

ACADÉMIE DE MONTPELLIER
UNIVERSITÉ MONTPELLIER II
— SCIENCES ET TECHNIQUES DU LANGUEDOC —

Mémoire de Stage de Master

SPÉCIALITÉ : Recherche en Informatique
Mention : Informatique, Mathématiques, Statistiques

effectué au laboratoire LIRMM/INFO

—
sous la direction de
MR JACQUES FERBER
en partenariat avec l'entreprise OSLO (Lyon)

**Modèle organisationnel de répartition de
charges pour des agents dans des contextes
contraints. Application à la résolution de
problèmes et à la simulation multi-agent
distribuée sur le GRID**

par

Arnaud Auzolat

Soutenu le 29 juin 2006

Remerciements

Je tiens à remercier Jacques Ferber, mon directeur de projet, qui m'a aidé et encadré durant ce stage. Je le remercie de m'avoir proposé ce sujet et de m'avoir encadré durant ces cinq mois tout en me laissant une grande liberté dans mes recherches. Son soutien m'a permis d'accomplir ce stage en toute sérénité et l'approche des Systèmes Multi-Agent qu'il m'a présenté m'a définitivement convaincu de poursuivre dans cette branche de l'informatique.

Je remercie également Noël Perez, responsable de la section Recherche & Développement de la société OSLO, ainsi que toute son équipe, qui m'ont permis de pouvoir observer le fonctionnement interne d'une entreprise et ainsi pouvoir comparer le monde du travail et le monde de la recherche.

Je remercie enfin, ma petite amie, ma famille ainsi que mes amis qui m'ont énormément soutenu tout au long de ce stage.

Table des matières

Introduction	11
1 Problématique et objectifs du stage	13
1.1 Sujet du stage	13
1.2 Résumé du contexte du stage	14
2 Etat de l'art sur la répartition de charges	15
2.1 Les clusters avec répartition des charges	15
2.2 Différentes techniques employées par le passé	17
2.2.1 Le clonage d'agents [1]	17
2.2.2 La migration d'agents [2]	19
2.2.3 Un modèle particulier : DRUM [3]	20
2.2.4 Algorithme "Branch & Bound" [4]	23
2.2.5 Répartition des charges avec adversaire [5]	25
2.2.6 Le problème de l'Agent Voyageant [6]	26
2.2.7 Analyse	27
3 Solution proposée	27
3.1 Motivations et positionnement	27
3.2 Structure du modèle	28
3.2.1 Les Agents	28
3.2.2 Les Groupes	29
3.2.3 Les Rôles	30
3.2.4 L'Environnement	30
3.3 Modèle du système	30
3.3.1 Un exemple de modèle	33
3.3.2 Un exemple avec communications	33
3.4 Le Seuil et ses emplois	35
3.4.1 Le seuil <i>fixe</i>	35
3.4.2 Le seuil <i>différentiel</i>	36
3.4.3 Le seuil <i>différentiel variable</i>	37
3.4.4 Analyse	37
4 Simulations	38
4.1 MadKit	38
4.2 Application pour notre étude	38
4.2.1 Algorithme de répartition	39
4.2.2 Simulations avec seuil fixe	43
4.2.3 Simulations avec seuil différentiel	44

4.2.4	Simulations avec seuil différentiel variable	47
4.2.5	Exemples du phénomène de répartition	51
4.3	Evaluation des expérimentations	53
5	Discussions et perspectives	54
6	Annexes	57
6.1	Capture d'écran 1	57
6.2	Capture d'écran 2	59

Table des figures

1	Aperçu d'un réseau de groupes	32
2	Réseau de groupes avec communications	33
3	Communications avant répartition	34
4	Communications après répartition	34
5	Code pour l'algorithme de répartition	42
6	Etat du réseau après répartition à seuil fixe	44
7	Amélioration de la répartition (seuil fixe)	45
8	Etat du réseau après répartition à seuil différentiel	46
9	Amélioration de la répartition (seuil différentiel)	46
10	Présentation du réseau avec un seuil différentiel variable	48
11	Etat du réseau à seuil différentiel variable à l'instant 0	48
12	Etat du réseau à seuil différentiel variable à l'instant t	49
13	Etat du réseau à seuil différentiel variable à l'instant $t+1$	50
14	Etat du réseau à seuil différentiel variable à l'instant $t+2$	50
15	Processus de répartition sans création de groupes	51
16	Processus de répartition avec création de groupes	52
17	Réseau à 100 groupes et 200 agents	58
18	Après Analyse (100G, 200A)	58
19	Réseau à 50 groupes et 200 agents	60
20	Après Analyse (50G, 200A)	60

List of Algorithms

1	Balancing algorithm	41
2	Function Analyse()	41
3	Function Repart()	41

Abstract

Ce document traite de la répartition des charges dans des Systèmes Multi-Agents à travers un modèle organisationnel. Il présente un méta-modèle générique des systèmes multi-agents basés sur des concepts organisationnels tels que les groupes, les rôles et les structures.

La répartition des charges tient une place importante dans l'informatique depuis de nombreuses années. L'utilisation efficace des calculs en parallèle nécessite que le calcul soit divisé entre les différents processeurs. Ce problème de répartition de charges apparaît dans de nombreuses études et tient une place prépondérante dans le secteur de la recherche depuis les dix dernières années et même plus. Dans le but d'améliorer les performances du système, nous proposons une nouvelle politique de répartition de charges basée sur l'emploi d'un seuil. Cette notion de seuil va permettre de faciliter la répartition et permettra de résoudre des problèmes en accord avec les agents. Une simulation a été développée pour étudier les performances du système avec la politique de répartition des charges proposée. Ce système a été implémenté sous la plateforme MadKit [19] mise en place par J. Ferber.

Mots clés : répartition des charges dynamiques, agent, Système Multi-Agent, seuil, MadKit

This paper discusses load balancing in Multi-Agent Systems (MAS) through an organisational model. It presents a meta-model of multi-agent systems based on organisational concepts such as groups, roles and structures. Load balancing takes a very important place in computer science for several years. Effective use of a parallel computer requires that a calculation be carefully divided among the processors. This load balancing problem appears in many guises and has been a fervent area of research for the past decade or more. In order to improve the system performance, we propose a novel threshold-based load balancing policy. This notion of threshold helps the load balancing and allows to solve problems in accordance with agents. A discrete event computer simulator is developed to study the system performance with the proposed load balancing policy. This system has been implemented under the MadKit platform created by J. Ferber.

Key words : dynamic load balancing, agent, Multi-Agent Systems, threshold, MadKit

Introduction

Depuis de nombreuses années, les chercheurs en informatique tentent d'améliorer les performances des ordinateurs et autres outils informatiques que l'on utilise tous les jours. Pour cela, le partitionnement des données et la répartition des charges ont été mis en place en tant que composants importants du calcul en parallèle.

Dans cette étude, nous nous intéresserons uniquement à la répartition des charges (load balancing) ou attribution du travail (tasks allocation) pour les agents. Elle est critique en simulation en parallèle. Elle maximise la performance des applications en gardant aussi bas que possible le temps des agents inactifs et minimise la communication entre ces agents. Dans des applications avec une charge de travail constante, une répartition des charges **statique** peut être utilisée comme un pré-processeur pour le calcul.

D'autres applications, telles que les méthodes adaptatives à éléments finis, ont des charges de travail imprévisibles ou qui changent durant le calcul; de telles applications ont besoin de répartiteurs de charge **dynamiques** qui ajustent la décomposition lorsque le calcul se déroule. La plupart des problèmes de stratégie de répartition de charges arrivent au début et à la fin du calcul distribué, en effet le réseau est généralement très peu chargé durant ces phases.

Depuis maintenant plus de vingt ans, l'idée de faire migrer des processus n'est pas nouvelle. Néanmoins, celle de faire migrer des agents autonomes l'est beaucoup plus. En effet, les systèmes déjà employés pour faire migrer des processus utilisent une technique assez simple : ils capturent l'état du processus, le transfèrent à une autre machine, et fournissent une transparence pour tous les accès aux systèmes.

Les systèmes à agents seront notre priorité et nous utiliserons le modèle **AGR** (Agent/Groupe/Rôle), proposé par J. Ferber [13], qui décompose un problème en termes de groupes et de rôles. De plus, la répartition des charges sera fondée sur des techniques de distribution "peer-to-peer" (P2P) qui sont des systèmes distribués basés sur le concept de partage de ressources par des échanges directs entre noeuds "peer" qui seront pour nous des noeuds "agents" ayant le même rôle et des responsabilités équivalentes. Afin de pouvoir correspondre aux caractéristiques des systèmes P2P, un changement de paradigme est nécessaire et requiert une organisation autonome, une adaptation et une résistance comme propriétés fondamentales. Les SMA (Systèmes Multi-Agents) peuvent être une base d'un nouveau paradigme de programmation pour les applications P2P. Dans la structure SMA, le système est décrit par un grand nombre de relativement simples et autonomes unités de

calcul, **les Agents**.

Une des questions importantes à se poser est "pourquoi utiliser des agents?".

Il existe plusieurs arguments permettant cette utilisation :

- parce que les applications à base des agents fonctionnent tout comme un groupe de personnes, ils peuvent assurer la prise de décision quotidienne,
- les agents peuvent prendre d'excellentes décisions très rapidement,
- les agents prospèrent dans des environnements volatils ou conflictuels, où les systèmes traditionnels ont des difficultés,
- les agents résolvent des problèmes "impossibles",
- les agents peuvent traiter les niveaux très élevés de complexité,
- les agents sont scalables et peuvent travailler avec un grand nombre de sources de données et des informations détaillées,
- les agents tiennent compte des préférences d'autres agents et des préférences de l'organisation entière pour prendre la meilleure décision pour l'attribution des ressources.

Ce qui rend les SMA particulièrement attirant par rapport au P2P est le fait que les propriétés comme l'adaptation, l'organisation autonome et la résistance sont réalisées sans les insérer dans les agents individuels.

De plus, les systèmes multi-agents fonctionnent comme des marchés virtuels où des ressources d'entreprise peuvent être échangées et/ou assignées aux différentes tâches à réaliser. Chaque ressource de l'entreprise est représentée sur le marché virtuel par un agent unique. La caractéristique principale d'une application à base d'agents est que cela fonctionne tout comme un groupe de personnes négociant des ressources entre eux.

Puisque les agents prennent des décisions basées sur les connaissances d'entreprise, et non sur des algorithmes prédéterminés, ils sont considérés comme intelligents et donc ils prennent les meilleures décisions, adaptées à l'entreprise.

1 Problématique et objectifs du stage

L'étude de répartition des charges pour des applications diverses n'est pas nouvelle dans le monde de l'informatique. L'innovation au niveau de notre étude réside dans l'utilisation d'agents pour résoudre ces nombreux problèmes. Le fait de mettre en place un nouveau modèle organisationnel de répartition de charges pour les systèmes multi-agents va permettre de prendre en compte plus facilement les demandes des modélisateurs et/ou utilisateurs, détecter automatiquement les ressources de calcul tout en utilisant des techniques réactives (phéromones, champs de potentiels) et organisationnelles (groupes, rôles) et permettre une mobilité des agents plus efficace. Dans la suite de notre travail, nous ne nous intéressons qu'à la répartition des charges dans des contextes à agents.

Dans cette partie, nous présentons dans un premier temps le sujet du stage. Ensuite, nous récapitulons le contexte de notre travail.

1.1 Sujet du stage

Ce stage de Master 2 Recherche a été effectué au sein du Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier (**LIRMM**) ainsi qu'en collaboration avec la société **OSLO** basée sur Lyon.

Le but de l'étude est de définir un modèle organisationnel multi-agent permettant la répartition de charges pour des applications de type simulation multi-agent ou résolution de problèmes à l'aide d'agents. Les techniques utilisées afin de réaliser cela ont été des techniques de répartition de charges dynamiques fondées sur des modèles à la fois réactifs (champs de potentiels, phéromones) et organisationnels (rôles, groupes) pour des systèmes eux-mêmes organisationnels (modèle AGR étendu).

De plus, ces techniques de répartition de charges doivent être adaptatives en fonction de l'évolution des demandes et des ressources, et seront fondées sur des techniques de distribution "peer-to-peer".

Ces agents sont amenés à fonctionner sur des grilles de calcul (GRID) et à collaborer avec des êtres humains. Les applications retenues sont la simulation massive (grand nombre de simulations, simulations distribuées) et la résolution de problèmes à l'aide de systèmes multi-agents réactifs.

1.2 Résumé du contexte du stage

La notion de "grid computing" (ou de "grille de calcul") devient de plus en plus importante. La possibilité de rendre l'utilisation de ressources de calcul transparentes aux utilisateurs amène, pour les agents, de nouveaux enjeux. Les mécanismes de répartition de charge consistent à répartir des agents ou des processus sur un ensemble d'ordinateurs, de manière à pouvoir utiliser ces ressources.

Pour réaliser un modèle de grille dans le domaine de la simulation ou de la résolution de problème multi-agents, les modèles de grille classiques ne fonctionnent pas, car ils supposent une décomposition en tâches indépendantes qui retournent un résultat une fois terminé.

Malheureusement ce modèle ne convient pas aux simulations distribuées car :

1. il y a une très forte interaction entre les agents,
2. il existe souvent un environnement qui introduit une certaine globalité dans l'ensemble,
3. l'ensemble des agents présente souvent une forte hétérogénéité,
4. les utilisateurs peuvent être amenés à intervenir dans les simulations (agents "humains", jeu d'entreprise, etc..)
5. les modélisateurs peuvent avoir des demandes en termes de priorité vis-à-vis de calculs, ou en termes de précision de résultats, ce qui introduit des contraintes dans l'ensemble du système.

D'autre part, des modèles d'organisation ont vu récemment le jour, et notamment le modèle AGR qui décompose un problème en termes de groupes et de rôles. Ce modèle a été récemment étendu par le modèle AGRE (Agent/ Groupe/ Rôle/ Environnement) qui permet d'introduire la notion d'espace physique dans le modèle AGR.

Ce modèle introduit un certain nombre de contraintes en termes d'interaction (les groupes sont avant tout des espaces d'interaction entre ses membres) mais ouvre aussi un certain nombre de perspectives en donnant la possibilité de gérer les problèmes de répartition de charge au niveau des organisations (introduction de méta-groupes et méta-rôles dont la fonction consiste à gérer la répartition de charges en fonction de certains critères).

2 Etat de l'art sur la répartition de charges

Comme cité précédemment, l'étude de répartition des charges, ou load balancing (voir aussi tasks allocation), n'est pas nouvelle. Depuis de nombreuses années, de nombreux chercheurs se sont penchés sur ce problème et la répartition des charges tient une place prépondérante dans l'étude des Systèmes Multi-Agents (SMA).

La migration de processus consiste à prendre un aperçu de l'état du processus, le transférer vers une autre machine, et fournir une transparence pour tous les accès systèmes.

