utilisation des ressources (%). Nous avons supposé que le nombre de clusters est 14 et nous avons considéré que chaque cluster est composé de plusieurs nombres de ressources. Par conséquent, le nombre de tâches est de 3000. La figure 4.9 montre l'utilisation des ressources pour les différents algorithmes.

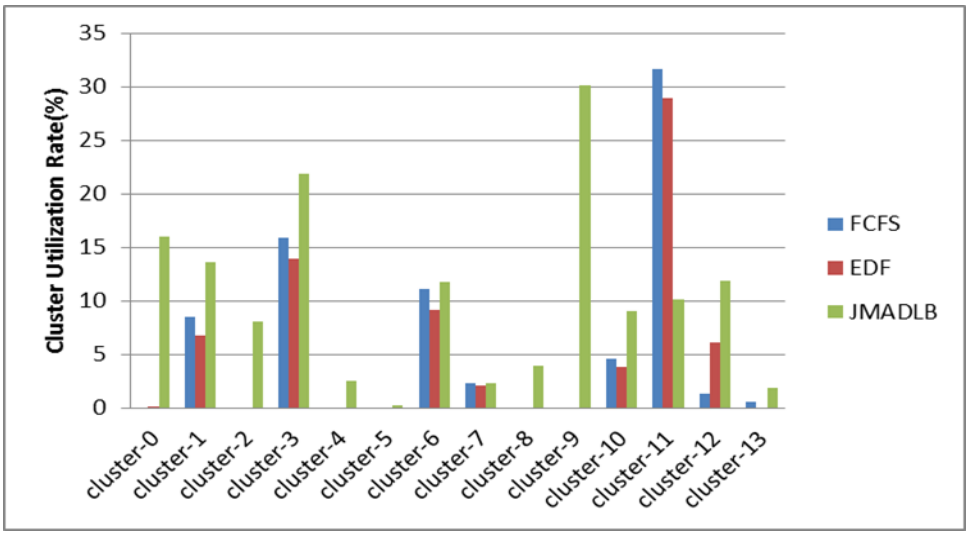


Figure 4.9: Comparaison de l'utilisation des clusters (%) entre le FCFS, l'EDF et le JMADLB avec 14 clusters.

L'examen de l'utilisation des clusters a montré que le FCFS et l'EDF ne fonctionnaient pas bien. Il montre que le cluster-11 qui possède les ressources de traitement les plus élevées est donc sur-utilisé, en tenant compte que les clusters 2, 4, 5, 8 et 9 sont inactifs dans les algorithmes d'ordonnancement FCFS et EDF. La cause de l'amélioration est bien l'utilisation de tous les clusters, ce qui est obtenu en raison de l'équilibrage de charge par le JMADLB entre les clusters au moment de l'ordonnancement. Il montre aussi que le JMADLB est mieux exécuté par rapport aux algorithmes d'ordonnancement traditionnels, pareillement parce que l'équilibrage de charge est utilisé pour s'assurer qu'aucun des clusters existants n'est plus inactif pendant que d'autres sont en train d'être utilisés. Les effets d'équilibrage de charge sont causés par des clusters sous-chargés. Dans l'algorithme proposé, il y a une augmentation d'utilisation de 0% du cluster 2 (avant l'algorithme JMADLB) à 8% (après) et de 0% du cluster 4 à 2,5%. Alors que pour le cluster 8 et 9 elle est estimé de 0% à 4% et du 0 % à 30%, respectivement. Dans ce cas là, les différentes utilisations des clusters participants sont équilibrées. En revanche, les tâches ayant des

besoins spécifiques doivent attendre jusqu'à que les ressources appropriées soient disponibles. Cela génère en fait une utilisation plus élevée du système sur des clusters particuliers. Afin de faire équilibrer la charge, nous avons essayé de résoudre ce problème en faisant migrer les tâches pour les ressources inactives et en empêchant les ressources surchargés. De plus, l'algorithme JMADLB permet la distribution des tâches sur les ressources les plus disponibles lorsqu'il n'y avait pas de ressource appropriée. Ce comportement représente une manière contraire du fonctionnement des autres algorithmes d'ordonnancement qui essaient de sélectionner la meilleure ressource ressemblant aux exigences de la tâche; sinon, la tâche restera dans la file d'attente globale, ce qui indique une sous-utilisation des ressources.

Expérimentations 3

Dans la troisième expérimentation, nous avons concentré sur l'utilisation du processeur en le comparant avec les processeurs demandés. Nous avons supposé que le nombre de clusters est de 20 et en considérant que chaque cluster est composé de différents nombres de ressources (figures 4.10, 4.11 et 4.12).

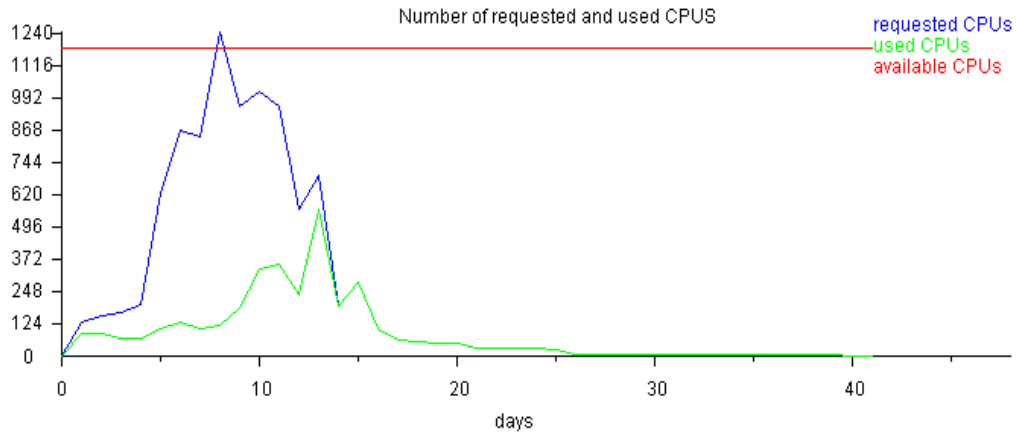


Figure 4.10: Comparaison entre les CPUS demandés et les CPUS utilisés par utilisation de l'algorithme JMADLB avec 20 clusters.

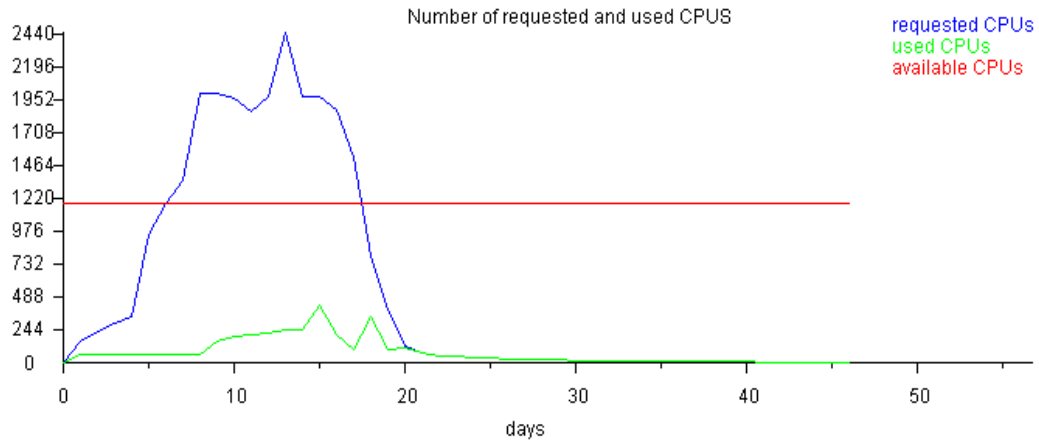


Figure 4.11: Comparaison entre les CPUS demandés et les CPUS utilisés en utilisant l'algorithme EDF avec 20 clusters

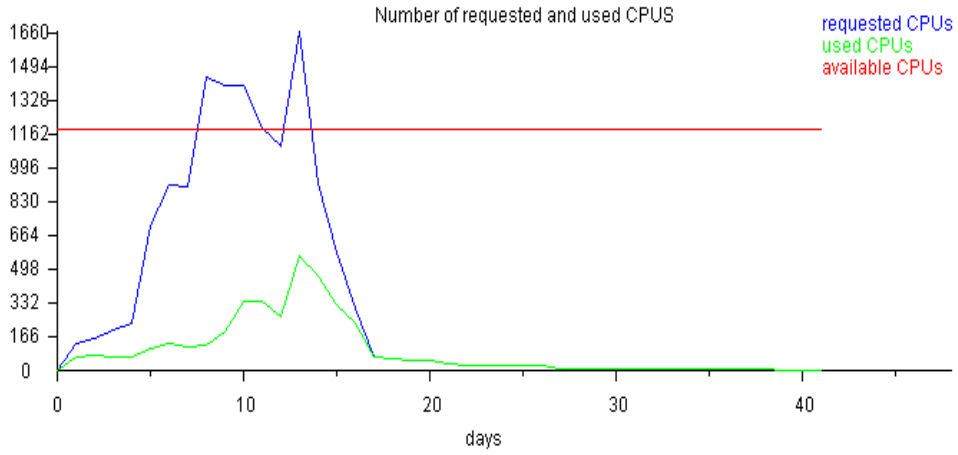


Figure 4.12: Comparaison entre les CPUS demandés et les CPUS utilisés en appliquant l'algorithme FCFS avec 20 clusters

À partir des figures ci-dessus, la courbe bleue indique le nombre total des CPU demandées, tandis que la courbe verte représente le nombre des CPU utilisées. Le haut de la courbe bleue indique qu'à ce moment particulier, les demandes pour le CPU étaient trop élevées. Ainsi, la situation favorable sera celle où le nombre de sommets bleus sera moindre. La ligne rouge est le nombre des CPU disponibles pour l'exécution. Avec 3000 Tâches en FCFS, le nombre des CPU demandés est passé au dessus de 1660 CPUs, alors qu'en EDF c'est 2440 CPUs. Nous voyons que dans l'algorithme proposé, le JMADLB a environ 1240 CPUs. Il est évident que dans le FCFS et dans l'EDF, certaines des tâches sont mises en état d'attente car il y a une augmentation des CPUs demandées par rapport aux CPUs en cours d'exécution. Il est évident que l'algorithme proposé supprime cette probabilité, mais certaines tâches doivent également attendre dans l'algorithme JMADLB,

néanmoins le temps d'attente est réduit par rapport aux deux autres algorithmes, ce qui est l'un des avantages de l'algorithme JMADLB proposé.

Expérimentations 4

Dans la quatrième expérimentation, nous avons concentré sur les tâches en attente dans la file d'attente et nous les avons comparés aux tâches en cours d'exécution. Nous avons supposé que le nombre de clusters est de 20 et en considérant que chaque cluster est composé de plusieurs nombres de ressources. Le nombre des tâches est de 3000 (figures 4.13, 4.14 et 4.15).

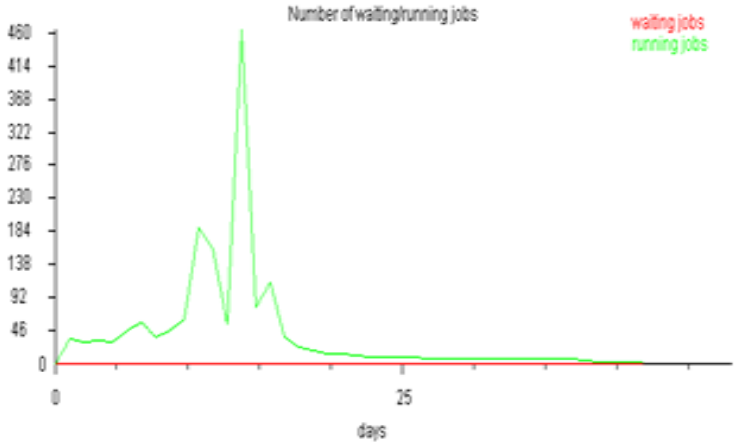


Figure 4.13 : Comparaison entre les tâches en cours d'exécution et les tâches en attente par utilisantion de l'algorithme JMADLB avec 20 clusters.

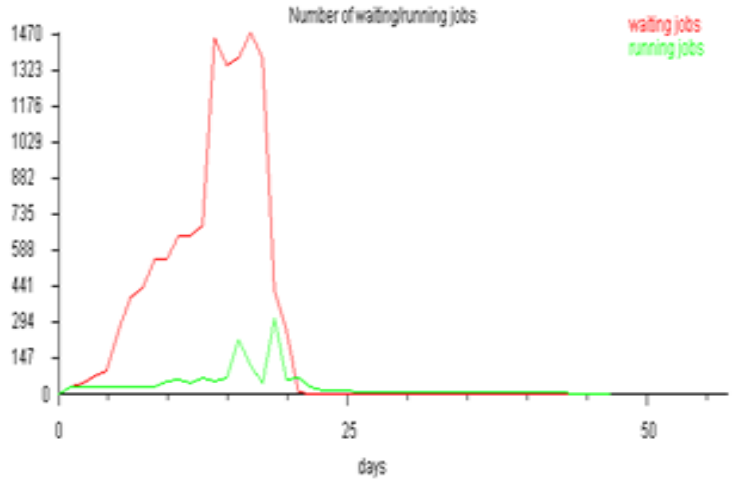


Figure 4.14: Comparaison entre les tâches en cours d'exécution et les tâches en attente en utilisant l'algorithme EDF avec 20 clusters.

