# Réseaux pair-à-pair et simulation distribuéeApplication à la simulation multiagent

Adnane Cabani

# INSTITUT NATIONAL DES SCIENCES APPLIQUÉES DE ROUEN

Laboratoire d'Informatique, de Traitement de l'Information et des Systèmes


# THÈSE

pour obtenir le titre de

**Docteur en Informatique**

---

## Réseaux pair-à-pair et simulation distribuée

### Application à la simulation multiagent

---


présentée par

**Adnane CABANI**


soutenue le 07/12/2007 devant le jury composé de

| | | |
|---|---|---|
| C. Bayrak | UALR, Etats-Unis | Rapporteur |
| Y. Pollet | CNAM, Paris | Rapporteur |
| S. Espié | INRETS, Paris | Examinateur |
| S. Ramaswamy | UALR, Etats-Unis | Examinateur |
| M. Itmi | INSA, Rouen | Co-directeur de thèse |
| J.-P. Pécuchet | INSA, Rouen | Directeur de thèse |

A mes parents,
A mes frères.

# Remerciements

Le travail de recherche présenté dans ce mémoire a été mené au sein de l'équipe Perception Système et Information (PSI - FRE CNRS 2645) qui est devenu partie intégrante du Laboratoire d'Informatique, de Traitement de l'Information et des Systèmes (LITIS EA 4108) depuis mars 2006. Je suis heureux d'avoir l'occasion de présenter ma reconnaissance et ma gratitude à tous ceux qui m'ont soutenu de près ou de loin pour mener à bien la réalisation de cette thèse.

Je remercie Jean-Pierre Pécuchet, professeur à l'INSA de Rouen, d'avoir accepté d'être mon directeur de thèse. Je remercie vivement Mhamed Itmi, maître de conférences et habilité à diriger des recherches, d'avoir co-dirigé ma thèse, et pour sa disponibilité, ses conseils et ses encouragements.

Je remercie Monsieur Yann Pollet, Professeur titulaire de la chaire d'intégratiojn des Systèmes au Conservatoire National des Arts et Métiers, et Monsieur Coskun Bayrak Professeur au département d'informatique à l'Université d'Arkansas à Little Rock - Etats-Unis d'avoir bien voulu rapporter sur mon travail de thèse.

Je remercie Monsieur Stéphane Espié, Directeur du laboratoire Modélisation, Simulation et Simulateurs de conduite (MSIS) à l'Institut National de Recherche sur les Transports et leur Sécurité, d'avoir bien voulu participer à mon jury de thèse. Je lui exprime toute ma gratitude pour m'avoir donné la possibilité de travailler durant deux mois avec son équipe en qualité de vacataire de recherche. Je remercie particulièrement, Jean-Michel Auberlet pour sa disponibilité et ses aides durant mon séjour au sein du laboratoire MSIS. Je remercie aussi toute

l'équipe pour son accueil chaleureux, en particulier : Arnauld Doniec, Fabrice Vienne, Laetitia Bonte et Vincent Airault.

J'exprime ma gratitude à Monsieur Srinivasan Ramaswamy, professeur et directeur du département informatique à l'Université d'Arkansas à Little Rock - Etats-Unis, de participer à mon jury et pour m'avoir invité à son établissement lors des échanges INSA-UALR. Je le remercie aussi pour sa disponibilité et ses aides précieuses de même que Monsieur Remzi Seker, Maître de conférences, et Madame Nancy Rea, Administrative Office Supervisor, pour leur disponibilité durant mon séjour à l'Université d'Arkansas.

Je remercie particulièrement Aziz Bensrhair, professeur à l'INSA de Rouen, et Nathalie Chaignaud, maître de conférences à l'INSA de Rouen, pour leurs encouragements.

J'adresse mes remerciements à toutes les personnes de l'équipe LITIS ainsi qu'à l'ancienne équipe PSI : Brigitte Diarra, Patricia Hambourier, Sylvianne Henocq, Sandra Le Bras, secrétaires de l'équipe qui se sont succédées, et Jean-François Brulard, administrateur réseau pour leur accueil et leur gentillesse.

Merci à Foued Al-Amri, Sami Al-Maqtari, Waled Al-Shabi, Xu Jin et Alain Loisel, pour les moments agréables passés ensemble et pour l'ambiance joviale de travail.

Merci à Vimal Athithan, GuruPrasad M Hegde, Vinay Rkraj, Jabir Shayma, Sithu Sudarsan, Kishore Yelupula et Chuanlei Zhang pour leur accueil, leur disponibilités et leurs aides durant mon séjour au Etats-Unis.

Ma pensée ne peut oublier de citer particulièrement, Arafat Sassi et sa femme Sameh ainsi que Majdi Mnejja pour leurs aides et leurs disponibilités.

Merci à mes amis : Mohamed Aziz, Mohamed Ali Belaïd, Sabeur Bouhbila, Charlène Floch, Mohamed Garès, Mustapha Harmazi, Simon Ranjith et Besma Zeddini ainsi qu'à tous les membres de l'association Jeunes-Science et Planète Sciences pour leurs encouragements.

Enfin, je tiens à remercier particulièrement toute ma famille pour leurs sacrifices et leur soutien inconditionnel.

A tous mon infinie gratitude.

# Table des matières

# Table des figures

# Liste des tableaux

# 1 Introduction

Avec l'évolution technologique dans divers domaines et la nécessité de simuler des situations dynamiques complexes que ce soit pour des raisons économiques ou sécuritaires (par exemple simulation d'une centrale nucléaire), les besoins en termes de ressources informatiques se font ressentir de plus en plus vue la masse colossale de données à traiter. La simulation distribuée est une solution qui permet d'exploiter différentes ressources informatiques pour le calcul. La distribution permet d'une part des gains de temps de calcul et d'autre part de faire interagir différents simulateurs géographiquement distribués. L'évolution technologique (web, plates-formes multiagents...) et la baisse des coûts des matériels oeuvre également dans ce sens.

Parmi les solutions possibles, on peut faire fédérer plusieurs machines interconnectées par un réseau. On parle alors de grappe ou *cluster* lorsque les machines sont localisées sur un même site. Si les ordinateurs sont répartis géographiquement sur différents sites, on parle alors de grille de calcul ou *grid computing*. Une telle infrastructure permet de fédérer un ensemble de grappes. Cette solution n'est pas toujours accessible car généralement, il faut un compte

utilisateur protégé par mot de passe pour être autorisé à utiliser la grille. De plus, vu que les ressources sont partagées, il faut réserver un créneau horaire à l'avance. Ce type de solution est basé sur une architecture Client/Serveur. Aucune communication directe entre les noeuds n'est possible. Généralement, cette solution est adopté pour des problèmes pouvant être décomposés en tâches indépendantes.

De nos jours, un nouveau paradigme est apparu. Il s'agit des réseaux dits pair-à-pair (*Peer-to-Peer*). Il vient compléter et améliorer l'architecture Client/Serveur. Avec ce nouveau paradigme la relation entre les clients et les serveurs n'est plus verticale. Ces derniers sont appelés pairs. Ils sont considérés à égalité et peuvent communiquer directement entre eux. Jusqu'à présent, les systèmes pair-à-pair ont été utilisés pour le partage de fichiers dans certaines applications collaboratives ou pour le partage de ressources à des fins de calcul. Avec les nouvelles possibilités offertes par les systèmes pair-à-pair, de nouvelles opportunités se présentent offrant l'amélioration des performances des simulations distribuées.

Les infrastructures pair-à-pair sont, par définition, formées par des pairs extrêmement hétérogènes et dynamiques. En effet, les machines connectées sur l'Internet sont équipées de processeurs et de mémoires très différents. La bande passante est également très variable. Cette hétérogénéité peut être pénalisante pour des simulations collaboratives distribuées car cette asymétrie peut ralentir l'évolution de la simulation. Si de plus, on sait que chaque pair peut joindre ou quitter le réseau sans aucun préavis, cela peut influer directement les résultats de la simulation. Par conséquent, il faut bien tenir compte des spécificités des systèmes pair-à-pair afin de ne pas compromettre le bon fonctionnement des simulations distribuées.

Jusqu'à présent, on ne connaît aucune plate-forme reposant sur le modèle pair-à-pair et permettant de réaliser une simulation multiagent distribuée. Le but de notre travail consiste à fournir une plate-forme permettant de distribuer une

telle simulation tout en tenant compte de la spécificité des réseaux pair-à-pair.

Nous proposons la plate-forme PHAC (*A P2P-based Highly Available Computing Framework*) qui permet de prendre en compte l'hétérogénéité et la volatilité des pairs. Il s'agit de créer un fichier journal sur chaque pair. Le fichier comporte des données sur les caractéristiques et la connectivité du pair à l'Internet. Ce fichier sera utile pour le choix des pairs et leurs affectations au groupe de pairs spécifiques. On distingue deux groupes appelés *JobPeerGroup* et *RedundancyPeerGroup*. Le premier groupe a pour rôle d'effectuer les calculs. Les pairs sont choisis de manière à assurer leurs présence sur le réseau durant les calculs et avec la meilleure configuration possible. Le deuxième groupe a pour rôle de répliquer les données. Ceci est utile au cas où un pair quitterait le réseau, il ne sera pas nécessaire de reprendre le calcul à zéro mais de le continuer en récupérant les données répliquées. L'organisation des noeuds est totalement décentralisée. Les pairs peuvent communiquer directement les uns avec les autres (par définition même du modèle pair-à-pair).

### Organisation du document

Ce document est organisé en cinq chapitres et structuré en deux parties. La première partie, composée de trois chapitres, présente le contexte de notre étude. Le premier chapitre définit les principes généraux de la simulation basée sur l'utilisation des architectures distribuées ainsi que leurs potentiels et leurs limites. Le deuxième chapitre présente les réseaux de pair à pair, leurs topologies, caractéristiques et classifications ainsi qu'une comparaison entre les grilles de calculs et les réseaux pair-à-pair. Une brève introduction à la plate-forme de développement JXTA y est décrite. Le troisième chapitre présente les principes généraux des systèmes multiagents et par ailleurs l'utilisation des systèmes distribués dans la simulation multiagent.

La seconde partie de cette thèse présente les méthodes et les outils développés

pour la simulation distribuée. Le chapitre quatre décrit notre modèle de plate-forme PHAC. Dans le dernier chapitre, nous présentons les résultats obtenus en utilisant notre plate-forme PHAC. Finalement, nous tirons les conclusions de notre étude et traçons un aperçu des travaux futurs qui s'inscrivent en perspectives dans la continuité de ces travaux de recherches.

## Publications

Les travaux que nous présentons dans ce document ont engendré des publications dans diverses conférences et journaux internationaux avec comité de lecture :

### Chapitre de livre

[1] A. Cabani, S. Ramaswamy, M. Itmi, S. Al-Shukri, J.P. Pécuchet, "Innovations and Advanced Techniques in Computer and Information Sciences and Engineering", Chapitre Distributed Computing Systems : P2P versus Grid Computing Alternatives, pp. 47-52, Springer Netherlands, 2007.

[2] A. Cabani, S. Ramaswamy, M. Itmi, and J. P. Pécuchet, "Distributed Computing and Internet Technology", LNCS4882, chapitre PHAC : An Environment for Distributed Collaborative Applications on P2P Networks, pp 240-247, Springer Berlin / Heidelberg, 2007.

### Articles dans des revues internationales

[3] S. Ramaswamy, R. Seker, S. Sudarsan, M. Itmi, A. Cabani, and W. Alsahbi, "Modeling and Simulation : The basis for education enrichment and software systems design," in Journal of Enterprise Information Systems, (à paraître).

[4] A. Cabani, S. Ramaswamy, M. Itmi, and J. P. Pécuchet, "PHAC : A P2P-based Environment for Distributed Collaborative Applications," in The

International Journal of Intelligent Control and Systems, (IJICS), Vol. 12, No.3, September 2007 ; pp 265-273, 2007.

**Conférences internationales avec comité de lecture**

[5] M. Itmi, A. Cabani, and J.-P. Pécuchet, "Simulation distribuée et gestion du temps," in 5ème conférence francophone de MOdélisation et SIMulation (MOSIM'04), Nante, France, pp. 1013-1019, 2004.

[6] A. Cabani, M. Itmi, and J.-P. Pécuchet, "Multiagent Distributed Simulation : Discussions and prototyping a P2P architecture," in Computer Simulation Conference (SCSC'05), Philadelphia, Pennsylvania, United States, pp. 281-286, 2005.

[7] A. Cabani, M. Itmi, and J. P. Pécuchet, "Improving Collaborative Jobs in P2P Networks," in 12th IEEE International Conference on Electronics, Circuits and Systems (IEEE ICECS'05), Gammarth, Tunisia, pp. 135-138, 2005.

[8] A. Cabani, M. Itmi, and J. P. Pécuchet, "Un prototype de simulation multiagents distribuée sur une architecture P2P," in 6ème conférence francophone de MOdélisation et SIMulation (MOSIM'06), Rabat, Maroc, pp. 326-321, 2006.

[9] A. Cabani, S. Ramaswamy, M. Itmi, S. Al-Shukri, and J. P. Pécuchet, "Distributed Computing Systems : P2P versus Grid Computing Alternatives," in International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering (CISSE'06), 2006.

[10] A. Cabani, S. Ramaswamy, M. Itmi, and J. P. Pécuchet, "PHAC : an Environment for Distributed Collaborative Applications on P2P Networks," in 4th International Conference on Distributed Computing and Internet Technology (ICDCIT'07), LNCS 4882, Bangalore, India, pp. 240-247, 2007. (Taux d'acceptation 20%)

[11] A. Cabani, M. Itmi, and J. P. Pécuchet, "Distributed Multiagent Simulation on P2P Architecture," in 11th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (IEEE/ACM DS-RT'07) Chania, Crete Island, Grece, pp. 76-79, 2007. (Taux d'acceptation 33%)

# Première partie

# Contexte d'étude

# 2

# Modélisation et simulation distribuée collaborative

## Sommaire



DANS ce chapitre, nous présentons les principes généraux de la simulation basée sur l'utilisation d'architectures distribuées ainsi que leurs potentiels et leurs limites. Après une brève introduction, nous exposons les différentes classes et méthodologies de la simulation. Nous présentons ensuite les différentes architectures de distribution d'une simulation. Dans la section 4, nous décrivons les différents mécanismes de distribution permettant de répartir l'exécution d'une

simulation ainsi que les différents modèles de programmation répartie et la gestion du temps avec les simulations distribuées. La section 5 présente différents protocoles de simulation avant de conclure le chapitre par une synthèse.

## 2.1 Généralité

Une simulation est définie comme *« the imitation of the operation of a real-world process or system over time »* (Banks *et al.*, 2000) tandis que le modèle est défini comme la représentation du système étudié.

Pour Quéau (Queau, 1986), *« la simulation consiste à mettre en œuvre des modèles dans des conditions variées, pour tenter d'explorer leurs possibilités, leurs défaillances, et éventuellement découvrir telle ou telle trajectoire comportementale encore inconnue »* .

L'étude des comportements de modèle indépendamment du temps constitue une simulation. Cela peut avoir plusieurs avantages. La modélisation et le développement de système permettent de prédire le comportement et donc améliorer les performances par l'utilisation de la simulation au lieu du développement physique. Ceci peut être utile surtout pour les systèmes où l'on ne peut pas tester le modèle dans la réalité. On peut citer par exemple : les stratégies militaires, les équipements de sécurité... Ainsi, la simulation peut être une excellente alternative.

A titre d'exemple, dans la pratique, la modélisation et la simulation peuvent inclure :

- Modélisation et test de technologie : peut être appliqué à la création et l'évaluation de différents types de réseaux.
- L'étude de systèmes complexes : corps humain, conditions météorologiques...
- Formation et éducation : simulateur de vol, conduite de voiture...

La taille des simulations est de plus en plus grandes d'où la nécessité de les distribuer. L'évolution technologique (web, plates-formes multiagents...), la baisse des coûts des matériels ainsi que les exigences temporelles quant à l'obtention des résultats de simulation œuvrent également dans ce sens. Ceci a entraîné l'ouverture d'un axe de recherche fortement orienté vers le domaine de la simulation distribuée.

Lorsqu'une simulation fait intervenir un environnement collaboratif virtuel (*Collaborative Virtual Environments*), on parle alors de simulation collaborative. L'environnement peut être des scènes 3D partagées plus communément connue sous le nom de mondes virtuels.

## 2.2 La simulation

### 2.2.1 Classes de simulation

Vu les nombreuses et différentes caractéristiques des systèmes, plusieurs types de classification des simulations existent. En général, les classes de simulation se réfèrent au type du modèle simulé. Une simulation peut être statique, dynamique, déterministe, stochastique, etc. Néanmoins, une classification exclusive peut être faite en prenant en compte l'évolution du système représenté par rapport au temps. Ainsi, il existe deux principaux types de systèmes :

- Les systèmes statiques dont l'état ne dépend que des paramètres courants. De tels systèmes réagissent instantanément. Ils ne possèdent pas de mémoire puisque le passé n'influence pas l'état présent. Ils sont indépendants du temps.
- Les systèmes dynamiques dont l'état dépend à la fois des paramètres courants et des états passés. Le système peut évoluer dans le temps de deux manières : de façon continue (les variations d'états interviennent sans interruption dans le temps), ou bien de façon discrète (les changements d'états se produisent d'une manière discontinue ou ponctuelle).

Ernest Page (Page, 1994) quant à lui a classé la simulation en trois catégories selon l'évolution des états simulés :

- Simulation de type Monte-Carlo : c'est une méthode dans laquelle un problème est résolu par un processus stochastique et dans laquelle une représentation explicite du temps n'est pas nécessaire.

- Simulation continue : c'est le cas où le système se présente sous forme d'équations différentielles à résoudre. Elle peut être une alternative à ce type de problème surtout lorsque la résolution analytique de ces dernières s'avère impossible. Historiquement, la résolution était effectuée sur des machines analogiques et, depuis 1975, sur des simulateurs hybrides développés sur des ordinateurs numériques. La partie calculateur analogique est résolue par un calcul numérique alors que la partie calcul événementiel et séquentiel est résolue par un programme informatique déclenché par un événement analogique (*trigger*).
- Simulation à événements discrets : dans cette classe de simulation, le système est soumis à une succession d'événements qui le modifient. On distingue deux grandes familles :
  - Dirigé par le temps (ou *time-slicing*) : la notion de pas de temps est introduite. A chaque incrémentation de ce dernier, le simulateur est exécuté. Seules les actions concernées par cette date sont exécutées. Il est possible qu'aucune action ne soit concernée par cette date.
  - Dirigé par les événements (ou *event-sequencing*) : les événements sont exécutés les uns à la suite des autres indépendamment du temps. Cette méthode est plus efficace que celle dirigée par le temps, particulièrement lorsque dans un système les événements sont peu fréquents. Elle est difficile à mettre en œuvre.

### 2.2.2 Méthodologie

Différentes méthodologies peuvent être utilisées pour la conception de modèles destinés à la simulation informatique. Gilbert et Troitzsh (Gilbert and Troitzsch, 2005) présentent quelques modèles dont la dynamique des systèmes, les modèles à files d'attente, les automates cellulaires ou encore l'approche agent.

**La dynamique des systèmes** constitue une méthodologie développée à partir d'équations différentielles. Elle est conçue pour la simulation des systèmes continus.

**Modèles à files d'attente** souvent appelés modèles à événements discrets représentent un système en fonction de ses entités, propriétés, ensemble d'événements, activités et délais. Cette méthodologie procède par événements successifs. Les événements sont estampillés par le temps. Ils sont ordonnés dans une liste par ordre croissant qui contient les événements futurs à exécuter. Une fois un événement est exécuté, il est retiré de la liste. Aucune action ne peut se produire entre deux événements. Les événements sont générés suite à d'autres événements ou par une source. Bernard Zeigler (Zeigler, 1976) a développé une méthodologie qui porte sur la modélisation et la simulation de systèmes à événements discrets. Il a introduit à un nouveau formalisme abstrait et universel pour la modélisation à événements discrets appelé *Discrete Event System Specification* (DEVS).

**Les automates cellulaires** décrivent l'espace sous la forme d'un réseau pouvant être représenté par un ensemble de sites reliés entre eux par un graphe de voisinage de portée unitaire. Un automate cellulaire consiste en une grille de « cellules » pouvant chacune prendre à un instant donné un « état » parmi un ensemble fini. Le temps est également discret et l'état d'une cellule au temps $t$ est fonction de l'état au temps $t-1$ d'un nombre fini de cellules appelé son « voisinage ». À chaque nouvelle unité de temps, les mêmes règles sont appliquées pour toutes les cellules de la grille, produisant une nouvelle « génération » de cellules dépendant entièrement de la génération précédente. Un des plus célèbres automate cellulaire est le jeu de la vie de Conway mis en avant par Gardner (Gardner, 1970) et qui semble avoir initié l'ère des simulations informatiques.

**Les systèmes multiagents** visent à appréhender la coordination de processus autonomes. Pour Weiss (Weiss, 1999), un agent est ainsi une entité

computationnelle (un programme informatique ou un robot) qui peut être vue comme percevant et agissant sur son environnement au sujet duquel on peut parler d'autonomie parce que son comportement dépend au moins partiellement de son expérience. Un système multiagent est constitué d'un ensemble de processus informatiques se déroulant en même temps, donc de plusieurs agents vivant un même moment, partageant des ressources communes et communicant entre eux.

## 2.3 La simulation distribuée

### 2.3.1 Objectifs

Deux raisons majeures peuvent motiver le choix de distribuer une simulation :

- Ressources système limitées
- Temps d'exécution élevé

Dans le cas du premier point la distribution permet en général d'atteindre l'objectif attendu c'est-à-dire permettre d'effectuer la simulation de plus grands modèles. Ce n'est pas toujours le cas pour le second point. En effet, la répartition de la simulation ne permet pas de garantir de meilleurs temps de calcul. Ceci est dû aux coûts supplémentaires des communications pour maintenir la causalité. Le choix de l'architecture, de la technique de distribution et des protocoles de synchronisation est déterminant pour assurer de meilleurs résultats. Ceci reste une tâche particulièrement difficile.

### 2.3.2 Architecture de simulation

Les simulations distribuées peuvent être construite sur deux types d'architectures :

- Les architectures centralisées
- Les architectures distribuées

Nous aborderons dans ce qui suit ces deux architectures et nous montrerons que les avantages de l'une sont les inconvénients de l'autre.

### Architecture centralisée

Dans ce type d'architecture, les calculs sont gérés par le serveur. Ce dernier a essentiellement pour rôle de s'occuper de l'état de cohérence générale puisque tout le calcul se fait sur une seule machine. Il n'y a donc qu'une seule décision prise pour chaque cas de figure (voir figure 2.1).

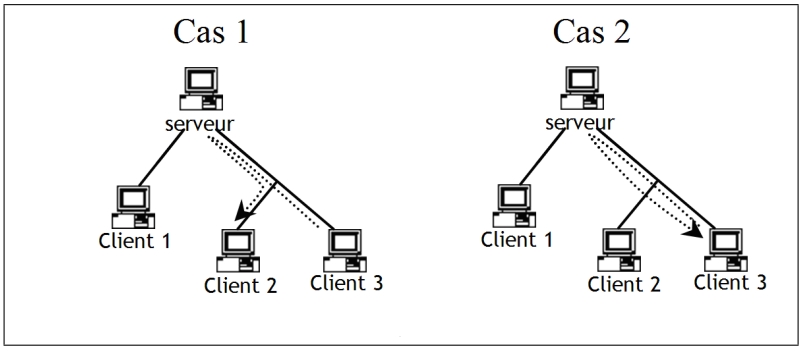



FIGURE 2.1 – Exemples de communications dans une architecture centralisée.

La mise en œuvre d'une telle architecture est assez simple. Elle est de plus sécurisée. La puissance des machines clientes n'a pas besoin d'être très élevée puisque le client ne fait que transmettre des informations et afficher les résultats calculés par le serveur. Toutes les informations proviennent du serveur et par conséquent l'architecture est sûre. Néanmoins, toute la simulation s'arrête lorsque le serveur plante. Un autre problème apparaît : il concerne la connexion des clients au serveur. Ce dernier est un goulot d'étranglement des performances car plus il y a de machines clientes plus le serveur doit gérer de connexions. Il perd donc beaucoup de temps dans les communications réseaux qui transitent entre lui et ses clients.

### Architecture distribuée

Dans ce type d'architecture (voir figure 2.2), les tâches de simulation sont distribuées entre différentes machines. Le calcul est effectué sur chacune d'elles à partir des informations échangées avec les diverses machines. L'avantage de cette architecture est sa tolérance aux pannes puisque toutes les tâches sont distribuées. Si une panne survient sur une machine, les autres peuvent continuer à tourner. De plus, il y a un gain de temps dans la communication réseaux.

Mais, rapidement des inconvénients surgissent. Il est très difficile d'avoir un état de cohérence générale. Il faut prévoir des mécanismes de synchronisations pour garder au mieux un état cohérent.

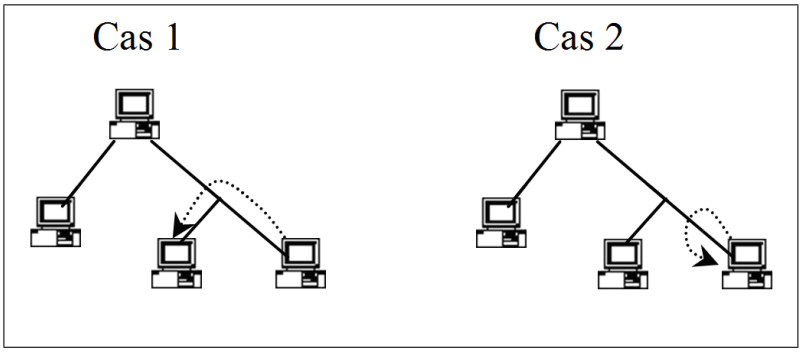



FIGURE 2.2 – Exemples de communications dans une architecture distribuée.

### Comparaison et discussion

L'architecture distribuée présente d'énormes avantages par rapport à l'architecture centralisée (voir tableau 2.1). Elle est plus stable puisque les ressources des machines clientes sont utilisées et les tâches sont réparties. À noter une marge de tolérance aux fautes appréciée vu que la simulation continue même lorsqu'une panne survient sur l'un des ordinateurs. Cela peut s'avérer intéressant dans les cas où, par exemple, les autres machines peuvent reprendre le travail de la machine défectueuse.

| | Architecture centralisée | Architecture distribuée |
|---|---|---|
| Temps d'un message | client → serveur → client | client → client |
| Limite des communications | Goulot d'étranglement du serveur | Seuls les sous-réseaux sont sollicités |
| Gestion du temps | Le serveur gère le temps global | Nécessité d'un mécanisme de synchronisation |
| Résistance aux pannes | Le serveur est le nœud critique | Pas de point faible |

TABLE 2.1 – Comparaison entre les architectures de simulation.

### 2.3.3 Technique de distribution

Dans ce qui suit on présente une liste non exhaustive des différentes techniques et méthodes existantes pour distribuer une simulation :

#### Distribution de l'application

Il s'agit d'exécuter une même simulation sur différentes machines avec différents paramètres. On parle alors d'applications parallèles. La mise en œuvre d'un tel type de distribution est assez aisée d'un point de vue développement puisqu'il s'agit d'exécution parallèle sans aucune interaction. Ce type de distribution est utilisé dans les problèmes d'optimisation et pour améliorer les qualités des résultats statistiques. On peut citer comme exemple : les programmes utilisant des algorithmes heuristiques, SETI@home...

#### Distribution fonctionnelle

Cela consiste à organiser le noyau de la simulation en modules. Chacun de ces derniers sera exécuté sur une machine. Dans le cas d'un nombre limité de modules, le gain du temps des calculs peut s'avérer limité. Par exemple, la simulation du fonctionnement d'une entreprise selon cette approche peut utiliser les différents modules : service commercial, service technique, service clientèle...

### Distribution des événements

Il s'agit de distribuer l'exécution des événements. Un processeur maître sélectionne les événements à exécuter et les distribue aux processeurs esclaves. Cette technique est appropriée pour les machines multiprocesseurs à mémoire partagée. Dans le cas d'une exécution concurrente, il faut faire attention au risque de violer la causalité [1]. Une amélioration des performances ne peut être envisagée que si la granularité des événements est forte c'est-à-dire qu'il faut beaucoup de temps pour simuler chaque événement.

### Distribution temporelle

Dans le cas où il est possible de simuler un système sur un intervalle $[T_0, T_1]$ et qu'on dispose de N processeurs, il peut être possible de diviser l'intervalle en N sous-intervalles qui seront exécutés sur les N processeurs. Lorsque la simulation est finie, on compare les extrémités des intervalles afin de vérifier qu'il y a bien égalité. Dans le cas contraire, on recalcule les conditions initiales et on relance la simulation. Dans la littérature, on trouve plusieurs études évoquant cette approche et des algorithmes de simulation ont été proposés :

- Chandy et Sherman (Chandy and Sherman, 1989) conçoivent la simulation comme le remplissage d'un rectangle espace-temps par les événements. Ils proposent de le découper au mieux et de le distribuer pour être simulé par des processus différents.
- Greenberg et al. (Greenberg *et al.*, 1990) ainsi que Bacelli et Canales (Baccelli and Canales, 1992) modélisent l'évolution du système par des équations de récurrence. Chaque processeur détermine la date d'occurrence des événements par une tranche de temps.