La nouveauté dans cette étude concerne l'emploi de systèmes à agents. Les agents migrent afin d'accéder aux ressources locales et la transparence ne tient plus une place prépondérante. Bien que la transparence soit une particularité intéressante pour les abstractions à hauts niveaux telles que les automates ou les systèmes de répartition de charges transparentes, la migration transparente n'est plus un but critique, voir immédiat pour les systèmes à agents. Plus encore, en fait, la migration transparente semble pouvoir être bâtie sur des infrastructures de migration d'agents.

Les services de répartition des charges représentent une des "briques" essentielles pour les solutions hautes performances : optimisation du flux d'informations et amélioration de la fiabilité et des temps de réponse.

La qualité exacte d'un service fournit par un serveur web aux utilisateurs, à la fin, dépend typiquement de deux paramètres :

1. la vitesse de transfert du réseau – importance de la bande passante,
2. le temps de réponse du serveur – importance des ressources (ordinateur rapide(CPU fast), beaucoup de mémoire (RAM) et bonne performance I/O).

2.1 Les clusters avec répartition des charges

La répartition de charge permet de faire évoluer les performances des programmes sur serveur, en répartissant les requêtes des clients sur plusieurs serveurs. Les répartiteurs de charge, terme couramment utilisé pour désigner les technologies de répartition de charge, reçoivent les requêtes des différents clients, voire à plusieurs requêtes du même client.

Ainsi, un navigateur Web peut obtenir les différentes images d'une même page Web à partir de différents hôtes du cluster. Une telle structure permet de répartir la charge, d'accélérer le traitement et de réduire les temps de

réponse pour les clients.

Les répartiteurs de charge utilisent différents algorithmes pour gérer le trafic. Leur but est de répartir intelligemment la charge et/ou d'optimiser l'utilisation de l'ensemble des serveurs du cluster. Voici quelques exemples de ces algorithmes :

Round-robin (répartition de charge équitable).

Un algorithme Round-robin répartit équitablement la charge entre chaque serveur, quels que soient le nombre actuel de connexions ou les temps de réponse. Cet algorithme est adapté si les serveurs du cluster disposent des mêmes capacités de traitement ; sinon, certains serveurs risquent de recevoir plus de requêtes qu'ils ne peuvent en traiter tandis que d'autres n'utiliseront qu'une partie de leurs ressources.

Weighted Round-robin (répartition de charge pondérée). Un algorithme Weighted Round-robin prend en compte la capacité de traitement de chaque serveur. Les administrateurs affectent manuellement un coefficient de performance à chaque serveur et une séquence d'ordonnancement est générée automatiquement en fonction de cette valeur. Les demandes sont ensuite affectées aux différents serveurs selon une séquence de répétition alternée.

Least-connection (répartition de charge selon la moindre connexion). L'algorithme Least-connection envoie des demandes aux serveurs d'un cluster en fonction de celui qui sert actuellement le moins de connexions.

Load-based (répartition basée sur la charge). Un algorithme Load-based envoie les demandes aux serveurs d'un cluster en fonction de celui dont la charge est actuellement la plus faible.

De plus, certains répartiteurs intègrent une détection des défaillances. Le répartiteur de charge assure un suivi du serveur ou de l'application qui s'y exécute, et cesse de lui envoyer des requêtes en cas de défaillance.

Lorsque le répartiteur choisit une requête du client, l'un des serveurs du groupe la traite. Chaque serveur est capable de traiter cette requête de façon indépendante. Si un serveur est indisponible par suite d'une erreur ou d'une intervention de maintenance, les autres peuvent continuer à traiter les requêtes sans en être affectés. De ce fait, la disponibilité générale du service est bien meilleure que lorsqu'un seul serveur traite l'ensemble des demandes.

Cependant, l'utilisation d'un seul répartiteur de charge physique ou d'un seul commutateur réseau devant un ensemble de serveurs employant un système logiciel de répartition de charge introduit un autre point de défaillance unique. Pour pallier ce risque, on peut faire appel à des dispositifs de répartition de charge et/ou des commutateurs redondants.

Contexte final

Le modèle avec des clusters de répartition des charges (*Load-Balanced Cluster*) présente les avantages et les inconvénients suivants :

Avantages

Evolutivité renforcée – Un niveau de répartition de charge évolutif permet de conserver des performances acceptables et de renforcer la disponibilité.

Disponibilité accrue – Dans un système avec répartition de charge, on peut mettre hors ligne un serveur pour assurer sa maintenance, sans rendre des applications indisponibles.

Economies potentielles – L'utilisation de plusieurs serveurs peu onéreux s'avère plus économique que le recours à des systèmes multiprocesseurs au coût élevé.

Inconvénients

Complexité du développement – Dans les situations où il est nécessaire de conserver l'état pour chaque transaction ou chaque utilisateur, le développement d'une solution avec répartition de charge peut être complexe.

Défaillances du réseau non prises en compte – En cas de défaillances d'un serveur ou du réseau pendant une session client, il peut être nécessaire de procéder à une nouvelle ouverture de session, afin d'authentifier à nouveau le client et de rétablir l'état de la session.

2.2 Différentes techniques employées par le passé

2.2.1 Le clonage d'agents [1]

Tout d'abord, observons en tant que première approche le **clonage** d'agents. Dans cette étude, les auteurs décrivent comment l'emploi du "clone" dans des systèmes multi agents peut apporter une solution aux problèmes de répartition de charges.

La nature distribuée du système et l'autonomie des agents contribuent à la

capacité de surmonter les problèmes de surcharge.

Dans cette structure, un agent est une entité de calcul autonome, consciente, intelligente et pro active. Un agent fournit des services en réalisant des tâches qu'ils génèrent lui-même ou qu'on lui a délégué (utilisateurs ou autres agents). Une tâche est soit un code exécutable, soit un but représenté à un haut niveau d'abstraction. Dans le dernier cas, les agents peuvent être capables de transformer le but abstrait en tâches plus concrètes. Ces tâches concrètes sont examinées par les agents pour vérifier si leur performance (des tâches) fait parti de ces aptitudes (agent). Si elles le sont, soit l'agent les réalise soit il les décompose en sous tâches. Autrement, les tâches sont allouées à des agents appropriés.

Les agents ont des aptitudes qui indiquent le type des tâches qu'ils peuvent réaliser, et des capacités qui indiquent le coût des ressources auxquelles l'agent peut accéder et utiliser pour exécuter les tâches. Les tâches sont catégorisées en type pouvant être manipulées uniquement par des agents ayant les aptitudes de le faire.

Le problème traité dans ce papier est celui où le déplacement de la tâche vers l'agent va le surcharger.

Il existe deux sortes de surcharge :

1. Un agent dans un SMA est surchargé, mais le SMA tout entier a les aptitudes et les capacités requises.
2. Le SMA tout entier est surchargé, i.e., les agents compris dans le SMA n'ont pas les aptitudes requises (cependant, cela peut être des ressources inactives dans le système de calcul où se trouve les agents).

Le résultat d'une telle surcharge montre que le SMA ne réalisera pas toutes les tâches à temps, bien que les ressources requises soient disponibles. Donc les solutions proposées sont les suivantes :

1. Premier cas – les agents surchargés passeront les tâches à d'autres agents qui auront les aptitudes et les capacités à les réaliser,
2. Deuxième cas – les agents surchargés créent de nouveaux agents pour réaliser les tâches en excès et utiliseront les ressources inutilisées ou migreront vers d'autres hôtes.

Le clonage d'agents est présenté pour implémenter ces solutions. La migration de l'agent peut être implémentée en créant un clone sur une machine distante, en transférant les tâches de l'agent original vers le clone, et ensuite il meurt. Ainsi, la mobilité de l'agent est une instance du clonage d'agent.

Rapidement, dans cette infrastructure, les agents sont de trois types :

1. *providers* : possédant certaines aptitudes,
2. *requesters* : ayant des tâches à réaliser et localisant les agents ayant les aptitudes requises,
3. *middle agents* : c'est par eux que les requesters trouvent les providers.

Lorsqu'un agent perçoit une surcharge, un agent cherche un provider, en passant par les middle agent, pour lui transférer les tâches. S'il n'en trouve pas, un clone est créé, qui sera connu de tous.

2.2.2 La migration d'agents [2]

Dans cet article, les auteurs nous présentent "comment migrer des agents". Les systèmes de migration de programme changent dans leur puissance expressive, d'après les auteurs. Ils classifient les systèmes à migration selon l'abstraction de migration qu'ils fournissent au programmeur de l'application :

1. *full-migration* : tout l'état du programme lancé est migré. Ceci inclut les piles d'appel thread, les espaces de nom gérés par le kernel tels que les descripteurs I/O, les noms de fichiers système,
2. *thread-state migration* : l'état interne du programme est migré, i.e., les variables et les piles d'appel. Les espaces de nom externe peuvent être modifiés.
3. *code-only migration* : le programme est relancé à partir d'un site distant, en initialisant des données dans un message fournit par le lancement précédent du programme. Le programmeur doit explicitement écrire le code pour enregistrer tous résultats partiels dans les messages d'initialisation, ainsi la nouvelle incarnation du programme n'a pas à répéter le travail.

Les systèmes à agents sont sans aucun doute mieux servis par la *thread-state migration* puisque l'explicité de l'endroit d'exécution est souvent un but de ces systèmes. Du coup, la transparence, tout en respectant l'état interne du programme, permet d'écrire les agents plus facilement.

La *code-only migration*, quant à elle, fournit seulement les aptitudes minimales nécessaires pour implémenter les agents.

Les auteurs montrent également comment la capture de l'état - en extrayant récursivement les "suites locales" de toutes les fonctions - peut être atteinte par de simples transformations syntaxiques au niveau du code source

de l'agent. Des transformations similaires peuvent être appliquées au niveau objet du code pour réaliser les mêmes buts. D'après les auteurs, cette technique n'est pas restreintes à Java et peut être appliquée à des programmes écrits dans d'autres langages. L'indépendance de la plateforme Java et les caractéristiques de chargement incrémentales font qu'elle est bien adaptée pour la programmation orientée objet, et le système de migration d'agents qu'ils ont mis en place est basé sur l'application de leurs techniques à Java.

2.2.3 Un modèle particulier : DRUM [3]

Le problème traité ici correspond au partitionnement et à la répartition des charges dynamiques sur des clusters à ressources logicielles hétérogènes. Les auteurs de cet article présentent le modèle DRUM (Dynamic Resource Utilization Model) qu'ils ont mis en place afin de résoudre le problème précédent.

Ce modèle (sous forme d'arbre), utilisé pour l'environnement d'exécution, a été créé sur une initialisation en utilisant un fichier de configuration au format XML qui contient une liste des noeuds et la description de leur topologie d'interconnexion.

Ce fichier de configuration peut être créé manuellement ou bien avec un outil de configuration graphique.

Cet outil va fournir différentes aptitudes incluant :

1. la spécification des noeuds du cluster et leurs caractéristiques,
2. la topologie du réseau,
3. les aptitudes, la spécification des procédures de répartition des charges,
4. les paramètres pour une répartition hiérarchique,
5. l'évaluation initiale de l'aptitude des noeuds en lançant des points de référence distribués,
6. les facilités à contrôler les disponibilités des capacités de gestion de réseau tel que SNMP (Simple Network Management Protocol)

Le système devra être relancé en cas de modifications de celui-ci.

Durant le déroulement d'un calcul, la disponibilité des ressources peut changer dramatiquement, particulièrement dans les environnements partagés. Dans ces environnements dynamiques, les aptitudes d'évaluation de DRUM peuvent être mises à jour par des *agents* : des threads qui fonctionnent en même temps que l'application pour surveiller chaque noeud.

Certaines fonctions permettent d'engendrer des agents threads. Des agents

de contrôle sont également mis en place, chargés de contrôler le trafic de communication, les charges du CPU et l'utilisation de la mémoire.

La puissance d'un noeud

Le modèle distille l'information vers une valeur de puissance pour chaque noeud, un seul nombre indiquant la portion de charge totale qui devrait être assigné à ce noeud.

La puissance de chaque noeud dépend de la capacité de traitement et de la puissance de communication.

$$power_n = w_n^{comm} * c_n + w_n^{cpu} * p_n \quad (1)$$

$$w_n^{comm} + w_n^{cpu} = 1 \quad (2)$$

La capacité de traitement

Pour un noeud de calcul n avec m CPUs sur lesquels k_n processus d'applications sont lancés, la capacité de traitement $p_{n,j}$ pour chaque processeur j , $j = 1, 2, \dots, k_n$ sur un noeud n est :

1. l'utilisation du CPU $u_{n,j}$ par processus j ,
2. la fraction de temps i_t où le CPU t est inactif,
3. et le taux de repère statique d'un noeud (en MFLOTS) b_n

Le temps total d'inactivité d'un noeud n est $\sum_{t=1}^m i_t$.

Cependant, lorsque $k_n < m$, les k_n processeurs peuvent seulement utiliser les k_n processeurs à tout moment, donc le temps d'inactivité total exploitable au maximum est $k_n - \sum_{j=1}^{k_n} u_{n,j}$. Ainsi, le temps d'inactivité total que les k_n processus peuvent exploiter est $\min(k_n - \sum_{j=1}^{k_n} u_{n,j}, \sum_{t=1}^m i_t)$.

Tant que le scheduler du système d'exploitation du CPU peut être attendu à donner à chacun k_n processus la même portion de temps sur les noeuds du CPU, on assigne la même capacité à tous les processus d'un noeud n .

On calcule l'utilisation moyenne d'un CPU et les temps d'inactivité par processus :

$$u_n = \frac{1}{k_n} * \sum_{j=1}^{k_n} u_{n,j} \quad (3)$$

$$i_n = \frac{1}{k_n} \min(k_n - \sum_{j=1}^{k_n} u_{n,j}, \sum_{t=1}^m i_t) \quad (4)$$

$$(5)$$

La capacité de traitement $p_{n,j}$ est estimé comme étant :

$$p_{n,j} = b_n(u_n + i_n), j = 1, 2, \dots, k_n \quad (6)$$

Tant que $p_{n,j}$ est la même pour tous les processus j d'un noeud n :

$$p_n = \sum_{j=1}^{k_n} p_{n,j} = k_n p_{n,1} \quad (7)$$

Pour les noeuds internes, p_n est la somme des capacités de traitement des fils des noeuds.

La puissance de communication

La puissance de communication d'un noeud est basée sur la communication d'un noeud. A chaque calcul et à chaque noeud du réseau, les agents de contrôle de DRUM calculent le taux de paquets entrants λ et de paquets sortants μ pour chaque interface de communication appropriée.

Pour chaque interface i , on calcule la largeur de bande disponible (ABW) durant un temps T telle que :

$$A_i(t, T) = \frac{1}{T} \int_t^{T+t} (C_i - (\lambda_i(t) + \mu_i(t))) dt, \quad (8)$$

où C_i est la largeur de bande maximale du lien.

Pour un noeud n avec s interfaces, on estime la puissance de communication comme :

$$c_n = \frac{1}{s} \sum_{i=1}^s A_i(t, T) \quad (9)$$

Au final, les expérimentations de ces chercheurs démontrent que l'emploi de leur modèle permet une meilleure répartition des charges pour des ressources "averties".

De plus, les réels avantages de la puissance de communication peuvent seulement être vus lorsque les ressources des réseaux sont non uniformes ou une partie du réseau est chargé lourdement.