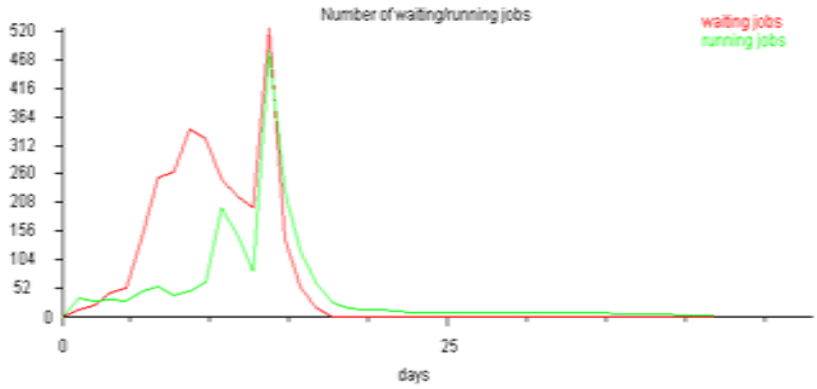


Figure 4.15: Comparaison entre les tâches en cours d'exécution et les tâches en attente en employant l'algorithme FCFS avec 20 clusters.

L'axe horizontal représente le temps (unités de jours) tandis que l'axe vertical indique le nombre de tâches. La courbe rouge montre les tâches en attente qui montre qu'une tâche dans la file d'attente est mise parmi les tâches en attente, alors que la courbe verte montre les tâches en cours d'exécution qui dit qu'une tâche est en cours d'exécution. L'EDF et le FCFS ne sont pas en mesure d'ordonnancer facilement les tâches, ce qui génère les plus grands tâches en attente durant le temps. Pour 3000 tâches et avec 20 clusters, l'algorithme JMADLB proposé est capable d'utiliser plus élevée des ressources et il n'y a pas de tâches en attente dans la file d'attente dans le temps comme le montre la figure 4.13.

4.2.5 Synthèse des expérimentations

Dans cette section, nous avons implémenté l'algorithme proposé sous le simulateur Alea 2. Les résultats expérimentaux sont encourageants car nous pouvons réduire considérablement le temps de réponse moyenne, le temps d'attente, le ralentissement en maximisant d'une autre part l'utilisation des ressources. À la section suivante, nous voulons implémenter l'algorithme proposé en ajoutant les systèmes multi-agents. Nous définissons également d'autres paramètres de performance pour évaluer et comparer notre algorithme avec d'autres algorithmes existants.

4.3 Implémentation du modèle d'équilibrage de charge basé sur des agents

Pour évaluer l'efficacité de nos algorithmes proposés, nous avons mis en place un environnement expérimental en utilisant l'Alea 2 comme simulateur de grille et le JADE (Java Agent Development Framework) pour l'implémentation des agents. Dans

l'infrastructure proposée, les agents peuvent communiquer au sein d'un environnement de grille à l'aide de la plateforme d'agent Jade.

4.3.1 Simulation de l'environnement de la grille

Nous avons utilisé l'Alea 2 pour simuler un environnement de la grille. L'AgentGrille lit le fichier de l'ensemble des données contenant les descriptions des clusters et crée les ressources de la grille en conséquence par le GridSim standard. Il crée des ressources de grille avec différentes compétences / vitesses de traitement similaires à celles de l'environnement réel à différentes politiques (ressources partagées dans l'espace ou dans le temps) et différents fuseaux horaires. Il crée également des utilisateurs avec des exigences différentes. L'AgentGrid lit également les données décrivant les tâches à partir d'un fichier de données et crée dynamiquement les instances des jobs tout le long du temps.

4.3.2 L'architecture de contrôle: bibliothèque d'équilibrage de charge basée sur l'agent

Nous avons développé une bibliothèque de classes qui simule les activités d'une plateforme d'agent. Une telle bibliothèque, appelée équilibrage de charge basé sur l'agent, comprend les classes: AgentGrid, AgentCluster, AgentLBC, AgentMigration et AgentNode et nous les avons intégré avec le simulateur Alea 2 comme montre la figure 4.16.

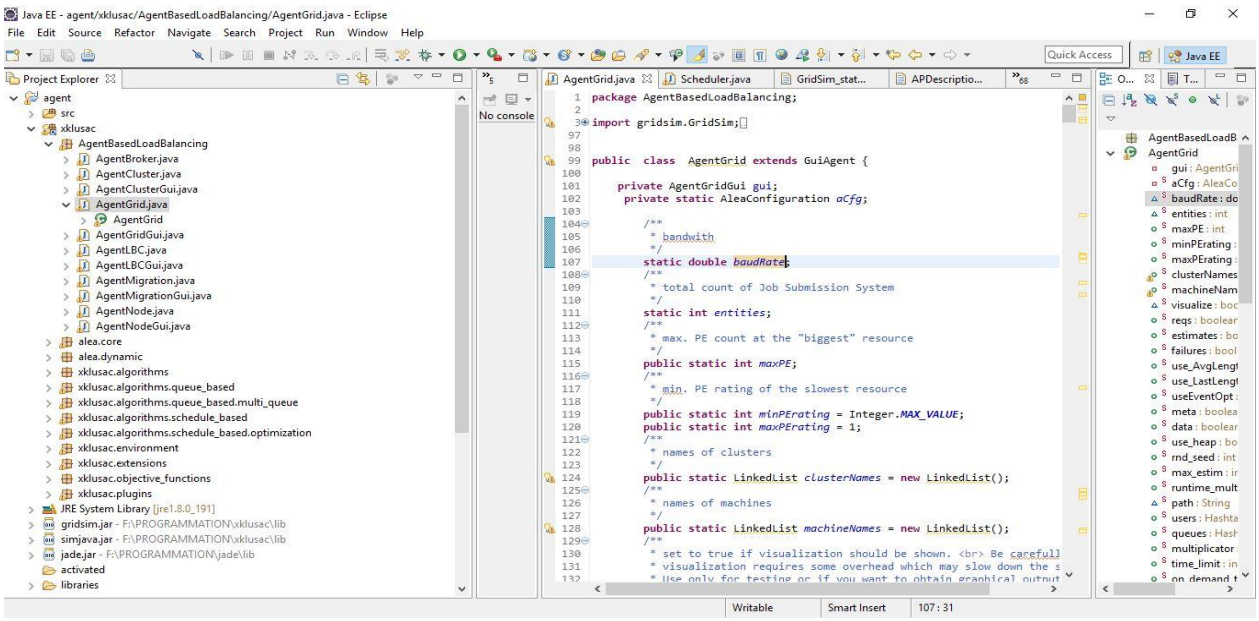


Figure 4.16: bibliothèque d'équilibrage de charge basée sur l'agent.

AgentGrid

Cet agent utilise un seul One Shot Behaviour, car il ne s'exécute qu'une seule fois. Dans ce comportement l'agent charge d'abord les fichiers de configuration et crée l'environnement de grille. Ensuite, il démarre les Agents Cluster et les Agents Migration, en appelant: createNewAgent ("cluster _" + i, "agents.AgentCluster", nouvel objet [] {}); et createNewAgent ("AgentMigration _" + i, "agents.AgentMigration", nouvel objet [] {}); la figure 4.17 montre l'interface de l'AgentGrid.

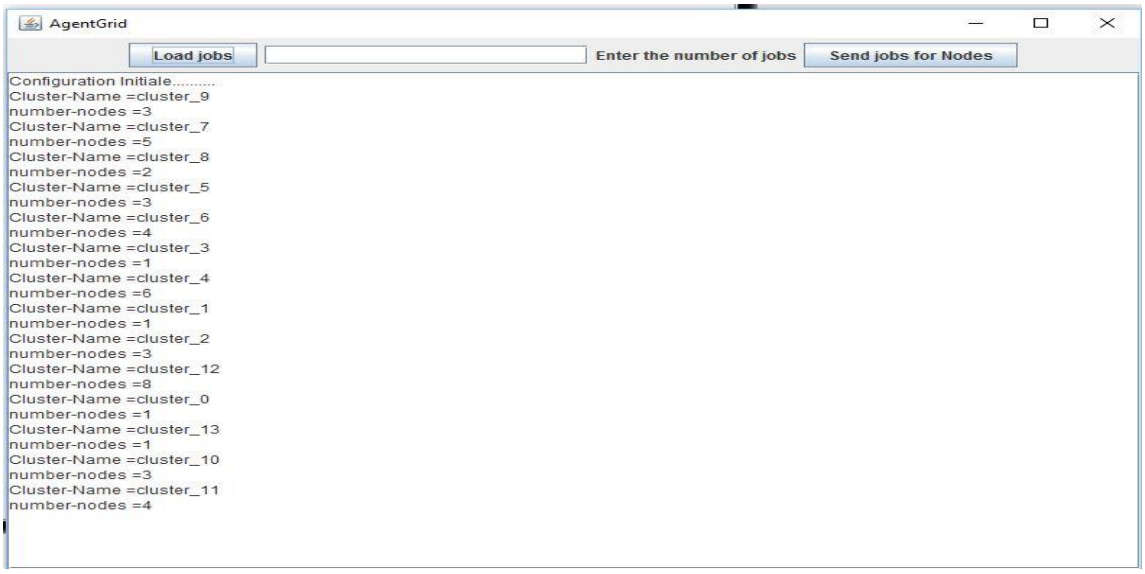


Figure 4.17: Interface de l'AgentGrid

AgentCluster

Cet agent utilise un Cyclic Behaviour, car il recherche constamment les messages entrants en appelant receive(). si (msg.getContent ().StartsWith("numNodes"), l'agent récupère le nombre de nœuds et il démarre les agents nœuds et les Agents LBC en appelant: createNewAgent ("AgentNode" + j + this.getAID (). getName (), "agents.AgentNode", nouvel objet [] { }); createNewAgent ("AgentLBC" + j + this.getAID (). getName (), "agents.AgentLBC", nouvel objet [] {}).

Si (msg.getSender () == AgentNode) et si (msg.getContent (). StartsWith ("LoadCPU"), cet agent récupère les informations de charge du nœud concernant la charge du processeur. Aussi si (msg.getContent (). StartsWith ("QueueLength"), l'agent cluster récupère les informations de charge concernant la longueur de la file d'attente de chaque nœud sous son contrôle. Ensuite l'agent crée un objet NodeInfo qui contient des informations à jour concernant les informations reçues des nœuds sous son contrôle. Une

fois la tâche terminée, l'objet NodeInfo est mis à jour en raison du nouvel état. Ici l'AgentCluster classe les nœuds sous son contrôle en fonction de leurs paramètres de charge (longueur de la file d'attente et la charge de processeur) et détermine les nœuds émetteur et récepteur comme illustré dans la figure 4.18. Ensuite, l'AgentCluster envoie un message de demande pour l'AgentMigration afin de démarrer le processus de migration. Le cycle se termine lorsqu'aucune des nouvelles tâches n'est arrivée et toutes les tâches soumises sont terminées et la charge est équilibrée sur les nœuds sous son contrôle. Enfin, la simulation se termine et l'agent appelle le doDelete () pour se tuer.

Figure 4.18: L'interface de l'AgentCluster

AgentMigration

Cet agent utilise un Cyclic Behaviour, car il recherche constamment les messages entrants l'AgentCluster et les AgentsNode. Lorsque l'AgentMigration reçoit le message de demande d'AgentCluster, il récupère l'ID de l'expéditeur et l'ID du récepteur et appelle Gridsim.gridletMove (gridletId, userId, senderId, receiverId, delay, ack). Si l'opération de migration réussit l'AgentMigration reçoit un accusé de réception de l'AgentNode du nœud récepteur. Dans ce cas l'AgentMigration envoie un message de confirmation pour l'AgentCluster associé, comme illustré dans la figure 4.19.

AgentNode

Cet agent utilise un Cyclic Behaviour, car il recherche constamment les messages entrants de ses AgentCluster, AgentMigration et AgentLBC associés. Si(msg.getSender () == AgentGrid) et si (msg.getContent(). StartsWith ("numJobs"), l'agent récupère le nombre de tâches qui lui sont attribués à partir de l'AgentGrid. Si(msg.getSender () = = AgentLBC), si (msg.getContent (). StartWith ("LoadCPU"), cet agent récupère la charge du processeur et envoie un message d'information pour son AgentCluster associé contenant les informations de charge (la charge de CPU). Si (msg.getContent ().StartWith ("QueueLength"), l'agent récupère la longueur de la file d'attente et envoie un message d'information pour son AgentCluster associé. Si (msg.getSender () == AgentMigration), s'il y a un message de type demande, cet agent exécute le processus de migration et envoie un message de confirmation pour l'AgentMigration associé.