1. Voir la section 3.5 de ce chapitre.

#### Coopération des simulateurs

Il s'agit de faire coopérer des simulateurs séquentiels existants. Cette technique est très utilisée par les militaires américains qui font coopérer des simulateurs dédiés (simulateur de l'armée de terre, d'aviation...) afin d'obtenir la simulation complète d'une zone de combat. Les protocoles Distributed Interactive Simulation (Fujimoto, 1999) et High Level Architecture (IEEE, 1996) sont apparus afin de faciliter la mise en œuvre d'une telle technique.

#### Distribution spatiale

Une autre approche consiste à organiser le modèle en sous-modèles. Chaque sous-modèle est exécuté sur une machine. Chaque machine est responsable de sa simulation. Le système évolue d'une manière concurrente et les communications sont faites par échange de messages estampillés et portant une date de simulation. Au sein d'une même machine les événements sont traités séquentiellement tout en respectant la causalité en synchronisant les messages échangés. Cette approche exploite le parallélisme intrinsèque au modèle.

### 2.3.4 Modèles de programmation répartie

La programmation répartie se fait suivant trois modèles qu'on présente brièvement dans ce qui suit.

**Client/Serveur** C'est le modèle de programmation le plus utilisé et le plus fréquent. Il est asymétrique. Un nœud joue le rôle d'un serveur et les autres nœuds jouent le rôle des clients. Un client envoie une requête avec des arguments au serveur, ce qui déclenche une action sur le serveur. Le résultat est envoyé à l'expéditeur de la requête. C'est le modèle typique utilisé dans les serveurs web, des RPC (R*emote Procedure Call)*, des RMI (*Remote Method Invocation*) ou des *Web Services*.

**Fédération** Dans ce type de modèle, une entité centrale appelée fédérateur a pour tâche de contrôler les calculs et toutes les interactions à l'intérieur du système. Les nœuds qui se rejoignent à la fédération sont appelés des fédérés. Ces derniers peuvent quitter ou joindre la fédération en se connectant ou se déconnectant du fédérateur. Les fédérés communiquent entre eux en échangeant des messages via le fédérateur. Ce modèle est utilisé dans l'architecture HLA (*High Level Architecture*) de simulateurs.

**Pair-à-pair** Un système pair-à-pair est un système réparti où tous les nœuds sont considérés comme équivalents. Les nœuds jouent à la fois le rôle de serveur et de client. Les caractéristiques principales de la programmation pair-à-pair est la volatilité et l'hétérogénéité des pairs. Ce modèle est souvent utilisé pour le partage de données. Une étude plus détaillée de ce modèle est présentée dans le chapitre suivant.

### 2.3.5 Gestion du temps

Un des points important dans la simulation distribuée est la gestion du temps. Les services de gestion du temps assurent la cohérence générale de la simulation. Ils assurent la synchronisation des entités distribuées.

Dans la simulation distribuée, le temps est souvent divisé en deux types : temps local (*Local Time*) qui représente les différentes entités formant la simulation ; et le temps virtuel global (*Global Virtual Time*) qui représente la plus petite estampille des messages non encore traités ou en cours de transit sur le réseau. La synchronisation de ces différents composants peut être assurée par différents types d'algorithmes (Fujimoto and Weatherly, 1996; Fujimoto, 1998).

La gestion du temps permet d'éviter qu'un effet ne se produise avant une cause. C'est ce qu'on appelle communément le principe de causalité. Ceci signifie simplement que le futur ne peut influencer le passé. Or le principe de causalité n'est pas toujours garanti. L'exemple ci-dessous (figure 2.3) illustre l'explosion d'un char suite au tir d'un missile. Du fait de la latence des réseaux, un utilisateur

peut recevoir les événements dans le désordre (Fujimoto, 1999). Il pourrait observer d'abord l'explosion du char avant d'avoir vu arriver le missile !

Pour remédier à ce problème, des algorithmes de synchronisation ont été élaborés. Les premiers travaux remontent à 1970 (voir par exemple (Chandy *et al.*, 1979; Misra, 1986)). Ils font partie d'une classe d'algorithmes désignés par algorithmes pessimistes (ou conservatifs) de synchronisation. Plus tard, avec le développement de l'algorithme *Time Warp algorithm* (Jefferson and Sowizral, 1985), les bases fondamentales d'une nouvelle classe d'algorithmes ont été définies : il s'agit des algorithmes optimistes de synchronisation. Considérons à présent les techniques pessimistes et optimistes de synchronisation. Elles sont les noyaux de tous les travaux effectués sur la simulation distribuée (Fujimoto, 1999; Chen and Szymanski, 2002*a*; Chen and Szymanski, 2002*b*)

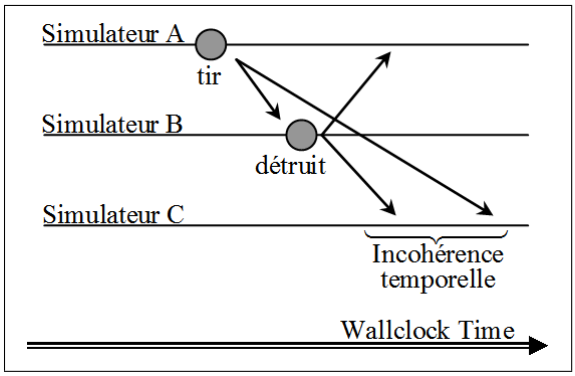



FIGURE 2.3 – Problème d'ordonnancement causal.

### Méthode pessimiste (ou conservative)

Cette première catégorie d'algorithmes est utilisée pour satisfaire les relations de causalités : dans le cas où deux événements seraient dépendants l'un de l'autre, il faut donner un ordre causal (Lamport, 1978). Il s'agit de la relation *happens-before*. Un exemple est illustré dans la figure 2.4.

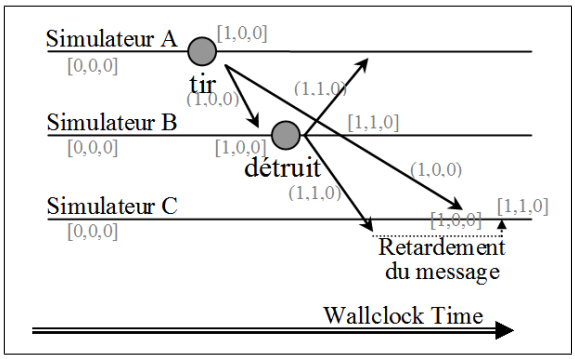



FIGURE 2.4 – Relation happens-before à l'aide d'un vecteur temps.

Pour assurer la synchronisation, on peut utiliser soit une approche synchrone dirigée par le temps, soit une approche asynchrone dirigée par les événements (Lawson and Park, 2000).

La simulation synchrone est une simulation dans laquelle une horloge globale définit un temps simulé commun. Le temps de la simulation $T_{sim}$ est à priori unique pour tous les processeurs quelle que soit la machine hôte. À chaque phase de calcul, chaque processeur exécute une action et attend que tout le monde ait terminé ses calculs avant de passer à la phase suivante. C'est une synchronisation forte.

La simulation asynchrone est une simulation où chaque processeur avance à sa propre vitesse. Les processeurs n'ont pas de vision globale. Les décisions sont prises en fonctions des informations locales. Mais avant toute exécution d'un événement estampillé au temps $t$, le simulateur doit être sûr de ne pas recevoir dans le futur un message avec une estampille plus petite en provenance des autres simulateurs. Ceci est assuré par un échange de messages avec estampille entre les ordinateurs. Afin d'éviter les interblocages, différentes méthodes sont employées par les environnements. La solution la plus courante consiste à utiliser des messages vides (Fujimoto and Weatherly, 1996), c'est-à-dire ne contenant aucun événement, afin de s'assurer de la progression des simulateurs. Ces messages vides contiennent les dates locales courantes. Une fois qu'un simulateur a au

moins un message (éventuellement vide) de chacun des autres simulateurs, il peut réordonner les événements afin de les livrer au processus en attente selon l'ordre défini par les estampilles. Afin d'éviter l'échange d'un nombre important de messages vides et donc de gagner en flux d'échange entre machines, chacune de ces dernières peut prédire une date logique d'avancement (*lookahead*).

Une machine $i$ prédit qu'elle ne va pas générer un nouvel événement et donc n'envoyer un message à la machine $j$ qu'après la date logique d'avancement $A_{ij}$. C'est la somme de la date locale ($L_i$) et le temps pendant lequel aucun événement ne sera généré ($L_a$). La machine $j$ ne recevrait donc pas de message avec une date inférieur ou égale à $A_{ij}$. De cette manière, la machine $j$ peut avancer sans attendre d'événement de la part de la machine $i$.

### Méthode optimiste

Un algorithme optimiste (Fujimoto, 2001) n'impose aucune contrainte sur le comportement des modèles. Il laisse les erreurs de causalité se produire, les détecte et les répare. Un modèle peut avancer dans le temps aussi vite qu'il le peut, consommant tous les événements présents en entrée, et en faisant le pari qu'il ne se produira aucune erreur. La vivacité est garantie car il n'y a pas d'attente. Les horloges locales sont incrémentées jusqu'à la fin de la simulation. Une violation de la contrainte de causalité est détectée lorsque l'estampille d'un événement arrivant est inférieure à l'horloge locale. Un mécanisme est alors mis en place pour annuler toutes les actions réalisées entre la date de l'événement et la date courante de l'horloge locale. Ce traitement est appelé retour arrière (*roll-back*) et le message qui déclenche le retour est appelé retardataire (*straggler*).

Pour pouvoir faire un retour en arrière, il est nécessaire de sauvegarder pour chaque hôte la liste des événements en entrée (événements traités), la liste des événements en sortie et une liste donnant l'état associé à chaque événement traité.

Lorsqu'un événement parvient en retard à un hôte, le simulateur positionne sa date locale à celle de l'événement retardataire. Il fait un retour en arrière au

moyen des données sauvegardées. Enfin, il envoie des messages d'annulation à tous les événements de la liste de sortie qui possèdent une date supérieure à celle de l'événement retardataire. L'envoi de tels messages peut engendrer un effet de cascade : un message d'annulation reçu par un simulateur peut engendrer à son tour d'autres messages d'annulations.

On peut distinguer deux types de messages d'annulations :

- Agressif : cela consiste à envoyer immédiatement les messages d'annulation.
- Paresseux : cela évite de transmettre les messages immédiatement. Il vérifie tout d'abord si le nouveau message est le même que le message à annuler. Si c'est le cas, alors il n'est pas nécessaire d'envoyer le message d'annulation.

### Discussion

Traditionnellement, on distingue deux familles d'algorithmes de synchronisation : optimistes et pessimistes. Ces derniers sont plus simple à implémenter mais souvent aux dépends des performances. Quant à l'approche optimiste, elle peut donner de meilleurs résultats. Mais, elle est assez complexe à implémenter et présente des inconvénients.

L'approche conservative ne permet l'exécution d'un événement que lorsqu'elle peut être déterminée avec certitude et que son exécution ne pourra pas produire de faute temporelle dans le futur. Totalement à l'opposé, l'approche optimiste se caractérise par son mode de fonctionnement qui ne fait aucune tentative pour éviter les fautes temporelles.

## 2.4 protocoles de simulation distribuée

Les premiers travaux ont débuté pendant les années 80. En 1983, le projet SIMNET [2] a débuté sous les auspices de l'ARPA [3]. La première démonstration a eu lieu en 1984. D'autres travaux ont suivi et ont permis d'affiner le protocole. En mars 1993, ces recherches ont donné naissance au protocole DIS [4]. Malgré son succès, DIS souffrait de nombreux manques d'extensibilité et de réutilisation. Ainsi, l'ARPA a décidé de développer une nouvelle architecture HLA [5]. Dans ce qui suit, on présente ces différents protocoles.

### 2.4.1 SIMulation NETwork (SIMNET)

A l'origine le but du système est de permettre à de petites unités militaires d'apprendre à s'organiser et à combattre en équipe. Ce système est totalement réparti. Il est fondé sur des objets et des événements. Les objets simulés interagissent entre eux via une série d'événements. Les objets statiques sont différenciés de tous les objets dynamiques. Les objets transmettent des informations concernant uniquement les changements de leur état. Ceci permet de limiter la transmission et la réception d'informations redondantes. Les objets utilisent des algorithmes de *Dead Reckoning* ou « estime » pour extrapoler leur état (voir figure 2.5). Il s'agit d'une technique prédictive qui permet de réduire le trafic du réseau. Avec cette technique tous les simulateurs gèrent des copies fantômes de tous les objets distants de façon à pouvoir prédire l'état courant grâce à un ensemble d'informations. Si la distance (par exemple euclidienne) séparant l'objet local et l'objet fantôme est supérieure à un seuil fixé alors une mise à jour est envoyée sur le réseau.

SIMNET comprend trois classes de protocoles : un protocole de simulation qui transmet les informations entre les différents simulateurs, un protocole de

2. SIMNET : SIMulator NETworking, développé par BBN, Bolt Beranek et Newman
3. Maintenant appelé DARPA, Defense Advanced Research Projects Agency
4. Distributed Interactive Simulation Standard Protocols, IEEE 1278
5. High Level Architecture

collecte de données pour la gestion de la simulation et un protocole d'association qui fournit des services de niveau transport et session au-dessus d'Ethernet.

Le protocole de simulation est constitué d'un ensemble d'unités de données de protocole (PDU [6]) qui transportent des informations d'état ou des événements.

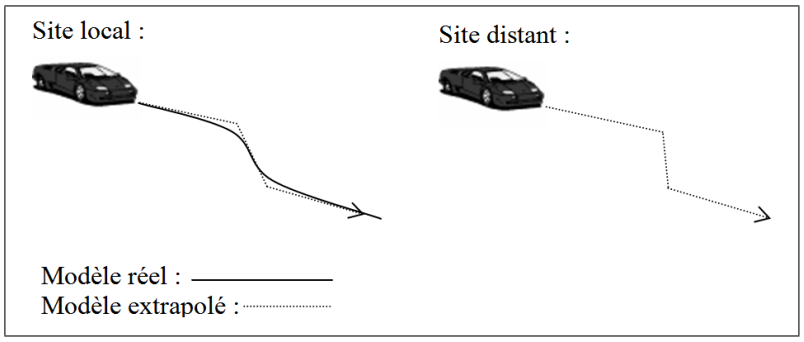



FIGURE 2.5 – Exemple d'utilisation de l'algorithme de Dead Reckoning.

### 2.4.2 Distributed Interactive Simulation (DIS)

Succédant à SIMNET, le protocole DIS (Committee, 1994) essaie d'en pallier les inconvénients. Le premier choix a été de modéliser le monde sous forme d'objets entrants en interaction les uns avec les autres à travers des événements. Le but est de permettre à plusieurs simulateurs d'interopérer.

DIS définit une architecture permettant de relier les simulations. Pour cela, les échanges de données s'effectuent de manière asynchrone et permettent d'avoir un environnement cohérent. Orientés messages, l'échange se fait sans serveur et chaque hôte doit maintenir une description complète et locale de l'environnement, des acteurs et des entités ainsi que leurs déplacements afin d'effectuer son propre rendu du monde simulé. Le fonctionnement d'un simulateur se fait suivant le paradigme « joueurs et fantômes » (*Players and ghosts*). Ce paradigme indique que chaque objet logiciel est géré sur sa propre machine hôte par un autre objet logiciel appelé joueur. Sur les machines participantes à la simulation une version

6. PDU : Protocole Data Unit

simplifiée du joueur est modélisée dynamiquement par un objet logiciel appelé fantôme. Les objets fantômes mettent à jour leur position grâce à un algorithme de *Dead Reckoning* (Fujimoto, 1999).

Le standard DIS a apporté de nombreuses innovations technologiques telles que de nouvelles approches de *Dead Reckoning* et le *Heartbeat*. Il s'agit d'émettre régulièrement des messages appelés *Heartbeat* de façon à informer les autres participants que l'objet local existe encore dans le système et doit donc être pris en compte par les machines distantes. Même si initialement la spécification du protocole s'est limitée au domaine militaire, elle a par la suite servi à d'autres domaines avec plus ou moins de succès (Bouché, 1997). La normalisation d'un format de données, par ce standard, a facilité l'interopérabilité. Néanmoins, ce format est spécifique au domaine applicatif. Il est donc nécessaire de proposer une architecture commune, indépendante du format des messages et apportant une base nécessaire à l'interaction entre différents processus. Cette réflexion a conduit au développement d'une nouvelle architecture appelée *High Level Architecture*.

### 2.4.3 High Level Architecture (HLA)

Le protocole HLA (IEEE-Std.1516.1, 2000; IEEE-Std.1516.2, 2000*a*; IEEE-Std.1516.2, 2000*b*; IEEE-Std.1516.3, 2003) concerne une architecture logicielle permettant de développer des simulations distribuées. Il a été proposé aux états-unis par le DMSO[7] du DoD[8]. HLA permet la création d'une simulation (fédération en terminologie HLA) composée elle-même de différents éléments de simulation appelés simulateurs ou fédérés (voir figure 2.6).

7. Defense Modeling and Simulation Office
8. Departement of Defense

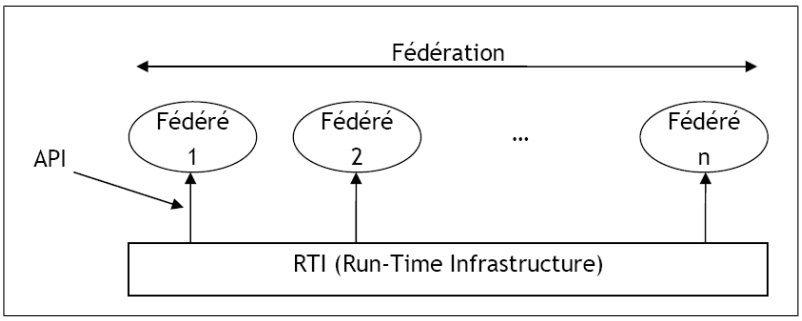



FIGURE 2.6 – Vocabulaire HLA et organisation de la fédération.

Une fédération est composée de plusieurs entités :

- n fédérés
- Un RTI (*Run Time Infrastructure*)
- Un FOM (*Federation Object Model*)

Les fédérés peuvent représenter diverses choses. Il peut aussi bien s'agir d'un élément de la simulation (comme une voiture), n éléments (toutes les voitures d'une ville), ou même n'être qu'un élément de visualisation.

Le RTI est le noyau central de la simulation. Tous les fédérés ont un point d'accès à cet élément. Le RTI est l'implémentation de la spécification HLA. Il fournit les outils logiciels nécessaires pour une simulation compatible HLA.

Il existe un unique FOM par fédération. Il présente toute l'information partagée. Il contient des données décrivant les échanges de données entre les fédérés pendant la simulation. En effet, les fédérés possèdent un certain nombre de données communes partagées au sein de la fédération. Ces données partagées sont recensées dans le FOM. On distingue deux types : les objets et les interactions. Les objets sont des informations partagées. Ils sont persistants. Alors que les interactions sont des données éphémères. Les objets et les interactions partagés dans une fédération sont définis sous forme de tables.

Les objectifs de HLA sont :

- Faciliter l'interopérabilité et la réutilisation des simulations.

- Réduire les coûts de la modélisation.
- Répondre aux besoins de la simulation distribuée.

L'architecture de HLA est composée de :

- Règles pour les fédérés et la fédération.
- Une spécification d'interface de programmation (API).
- Un patron pour les modèles d'objets appelé *Object Model Template* (OMT).
- Un FEDEP (*Federation Development and Execution Process*)/

Un jeu de dix règles est définit dans (IEEE-Std.1516.1, 2000). La fédération et chaque fédéré doivent nécessairement respecter ces règles afin d'être considérés comme conformes à la norme HLA.

On distingue cinq règles pour les fédérations :

- Les fédérations auront un FOM documenté conformément à l'OMT.
- La représentation des objets dans le FOM appartiendra au fédéré et non au RTI.
- Pendant une exécution de fédération, tout échange de données FOM parmi les fédérés se fera via le RTI.
- Pendant une exécution de fédération, les fédérés agiront réciproquement avec le RTI conformément à la spécification d'interface HLA.
- Pendant une exécution de fédération, à une date donnée, un attribut d'une instance d'un objet appartiendra seulement à un fédéré.

De même, pour les fédérés, cinq règles sont définies :

- Les fédérés ont un SOM documenté conformément à l'OMT.
- Les fédérés seront capable de mettre à jour et/ou diffuser n'importe quels attributs d'objets dans leur SOM et enverront et/ou recevront des interactions SOM extérieur comme spécifié dans leur SOM.
- Les fédérés seront capable de transférer et/ou d'accepter la propriété d'attributs dynamiquement pendant une exécution de fédération comme spécifié dans leur SOM.

- Les fédérés seront capables de varier les conditions dans lesquels ils fournissent les mises à jour des attributs d'objets comme spécifié dans leur SOM.
- Les fédérés seront capables de gérer le temps local de façon à permettre de coordonner l'échange de données avec d'autres membres d'une fédération.

La spécification d'interface définit les interfaces fonctionnelles entre les fédérés et le RTI. Elles doivent être respectées pendant l'exécution afin d'obtenir une simulation compatible à la norme HLA.

On peut définir un patron pour que les interactions et les objets gérés par un fédéré soient visibles et compris de l'extérieur de ce fédéré. Il s'agit de l'*Object Model Template* (OMT). Il fournit une norme pour documenter l'information du modèle d'objet HLA.

Le *FEDEP* représente un modèle de développement de fédérés et de fédérations basé sur les modèles en cycle du génie logiciel. En effet, dès l'origine de l'élaboration du protocole HLA, un besoin d'avoir un protocole de haut degré de flexibilité dans le processus de composition et d'exécution d'application adoptant HLA s'est fait sentir. D'où le FEDEP qui a pour but d'éviter de se faire répéter et de poser de multiples questions quant à la construction d'applications avec HLA. En 2003, le DMSO a identifié un cycle de sept étapes de base que toutes les fédérations HLA devraient suivre pour développer et exécuter leurs fédérations. Ces étapes sont résumées dans la figure 2.7. Elles sont maintenant détaillées et bien identifiées (IEEE-Std.1516.3, 2003).

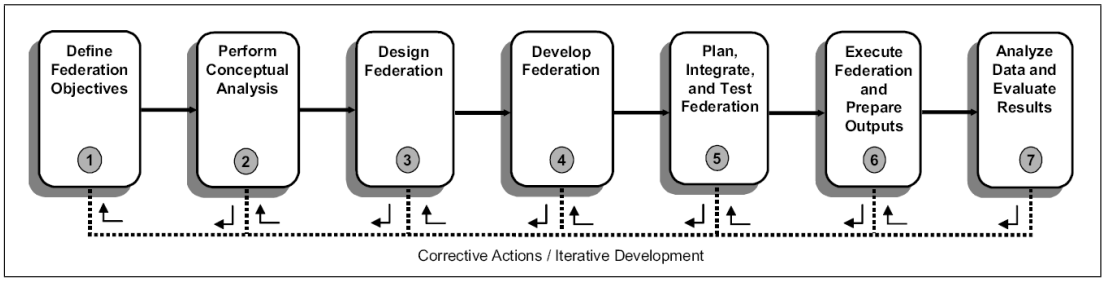



FIGURE 2.7 – FEDEP, vue de haut niveau.

L'échange de données n'est pas effectué directement de fédéré à fédéré. De toute manière, un fédéré ne connaît pas l'existence des autres fédérés dans la simulation globale. Pour pouvoir fournir des données, le fédéré s'adresse au RTI. Ce dernier les rend disponibles à la fédération. Le RTI a ensuite la charge de délivrer ces mises à jour à tous les fédérés qui ont fait connaître leur intéressement. C'est donc le RTI qui gère le routage des données entre les fédérés. On peut faire l'analogie avec le multicast, où la station ne reçoit les données que si elle s'est abonnée.

La spécification d'interface HLA décrit les services fournis aux fédérés par le RTI et par la fédération au RTI durant l'exécution d'une simulation. Il y a six classes de services :

- Les services de gestion de fédération (*Federation Management - FM*) offrent des fonctions de base exigées pour créer et faire fonctionner une fédération.
- Les services de gestion de déclaration (*Declaration Management - DM*) permettent la gestion efficace des échanges de données envoyés par les fédérés.
- Les services de gestion d'objet (*Object Management - OM*) fournissent la création, l'effacement, l'identification et d'autres services au niveau des objets définis.
- Les services de la gestion de la propriété (*Ownership Management -*

*OwM*) fournissent le transfert dynamique de propriété d'un attribut pendant une exécution.

- Les services de gestion du temps (*Time Management - TM*) permettent la synchronisation des échanges de données durant la simulation.
- Les services de gestion de distribution de données entre les fédérés lors de l'exécution d'une fédération.

La spécification d'interface HLA définit la normalisation pour que ces services soient accessibles et fonctionnels à travers l'interface d'application (*API*).

Les services assurant la gestion du temps sont fondamentaux. Ils permettent en effet de contrôler l'avancement dans le temps de chaque fédéré sur le temps de la fédération. Le tableau 2.2 résume les quatre catégories principales de fédérés.

| | | **Régulateur** | |
|---|---|---|---|
| | | OUI | NON |
| **Contraint** | OUI | Fédéré à synchronisation stricte | Moniteur |
| | NON | Fédéré à synchronisation agressive | Fédéré synchronisé par horloge externe |

TABLE 2.2 – Mécanisme de régulation temporel.

La première catégorie de fédérés concerne les fédérés à synchronisation stricte. Un tel fédéré est déclaré à la fois régulateur et contraint.

La seconde catégorie de fédérés concerne les fédérés à synchronisation agressive. Dans cette catégorie le fédéré n'est pas contraint. Il est régulateur. Il permet d'avancer l'horloge globale. Il peut générer des événements horodatés appelés événements TSO (*Time-Stamp Order*) c'est-à-dire des événements ordonnés et estampillés. Il peut faire attendre les fédérés contraints.

La troisième catégorie de fédérés concerne les fédérés moniteurs. Ce sont des

fédérés contraints. Un fédéré contraint subit l'horloge globale. Il attend l'avance des fédérés régulateurs. Un tel fédéré peut recevoir des événements TSO.

La dernière catégorie de fédérés concerne les fédérés synchronisés par une horloge externe. Le fédéré n'est ni contraint ni régulateur. Pour que le RTI puisse diffuser les événements TSO en respectant l'ordre causal, chaque fédérateur qui se déclare régulateur doit fournir un *Lookahead* c'est-à-dire une durée pendant laquelle le fédéré régulateur garantit qu'il n'émettra pas d'événements avec un horodatage inférieur à $t + t_{lookahead}$ où t est l'heure locale du fédéré régulateur.

Chaque fédéré contraint a un LBTS (*Lower Bound Time Stamp*) qui lui est associé. Le LBTS spécifie l'instant à partir duquel, au plutôt, ce fédéré pourra recevoir un événement TSO venant de l'un des fédérés régulateurs. Lorsqu'un fédéré contraint demande au RTI à avancer dans le temps à une date $t+\Delta t$, celui-ci recevra l'autorisation si $t + \Delta t$ est inférieur au LBTS. Cette valeur correspond au minimum de tous les $t_i + t_{lookaheadi}$ déclarés pour les fédérés régulateurs.

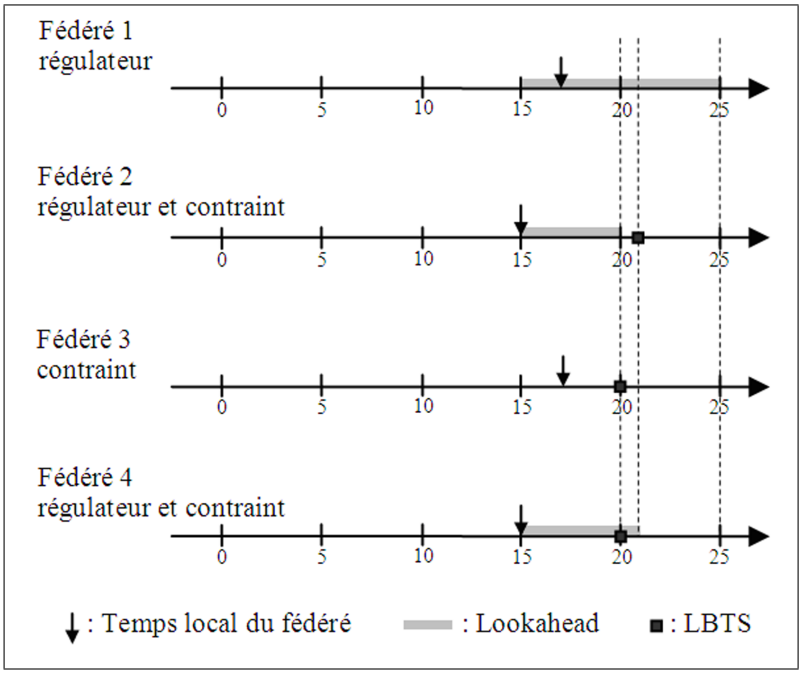



FIGURE 2.8 – Exemple de gestion du temps avec HLA.