Les agents DRUM contrôlent la mémoire disponible et la mémoire totale pour chaque noeud de calcul. De ce fait, l'utilisation de la mémoire devrait être un facteur pour le calcul de la puissance de noeud seulement lorsque le taux de mémoire disponible par rapport à la mémoire totale est plus petit qu'un seuil donné.

2.2.4 Algorithme "Branch & Bound" [4]

L'article traite d'une stratégie simple qui permet de trouver une solution optimale de manière forte, i.e., qui calcule la solution optimale en examinant tous les éléments de l'espace des solutions. Tant que l'espace des solutions augmentent exponentiellement en fonction de la taille des entrées, cette méthode tend vers des temps de calcul énormes.

Les algorithmes dits "branch & bound" sont des techniques algorithmiques pour résoudre ce type de problème. Il réalise une recherche à travers l'espace des solutions, mais utilise une connaissance heuristique des solutions pour couper l'arbre de recherche en morceaux.

Afin de décrire une stratégie de répartition des charges, il faut tout d'abord définir la charge du processeur. La charge d'un processeur est le résultat d'une fonction de poids w sur les éléments locaux du processeur. Soit p_i un processeur, c la limite (bound) de la meilleure solution trouvée jusque là et $\{x_1, \dots, x_k\}$ les limites des éléments du processeur p_i qui peut mener à une meilleure solution ($x_i < c$).

Quelques possibilités pour la fonction de poids $w : \{p_1, \dots, p_n\} \rightarrow N$ sont :

$$w(p_i) = k, \quad (10)$$

$$w(p_i) = \min_j * \{x_j\}, \quad (11)$$

$$w(p_i) = \sum_{j=1}^k (c - x_j)^2, \quad (12)$$

$$w(p_i) = \sum_{j=1}^k e^{c-x} \quad (13)$$

Algorithme distribué

L'algorithme de répartition des charges, présentés dans cet article, est réalisé sur chaque processeur en parallèle avec l'algorithme branch et bound, et connaît la situation de charge locale. Deux processus différents sont construits durant la répartition. Un qui est responsable de la répartition des sous problèmes, les *acteurs*. L'autre, le *contrôleur*, contrôle l'activité de répartition des charges des acteurs et est responsable de l'estimation dynamique de certains paramètres du processus acteur.

Processus acteur

Les algorithmes de répartition des charges distribués dynamiquement peuvent être caractérisés par la connaissance utilisée pour décider quand distribuer les charges (base de décision) et pour l'espace dans lequel un processeur migre une telle charge (espace de migration)

Base de décision :

1. Locale : la décision de migration est basée sur une situation locale donc sur les processeurs voisins dans le réseau.
2. Globale : un processeur doit connaître le système global ou bien tout processeur dans le réseau doit connaître la migration de charges.

Exemple d'algorithme

1. *Initialisation*
2. *Diffusion de nouvelles solutions sur le réseau*
3. *Diffusion de nouvelles solutions sur le réseau (différentes des précédentes)*
4. *Si la charge locale diminue plus que Δ_{down} ¹ alors on envoie REQUEST pour travailler*
5. *Réponse en envoyant du travail qui provoque la prochaine requête*
6. *Un processeur a charge basse est fourni par ses voisins chargés plus lourdement*
7. *Si la charge locale ne fait qu'augmenter, un processeur envoie certains de ses meilleurs sous problèmes à ses voisins, de façon aléatoire, pour éviter la concentration de "bons" sous problèmes et informe ses voisins de la situation de charge réelle*
8. *Requête (REQUEST) de charge de travail de certains voisins.*

Processus contrôleur

Les réseaux de grands diamètres ont besoin de valeurs Δ ² petites pour accomplir une bonne répartition de charges de travail.

La plupart des problèmes de stratégie de répartition des charges arrivent au

¹ Δ_{down} = si le poids de la charge diminue plus que Δ_{down} pourcent, on fait une répartition.

² Δ = un processus p_i participe à une activité de répartition des charges de son voisin p_j , ssi leurs poids est plus grand que *min weight*

début et à la fin du calcul distribué. En effet, le réseau est généralement chargé très bas durant ces phases.

Les décisions de répartition de charges sont les éléments de sortie du processus d'acteur. Les conséquences de ces décisions sont utilisées par les contrôleurs pour calculer les nouveaux paramètres des acteurs.

Finalement, on peut observer que l'algorithme de contrôle tend à décroître les paramètres vers les limites les plus basses et réagit dynamiquement avec une activité de répartition de charges décroît.

En général, l'algorithme peut être appliqué à toutes sortes de problèmes de répartition de charges. Ce qui correspond bien aux problèmes, si ce n'est pas seulement les temps d'inactivité qui ont à être évités mais également tous les processeurs doivent travailler au même niveau, ce qui est défini par un problème dépendant de la fonction de poids. C'est également très robuste pour des situations de charge variant énormément et évite les situations de crash grâce à une stratégie de feed-back.

2.2.5 Répartition des charges avec adversaire [5]

Modèle

Dans cet article, les auteurs modélisent un système distribué par un réseau arbitraire (graphe) dans lequel les noeuds représentent les processeurs et les arêtes représentent les liens de communication. Les charges correspondent, quant à elle, à des tokens indépendants qui peuvent être produits n'importe où.

Dans le but de répartir la distribution de charges, les *tokens* peuvent être communiqués entre différents noeuds. A chaque étape, chaque noeud peut envoyer ou recevoir au plus 1 token sur ses liens de communication incidents. Dans le but d'étudier l'aspect dynamique de la répartition, les auteurs introduisent un modèle *adversaire*. Dans ce modèle, les tokens sont créés et/ou détruits dans les divers noeuds à chaque étape, et un adversaire décide du nombre et de la localisation de ces tokens.

Ainsi, le but de cet algorithme de répartition est de garder le déséquilibre aussi bas que possible à tout moment.

Ainsi, la principale contribution technique de cet article est de démontrer qu'un *simple algorithme de contrôle local* est stable sous le modèle adversaire.

A chaque étape de l'algorithme, d étant le degré du graphe en question :

1. Pour chaque arête (u,v) ,
2. Si u a au moins $2d+1$ tokens de plus que v ,

3. Alors u envoie un token à v .

Cet algorithme représente l'algorithme de répartition locale. Afin de démontrer la stabilité de cet algorithme, une fonction de potentiel appropriée est mise en place et argumente que si le potentiel au début d'une étape est au-dessus d'un certain "seuil", alors l'augmentation du potentiel due à l'adversaire (en ajoutant ou supprimant des tokens), notée A , n'est pas plus grande que la diminution du potentiel, notée B , obtenue en lançant une étape de l'algorithme de répartition local.

L'analyse des chercheurs repose sur la partition des noeuds du graphe sous forme de groupes basés sur les tokens qu'ils ont, et ensuite identifie la stratégie pour l'adversaire dans le but de changer la distribution des tokens entre les groupes pour minimiser $B-A$.

Derrière ce modèle adversaire, on cherche un algorithme de répartition définit de la façon suivante. Un algorithme A est dit *stable* pour un taux r s'il existe un Δ indépendant de t tel que pour tout $t \geq 0$, le déséquilibre de G avec respect de A au début de l'étape t est au plus Δ .

Au final, les auteurs ont montré que l'algorithme de répartition local est stable pour un taux inférieur à 1 et pour n'importe quel réseau adversaire.

2.2.6 Le problème de l'Agent Voyageant [6]

Définition du problème

Soit $n+1$ sites, s_i avec $0 \leq i \leq n$.

Chaque site a une probabilité connue, $0 \leq p_i \leq 1$, qui détermine la capacité à accomplir la tâche de l'agent, et un temps $t_i \geq 0$ est requis pour l'agent afin d'essayer la tâche au site s_i sans tenir compte du fait qu'il y ait succès ou non. Ces probabilités sont indépendantes les unes des autres.

Les temps de voyage, ou *latences*, que l'agent effectue afin de bouger entre les sites sont aussi connus et donnés par $l_{i,j} \geq 0$ (entre les sites i et j). Lorsque la tâche de l'agent a été complétée avec succès à n'importe quel site, l'agent doit retourner au site 0 d'où il est parti. Pour le site 0, $p_0 = t_0 = 0$.

Le but de ce problème est de minimiser le temps attendu pour compléter avec succès la tâche.

Ce problème est NP-complet et les latences sont constantes, se sont toutes les mêmes entre les noeuds.

Ce type de problème pour des agents mobiles est de trouver la meilleure suite de localisations de façon à ce qu'un agent puisse trouver l'information désirée en un minimum de temps s'il y a un trop grand nombre de sites ca-

pable de contenir l'information avec des probabilités et des statistiques telles que la latence, la largeur de bande et la charge de la machine qui sont disponibles.

2.2.7 Analyse

Dans cette partie, nous avons présenté un large éventail des études passées effectuées sur la répartition de charges avec agents. En effet, si on ne considère pas la répartition uniquement sur le cas des agents, il y aurait encore plus d'articles à traiter ou du moins à lire pour approfondir nos connaissances du sujet. La différence notable à considérer entre ces différents modèles et le notre réside dans le fait que notre nouvelle structure peut être mis en place pour le développement d'applications P2P basés sur les idées empruntés des SMA. Le but de cette structure est de fournir un environnement qui simplifie le design et le déploiement des systèmes P2P basés sur ces paradigmes, fournir une simulation pour étudier et expérimenter ces systèmes et dans le but de comprendre leurs propriétés et de faire évoluer leurs performances. Comme nous pourrons le voir dans le chapitre suivant, notre type de système est composé d'une collection de "groupes" interconnectés. Durant le cycle, les agents interagissent avec les services fournis par les groupes, tels que la gestion d'enregistrement et le scheduling d'agents. Les charges sont réparties uniformément dans l'environnement plutôt qu'en pile. Chaque noeud du système génère du travail nouveau et les soumet au réseau.

3 Solution proposée

Notre modèle repose sur 3 principes de base : l'Agent (A), le groupe (G) et le Rôle (R).

Une organisation peut être décrite uniquement par sa structure inhérente. Les relations entre agents sont donc primordiales par rapport à l'agent lui-même et à son comportement. Les structures d'agents formés ne sont pas statiques et évoluent dans le temps. Un agent (cognitif) peut acquérir ou perdre une compétence.

3.1 Motivations et positionnement

Dans le domaine de l'informatique, les réseaux d'ordinateurs sont un exemple typique de système dynamique, complexe, sujet à des perturbations

imprévisibles et pouvant être le siège de phénomène émergent. Dans ce type d'environnement, le problème d'équilibrage de charge et de placement dynamique de processus est plus que jamais d'actualité et représentatif des contraintes rencontrées.

Les algorithmes déjà parus dans la littérature ([7]) se différencient essentiellement par leurs critères de décision (local ou global) et par la distance à laquelle les processus migrent. Mais ils se caractérisent surtout par le fait qu'ils utilisent rarement l'approche naturellement distribuée du paradigme multi-agents. L'originalité de notre travail se situe au niveau des critères utilisés afin de définir l'allocation des processus. En effet, par le passé, tout d'abord, pour les algorithmes d'allocation de processus classiques, essentiellement issus du monde du parallélisme ([8],[9]), le seul critère était la charge des machines. Ensuite, ([10]) présente un algorithme faisant appel à deux types d'agents, les "*Agents Systèmes*" et les "*Agents Application*", qui utilisent cette fois-ci deux critères : la charge des machines et le coût de communication.

Ces travaux relatifs à l'équilibrage de charge et au placement dynamique de processus font face à des problèmes fortement associés aux réseaux tels que la forte dynamique de l'environnement ou la non fiabilité des connexions. Là encore, les méthodes issues de l'étude des insectes sociaux montrent une grande efficacité. Par exemple, [11] ont été les premiers à proposer une approche basée sur les agents pour des problèmes de contrôle dans les réseaux. Comme nous allons le présenter par la suite, notre propre étude reprend des idées inspirées de ces études passées et utilise de nouveaux critères afin de montrer l'évolution de notre système en comparaison avec ceux déjà existants.

3.2 Structure du modèle

Le système que nous avons mis en place va maintenant décrit afin de mieux comprendre notre approche du problème.

Tout d'abord, nous allons présenter les 3 principes indépendamment les uns des autres afin de mieux comprendre leur action dans le modèle.

3.2.1 Les Agents

Afin de comprendre ce qu'est un **agent**, prenons la définition donné par J. Ferber ([12]) :

"On appelle agent intelligent une entité réelle ou abstraite qui est capable d'agir sur elle-même et sur son environnement, qui dispose d'une représentation partielle de cet environnement, qui,

dans un univers multi-agent, peut communiquer avec d'autres agents et dont le comportement est la conséquence de ses observations, de sa connaissance et des interactions avec les autres agents.”

Un agent reste défini à un niveau de description conceptuelle, une entité plus ou moins autonome et communicante plongée dans un environnement implicite ou explicite.

Un agent peut être, au départ, de type "réactif". Un agent *réactif* n'est créé que lorsque des compétences ne pourront être ajoutées dynamiquement. Si on souhaite au contraire que l'agent puisse évoluer au cours du temps, il faut alors instancier directement un agent dit "cognitif" héritant ses caractéristiques directement d'un agent réactif. Un agent *cognitif* saura utiliser des compétences réactives et/ou des compétences cognitives et pourra prendre de lui-même de nouvelles compétences.

Dans notre structure, les agents sont initialisés avec pour données : leur charge et les groupes auxquels ils appartiennent. Ils sont générés par des groupes en réponse aux requêtes de l'utilisateur ; chaque agent tente de satisfaire la requête pour laquelle il a été généré. Chaque agent migrera d'un groupe à l'autre si sa charge de travail engendre une saturation des ressources du dernier groupe visité. Les agents ne pouvant satisfaire leur tâche durant un paramètre TTL (TimeToLive) sont terminés.

Lors de leur déplacement, les agents transportent leur état contenant la requête, les résultats ou des données spécifiques aux autres agents. Si nécessaire, le code de l'agent est transmis avec son état ; le comportement de l'agent est déterminé par son état actuel, son interaction avec le gestionnaire de ressources et son algorithme.

3.2.2 Les Groupes

Un **groupe** est composé de trois modules : un scheduler, une couche de communication et des gestionnaires de ressources.

- Le **scheduler** multiplie la ressource de calcul du groupe entre les agents visiteurs.
- La **couche de communication** est responsable de la gestion de la topologie du réseau (voisinage) et du mouvement d'agents entre groupes. Dans le réseau, chaque noeud a un identifiant unique qui doit être connu en cas de noeud distant. L'ensemble des noeuds distants connus

d'un noeud spécifique est appelé ses "voisins" (*le concept de voisinage n'implique aucune notion de distance*).

- Les groupes offrent leurs ressources aux agents visiteurs à travers un ou plusieurs **gestionnaire de ressources**. Les gestionnaires appliquent un ensemble de politique pour gérer les ressources. Chaque service installé par un groupe est associé à un ensemble de gestionnaire de ressources, par exemple un service de partage de fichier ou le calcul de GRID.

Chaque groupe est initialisé avec un **seuil** d'utilisation de ressources à ne pas dépasser (par défaut 70%).