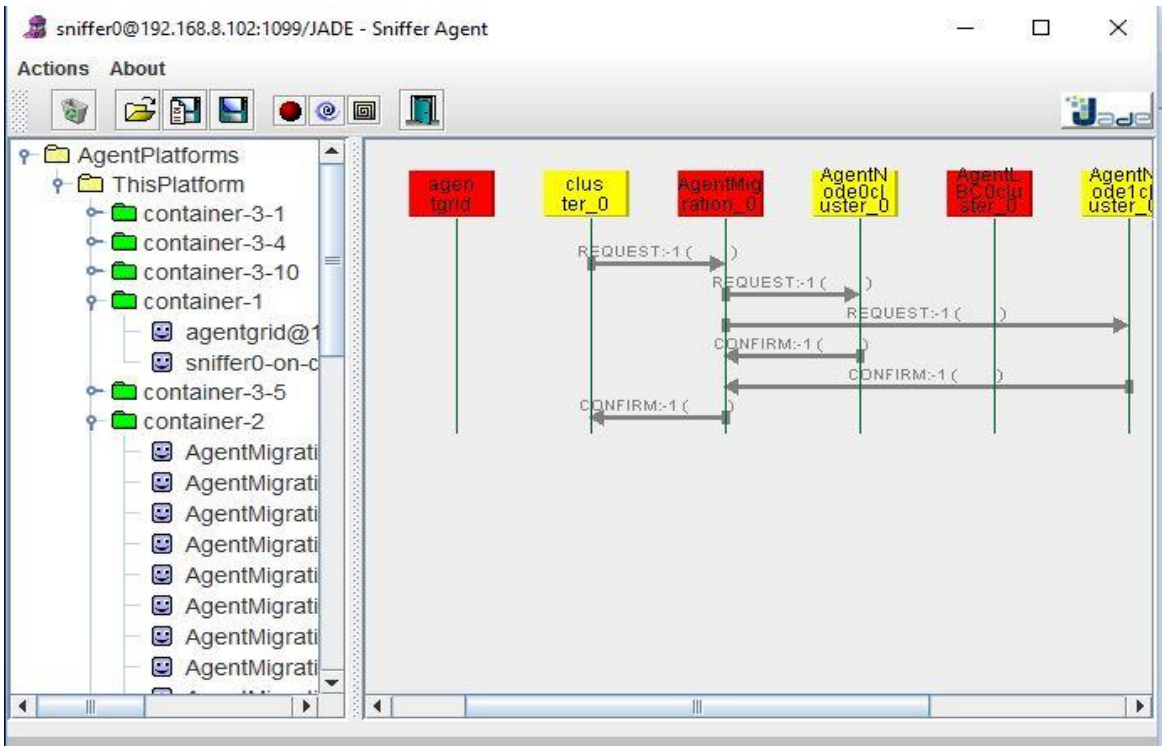


Figure 4.19 : Interactions des agents pendant le processus de migration.

AgentLBC

Cet agent utilise un Cyclic Behaviour, il prend la tâche entrante et la place dans la file d'attente selon l'algorithme d'ordonnancement FCFS en appelant la queue.addLast (LinkedList <GridletInfo> ()); après cela, l'agent collecte diverses données telles que le

nombre de tâches en attente et en cours d'exécution, l'heure de début, l'heure de fin s'il y a des nouveaux événements, l' AgentLBC exécute le code suivant:

// Attendre l'événement et le type d'identité de l'événement reçu

{// si une nouvelle tâche est arrivé}

```
 if (ev.get_tag () == AleaSimTags.GRIDLET_INFO)
{
ComplexGridlet gl = (ComplexGridlet) ev.get_data ();
GridletInfo gi = new GridletInfo (gl); // crée un objet GridletInfo
gi.setDue_date (gl.getDue_date () + GridSim.clock ());
last_job_id = gi.getID (); setLengthStatistics (gi);
}
```

4.3.3 Métriques utilisées

Dans l'objectif d'évaluer nos algorithmes et d'étudier leurs performances, nous avons proposé les métriques suivantes : le passage à l'échelle et l'utilisation de ressources.

4.3.4 Expérimentations et résultats

Afin d'évaluer le comportement du modèle proposé et valider les résultats obtenus, nous avons effectué une série d'expérimentations en utilisant le simulateur Alea 2 qui constitue un bon environnement pour le test des algorithmes d'équilibrage de charge dans les grilles de calcul.

Expérimentation 1 : Évaluation de L'évolutivité du système

Pour évaluer le passage à l'échelle du modèle proposé et pour comparer leurs performances, certaines expérimentations peuvent être utilisées. L'évolutivité du système peut être consultée comme le rapport entre les performances et les ressources. À mesure que les ressources disponibles augmentent, les performances devraient être augmentées.

Dans la première expérimentation, nous concentrons sur le nombre d'agents que le système peut stabiliser avant qu'il ne devienne incontrôlable.

La figure 4.20 montre que la limite de l'algorithme proposé est d'environ 100 agents cluster. La simulation ne prend que quelques minutes, mais elle crée un énorme message. Cela est simplement dû au nombre d'agents participant au système. Ce problème augmente à mesure que de nouveaux agents sont ajoutés, mais autour de 1 Agent Grille, 100 Agents

Cluster, 100 Agents Migration, 462 Agents Nœud, 462 Agents LBC, le système fonctionne toujours correctement. Le modèle hiérarchique d'équilibrage de charge résout les problèmes d'évolutivité: un plus grand nombre d'agents dans l'hiérarchie entraînera une plus grande activité sans l'intervention d'un agent central. Dans ce modèle, le système s'adapte correctement.

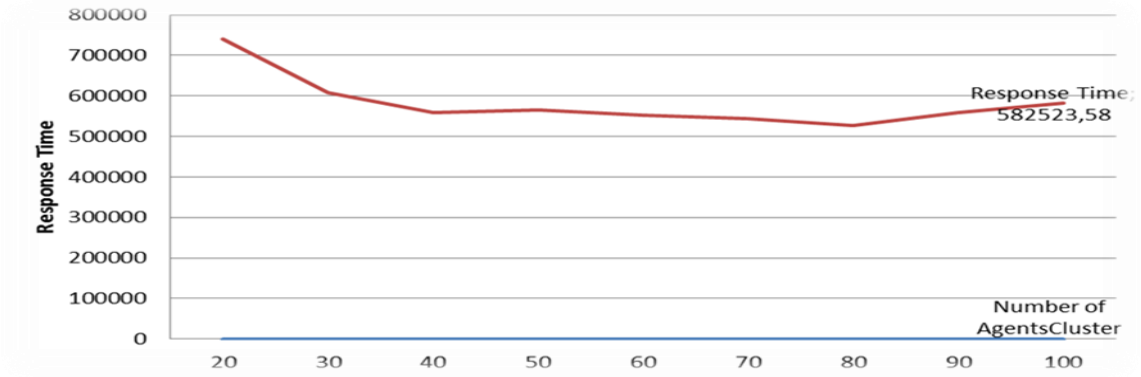


Figure 4.20: Temps de réponse moyen (en seconde) en fonction des différents nombre des agents Cluster.

Expérimentation 2 : L'utilisation des ressources

Dans la deuxième expérimentation, l'utilisation des ressources (%) était le principal objectif. Le nombre de clusters était supposé être de 14 et chaque cluster était composé de différents nombres de ressources. Le nombre de tâches était de 3000.

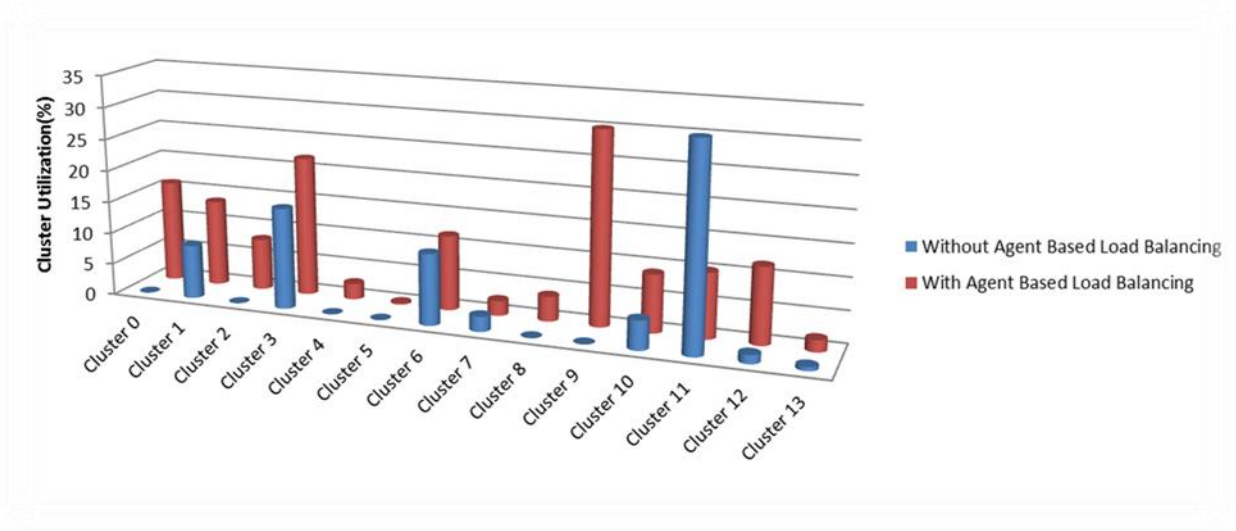


Figure 4.21: Comparaison de l'utilisation des clusters (%) avec et sans l'algorithme d'équilibrage de charge basé sur l'agent à l'aide de 14 clusters.

La figure 4.21 montre l'utilisation du cluster avec et sans l'algorithme d'équilibrage de charge basé sur l'agent proposé. Elle montre que l'algorithme proposé est plus efficace pour maximiser l'utilisation des ressources.

L'algorithme proposé permet de répartir les tâches sur les ressources les plus disponibles lorsqu'il n'y avait pas de ressources appropriées, contrairement à d'autres algorithmes traditionnels qui essaient de sélectionner la meilleure ressource qui ressemble aux exigences de la tâche. Sinon, la tâche restera dans la file d'attente globale, indiquant une sous-utilisation de ces ressources.

4.3.5 Comparaison du modèle proposé avec des travaux connexes

Un certain nombre d'efforts ont été déployés pour développer des modèles d'équilibrage de charge basés sur des agents pour les grilles de calcul. Il est souvent difficile de faire des comparaisons entre différents efforts car chaque modèle d'équilibrage de charge basé sur un agent est généralement développé pour un environnement d'un système particulier ou une application particulièrement gourmande avec différentes suppositions et restrictions.

Le tableau 4.6 présente une comparaison théorique entre le modèle proposé (ABLBM) avec quelques modèles d'équilibrage de charge basés sur l'agent dans les grilles de calcul cités dans le chapitre 2. Chaque ligne présente le nom de modèle, la contribution, son amélioration des mesures de performance, ses modèles comparés, les logiciels utilisés, les travaux futurs et bien ses inconvénients.

Dans le tableau 4.6, nous avons comparé certains modèles actuels d'équilibrage de charge basés sur les agents pour les environnements de grille, car tous les modèles existants que nous avons considérés, ont été proposés pour les environnements de grille de calcul. Par conséquent, tous ont appliqué une approche dynamique qui convient à des systèmes comme la grille de calcul. Ces systèmes ont des charges de travail dynamiques, imprévisibles et diverses. De plus, la plupart des modèles proposés que nous ayons examinés ont été conçus sous des formes hiérarchiques et hétérogènes. De ce fait, nous pouvons dire que tous ces modèles ne sont pas des solutions tolérantes aux pannes et ce serait un inconvénient majeur pour la plupart d'entre eux. Nous avons également observé que tous ces modèles existants dans cette étude sont évolutifs.