Dans HLA, il n'y a pas de temps universel pour une fédération. Chaque fédéré peut avoir une heure locale différente des autres comme on peut le voir sur

la figure 2.8. La date locale du fédéré 1 est 15. Son *lookahead* est de 10. Donc, le fédéré 1 ne peut pas générer d'événement horodaté (TSO) avant la date 25. Le fédéré 2 a une date locale égale à 15. Son *lookahead* est de 5. Cela permet d'assurer qu'il ne générera pas d'événement horodaté avant la date 20. De plus, le fédéré 2 est contraint par le fédéré 4. Sa LBTS est de 22.

## 2.5 Synthèse

Avec des simulations de plus en plus grandes, les besoins en puissance de traitement se sont imposés. La simulation distribuée est une excellente alternative. Cependant, la simulation distribuée implique un surcoût de calcul afin d'assurer la cohérence générale. Le réseau est très sollicité et est prépondérant. Une grande quantité de messages de contrôle sont transités via ce dernier. Ces messages n'ont aucune action pertinente pour la simulation. Ils n'existent pas dans le cas d'une simulation séquentielle. Les échanges de messages qui ont pour but de préserver la causalité impliquent une gestion de la synchronisation et des erreurs pour éviter les inter-blocages qui peuvent survenir.

Différents protocoles de simulation ont vu le jour ces dernières années. DIS était une première solution et une innovation à son époque. Malheureusement, il était spécialisé au domaine de la simulation militaire. HLA s'est ouvert à d'autres types d'applications en séparant la partie service de la partie donnée. Néanmoins, son architecture basée sur le modèle fédération où tous les messages passent par le RTI pose des problèmes de robustesse. Le RTI peut devenir rapidement un goulot d'étranglement.

Jusqu'ici dans la simulation distribuée, les ressources informatiques sont utilisées sans aucun a priori et sans prendre en compte l'asymétrie potentielle de performances des ordinateurs (processeur et/ou bande passante et performance de communication). Or, dans la simulation distribuée, et comme déjà expliqué, le synchronisme est un point important. Le système peut être ralenti à cause d'une ressource du fait du décalage de l'avancement des simulations ou à cause

d'éventuels retours en arrière. Ceci peut limiter l'utilisation de la puissance du système distribué. Le choix des algorithmes de synchronisation influe directement sur les résultats.

# 3

# Le modèle pair-à-pair

## Sommaire

DANS ce chapitre, nous présentons les notions de base du paradigme pair-à-pair et ses différentes topologies. Ensuite, différentes caractéristiques. Dans la section 4 sont énumérées des classifications selon les applications et les systèmes sont présentés. La section 5 présente une étude comparative entre les deux paradigmes pair-à-pair et grille de calcul. Nous présentons ensuite brièvement la plate-forme JXTA qui sert à développer des applications pair-à-pair avant de conclure le chapitre par une synthèse.

## 3.1 Généralité

Le modèle pair-à-pair (*peer-to-peer*) est un modèle distribué de nœuds (*pairs*),qui jouent à la fois le rôle de client et de serveur. Depuis la fin des années 90, ce modèle est en plein expansion. Il doit sa popularité à Napster (Napster, web) qui est une application de partage de fichiers. Elle permettait à des millions d'internautes de télécharger et de partager librement des fichiers multimédias.

Même si ce logiciel a été rapidement abandonné pour des raisons de droits d'auteurs et sur décision de justice, le modèle pair-à-pair n'a pas eu le même sort. En effet, les scientifiques se sont aperçus de l'énorme potentiel de ce modèle alors que jusqu'ici c'est le modèle Client/Serveur qui a été adopté vu la simplicité de sa mise en œuvre et la maîtrise acquise au fils des années. Le modèle pair-à-pair profitant de l'expansion des accès à l'Internet et de la disponibilité de millions de machines, repousse considérablement les limites imposées par le modèle Client/Serveur.

Après l'épisode de Napster et son interdiction, un nouveau logiciel a vu le jour. Il n'est autre que Gnutella (Gnutella, web). Ce dernier s'est basé sur une architecture totalement décentralisée. Il n'y a aucun serveur. D'autres sont apparus comme Kazaa (Kazaa, web), Bitcomet (Bitcomet, web), eMule (emule, web)...

Après cette phase où le modèle pair-à-pair a démontré tous ses avantages par rapport à l'architecture Client/Serveur pour le partage de fichiers multimédia, les industriels et les chercheurs se sont intéressés à ce dernier. Et de plus en plus, des applications qui étaient développées sur le modèle Client/Serveur sont revues et repensées pour les faire tourner sous le modèle pair-à-pair. En outre, d'autres types d'applications sont apparus. Ils profitent des potentiels offerts par le modèle pair-à-pair et qui n'étaient pas offerts par le modèle Client/Serveur. Parmi celles-ci, on peut citer le calcul distribué qui utilise les machines connectées à l'Internet ainsi que les espaces collaboratifs basés sur les échanges instantanés de messages.

### 3.1.1 Terminologie

Dans ce qui suit on présente les différents termes utilisés dans la taxonomie des systèmes informatiques présentés dans la figure 3.1. Cette dernière permet de situer les systèmes de pair à pair par rapport aux systèmes actuels.

**Les systèmes centralisés** représentent des solutions d'unité simple et des machines multiprocesseurs.

**Les systèmes distribués** sont ceux où des composants appartenant à un réseau d'ordinateurs communiquent et coordonnent leurs actions seulement par passage de messages (Coulouris *et al.*, 2006).

**Le client :** c'est une entité (nœud, programme, module, etc.) qui lance des requêtes et qui n'est pas capable d'en répondre.

**Le serveur :** c'est une entité qui est capable de répondre aux requêtes des autres entités. Le serveur n'est pas capable de lancer des requêtes. Typiquement, on a un serveur pour plusieurs clients.

**Le modèle Client/Serveur** désigne un mode de communication entre plusieurs ordinateurs d'un réseau qui distingue un ou plusieurs postes clients du serveur. Chaque entité peut envoyer des requêtes à un serveur. Un serveur peut être spécialisé en serveur d'applications, de fichiers, de terminaux, ou encore de messagerie électronique. Le rôle du serveur est de répondre aux requêtes reçues par les clients. Un serveur est capable de répondre à plusieurs requêtes simultanément.

**Le modèle pair-à-pair** désigne un modèle de réseau informatique où tous les nœuds (pairs) jouent à la fois le rôle de serveur et de client.

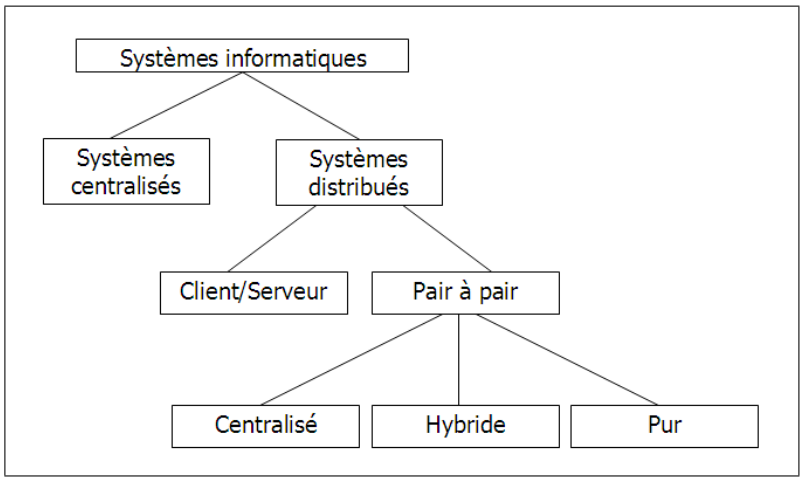



FIGURE 3.1 – Exemples de communications dans une architecture centralisée.

### 3.1.2 Définitions du pair-à-pair

Selon les communautés, différentes définitions du pair-à-pair existent.

Intel P2P Working Group, le définit : *« peer-to-peer computing is the sharing of computer resources and services by direct exchange between systems »* (P2PWG, 2001).

Ross Lee Graham (Graham, 2001) définit le pair-à-pair à travers trois critères : il doit y avoir un pair qui a les qualités d'un serveur, un système d'adressage indépendant de DNS et être capable de gérer les connectivités aléatoires.

Schoder et Fischbach (Schoder and Fischbach, 2003) l'ont définit : *« The term peer-to-peer refers to the concept that in a network of equals (peers) using appropriate information and communication systems, two or more individuals are able to spontaneously collaborate without necessarily needing central coordination »* .

Une autre définition (Aberer and Hauswirth, 2001) est : *« peer-to-peer refers to a class of systems and applications that employ distributed resources to perform a critical function in a decentralized manner »* .

### 3.1.3 Avantages et inconvénients du pair-à-pair

Le pair-à-pair est une technologie émergeante. Son utilité dans de différentes applications devient de plus en plus évidente. Pour pouvoir démocratiser cette technologie, il est nécessaire de voir ses avantages et ses inconvénients.

De nouvelles caractéristiques sont intégrées à ce paradigme tel que : passage à l'échelle, connectivité ad hoc, passage des pare-feux, disponibilité de nombreux ordinateurs, anonymat... Ces nouveautés contribuent à la fois à de nouveaux avantages qu'à de nouveaux inconvénients.

L'administration d'un tel système est assez difficile. La décentralisation du système accroît cette difficulté. De plus, la confidentialité et la fiabilité des résultats ne sont pas assurées vue qu'en général les pairs utilisés ne sont pas identifiés. Donc, aucune garantie quant à l'intention du pair participant au système.

Néanmoins, le modèle pair-à-pair ne présente pas que des inconvénients. En l'utilisant, on peut réduire les prix des projets (matériel, maintenance...). La disponibilité des ordinateurs permet de profiter des avantages de la redondance des données. Le système est plus stable parce que l'échange de messages est plus rapide et direct entre les pairs. Il n'est pas nécessaire de transiter les données à chaque fois par le serveur comme dans le modèle Client/Serveur. Ainsi, le modèle pair-à-pair permet d'améliorer le passage à l'échelle en évitant les goulots d'étranglement qui sont fréquents dans le modèle Client/Serveur.

## 3.2 Topologie

La taxonomie des systèmes informatiques est présentée dans la figure 3.1. On peut voir que les systèmes informatiques sont formés de deux grandes familles qui sont les systèmes centralisés et les systèmes distribués. Cette dernière famille est divisée en deux classes : les systèmes Client/Serveur et les systèmes pair-à-pair.

Le modèle Client/Serveur peut être : soit plat dans le cas où tous les clients

communiquent seulement avec un seul serveur, soit hiérarchique dans le cas où les clients n'ont de contacts qu'avec les serveurs de plus haut niveau qu'eux. Pour le modèle plat, on peut citer comme exemple la solution des intergiciels (*middleware*) comme *Object Request Broker* (ORB). Pour le modèle hiérarchique, on peut citer comme exemple les serveurs DNS.

Le modèle pair-à-pair peut être soit centralisé, soit hybride ou bien pur.

### 3.2.1 Le modèle centralisé

Il est très proche du modèle Client/Serveur. Il repose sur un serveur central (figure 3.2) qui détient l'ensemble des connaissances. Les clients envoient et reçoivent les informations à travers le serveur mais les ressources sont hébergées par les clients. Une fois qu'un client a reçu du serveur la liste des ressources et des clients les hébergeant, il peut interagir directement avec les autres. C'est la principale différence avec le modèle Client/Serveur.

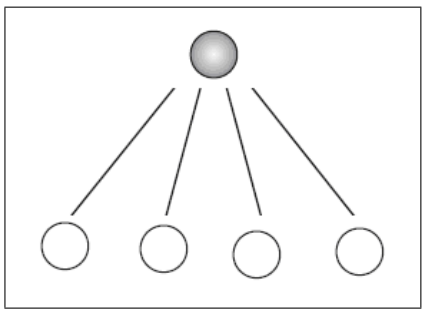


FIGURE 3.2 – Topologie construite sur le modèle centralisé.

L'avantage du modèle centralisé est la facilité de sa mise en œuvre. Sa gestion est très facile. Il n'y a aucun problème de cohérence des données. Il est de plus aisé de sécuriser un tel modèle puisqu'il s'agit de protéger un seul ordinateur : le serveur. Des systèmes s'appuyant sur ce modèle existent : serveurs de base de données, serveurs web, etc.

L'inconvénient d'un tel modèle est sa forte centralisation. Le serveur devient

rapidement un point vulnérable : s'il tombe en panne c'est tout le système qui le sera. De plus, c'est un handicap majeur pour l'augmentation des capacités et des performances du système. Quelques solutions sont utilisées pour améliorer ce modèle comme la réplication du serveur et l'utilisation de processeurs spécialisés puissants.

### 3.2.2 Le modèle hybride

Avec le modèle hybride (figure 3.3), le contrôle des informations s'échange à travers le serveur alors que les flux de données sont échangés de pair à pair. Le serveur de commande agit en tant qu'agent de surveillance pour tous les autres pairs et assure la concordance de l'information.

Les inconvénients reliés à la gestion centralisée restent encore valables. En effet, si le serveur tombe en panne, le système perd la capacité d'échanger les informations mais les échanges de flux de données entre pairs continuent à fonctionner. Le rôle exact des serveurs dans le modèle hybride dépend des infrastructures (Doyen, 2005). Ils sont en général utilisés pour assurer des fonctions relatives au routage, à la comptabilité (Hausheer *et al.*, 2003) ou à l'organisation fonctionnelle des pairs. Par exemple, Kazaa (Kazaa, web) les utilise pour la découverte et la localisation de ressources.

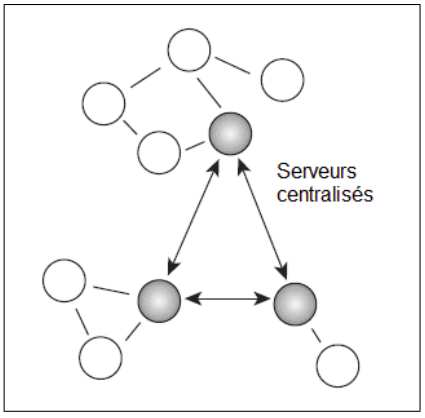



FIGURE 3.3 – Topologie construite sur le modèle hybride.

### 3.2.3 Le modèle pur

Dans ce type de modèle (figure 3.4), il n'y a aucun serveur. Le fait qu'un pair quitte le réseau n'affecte pas le système. Les pairs sont considérés comme strictement équivalents. Différentes applications sont construites sur ce type de topologie. On peut citer à titre d'exemple Gnutella tel qu'il était déployé dans sa première version ainsi que toutes les infrastructures à base de table de hachage distribuée telles que Chord (Stoica *et al.*, 2001), Pastry (Rowstron and Druschel, 2001), Tapestry (Zhao *et al.*, 2001), CAN (Ratnasamy *et al.*, 2001)...

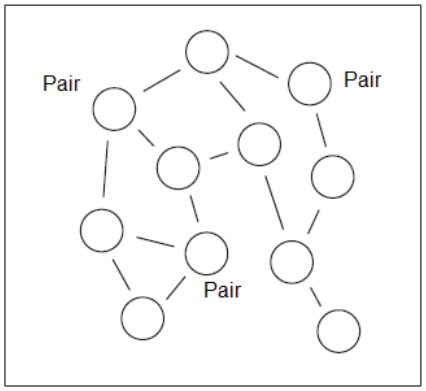



FIGURE 3.4 – Topologie construite sur le modèle pur.

#### Modèle structuré

Dans les modèles structurés (Ratnasamy *et al.*, 2001; Rowstron and Druschel, 2001; Stoica *et al.*, 2001; Zhao *et al.*, 2001), l'évolution du réseau ainsi que l'emplacement des nœuds sont strictement contrôlés. Ces contraintes imposées influent sur la robustesse du réseau et l'autonomie des nœuds. Les modèles pair-à-pair structurés sont une excellente solution pour bâtir des systèmes où une importance particulière est donnée au placement des ressources contrôlées tel que le stockage de fichier distribué. Toutefois, ce type de modèle n'est pas approprié dans le cas où les nœuds seraient très dynamiques. Le principal avantage est que ces modèles utilisent des mécanismes de recherche native afin de tirer profit de la structure particulière ajoutée au réseau.

**CAN (The Content Adressable Network)** proposé par Ratnasamy et al. (Ratnasamy *et al.*, 2001), est une plate-forme pour les systèmes pair-à-pair structurés basé sur des coordonnées spatiales cartésiennes virtuelles. Chaque nœud est responsable de sa part d'espace. CAN permet de stocker des données à un point donné de l'espace dans un réseau pair-à-pair et de router d'un point de l'espace à un autre. Une visualisation d'un graphe formé de 32 nœud est illustrée sur la figure 3.5.

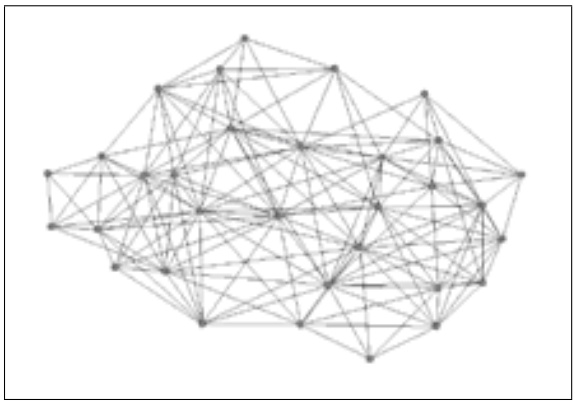


FIGURE 3.5 – Réseau de type CAN.

**Chord** proposé par Stoica et al. (Stoica *et al.*, 2001), est un autre modèle de système de pair à pair structuré. Il repose sur une topologie en anneau. Un nœud de Chord a la connaissance de son prédécesseur et de son successeur. Une fonction de hachage régulière génère une clé pour chaque nœud à partir de son adresse IP. Ensuite, chaque nœud est placé dans l'anneau de manière à ordonner les clés par ordre croissant. La complexité de cet algorithme est au plus de $O(logN)$ requêtes pour trouver une information dans un anneau de N éléments grâce à une table de hachage distribuée. Une illustration de ce type de réseau est présentée sur la figure 3.6 pour le cas de 32 nœud.

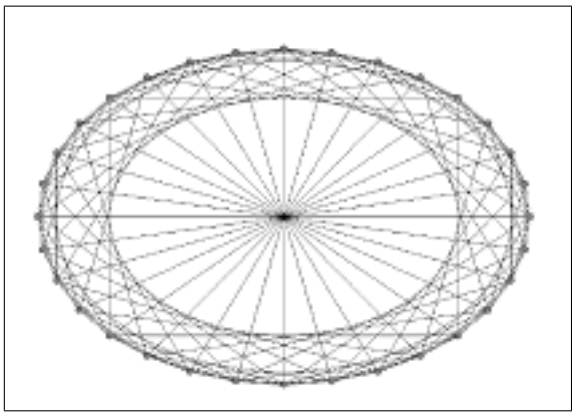


FIGURE 3.6 – Réseau de type Chord.

**Modèle non structuré**

Dans ce type de modèle, aucune règle de construction n'est imposée. Il peut être une excellente solution pour des systèmes très dynamiques. Gnutella dans sa première version reposait sur ce type de modèle.

**PRU (Pandurangan-Raghavan-Upfal)** proposé par Pandurangan et al. (Pandurangan *et al.*, 2003), est un modèle de système pair-à-pair non structuré. Il est basé sur une simple politique de croissance du réseau qui assure un graphe à petit diamètre.

## 3.3 Caractéristiques

Le modèle pair-à-pair a ses caractéristiques intrinsèques (Milojicic *et al.*, 2002; Doyen, 2005). Certaines d'entre elles permettent de pallier à des problèmes posés par le modèle Client/Serveur comme le passage à l'échelle, le coût financier et la maintenance des équipements. Mais d'autres apparaissent comme la sécurité, la disponibilité et la volatilité des pairs. Le tableau 3.1 présente les différences fondamentales entre les deux modèles pair-à-pair et Client/Serveur.

D'une manière générale, on remarque que le modèle Client/Serveur est basé sur des ressources statiques alors que le modèle pair-à-pair est dynamique.

| Critère | Modèle Client/Serveur | Modèle pair-à-pair |
|---|---|---|
| Gestion | Supervisé | Auto-organisé |
| Présence | Permanente | Ad hoc |
| Accès aux ressources | Recherche | Découverte |
| Organisation | Hiérarchique | Distribuée |
| Mobilité | Statique | Mobile |
| Disponibilité | Dépendante du serveur | Dépends des internautes |
| Nommage | Reposant sur le DNS | Indépendant |
| Modèle de programmation | RPC | Asynchrone |

TABLE 3.1 – Comparaison des infrastructures Client/Serveur et pair-à-pair.

Dans ce qui suit, on passe en revue l'ensemble des caractéristiques du modèle pair-à-pair.

### 3.3.1 Décentralisation

La décentralisation est la principale caractéristique du modèle pair-à-pair. Elle s'applique aux différents types de topologie. Dans le cas du modèle centralisé, seules les ressources sont décentralisées, mais les mécanismes de recherches et de localisations restent centralisés. Alors que dans le cas du modèle pur, tout est décentralisé : les ressources, les mécanismes de recherche, la localisation, la sécurité, le routage... (Barkai, 2001*a*; Doyen, 2005)

### 3.3.2 Passage à l'échelle

Le passage à l'échelle est limité par des facteurs comme la quantité de messages échangés entre différents pairs. La décentralisation permet le passage à l'échelle. Celui-ci est assuré pour les différentes topologies du pair-à-pair même pour le modèle centralisé car ce dernier ne centralise que l'index de références. La charge et le trafic sollicité au serveur restent acceptables. Différentes applications pair-à-pair de partage de fichiers ont montré cela. On dénombre un très grand nombre de participants aux différents réseaux de pair à pair sans pour autant poser de problème au bon fonctionnement des applications. Dans (Milojicic *et*

*al.*, 2002; Cappello and Djilali, 2004; Doyen, 2005) il est cité que : Napster permet l'échange de fichiers de pair à pair sans passer par un serveur. Il a atteint des pics de 6 millions d'utilisateurs. Une étude menée en 2001 (Saroiu *et al.*, 2002) comptabilise une moyenne de dix mille participants simultanés pour Gnutella. OceanStore (Kubiatowicz *et al.*, 2000), une application pair-à-pair de stockage de fichier, permet de gérer $10^{10}$ utilisateurs stockant plus de $10^{14}$ fichiers. Seti@Home, une application de calcul distribué, compte plusieurs milliers d'utilisateurs.

### 3.3.3 L'anonymat

L'anonymat est présentée dans (Milojicic *et al.*, 2002; Doyen, 2005) comme une fonction qui permet de ne pas être identifiable sur un réseau. On peut distinguer six formes d'anonymat (Dingledine *et al.*, 2001) : l'auteur, l'éditeur, le serveur, le document et la requête.

Plusieurs applications implémentent cette caractéristique. On peut citer Freenet (freenet, web) qui utilise une forme de routage qui garantit l'anonymat du serveur, de l'auteur et du lecteur ne permettant à aucun nœud de savoir qu'elle est la source et la destination de la requête. D'autres implémentent explicitement des mécanismes d'anonymat comme FreeHaven (FreeHaven, web) et Publius (Waldman *et al.*, 2000).

### 3.3.4 L'auto-organisation

Dans les systèmes pair-à-pair, l'auto-organisation est nécessaire en raison du passage à l'échelle et de l'absence d'un élément central. Le passage à l'échelle est imprévisible vu la volatilité des pairs. Il est très difficile de prévoir le nombre des utilisateurs et la charge sollicitée. Cela requiert une auto-organisation fréquente. Plus le passage à l'échelle augmente, plus la probabilité de fautes augmente d'où la nécessité d'une auto-maintenance et d'une autoréparation du système. La gestion d'un tel environnement si fluctuant est coûteuse en équipement et/ou en personnes si elle était manuelle. C'est pour cette raison qu'elle est distribuée

parmi les pairs.

L'auto-organisation est formulée par (Doyen, 2005) comme pouvant couvrir différents aspects tel que fonctionnels, communautaires ou topologiques. L'auto-organisation fonctionnelle est très claire dans les modèles hybrides où certains pairs sont chargés d'exécuter des fonctions particulières. L'aspect communautaire est lisible dans des applications pair-à-pair qui regroupent les pairs par centre d'intérêt. On trouve ce type d'organisation dans Jabber (Jabber, web) qui regroupe les pairs en fonction des sujets sur lesquels ils souhaitent échanger. Quant à l'aspect topologique, on distingue deux grandes familles : les réseaux pair-à-pair structurés et les réseaux pair-à-pair non structurés.

Dans OceanStore (OceanStore, web), l'auto-organisation est appliquée à la localisation des pairs et à la table de routage (Kubiatowicz *et al.*, 2000). L'infrastructure est continuellement adaptée vu la volatilité des pairs ainsi que l'hétérogénéité des bandes passantes. SearchLing (SearchLing, 2000) utilise l'auto-organisation pour adapter au mieux le réseau aux types de recherches et de résultats dans le dessein de réduire le trafic réseau et de diminuer le nombre de recherches infructueuses.

### 3.3.5 Connectivité Ad Hoc

C'est une des spécificités des réseaux pair-à-pair. Ceci est dû à la nature des pairs. En effet, le comportement des usagers influe considérablement sur la disponibilité des pairs. Les usagers peuvent à leurs grés se connecter ou se déconnecter de manière spontanée et donc imprévisible. Ainsi, les pairs sont volatils et il est assez difficile de prévoir leurs comportements.

De plus, avec l'essor des portables et du *WiFi*[1], les usagers ont de plus en plus un comportement nomade. L'accès à l'Internet est donc dépendant de la localisation géographique de l'utilisateur et le débit dépend de la connexion mise à disposition du nomade. Selon que cette dernière s'effectue directement via un

1. C'est une technologie de réseau informatique sans fil. Il est basé sur la norme IEEE 802.11 (ISO/CEI 8802-11).

fournisseur d'accès à Internet haut débit ou à travers différents terminaux mobiles qui forment un réseau Ad Hoc.

Du fait de la volatilité des pairs, des mécanismes de duplication et de synchronisation sont mis en place pour pallier ce problème. L'absence d'un pair ou d'une ressource ne doit pas être considérée comme une faute.

## 3.4 Classification du pair-à-pair

### 3.4.1 Les applications pair-à-pair

Les auteurs de (Barkai, 2001*b*; Milojicic *et al.*, 2002) ont classé les applications pair-à-pair en trois catégories (Figure 3.7) :

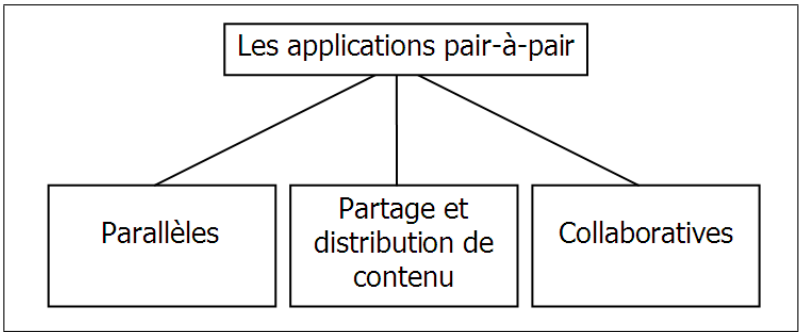



FIGURE 3.7 – Taxonomie des applications pair-à-pair.

#### Applications parallèles

Dans ce type d'applications, les calculs sont divisés en petites entités indépendantes qui seront exécutées sur les différents pairs. D'autres possibilités adoptées par SETI@Home , genome@Home, etc. où un même calcul est répété plusieurs fois avec différents paramètres.

#### Le partage et la distribution de contenu

Le mot contenu est utilisé pour faire référence à n'importe quelle entité qui peut être numérisé comme un message, un fichier, un document, etc. Cela consiste

essentiellement au partage des fichiers qui sont stocké sur les pairs. Différents protocoles ont été implémentés. Les plus connus sont Kazaa, BitTorent, eDonkey. Aucune interopérabilité n'existe entre ces différents logiciels. Mais, désormais certaines applications implémentent plus d'un protocole afin d'augmenter le volume de données accessibles.