3.2.3 Les Rôles

Un **rôle** peut représenter un service, une fonction ou une forme d'identification d'un agent.

Un rôle est toujours local à un environnement et repose sur une liste de tâches à accomplir séquentiellement, parallèlement avec ou sans répétition. Un rôle implique donc la tenue d'une ou plusieurs tâches. On considère qu'un agent peut tenir un rôle s'il sait exécuter toutes les tâches impliquées dans le rôle et donc s'il possède les compétences nécessaires pour cela. Un rôle implique également souvent la planification de plusieurs tâches. Pour cela, un agent doit avoir une compétence de planification de tâches.

3.2.4 L'Environnement

Un environnement de simulation est mis en place afin de pouvoir observer l'évolution de notre étude.

Les classes composantes devant être spécifiées comprennent le réseau de groupe simulé, le générateur de requête à utiliser et l'algorithme d'agent à simuler. Les paramètres d'initialisation incluent la durée de simulation, la taille du réseau, la probabilité d'échec, le nombre de requêtes devant être générées et le type et la capacité du gestionnaire de ressources utilisés par l'agent.

3.3 Modèle du système

Le modèle définit repose donc sur le concept d'Agent/ Groupe /Rôle défini par J. Ferber ([13]) comme expliqué précédemment. L'architecture

est axée sur le niveau organisationnel.

Ce système est composé d'un réseau de groupes interconnectés comme on peut le voir sur la Figure 1. Chaque groupe est une couche logicielle capable d'accueillir des ressources et d'accomplir des calculs. Le réseau est caractérisé par l'absence de structure fixée, les groupes vont et viennent et se découvrent au fur et à mesure de la communication.

Les groupes interagissent avec des instances locales d'une ou plusieurs applications et leur fournissent un ensemble de services. Les applications sont les interfaces entre l'utilisateur et le réseau P2P, tant que les services sont distribués et sont basés sur la collaboration entre groupes.

Une application accomplit des *requêtes* et écoute les *réponses* à travers son groupe local. Les requêtes et les réponses constituent l'interface entre les applications et les services. Lorsqu'un groupe reçoit une requête d'une application locale, un service approprié pour manipuler la requête est sélectionné à partir d'un ensemble de services disponibles. Cet ensemble est dynamique. Les services sont implémentés comme des agents autonomes capables de voyager à travers le réseau de groupes.

En réponse à une requête, un ou plusieurs agents sont générés et assignés à une tâche particulière. Les agents, une fois lancés, appartiennent à des groupes spécifiques et ne peuvent se déplacer uniquement vers des groupes auxquels ils ont droit d'être assignés. Si les droits d'appartenance aux groupes ne sont pas en leur faveur, ils seront dans l'incapacité d'y entrer. La provision de services et l'interprétation des requêtes sont délégués par les agents.

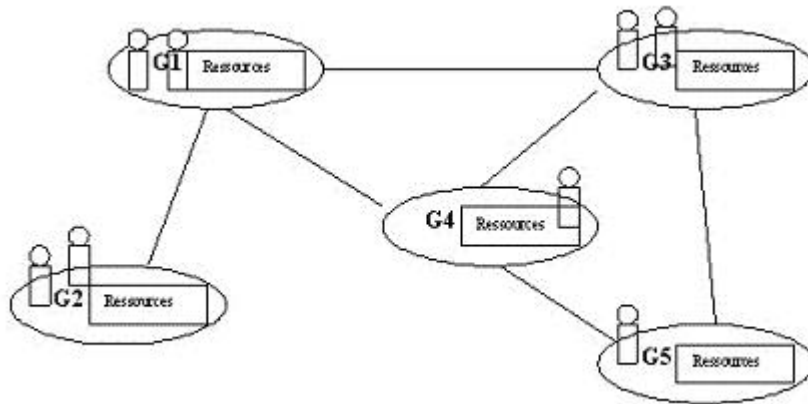


FIG. 1 – Aperçu d'un réseau de groupes

De plus, deux groupes appartenant à la même partition peuvent communiquer sans problème, alors que deux groupes appartenant à des partitions distinctes ne peuvent communiquer. Chaque groupe peut soumettre des "travaux" au réseau de groupes où chaque travail est composé de données en entrée et de l'algorithme calculé sur ces données.

Les travaux sont "planifiés" et exécutés par le groupe dans lequel le travail réside, en invoquant l'algorithme de travail (un travail est complété lorsque l'algorithme associé a été réalisé). Tout d'abord, prenons un réseau composé de différents groupes. Chaque groupe comprend plusieurs agents autorisés à appartenir à ces différents groupes. Chaque agent possède une charge de travail qui lui est propre pouvant évoluer ou non dans le temps. Un agent appartenant à un groupe peut communiquer avec tous les agents présents dans ce même groupe et uniquement avec eux, sauf s'ils appartiennent à deux groupes différents. Dans ce cas, ils peuvent communiquer avec ceux du groupe distant. Chaque groupe du réseau a un seuil défini, le seuil pourra varier, ceci sera décrit un peu plus loin dans ce rapport.

Une des nouveautés particulièrement intéressante de ce système est donc la mise en place d'un **seuil** de ressources à ne pas dépasser pour chaque groupe. En effet, le principal problème traité dans cette étude est l'utilisation plus ou moins excessive des ressources. Le fait de vouloir mettre en place un seuil pour les ressources réside dans le fait qu'il n'y aura jamais de ressources utilisées à l'excès alors que d'autres ressources seront inutilisées. Si chaque groupe définit un seuil à ne pas dépasser, on maintient une certaine stabilité et une certaine homogénéité dans le système.

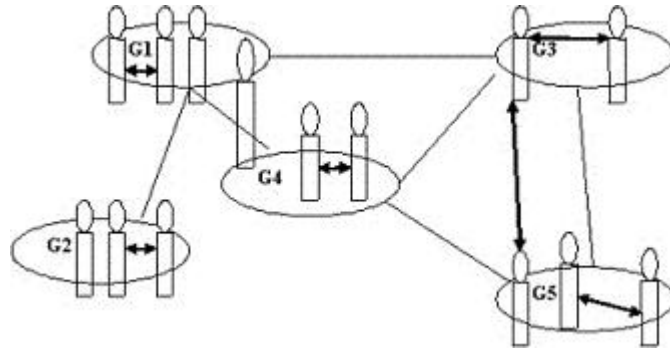


FIG. 2 – Réseau de groupes avec communications

3.3.1 Un exemple de modèle

Prenons pour base d'exemple la Figure 2 vue précédemment, qui est le même réseau de groupes avec les communications retranscrites.

Prenons un des agents du groupe G1. Si cet agent effectue un nombre de travaux trop conséquents, et ainsi met le groupe au-dessus du seuil défini au préalable, c'est à ce moment là que la répartition va entrer en jeu. En effet, devant l'emploi abusif des ressources exercé par cet agent, on va vérifier l'état des autres groupes afin de voir lequel d'entre eux utilise le moins de ressources. Ensuite, on va éliminer l'agent du groupe qu'il sature et le déplacer dans le groupe qui aura le moins de travail afin d'équilibrer au mieux les charges et le travail sur le réseau. Si l'on considère que le groupe le moins chargé dans le réseau à ce stade là est le groupe G3, alors on va déplacer l'agent du groupe G1 vers le groupe G3.

Pour être un peu plus compréhensible, ou du moins plus concis, si l'agent du groupe G1, qui a saturé son groupe, utilisait 13% des ressources alors que le groupe était à 67% (donc en dessous du seuil), alors G1, a atteint 80% , ce qui ne va pas. En parallèle, si G3 utilisait seulement 55% des ressources, alors le fait d'éliminer l'agent de G1 va maintenir G1 à 67% et G3 va maintenant passer à 68% , ce qui correspond à une bonne répartition dans notre étude.

3.3.2 Un exemple avec communications

Comme précédemment, prenons la Figure 2 comme base. Regardons sur la Figure 3, qui n'est qu'une portion du réseau extraite de la Figure 2, les échanges de communication ayant lieu.

Soit un agent i de G2 communiquant avec un agent ii du groupe G3.

Le groupe G2 a une utilisation des ressources supérieure au seuil mis en place alors que G3 est peu chargé. Comme définit précédemment, on va supprimer

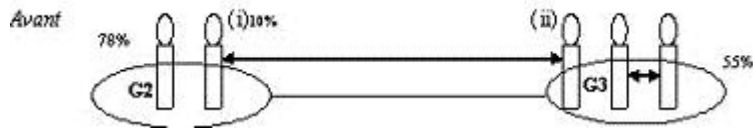


FIG. 3 – Communications **avant** répartition



FIG. 4 – Communications **après** répartition

i du groupe G2, afin de baisser l'emploi des ressources, et transférer cet agent dans le groupe G3. La différence avec la répartition précédente réside dans le fait que la communication entre ces deux agents va être conservée avant, pendant et après la répartition. En effet, les agents vont continuer à communiquer ensemble sauf qu'ils appartiendront au même groupe (voir Figure 4).

Concernant ces déplacements, c'est le gestionnaire de travail de chaque machine qui va définir quel agent chargé, ou peu chargé, doit être transféré d'un groupe à l'autre afin de répartir les charges uniformément sur le réseau. L'essentiel étant de conserver un seuil de travail à ne pas dépasser.

Un autre problème apparaît dans cette étude : la *charge* de l'agent. Si on regarde plus précisément cette charge, le problème correspond à savoir si cette charge est temporaire ou bien constante dans le temps.

En effet, si la charge est temporaire, cela ne vaudra pas le coût de déplacer l'agent car il peut disparaître à tout moment emmenant sa charge avec lui, cela va de soi. Alors que si on prend le cas d'un agent ayant une charge de travail constante dans le temps, il sera plus propice à être déplacé du fait de sa constance.

Pour revenir sur l'emploi et l'importance de la mise en place d'un seuil, il ne faut pas oublier que le problème de répartition de charges réside dans le fait que dans un réseau on est parfois confronté à une utilisation excessive et/ou insuffisante des ressources. En effet, la mise en place d'un seuil implique que toutes les machines présentes sur le réseau travailleront sans excès ET aucun processus ne devrait se trouver en état d'inactivité. Néanmoins, si on se trouve dans le cas où chaque machine a atteint son seuil d'utilisation

de ressources et que d'autres agents sont lancés, la solution efficace sera de créer une nouvelle machine, donc un nouveau groupe, qui va permettre de soulager le réseau et de répartir les charges afin de baisser le taux d'utilisation de chaque groupe en dessous du seuil, sinon on se trouve dans le cas où les groupes sont saturés.

Afin de mieux saisir cette notion de "seuil", une explication des différentes possibilités de mise en place d'un seuil seront décrites dans le chapitre suivant.

3.4 Le Seuil et ses emplois

Ce chapitre va présenter les différentes possibilités de mise en place de la notion de **seuil**.

3.4.1 Le seuil *fixe*

Une fois que la décision de définir un seuil a été prise, par défaut, nous avons mis en place un seuil que nous définirons de **fixe**.

Ce seuil, que nous initialisons par défaut à un taux fixe d'utilisation des ressources de 70% , va nous permettre de répartir les charges entre les groupes de deux façons distinctes :

- Une allocation directe sur le groupe le plus faible, i.e. si un groupe a son taux de ressources proche du seuil, donc proche de la saturation, la répartition va se faire sur le groupe ayant le moins de ressources utilisées à cet instant donné.
- Une allocation tenant compte des mouvements, i.e. si un groupe a tendance à se remplir plus rapidement vis à vis des autres, les agents vont se retrouver dans une configuration où ils vont bouger rapidement vers d'autres groupes ayant, peut-être, une capacité beaucoup plus lente de remplissage.

On va donc se retrouver dans une situation où l'attitude des groupes prend une part prépondérante de notre étude, et c'est là que ressurgit l'efficacité du seuil.

En effet, ce seuil *fixe* va mettre tous les groupes a un même pied d'égalité tout en gardant en tête que le comportement du groupe est assez aléatoire. Si on regarde les études précédentes où l'idée de mettre en place un seuil n'apparaît pas, la répartition des charges se fait uniquement en fonction de l'attitude de l'agent alors qu'ici, nous traitons les agents et les groupes à un même niveau. De plus, l'utilisation des systèmes à multi-agent pour ce type

d'étude n'est pas très fréquente, ce qui laisse un large éventail de solutions pouvant être mises en place pour cette branche de la recherche.

L'idée qui ressurgit également est le fait de pouvoir voir, avec un seuil fixe, le comportement de chaque groupe au fur et à mesure du phénomène de répartition. En effet, lorsque les agents commencent à se déplacer entre les différents groupes, ils vont faire ressurgir les activités effectuées par chaque groupe (vitesse de remplissage, ordre de priorité).

Ce système peut être vu comme décentralisé.

Néanmoins, si on regarde le système de façon centralisée, on est en présence d'un phénomène dit *d'appel d'offre*. En effet, chaque groupe va demander à ce qu'il lui soit affecté un agent à un moment donné car il ne travaille pas assez ou bien à l'inverse dira qu'il est saturé, ou proche de la saturation, et donc qu'il n'a plus besoin d'affectations.

Comme dit précédemment en début de ce chapitre, le seuil fixe est le seuil qui pour nous sera donné par défaut afin de pouvoir voir plus facilement les attitudes des agents au sein même du réseau.

3.4.2 Le seuil *différentiel*

Une deuxième cas d'étude s'est directement présenté à nous après la mise en place du seuil fixe : un seuil **différentiel**.

Ce type de seuil, comme son nom l'indique, va représenter un seuil différent pour chaque groupe du réseau.

En effet, au départ, la première approche de mise en place d'un seuil était de mettre le même pour tous. Néanmoins, en pratique, chaque machine (représentée par les groupes) n'utilise pas nécessairement le même nombre de ressources. A partir de là, il a été assez facile, et assez logique de définir un seuil dit *différentiel*.

Compte tenu de la capacité assez grosse des réseaux à contenir des machines à utilisation de ressources variables (toutes les machines n'étant pas les mêmes), ce seuil apparaît comme étant le plus représentatif d'un réseau réel.

Une fois la mise en place d'un tel seuil établie, le problème réside au niveau de l'accessibilité dans un groupe pour les agents. En effet, le seuil de chaque groupe n'étant pas le même, un agent possédant une charge élevée ne pourra certainement pas accéder à tous les groupes. De plus, un groupe pourra se trouver avec une utilisation faible du fait d'un seuil élevé, alors qu'un groupe pourra se trouver presque saturé avec un seuil faible.

Néanmoins, la répartition pourra se faire plus facilement du fait de l'emploi

de ce type de seuil. Avec une limite faible à ne pas dépasser, un groupe ne se trouvera jamais inutilisé et un groupe ayant une limite plus élevée se verra alloué des agents beaucoup plus utiles à son bon fonctionnement.

3.4.3 Le seuil *différentiel variable*

La dernière approche, découlante directement des deux précédentes, est la mise en place d'un seuil **différentiel variable**.

Ce seuil, le plus complexe des trois, se rapproche de la configuration du précédent (le seuil différentiel), sauf qu'il aura une variation dans le temps. En effet, chaque groupe aura un seuil différent des autres mais en plus le réseau se verra subir des variations au niveau des charges de ces groupes.