Tableau 4.6: Comparaisons des modèles d'équilibrage de charge basés sur l'agent dans les grilles de calcul

Modèles d'équilibrage de charge basés sur l'agent	**Contribution**	**Amélioration des mesures de performance**	**Algorithmes comparés**	**Logiciels**	**Travaux futurs**	**Les inconvénients**
ABLBM	Agent Based Load Balancing Model : le modèle proposé prend en charge l'hétérogénéité, l'évolutivité et la dynamique des grilles. De plus, une architecture multi-agents a été suggérée, ainsi qu'une technique de migration des processus.	Utilisation des ressources Propriétés de passage à l'échelle temps de réponse	FCFS, EDF	Alea 2 simulator + JADE	Utiliser un agent mobile pour l'équilibrage de charge utiliser les règles de logique floue pour déterminer les états des nœuds et des clusters	Coûts de communication, Coûts de migration
AGLBM [71]	Agent Grid Load balancing : analyser la charge des nœuds et la migration des machines virtuelles des nœuds surchargés vers les sous-échargés. et utiliser le concept Hadoop-MapReduce	non considéré	Round robin cluster balancing, Least connection load balancing	JADE	se concentrer sur les performances d'exécution	L'instabilité, Non-évolutivité, Non -Tolérance aux pannes
DRAPM [72]	Distributed Resource Allocation Protocol : algorithme décentralisé s'inspire de la biologie, créant et dissociant de manière adaptative des clusters à partir de nœuds pour répondre à la demande des tâches.	propriétés de Passage à l'échelle, Coûts de latence	First-in First-out (FIFO) scheduling system	MASON	comparer avec d'autres algorithmes SRTF (Shortest Remaining Time First) avec d'autres mesures (le temps de réponse) et la collecte de données sur la distribution exacte de la demande de processus dans une file d'attente dans un scénario réel	Frais généraux de collecte d'information, Non tolérance aux pannes

ABRMM [73]	Agent-Based Resource Management: une gestion basée sur l'agent des ressources dans l'architecture de grille de calcul	La disponibilité des ressources	gLite grid middleware architecture	Simevents matlab	pas donné	Coûts de communication
ACOM [74]	Ant Colony Optimization: en utilisant les agents mobiles et l'approche d'optimisation par colonies de fourmis	Temps d'exécution	The distribution alternately in turns + Load balancing Theoretic	JADE	minimiser les mouvements des tâches et entraîner des coûts supplémentaires dans le processus de migration.	Coûts de migration, Non tolérance aux pannes
PBFLBM [75]	Priority Based Fuzzy Load Balancing: ordonnancement prioritaire basé sur agent et algorithme d'équilibrage de charge en utilisant la logique floue.	Temps de réponse, Capacité de ressource	PE_MinRC, FNN scheduler, et AlgHybird_LB	Gridsim Simulator	pas donné	Imprécision non fiabilité

Après avoir examiné les modèles existants de manière exhaustive, on peut constater que différents modèles ont pris en compte différentes métriques pour achever l'évaluation. Certains articles ont considéré une seule métrique de performance telle que R.Suros et al[73] et [74] H.Younes et al [74], tandis que certains ont considéré plusieurs objectifs pour des métriques telles que S.Banerjee and J.P. Hecker [72],et N.Rathore [75]. Ces modèles existants ont des limites qui doivent être corrigées. Il existe donc un besoin d'un algorithme qui peut offrir une utilisation maximale des ressources, un débit maximal, un temps de réponse minimal, un équilibrage de charge dynamique des ressources avec évolutivité et une certaine fiabilité. Le modèle proposé aborde la plupart des problèmes ci-dessus en prenant en charge l'hétérogénéité, l'évolutivité et la dynamique des grilles. De plus, il diminue le temps de réponse moyen et maximise l'utilisation des ressources.

Les modèles d'équilibrage de charge sont généralement multi-objectifs en fournissant des fonctionnalités telles que l'amélioration des performances et la réduction des coûts d'exploitation. Par conséquent, un compromis optimal entre divers objectifs contradictoires doit être préservé.

4.4 Conclusion

Dans ce chapitre, nous avons présenté et analysé un certain nombre d'expérimentations pour mettre en évidence le comportement de nos propositions et comparer les modèles proposés par autre modèles d'équilibrage de charge. L'analyse des résultats obtenus nous a permis de prouver que nos modèles permettent d'améliorer les performances du système en réduisant le temps moyen de réponse et maximiser l'utilisation de ressources.

Afin d'implémenter et d'évaluer nos modèles proposés, nous avons utilisé un simulateur de grille Alea 2. Ce simulateur a servi à mener des expérimentations que nous avons décrites et interprétées au fur et à mesure dans ce chapitre.

Conclusion & Perspectives

Dans cette thèse, nous avons tout d'abord, présenté un état de l'art sur les systèmes distribués et les grilles de calcul. Ensuite, nous avons décrits les composants et les fonctionnalités d'un système d'équilibrage de charge, notamment à travers les travaux qui s'intéressent à l'équilibrage de charge dans les grilles de calcul. Aussi, nous avons donné un aperçu général sur les systèmes multi-agents.

L'utilisation des systèmes multi-agents dans les grilles de calcul et la combinaison entre les deux techniques passant par les difficultés et les avantages qui entourent la tâche d'intégration des systèmes multi-agents dans les grilles de calcul, ont été tous traités par la suite avec une présentation des travaux connexes relatifs à l'équilibrage de charge basé sur des agents dans les grilles de calcul.

Nous avons constaté qu'il y a peu de travaux qui sont basés sur une technologie multi-agents traitant les problèmes liés à l'équilibrage de charge dans les grilles de calcul. Pour ces raisons, nous avons proposé dans le même contexte un modèle d'équilibrage de charge basé sur l'agent dans l'environnement de grille, dont l'objectif principal est d'avoir un système d'équilibrage de charge qui prend en compte les aspects d'hétérogénéité, de dynamicité et de fonctionnalité à grande échelle fortement présents dans une infrastructure de type grille. D'une autre part, la structure hiérarchique et distribuée du modèle proposé sert à faciliter à travers les nœuds de l'arbre tous les flux d'informations nécessaires à la stratégie d'équilibrage proposée.

Partant de ce modèle, nous avons ensuite développé une stratégie d'équilibrage qui privilège, quand cela est possible, un équilibrage de charge local pour éviter le recours au réseau de communication à grande échelle.

La stratégie d'équilibrage proposée est de type distribuée en procédant à la mesure où plusieurs opérations d'équilibrage peuvent être déclenchées en utilisant des informations de charge locales par rapport aux éléments composant une grille.

Nous avons aussi intégré une approche multi-agents pour améliorer l'adaptabilité, dont le modèle proposé vise à tirer parti des caractéristiques de l'agent pour créer un système autonome. L'implémentation de notre modèle a montré la pertinence de l'exploitation du système multi-agent qui est bien adapté au fonctionnement distribué et coopératif, comme il permet d'avoir un réseau évolutif avec un comportement dynamique selon les besoins.

En termes de simulation, des difficultés ont été rencontrées pour choisir une plateforme de test, bien qu'il existe un nombre important des simulateurs de grille disponibles. Cette difficulté est due à la nature de la problématique qui consiste à optimiser l'utilisation des ressources et à minimiser le temps de réponse en utilisant la technologie d'agent. L'implémentation d'un système multi-agent dédié à une grille de calcul nécessite une vraie plateforme qui prend en charge les caractéristiques des deux types de systèmes qui sont les grilles de calcul et le système multi-agent lui-même.

Pour atteindre ce but, nous avons tout d'abord développé une bibliothèque de classes qui simule les activités des agents. Ce développement a été fait en utilisant la plate-forme JADE. Ensuite, nous l'avons intégré avec le simulateur Alea 2.

Outre l'expérimentation de notre stratégie, nous avons utilisé un simulateur connu dans le domaine d'équilibrage de charge dans les grilles, à savoir Alea 2. A travers cette expérimentation, nous avons visé deux objectifs : montrer en premier lieu comment une stratégie d'équilibrage peut être intégrée à un simulateur de grilles et puis étudier le comportement de notre algorithme dans un simulateur de grilles. Par la suite, nous avons voulu à travers la deuxième expérimentation, voir comment le concept d'agent peut contribuer à la mise en place d'une stratégie d'équilibrage dans les grilles de calcul.

Les résultats présenté et discuté dans cette thèse sont assez encourageants, surtout après les avoir les comparé avec des algorithmes implémentés dans le simulateur Alea 2, tel que la stratégie d'ordonnancement FCFS et EDF. L'analyse de ces résultats a donné de bons indices sur le comportement de la stratégie proposée. Nous sommes arrivés à améliorer sensiblement les paramètres de performance que nous avons définie, notamment le temps de réponse moyen des tâches et l'utilisation des ressources de grille.

Dans la présente thèse, nous avons essayé d'exploiter les avantages offerts par les systèmes multi-agents dans le domaine d'équilibrage de charge à travers les grilles de calcul qui sont deux technologies qui peuvent être considérées comme complémentaires.

En effet, les SMA permettent de concevoir des systèmes émergents et coopératifs, chose qui caractérise le fonctionnement de l'environnement de grille.

Cette expérience a permet d'ouvrir plusieurs perspectives pour les travaux futures dans ce domaine. Ces perspectives concernent l'algorithme lui-même, l'architecture multi-agent et sa réalisation.

- **Optimisation de l'algorithme:** Nous pensons qu'il sera possible d'enrichir les algorithmes proposés par d'autres fonctionnalités telles que :

  1. La mise en place d'un outil d'évaluation du réseau de la grille, qui permettra d'évaluer la bande passante disponible entre les clusters. Ainsi, les heuristiques de transfert de tâches seraient en mesure d'ajuster leurs décisions en fonction des valeurs calculées par cet outil. Ceci permettra d'éviter le choix des clusters dont les liens réseau sont congestionnés.

  2. L'ajout dans la stratégie d'équilibrage, d'autres paramètres tels que : l'ensemble de seuils (les valeurs de seuils inférieur et supérieur) d'une part et d'une autre part les périodes de collecte d'information de charge. Les valeurs attribuées à ces paramètres sont en effet essentielles pour régler le comportement de la stratégie. A ce sujet, il nous semble qu'il serait important de maîtriser leur évaluation en étudiant les facteurs de corrélation qui peuvent exister entre ces paramètres et les différentes caractéristiques des tâches et/ou des nœuds d'une grille.

  3. Trouver d'autres solutions pour offrir plus de fiabilité aux migrations de processus.

  4. Pour mieux évaluer les apports et les performances de notre stratégie. Il nous semble qu'elle devrait être intégrée dans d'autres simulateurs connus dans le domaine des grilles de calcul, tel que HyperSim ou SimGrid. Ceci nous permettra de mesurer l'efficacité des simulateurs et de comparer les performances de notre stratégie.

5. Utiliser les règles de logique flou pour déterminer l'état de cluster et du nœud.

- **L'architecture multi-agent** : L'architecture multi-agent proposée comporte plusieurs agents. Elle peut être restructurée pour permettre plus de dynamique et paramétrage comme par exemple :

    1. Ajouter des agents mobiles, nous pouvons prendre l'aspect mobilité des agents. En effet, les agents mobiles peuvent être migrés à travers les nœuds de grille pour équilibrer la charge.

    2. Ajouter un mécanisme de sécurité comme la signature des agents ou carrément utiliser des agents de détection de comportement malveillant ou des mesures de confiance.

    3. Il est intéressant d'étudier les autres mécanismes de coordination (par exemple la négociation) et de les adapter au contexte spécifique de grille de calcul.

Annexe A

Alea 2 – Job Scheduling Simulator

Le simulateur Alea 2 [77] est basé sur les événements et est capable de traiter les problèmes courants liés à l'ordonnancement des tâches, tels que l'hétérogénéité des tâches ou des ressources et les modifications dynamiques de l'exécution (l'arrivée de nouveaux tâches ou les ressources échouent et redémarrent). L'Alea 2 est basé sur le simulateur plus connu "GridSim" [78]. Alea 2 représente une extension majeure du simulateur GridSim

L'extension couvre à la fois une conception améliorée des fonctionnalités étendues ainsi qu'une évolutivité améliorée avec une vitesse de simulation plus élevée. Enfin, une nouvelle interface de visualisation est introduite dans le simulateur. La partie principale du simulateur est un planificateur complexe qui incorpore plusieurs algorithmes d'ordonnancement fonctionnant soit sur la file d'attente ou bien sur le principe de planification (plan). Des structures de données supplémentaires sont utilisées pour conserver des informations sur l'état des ressources, les fonctions d'objectif et pour la collecte et la visualisation des résultats de la simulation. De nombreux objectifs typiques tels que l'utilisation de la machine, le ralentissement moyen ou le temps de réponse moyen sont inclus.

1. Taxonomie des outils de simulations

Il existe de nombreux simulateurs et boîtes à outils qui fournissent diverses fonctionnalités pour les simulations des clusters, du réseau et des environnements de grille. Parmi lesquels nous pouvons citer [77] :

MicroGrid [85] qui est plutôt un émulateur qu'un simulateur. Il peut être utilisé pour l'étude systématique du comportement dynamique des applications, des middlewares, des ressources et des réseaux. Le MicroGrid utilise le Globus API Toolkit 2.2 pour son exécution qui permet d'émuler avec précision des systèmes basés sur GTK 2.2 - cependant

de tels systèmes sont obsolètes puisque la version actuelle 4.2.1 utilise un modèle différent basé sur le concept de services Web.

Bricks [86] simule divers schémas d'ordonnancement sur un système de composants informatiques mondiaux à haute performance typique. Les briques peuvent simuler divers comportements des systèmes de composants informatiques mondiaux, en particulier le comportement des réseaux et les algorithmes d'ordonnancement. De plus, sa conception modulaire permet d'incorporer différents algorithmes d'ordonnancement et permet également l'incorporation de composants informatiques mondiaux existants via son interface étrangère.