#### La collaboration

Les applications pair-à-pair collaboratives proposent à des communautés d'utilisateurs de collaborer en temps réel à travers des services de communication et d'échanges de données sans passer par un serveur. Différents moyens de collaborations sont offerts : la messagerie instantanée, la visioconférence, l'échange des documents de travail, etc. Ce type d'application est destiné à deux catégories d'usagers qui sont les professionnels et les particuliers. Différentes applications ont vu le jour pour répondre aux besoins des uns et des autres. On peut citer parmi eux : YM ! (YM !, web), MSN (MSN, web), ICQ (ICQ, web), Skype (Skype, web), Jabber (Jabber, web), etc. pour la communication et Groove (Groove, web), Ocolus pour le travail collaboratif.

### 3.4.2 Les systèmes pair-à-pair

On distingue actuellement quatre grandes familles de systèmes pair-à-pair représentées sur la figure 3.8. Il s'agit du calcul distribué, du partage et distribution de contenu, de la collaboration et des plates-formes de développement.

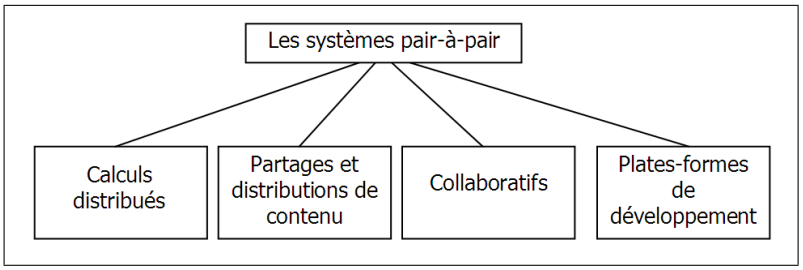



FIGURE 3.8 – Taxonomie des systèmes pair-à-pair.

### Le calcul distribué

La puissance de calcul offerte par le pair-à-pair a intéressé la communauté du calcul distribué. Le projet Seti@Home, qui a pour objectif d'analyser des données acquises par des récepteurs du projet SETI, était le premier à adopter le modèle pair-à-pair vu la quantité colossale des données à analyser. Il s'agit d'un programme qui se lance une fois que le pair participant est inactif. Il télécharge une quantité de données du serveur, les analyse et renvoie le résultat. Ce projet, qui a débuté depuis 1999, est considéré comme pionnier dans le domaine. D'autres ont suivi comme Genome@Home (Genome, web), Folding@Home (Folding, web)...

La distribution de calcul via une telle architecture est actuellement bien maîtrisée. Les travaux actuels penchent vers la convergence du domaine grille de calcul (grid computing) et du pair-à-pair (Cabani *et al.*, 2006). Actuellement, ces deux derniers présentent de nombreuses différences qui seront détaillées dans le paragraphe suivant.

### Partage et distribution de contenu

Il s'agit de créer des communautés de pairs qui partagent des fichiers stockés sur leurs machines. Le pionnier est Napster. D'autres protocoles ont vu le jour par la suite. On peut citer CAN (Ratnasamy *et al.*, 2001), Chord (Stoica *et al.*, 2001), Past (Druschel and Rowstron, 2001) ou OceanStore (Kubiatowicz *et al.*, 2000) qui sont construits sur des tables de hachage distribuées. Ils proposent de construire un système de fichiers de type Unix qui soit distribué sur une communauté de pairs. Ces protocoles offrent des possibilités de stockage et de partage très importants sans aucun investissement spécifique. Néanmoins, un nouveau comportement, appelé *free riding* (Adar and Huberman, 2000), est apparu. Il s'agit de télécharger des données sans pour autant les partager, ce qui met à mal la fiabilité du système.

### La collaboration

Différents moyens sont offerts pour communiquer entre différents usagers formant des communautés comme la causette (*chat*), la visioconférence et la messagerie instantanée. On peut citer parmi eux : YM !, MSN, ICQ, Skype, Jabber, etc. pour la communication et Groove, Ocolus pour le travail collaboratif.

### Les plates-formes de développement

Différentes plates-formes de développement permettent aux développeurs de s'abstraire des mécanismes de bas niveau du modèle pair-à-pair.

*Microsoft* propose deux infrastructures pour le développement des applications pair-à-pair. La première est une extension de la plate-forme de développement de service web .NET (Olson, 2001). Cette solution est utilisée beaucoup plus pour adapter une plate-forme initialement dédiée à un fonctionnement centralisé plutôt qu'une véritable plate-forme pair-à-pair intégrant les mécanismes de base pour construire des applications de pair à pair. La seconde est le *Windows Peer-to-Peer* (Davies *et al.*, 2006). Elle offre un ensemble de mécanisme pour les concepteurs d'applications afin de développer des services pair-à-pair.

*Sun Microsystems* a eu l'initiative de proposer Jxta (Gong, 2001) qui se lit « juxta ». Depuis avril 2001, Jxta est passée dans le domaine de l'*Open Source*. Cette plate-forme a pour but de faciliter l'interopérabilité entre les applications, d'être indépendante du matériel et du système utilisé et d'être intégrée à n'importe quel type de matériel possédant un processeur. Cette plate-forme sera présentée plus en détail par la suite.

## 3.5 Pair-à-pair et grille de calcul

Le modèle pair-à-pair ainsi que les grilles de calculs sont de plus en plus utilisés dans les calculs distribués. Ce sont deux solutions prometteuses. Mais, elles sont radicalement différentes et souvent mal comprises.

Avec le modèle pair-à-pair, chaque nœud peut jouer à la fois le rôle d'un serveur et d'un client. Tous les nœuds sont identiques et considérés de la même façon (dans le cas général du modèle pur) ce qui n'est pas le cas dans les grilles de calculs. Une des propriétés principales du modèle pair-à-pair est que l'échange des messages se fait directement entre les nœuds. Cette propriété est absente dans les grilles.

Récemment, des grilles de calculs basées sur un modèle pair-à-pair commencent à émerger. Ceci a été dicté par le souci de profiter des avantages du pair-à-pair notamment pour un meilleur passage à l'échelle. De tels systèmes permettent la création de grilles de calculs basées sur des ordinateurs communiquant avec des mécanismes adoptés par le modèle pair-à-pair (Sunaga *et al.*, 2005).

Dans ce qui suit nous discutons de quelques caractéristiques concernant ces deux paradigmes :

**Décentralisation :** la décentralisation tient compte de la flexibilité. À la différence du modèle Client/Serveur, elle ne souffre pas des nœuds simples d'échec où le serveur devient vite un goulot d'étranglement. Les systèmes distribués, qui sont flexibles et assurent un meilleur passage à l'échelle dans des environnements instables, sont très importants. La décentralisation permet de rapprocher les ressources ce qui permet de diminuer et réduire les temps de réponse, ou même éliminer la latence due au réseau. Cela permet aussi une meilleure utilisation des capacités du réseau.

**Coût et efficacité :** en ce qui concerne la performance, les réseaux évoluent à une vitesse plus rapide que le matériel et de plus en plus de PCs sont reliés au Web par l'intermédiaire des réseaux à large bande passante. Ceci tient compte d'une meilleure exploitation de ces ressources qui étaient précédemment non reconnues. Ce qui a mené à donner une importance considérable à trois paramètres importants dans le calcul moderne : le stockage, la bande passante et les ressources informatiques. Ceci prend plus de sens si nous savons qu'il devient plus facile de relier des centaines de

millions d'ordinateurs dans le monde entier pour former un réseau surtout avec la révolution globale de l'Internet. Robert Metcalfe a formulé une loi empirique mesurant l'utilité d'un réseau (Metcalfe, 1995).

$$\text{Utilité d'un réseau} = k * N^2.$$

Selon la Loi de Metcalfe, l'utilité d'un réseau de $N$ nœuds maillés est fonction du nombre de connexions possibles : elle est d'ordre $N^2$ . Selon la Loi de Reed (Reed, 1999), comme de plus en plus d'applications en réseaux s'appuient sur des sous-groupes d'usagers (communautés), à la valeur d'usage de Metcalfe vient s'ajouter une utilité d'ordre $2^N$, puisque le nombre de sous-groupes possibles (d'au moins 2 personnes) est égal à $2^N - N - 1$ où $N$ est le nombre de participants. Toutes ces lois montrent que le nombre de pairs raccordés est très important pour augmenter l'utilité d'un réseau. Et comme dans des systèmes de pair à pair le nombre de nœuds peut atteindre des centaines de millions, cela permet d'accroître considérablement l'utilité du réseau.

**Informatique pervasive :** avec les systèmes pair-à-pair, il est possible de raccorder n'importe quel machine avec processeur au réseau (Agenda électronique, téléphone cellulaire...). Ceci est utile pour une informatique plus hétérogène. Elle est assez flexible pour soutenir de nouveaux protocoles de communication qui serviront à l'échange d'informations dont l'informatiques pervasives a besoin. Quelques travaux ont été faite en développant des modèles conceptuels pour la gestion des données dans les environnements informatiques pervasives basés sur l'action réciproque de trans-couche entre la gestion des données et les couches de communication (Perich, 2004).

**Communautés ciblées et motivation :** bien que la technologie des grilles de calculs ait été au départ développée pour s'adresser aux besoins de collaborations scientifiques ; l'intérêt commercial grandit (Foster and Iamnitchi, 2003). Les participants aux grilles contemporaines font ainsi partie des communautés établies qui sont disposées à consacrer des efforts pour la

création et l'exploitation des infrastructures voulues. Ces infrastructures assurent un degré de confiance, de responsabilité et d'opportunité de sanctionner les conduites inconvenantes. Au contraire, le modèle pair-à-pair a été popularisé par de simples internautes. Le partage massif de fichier (musique, films, etc.) et les applications extrêmement parallèles (LimeWire, web; Abramson *et al.*, 1995) ont atteint un passage à l'échelle atteignant dans quelques cas des centaines de milliers de nœuds. Les communautés qui forment les réseaux pair-à-pair comprennent divers individus anonymes sans aucune motivation particulière.

**Ressources :** en général, les grilles de calcul intègrent des ressources qui sont plus puissantes, plus diverses et mieux raccordées que les ressources pair-à-pair (Foster and Iamnitchi, 2003). Une ressource d'une grille pourrait être une grappe, un système de stockage, une base de données, ou un instrument scientifique administré d'une manière organisée en fonction d'une politique bien définie.

**Gestion de cohérence :** les systèmes pair-à-pair courants sont basés principalement sur le partage des fichiers statiques. Cependant, en utilisant des réseaux pair-à-pair dans les grilles de calcul, ils devront supporter le partage des fichiers fréquemment modifiés par leurs utilisateurs. La cohérence a été étudiée pour le *web caching* (Yin *et al.*, 1999; Duvvuri *et al.*, 2000). Dans (Lan *et al.*, 2002) les auteurs présentent trois techniques pour la gestion de cohérence dans des systèmes pair-à-pair : *push* (initié par le propriétaire), *pull* (initié par le client) et une technique hybride *push/pull*.

**Services et infrastructure :** les systèmes pair-à-pair ont tendance à se concentrer sur l'intégration des ressources simples (ordinateurs personnels) par l'intermédiaire de protocoles spécialement conçus.

Dans le tableau 3.2 (Cabani *et al.*, 2006), on compare et l'on met en contraste le modèle pair-à-pair au grille de calcul en utilisant un ensemble

de caractéristiques communément utilisées pour les solutions d'informatique distribuée. Dans (Buyya, 2002) un essai semblable d'identifier et de caractériser les différences est présenté. Les principales caractéristiques que nous avons prises en compte sont : le public visé, la découverte des ressources, la gestion des ressources et des utilisateurs, l'allocation des ressources, l'ordonnancement, l'interopérabilité, le passage à l'échelle, le débit...

| Caractéristique / Critère | Grille de calcul | Pair-à-pair |
|---|---|---|
| **CRITERES GENERAUX** | | |
| But | Organisation virtuelle | System virtuelle |
| Rôle des entités | Serveur de grille | Pair à la fois serveur et client |
| Nombre des entities | 10 – 1000 utilisateurs | Millions d'utilisateurs |
| Nœud | Dédidé | Non dédié |
| Classe | Pré-organisé | Auto-organisé |
| **CRITERES TECHNIQUES** | | |
| Structure | Hiérarchique statique | Totalement distribué et dynamique |
| Totalement décentralisé | Non | Oui |
| Connectivité direct | Non | Oui |
| Passage à l'échelle | Limité | Non limité |
| Mécanisme de contrôle | Centrale | Totalement distribué |
| Connectivité | Statique à grande vitesse | Entrée/sortie à tout moment |
| Disponibilité | Elevé | Volatile |
| Risque de panne | Elevé | Bas |
| Ressources | Plus puissant | Moins puissant |
| Découverte des ressources | Statique, inscription centralisé sur un modèle hiérarchique | Limité par le nombre des nouveaux pairs dans le réseau |
| Transparence de la localisation | Oui | Non |
| Formation Ad-hoc | Non | Oui |
| **CRITERES ECONOMIQUES** | | |
| Communauté | Fermé aux inscrits | Ouvert |
| Participants | Enregistrés | Volontaires |
| Fiabilité | Confiance assurée | Partielle (pairs non sûrs) |
| Standards | Oui | Non |
| Sécurité | Sûr | Pas sûr |
| Applications | Scientifique – Données intensives | Partage de fichier ou calcul distribué |

TABLE 3.2 – Des critères comparatifs entre le pair-à-pair et les grilles de calculs (Cabani *et al.*, 2006).

On s'aperçoit que, quoique ces deux paradigmes reposent sur des infrastructures distribuées, ils sont contrastés. En effet, le modèle pair-à-pair est dynamique, ouvert et non sûr or les grilles de calculs sont statiques, closes et sûres.

## 3.6 Plate-forme de développement - JXTA

Le projet *JXTA* est l'initiative lancée par *Sun Microsystems* pour étudier les possibilités du modèle d'applications pair-à-pair. Depuis Avril 2001, une première mouture de leur plate-forme de développement a été placée dans le domaine de l'*Open Source*. Cette plate-forme a pour but :

- De faciliter l'interopérabilité entre applications.
- D'être indépendante du matériel (*hardware*) et des logiciels (*software*) utilisé.
- De pouvoir être intégrée dans n'importe quel type d'équipement pouvant être relié à d'autres (l'ubiquité).

### 3.6.1 Architecture

En Janvier 2000, *Sun Microsystems Inc.* a proposé l'architecture de la plate-forme JXTA (Gradecki, 2002), suite à une étude portant sur un grand nombre d'architectures pair-à-pair existantes et visant à identifier les fonctionnalités et mécanismes indispensables au support d'applications pair-à-pair multiplates-formes, offrant des fonctionnalités de haut niveau. JXTA est un projet open source. Il offre des fonctions essentielles et indispensables au support des applications pair-à-pair multi plates-formes. Les concepts de JXTA sont décrits en détail dans les références suivantes (Wilson, 2002; Halepovic and Deters, 2005).

L'architecture JXTA comporte trois couches qui sont représentées dans la figure 3.9).

Il s'agit de :

- La couche noyau : elle offre au développeur un ensemble de mécanismes de base comme les *PeerGroups*, les *Pipes* et le *Peer Monitoring*. Elle permet aussi de définir des politiques de sécurités (anonymat et cryptage).

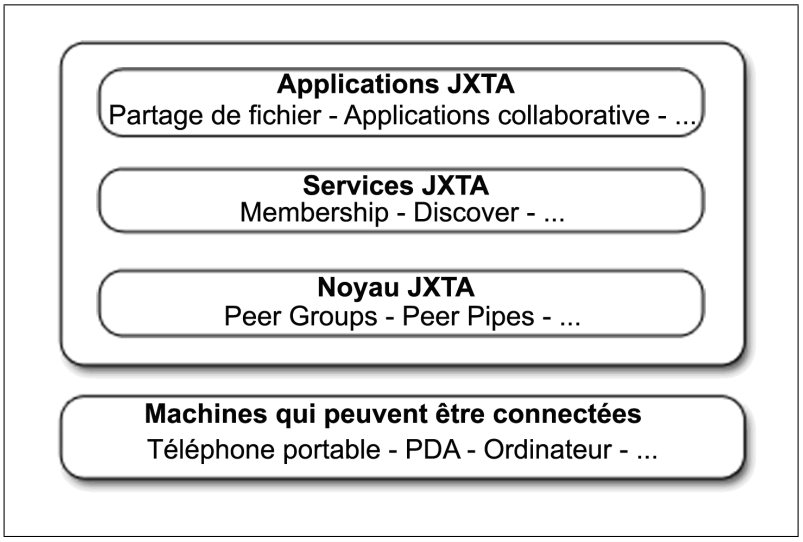



FIGURE 3.9 – Architecture de la plate-forme JXTA.

- La couche services : elle permet d'étendre les mécanismes de base offerts par la couche noyau. Les principaux services proposés dans cette couche sont des mécanismes de recherche et d'indexation, de partage de ressources, de stockage, de traduction de protocole...
- La couche applications : les applications y sont développées sur la base des services disponibles dans la couche services.

### 3.6.2 Les concepts

Dans ce qui suit, on identifie les différents concepts utilisés par JXTA :

**Pair :** un pair est l'unité structurelle de base de tout réseau pair-à-pair. Un pair peut être toute ressource capable de se connecter au réseau et de fournir un service : poste de travail, grappe de machines ou même un assistant personnel électronique de type PDA. On distingue quatre types de pairs qui sont :

- Pairs minimaux : ces pairs sont réduits à des fonctionnalités élémentaires de communication comme l'envoi et la réception de messages. Ils ne sont capables ni de stocker de l'information ni de router les messages.

- Pairs simples : les pairs peuvent envoyer et recevoir des messages et en plus stocker dans un cache local des informations sur les ressources du réseau sous forme d'annonces. Ces annonces leur permettent de répondre à des requêtes d'informations et de participer ainsi au protocole de découverte de ressources.
- Pairs de rendez-vous : ces pairs sont essentiels à la constitution du réseau. En plus des fonctionnalités décrites plus haut ces pairs permettent de faire suivre les requêtes vers d'autres pairs connus, pairs simples ou pairs de rendez-vous.
- Pairs de routage : ces pairs stockent des informations de routage vers les autres pairs. Ils permettent aussi de transmettre les requêtes des pairs isolés du réseau situés derrière un pare-feu.

**Communauté :** une communauté Jxta (JXTA-v2.3.x, 2005) est appelé Peergroup ou encore groupe de pairs. Elle a été reformulée par (Doyen, 2005) comme un ensemble de pairs qui exécutent un ensemble de service communs. La création d'une telle communauté peut être dictée par plusieurs raisons : regroupement des pairs autour d'intérêts communs, établissement d'un espace sécurisé...

**Service :** c'est une interface de programmation qui offre une ou des fonctions particulières dans un groupe de réseau Jxta (Wilson, 2002). On distingue deux types de service avec Jxta : les services de groupes et les services de pairs. Le rôle de ces deux services ont été reformulés en français par (Tedeschi, 2003; Doyen, 2005). Les services de groupes sont des services offerts aux pairs participant au *Peergroup* qui les héberge. Ils ont une durée de vie liée à celle du groupe et pour rejoindre le groupe dans lequel les services s'exécutent, un pair doit obligatoirement en posséder une implémentation. Par contre, les services de pairs sont offerts par un

pair particulier. Et donc, leur existence est liée à la présence du pair qui les héberge.

**Annonce :** c'est un document XML qui décrit sous la forme de méta données une ressource disponible sur un réseau Jxta. Ainsi, les pairs, les groupes de pairs, les services, les pipes, les messages, les *endpoints*, et les autres ressources Jxta sont formalisés à travers ce concept qui unifie la représentation des ressources. Découvrir un pair revient à découvrir son annonce (Doyen, 2005).

**La communication :** (Doyen, 2005) la présente comme suit : elle s'effectue au moyen d'un *pipe*. Il est identifié de manière unique par un *pipeID* et indépendant de la localisation des pairs. La communication entre les pairs Jxta est assurée par le *Network Transport*, un service constitué de trois éléments qui sont les *pipes*, les *endpoints* et les messages. Un *pipe* représente un canal de communication unidirectionnel virtuel par lequel des données sont échangées au sein d'un *Peergroup*. Jxta distingue deux types de canaux qui sont les *pipes unicasts* qui relient une ou plusieurs sources à une destination et les canaux propagés qui relient une source à plusieurs destinations. Chaque extrémité d'un canal est représentée par un *endpoint* qui représente l'interface entre le réseau virtuel Jxta et le niveau transport par laquelle la communication est effectuée. Enfin, les données échangées entre des pairs sont encapsulées dans des messages qui sont des conteneurs génériques de données. La figure 3.10 illustre les trois concepts qui constituent le principe de communication de Jxta.

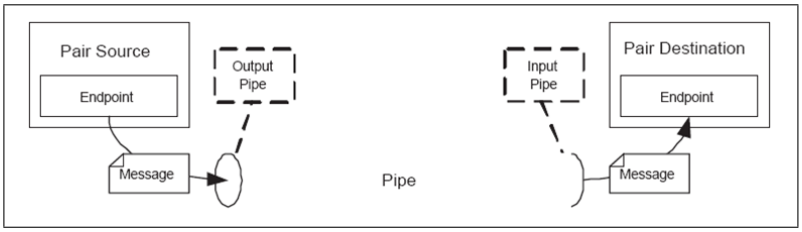



FIGURE 3.10 – Modèle de communication Jxta fondé sur des pipes, des endpoints et des messages. (Doyen, 2005)

### 3.6.3 Les protocoles

JXTA est composé de six protocoles de base qui servent à la recherche, l'organisation, l'identification et l'échange de messages entre les pairs.

- Le *Endpoint Routing Protocol* (ERP) : il permet à un pair de trouver les routes disponibles pour l'envoi d'un message à un autre pair. Il fournit des mécanismes essentiels au routage des messages et laisse la possibilité aux développeurs d'implémenter des services de routage plus efficaces et plus intelligents.
- Le *Rendezvous Protocol* (RVP) : Il permet la propagation des messages et son contrôle au sein d'un *PeerGroup*.
- Le *Peer Resolver Protocol* (PRP) : Il permet au sein d'un *PeerGroup* l'envoi de requêtes génériques adressées à un ou plusieurs contacts et l'identification des réponses correspondantes.
- Le *Peer Discovery Protocol* (PDP) : Il permet de découvrir les ressources publiées par les pairs d'un *PeerGroup* au moyen d'avertissements. Il aide les services de couches supérieurs en prenant en charge les détails de gestion des avertissements (messages, cache, expiration...)
- Le *Peer Information Protocol* (PIP) : Il permet d'obtenir au moyen d'échange de messages et d'informations sur l'état du pair.
- Le *Pipe Binding Protocol* (PBP) : Il permet aux pairs de lier des *PeerEndPoints* (les pairs aux extrémités du réseau) au moyen de

pipes en fonction des avertissements de pipe reçus ou envoyés.

Ces 6 protocoles ne sont pas systématiquement disponibles sur toutes les plates-formes. Seuls sont présents ceux qui peuvent ou veulent être supportés par l'équipement.

Le seul protocole à implémenter obligatoirement est le *Peer Discovery Protocol* car c'est lui qui permet de s'annoncer et de trouver les autres pairs. Par contre, un protocole tel que le *Peer Endpoint Protocol* ne sera implémenté que par des machines capables d'assurer un certain débit de transit donc, ayant des connexions conséquentes.

## 3.7 Synthèse

Le modèle pair-à-pair est un modèle distribué où les entités appelées pairs jouent le rôle à la fois d'un client et d'un serveur. On distingue trois modèles de décentralisation :

- Le modèle totalement décentralisé dit le modèle pur. Les pairs sont considérés de façon strictement équivalente.
- Le modèle centralisé : la connexion à travers un serveur y est obligatoire pour pouvoir se connecter au réseau pair-à-pair. Ce dernier assure les fonctions de découverte et de localisation des autres pairs.
- Le troisième modèle est hybride. Quelques pairs y jouent de serveurs.

Les principales caractéristiques du modèle pair-à-pair sont la décentralisation, un meilleur passage à l'échelle par rapport au modèle Client/Serveur, l'anonymat des participants, une auto-organisation permettant aux communautés de pairs de délivrer leurs services de manière autonome, ainsi qu'une connectivité ad hoc.

Les systèmes pair-à-pair sont classés en quatre catégories qui sont les calculs distribués, le partage et la distribution de contenu, la collaboration et les plates-formes de développement. Pour chacune des catégories, on a présenté des exemples

d'applications. Dans le cas du calcul distribué, une comparaison est présentée entre le paradigme pair-à-pair et les grilles de calculs. Et on a conclu que leurs caractéristiques sont contrastés. Néanmoins, une tendance à la convergence des deux systèmes se profile dans le futur.

Enfin, nous avons présenté brièvement JXTA. C'est une plate-forme générique pour le développement d'applications pair-à-pair. Elle a été utilisée pour mettre en œuvre notre modèle.

# 4

# Les systèmes multiagents et la simulation

## Sommaire



DANS ce chapitre, nous présentons les principes généraux des systèmes multiagents. Nous poursuivons par les motivations de l'utilisation de ce paradigme dans le domaine de la modélisation et de la simulation et des applications possibles. Au vu de l'intérêt grandissant à la distribution des simulations multiagents, nous exposons quelques méthodologies et architectures. Quelques travaux existants dans la littérature adoptant le protocole HLA sont cités. Nous concluons le chapitre par une synthèse.

## 4.1 Genèse

Depuis longtemps, dans le domaine de l'intelligence artificielle, les programmes ont été considérés comme des entités individualisées et capable de rivaliser avec l'être humain. Le terme *intelligence artificielle* a été utilisé pour désigner un projet de recherche consistant à concevoir une machine intelligente capable de réussir des tâches complexes aussi bien qu'un être humain. Cette manière de faire se retrouve dans les systèmes experts où les programmes informatiques sont capables de remplacer des humains dans les tâches les plus complexes qui réclament de l'expérience, du savoir-faire et du raisonnement. Néanmoins, ces systèmes manquent de capacités d'interaction avec leur environnement.

Avec la complexité croissante des systèmes informatiques, il est de plus en plus recommandé de découper le système en modules et unités indépendantes. Les interactions sont limitées et parfaitement contrôlées. Malgré ceci, les concepteurs des systèmes experts ont conclu que le savoir-faire, les compétences et les connaissances sont détenus par des individus différents. Ces derniers communiquent, échangent leurs connaissances et collaborent afin de réaliser une tâche commune. La somme des connaissances des différents individus qui forment un groupe est différente des connaissances des individus. La réalisation d'une tâche commune nécessite des échanges de messages, des mises au point et des négociations afin de résoudre d'éventuels conflits.

Les systèmes experts ont contribué à résoudre plusieurs problèmes complexes. Leur mise en place est difficile ainsi que la maîtrise de leur fonctionnement. De plus, la distribution des connaissances nécessaires à la résolution de certains problèmes a fait apparaître certaines de leurs limites.

À partir de ces difficultés et limites, la communauté scientifique a réfléchi à diminuer la complexité de la résolution des problèmes en les distribuant ce qui a donné naissance à l'Intelligence Artificielle Distribuée (IAD) qui s'est intéressée à la distribution de système experts. Le but est de faire interagir et communiquer

des systèmes experts de petites tailles afin de résoudre des problèmes plus importants. La résolution des problèmes complexes est restée une tâche difficile. L'intelligence artificielle distribuée a donné naissance à des architectures plus appropriées. Les activités sont modélisées sous formes d'entités appelées agents. Ces entités sont autonomes et indépendantes et interagissent entre elles. Elles travaillent au sein d'un groupe selon des modes de coopération, de conflits et de concurrence. Les interactions entre ces différents agents peuvent donner naissance à des comportements totalement différents de ceux des entités. Ainsi, un nouveau paradigme « les systèmes multiagents » vient enrichir le domaine de l'intelligence artificielle que nous présentons dans ce qui suit.

## 4.2 Définitions

Les agents et les systèmes multiagents sont la résultante d'une évolution de l'intelligence artificielle. Dans ce qui suit nous définissons ces deux concepts.

### 4.2.1 Agent

Plusieurs définitions de la notion agent existent. La définition la plus large et pouvant correspondre aux humains, aux animaux, aux robots ou aux agents logiciels est : « un agent représente une entité qui agit, c'est-à-dire une entité capable de modifier son environnement ». J. Ferber [1] propose comme définition « un agent est une entité autonome, réelle ou abstraite, qui est capable d'agir sur elle-même et sur son environnement, qui, dans un univers multiagent, peut communiquer avec d'autres agents, et dont le comportement est la conséquence de ses observations, de ses connaissances et des interactions avec les autres agents ». Jennings, Wooldridge et Sycara dans (Jennings *et al.*, 1998) le définissent comme étant « un système informatique, situé dans un environnement, et qui agit d'une façon autonome et flexible pour atteindre les objectifs pour lesquels il a été conçu ».

1. Chercheur et auteur d'ouvrages de références dans le domaine système multiagent.

Une des questions qui se pose fréquemment est « qu'est-ce qu'un agent intelligent ? ». Un agent logiciel est-il intelligent au moment où il démontre des capacités à imiter un comportement humain (Test de Turing [2]) ?