Par exemple, si on se trouve en présence d'une machine ayant un seuil de ressources utilisés assez bas et qu'il se trouve dans une configuration où il nécessite un niveau d'utilisation de ressources plus élevé, son seuil variera en fonction de cette nécessité.

De plus, cette variation de seuil va permettre aux différentes machines du réseau d'effectuer une répartition nettement plus efficace qu'avec les seuils précédents. Les agents seront répartis sur le réseau en fonction des demandes des groupes et ainsi, la répartition des charges au sein du réseau sera optimale.

3.4.4 Analyse

Grace à la mise en place de ces notions de seuil, notre phénomène de répartition se retrouve facilité – le seuil permettant une bonne répartition des charges tout en évitant une migration excessive. En effet, si on prend en considération ne serait-ce que pour base de départ le cas du seuil fixe, il apparaît assez bien qu'avec une telle contrainte le réseau se verra plus facilement réparti et donc beaucoup moins saturé. Le fait qu'un groupe ne puisse dépasser un seuil d'utilisation de ressources que l'on établit à 70%, l'obligera à transférer des agents vers d'autres groupes et ainsi, d'une part il se soulagera de travaux dont il ne pouvait assumer l'exécution, et d'autre part il permettra à d'autres groupes qui se trouvaient en état d'inactivité (ou d'activité assez basse) de pouvoir se lancer. Ainsi, le réseau ne se retrouvera pas déséquilibré avec des groupes travaillant avec excès et des groupes ne travaillant pas.

Pour les deux autres cas que sont le seuil différentiel et le seuil différentiel variable, le constat est le même. En effet, la différence qui existe avec le premier cas réside dans le fait que ces deux seuils seront des seuils à va-

leurs différentes pour les groupes composants le réseau mais le phénomène de répartition, à l'aide de son algorithme présenté par la suite, s'effectuera de la même manière, seule la durée de la répartition diffèrera.

4 Simulations

Comme expliqué dans les sections précédentes, l'ensemble de notre étude a été implémentée et appliquée sous la plateforme multi-agent MadKit.

4.1 MadKit

La plateforme **MadKit**, mise en place par J. Ferber, O. Gutknecht et F. Michel (et toute l'équipe MadKit), est un logiciel orienté agent avec lequel on peut construire des applications distribuées. La fonction la plus intéressante de MadKit est que la distribution se fait de manière transparente. Il n'est pas nécessaire de prendre soin de voir si l'agent est éloigné ou non, MadKit manipule toutes les communications et les connexions nécessaires pour faire en sorte que les agents situés sur des plateformes différentes travaillent ensemble avec transparence. Tout ce dont on doit se soucier est :

- l'utilisation des techniques de passage de messages lors des communication entre les différents agents
- lancer MadKit sur les deux noyaux(kernel) éloignés, et ouvrir un agent *Communicator* sur les deux. Une fois fait, sur un des deux noyaux on donne le nom de l'autre et les deux seront alors connectés.

Il ne faut pas oublier de noter que la plateforme MadKit travaille sur les bases du "peer-to-peer", i.e. qu'il n'y a pas de "serveur", ni de "client". Chaque noyau est à la fois serveur et client.

Un autre point fort de ce logiciel est bien sûr la possibilité de créer de nombreuses simulations à base d'agents, mais également de pouvoir le faire dans tous les langages de programmation pouvant être compilé sous Java. A l'heure actuelle, 4 langages scripts sont utilisables : Scheme(Kawa), Jess(langage basé sur les règles), BeanShell(interprété en Java) et Python(jython).

4.2 Application pour notre étude

Une fois la prise en main du logiciel MadKit effectuée, l'application de notre modèle pour notre étude s'est trouvée facilitée. En effet, la répartition

des charges dans notre cas est basée sur l'emploi, la mise en place et la manipulation d'agents et, comme expliqué dans les chapitres précédents, cette étude est assez nouvelle au niveau des systèmes multi-agents. Ainsi, MadKit apparaît comme l'outil le plus approprié afin de pouvoir visualiser l'impact important de l'application des agents pour la répartition de charges au sein d'un réseau complexe.

4.2.1 Algorithme de répartition

Avant de présenter les différents résultats obtenus lors de cette étude, la présentation de l'algorithme de répartition qui a été mis en place est nécessaire.

Tout d'abord, l'environnement dans lequel les agents se trouvent est donné par le réseau des groupes. Chaque groupe est initialisé avec un seuil d'utilisation des ressources à ne pas dépasser (3 cas distincts). Durant la durée de vie, les agents créés obtiennent des charges de travail qui diffèrent selon les besoins que les agents ont à l'intérieur même du réseau et du groupe auxquels ils appartiennent.

Cet algorithme d'évolution du modèle peut être vu de la sorte :

- Tout d'abord, les groupes sont créés à l'intérieur d'une même **communauté**. Si les groupes sont tous détruits, la communauté disparaît avec eux.
- Une fois ces groupes créés, les agents sont lancés les uns après les autres avec des charges aléatoires. Ces agents appartiennent uniquement aux groupes pour lesquels ils ont un droit d'accès.
- Les groupes, à partir du moment où les agents ont été lancés, vont voir leur charge de travail, ou taux d'utilisation de leurs ressources, calculée par le gestionnaire de travail afin de vérifier qu'ils ne dépassent pas leur seuil donné en initialisation.
- Si on se trouve dans le cas où le seuil n'a pas été dépassé, on continue de vérifier l'état des groupes sans modifications particulières pour l'instant.
- Néanmoins, si le seuil est dépassé, le gestionnaire de travail du groupe en question va éliminer l'agent ayant saturé le groupe et le transférer vers un groupe ayant une charge de travail peu élevée.
- Dernier cas de figure, si on est en présence de saturation mais que tous les groupes sont au maximum d'utilisation de leurs ressources, alors un nouveau groupe devra être créé afin de pouvoir transférer l'agent saturant.

Le processus de transfert a lieu par lien direct entre deux groupes. La **charge** d'un groupe est définie par le nombre de travaux actuellement dans la file d'attente de travail de chaque groupe, alternativement, si l'information de la puissance de calcul nécessaire pour réaliser les travaux est disponible, alors la charge d'un groupe est basée sur cette information. Toutefois, chaque groupe doit avoir une charge de travail ne devant pas excéder leur seuil.

Les concepts de surcharge et sous charge sont relatifs à la charge moyenne des groupes récemment occupés par les agents. Cette définition permet aux gestionnaires de travail de chaque groupe de prendre les décisions au sujet des transferts de travail entre groupes.

Présentation rapide d'un algorithme

Une idée de l'algorithme :

Algorithm 1 Balancing algorithm

```
0: Group  $\leftarrow$  group_nb ; group_threshold ; ressources_level ;  
   Agent  $\leftarrow$  agent_nb ; agent_load ;  
   Creating groups ;  
   Launching agents ;  
   while Launching agents do  
     Agents affectation ;  
     Check ressources_level ;  
   end while  
   if reception_message then  
     Analyse() ;  
   end if  
   return ressources_level
```

Algorithm 2 Function Analyse()

```
if (agent_load > group_threshold) then  
  Create group with threshold > agent_load ;  
else if ( $\sum$ (agent_load) > group_threshold) then  
  Repart() ;  
end if
```

Algorithm 3 Function Repart()

```
while Repart() do  
  Leave group saturated by agent ;  
  Look for group with ressources_level low ;  
  Request role in group ;  
end while
```

```

public void activate(){
    createGroup(true, AllocationHello.Allocation, myGroup, null, null);
    requestRole(AllocationHello.Allocation,"Group 2","test",null);
    requestRole(AllocationHello.Allocation,"Group 3","test",null);
}
public void live()
{while(true)
    {
        Message msg = nextMessage();
        if (msg != null)
            {System.err.println("You know what ? I receive a message ! "+msg);
             handleMessage(msg);
            }
    }
}

public void Analyse()
{
    double x = Math.round(Math.random()*20); // x représente la load utilisée par l'agent dans le Group
    double X = Math.round(Math.random()*20);
    System.err.println(" Agent load = "+somme(x+X));

    if(niveau_ressources<=threshold)
    {
        niveau_ressources = somme(somme(x+X)+somme(niveau_ressources));
        if(niveau_ressources>69) {
            group_load=(niveau_ressources-somme(x+X));
            System.err.println( "----- Group load = "+(niveau_ressources-somme(x+X))+"----- ");
            StringMessage msg = new StringMessage("Group load");
            broadcastMessage(AllocationHello.Allocation, "start","startInput", msg);

            niveau_ressources=niveau_ressources-(niveau_ressources-somme(x+X)); // si un agent sature le groupe
                                                    // on conserve sa charge pour
                                                    //calculer le taux du groupe suivant
            niveau_ressources=somme(x+X); // on continue à additionner les charges des agents dans les
                                                    // nouveaux groupes
            StringMessage msg2 = new StringMessage("Balancing");
            broadcastMessage(AllocationHello.Allocation, "start","startInput", msg2);
            repart();
        }
    }
    else {
        println("-----");
        requestRole(AllocationHello.Allocation,"Group 2","SOMme",null);
        println("Use of resources OK");
        println("-----");
    }
}

public void Repartir_charge() //on va tuer l'agent saturant les ressources
{
    pause(50);
    leaveGroup(Repartition.myGroup);
    leaveRole(Repartition.myGroup,"launcher");
    leaveRole(Repartition,"Groupe 2","launcher");
    //requestRole(Repartition,"Groupe 1","saturé",null);
}

```

FIG. 5 – Code pour l'algorithme de répartition

4.2.2 Simulations avec seuil fixe

Prenons donc comme base de départ, un petit réseau composé de seulement 5 machines. Pour chacune de ces 5 machines, on va définir un seuil fixe, dont on a parlé dans les chapitres précédents, d'utilisations des ressources de 70% au maximum.

Une fois ce paramètre initialisé, les machines, ou groupes pour nous, vont attendre les différents agents qui vont être produits dans le réseau. Chaque agent aura sa propre charge, i.e. son propre taux d'utilisation de ressources dans le système.

Au départ, avant que les agents soient lancés, le réseau se trouve en attente et chaque groupe n'utilise qu'une infime partie des ressources. Après la création des différents agents (10 pour commencer), l'algorithme de répartition va se lancer en tenant compte de l'arrivée de ces nouveaux éléments.

Tout d'abord, comme nous sommes dans le cas où le seuil est fixe, les agents vont rentrer dans le réseau en commençant par le premier groupe et pas un autre. Ainsi, ce groupe G1 va se remplir en accueillant les agents qui auront les droits d'accès à ce groupe jusqu'à ce que son taux d'utilisation atteigne le seuil défini de 70%.

Pour cela, au fur et à mesure que les agents rentrent dans le(s) groupe(s), la somme des charges des agents entrés est calculée et si cette somme dépasse le seuil alors le dernier agent entré dans le groupe sera enlevé et transféré dans le groupe suivant.

Ainsi, les groupes seront chargés les uns après les autres. C'est ce que l'on peut voir sur la Figure 6 avec pour données :

- Agent1 = 10%
- Agent2 = 14%
- Agent3 = 20%
- Agent4 = 5%
- Agent5 = 35%
- Agent6 = 14%
- Agent7 = 38%
- Agent8 = 25%
- Agent9 = 19%
- Agent10 = 49%

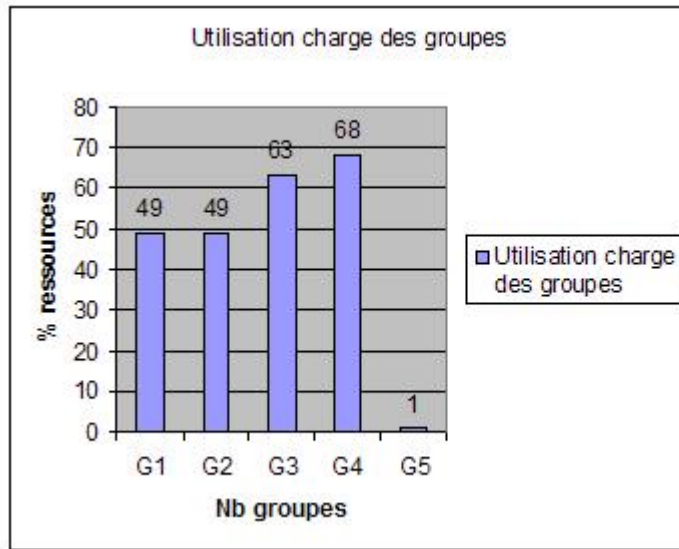


FIG. 6 – Etat du réseau après répartition à seuil fixe

Avec $G1=(Agent1+Agent2+Agent3+Agent4)$, $G2=(Agent5+Agent6)$, $G3=(Agent7+Agent8)$ et finalement $G4=(Agent9+Agent10)$.

Une fois que ces agents ont été répartis au sein des différents groupes, il ne faut pas oublier que le but de cette étude est de faire en sorte que les différents groupes du réseau ne soit pas surchargés, ni même en état d'inactivité. Si on regarde la Figure 6, il apparaît que le groupe G5 n'a reçu aucun agent du fait de la non saturation du groupe qui le précède. Afin de pallier ce problème, une nouvelle analyse s'opère consistant à prendre les groupes les plus chargés (les plus proches de leur seuil) et à les soulager des agents qui n'utilisent pas plus de la moitié de leurs ressources.

Dans le cas de la Figure 6, les deux groupes qui vont être soulagés sont les groupes G3 et G4 à qui on va enlever respectivement l'Agent8 et l'Agent9. Ainsi, on obtient une répartition correcte sur le réseau comme le montre la Figure 7.

4.2.3 Simulations avec seuil différentiel

Cette fois-ci, nous allons partir sur la même base de réseau que précédemment, c'est à dire en prenant un réseau composé de 5 groupes.

Pour ces différents groupes, on va maintenant définir un seuil différentiel (qui a été défini dans les sections précédentes). On va également lancer seulement 10 agents afin de voir plus facilement et plus rapidement les effets de la

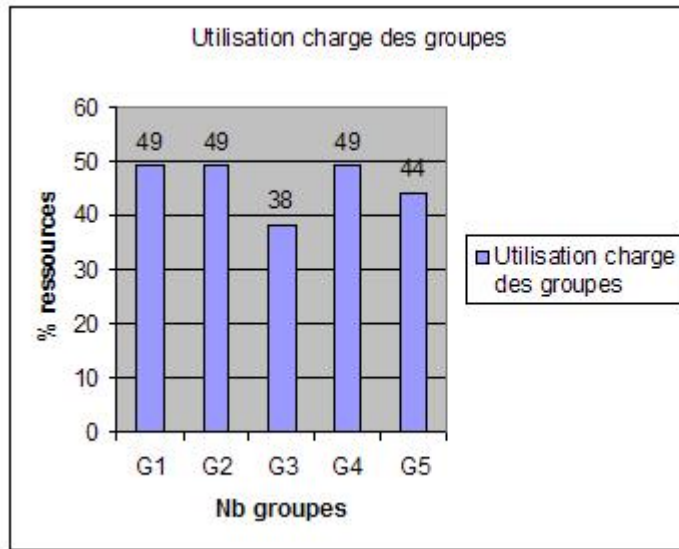


FIG. 7 – Amélioration de la répartition (seuil fixe)

répartition. On va prendre comme éléments du réseau les dix agents déjà définis dans l'étude des "premiers résultats" citée précédemment.