BeoSim [87] a été mis en œuvre dans le but de l'étude d'algorithmes d'ordonnancement de tâches parallèles multi-sites dans le contexte d'une grille de calcul multi-cluster. Le BeoSim peut être piloté par une charge de travail synthétique ou une charge de travail réelle. Il fournit également un outil de visualisation basé sur le Java.

SimGrid [88] est un simulateur basé sur le langage de programmation "C" , il est très utilisé pour la simulation et le développement d'applications distribuées en environnements hétérogènes et distribués. Le SimGrid résout différents problèmes en utilisant différents environnements de programmation qui constituent différents paradigmes. L'outil MSG de Sim-Grid est largement utilisé pour l'évaluation de base et la simulation d'algorithmes d'ordonnancement, tandis que d'autres outils tels que le GRAS et le SMPI sont utilisés pour développer et étudier les applications réelles. Le SimDag fournit des fonctionnalités pour simuler l'ordonnancement des tâches parallèles avec les modèles de flux de travail DAG (Direct Acyclic Graphes).

Simbatch [89] est basé sur le MSG du SimGrid, permet d'évaluer les algorithmes de planification pour les ordonnanceurs de lots. Ni le Simgrid ni le Simbatch n'offrent de sortie de visualisation intégrée .Cependant, les deux simulateurs peuvent générer une trace qui peut être visualisée récemment via les outils de visualisation Pajé [90] ou le ViTE [91].

Simboinc [92] est un autre simulateur basé sur le Simgrid. Il est conçu pour simuler des grilles de bureau hétérogènes et volatiles et des systèmes informatiques volontaires. Le Simboinc simule une plate-forme client-serveur où plusieurs clients demandent de travailler à partir d'un serveur central. Le but de ce projet est de mettre à l'essai de nouvelles stratégies d'ordonnancement dans le BOINC (Berkeley Open Infrastructure for

Network Computing) et dans d'autres systèmes de bureau et de bénévoles. Les caractéristiques du client telles que la vitesse, la disponibilité de la charge du travail ou la disponibilité du réseau peuvent toutes être spécifiées dans les entrées de simulation. Alors que le SimGrid, le Simbatch et le SimBOINC sont tous basés sur le langage de programmation C, dont tous les simulateurs suivants sont basés sur le langage Java [93]. Tandis que Java est généralement moins efficace que C, où il permet un développement facile et portabilité sans effort [77].

Monarc 2 [94] est un cadre de simulation dont le but est de fournir un outil de conception et d'optimisation pour les systèmes informatiques distribués à grande échelle, en se concentrant sur les Expériences LHC (Large Hadron Collider) au CERN. Bien qu'il intègre un module d'ordonnancement simple, le principal but du Monarc 2 est de fournir une simulation réaliste des systèmes informatiques distribués et personnalisés pour des physiques des données, en même temps le Monarc 2 sert à offrir un environnement flexible et dynamique pour l'évaluation des performances d'une gamme d'architectures de traitement de données possibles. Le Monarc 2 étend le simulateur obsolète Monarc [95] en améliorant sa flexibilité et ses performances.

GridSim [78] est une boîte à outils de simulation de grille modulaire et flexible avec une très bonne documentation. Il est écrit en Java au-dessus d'une bibliothèque de simulation d'événements appelée SimJava [96]. Grâce au langage Java, le GridSim est une boîte à outils indépendante de la plateforme. Le GridSim fournit une fonctionnalité pour simuler l'environnement de base de la grille et son comportement en fournissant des implémentations simples des entités telles que les ressources de calcul ou les utilisateurs. Il permet également de simuler des tâches simples comme la topologie du réseau, le stockage de données et d'autres fonctionnalités utiles. Les implémentations fournies représentent la fonctionnalité de base, dont il est nécessaire de les étendre lors de l'exécution des simulations avec des exigences plus complexes. Cela peut être exécuté par la mise en œuvre de nouvelles classes Java héritées du GridSim existant, ce qui est également le cas de l'Alea 2 [77].

GridSim est utilisé par divers chercheurs dans leurs simulations. Il existe un ordonnanceur décentralisé mis en œuvre dans une version obsolète de GridSim, mais il ne prend pas en charge le comportement dynamique du système et il n'est pas capable de

simuler la topologie du réseau ou d'autres fonctionnalités récentes. La raison réside dans l'incompatibilité entre les anciennes et les récentes versions du GridSim [77].

Grid Scheduling Simulator (GSSIM) [97] est basé sur la boîte à outils GridSim.Il est développé depuis plusieurs années et est devenue accessible au public en 2009. Il devrait fournir une plateforme d'ordonnancement de grille facile à utiliser pour permettre des simulations d'un large éventail d'algorithmes d'ordonnancement dans des infrastructures de grille hétérogènes à plusieurs niveaux. Cependant, le GSSIM rencontre certains problèmes tels qu'une exécution lente, une faible évolutivité et de mauvais résultats de visualisation. De plus, le GSSIM n'est pas compatible avec les versions standards de GridSim (y compris le dernier GridSim 5) car il utilise sa propre branche basée sur une version modifiée de version 4.

Finalement, il est à conclure qu'il existe plusieurs boites d'outils de simulation différents qui couvrent divers aspects des simulations de grille et de cluster. Mais malheureusement plusieurs d'entre elles ne sont plus utilisables actuellement en raison de l'incompatibilité (MicroGrid), de la reconstruction majeure (SimBOINC), ou même en raison des décisions des auteurs (Bricks, BeoSim) [77].

2. Conception de simulateur

L'Alea 2 étend les classes Gridsim existantes et offre également de toutes nouvelles classes pour fournir un simulateur "prêt à l'emploi" d'ordonnancement des tâches. Il a été conçu en fonction de nos besoins en matière de sa capacité de simulation. Cela implique la possibilité d'effectuer des simulations complexes qui impliquent des caractéristiques réalistes des systèmes réels. Pour être plus précis, Alea 2 est capable de simuler des tâches séquentielles et parallèles des clusters de calcul, des exigences de tâches spécifiques ainsi que la dynamique du système comme les pannes de machine ou l'activité des utilisateurs. Il prend également en charge divers critères d'optimisation, notamment l'utilisation, le ralentissement, le temps de réponse, le temps d'attente et des critères d'équité (le temps d'attente normalisé de l'utilisateur autant qu'exemple) [77].

Alea 2 comme le GridSim, est un simulateur modulaire basé sur des événements et est composé d'entités indépendantes qui implémentent la fonctionnalité de simulation souhaitée comme il est présenté dans la figure 22.

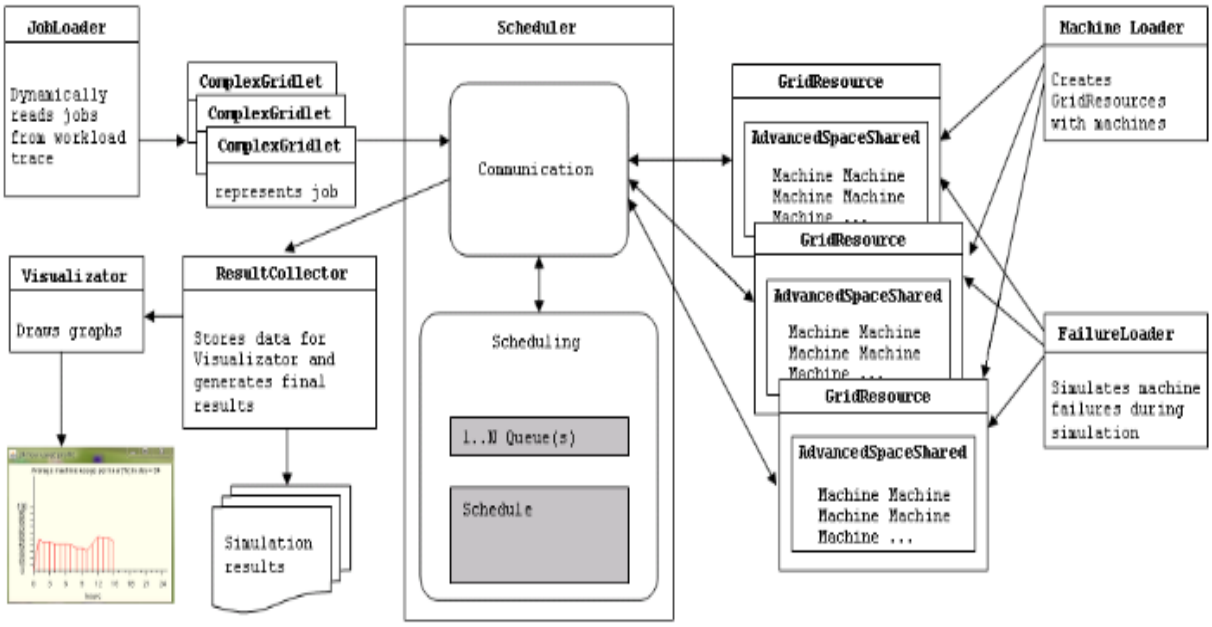


Figure 22 : Principales parties du simulateur Alea 2 [77]

La simulation est initialisée par la classe **ExperimentSetup** qui crée des instances d'ordonnanceur du chargeur de tâches et machines, du chargeur d'échecs et de d'autres entités comme requis par le GridSim standard [77].

L'entité **MachineLoader** effectue l'initialisation de l'environnement de calcul simulé. Elle lit les données décrivant les machines à partir d'un fichier et crée des ressources de grille en conséquence. GridSim lui-même ne fournit pas de telles fonctionnalités et les paramètres de la machine doivent être "codés en dur" dans le fichier java source. Alea2 permet de modifier les paramètres des machines sans avoir recompilé l'ensemble du programme [77].

La classe **JobLoader** lit le fichier contenant les descriptions de tâches et crée des instances de tâches de manière dynamique au fil du temps. **JobLoader** prend en charge plusieurs formats de trace, notamment le format Grid Workloads (GWF) de Grid Workloads Archive et le format standard des charges de travail (SWF) des archives des charges de tâches parallèles. Lorsque le temps de simulation est égal au temps de soumission du tâche, le **JobLoader** envoie la tâche à l'ordonnanceur. Le **JobLoader** lit une seule tâche à la fois pour limiter l'espace de mémoire requis et aussi pour permettre l'utilisation de très grandes traces de charge de la tâche. Cette approche est nécessaire car l'entité de charge de tâche d'origine de GridSim lit toutes les données à la fois, ce qui, pour

les charges de tâche importantes, entraîne l'échec de la simulation en raison de l'erreur de mémoire insuffisante de Java [77].

La tâche lui-même est représentée par l'instance de la classe **ComplexGridlet**. GridSim ne fournit qu'une implémentation triviale d'une tâche dans sa classe **Gridlet**. Le **ComplexGridlet** étend cette classe en permettant de simuler des scénarios plus réalistes où chaque tâche peut nécessiter des propriétés supplémentaires telles que la date limite, le temps d'exécution estimé, les paramètres spécifiques de la machine et d'autres contraintes basées sur la vie réelle [77].

Comme dans GridSim, la ressource est représentée par l'instance **GridResource** et est gérée par la stratégie d'ordonnancement locale. Malheureusement, aucune des politiques actuellement fournies ne prend en charge l'exécution de tâches parallèles et la simulation des pannes de la machine en même temps. De plus, la co-allocation de plusieurs machines pour l'exécution des tâches n'est pas disponible. Par conséquent, une nouvelle politique d'allocation appelée **AdvancedSpaceShared** a été développée pour l'Alea 2 sur la base de la politique **SpaceShared** de GridSim. Elle permet d'exécuter des tâches séquentielles et parallèles sur le nombre spécifié de CPU en utilisant la politique d'allocation de processeur de partage d'espace. Ceci permet de simuler des scénarios plus réalistes impliquant les tâches parallèles ainsi que les simulations de pannes de machine. De plus, il comprend une implémentation plus efficace du passage de message qui permet des améliorations importantes de la vitesse de simulation [77].

Le **FailureLoader** sert à lire le fichier contenant les descriptions des pannes de machine. Une fois que le temps de simulation a atteint l'heure de début de l'échec, la machine appropriée est définie pour échouer en tuant tous les tâches en cours d'exécution sur cette machine. Lorsque la période d'échec passe, la machine sera automatiquement redémarrée. Les pannes de machine peuvent être utilisées pour simuler l'ajout d'une nouvelle machine ou le retrait permanent d'une machine [77].

La nouvelle classe **Visualizator** génère la sortie graphique de la simulation, dont le Gridsim lui-même permet des visualisations affichant le processus d'allocation des tâches sur les machines au fil du temps [77]. Le visualiseur étend cette fonctionnalité en affichant des informations supplémentaires utiles pour le réglage et le débogage des algorithmes d'ordonnancement. Jusqu'à présent, plusieurs résultats couvrant différents objectifs sont

pris en charge et affichés. Il s'agit de l'utilisation globale des ressources, l'utilisation du cluster et l'usage du nombre de tâches en attente et en cours d'exécution et du nombre de CPU demandés, soit utilisés et disponibles. En plus de cela, le pourcentage de CPU défaillants et en cours d'exécution par cluster peut également être affiché. Le visualiseur peut fonctionner de deux manières différentes. Pour la première, la visualisation est générée en continu au fur et à mesure de la simulation, alors que lors de la deuxième, les graphiques sont générés lorsque la simulation est terminée, en utilisant les résultats de la simulation comme entrée. Les résultats sont collectés en continu par la classe **ResultCollector**. Une fois la simulation terminée, le **ResultCollector** les stocke dans des fichiers de format csv qui peuvent être facilement utilisés comme entrée pour d'autres outils (Calc, Excel, Feuille de calcul, etc.) et il peut aussi enregistrer les graphiques générés dans un fichier bitmap préféré (png, jpg, bmp, gif) [77].