À défaut de donner une définition précise de l'agent intelligent, il est possible d'en dégager les fonctionnalités principales. La définition commune à la plupart des littératures est : « L'agent agit en fonction d'un certain nombre d'informations reçues et perçues de son environnement et en fonction des buts qu'il poursuit. Son comportement donne l'impression qu'il raisonne et que ses actions sont dirigées par ce raisonnement ».

En résumé, un agent est capable de :

- Percevoir, au moins partiellement, son environnement ;
- Agir sur son environnement ;
- Se comporter d'une manière rationnelle : ses actions sont le produit d'un raisonnement ou du moins d'un ensemble de règles de comportement dans le but d'obtenir un état souhaité de l'environnement.

Un agent logiciel n'est-il qu'un objet (au sens objet informatique) ? Un agent logiciel dispose d'une certaine autonomie décisionnelle vis-à-vis des messages qu'il reçoit. Alors qu'un objet ne peut refuser de s'exécuter si un autre objet le lui demande. Un objet donne une réponse parce qu'il fait appel à une de ses méthodes, un agent peut effectuer une action sans qu'on le lui ait demandé. C'est en cela qu'un agent est dit autonome. Il est capable d'un comportement mû par des buts internes.

La différence entre objet et agent provient de la notion de réflexivité organisationnelle que n'a pas à priori un objet contrairement à un agent. A. Cardon [3] avance la distinction suivante : « L'agent est pro-actif et pas l'acteur. La notion de pro-activité est génératrice de celle d'autonomie, qui est plus cognitive

2. Test inventé par Turing en 1950 qui consiste à essayer de deviner si c'est un humain ou une machine qui donne des réponses à un ensemble de questions par l'intermédiaire d'un ordinateur. Si c'est impossible de distinguer la machine de l'homme par les réponses qu'elle fournit, la machine est dite intelligente.

3. Chercheur ayant plusieurs travaux sur la conscience artificielle.

et encore plus générale ». Le terme acteur cité dans cette définition est équivalent à objet. La pro-activité signifie que l'agent cherche à accomplir un objectif et ne reste pas inerte sans activité.

La figure 4.1 donne l'aperçu externe d'un agent, c'est-à-dire l'agent vu en tant que boite noire dont seules les entrées et les sorties sont représentées.

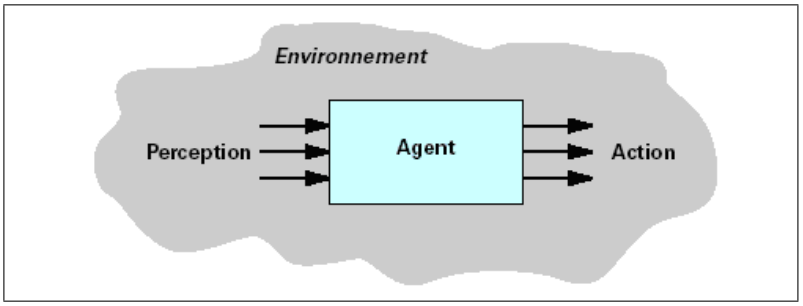



FIGURE 4.1 – Aperçu externe et général d'un agent.

Les capacités de l'agent sont de trois ordres qui répondent aux trois phases générales de réalisation d'une tâche (figure 4.2) :

- Une phase de perception : l'agent perçoit son environnement et met à jour ses représentations internes de l'environnement et des autres agents ;
- Une phase de cognition : l'agent détermine ce qui est à faire (sa tâche) et décide quand et comment le faire. Une large palette de types de raisonnements est utilisée et implémentée, les extrêmes étant les raisonnements réactif et cognitif ;
- Une phase d'action : la réalisation effective des actions qui ont été décidées.

Ces trois phases forment un cycle par le bouclage de la phase action sur la phase perception. L'environnement évolue continûment et offre donc au système, à tout moment, de nouvelles données à percevoir et de nouvelles tâches à réaliser.

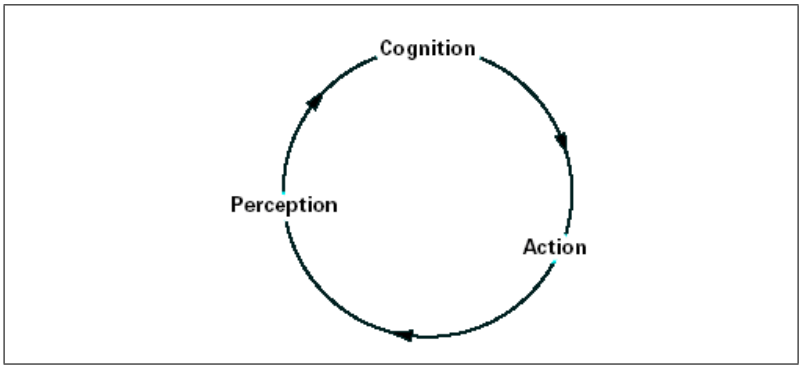



FIGURE 4.2 – Les trois phases générales de réalisation d'une tâche par un agent.

### 4.2.2 Système multiagent

L'agent est destiné à interagir avec ses semblables au sein d'un groupe, dénommé système multiagent (SMA).

Un système multiagent est « un ensemble d'entités qui coordonnent leurs connaissances, buts, expériences et plans pour agir ou résoudre des problèmes, incluant le problème de la coordination inter-agent lui-même » (Bond and Gasser, 1998) ou encore « un monde artificiel peuplé de processus inter-agissants est appelé système multiagent » (Avouris and Gasser, 2003).

Un système multiagent peut donc être défini comme une entité logicielle composée de plusieurs agents intelligents (figure 4.3). Jacques Ferber (Ferber, 1999) défini un système multiagent comme un système composé des éléments suivants :

- Un environnement $E$ identifié et muni d'un système de repérage dans l'espace (souvent Euclidien).
- Un ensemble d'objets $O$ passifs pouvant être perçus, crées, modifiés ou détruits par des agents.
- Un ensemble d'agents $A$ actifs ($A \subseteq O$).
- Un ensemble de relations $R$ qui unissent des objets entre eux.
- Un ensemble d'opérations $Op$ offrant la possibilité aux agents de $A$ de percevoir, produire, consommer, transformer et manipuler des

objets de $O$.

– Un ensemble de lois universelles qui sont des opérateurs chargés de représenter l'application des actions des agents sur le monde et la réaction du monde à ces actions.

Un système multiagent peut être ouvert (cas d'un magasin où des clients entrent et sortent librement) ou fermé (l'ensemble d'agents reste le même comme dans un match de football).

Un système multiagent peut être homogène, c'est-à-dire que tous les agents sont construits sur le même modèle (ex : une réunion de travail, une colonie de fourmis) ou hétérogène, c'est-à-dire que les agents sont de modèles différents et/ou de granularités différentes (ex : un écosystème).

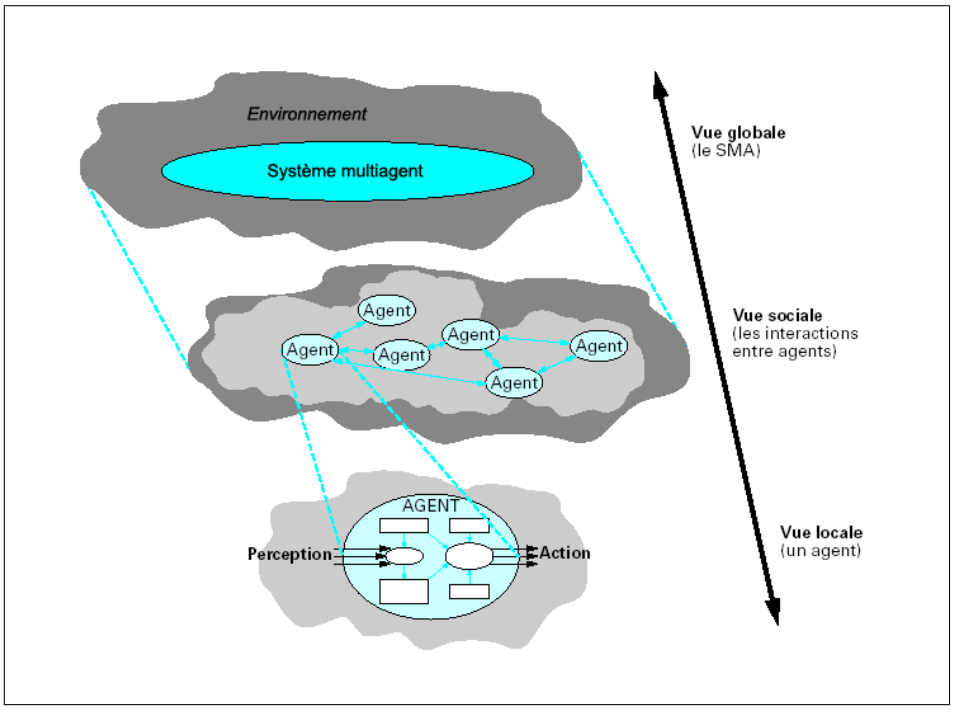



FIGURE 4.3 – Système multiagent vu selon différents niveaux de détail.

Les systèmes multiagents sont utilisés dans plusieurs domaines d'application : commerce électronique (PersonaLogic (Guttman and Maes, 1998), Firefly

(Shardanand and Maes, 1995), Kasbah (Chavez and Maes, 1996)), systèmes industriels distribués (Chaib-draa, 1995; Galland *et al.*, 2006), traitement d'image (Lefèvre *et al.*, 1996; Mazouzi *et al.*, 2007), ordonnancement (Rus *et al.*, 2001; Mouaddib, 2004), recherche d'information sur le web (Enembreck and Barthès, 2004; Falou and Bourdon, 2006)...

### 4.2.3 Agent cognitif

Par extension à la notion agent, on appelle agent cognitif « un agent capable d'autonomie et de flexibilité dans le choix de ses actions pour atteindre un objectif » (Wooldridge and Jennings, 1995). Ce qui distingue donc ce type d'agent des autres est sa capacité de raisonner en choisissant les actions à mettre en œuvre pour atteindre ses objectifs. Il peut être défini par sa capacité à posséder un comportement dirigé par des buts comme dans l'architecture BDI (*Belief-Desire-Intention*) proposée dans (Rao and Georgeff, 1995). Les agents cognitifs peuvent mettre en place des mécanismes de planification afin d'élaborer et d'ordonner une liste d'actions à réaliser pour atteindre un objectif. Ils possèdent une représentation du monde dans laquelle ils évoluent et sont capables de se construire des plans d'actions pour agir sur leur environnement.

### 4.2.4 Agent réactif

Un agent est dit réactif s'il réagit uniquement à la perception de son environnement. Rodney Brooks[4] estime que les comportements intelligents doivent émerger des interactions entre des comportements simples. De ce fait, l'environnement prend une place prépondérante. En effet, dans la plupart des systèmes multiagents réactifs, les agents communiquent uniquement à travers l'environnement. Les agents ont des comportements correspondants à des machines à états finis. Ils réagissent aux stimuli de leur environnement.

Weiss (WEISS, 2000) définit les agents réactifs : « Les agents réactifs basent

4. Directeur du MIT Computer Science and Artificial Intelligence Laboratory

leurs décisions entièrement sur le moment présent, sans aucune référence à des actions ou des événements passés. Ils répondent simplement à des stimuli provenant de l'environnement ».

## 4.3 Simulation multiagent

### 4.3.1 Motivation

Comme déjà introduit dans le premier chapitre plusieurs approches sont possibles pour concevoir des modèles destinés à la simulation informatique. L'approche agent en fait partie. Le besoin de simuler des entités a fait émerger ce paradigme. En effet, il était difficile de modéliser par les modèles classiques (mathématiques ou autre) des systèmes dynamiques formés par une population d'entités individuelles.

La force de la modélisation et la simulation multiagent fait qu'à partir d'interactions simples entre des entités individuelles, il peut résulter des phénomènes émergents difficiles à modéliser autrement. Par exemple, la formation de bancs de poisson ou de nuées d'oiseaux se fait suite à des interactions simples entre les individus. L'évolution de tels phénomènes est assez difficile à comprendre et à prédire. Un système multiagent est une bonne solution pour les modéliser.

L'utilisation d'un système multiagent devient intéressante. Elle permet de décrire aisément des systèmes composés d'entités comportementales. On peut mettre facilement en valeur les comportements et suivre l'évolution des agents. L'interprétation du résultat est plus simple par un observateur humain. D'autant plus que cette approche est flexible c'est-à-dire qu'il est simple d'ajouter des agents au système, de jouer sur la granularité des agents en travaillant avec des agents isolés, des groupes d'agents, des sous-groupes d'agents...

La simulation multiagent permet de valider des généralisations sur des types d'organisations de sociétés afin de découvrir leurs propriétés indépendamment de toute modélisation particulière. Les systèmes construits pour réaliser ces

simulations sont appelés sociétés artificielles.

Cette approche de la simulation est utilisée dans un grand nombre de domaines comme la sociologie (Prietula *et al.*, 1998; Goldspink, 2002), la biologie (Drogoul *et al.*, 1995), l'écologie (Grimm *et al.*, 1999), l'économie (Said *et al.*, 2002), le transport (Vanbergue and Drogoul, 2002; Gruer *et al.*, 2003). Elle remplace progressivement les techniques de simulation traditionnelles comme la micro simulation (Orcutt, 1957), les techniques orientées objets et les techniques d'individus. Ceci est dû à sa compatibilité avec les différents modèles d'individus composés d'entités simples (agents réactifs) ou d'entités complexes (agents cognitifs).

### 4.3.2 Applications

La simulation multiagent a été utilisée dans divers domaines d'applications comme dans les sciences sociales, politiques ou économiques. On peut citer à titre d'exemple : les simulateurs de trafic routier tel que *ARCHISIM*[5], la modélisation et le contrôle de ressources renouvelables tels que dans le projet *COMORE*[6], des marchés boursiers, en météorologie, en écologie...

Dans ce qui suit, nous présentons quelques travaux appliqués et quelques plates-formes de simulation.

#### Robotique Collective Mobile

Il est évident que si dans un tel domaine on voulait faire des expérimentations réelles, il faudrait beaucoup de temps et le coût serait élevé. La simulation multiagent est une excellente alternative pour tester différents algorithmes comportementaux avant de les implémenter dans les robots. Ceci permet de tester différents scénarios sans coût supplémentaire et permet de se concentrer

5. Permet la modélisation de la conduite automobile et simulation de trafic développé par l'équipe MSIS - INRETS d'Arcueil - Chef d'équipe Stéphane Espié.

6. C'est un projet commun INRIA (Unité de Recherche de Sophia-Antipolis) et CNRS, LOV, Laboratoire d'Océanographie de Villefranche sur mer, UMR 7093, CNRS/UPMC.

sur les études comportementales entre différents robots en faisant abstraction des difficultés pouvant apparaître avec les capteurs. Différentes études se sont intéressées à ce type de simulation. On peut citer à titre d'exemple : le problème des robots footballeurs qui a été étudié par plusieurs laboratoires de recherche. Ce problème a donné naissance à la compétition *RoboCup* (RoboCup, web; Kitano *et al.*, 1997). D'autres projets, plus utiles ont vu le jour, comme le projet V.A.H.M.(Véhicule Autonome pour Handicapés Moteur) qui est une application originale de la robotique mobile à l'assistance à la conduite de fauteuils électriques mené par Laboratoire d'Automatique des Systèmes Coopératifs (LASC) (Pruski *et al.*, 2003*a*; Pruski *et al.*, 2003*b*).

### Ecologie et biologie

Ces dernières années, on s'intéresse de plus en plus à notre planète et donc à la gestion de l'environnement. Dans ce contexte, la simulation multiagent est une excellente alternative permettant de faire interagir différents acteurs sur leur environnement.

Le problème de la gestion des ressources naturelles a donné naissance à la plate-forme de simulation multiagent *Cormas* (Cormas, web*a*; Bousquet *et al.*, 1998). Elle a été utilisée dans de nombreux projets concernant la gestion d'une ressource renouvelable (eau, faune sauvage, arbres et bois, sol et érosion, pâturages), les filières de ressources naturelles et les échanges économiques, les dynamiques d'occupation de l'espace. Une liste complète des différents projets est présentée sur le site internet de la plate-forme (Cormas, web*b*).

D'autres plates-formes comme *Mobidyc* (Mobidyc, web) se sont intéressées à l'écologie, la biologie et à l'environnement. Parmi les grands classiques repris sous *Mobidyc*, on peut citer : le cannibalisme chez les poissons (Deangelis *et al.*, 1980), le jeu de la vie de John Conway (Gardner, 1970)...

**Sciences sociales**

Le terme agent est de plus en plus utilisé dans le domaine des sciences sociales. Avec ce nouvel outil de nouvelles possibilités d'analyse sont apparues. Comme exemple de travaux on peut citer : la théorie des jeux (Bousquet *et al.*, 1998), la modélisation des phénomènes urbains (Vanbergue *et al.*, 2000), la dynamique du changement d'opinion dans une population (Deffuant *et al.*, 2002)... Différentes plates-formes offrent des possibilités de simulation des modèles liés aux sciences sociales telles que *Moduleco* (Moduleco, web; Phan, 2002), *Ascape* (Ascape, web; Parker, 2001). Et récemment deux plates-formes sont de plus en plus utilisées et ont un succès grandissant, ce sont *Mason* (Mason, web; Luke *et al.*, 2004) et *RePast* (RePast, web).

## 4.4 Simulation multiagent distribuée

### 4.4.1 Méthodologies pour la simulation multiagent distribuée

La simulation a joué un rôle important dans la recherche et le développement des systèmes multiagents. Elle permet le contrôle des conditions expérimentales et facilite la réplique des résultats d'une telle manière qu'il est difficile, voire parfois impossible, de le faire avec un prototype ou un système in situ. Elle Libère ainsi le concepteur d'agents ou le chercheur pour se concentrer sur les aspects importants d'un système. Comme les chercheurs essayent toujours de simuler des systèmes multiagents de plus grande taille et plus complexes, la distribution de la simulation est devenue attrayante. Une telle approche simplifie l'intégration d'agents hétérogènes et exploite le parallélisme des systèmes multiagents. De cette façon, différents composants de simulation peuvent être distribués afin de mieux utiliser les ressources informatiques disponibles. Cependant la distribution d'un système multiagent présente des défis particuliers à surmonter.

Un des problèmes concerne les ressources de calculs. Les ressources requises

pour les simulations multiagents excèdent de loin les capacités des systèmes séquentiels conventionnels. Chaque agent est typiquement un système complexe à part entière (du fait de sa perception, son programme, ses inférences, ses capacités, etc.). D'où la nécessité de plus en plus de ressources matérielles. De plus beaucoup d'agents peuvent avoir besoin d'examiner le comportement du système de manière générale ou même le comportement d'un agent particulier (Sloman, 1998). Une solution possible est d'essayer d'exploiter le parallélisme par essence même des systèmes multiagents. Cependant, les travaux réalisés jusqu'ici tentent d'employer différentes approches ad hoc par rapport à la simulation parallèle. Par exemple, distribuer les agents sur un réseau de machines. Les interactions entre les agents se font via des protocoles de communications. Mais, les résultats ont été décevants (Baxter and Hepplewhite, 1996; Vincent *et al.*, 1998). Ces limitations ont mené les chercheurs à exploiter les techniques de la simulation distribuée pour les systèmes basés agents.

En général, on a besoin d'une plate-forme. Cette dernière doit être capable de supporter une large variété d'agent et d'environnement afin d'avoir une plus large possibilité d'utilisation.

Mais comment modéliser les agents et leur environnement ? Différentes approches ont été développées pour exploiter le parallélisme selon différentes granularités dans les problèmes de simulation (Fujimoto, 1990; Ferscha and Tripathi, 1994). La simulation distribuée dirigée par les événements est particulièrement convenable pour la modélisation des systèmes avec un parallélisme asynchrone inhérent comme dans les systèmes basés agents. Cette approche divise le modèle de la simulation sur un réseau de processus concurrents (*Logical Processes - LPs*). Chacun maintient une portion disjointe de l'environnement.

On distingue deux d'événements : événement interne qui n'agit que sur l'environnement local appartenant au LP et événement externe qui peut également influencer les autres LPs. Les événements externes sont modélisés par des messages estampillés échangés entre les différents LPs impliqués. Les agents et

l'environnement sont modélisés comme étant un ou plusieurs processus.

Un deuxième problème est de définir les zones d'intérêts de chaque agent afin d'éviter une communication *broadcast*. Cette problématique a été relevée en premier dans le domaine de la simulation temps réel à grande échelle. C'est là où a été défini pour la première fois le mot « gestion d'intérêt » (*Interest Management - IM*) (Morse, 1996). Les techniques de gestion d'intérêt utilisent des mécanismes de filtrage basées sur des expressions d'intérêts (*Interest Expresssion - IE*) pour fournir les processus dans la simulation avec seulement le sous-ensemble des informations approprié. Les données qui intéressent un processus sont désignées sous le nom de domaine d'intérêt (*Domain of Interest - DOI*).

Différents schémas de gestion d'intérêt ont été conçus. Ils utilisent différents modèles de communication et de filtrage (Morse, 1996). Dans la majorité des systèmes existants, la gestion d'intérêt s'effectue via l'utilisation d'adressage *multicast* d'IP par lequel les données sont envoyées à un sous-réseau choisi parmi tous les récepteurs potentiels. Un groupe de *multicast* est défini pour chaque type de message ou zone de l'environnement. HLA utilise la construction de l'espace de cheminement (*routing space*) selon un système de coordonnées multidimensionnel. Les fédérés manifestent par ce système leurs intérêts pour la réception des données de la région à laquelle se sont abonnés à l'avance (*subscription regions*) ou déclarent leur responsabilité pour la publication des données (*update regions*).

### 4.4.2 Architectures de simulation multiagent distribuée

Inspirée du monde des jeux vidéo en réseau et de la réalité virtuelle (Michel *et al.*, 2002), l'approche multiagent peut proposer des solutions élégantes à la distribution d'une simulation pour mettre en interaction des acteurs distants à travers un réseau. Le but est donc de trouver la meilleure solution appropriée pour distribuer la simulation.

Par analogie avec les techniques utilisées dans la réalité virtuelle, plusieurs solutions sont possibles :

### Architecture centralisée

Avec une architecture centralisée (voir figure 4.4), la simulation est exécutée sur une seule machine, la visualisation et le contrôle se font à distance sur les machines clientes. Pour cela il suffit de placer des agents « écouteurs » à l'écoute de l'interaction d'opérateurs distants sur une des machines clientes, et des agents « projectionnistes » projetant les modifications aux autres machines clientes. Pour cette solution, il n'y a pas d'ambiguïté sur l'état du monde. Cependant, le trafic réseau est trop important vu la quantité d'informations échangées.

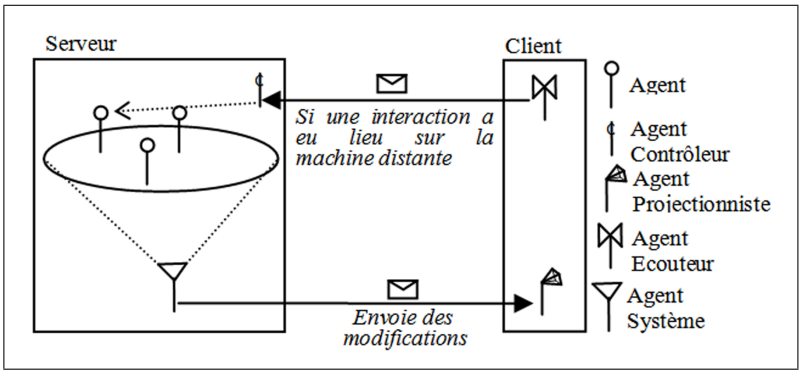



FIGURE 4.4 – Exemple d'une architecture centralisée.

### Architecture par duplication

Avec une architecture par duplication (voir figure 4.5), très utilisée dans le monde des jeux vidéo, les échanges des messages sont limités. Chaque machine a une copie de la simulation dans son ensemble. Mais il peut y avoir divergences entre les mondes simulés du fait, par exemple, d'une inévitable latence. Ceci ne permet pas aux différents événements utilisateurs d'être pris en compte au même moment sur l'ensemble des machines. Pour cela, une machine dite machine « maître » est sélectionnée. Elle détient la vérité sur le monde simulé. Les autres machines seront appelées « duplica ». Un agent sera chargé d'envoyer toutes les positions des agents de la simulation de la machine « maître » aux machines clientes toutes les x secondes prédéfinies suivant le type de synchronisation voulu

(forte, moyenne ou faible).

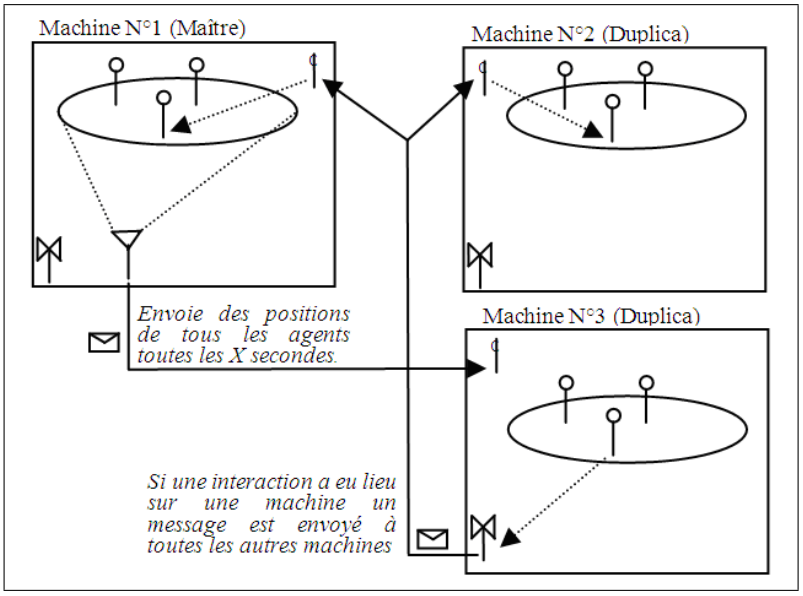



FIGURE 4.5 – Exemple d'une architecture par duplication.

### Architecture distribuée

Avec une architecture distribuée, les agents doivent être autonomes. Les interactions entre eux se feront par échange de messages. Chaque plate-forme a un agent appelé « scheduler » chargé d'indiquer le début et la fin de chaque pas de temps. Cet agent aura la tâche d'ordonner l'évolution et le comportement de chaque agent. Un agent « scheduler » principal sera chargé d'envoyer un message aux autres agents « scheduler » afin de minimiser le nombre de messages échangés de $n$ (scheduler $\rightarrow$ $n$ agents) à 1 (scheduler principal $\rightarrow$ scheduler). La figure 4.6 illustre une telle situation. Étant donné que tous les agents envoient des messages à l'environnement, il serait judicieux de le distribuer pour éviter qu'il ne devienne un goulot d'étranglement de la simulation. Pour cela il faudrait penser à faire migrer les agents d'une plate-forme à une autre car il est préférable que les agents communicants ensemble se trouvent sur une même plate-forme. En particulier, dans le cas d'un environnement distribué, il est préférable que les agents évoluant dans la même zone soient sur la même plate-forme encore une

fois pour minimiser les transactions de messages via le réseau. Mais, une telle distribution dynamique qui consiste à prendre en compte la mobilité des agents et éventuellement l'optimisation de cette distribution est complexe à mettre en œuvre.

D'autre part, pour que les agents distribués physiquement sur différentes machines puissent communiquer entre eux, il faudrait passer par :

- Un faciliteur : il connaît les caractéristiques de tous les agents du système multiagent. Il tient cette connaissance à disposition des autres agents en fournissant un service de pages jaunes.

ou bien par

- Un courtier (broker) : il joue le rôle d'intermédiaire entre les agents. Il permet la recherche de partenaires pour toute coopération.

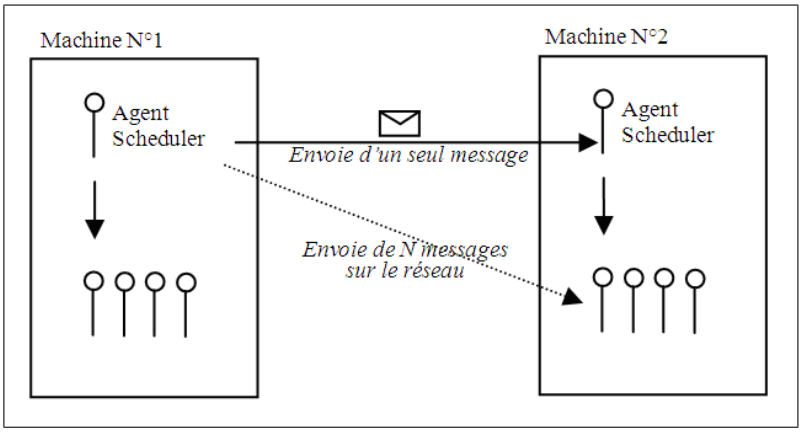



FIGURE 4.6 – Exemple d'une architecture distribuée.