Par contre, les seuils des groupes étant différents, on va les définir de la façon suivante :

- G1 = 40%
- G2 = 64%
- G3 = 70%
- G4 = 52%
- G5 = 35%

Et du coup, on a : $G1=(Agent1+Agent2)$, $G2=(Agent3+Agent4+Agent5)$, $G3=(Agent6+Agent7)$, $G4=(Agent8+Agent9)$ et $G5=Agent10$.

Néanmoins, on peut remarquer sur la Figure 8 que le groupe G5 pose encore problème. En effet, après le premier tour d'itération de l'algorithme, les agents ayant été lancés, la fonction *Analyse()* entre en jeu due aux échanges de messages entre les groupes (ces messages concernent le niveau d'utilisation des ressources des groupes). Cette dernière fait ressortir le fait que le dernier agent (l'Agent10) possède une charge supérieure à celle du seuil de tolérance du groupe G5 (49% > 35%). Dans ce cas, vu que l'agent utilise un pourcentage trop élevé de ressources, il va être transféré vers un nouveau groupe créé pour l'occasion. Ce groupe (le groupe G6) va posséder un seuil permettant d'accueillir cet agent. Une fois ce groupe G6 créé, une deuxième itération est effectuée permettant la répartition des agents en fonction de la présence de

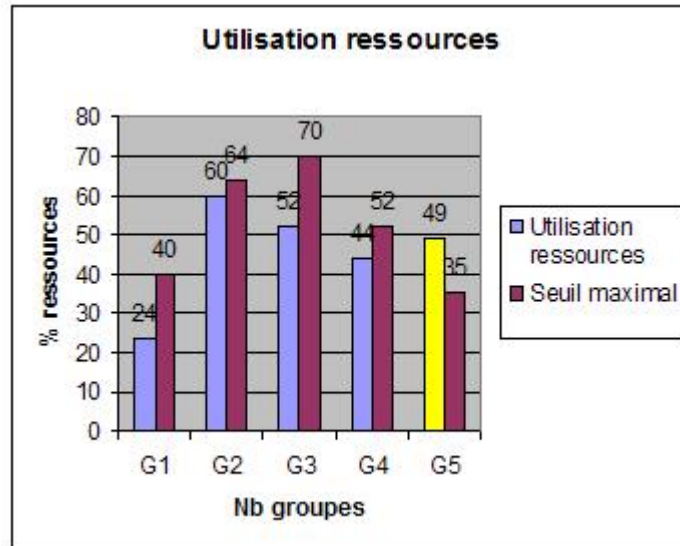


FIG. 8 – Etat du réseau après répartition à seuil différentiel

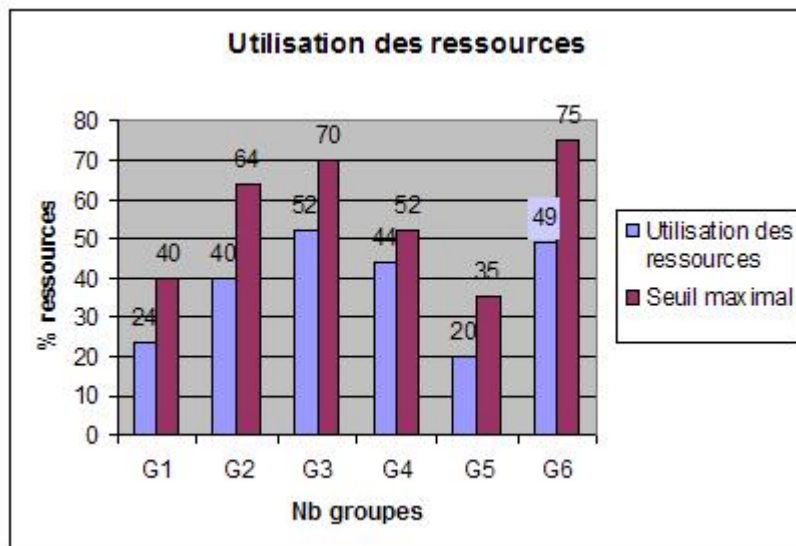


FIG. 9 – Amélioration de la répartition (seuil différentiel)

ce nouveau groupe. Ici, c'est l'Agent3 qui va être transféré du groupe G2 (car c'est celui qui est le plus proche de la saturation) vers le groupe G5 de sorte que les groupes ne soient ni trop, ni peu chargés. Le résultat peut se voir sur le diagramme de la Figure 9.

4.2.4 Simulations avec seuil différentiel variable

Toujours pour le même réseau (5 groupes et 10 agents), cette fois-ci, le seuil mis en place est un seuil différentiel mais variable.

De la même façon que pour le seuil différentiel, il faut définir les différents seuils possible pour chaque groupe en tenant compte du fait qu'ils évoluent dans le temps. Pour cela, on définit les groupes comme ci-dessous :

- G1 = 20-40%
- G2 = 44-64%
- G3 = 50-70%
- G4 = 32-52%
- G5 = 15-35%

Les agents quant à eux ne changent pas et on se reportera à ceux donnés dans les sections précédentes.

Dans cette configuration, la valeur des seuils des différents groupes varie au cours du temps. Par exemple, d'après les données précédentes, le groupe G1 pourra avoir un seuil de ressources établit à 20%, ce qui sera assez bas, alors qu'il pourra atteindre les 40% à un instant donné. A partir de ce constat, il apparaît que la solution la plus probante à mettre en place est la création de nouveaux groupes qui sera appliquée au moment où tous les autres se trouveront dans des contextes de saturation.

Prenons par exemple la Figure 11 où les groupes, à l'instant t , auront comme valeur de seuil :

- G1 = 25% (compris entre 20 et 40)
- G2 = 60% (compris entre 44 et 64)
- G3 = 52% (compris entre 50 et 70)
- G4 = 45% (compris entre 32 et 52)
- G5 = 28% (compris entre 15 et 35)

Le problème qui se pose reste le même que pour les cas précédents : le groupe G5 est saturé. Et comme précédemment, après deux itérations de l'algorithme, un nouveau groupe G6 va être créé pouvant accueillir cet Agent10. Il ne faut tout de même pas oublier que ce nouveau groupe aura lui aussi un seuil différentiel variable, mais à l'instant où il est créé, il doit avoir un seuil permettant d'accueillir cet agent transféré sous peine de n'être d'aucune

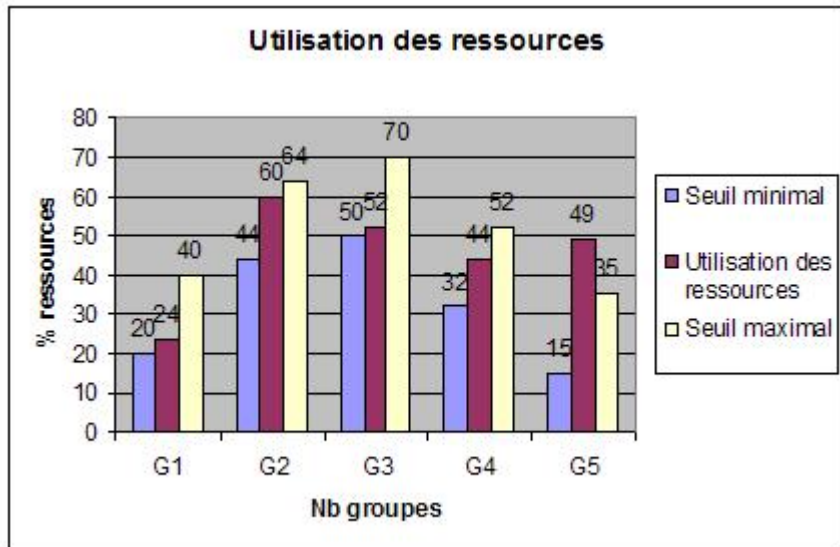


FIG. 10 – Présentation du réseau avec un seuil différentiel variable

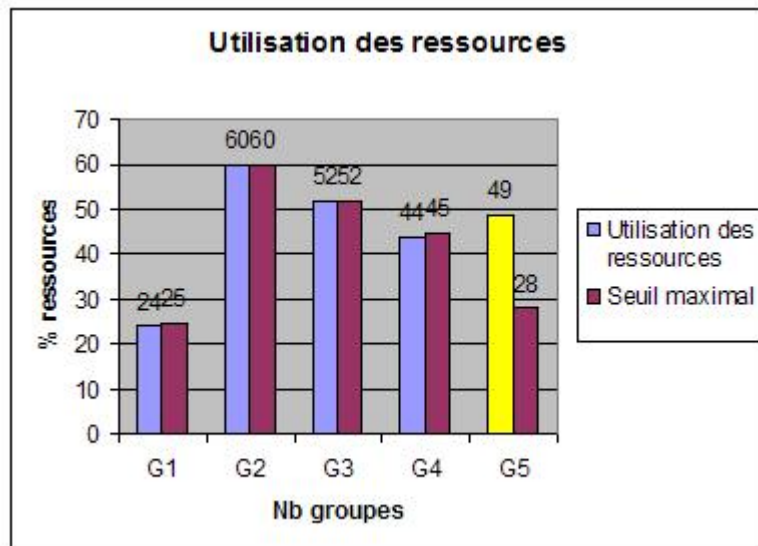


FIG. 11 – Etat du réseau à seuil différentiel variable à l'instant 0

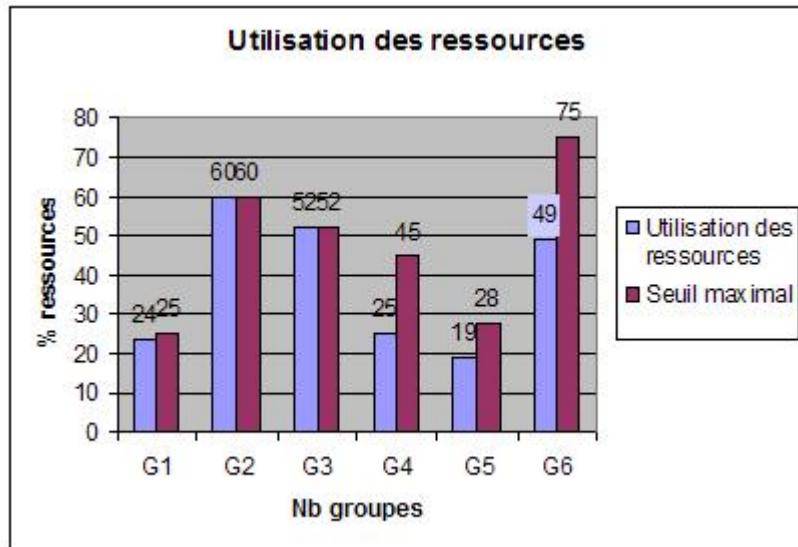


FIG. 12 – Etat du réseau à seuil différentiel variable à l’instant t

utilité. C’est ce que l’on peut voir sur le diagramme de la Figure 12.

A noter que toutes ces modifications ne sont possible que grace à l’algorithme défini précédemment.

Une fois que cet Agent ”saturant” a été transféré dans le nouveau groupe créé, il reste le problème de la variation des seuils des groupes. En effet, sur la Figure 12, à l’instant t , on remarque que de nombreux groupes sont à la limite de leur seuil et à l’instant $t+1$ on peut se retrouver dans une configuration correspondant au diagramme de la Figure 13.

Dans ce cas, ce n’est plus 1 groupe qui est saturé, mais 3. A partir de ce constat, l’algorithme de répartition va une nouvelle fois se lancer afin de fournir au réseau une répartition des charges uniformes en fonction de l’instant donné. Ainsi, on obtient le réseau du diagramme de la Figure 14. Deux groupes supplémentaires ont été créé lors de l’exécution de l’algorithme et la répartition s’est opérée sur le réseau.

Néanmoins, concernant cette partie sur le seuil différentiel variable, tout n’a pas pu être testé sous notre simulateur car il existe un trop grand nombre de probabilité d’évènements. En effet, les groupes peuvent voir leur seuil varier rapidement ceci en fonction des besoins proposés par le réseau. Un groupe peut passer d’un état où il n’aura pas besoin d’un trop grand nombre d’agents, à un état où il lui en faudra beaucoup plus dû au fait qu’il rencontre une demande de travail grandissante. Le but de l’algorithme n’est pas d’improviser une répartition pour le réseau, mais il s’agit d’agir en fonction