3. Exactitude du simulateur

L'exactitude du simulateur a été vérifiée en analysant les implémentations d'algorithmes et les sorties de simulation. Premièrement, les implémentations d'algorithmes d'ordonnancement ont été vérifiées par rapport à leurs pseudo-codes connus. Ensuite, les résultats de simulation des algorithmes existants tels que FCFS, EDF, EASY ou Conservative Backfilling ont été comparés aux résultats connus de la littérature. Dans le cas du remblayage flexible, les détails de la mise en œuvre et les résultats de la simulation ont été directement vérifiés par les auteurs de cet algorithme. Les politiques proposées et les algorithmes d'optimisation ont été vérifiés dans le cadre de plusieurs expériences où le comportement attendu (spécification de l'algorithme) a été comparé aux résultats de la simulation et des tracés. De même, la sortie de simulation graphique a joué un rôle important dans l'analyse [77]

4. Caractéristiques de simulateur Alea 2

Alea 2 peut être facilement étendu grâce à l'adoption de paradigme orienté objet. Le simulateur est modulaire, ce qui signifie que différentes fonctionnalités sont mises en œuvre dans différents classes. En outre, la classe cruciale **Scheduler** est divisée en pièces séparées. Ainsi, un nouvel algorithme d'ordonnancement ou une nouvelle fonction objective peut être ajoutée via l'extension ou une modification des classes existantes est produite. De même, si une nouvelle tâche ou un nouveau type de tâche est demandé, seul la

classe Complex-Gridlet doit être étendue ou modifiée, laissant les autres classes intactes. Par conséquent, toutes les modifications sont encapsulées et le développement sera simple.

Annexe B

La platforme Jade

JADE est une plate-forme logicielle qui fournit des fonctionnalités de base de la couche middleware qui sont indépendantes de l'application spécifique et qui simplifient la réalisation d'applications distribuées qui exploitent l'abstraction des agents logiciels [47]. Un avantage important de JADE est qu'il implémente cette abstraction sur un langage orienté objet bien connu ou d'une Java en fournissant une API simple et conviviale.

1. Architecture de JADE

Une plateforme JADE est composée de conteneurs d'agents qui peuvent être distribués sur le réseau. Les agents vivent dans des conteneurs qui sont le processus Java qui fournit le runtime JADE et tous les services nécessaires à l'hébergement et à l'exécution des agents. Il existe un conteneur spécial, appelé conteneur principal qui représente le point d'amorçage d'une plate-forme: c'est le premier conteneur à être lancé, dont tous les autres conteneurs doivent se joindre à un conteneur principal en s'enregistrant avec lui. La figure 23 montre les principaux éléments architecturaux d'une plate-forme JADE [98].

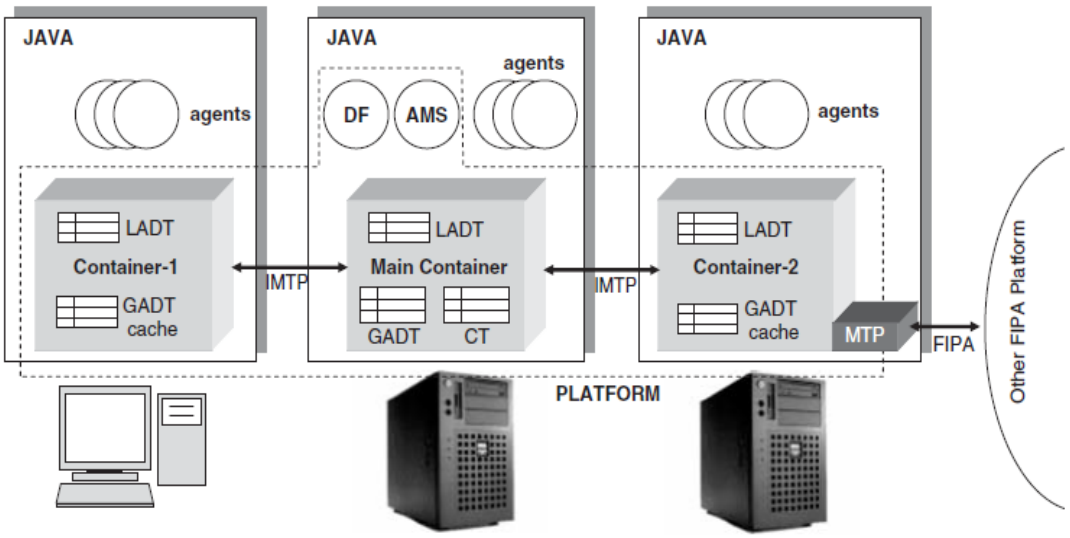


Figure 23 : La relation entre les principaux éléments architecturaux [98].

Lorsque le conteneur principal est lancé, deux agents spéciaux sont automatiquement instanciés et démarrés par le JADE, dont les rôles sont définis par la norme de gestion des agents FIPA:

- **AMS (Agent Management System)** est l'agent qui supervise l'ensemble de la plateforme. C'est le point de contact de tous les agents qui ont besoin d'interagir pour accéder aux pages blanches de la plateforme ainsi que pour gérer leur cycle de vie. Chaque agent est tenu de s'inscrire auprès de l'AMS (effectué automatiquement par JADE au démarrage de l'agent) afin d'obtenir un AID valide.
- **DF (Directory Facilitator)** est l'agent qui met en œuvre le service des pages jaunes, utilisé par tout agent souhaitant enregistrer ses services ou rechercher d'autres qui sont disponibles. Le DF accepte également les abonnements d'agents qui souhaitent être notifiés à chaque fois qu'un enregistrement ou une modification de service (correspond à certains critères spécifiés) est effectué. Plusieurs DF peuvent être démarrés simultanément afin de distribuer le service de pages jaunes sur plusieurs domaines. Ces DF peuvent être fédérés si nécessaire, en établissant des enregistrements croisés entre eux. Ces enregistrements permettent la propagation des demandes d'agent à travers toute la fédération.

2. Programmation avec JADE

JADE est un outil Java complet et donc la création d'un système multi-agent basé sur JADE implique seulement la création de classes Java sans aucune expertise significative en programmation Java.

Création des agents

La création d'un agent JADE est aussi simple que de définir une classe qui étend la classe jade.core.Agent et implémenter la méthode setup (). Le code ci-dessous est utilisé pour créer le conteneur principal et l'AgentGrid (code de notre programme)

```
AgentGrid agentgrid = new AgentGrid() ;
Runtime rt= Runtime.instance();
Properties p=new ExtendedProperties();
p.setProperty("gui","true");
```

```
ProfileImpl pc=new ProfileImpl(p);
jade.wrapper.AgentContainer container
=rt.createMainContainer(pc);
  try {
        container.start();
         } catch (ControllerException e) {
              // TODO Auto-generated catch block
              e.printStackTrace();
                        }
AgentController agentcontroller;
Profile pp;
pp=new ProfileImpl();
pp.setParameter(Profile.CONTAINER_NAME,"container-1");
jade.wrapper.AgentContainer
agentcontainer=rt.createAgentContainer(pp);
agentcontroller=agentcontainer.createNewAgent("agentgrid",
"AgentBasedLoadBalancing.AgentGrid", new Object[]{});
agentcontroller.start();
```

Le code ci-dessous est employé pour but de créer les AgentsCluster et les AgentsMigration.

```
AgentController agentcontroller;
AgentController agentcontroller1;
Profile pp;
pp=new ProfileImpl();
pp.setParameter(Profile.CONTAINER_NAME,"container-2");
jade.wrapper.AgentContainer
agentcontainer=rt.createAgentContainer(pp);
for (int i = 0; i <14 ; i++)
{
agentcontroller=agentcontainer.createNewAgent("cluster_"+i,"
AgentBasedLoadBalancing.AgentCluster", new Object[]{});
agentcontroller1=agentcontainer.createNewAgent("AgentMigration_"+
i," AgentBasedLoadBalancing.AgentMigration", new Object[]{});
agentcontroller.start();
agentcontroller1.start();}
```

Identificateurs d'agents

Conformément aux spécifications du FIPA, chaque instance d'agent est identifiée par un «identifiant d'agent». Dans le JADE, un identifiant d'agent est représenté comme une instance de la classe jade.core.AID. La méthode getAID () de la classe Agent permet de récupérer l'identifiant d'agent local. Un objet AID comprend un nom unique autour du monde (GUID) plus un certain nombre d'adresses. Le nom dans JADE a la forme <local-name> @ <platform-name> de sorte qu'un agent appelé Peter vivant sur une plate-forme appelée foo-platform aura Peter @ foo-platform comme nom unique au monde. Les adresses incluses dans l'AID sont les adresses de la plate-forme occupée par l'agent. Ces adresses ne sont utilisées que lorsqu'un agent doit communiquer avec un autre agent résidant sur une autre plateforme FIPA conforme.

La classe AID fournit des méthodes pour récupérer le nom local (getLocalName ()), le GUID (getName ()) et les adresses (getAllAddresses ()).

Initialisation de l'agent

Certainement, les bibliothèques JADE doivent se trouver dans le chemin de classe pour que la compilation réussisse. À ce stade, afin d'exécuter un Hello-World-Agent, c'est-à-dire une instance de la classe HelloWorldAgent, le runtime JADE doit être démarré et un nom local pour l'agent à exécuter doit être choisi:

```
java -classpath <JADE-classes>;. jade.Boot Peter:HelloWorldAgent
```

Résiliation de l'agent

Pour faire terminer un agent, sa méthode doDelete () doit être appelée. Similaire à la méthode setup () qui est invoquée pour initialiser un agent, la méthode takeDown () est invoquée juste avant la fin d'un agent afin d'effectuer diverses opérations de nettoyage.

OneShotBehaviours, CyclicBehaviours and GenericBehaviours

Les trois principaux types de comportement disponibles avec JADE sont les suivants:

- Les comportements «one-shot» sont conçus pour se terminer en une seule phase d'exécution; leur méthode action () n'est donc exécutée qu'une seule fois. La classe jade.core.behaviours.OneShotBehaviour implémente déjà la méthode

done () en renvoyant true et peut être facilement étendu pour mettre en œuvre de nouveaux comportements ponctuels.

```
public class MyOneShotBehaviour extends OneShotBehaviour {
public void action() {
// perform operation X  }}
```

Dans cet exemple, l'opération X n'est effectuée qu'une seule fois.

- Les comportements «cycliques» sont conçus pour ne jamais se terminer; leur méthode action () exécute les mêmes opérations à chaque appel. La classe jade.core.behaviours.CyclicBehaviour implémente déjà la méthode done () en renvoyant false et peut être facilement étendue pour mettre en œuvre de nouveaux comportements cycliques. Exemple de notre programme est montré ci dessous.

```
addBehaviour( new CyclicBehaviour( this ) {
public void action() {
ACLMessage msg = receive();
if (msg != null) {
if (msg.getContent().startsWith("ClusterName")) {
ClusterName=msg.getContent().substring(11);
filesize=msg.getContent().substring(11);
gui.showMessage("sender:"+msg.getSender(),true);
//gui.showMessage(ClusterName,true);
gui.showMessage("ClusterName:"+filesize,true);
machlist(ClusterName);
}}}});
```

- Les comportements génériques incorporent un déclencheur d'état et exécutent différentes opérations en fonction de la valeur d'état. Ils se terminent lorsqu'une condition donnée est remplie.

```
public class ThreeStepBehaviour extends Behaviour {
private int step = 0;
public void action() {
switch (step) {
```

```
case 0:  // perform operation X
step++; break;
case 1: // perform operation Y
step++; break;
case 2: // perform operation Z
step++; break;
}}
public boolean done() {
return step == 3; }}
```

Dans cet exemple, la variable step implémente l'état du comportement. Les opérations X, Y et Z sont exécutées séquentiellement, après quoi le comportement se termine.

JADE offre également la possibilité de composer des comportements d'ensemble pour créer des comportements complexes. Cette fonctionnalité est particulièrement très pratique lors de la mise en œuvre de tâches complexes.

Tous les comportements héritent des méthodes onStart () et onEnd () de la classe Behavior. Ces méthodes sont exécutées une seule fois juste avant le premier appel à la méthode action () et juste après la méthode done () renvoie true. Ils sont destinés à effectuer des opérations d'initialisation et de fin spécifiques à la tâche. Contrairement aux méthodes action () et done () déclarées abstraites, elles ont une implémentation vide par défaut permettant aux développeurs de les implémenter uniquement si elles le souhaitent.