Ce type d'agent est très sollicité et une surcharge en communications à traiter peut réduire les performances du système multiagent. Le volume de ses connaissances est proportionnel à la cardinalité du système multiagent et donc au temps de recherche et de sélection des données correspondant à une requête donnée (Vercouter, 2001). Selon (Nodine *et al.*, 1999), le système est acceptable pour une douzaine d'agents. Au-delà, le système devient vulnérable puisqu'il

dépend du bon fonctionnement d'un seul agent et des canaux de communication vers ce dernier. D'après (Vercouter, 2001), il peut être intéressant de se passer d'un agent intermédiaire (faciliteur ou *broker*). L'idée est que chaque agent doit avoir une représentation locale des autres agents et d'impliquer plusieurs agents dans la gestion d'ouverture du système (ajout, retrait, évolution d'un agent). Cette proposition reste assez coûteuse en termes de messages échangés. Mais, c'est le prix à payer pour avoir un système robuste.

### 4.4.3 Exemples de distribution avec HLA

Ces dernières années, différents travaux se sont orientés vers la distribution des simulations multiagents avec le protocole HLA. Une manière de faire est de considérer chaque fédéré comme un agent. Les fédérés communiquent entre eux via le RTI pour réaliser leurs buts. Chaque fédéré maintient son sous-ensemble de l'environnement. Les agents peuvent communiquer grâce aux caractéristiques de HLA (Kuhl *et al.*, 1999). Il s'agit des opérations de publications et réceptions des changements des attributs des classes une fois ceux-ci découverts.

Avec HLA, les fédérés publient les mises à jour à la fédération. Chaque fédéré peut s'enregistrer auprès de la fédération afin de recevoir les mises à jour qui l'intéressent. Les mises à jour sont faites d'une façon potentiellement synchrone avec un respect de l'horloge virtuelle partagée. Les fédérés ont besoin de synchroniser leurs opérations avec l'avancement de l'horloge virtuelle. Un fédéré peut retarder l'exécution d'autres fédérés tant qu'il n'a pas accompli une action qui doit se produire à un temps $t_1$ avant qu'aucun fédéré ne puisse avancer au pas de temps $t_2$.

Dans la littérature, on trouve quelques tentatives de distribution des simulations multiagents en utilisant le protocole HLA. Nous pouvons citer les travaux de (Lees *et al.*, 2003; Kratkiewicz *et al.*, 2004; Lees *et al.*, 2004; Lees *et al.*, 2007).

L'approche adoptée par (Kratkiewicz *et al.*, 2004) est de découper la société d'agents sur la plate-forme *Cougaar* (*Cognitive Agent Architecture*) (Cougaar,

web) en deux parties, voire plus, qui sont modélisées par la même organisation. Les sociétés opèrent comme des fédérés de HLA. Elles sont connectées à travers le RTI. Plusieurs réalisations ont été testées avec diverses implémentations de RTI (Pitch 1516LE, Pitch 1.3 LE, DMSO NG1.3). Le travail réalisé montre comment relier une société d'agents au RTI HLA, la synchronisation de la société d'agents avec celle du RTI HLA et comment tracer les agents, les objets et les actions de la société d'agent avec celles des objets et des interactions dans HLA.

Les travaux suivants (Lees *et al.*, 2003) et (Lees *et al.*, 2004) ont montré comment HLA peut être employé pour distribuer une simulation multiagent. La bibliothèque résultante incorpore une solution simple de résolution de problème de conflits dans un environnement distribué qui exploite les services de gestion de propriété de HLA (Ownership Management). Un des problèmes est la mise à jour de l'environnement partagé. Le service de la gestion de distribution des données (Data Distribution Management) de HLA n'a pas été utilisé. Des travaux sur ce point ont été élaborés par (Logan and Theodoropoulos, 2001). Pour résoudre ce problème, des outils additionnels pour la collecte des données et le débogage sont en cours de réalisation.

Ces différents essais sont encore légers et plusieurs problèmes restent à surmonter. En outres, adopter une telle solution est coûteux en termes de temps et d'énergie. L'architecture d'une telle distribution est basée sur le modèle Client/Serveur. Le RTI joue le rôle d'un serveur. Ce dernier devient rapidement un goulot d'étranglement puisque tous les messages passent par lui-même.

## 4.5 Conclusion

Une genèse sur les systèmes multiagents ainsi que quelques types d'applications sont présentées dans la première partie de ce chapitre. L'utilité étant de montrer la grande diversité et la richesse des applications adoptant le paradigme agent. Ce dernier, au fur et à mesure des années s'est mûri et des plates-formes spécifiques ont vu le jour. Elles sont de plus en plus utilisées pour développer

des simulateurs. Mais, les limites se font ressentir lorsque celles-ci deviennent de plus en plus étoffées, lorsque le nombre des acteurs augmente ou bien lorsque les actions des acteurs nécessitent une grande quantité de ressources. On a vu qu'un des moyens possibles pour palier à cet inconvénient est de distribuer la simulation afin de répartir les diverses charges de chaque élément qui la constitue.

Différentes solutions basées sur le protocole HLA ont été présentées. Cependant, ces dernières restent sommaires et souffrent de lacunes. Néanmoins, il est intéressant de noter sa différence avec d'autres techniques par ses propriétés de réutilisation et de gestion de temps présentées dans cette norme. La réutilisation des objets partagés est facilitée par la définition des documents décrivant la structure de chaque objet par un fédéré HLA.

Toutefois, il est important de rappeler que HLA est une norme et non pas une implémentation. Différentes implémentations existent : Pitch 1516LE, Pitch 1.3 LE, DMSO NG1.3. L'inconvénient majeur est l'impossibilité d'effectuer une communication point à point entre les agents dans une architecture multiagent développée au-dessus de HLA. Ceci est dicté par la norme même de HLA. Tout message doit obligatoirement passer par le RTI. D'où le point de faiblesse du système puisque ce dernier peut devenir rapidement un goulot d'étranglement. La robustesse du système est alors lourdement affectée.

# Deuxième partie

# Contribution

# 5

# PHAC, principes et opportunités pour la simulation distribuée

## Sommaire



DANS ce chapitre, nous présentons une taxonomie plus raffinée des systèmes de l'informatique répartie. Ensuite, nous décrirons notre plate-forme *PHAC* (*A P2P-based Highly Available Computing Framework*) qui a pour but de construire un réseau de pair à pair hautement disponible pour être utilisé à des fins de calculs. Nous décrirons en détail l'organisation des pairs et nous présentons les performances attendues de cette plate-forme.

## 5.1 Introduction

Le rôle d'une plate-forme distribuée est de fournir un ensemble de services essentiels pour permettre l'exécution d'une simulation distribuée. Dans le contexte de la simulation multiagent, un des points importants est la gestion des agents et de leurs interactions. Ceci nécessite un contrôle des communications et un contrôle des migrations des agents entre différents ordinateurs. Un autre point important est la gestion du temps qui permet de préserver un système cohérent.

Notre but est d'offrir une plate-forme pour la simulation distribuée et exploitant au mieux les ressources informatiques disponibles. Une étude menée par l'*Omni Consulting Group*[1] montre que 47% des ressources sont non utilisées dans une entreprise. Or jusqu'à présent, les architectures utilisées sont coûteuses en termes d'investissement financier et sont toutes statiques.

Dans ce qui suit nous présentons les principes et les différents aspects de la mise en œuvre de la plate-forme PHAC[2] (Cabani *et al.*, 2007*b*; Cabani *et al.*, 2007*c*) qui constitue une plate-forme flexible pour distribuer une simulation.

## 5.2 Choix technologique

Jusqu'à présent et comme déjà vu dans les chapitres précédents, les architectures utilisées pour distribuer une simulation étaient coûteuses à mettre en place. La plupart se basent sur une architecture Client/Serveur qui souffre de plusieurs inconvénients.

Avec l'apparition du paradigme pair-à-pair, de nouvelles possibilités se sont offertes. Nous avons pensé à utiliser cette technologie pour distribuer des simulations multiagents. Vu les caractéristiques des réseaux pair-à-pair, la particularité et la spécificité de la simulation multiagent distribuée, plusieurs défis sont à surmonter .

1. Société de consulting. http ://www.omniconsultinggroup.com/
2. Pour *A P2P-based Highly Available Computing Framework*

La première démarche est de se reporter à la taxonomie des applications pair-à-pair afin de vérifier si notre démarche peut être adoptée ou non. Or la taxonomie existante dans la littérature est simpliste et grossière. Plusieurs points sont à relever :

- Premièrement, toutes les applications parallèles sont mises dans la même classe. Ce n'est pas suffisant pour identifier les applications qui peuvent être distribuées sur une architecture pair-à-pair. Les applications parallèles peuvent comporter la distribution : de l'application sur plusieurs nœuds comme dans le cas des grappes de calculs (*cluster computing*), à la localisation des données stockées, ou inversement la distribution des éléments pour les applications résidentes comme dans le calcul de grille (*grid computing*). Dans les deux cas les coûts sont complètement différents.
- Deuxièmement, un des points importants ignoré dans l'actuelle taxonomie est le besoin ou pas de la synchronisation. Le fait que les nœuds soient faiblement ou fortement couplés influence sensiblement les résultats. Ce critère est donc important lors du choix ou du rejet de la solution pair-à-pair. De même, on note que si l'application est fortement couplée, le pair-à-pair ne peut pas être un bon choix.
- On peut remarquer aussi qu'aucune importance n'a été donnée à la bande passante. Pourtant des études (Andersen *et al.*, 2001; Saroiu *et al.*, 2002; Srivatsa *et al.*, 2006) prouvent que les systèmes pair-à-pair sont extrêmement hétérogènes ce qui n'est pas le cas pour les grilles de calcul.

Pour les raisons ci-dessus, nous avons proposé une nouvelle taxonomie (Cabani *et al.*, 2006) plus raffinée (voir figure 5.1). Elle propose de faire le choix entre une solution pair-à-pair ou grille de calcul surtout qu'une certaine ambiguïté entour ces deux paradigmes. Ces derniers ont par ailleurs tendance à converger dans le futur. La nouvelle taxonomie peut aider les concepteurs et développeurs

à fixer leurs choix technologiques.

La nouvelle taxonomie est définie selon deux dimensions. La première est liée aux besoins et capacités de l'infrastructure. Ceux-ci incluent l'architecture, le domaine d'application et l'interaction entre les différents nœuds. La deuxième dimension se relie aux diverses contraintes applicatives ignorées jusqu'ici. Or, ces critères influencent directement les perspectives des applications déployées. Ces critères sont :

- L'interconnectivité : les réseaux pair-à-pair se distinguent par la présence d'une connectivité volatile. Chaque pair peut joindre ou quitter le réseau sans préavis. Par conséquent, faiblement ou fortement interconnecté, influence directement les résultats attendus et les techniques de synchronisation ne sont pas les mêmes (Garg, 2004).
- La taille des données : le taux et la taille des données qui sont transférées entre les différents nœuds sont des points éliminatoires pour le choix du pair à pair. Les techniques existantes d'optimisation sur les réseaux pair-à-pair ne permettent pas de faire des recherches par mot-clé à grande échelle puisque la bande passante exigée par de telles recherches excède la capacité disponible sur les réseaux pair-à-pair.
- La bande passante : c'est un autre critère critique à considérer. Il peut influencer considérablement les temps de transfert des données entre les nœuds des réseaux pair-à-pair car ces réseaux sont asymétriques.

Cette nouvelle taxonomie est utile pour choisir parmi les deux paradigmes pair-à-pair et grille de calcul surtout qu'à première vue, ils se ressemblent. Pour la prise de décision trois étapes sont nécessaires :

- Après avoir défini l'objectif de l'application, le développeur choisit son type : calcul distribué, partage de contenu et de fichiers ou col-

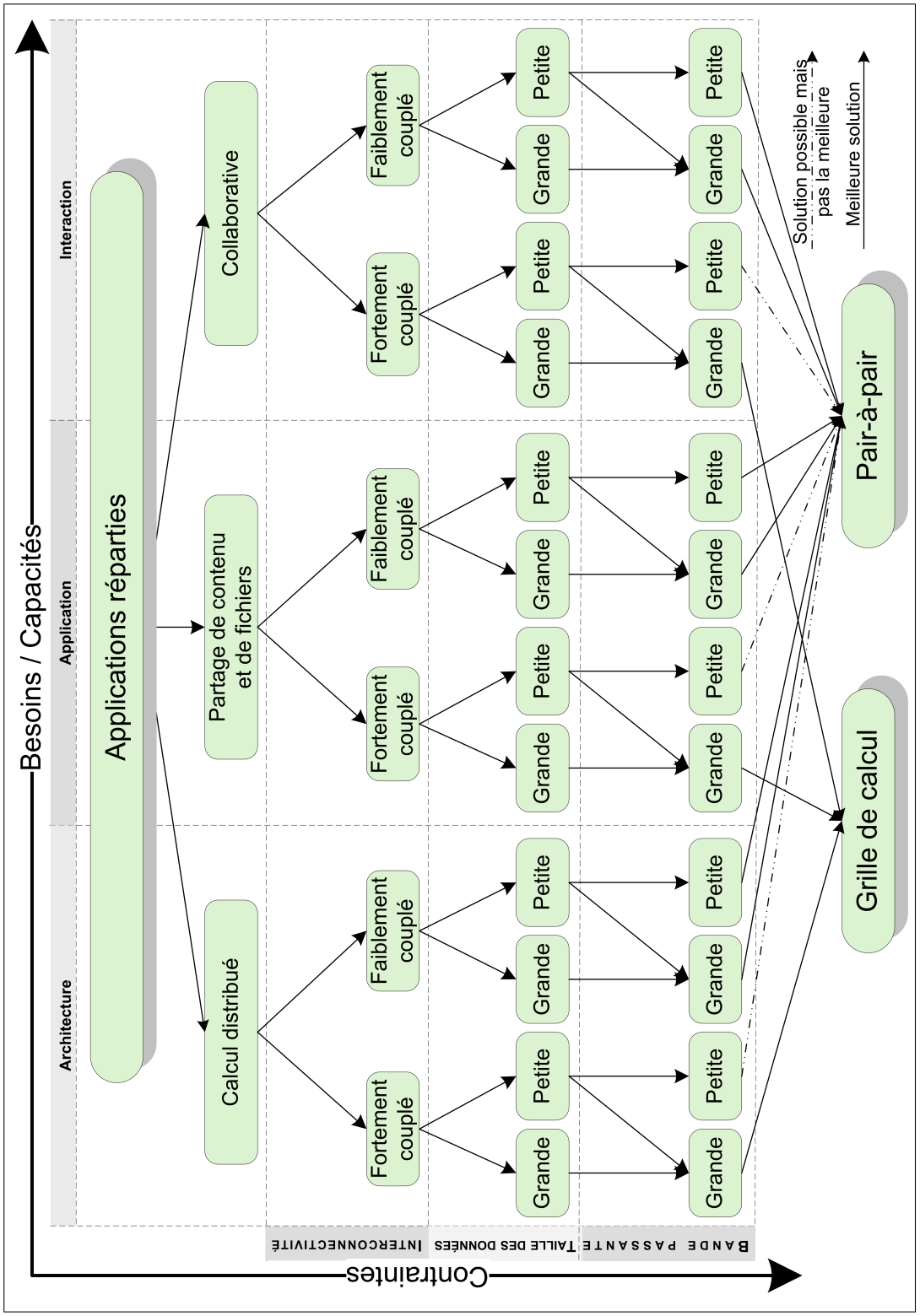



FIGURE 5.1 – Taxonomie améliorée des systèmes de l'informatique répartie (Cabani *et al.*, 2006).

laboratif. A cette étape, la première dimension (besoins/capacités) est fixée.
- Il faut ensuite déterminer les spécifications de l'application selon les trois critères déjà prédéfinis (voir ci-dessus).
- Enfin, en se reportant à la taxonomie, le meilleur paradigme entre pair-à-pair et grille de calcul, est proposé.

## 5.3 La plate-forme PHAC

### 5.3.1 Introduction

Comme déjà évoqué il n'existait jusqu'à présent aucun environnement prenant en compte la disparité des pairs et destiné à être utilisé pour la simulation distribuée. Afin de remédier à ce manque, nous proposons la plate-forme PHAC (Cabani *et al.*, 2007*b*; Cabani *et al.*, 2007*c*) pour *A P2P-based Highly Available Computing Framework*. Le but est d'offrir un réseau formé de pairs hautement disponibles. Le réseau est formé par des pairs qui sont connectés au réseau pendant leur période d'inactivité. Si un pair quitte le réseau inopinément, ceci ne devrait pas gêner le bon déroulement de l'application puisque chaque tâche est répliquée sur d'autres pairs.

PHAC est basé sur la plate-forme JXTA de *Sun Microsystems*. Les fonctions de base pour la manipulation des pairs sont héritées de JXTA. La topologie adoptée est le modèle pur structuré.

Dans ce qui suit, nous présentons l'organisation des pairs au sein de notre plate-forme.

### 5.3.2 Classification des pairs

Nous commençons par introduire les notations suivantes :

Soit $P$ l'ensemble des pairs disponibles.

Nous appellerons par la suite critère d'un pair toute caractéristique de ce pair pouvant influencer les traitements (CPU, RAM, bande passante...).

Hypothèse : Tout pair a au moins un critère.

Pour un critère donné $ci$, nous répartissons en trois familles les pairs selon leurs capacités dans ce critère :

- Capacité haute (H) ;
- Capacité moyenne (M) ;
- Capacité faible (F).

Les pairs sont affectés selon leurs fiabilités à trois zones (voir figure 5.2) :

- Partenaires fiables (zone 1) ;
- Partenaires amis (zone 2) ;
- Autres partenaires (zone 3).

Pour $j \in \{H, M, F\}$ et $l \in \{1, 2, 3\}$, nous définissons $P_{ci}^{j,l}$ comme l'ensemble des pairs répondant au critère $ci$ avec capacité $j$ et une une localité $l$.

Posons :

$$P_{ci} = \bigcup_{\substack{j \in \{H,M,F\} \\ l \in \{1,2,3\}}} P_{ci}^{j,l} \ . \tag{5.1}$$

**Propriété 1**

*On a :*

$$P = \bigcup_{1 \leq i \leq n} P_{ci} = \bigcup_{l \in \{1,2,3\}} P_l \ . \tag{5.2}$$

*Avec*

- $P_{ci}$ *est un groupe de pairs formé selon un critère.*
- $P_l$ *: tous les pairs appartenant à la zone* $l$ *des pairs.*

**Démonstration :** Il suffit d'observer que tout pair a au-moins un critère et se situe dans l'une des trois zones 1, 2 ou 3.

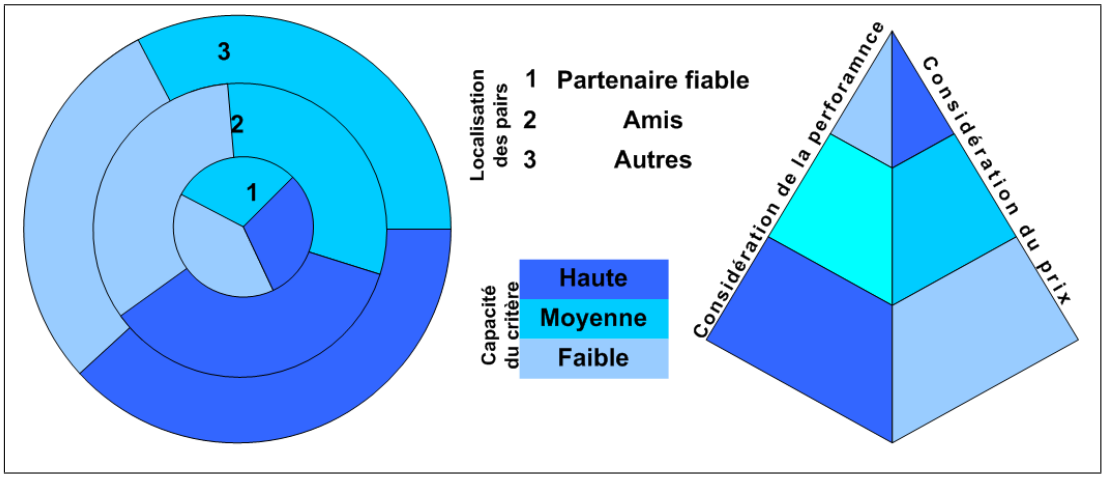



FIGURE 5.2 – Organisation des pairs dans PHAC.

### 5.3.3 Objectif

Soit un problème défini par un utilisateur. Celui-ci souhaite disposer de pairs ayant certains critères minimaux et des ressources spécifiques. Ces pairs formeront un groupe appelé groupe de pairs attendus $E_{JPG}$ (*ExpectedJobPeerGroup*). Le système lui alloue un JobPeerGroup ($JPG$). Ce dernier se compose des pairs ayant des critères et ressources qui serviront à effectuer du calcul.

L'objectif est que l'actuel JobPeerGroup (*Actual JobPeerGroup* $= A_{JPG}$) soit le plus proche possible du *JobPeerGroup* attendu par l'utilisateur en minimisant le coût et en augmentant les performances.

$$A_{JPG} \approx E_{JPG} \ . \tag{5.3}$$

où :
Prix($A_{JPG}$) est minimum.
Performance($A_{JPG}$) est maximum.

Remarquons que $A_{JPG}$ sera composé de pairs disposant de critères $A^{R}_{JPG}$ et de ressources $A^{P_{ci}}_{JPG}$.

On désigne par $R$ l'ensemble des pairs disposant de ressources particulières.

Ces ressources peuvent être exigées pour l'accomplissement d'un calcul. On a $R \subset P$. Par exemple : $R = \{serveursBD, serveursHTTP...\}$

Pour former l'*ActualJobPeerGroup*, on utilise l'algorithme suivant :

**Algorithme 1** : Formation de l'actuel JopPeerGroup

**début**
REM | /* Initialisation */
| $A^{R}_{JPG} \leftarrow \{\}$
| $A^{P_{ci}}_{JPG} \leftarrow \{\}$
REM | /* Choix des pairs ayant des ressources spécifiques */
| **pour tous les** *éléments de* $E_{Rt}$ **faire**
| | $P_i \in P_{Rt} : \forall i, j; i \neq j; 0 \leq i, j \leq |P|; l, l' = 1, 2, 3$ tel que $P^{l}_{i} \leq P^{l'}_{j}$;
| | $A^{R}_{JPG} \leftarrow A^{R}_{JPG} \cup P_i$;
| **fin**
REM | /* Choix des pairs ayant les ressources requis */
| **pour chaque** $E_{P_{ci}} <>$ *"undefined"* **faire**
| | $P_x \in P_{ci} : \forall x, y; x \neq y; k = H, M, L; P^{ci}_{x} \leq P^{ci}_{y} et P^{ci}_{x} = k$;
| | $A^{P_{ci}}_{JPG} \leftarrow A^{P_{ci}}_{JPG} \cup P_x$;
| **fin**
| **retourner** $A^{R}_{JPG} + A^{P_{ci}}_{JPG}$
**fin**

Pour former le *RedundancyPeerGroup*, on utilise l'algorithme suivant :

**Algorithme 2** : Formation du RedundancyPeerGroup

**début**
REM | /* Formation du RedundancyPeerGroup */
| **retourner** $P - AJPG$
**fin**

Selon les besoins de l'utilisateur, il est possible de classer les pairs appartenant au *RedundancyPeerGroup* selon leurs capacités pondérées aux priorités prédéfinies par lui même. Pour ceci, on utilise l'algorithme ci-dessous :

**Algorithme 3** : Formation du ClusteredRedundancyPeerGroup

```
début
REM | /* Initialisation */
    | UnRPG ← P − AJPG
    | CRPG ← {}
    | pour tous les pair de UnRPG faire
    |   score[pair] ← 0;
    | fin
REM | /* Calcul du score */
    | pour tous les pair de UnRPG faire
REM |   /* Chaque critère a une prioriété prédéfinie p_i; ∑ p_i = 1 */
    |   pour chaque critère faire
    |     score[pair] ← score[pair] + p_i * P_ci;
    |   fin
    | fin
REM | /* Classification des pairs */
    | CRPG ← Tri(score)
    | return CRPG
fin
```

### 5.3.4 Performance

#### Notation

On définit les paramètres suivant :

Trois différent types de processeurs :

- H : processeur très rapide.
- M : processeur rapide.
- L : processeur lent.

Soit $P_l^i$ le nombre de pairs appartenant à la $i^{eme}$ zone avec un processeur de type $j$ ($l \in \{1,2,3\}\,; i \in \{H,M,L\}$).

On définit $T_H$, $T_M$ et $T_L$ le temps d'exécution d'une instruction sur, respectivement, les processeurs $P^H$, $P^M$ et $P^L$.

$T_H$, $T_M$ et $T_L$ satisfont les conditions suivantes : $T_H < T_M < T_L$.

Soit $N$ le nombre total des instructions qui seront exécutées lors de l'exécution d'un programme.

Soit $N_i$ le nombre d'instructions exécutées par $P_i$.

**Temps d'exécution**

Les temps de calcul d'un programme sur un seul pair ainsi que sur P pairs sont :

**Cas 1 :** un pair avec un processeur de type H.

$$ET_1 = N.T_H \ . \tag{5.4}$$

**Cas 2 :** avec $P$ pairs appartenant à la localisation 1, 2 et 3.

$$P = \sum_{i=1}^{3} \left( P_i^H + P_i^M + P_i^L \right) ; \ N = \sum_{i=1}^{P} N_i$$

$$ET_2 = \max_{i=1,\cdots,P} (N_i T_i + t_i) \ . \tag{5.5}$$

où $t_i$ le temps nécessaire au transfère entre le pair maître et les autres pairs. Ce temps dépend de la capacité et de la charge de la bande passante.

À partir des équations (5.4) et (5.5), on peut déterminer l'accélération (*speed up*). Il s'agit de voir combien de fois l'algorithme parallèle est plus rapide que dans le cas séquentiel. Si l'accélération est inférieure à un, c'est que l'algorithme séquentiel donne de meilleurs temps de calculs.

$$S_1 = \frac{ET_1}{ET_2} = \frac{N.T_H}{\max_{i=1,\cdots,P} (N_i T_i)} \tag{5.6}$$

Dans le meilleur des cas, si $N_i = \frac{N}{P}$ et $P = P_1^H$ alors l'accélération sera de $P$ fois. Dans le cas où un pair quitterait le réseau pair-à-pair alors $ET_2 \to \infty$ et $S_1 \to 0$.

## 5.4 Caratéristiques

### 5.4.1 Topologie

*PHAC* offre un réseau organisé selon une topologie « pur structuré ». Les différents pairs peuvent communiquer directement entre eux sans passer par aucun

serveur. Les pairs formant le réseau sont volontaires. Les pairs voulant participer à *PHAC* n'ont qu'à lancer un daemon [3]. Ces différents pairs se mettent à disposition d'un éventuel pair sollicitant du calcul. Un tel pair est appelé pair maître. Il a la tâche de former le réseau pair-à-pair en cherchant les pairs disponibles par la technique de *bootstrapping* [4]. Ensuite, il a la charge de construire les deux groupes de pairs *JobPeerGroup* et *RedundancyPeerGroup* (voir paragraphe 5.4.2 ci-dessous). Après cette phase d'initialisation, le pair maître devient un pair comme tous les autres. Chaque pair a une connaissance globale du réseau et des affectations des sous-tâches. Ce choix a été dicté par un souci de respecter la définition même d'un modèle pur où aucun serveur ne doit exister. Un exemple est illustré dans la figure 5.3.

```
<peers>
  <JobPeerGroup>
    <Peer Name="Peer1" Adress="adresseDuPair1">
      <param1>0</param1>
      <param2>5</param2>
    </Peer>
    <Peer Name="Peer2" Adress="adresseDuPair2">
      <param1>6</param1>
      <param2>10</param2>
    </Peer>
  </JobPeerGroup>
  <RedundancyPeerGroup/>
</peers>
```

FIGURE 5.3 – Exemple de fichier de configuration.

Ainsi, au lancement, on a :

- Un pair sollicitant l'aide d'autres pairs pour effectuer une tâche. Ce pair est appelé pair maître ou encore *RequestPeer*.
- Des pairs exécutant le daemon sont à disposition de tout pair sollicitant du calcul. Leur statut est déclaré occupé tant qu'ils ont une tâche à réaliser. Il change à libre dès qu'ils ont fini les tâches qui

3. Abréviation de Disk And Executing MONitor. C'est un programme qui s'exécute en tâche de fond et qui fournit un service.

4. il s'agit d'une technique de formation du réseau pair-à-pair par la technique d'inondation du réseau de requêtes de proche en proche.

leurs ont été attribuées. De tels pairs sont appelés *pairs exécutants* ou encore *ExecutantPeers*.