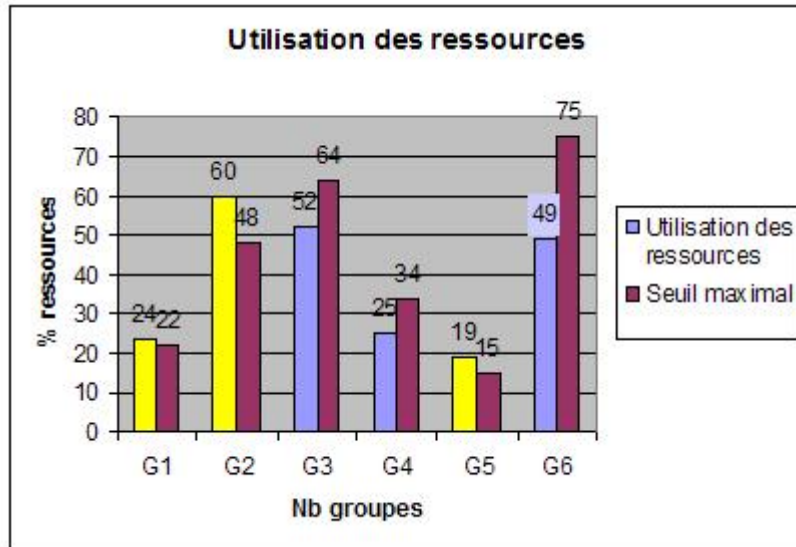


FIG. 13 – Etat du réseau à seuil différentiel variable à l'instant $t+1$

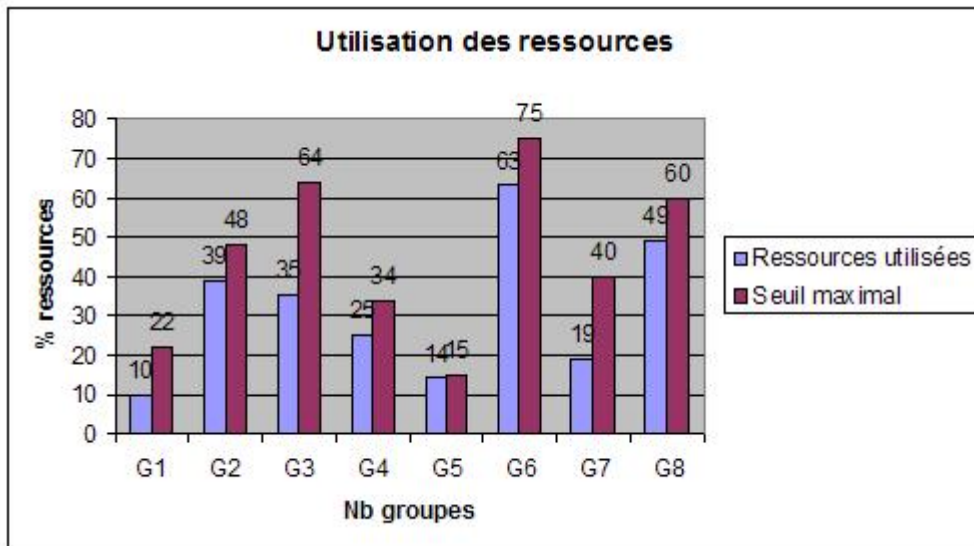


FIG. 14 – Etat du réseau à seuil différentiel variable à l'instant $t+2$

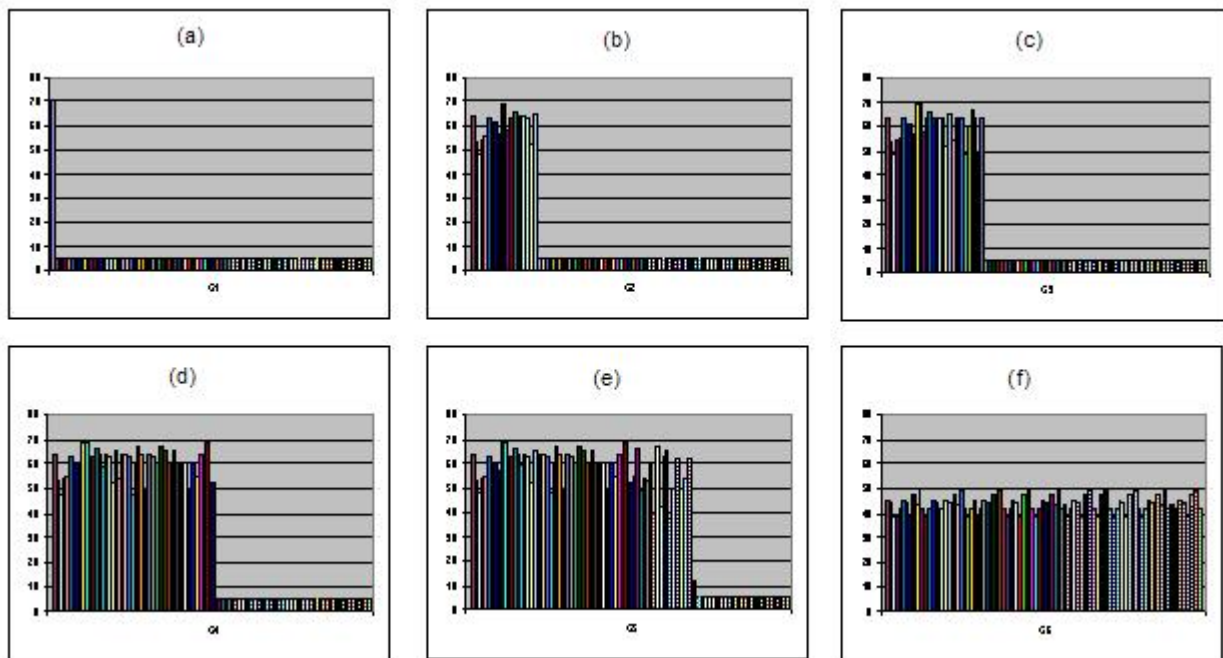


FIG. 15 – Processus de répartition sans création de groupes

des attitudes et des besoins rencontrés par les groupes composants ce réseau. Malgré tout cela, cet algorithme de répartition reste le plus efficace pour notre étude et un approfondissement, voir une amélioration, de son comportement sera nécessaire dans le futur.

4.2.5 Exemples du phénomène de répartition

Afin de comprendre le phénomène de répartition qui s'exerce durant le lancement de l'algorithme, un petit exemple graphique s'impose. Sur les 6 diagrammes de la Figure 15, on peut voir les différents déplacements s'opérant à chaque itération de l'algorithme.

Les diagrammes ont été obtenus sous un réseau composé d'une centaine de groupes interconnectés les uns aux autres. A l'initialisation, 200 agents ont été lancés sur le réseau, chacun détenant une charge lui étant propre. Les différents histogrammes ((a),(b),(c),(d),(e) et (f)) présentent l'évolution de la répartition après 0, 5, 10, 20, 40 et 70 itérations de l'algorithme.

Une fois les agents lancés, ils entrent dans le premier groupe, ce que l'on peut voir sur le diagramme (a). A partir de là, l'algorithme effectue ses tours de boucle de façon à répartir les agents, et donc leurs charges, sur le réseau.

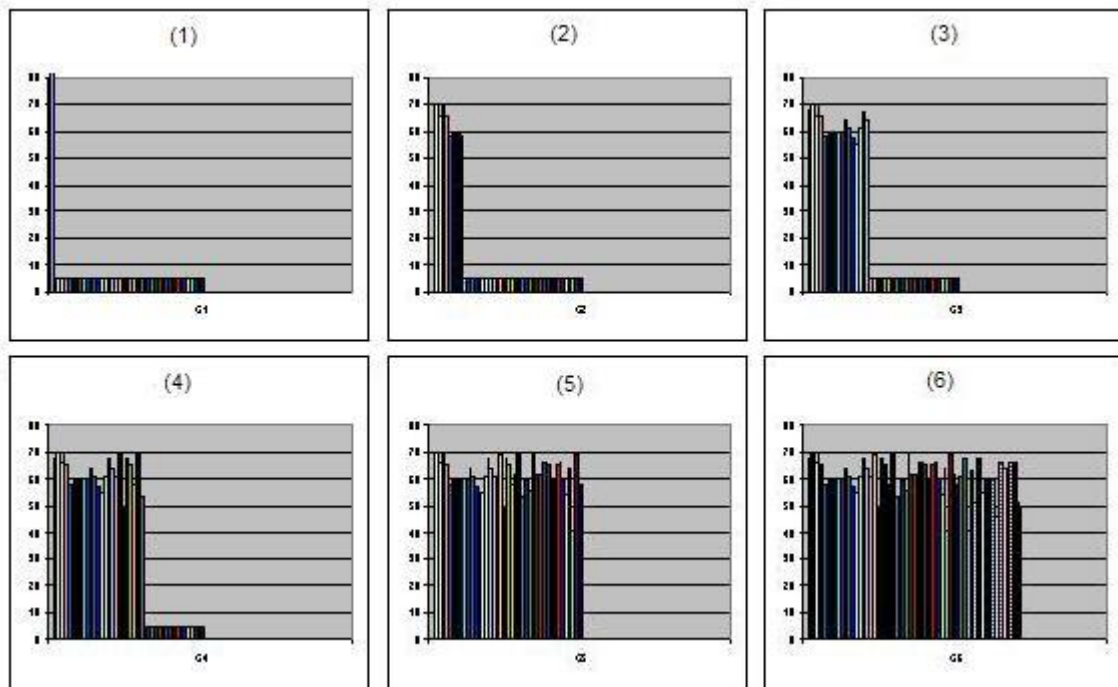


FIG. 16 – Processus de répartition avec création de groupes

A chaque itération, en fonction des niveaux de charge de chaque groupe, un ensemble d'agents réalise un pas, i.e. la fonction *Analyse()* est exécutée, et migre vers son groupe voisin en cas de saturation du groupe. Afin que tous les agents lancés soient affectés au moins à un groupe, soixante-dix itérations sont nécessaires (diagramme (e)). En effet, après ces itérations, les charges des 200 agents ont été réparties sur le réseau entre les premiers groupes. Hélas, les groupes restant ne sont pas encore actifs (cf diagramme (e)), ils n'ont reçu aucun agent. Ainsi, une dernière analyse va être effectuée afin que les groupes restants puissent travailler et pour que la répartition soit, au final, uniforme sur le réseau (avec un taux moyen d'utilisation des ressources de 45% pour les groupes). Le diagramme (f) de la Figure 15 présente cela. On peut également voir le fonctionnement du phénomène sur la capture d'écran de la Figure 17 située en Annexes.

Sur la Figure 16, on a le même phénomène que précédemment, excepté que le nombre de groupes lancés à l'initialisation est de 50. On constate, avec le même nombre d'agents lancés (200), que le nombre de groupes au bout de la 50ème itération (diagramme (5)) n'est pas suffisant pour que la répartition soit bonne. En effet, une fois que tous les agents ont été lancés, les 50 groupes

reçoivent les agents entrés dans le réseau ainsi que leurs charges respectives. Le problème est qu'à partir de la 50ème itération, les groupes ont tous atteint leur seuil de ressources et donc ne peuvent plus accueillir d'agents. Pourtant, il reste encore des agents n'ayant pas reçu d'affectations. C'est pour cela que des groupes supplémentaires vont être créés, dans notre cas une vingtaine (diagramme (6)); ce nombre de groupes créés dépend du nombre d'agents n'ayant pas encore été affectés. De plus, bien entendu, ces groupes créés sont lancés avec comme seuil le même que les autres puisque nous sommes dans une configuration avec seuil fixe. (voir également l'Annexe)

L'essentiel des résultats de ce phénomène est issu d'une répartition des charges avec seuil fixe car celle-ci permet de mieux observer ce phénomène et ainsi permet de mieux se rendre compte de l'efficacité de cet algorithme. Cette notion de seuil permet finalement de réaliser une répartition efficace, bien entendue, mais surtout assez rapide comparée aux études précédentes.

4.3 Evaluation des expérimentations

Dans la section précédente, nous avons présenté les résultats préliminaires du modèle organisationnel mis en place.

Les différents diagrammes des résultats précédents ont été obtenus sur des réseaux de 5 groupes connectés sous forme d'anneau pour des raisons de visualisations. A l'initialisation, 10 travaux (agents) sont générés dans un seul groupe. Les diagrammes représentent les charges observées au sein des groupes après seulement trois itérations de l'algorithme de répartition. A chaque itération, les groupes réalisent une seule étape, i.e. ils exécutent la méthode *live()* et agissent en fonction des messages qu'ils émettent et qu'ils reçoivent. Comme on peut le voir sur les diagrammes, environ trois itérations sont nécessaires (avec le nombre d'agents établis à 10) pour répartir les charges uniformément sur le réseau. En effet, à l'initialisation les groupes sont lancés avec un seuil défini mais une charge quasiment nulle. Ce n'est qu'à partir de la première itération que les agents entrent en jeu et donc que les groupes sont explorés. La deuxième itération va continuer le processus de répartition et d'autres itérations seront effectuées si nécessaire afin que les charges dans le réseau soit au final correctement réparties.

Concernant des cas plus généraux comme des réseaux comprenant un nombre beaucoup plus vaste de groupes et d'agents (ce qui correspond nettement plus à la réalité), voir la Figure 15, le résultat restera sensiblement le même hormis le fait que le nombre d'itérations de l'algorithme sera beaucoup plus grand et

donc beaucoup plus long. En règle générale, on se trouvera avec un nombre d'itérations de l'algorithme de l'ordre de :

$$Nbiterations \simeq \frac{NbAgents}{2,8} \quad (14)$$

Les résultats des simulations précédentes ont été obtenus sans utiliser une répartition des charges dynamiques, mais dans une configuration statique. Les agents sont assignés automatiquement aux groupes et l'attribution initiale est réalisée de sorte à ce que les agents soient répartis de façon optimale entre les différents groupes.

5 Discussions et perspectives

Après avoir étudié les différentes approches effectuées par le passé, le fait le plus marquant qui ressort est la diversité des solutions proposées. Néanmoins, la répartition des charges vue sous le contexte des systèmes multi-agents reste une nouveauté. En effet, peu de chercheurs se sont penchés sur l'emploi des agents pour la répartition des charges.

Les systèmes multi-agents sont assez récents et méconnus du grand public. De plus en plus d'informaticiens se tournent vers ces systèmes du fait de leur efficacité. En effet, aujourd'hui, la plupart des applications nécessitent de distribuer des tâches entre des "entités" autonomes (ou semi-autonomes) afin d'atteindre leurs objectifs d'une manière optimale. Puisque les approches classiques sont en générales monolithiques et leur concept d'intelligence centralisés, les applications actuelles sont établies à base de système multi-agents. Evidemment, il y a plusieurs domaines d'application pour les applications agents dû au fait que ces architectures (basées sur les agents) fournissent une manière bien particulière d'aborder des problèmes rapidement. C'est pour cette raison que les agents sont largement utilisés dans les domaines les plus fréquemment étudiés (énergie, industrie, communication, information santé,...).

Une des parties la plus importante dans cette étude réside sur la mise en place d'un seuil pour les ressources de chaque groupe. Nous avons proposé une solution qui maximise la répartition des charges localement, sur chaque groupe, et réduit autant que possible le besoin d'une redistribution de la charge du travail entre les noeuds du réseau. De plus, cette méthode minimise les coûts de communication en maximisant la répartition entre et à l'intérieur de chaque groupe du réseau.

Les groupes adaptent leur comportement en fonction des conditions de charge (dues aux agents) et de leur seuil respectif. Ce système a une organisation autonome tant que de nouveaux groupes peuvent se joindre au réseau, leur puissance de calcul sera rapidement exploitée pour continuer le calcul.

L'avantage d'une telle structure réside dans le fait qu'elle présente une architecture complètement décentralisée.

Si on prend l'approche du clonage d'agents [1], elle facilite la répartition des charges à travers l'utilisation d'agents entremetteurs. Ceci permet aux agents de trouver des agents sous chargés pour lesquels ils peuvent se cloner (migrer). Néanmoins, cette approche n'est pas appropriée pour les systèmes dits "peer-to-peer" et les systèmes de calcul de Grid car ils imposent un degré de centralisation.

De plus, si on prend par exemple le système MOSIX [20], système utilisant un algorithme de répartition des charges centralisé et probabiliste, le problème est qu'il nécessite une connaissance du système distribué et du système de migration de processus au niveau du noyau (kernel), ce qui n'est pas souhaitable dans des environnements hétérogènes.

Au final, l'utilité et de l'efficacité des systèmes multi-agents, et surtout des agents, ressurgissent avec la répartition des charges. En effet, le modèle organisationnel mis en place par nos soins, basé sur les SMA, fait ressortir d'autant plus fortement l'impact des agents et des groupes pour la répartition de charges au sein d'un réseau.

Le fait le plus important, et il ne faut surtout pas l'oublier, est que ces agents sont des entités **autonomes**. Ils permettent de répondre immédiatement aux changements de situation sans devoir compter constamment sur des instructions manuelles. Ainsi, pour une répartition des charges, ces agents vont pouvoir, comme expliqué dans ce rapport, s'adapter en fonction des changements que les groupes subiront, ce qui représente une fonctionnalité importante dans le cadre de cette étude.

Comme expliqué en début de ce rapport, les répartiteurs de charge peuvent être décrits selon deux critères : une répartition des charges statique ou une répartition des charges dynamique. Dans notre étude, on a essentiellement traité la répartition d'un point de vue statique mais sans oublier qu'il existe une approche dynamique. De ce fait, les agents peuvent être interprétés comme des *tâches* mais ceci n'est uniquement dû qu'au fait que le côté dynamique n'est pas représenté ici. Ceci est précisé afin de comprendre que nous n'avons pas oublié cette possibilité et que cette dernière sera plus amplement étudiée par la suite.