Communication des agents

La communication entre les agents est probablement la caractéristique la plus fondamentale du JADE et est implémentée conformément aux spécifications du FIPA. Le paradigme de communication est basé sur le passage de messages asynchrones. Ainsi, chaque agent possède une «boîte aux lettres» (la file d'attente de messages de l'agent) où le runtime JADE publie des messages envoyés par d'autres agents. A chaque fois qu'un message est publié dans la file d'attente de messages de la boîte aux lettres, l'agent destinataire est averti. Cependant, la récupération du message de la file d'attente par l'agent pour achever le traitement est une conception qui suit le choix du programmeur d'agent.

Chaque message comprend les champs suivants:

- L'expéditeur du message.
- La liste des récepteurs.
- L'acte de communication (également appelé «performatif») indiquant ce que l'expéditeur a comme intention de réalisation en envoyant le message. Par exemple, si le performatif est REQUEST, l'expéditeur veut que le destinataire effectue une action, et s'il est INFORM, l'expéditeur souhaite que le destinataire soit informé. En effet, s'il s'agit d'un PROPOSE ou d'un CFP (Appel à Propositions), l'expéditeur souhaite engager une négociation.
- Le contenu contenant les informations réelles à échanger par le message (par exemple, l'action à effectuer dans un message REQUEST ou le fait que l'expéditeur souhaite divulguer un Message INFORM, etc.).
- La langue du contenu indiquant la syntaxe utilisée pour exprimer le contenu. L'expéditeur et le récepteur doivent être capables de coder et d'analyser des expressions conformes à cette syntaxe pour que la communication soit efficace.
- L'ontologie indiquant le vocabulaire des symboles utilisés dans le contenu. L'expéditeur et le destinataire doivent tous attribuer la même signification à ces symboles pour que la communication soit efficace.
- Certains champs supplémentaires sont utilisés pour contrôler plusieurs conversations simultanées et pour spécifier aussi des délais d'expiration pour la réception d'une réponse, tels qu'une id de conversation, reply-with, in-reply-to et reply-by.

Un message dans JADE est implémenté en tant qu'objet de la classe jade.lang.acl.ACLMessage qui fournit des méthodes get et set pour accéder à tous les champs spécifiés par le format ACL. Tout les performatives définies dans la spécification FIPA sont mappées en tant que constantes dans la classe ACLMessage. En ci-dessus un exemple de notre programme.

```
ACLMessage msg = receive();
if (msg != null) {
if (msg.getContent().startsWith("numnodes")) {
gui.showMessage("sender:"+msg.getSender(),true);
numnode=msg.getContent().substring(8);
numnodes=Integer.parseInt(numnode);
createnode(numnodes);
gui.showMessage("number of nodes: " + numnode,true);
}
if (msg.getContent().startsWith("ClusterID")) {
Clusterid=msg.getContent().substring(9);
clusterid=Integer.parseInt(Clusterid);
gui.showMessage("sender:"+msg.getSender(),true);
gui.showMessage("ClusterID: " + clusterid,true);
}
if (msg.getContent().startsWith("numJobs")) {
gui.showMessage("sender:"+msg.getSender(),true);
numjobs=msg.getContent().substring(7);
numjob=Integer.parseInt(numjobs);
gui.showMessage("number of jobs: " + numjob,true);
}}
```

L'envoi des messages

L'envoi d'un message à un autre agent est assez simple que de remplir les champs d'un objet ACLMessage et puis appeler la méthode send () de la classe Agent. Le code ci-dessous exemple de notre programme.

```
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
ACLMessage msg11 = new ACLMessage(ACLMessage.INFORM);
ACLMessage msg12 = new ACLMessage(ACLMessage.INFORM);
msg.setLanguage("English");
msg.setOntology("ClusterName");
msg.addReceiver(new AID("cluster_"+j, AID.ISLOCALNAME));
msg11.addReceiver(new AID("cluster_"+j, AID.ISLOCALNAME));
msg12.addReceiver(new AID("cluster_"+j, AID.ISLOCALNAME));  j++;
msg.setContent("ClusterName"+msg1);
```

```
msg11.setLanguage("English");
msg11.setOntology("numnodes");
msg11.setContent("numnodes"+msg4);
send(msg);
send(msg11);
```

Recevoir des messages

Comme il a été mentionné précédemment, le moteur d'exécution JADE publie automatiquement les messages dans la file d'attente des messages privés d'un destinataire dès leur arrivée. Un agent peut récupérer des messages dans sa file d'attente de messages par le moyen de la méthode receive (). Cette méthode renvoie le premier message dans la file d'attente des messages (entraînant ainsi sa suppression) ou le résultat soit null si la file d'attente de messages est vide, dont le message revient immédiatement. Le code ci-dessous montre l'exemple de notre programme adopté.

```
protected void setup(){
gui =new AgentLBCGui();
gui.setAgentLBC(this);
ACLMessage msg = receive();
if(msg.getContent()!=null){
gui.showMessage(msg.getContent(),true);
}}
```

Bibliographie

[1] R. Buyya, S. Venugopal, “A Gentle Introduction to Grid Computing and Technologies”, *CSI Communications*, Vol. 29, No. 1, pp. 9–19, July 2005.

[2] S.A. Tanenbaum, V.M.Steen ,"Distributed systems : principles and paradigms",. *Book*, Prentice Hall, ISBN 0130888931, 2002.

[3] F.Wautelet , "Régulation de charge dans les systèmes distribués : architecture et algorithmes", *Mini-Workshop ” Systèmes Coopératifs. Matière Approfondie”*, Institut d’informatique, Namur, Belgique, 2002.

[4] M. Meddeber , Y. Belabbes, "équilibrage de charge pour les grilles de calcul : classe des taches dépendantes et indépendantes", *Proceedings of the 2nd Conférence Internationale sur l'Informatique et ses Applications (CIIA'09)* , Saida, Algeria, 2009.

[5] S.Idrissa , " Routage des Transactions dans les Bases de Données à Large Echelle ",. *Thèse de Doctorat*, l'université Pierre et Marie Curie (Paris VI), France, 2010.

[6] I.Foster , C.Kesselman,"The Grid: Blueprint for a New Computing Infrastructure", *Morgan Kaufmann Publishers*, USA, 1999.

[7] B.Yagoubi, "Equilibrage de charge dans les grilles de calcul", *PhD Thesis* , university of Oran, Novembre 2007.

[8] R.Buyya,"Economic-Based Distributed Resource Management and Scheduling for Grid Computing", *PhD Thesis* , Monash University, Melbourne, Australia, April 2002.

[9] I.Foster , C.Kesselman AND S.Tuecke, " The anatomy of the Grid : Enabling scalable virtual organisations", *International Journal of Supercomputing Applications*, pp.6–7, 10 May 2001.

[10] M.Meddeber, "Placement dynamique de tâches dans une Grille de Calcul" , *PhD Thesis*, university of Oran , Juin 2012.

[11] M.AL-Fawair, "A Framework for Evolving Grid Computing Systems" ,*PhD Thesis* ,De Montfort University ,United Kingdom, England, 2009

[12] L.Ferreira,V.Berstis AND J.Amstrong, "Introduction to grid computing with globus", *IBM RedBook*, pp. 89–93, 2002.

[13] T.Casavant , J.G.Kuhl,"A taxonomy of scheduling in general-purpose distributed computing systems" , *IEEE Transactions on Software Engineering*,Vol. 14, No. 2, pp. 141–145, February 1988.

[14] D.L. Eager, E.D. Lazwska AND J. Zahorjan,"A comparison of receiver-initiated and sender-initiated adaptative load sharing", *Performance Evaluation*,Vol. 6, No.1, pp. 53–68, 1986.

[15] B.Yagoubi , "Modèle d’équilibrage de charge pour les grilles de calcul", *Revue Africaine de la Recherche en Informatique et Mathématiques Appliquées: ARIMA*, vol.7, pp. 1–19, 2007.

[16] P. Bouvry," Placement de tâches sur ordinateurs parallèles à mémoire distribuée", *PhD thesis*, Institut National Polytechnique de Grenoble, Laboratoire de Modélisation et de Calcul de l’IMAG, 1994.

[17] F.Guinand,"Ordonnancement avec communicatins pour architectures multiprocesseurs dans divers modèles d'exécution", *PhD thesis*, Institut National Polytechnique de Grenoble, 1995.

[18] M.Theimer , K. Lantz,"Finding idle machines in a workstation based distributed system", *IEEE Transactions on Software Engineering*, Vol. 15, No.11, pp.1444–1458, November 1989.

[19] D. Eager, E. Lazowska, J. Zahorjan, "A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing", *Performance Evaluation*,Vol.6, No.1, pp. 53–68, May 1986.

[20] M. Baker, R. Buyya AND D. Laforenza, "Grids and grid technologies for wide-area distributed computing", *Journal of Software Practice and Experience*, Vol.32, No.15, pp.1437–1466, December 2002.

[21] N.Muthuvelu, J. Liu, N. Lin Soe, S.Venugopal, A. Sulistio AND R. Buyya."A dynamic job grouping-based scheduling for deploying applications with ne-grained tasks on global grids", *In Proceedings of the Australasian Workshop on Grid Computing and e-Research (AusGrid2005)*, vol.44, Newcastle, NSW, Australia, January/February 2005.

[22] A. K. Aggarwal , R. D. Kent, " An adaptive generalized scheduler for grid applications", *In Proceedings of the 19th International Symposium on High Performance Computing Systems and Applications*, pp.15–18, Guelph, Ontario Canada, May 2005.

[23] M. Arora, S.K. Das, R. Biswas, " A decentralized scheduling and load balancing algorithm for heterogeneous grid environments", *In Proceedings of Workshop on Scheduling and Resource Management for Cluster Computing*, pp. 499–505, Vancouver, Canada, August 2002.

[24] M. Dobber, G. Koole AND R. van der Mei,"Dynamic load balancing for a grid application", *In Proceedings of the 11th Annual International Conference on High Performance Computing (HiPC2004)*, pp.342–352, Bangalore, India, December 2004.

[25] D.Ferrari , S. Zhou ,"An empirical investigation of load indices for load balancing applications",*Technical Report UCB/CSD-87-353*, EECS Department, University of California, Berkeley, 1987.

[26] Y. Belabbas," Modéle d'équilibrage de charge pour les grilles de calcul", *Revue Africaine de la Recherche en Informatique et Mathématiques Appliquées*, 2007.

[27] J. M. Schopf, " Ten Actions When Grid Scheduling" ,*Chapter 2 in Grid Resource Management for Grid Computing.* Kluwer Publishing, October 2003.

[28] N.Rathore , I.Channa ," Load balancing and job migration techniques in grid:a survey of recent trends", *Wireless Pers Commun* , Vol.79, No.3,pp.2089–2125,2014.

[29] D.Nanthiya, P.Keerthika,"Hierarchical load balancing GridSim architecture with fault tolerance" , *International Journal of Scientific and Engineering Research*, Vol.4,No.5,pp.399–403,2013.

[30] S.Goswami , A.AD.Sarkar ,"comparative study of load balancing algorithms in computational grid environment", *In Proceedings of the 5th IEEE international conference on computational intelligence, modelling and simulation*; pp. 99–104,2013.

[31] B.Yagoubi , M. Meddeber, "Distributed Load Balancing Model for Grid Computing", *Revue ARIMA*,Vol.12, pp. 43-60,2010.

[32] N.Malarvizhi ,VR.Uthariaraj,"Hierarchical load balancing scheme for computational intensive jobs in grid computing environment", *In:Proceedings of the 1st IEEE international conference on advanced computing*, pp.97–104, 2009.

[33] R.Shah ,B.Veeravalli AND M.Misra ,"On the design of adaptive and decentralized load-balancing algorithms with load estimation for computational grid environments", *IEEE Transactions on Parallel and Distributed Systems*,Vol.18,No.12, pp.1675–1686, 2007.

[34] KQ .Yan, SS .Wang, SC .Wang AND CP .Chang," Towards a hybrid load balancing policy in grid computing system" , *Expert Systems with Applications* ,Vol.36 ,No.10, pp.12054-12064, 2009.

[35] Y.Li ,Y.Yang ,M.Ma AND L.Jhou ,"A hybrid load balancing strategy of sequential tasks for grid computing environments", *Future Generation Computer Systems*,Vol.25,No.8, pp.819-828 2009.

[36] S.Anousha, M.Ahmadi ,"An improved Min–Min task scheduling algorithm in grid computing",*In Proceedings of the international conference on grid and pervasive computing(GPC'13)*, vol.7861 ,pp.103–113, 2013.

[37] S.Penmatsa, AT.Chronopoulos ,"Game- theoretic static load balancing for distributed systems", *Journal of Parallel and Distributed Computing*,Vol.71,No.4, pp.537-555,2011.

[38] Zhang .Lei, Li .Xueyong AND Ru.Bei, "The research of mobile agent-based mobile grid resource discovery", *International Conference on Intelligent Computing and Cognitive Informatics (ICICCI 2010)*, 2010.