### 5.4.2 Groupes de pairs

Jusqu'ici les groupes de pairs avaient pour objectif de partager des intérêts communs ou d'établir un espace sécurisé sans prendre en compte la dissymétrie et l'hétérogénéité des pairs.

Afin de répondre à notre objectif qui est d'offrir un réseau de pair à pair pouvant supporter la simulation multiagent distribuée, nous proposons de prêter une attention particulière à la configuration des pairs. Ceci dans le dessein de prendre en compte et de gérer au mieux les disparités qui peuvent exister entre les différents pairs. En effet, les pairs influencent directement les temps de calcul, la fiabilité et la robustesse du système. Comme le choix des pairs est crucial, nous avons défini (Cabani *et al.*, 2005*a*) deux types de groupes particuliers qui sont le *JobPeerGroup* et le *RedundancyPeerGroup*.

Le rôle du *JobPeerGroup* est d'effectuer les calculs. Les pairs doivent avoir les caractéristiques particulières suivantes :

- Une large bande passante. Elle permettra d'échanger facilement les données et/ou résultats entres les différents pairs sans qu'il y est un embouteillage dû au faible débit.
- Bonnes ressources matérielles (processeur, mémoire, espace disque).
- Durée de fonctionnement la plus élevée. En effet, il faut choisir les pairs qui sont les plus disponibles et connectés à l'Internet.
- Le temps de propagation en boucle (Round-Trip delay Time) : c'est le temps correspondant à la durée en millisecondes d'un aller-retour entre la machine source et la machine cible. Afin d'éviter les latences et les retards de livraison des messages, seulement des temps de propagation inférieur à 200 ms sont acceptés.

La fiabilité est le maillon faible des systèmes pair-à-pair. Par nature, il est

difficile de garantir un comportement fiable des pairs. La solution la plus commune à ce type de problème est de répliquer les données. Avec la disponibilité d'un nombre important de pairs, il sera aisé de mettre en œuvre la redondance. Un pair n'appartenant pas au *JobPeerGroup* appartiendra au *RedundancyPeerGroup* s'il vérifie les conditions suivantes :

- Une large bande passante.
- Un espace disque libre.
- Une durée de fonctionnement et une disponibilité élevées.

Ces caractéristiques sont importantes afin d'assurer un système de redondance pratique et efficace.

Pour avoir de meilleurs temps de réponse entre les pairs, on a donné une importance particulière à la bande passante et au temps de propagation en boucle ($RTT$). La durée de fonctionnement et la disponibilité sont des facteurs important pour la délivrance de calculs à effectuer aux pairs. Il faut éviter d'utiliser des pairs très volatils pour ne pas perdre de données en cours de traitement. Par exemple, supposons qu'on délègue un calcul à des pairs appartenant à une société et que ces calculs vont durer une demi-heure. Si on sait que cette société ferme ses portes à une heure précise tout en éteignant tout son parc informatique, il serait judicieux de ne pas utiliser ces pairs et par là, éviter le risque de perdre les calculs en cours. Dans un second cas, on peut supposer le cas de figure où les machines restent fonctionnelles. Avec cette possibilité, il serait intéressant d'utiliser les pairs de la société vu que leurs ressources sont inutilisées. De même, il serait intéressant d'utiliser les pairs pendant les horaires des repas ou autres moments où les ressources sont inutilisées. Ces horaires dépendent des habitudes qui diffèrent d'un pays à un autre. Pour déceler ces moments, on propose d'enregistrer l'activité de chaque machine dans un fichier journal (voir exemple figure 5.4).

Les données sont enregistrées dans le format XML. On s'intéresse particulièrement à l'inactivité des ressources :

- Unité centrale de calcul : cela permet de bien vérifier que la

puissance du processeur est bien libre et qu'aucune tâche n'est en cours d'exécution.

- Mémoire : cela permet de s'assurer qu'il y a suffisamment de mémoire libre pour utiliser la machine dans de bonnes conditions. Plusieurs applications pourraient être résidante en mémoire ce qui ralentirait et diminuerait les performances de la machine.
- Bande passante : le but est d'analyser le trafic pouvant exister sur le réseau. On peut imaginer une application de téléchargement qui consomme la bande passante tout en ayant une charge faible du processeur et de la mémoire disponible.

```
<Dashboard>
  <Month m="September">
    <Day d="Tuesday">
      <Uptime>08:42#9:00</Uptime>
      <CPU>20</CPU>
      <RAM>27</RAM>
    </Day>
    <Day d="Tuesday">
      <Uptime>09:00#9:52</Uptime>
      <CPU>68</CPU>
      <RAM>87</RAM>
    </Day>
    <Day d="Wednesday">
      <Uptime>08:30#9:00</Uptime>
      <CPU>5</CPU>
      <RAM>32</RAM>
    </Day>
    <Day d="Wednesday">
      <Uptime>09:00#10:00</Uptime>
      <CPU>62</CPU>
      <RAM>76</RAM>
    </Day>
    <Day d="Wednesday">
      <Uptime>14:00#14:55</Uptime>
      <CPU>18</CPU>
      <RAM>40</RAM>
    </Day>
  </Month>
</Dashboard>
```

FIGURE 5.4 – Exemple de fichier journal.

Différentes stratégies peuvent être adoptées quant à la réplication des données. Chaque pair appartenant au *JobPeerGroup* peut être en relation avec

un ou plusieurs pairs appartenant au *RedundancyPeerGroup*. Le choix est laissé au développeur.

### 5.4.3 Fonctionnement

L'utilisateur ayant besoin d'une application distribuée peut solliciter notre plate-forme (figure 5.5). Comme première étape, il doit vérifier que les caractéristiques de son application sont bien conformes à une distribution sur un réseau de pair à pair. Ensuite, à partir de sa machine, qui n'est autre que la machine maître, l'utilisateur définit ses besoins en établissant un ensemble de critères. Une recherche est effectuée afin d'identifier les pairs disponibles. Aussitôt, les deux groupes *JobPeerGroup* et *RedundancyPeerGroup* sont formés. Dans un souci d'augmentation de performance, il est possible de regrouper les pairs selon leurs caractéristiques. Ensuite, nous identifions les pairs se ressemblants. Ceci sera utile en cas de panne ou de déconnexion d'un pair. Il est évidemment judicieux de remplacer ce pair par un autre ayant des caractéristiques comparables. Une fois cette tâche réalisée, l'utilisateur dispose d'une proposition de réseau pair-à-pair à utiliser. Il a la possibilité de modifier sa construction manuellement. Une fois la construction du réseau avec ses groupes finalisée, l'utilisateur pourra configurer la décomposition de son application en sous-tâches destinées aux pairs sollicités.

### 5.4.4 Détection des déconnexions

On rappelle qu'une des spécificités des réseaux pairs-à-pair est la volatilité des pairs le formant. Il est important de vérifier à des pas de temps réguliers la présence des pairs sur le réseau. À défaut, le pair sera considéré comme déconnecté. Un pair ayant quitté le réseau délibérément ou suite à une panne sera considéré comme déconnecté.

Pour pouvoir détecter les pannes, différentes méthodes sont présentées dans la littérature. Nous avons choisi d'adopter le protocole *gossip-style* (Renesse *et al.*, 1998) pour sa possibilité de passage à l'échelle sans aucune difficulté. Avec ce

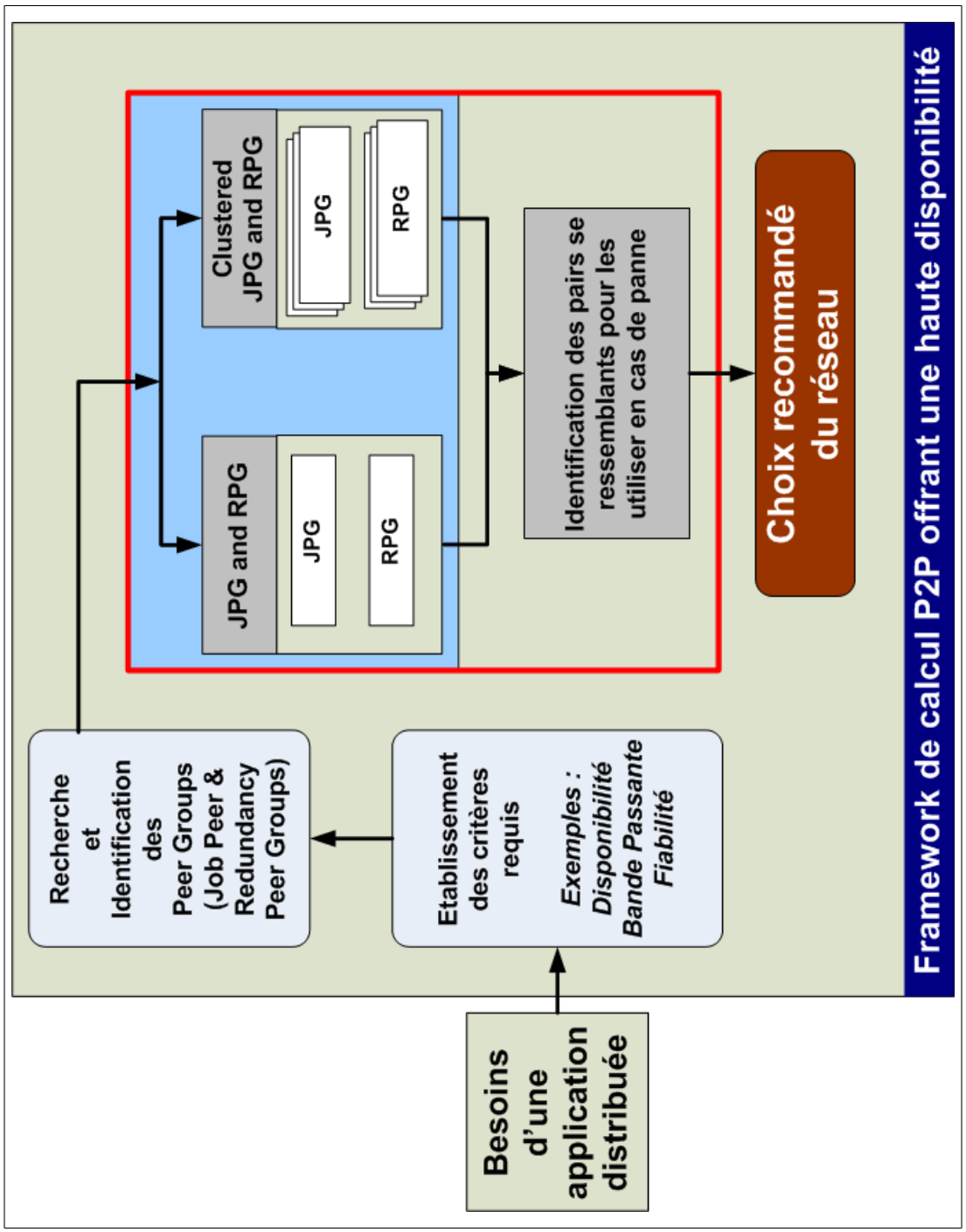



FIGURE 5.5 – La plate-forme PHAC.

modèle, la détection des fautes est effectuée d'une manière distribuée et réside sur chaque pair appartenant au réseau. Le choix de ce protocole est en accord avec la topologie pur adoptée pour notre plate-forme. Chaque pair maintient une table avec pour chaque membre connu son adresse et un entier qui sera utilisé pour détecter les pannes. Cet entier n'est autre qu'un compteur appelé communément *heartbeat counter*. A chaque pas de temps prédéfini, chaque membre incrémente son propre compteur *heartbeat* et sélectionne un autre membre aléatoirement et lui envoie sa liste. A la réception du message contenant la liste, le membre fusionne la liste reçue dans le message avec la sienne et adopte la valeur maximale du compteur *heartbeat*. Chaque membre maintient aussi, dans sa liste pour chacun des autres membres, le dernier temps dont le compteur *heartbeat* a été incrémenté. Si le compteur des messages *heartbeat*, pour une même entrée, n'est pas incrémenté après une certaine période, cette entrée est considérée comme défaillante.

En cas de panne détectée, le pair est aussitôt remplacé par un autre pair lui ressemblant. Les données perdues sont récupérées à partir d'un pair redondant.

### 5.4.5 Sauvegarde et reprise des données

Afin d'assurer la reprise des données en cours de traitement lors de la déconnexion d'un pair, les données sont stockées à pas de temps réguliers prédéfini par l'utilisateur sur des machines appartenant au *RedundancyPeerGroup*. La technique utilisée est la sérialisation et la persistance des objets. Les objets sont tout d'abord encodés sous la forme d'une suite d'octets pour créer une copie conforme des objets d'origine et ensuite sauvegardés sur un ou plusieurs pairs redondants selon la stratégie adoptée par l'utilisateur.

Lors de la reprise des données des pairs redondants, la technique utilisée est la désérialisation. Il s'agit d'un processus de conversion d'un flux d'octets stockés en un objet actif.

## 5.5 Conclusion

Dans ce chapitre, nous avons présenté notre plate-forme *PHAC* pour *A P2P-based Highly Available Computing Framework*. Ce dernier permet d'offrir un réseau de pair à pair hautement disponible et selon les choix de l'utilisateur. Elle permet également de distribuer du calcul sur un réseau construit selon le modèle pur.

Nous avons commencé ce chapitre par proposer un raffinement de la taxonomie des systèmes de l'informatique répartie qui était jusque-là très sommaire. Elle n'offrait pas suffisamment de critères aux développeurs afin d'arrêter leur choix sur la technologie à adopter. Nous avons ensuite présenté notre organisation des pairs au sein de la plate-forme que nous proposons ainsi que les objectifs attendus.

# 6

# Mise en oeuvre et expérimentations

## Sommaire



DANS ce chapitre, nous montrons l'utilité de notre plate-forme. Nous présentons pour cela les résultats obtenus avec deux exemples : l'un déterministe et l'autre non déterministe. Le premier exemple est un calcul itératif de la énième décimale de $\pi$ modélisé selon une approche agent. Le second exemple est la distribution de l'évolution d'une colonie de fourmi dans un environnement distribué.

## 6.1 Introduction

Dans ce chapitre, nous présentons les expérimentations réalisées afin d'évaluer la robustesse et la validité :

- De la plate-forme PHAC,
- De son application dans le cadre d'une simulation multiagent.

Pour cela, nous décrivons deux problèmes :

- Déterministe : calcul de $\pi$.
- Non-déterministe : simulation d'une colonie de fourmis avec une approche multiagent.

Nous étudions le comportement de ces problèmes dans différents contextes de volatilité et d'hétérogénéité.

La première application est présentée dans le but de tester la robustesse de l'infrastructure pair-à-pair et la validité de notre approche.

La deuxième application est un problème non déterministe exécuté sur des nœuds stables afin d'étudier uniquement le comportement de l'application sur différents types d'architectures.

## 6.2 Exemple déterministe

### 6.2.1 Méthodologie

Nous commençons par définir quelques termes qui nous serons utiles ultérieurement :

- *MasterPeer* : c'est le pair maître, c'est-à-dire le pair à partir duquel un utilisateur sollicite le réseau pair-à-pair.
- *RequestPeer* : c'est un pair qui sollicite du calcul. À l'instant $t = 0$, lorsque l'utilisateur sollicite le réseau pair-à-pair, le *MasterPeer* est considéré comme un *RequestPeer*.

- *ExecutantPeer* : c'est un pair qui est libre jusqu'à ce qu'il reçoive une demande d'un *RequestPeer* pour réaliser un calcul. Il redevient libre une fois le calcul terminé.
- *JobPeerGroup* : c'est le groupe formé des pairs qui ont pour rôle d'effectuer les calculs.
- *RedundancyPeerGroup* : c'est le groupe formé des pairs qui ont pour rôle de stocker les calculs au fur et à mesure de leurs évolutions. Son but est de rendre le système plus fiable.

Notons qu'un *ExecutantPeer* trop sollicité par le calcul et qui se rend compte de l'existence de pairs libres dans le *JobPeerGroup* peut les solliciter en devenant un *RequestPeer*.

Notre approche pour distribuer une application sur un réseau pair-à-pair est décrite dans la figure 30. Les étapes ci-dessous concordent par leur numérotation avec celles de la figure. Ces étapes sont :

1. Construire le réseau pair-à-pair avec les deux groupes *JobPeerGroup* et *RedundancyPeerGroup* en utilisant PHAC.

2. Décomposer l'application en sous-tâches.

3. Chaque tâche est envoyée à un *ExecutantPeer* appartenant au *JobPeerGroup*.

4. À chaque pas de temps prédéfini, les résultats sont enregistrés sur des pairs appartenant au *RedundancyPeerGroup*.

5. Une fois la tâche effectuée par un *ExecutantPeer*, le résultat est envoyé au *MasterPeer*.

6. Dès que tous les résultats des tâches distribuées sont récupérés par le *MasterPeer*, le résultat final est affiché à l'utilisateur.

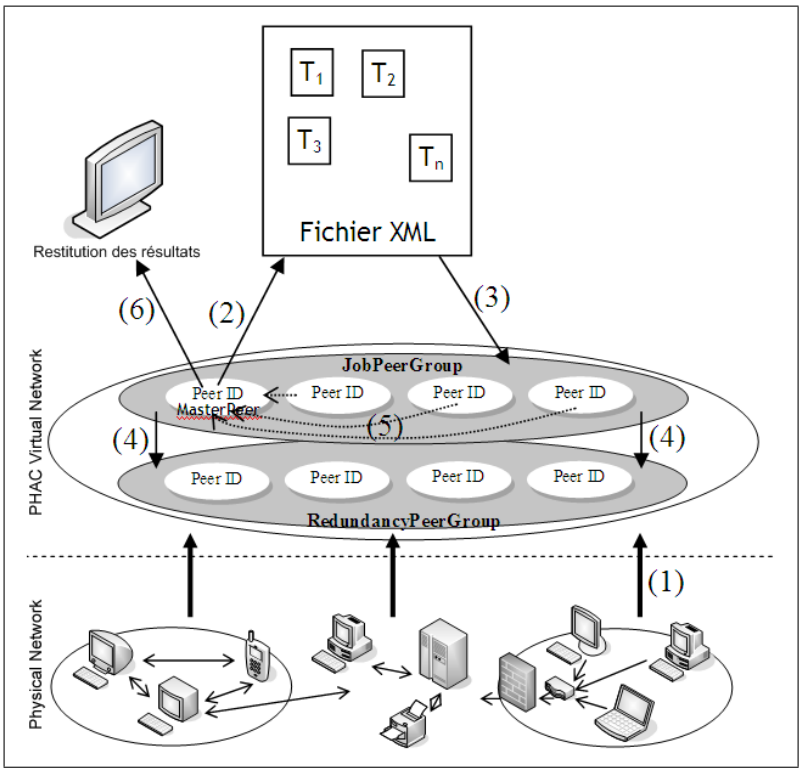



FIGURE 6.1 – Les étapes pour distribuer une application.

### 6.2.2 Cas d'étude au calcul de $\pi$

Dans cette section, nous présentons quelques résultats obtenus en mettant en œuvre notre méthodologie. Nous avons choisi de calculer $\pi$ en utilisant la formule BBP (*Bailey-Borwein-Plouffe*) (Bailey *et al.*, 1997; Gourdon and Sebah, 2003) qui permet de calculer l'énième décimale de $\pi$.

$$\pi = \sum_{k=0}^{\infty} \frac{1}{16^k} \left( \frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right) \tag{6.1}$$

La modélisation a été faite selon une approche agent. Chaque pair est considéré comme un agent qui a la tâche d'effectuer une partie du calcul. L'ensemble des pairs forment un système multiagent.

Pour cela, nous disposons de 10 machines interconnectées par un réseau Fast Ethernet 100 Mbps :

- Huit machines ont un processeur de 2 Ghz et disposent chacune 512 Mo de mémoire vive.
- Deux machines sont équipées d'un processeur de 600 Mhz et de 256Mo de mémoire vive chacune.

La première étape est la construction du réseau. Le *JobPeerGroup* est formé des huit machines à 2Ghz et le *RedundancyPeerGroup* est composé des deux autres machines. Le calcul est décomposé en sous-tâches définies dans un fichier *XML*. Cette décomposition prend en compte un équilibrage de charge effectué manuellement. *PHAC* envoie les sous-tâches aux *ExecutantPeers*. Chaque pair effectue le calcul et, à un pas de temps prédéfini, il enregistre les résultats dans des *RedundancyPeers*. Une fois le calcul terminé, le pair envoie le résultat au pair maître. Dès que les résultats des tâches distribuées sont restituées, le pair maître les combine et affiche le résultat. Ces différentes étapes sont représentées dans le diagramme d'activité de la figure 6.2.

La figure 6.3 représente la précision de calcul de $\pi$ sur l'axe des abscisses. Une précision de 3000 signifie le calcul de la $3000^{\text{ème}}$ décimale de $\pi$. L'axe des ordonnées représente les temps de calcul. Nous avons testé différentes configurations : une seule machine, quatre pairs et huit pairs. Pour chaque précision (500 à 5500) nous avons effectué les calculs sur chacune des configurations.

Nous remarquons que les temps de calculs augmentent exponentiellement en fonction de la précision. Le fait de distribuer les calculs permet des gains de temps presque linéaires par rapport au nombre de pairs. Ceci est dû à l'équilibrage de charge entre les différents pairs.

Dans ce qui suit, on représente l'accélération (*speed-up*), figure 6.4, et l'efficacité, figure 6.5, dans les cas de 4 et 8 pairs. L'accélération est un indice important. Il reflète le degré d'amélioration des temps de calcul dans une approche parallèle par rapport à une approche séquentielle. Notons qu'une accélération inférieure à 1 signifie que le calcul itératif donne de meilleurs résultats que le calcul parallèle.

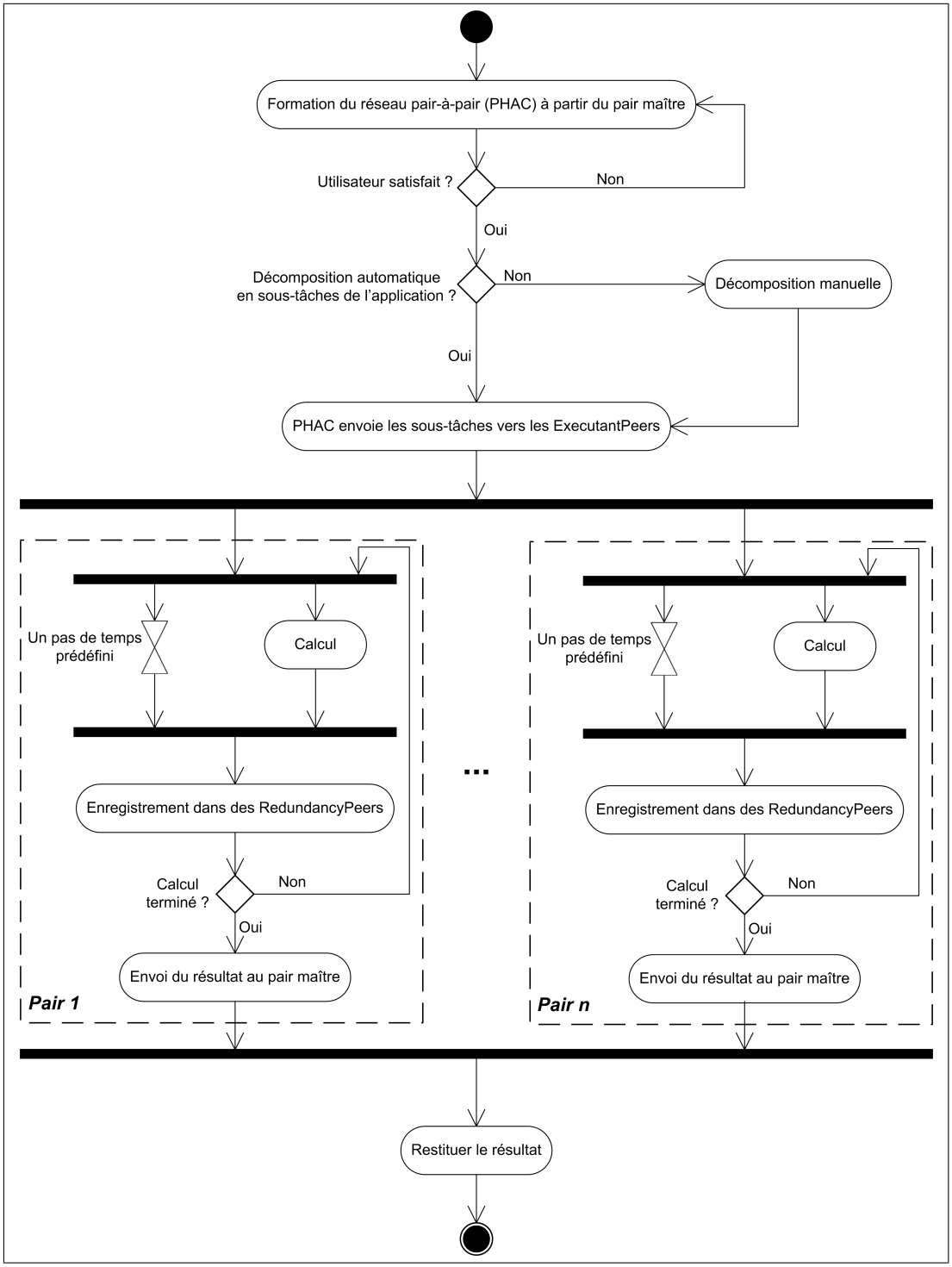



FIGURE 6.2 – Diagramme d'activité modélisant la distribution de l'application calcul de $\pi$.

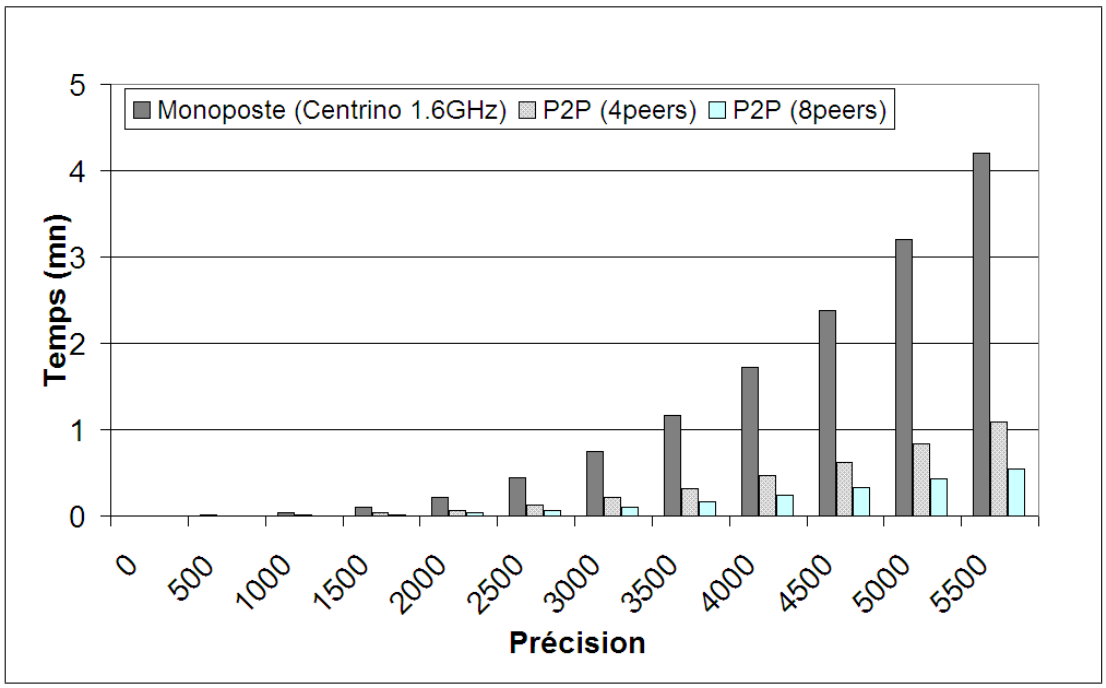



FIGURE 6.3 – Temps de calcul de $\pi$ en mn.