Dans l'état actuel des choses, la charge de calcul est attribuée au début du calcul et n'est plus modifiée tout au long de la simulation, seul le taux d'utilisation des ressources des groupes varie. Beaucoup d'applications présente un déséquilibre de charge et il est bon de pouvoir le gérer. On notera également que les machines elles mêmes peuvent aussi engendrer un déséquilibre de charge, ce qui peut être géré par l'emploi des deux seuils différentiels présentés au préalable dans ce rapport.

Le but du travail restant sera donc de créer une extension à cette étude afin de lui apporter la possibilité de gérer ces déséquilibres de charge de manière dynamique et ainsi devenir optimale.

6 Annexes

Les simulations présentées ci-dessous ont été effectuées sous le logiciel MadKit et concerne l'emploi d'un seuil fixe. Ceci a été choisi dans le but de pouvoir comprendre plus facilement le fonctionnement de ce système. D'autres simulations avec les autres seuils seront réalisés par la suite.

6.1 Capture d'écran 1

Sur cette capture d'écran, tirée de la simulation réalisée sous le logiciel MadKit, on peut voir le déroulement de l'algorithme de répartition avec 100 groupes et 200 agents. Au départ, on lance l'*Agent AllocationHello* et l'*Agent startInput*. Une fois fait, on rentre le nombre de groupes et d'agents que l'on souhaite voir agir (ici 100 et 200). Une fois fait, on choisit le seuil que l'on veut définir pour le réseau (fixe, différentiel ou différentiel variable). Le choix du seuil effectué (ici *fixe*), l'algorithme se lance et réparti les charges entre les groupes comme expliqué plus haut. C'est ce que l'on peut voir sur la Figure 17. Après, on fait l'*Analyse* et on voit le nombre de groupes correctement réparti, ainsi que leur charge moyenne (Figure 18).

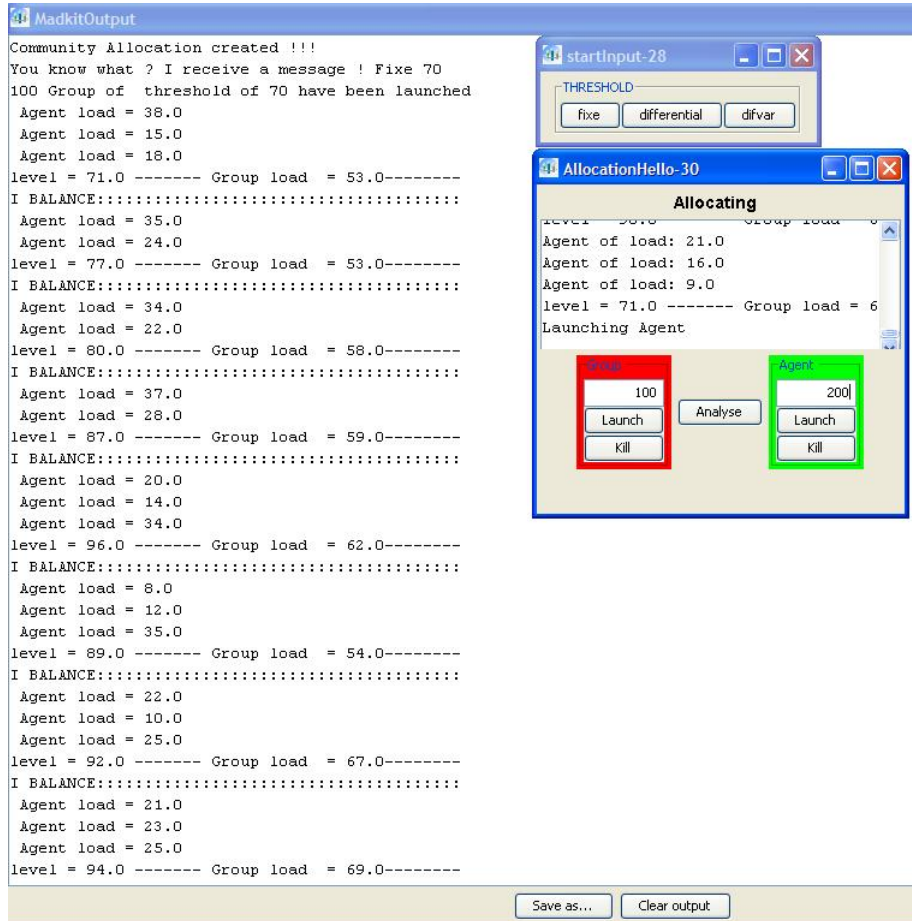


FIG. 17 – Réseau à 100 groupes et 200 agents

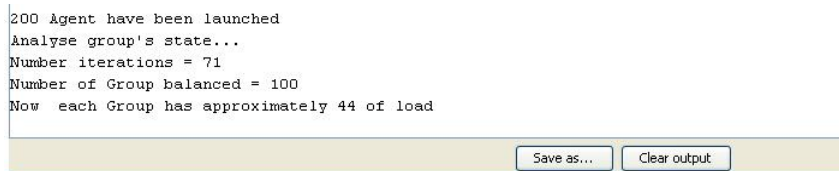


FIG. 18 – Après Analyse (100G, 200A)

6.2 Capture d'écran 2

Pour cette simulation, le système reste le même, excepté que l'on est en présence du cas où des groupes vont être créés afin de soulager le réseau (50 groupes, 200 agents). Une fois l'*Analyse* effectuée, vu que l'on a dû créer des groupes nécessaire à une bonne répartition, une deuxième analyse se fait qui va finaliser la répartition sur le réseau (Figure 19 et Figure 20 de la page suivante).

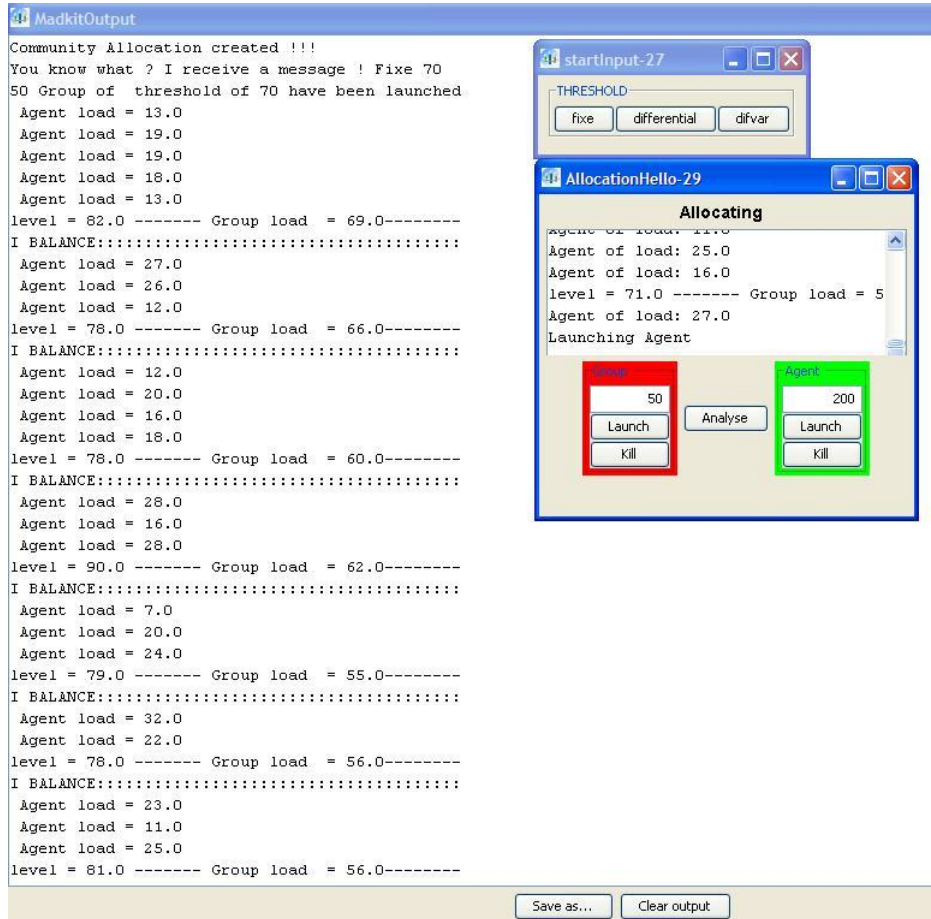


FIG. 19 – Réseau à 50 groupes et 200 agents

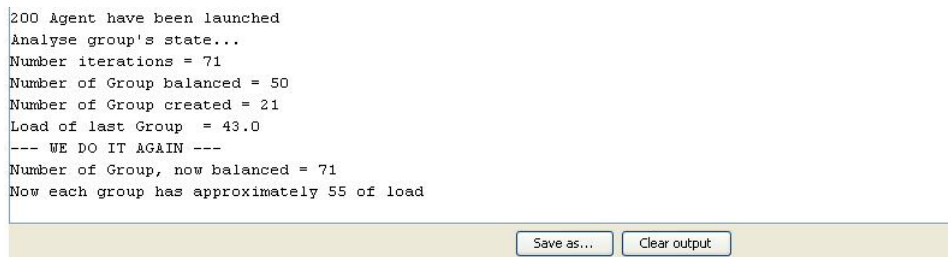


FIG. 20 – Après Analyse (50G, 200A)

Références

- [1] O.Shehory, K.Sycara, P.Chalasani, and S.Jha. Agent Cloning : An Approach to Agent Mobility and ResourceAllocation. *IEEECommunicationsMagazine*,36(7) :58-67, July1998.
- [2] M.Hohlfeld, B.Yee. How to migrate Agents. Technical Report CS98-588, Computer Science and Engineering Department, University of California at San Diego, La Jolla, CA, August 1998.
- [3] Jamal Faik, Joseph E.Flaherty, Luis G.Gervasio, James D.Teresco, Karen D.Devine. A model dor resource-aware load balancing on heterogeneous clusters. Williams College Department of Computer Science Technical Report CS-05-01, 2005.
Supercedes Williams College Department of Computer Science Technical Report CS-04-03.
- [4] R.Lfiling and B. Monien. Load balancing for distributed branch bound algorithms. Intern. Par. Processing Symp., IPPS 1992.
- [5] S.Muthukrishnan and R.Rajaraman, An adversarial model for distributed dynamic load balancing, in Proceedings of the 10th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'98), 1998, pp. 47–54.
- [6] K.Moizumi and G. Cybenko. The travelling agent problem. Technical report, Dartmouth College, Hanover, NH, February 1998.
- [7] R.Lüling, B.Monien, and F.Ramme. Load balancing in large networks : A comparative study. In *3rd IEEE Symposium on Parallel and Distributed Processing (SPDP'91)*, 1991.
- [8] T.Decker, R.Diekman, R.Lüling and B.Monien. Towards developing universal dynamic mapping algorithms. In *7th IEEE Symposium on Parallel and Distributed Processing (SPDP'95)*, 1995.
- [9] A.Schaerf, Y.Shoham, and M.Tennenholtz. Adaptive load balancing : a study in co-learning. In *IJCAI-95 Workshop on Adaptation and Learning in Multiagent Systems*, 1995.

- [10] C.Bernon, C.Bétourné, and A.Sayah. Placement et migration de processus communicants. In *Journées de recherche sur "Le placement dynamique et la répartition de charge : Application aux Systèmes répartis et parallèles"*, 1995.
- [11] S.Appleby and S.Steward. Mobile software agents for control in telecommunications networks. *Telecom Technol. Journal*,12 :104-113,1994.
- [12] J.Ferber. *Multi-Agent Systems : An Introduction to Distributed Artificial Intelligence*. Addison Wesley Longman, Harlow, UK, 1999.
- [13] J.Ferber. *Les systèmes multi-agents*. InterEdition, Paris. ISBN 2-7296-0572-X.
- [14] J.Ferber, F.Michel, J.Baez. AGRE : Integrating Environments with Organizations. in D. Weyns, V. D. Parunak, F. Michel (eds). *Environment for Multi-Agent Systems. Revised selected papers of E4MAS 2004*. LNAI. Vol. 3374. pp3-52. 2005.
- [15] J.Ferber, O.Gutknecht, F.Michel. From Agents to Organisations : an organizational view of multi-agentsystems, paper submitted to AAMAS 2003.
- [16] F.Michel, P.Bommel and J.Ferber. Simulation distribuée interactive sous MadKit, in JFIADSMA'02, Mathieu, P. andMüller, J.-P. Hermès, Lille, pp. 175-178, 2002.
- [17] J.Ferber, O.Gutknecht. Operational Semantics of a Role-Based Agent Architecture, Proc. of ATAL 99 Workshop, Orlando, 1999.
- [18] J.Ferber, O.Gutknecht. A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems, Proceedings of the third international conference on multi-agent systems. IEEE Press Paris 1998.
- [19] Logiciel MadKit. [http ://www.madkit.org](http://www.madkit.org)

- [20] A.Barak and A.Shiloh. A Distributed Load Balancing Policy for a Multi-computer. *Software Practice and Experience*. 15(9) :901-913,Sept.1985.

Résumé

Ce document traite de la répartition des charges dans des Systèmes Multi-Agents à travers un modèle organisationnel. Il présente un méta-modèle générique des systèmes multi-agents basés sur des concepts organisationnels tels que les groupes, les rôles et les structures.

La répartition des charges tiens une place importante dans l'informatique depuis de nombreuses années. L'utilisation efficace des calculs en parallèle nécessite que le calcul soit divisé entre les différents processeurs. Ce problème de répartition de charges apparaît dans de nombreuses études et tient une place prépondérante dans le secteur de la recherche depuis les dix dernières années et même plus. Dans le but d'améliorer les performances du système, nous proposons une nouvelle politique de répartition de charges basée sur l'emploi d'un seuil. Cette notion de seuil va permettre de faciliter la répartition et permettra de résoudre des problèmes en accord avec les agents. Une simulation a été développée pour étudier les performances du système avec la politique de répartition des charges proposée. Ce système a été implémenté sous la plateforme MadKit [19] mise en place par J. Ferber.

Mots clés : répartition des charges dynamiques, agent, Système Multi-Agent, seuil, MadKit

Abstract

This paper discusses load balancing in Multi-Agent Systems (MAS) through an organisational model. It presents a meta-model of multi-agent systems based on organisational concepts such as groups, roles and structures.

Load balancing takes a very important place in computer science for several years. Effective use of a parallel computer requires that a calculation be carefully divided among the processors. This load balancing problem appears in many guises and has been a fervent area of research for the past decade or more. In order to improve the system performance, we propose a novel threshold-based load balancing policy. This notion of threshold helps the load balancing and allows to solve problems in accordance with agents. A discrete event computer simulator is developed to study the system performance with the proposed load balancing policy. This system has been implemented under the MadKit platform created by J. Ferber.

Key words : dynamic load balancing, agent, Multi-Agent Systems, threshold, MadKit