[39] M.Paletta, P.Herrero, "A Multi-agent Task Delivery System for Balancing the Load in Collaborative Grid Environment", *in IFIP International Federation for information Processing*, Vol.296, pp. 365–371, 2009.

[40] J. Cao, DP.Spooner , SA.Jarvis AND GR.Nudd ,"Grid load balancing using intelligent agents", *Future Generation Computer Systems* ,Vol.21,No.1,pp.135-149 ,2005.

[41] MA.Salehi , H.Deldari AND BM.Dorri, "MLBLM: A multi-level load balancing mechanism in agent-based grid", *In: Proceedings of the 8th international conference on distributed computing and networking*,pp.157–62,2006.

[42] J.Balasangameshwara , N.Raju ,"Performance-driven load balancing with a primary-backup approach for computational grids with low communication cost and replication cost", *IEEE Transactions on Computers*,Vol.62,No.5, pp.990-1003,2013.

[43] K. Sycara, T. Norman, J. Giampapa, M. Kollingbaum, C. Burnett, D. Masato, M. McCallum, AND M. Strub," Agent support for policy-driven collaborative mission planning", *The Computer Journal*, Vol.53,No.5,pp.528--540, June 2010.

[44] J.P.Jamont,"DIAMOND: Une approche pour la conception de systèmes multi-agents embarqués", *PhD thesis*, INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE, France,2005

[45] J.Ferber,"Les systèmes multi-agents Vers une intelligence collective",InterEditions, Paris,1995.

[46] M.WOOLDRIGE ,N. R. JENNINGS,"Intelligent agents : Theory and practice", Journal of The knowledge engineering review,Vol.10,No.2, pp.115-152, 1995.

[47] M.Woolddridge,N.R.Jennings AND D.Kinny,"The Gaia methodology for agent-oriented analysis and design", *Journal of Autonomous Agents and Multi-Agent Systems*, Vol.3,No.3, pp.285-312, 2000.

[48] H.Nwana, L. Lee AND N. Jennings, "Coordination in software agent systems" ,*BT Technology Journal* ,Vol.14,No.4 ,1994.

[49] T.W.Malone,K.Crowston ,"What is coordination theory and how it can help design cooperative", *Proceedings of the 1990 ACM conference on Computer-supported cooperative work* , pp. 357-370.1990.

[50] Workshop on Agent based Cluster and Grid Computing, 2001, Available from: *http://www.cs.cf.ac.uk/User/O.F.Rana/agent-grid/*

[51] A. Singh, D. Juneja, and A. K. Sharma, "Agent Development Toolkits", *International Journal of Advancements in Technology*,Vol. 2, No. 1, pp. 158–164, 2011.

[52] A. Dorri, S. S. Kanhere, and R. Jurdak," Multi-Agent Systems: A Survey", *IEEE Access*, Vol. 6, pp. 28573–28593, 2018.

[53] J. Fu , J.Wang, "Adaptive coordinated tracking of multi-agent systems with quantized information", *Systems & Control Letters*, Vol.74, pp.115–125, 2014.

[54] S. Liu, L. Xie, and H. Zhang,"Containment control of multi-agent systems by exploiting the control inputs of neighbors", *International Journal of Robust and Nonlinear Control*, Vol. 24, No. 17, pp. 2803–2818, 2014.

[55] S. Khodaverdian, "On the synchronization of linear heterogeneous multiagent systems in cycle-free communication networks", *Proceedings of 2014 International Conference on Modelling, Identification and Control, ICMIC 2014*, pp. 190–195, 2014.

[56] Y. Q. Chen , Z. Wang, "Formation control: a review and a new consideration, in Intelligent Robots and Systems", *IEEE/RSJ International Conference on. IEEE, (IROS 2005)* , pp. 3181–3186,2005.

[57] M. H. Bowling,"Convergence and no-regret in multiagent learning*", in Conference : Advances in Neural Information Processing Systems 17 , NIPS*, pp.209–216,2004.

[58] D. Chakraborty , P. Stone,"Multiagent learning in the presence of memory-bounded agents", *Autonomous agents and multi-agent systems*,Vol. 28, No. 2, pp.182–213, 2014.

[59] K.-S. Hwang, W.-C. Jiang, and Y.-J. Chen, "Model learning and knowledge sharing for a multiagent system with dyna-q learning", *IEEE transactions on cybernetics*, vol. 45, no. 5, pp. 978–990, 2015.

[60] M. R. Davoodi, K. Khorasani, H. A. Talebi, and H. R. Momeni, "Distributed fault detection and isolation filter design for a network of heterogeneous multiagent systems", *IEEE Transactions on Control Systems Technology*, Vol. 22, No. 3, pp.1061–1069, 2014.

[61] A. Zidan, M. Khairalla, A. M. Abdrabou, T. Khalifa, K. Shaban, A. Abdrabou, R. El Shatshat, AND A. M. Gaouda, "Fault detection, isolation, and service restoration in distribution systems: State-of-the-art and future trends", *IEEE Transactions on Smart Grid*,Vol.99, pp.1-16, 2016.

[62] X. Liu, X. Gao, J. Han, "Robust unknown input observer based fault detection for high-order multi-agent systems with disturbances", *ISA transactions*, vol. 61, pp. 15–28, 2016.

[63] M. R. Davoodi, N. Meskin, and K. Khorasani, "Simultaneous fault detection and control design for a network of multi-agent systems", *in Control Conference (ECC), 2014 European. IEEE,* , pp.575–581.2014.

[64] N. K. Krothapalli and A. V. Deshmukh, "Distributed task allocation in multi-agent systems", *In Proceedings of the Institute of Industrial Engineers Annual Conference*, 2002.

[65] Y. Jiang , Z. Li,"Locality-sensitive task allocation and load balancing in networked multiagent systems: Talent versus centrality" , *Journal of Parallel and Distributed Computing*, Vol. 71, No.6, pp.822–836, 2011.

[66] C. Lin, Z. Lin, R. Zheng, G. Yan, and G. Mao, "Distributed source localization of multi-agent systems with bearing angle measurements", *IEEE Transactions on Automatic Control*,Vol. 61, No.4, pp.1105–1110, 2016.

[67] B. Horling , V. Lesser, "A survey of multi-agent organizational paradigms",*The Knowledge Engineering Review*, Vol.19, No.4, pp.281–316, 2004.

[68] A. Esmaeili, N. Mozayani, M. R. J. Motlagh, AND E. T. Matson, " The impact of diversity on performance of holonic multi-agent systems",*Engineering Applications of Artificial Intelligence*, Vol. 55, pp. 186–201,2016.

[69] C. H. Brooks , E. H. Durfee, "Congregation formation in multiagent systems", *Autonomous Agents and Multi-Agent Systems*, Vol.7, No.1, pp.145–170, Jul 2003.

[70] M.Wooldridge,"An introduction to multiagent systems", *2nd Edition, John Wiley & Sons Ltd, Chichester*, 2009.

[71] M.N. Satymbekov, I.T. Pak, L. Naizabayeva, AND Ch.A. Nurzhanov, "Multi-agent grid system Agent-GRID with dynamic load balancing of cluster nodes", *Open Engineering*, Vol.7, No.1,pp.485-490, December 2017.

[72] S.Banerjee , J.P. Hecker,"Multi-Agent System Approach to Load-Balancing and Resource Allocation for Distributed Computing", *First Complex Systems Digital Campus World E-Conference* , 2015 .

[73] Rina Suros, Juan Francisco Serrano,"Communication complexity in high-speed distributed computer network in an agent based architecture for grids service Ray Tracing View project", *International Journal of Advanced Computer Research*, Vol.8,No.35,pp.2249-7277, 22-February-2018

[74] HAJOUI YOUNES et al., "New load balancing Framework based on mobile AGENT and ant-colony optimization technique", *Conference Intelligent Systems and Computer Vision (ISCV)*, At Fès,2017.

[75] Rathore, N,"Efficient Agent Based Priority Scheduling and Load Balancing Using Fuzzy Logic in Grid Computing", *i-manager's Journal on Computer Science*, Vol.3,No.3, pp.11-22, 2015.

[76] B.Yagoubi ,Y. Slimani,"Job Load Balancing Strategy for Grid Computing",*Journal of Computer Science*,Vol.3,No.3,pp.186-194, 2007.

[77] D.Klusáček , H.Rudová, "Alea 2 job scheduling simulator", *In Proceedings of the 3rd International Conference on Simulation Tools and Techniques (SIMUTools 2010*), Torremolinos, Malaga, Spain.2010.

[78] R.Buyya , M.Murshed, "Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing", *The Journal of Concurrency and Computation: Practice and Experience (CCPE)*,Vol.14, pp.13–15.2002

[79] MetaCentrum Czech National Grid ,WebSite: http://www.cs.huji.ac.il/labs/parallel/workload/l_metacentrum

[80] I.Foster , Y.Zhao , I.Raicu AND S.Lu," Cloud computing and grid computing 360-degree compared", *In Grid Computing Environments Workshop, GCE'08*, pp. 1-10,November 2008.

[81] F. J. Seinstra, J. Maassen et al."Jungle Computing: Distributed Supercomputing beyond Clusters, Grids, and Clouds", *Technical report* , Department of Computer Science, Vrije Universiteit, De Boelelaan 1081A, 1081 HV Amsterdam, The Netherlands, pp.1-31, 2010.

[82] B.Yagoubi B, "Equilibrage de charge dans les grilles de calcul", *PhD Thesis*, University of Oran. Novembre 2007.

[83] D. England , J. Weissman. "Costs and benets of load sharing in the computational grid", *In Proceedings of the 10th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 2004)*, pp.160-175, New York, NY, USA, June 2004.

[84] A. Giersch,"Ordonnancement sur plates-formes hétérogènes de tâches partageant des données", *PhD thesis*, University of Louis Pasteur, Strasbourg, Décembre 2004.

[85] X. Liu, H. Xia, and A. Chien. "Validating and scaling the MicroGrid: A scientic instrument for Grid dynamics", *Journal of Grid Computing*,Vol.2,No.2 , pp.141-161, 2004.

[86] A. Takefusa, S. Matsuoka, K. Aida, H. Nakada, and U. Nagashima."Overview of a performance evaluation system for global computing scheduling algorithms", *In Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing*, pp. 11. IEEE, 1999.

[87] W. M. Jones, W. B. Ligon, III, L. W. Pang, and D. Stanzione, " Characterization of bandwidth-aware meta-schedulers for co-allocating jobs across multiple clusters, *The Journal of Supercomputing*, Vol.34,No.2, pp.135-163, 2005.

[88] A. Legrand, L. Marchal, and H. Casanova, "Scheduling distributed applications: the SimGrid simulation framework, *In Proceedings of the third International Symposium on Cluster Computing and the Grid*, pp.138-145 , 2003.

[89] Y. Caniou and J. S. Gay, " Simbatch: An API for simulating and predicting the performance of parallel resources managed by batch systems", *In Euro-Par 2008 Workshops - Parallel processing*, vol.5415 of LNCS, pp. 223-234. Springer, 2009.

[90] J. C. de Kergommeaux, B. de Oliveira Stein, and B. P.E. Pajé," an interactive visualization tool for tuning multi-threaded parallel applications", *Parallel Computing*, Vol.26,No.10,pp.1253-1274, 2000.

[91] K. Coulomb, M. Faverge, J. Jazeix, O. Lagrasse, J. Marcoueille, P. Noisette, A. Redondy, and C. Vuchener,"Visual trace explorer (ViTE), October 2009. *WebSite: http://vite.gforge.inria.fr/.*

[92] D. Kondo," SimBOINC: A simulator for desktop Grids and volunteer computing systems", October 2009, *WebSite: http://simboinc.gforge.inria.fr/.*

[93] P. Niemeyer , J. Knudsen,"Learning Java", *O'Reilly Media*, third edition, 2005.

[94] C. Dobre, F. Pop, and V. Cristea," A simulation framework for dependable distributed systems", *In Proceedings of the 2008 International Conference on Parallel Processing - Workshops*, pp.181-187 , 2008.

[95] I. C. Legrand and H. B. Newman,"The MONARC toolset for simulating large network-distributed processing systems", *In Proceedings of the 32nd conference on Winter simulation*, pp.1794-1801. Society for Computer Simulation International, 2000.

[96] F. Howell and R. McNab. Simjava,"A discrete event simulation library for Java", *In International Conference on Web-Based Modeling and Simulation*,pp.51-56. Society for Computer Simulation International (SCS), 1998.

[97] K. Kurowski, J. Nabrzyski, A. Oleksiak, and J. Weglarz," Grid scheduling simulations with GSSIM", *In ICPADS '07: Proceedings of the 13th International Conference on Parallel and Distributed Systems* , Vol.2, pages 1-8. IEEE, 2007.

[98] Bellifemine F., Caire G., Greenwood D," Developing Multi-Agent Systems with JADE", *John Wiley & Sons*, England , ISBN: 978–0–470–05747–6 (HB),2007.