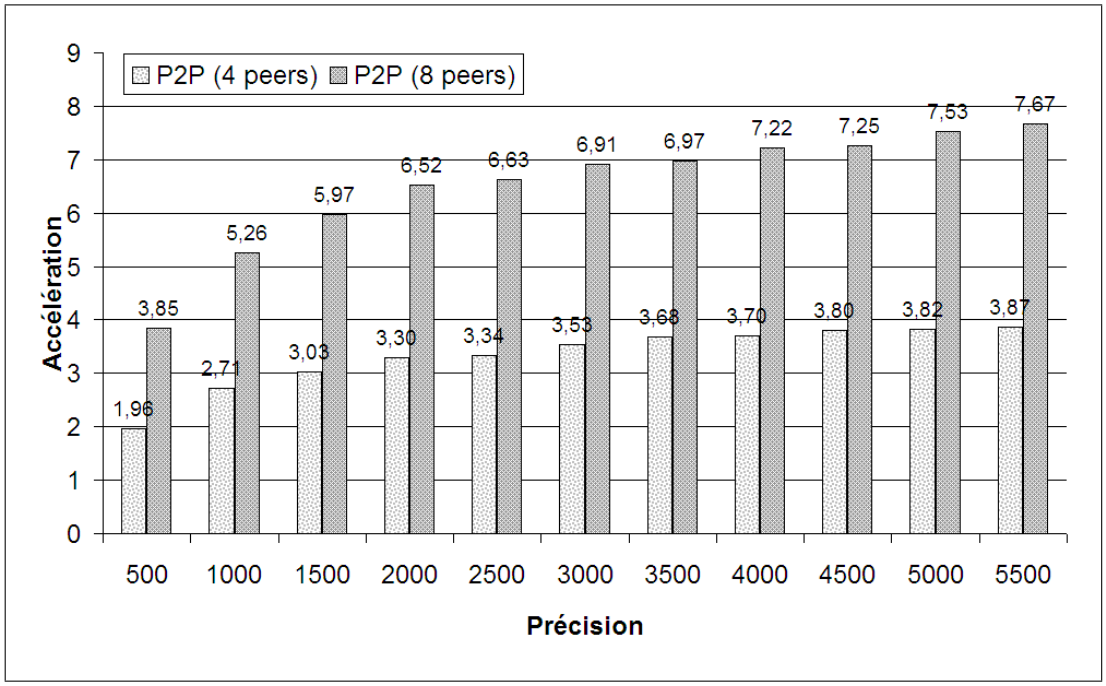



FIGURE 6.4 – L'accélération.

L'efficacité est un indice qui relie l'accélération au nombre de pairs participant au calcul. Dans notre cas, l'accélération est divisée par quatre ou huit suivant le nombre de pairs employés. Ainsi, pour une précision de 5500, les processeurs de quatre pairs sont utilisés à 97%. Ils le sont à 96% dans le cas de 8 pairs.

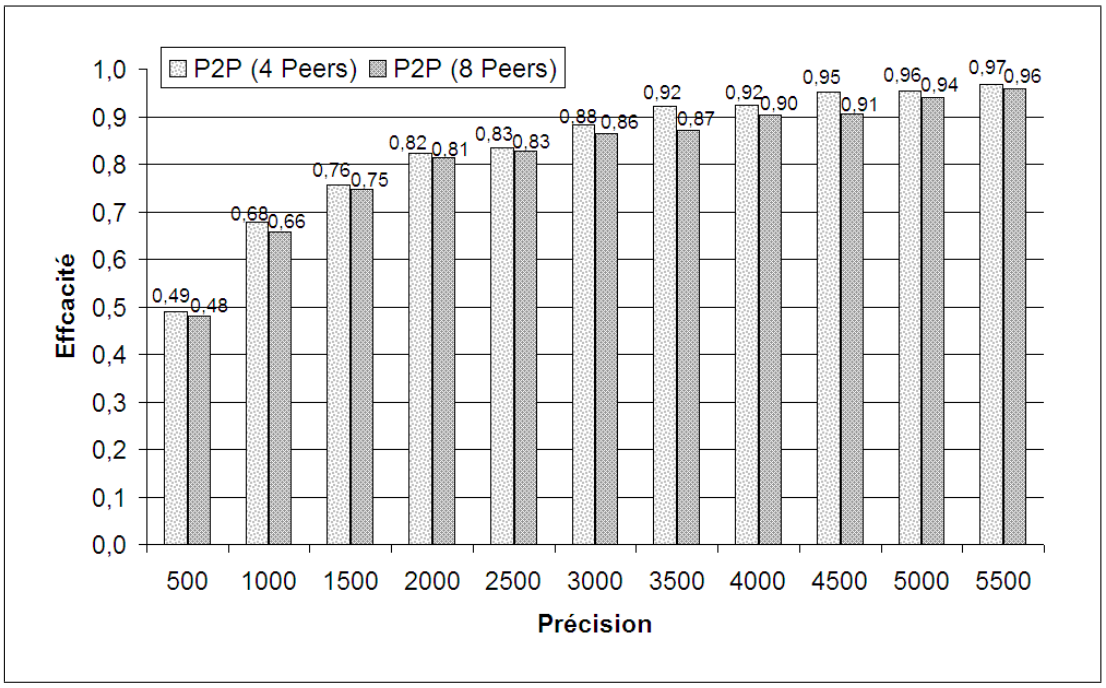



FIGURE 6.5 – L'efficacité.

Les résultats obtenus montrent que les temps de calculs sont meilleurs que dans le cas de calculs en séquentiel.

Afin de terminer le processus du test de notre démarche, nous allons étudier le scénario d'une panne sur des machines participantes aux calculs. Les pairs étant volatiles, il faut considérer la tolérance aux pannes. Nous présentons dans ce qui suit les résultats obtenus pour une précision de 5500 avec huit pairs pour le calcul. Dans la figure 6.6, nous percevons trois courbes : la première (cas n°1) pour le cas sans panne, la seconde (cas n°2) et la troisième (cas n°3) représentent les temps de calcul lors d'une panne respectivement après trois secondes et vingt secondes du démarrage des calculs.

Nous pouvons voir que lorsque la panne a été provoquée après trois secondes,

le temps de calcul a atteint les cent secondes. Alors que dans le cas où la panne s'est produite après vingt secondes, le calcul est fini à 61s. Ceci s'explique par le fait que les machines qui servent de réplication pour les calculs ne sont pas puissantes (Pentium 600Mhz) et qu'en reprenant les calculs depuis leur arrêt, elles mettent beaucoup plus de temps que des machines appartenant au *JobPeerGroup*. Néanmoins, les résultats obtenus sont meilleurs que dans le cas du calcul séquentiel. Ils sont même meilleurs que dans le cas de reprise à nouveau des calculs en parallèles. À noter également qu'en cas de présence de *RequestPeers*, c'est-à-dire de pairs libres appartenant au *JobPeerGroup*, la reprise des calculs aurait été faite par ces derniers en récupérant la réplication de la machine appartenant au *RedundancyPeerGroup*. Les temps de calcul seraient nécessairement meilleurs que ceux qui sont obtenus dans cette expérience où un *RedundancyPeer* est devenu un *ExecutantPeer*.

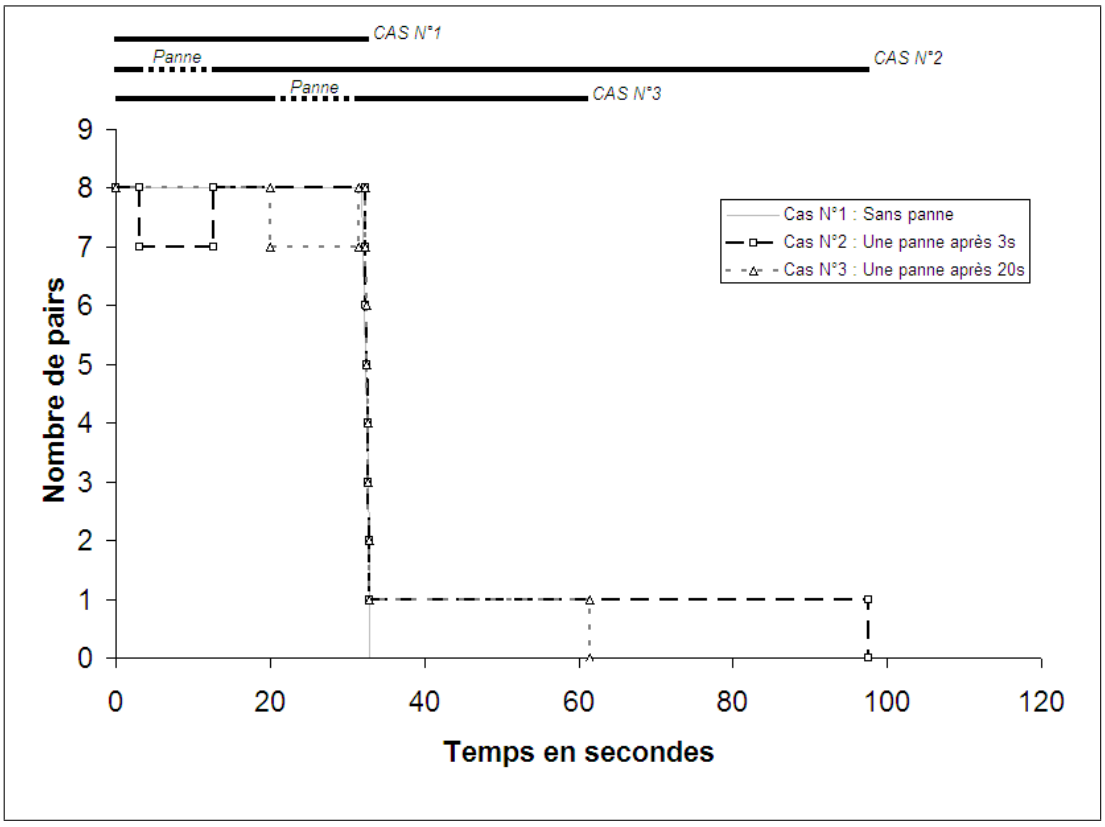
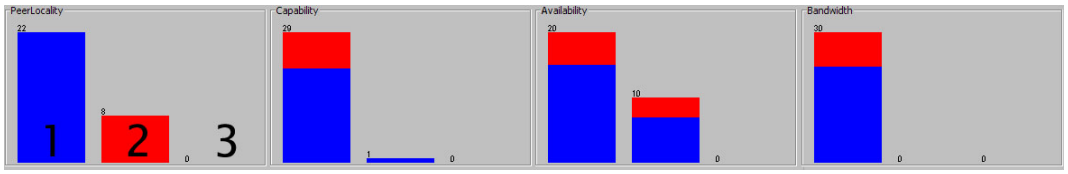
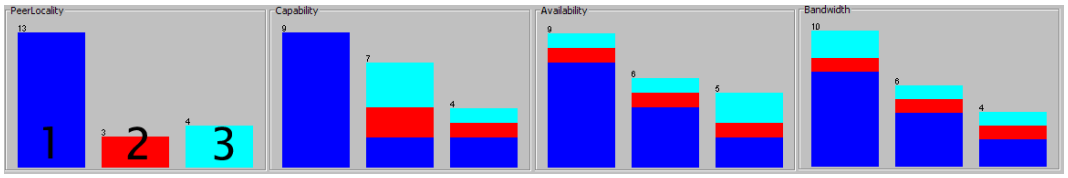



FIGURE 6.6 – Temps de calcul de $\pi$ avec tolérance aux pannes.

Afin de simuler notre approche de classification, beaucoup de scénario sont possibles. Nous en avons choisi un significatif. On dispose de 50 pairs dont 30 appartiennent au *JobPeerGroup* et 20 appartiennent au *RedundancyPeerGroup* disposés selon la configuration présentée dans la figure 6.7. Les pairs sont hétérogènes avec différentes caractéristiques et localisation sur le réseau.




(a) JobPeerGroup




(b) RedundancyPeerGroup

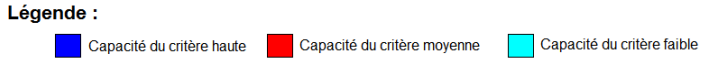



FIGURE 6.7 – Caractéristiques des pairs formant le réseau pair-à-pair.

Nous avons distribué le calcul de $\pi$ pour atteindre une précision de 8000. Nous avons adopté un scénario qui suppose qu'un pair quitte le réseau à des pas de temps réguliers de 10 secondes. Notre objectif est de tester la robustesse du système et spécialement le mécanisme de classification et d'identification des pairs ressemblants au pair défaillant. Les résultats obtenus sont présentés dans la figure 6.8. On peut y voir trois courbes. La première (cas n°1) représente le cas où il n'y a eu aucune panne. La deuxième (cas n°2) représente les temps de calcul dans le cas où il y a des pannes ou des pairs qui ont quitté le réseau délibérément. Dans ce cas, lorsqu'un pair est déclaré défaillant, l'algorithme de classification est utilisé pour identifier les pairs ressemblants. La troisième courbe (cas n°3) représente les temps de calcul comme pour le cas n°2 mais cette fois-ci sans l'utilisation

de l'algorithme de classification c'est-à-dire lorsqu'une défaillance est détectée, le pair défaillant est échangé par un autre appartenant au *RedundancyPeerGroup*. Aucune considération des caractéristiques de ce dernier pair n'est prise en compte.

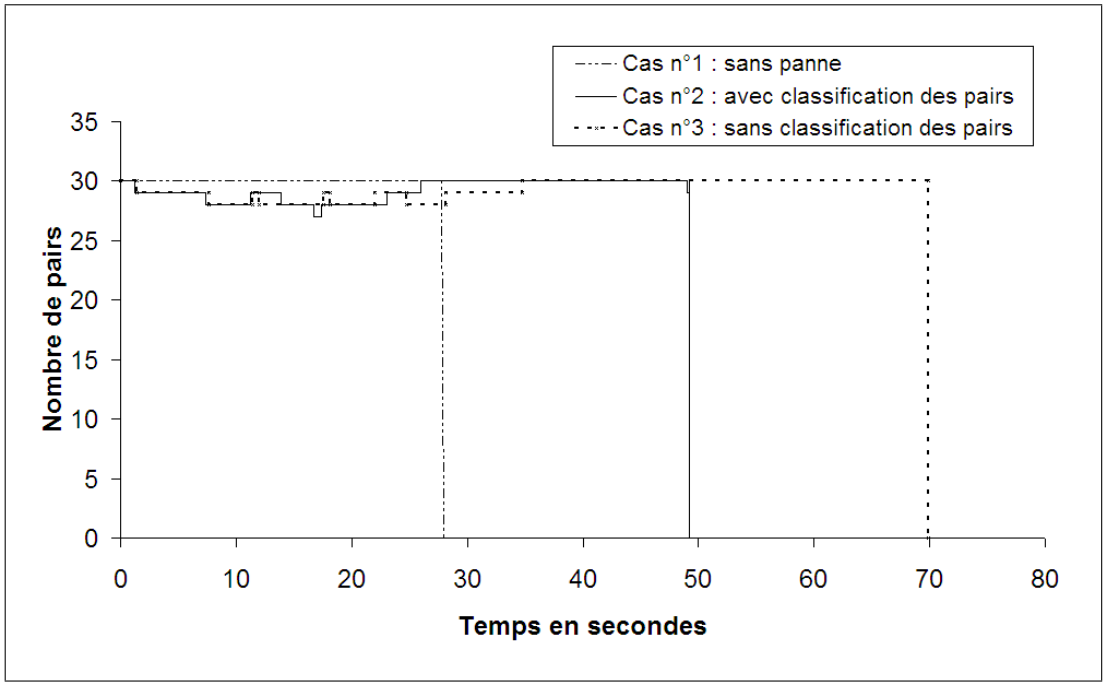



FIGURE 6.8 – Temps de calcul de $\pi$ - Comparaison des cas pairs classifiés et pairs non classifiés.

On peut voir aisément sur les courbes de résultats la dynamique des pairs. Le nombre de pairs diminue de temps à autre vu le scénario adopté. Aussitôt qu'un pair quitte le *JobPeerGroup*, un autre reprend sa place. Le nombre initialement choisi par l'utilisateur (30 pairs) est conservé. Les temps de calculs prouvent que la stratégie de classifier les pairs est bénéfique puisqu'on a de meilleurs temps de calculs.

## 6.3 Exemple non déterministe

### 6.3.1 Architecture

Dans cette section, nous allons présenter notre modèle (Cabani *et al.*, 2005*b*) pour distribuer un système multiagent sur une architecture pair-à-pair. Nous

proposons d'utiliser l'architecture suivante (voir figure 6.9) basée sur *JXTA* et *PHAC*. La gestion des différents pairs connectés au web se fait par la couche noyau de *JXTA*. Le réseau pair-à-pair qui sera utilisé pour distribuer la simulation est offert par notre plate-forme *PHAC*. Au-dessus de celle-ci, un nouveau service est développé. Il aura pour fonction de distribuer la simulation sur différents pairs et de maintenir un état cohérent pour l'ensemble du système. Ce dernier point sera assuré par l'implémentation d'algorithmes de synchronisations (Chandy and Misra, 1981; Jefferson and Sowizral, 1985; Itmi *et al.*, 2004).

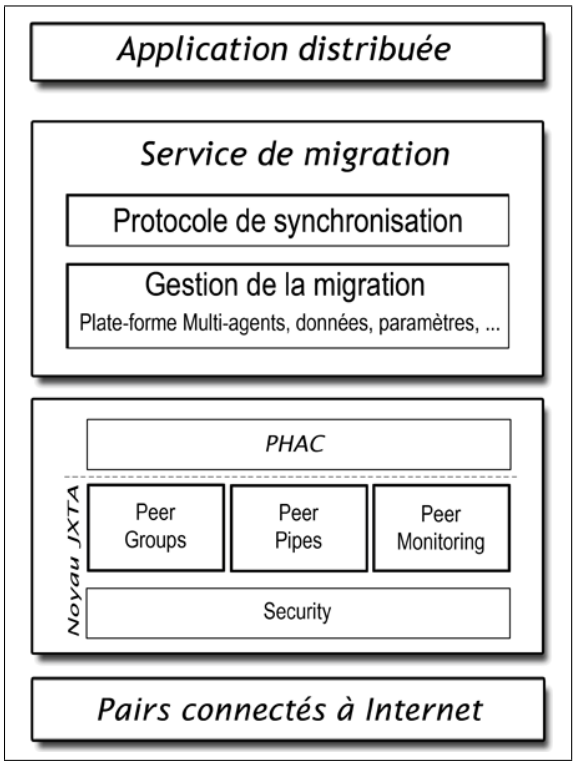



FIGURE 6.9 – Architecture pour distribuer une simulation multiagent.

Une fois le réseaux pair-à-pair construit en utilisant PHAC, la simulation est distribuée suivant une approche environnementale. C'est une vision qui tient compte de l'agent dans son contexte environnemental. Souvent, en simulation événementielle, chaque agent a besoin, pour évoluer, d'une vision locale de son environnement proche (simulation de la circulation de véhicules par exemple). Ce

choix trouve sa justification dans la minimisation des échanges de messages entre pairs. La figure 6.10 illustre un exemple de distribution. Une machine exécutante de la simulation (maître) a délégué une tâche à quatre pairs (A, B, C, D). L'environnement est distribué sur les quatre machines. Comme la machine A est très sollicitée, elle a délégué une partie de sa tâche aux pairs A1, A2 et A3. Pratiquement, le pair maître cherchant à distribuer sa tâche, va se joindre au JobPeerGroup et créer des canaux de communications avec les autres pairs. Il récupère les services proposés par ces derniers, sélectionne ceux qui l'intéressent et se met en relation avec eux. Il leur envoie une tâche à effectuer et attend le résultat (voir figure 6.11).

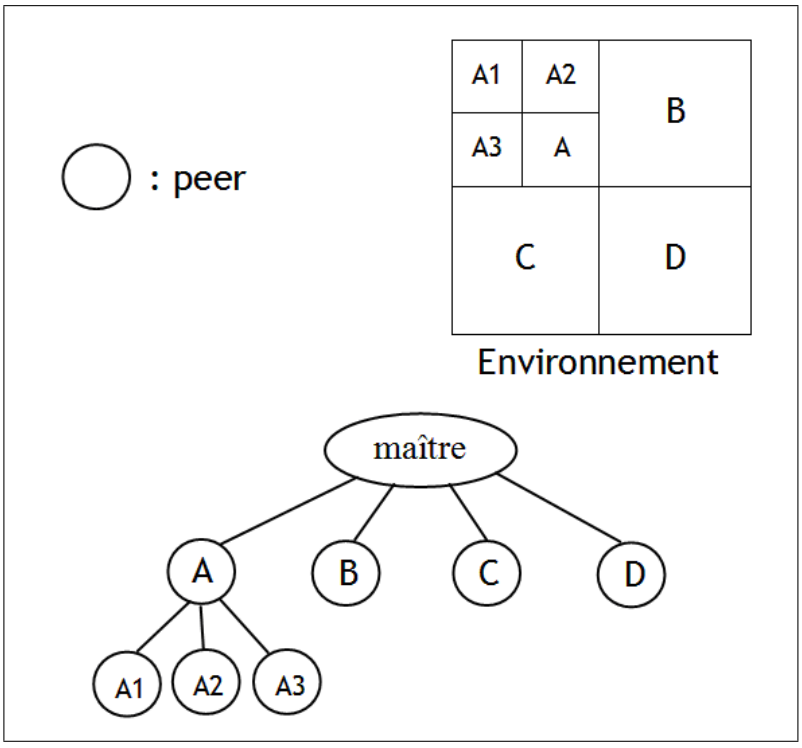



FIGURE 6.10 – Topologie de la distribution.

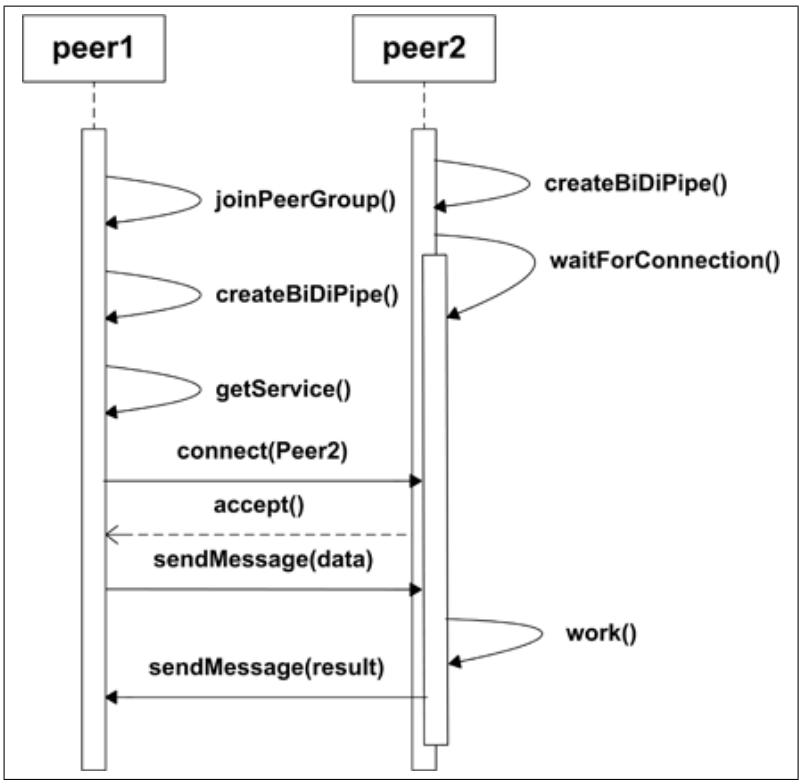



FIGURE 6.11 – Communication entre deux pairs.

Nous pouvons distinguer deux types de pairs qui appartiennent à un *JobPeerGroup* :

- *RequestPeer* : Ce pair sollicite le calcul.
- *ExecutantPeer* : C'est un pair libre jusqu'à ce qu'il reçoive une demande de *RequestPeer* afin d'accomplir un calcul. Il devient libre dès qu'il finit le calcul demandé.

Sur chaque pair, on peut distinguer :

- Un *AgentMap* qui détient la liste de tous les *ExecutantPeer*. Cet agent est sollicité dans chaque migration d'agents d'un pair à un autre.
- Un *AgentMonitor* qui permet d'avoir une surveillance permanente de l'état du pair (connecté, déconnecté, processeur très sollicité...).

Il aide à prendre les décisions appropriées telles que déléguer du calcul à d'autres pairs.

- Un *AgentBroker* qui connaît toutes les caractéristiques des agents dans le système multiagent. Il partage cette connaissance avec d'autres agents en assurant un service de pages jaunes. Cet agent joue le rôle d'intermédiaire. Il permet de rechercher des agents partenaires afin de coopérer pour réaliser une tâche.
- Un *AgentScheduler* qui implémente des algorithmes de synchronisation permettant d'assurer une cohérence générale de la simulation. Il a la responsabilité d'indiquer le commencement et/ou la fin de chaque pas de temps.

### 6.3.2 Cas d'étude : colonie de fourmis

Afin de tester notre modèle et son utilisation dans le cas des simulations multiagents, nous avons implémenté une simulation de colonie de fourmis selon les algorithmes 4,5,6,7,8 et 9 de (Panait and Luke, 2004) et présentés dans ce qui suit. Il s'agit d'utiliser deux types de phéromones. L'un utilisé pour la localisation de la nourriture et l'autre pour la localisation de la fourmilière (ou nid). Les phéromones sont déposées par les fourmis et peuvent s'évaporer et se diffuser. L'algorithme permet la coexistance, sur une même localisation, des deux phéromones. Une fois une fourmi quitte le nid, elle suit le gradient de phéromone emmenant vers la nourriture et pose le second type de phéromones indiquant ainsi le chemin de retour vers le nid. Une fois les fourmis trouvent la nourriture, elles rebroussent chemin vers le nid tout en laissant une traînée de phéromones derrières elles afin de marquer l'emplacement de la nourriture.

**Algorithme 4** : Déplacement d'une fourmi

```
début
    si aDeLaNourriture alors
        retournerVersFourmilière()
    sinon
        rechercheDeLaNourriture()
    fin
fin
```

**Algorithme 5** : Retour vers la foumilière (retournerVersFourmilière())

```
début
    si fourmi trouve la nourriture alors
        orientation ← position avoisinante avec un max de phéromone
    fin
    X ← position prochaine avec un max de phéromone
    si X = NULL alors
        X ← position avoisinante avec un max de phéromone
    sinon
        poserPhéromoneNourriture()
        orientation ← avancement vers X de la position courrante
        bouger vers X
        si fourmi localisée au nid alors
            laisser la nourriture
            aDeLaNourriture ← False
        fin
    fin
fin
```

**Algorithme 6** : Recherche de la nourriture (rechercheDeLaNourriture())

```
début
    si fourmi localisée au nid alors
        orientation ← position avoisinante avec un max de phéromone
    fin
    X ← choixDeLaLocalisation(localisationsProchaine)
    si X = NULL alors
        X ← choixDeLaLocalisation(localisationsAvoisinantes)
    sinon
        poserPhéromoneNid()
        orientation ← avancement vers X de la position courrante
        bouger vers X
        si fourmi localisée à la nourriture alors
            prendre la nourriture
            aDeLaNourriture ← True
        fin
    fin
fin
```

**Algorithme 7** : Choix de la localisation (choixDeLaLocalisation(ensembleDeLocalisations)

```
début
    ensembleDeLocalisations ← ensembleDeLocalisations - obstacles
    ensembleDeLocalisations ← ensembleDeLocalisations -
    localisationsChargées
    si ensembleDeLocalisations = NULL alors
        Retour NULL
    sinon
        sélectionner une localisation de ensembleDeLocalisations où chaque
        localisation est choisie avec une probabilité prédéfinie dans les
        paramètres.
    fin
fin
```

**Algorithme 8** : Dépôt du phéromone nid (poserPhéromoneNid())

```
début
    si fourmi localisée au nid alors
        mettre la phéromone au max
    sinon
        MAX ← la valeur maximale du phéromone nid des localisations
        avoisinantes
        dep ← (MAX/2) - la phéromone nid de la localisation actuelle
        si dep > 0 alors
            déposer dep comme phéromone nid à la position courrante
        fin
    fin
fin
```

**Algorithme 9** : Dépôt du phéromone nourriture (poserPhéromoneNourriture())

```
début
    si fourmi localisée à la nourriture alors
        mettre la phéromone au max
    sinon
        MAX ← la valeur maximale du phéromone nourriture des
        localisations avoisinantes
        dep ← (MAX/2) - la phéromone nourriture de la localisation
        actuelle
        si dep > 0 alors
            déposer dep comme phéromone nourriture à la position
            courrante
        fin
    fin
fin
```

Nous avons évalué et comparé les performances des cas de simulation distribuée et non distribuée (Cabani *et al.*, 2007*a*). Pour cela, nous disposions de 12 machines ayant un processeur 2 GHz et 512Mo de mémoire chacune.

Les machines sont interconnectées par un réseau Fast Ethernet 100 Mbps. L'environnement de la simulation multiagent est une grille de taille 100 x 100. La fourmilière couvre la zone (Xmin, Ymin, Xmax, Ymax) = (75,75,80,80) et la nourriture couvre la zone (25,25,30,30). Afin de montrer l'intérêt de notre approche, nous avons investigué deux scénarios de distribution : simulation sur une architecture Client/Serveur et simulation sur une architecture pair-à-pair.

Dans la figure 6.12, nous avons illustré les temps de calcul en distribuant la simulation sur 2, 4, 8 et 12 ordinateurs.


FIGURE 6.12 – Temps de calcul selon l'architecture pair-à-pair et Client/Serveur.

On peut voir que les temps de calcul sont meilleurs avec l'approche pair-à-pair. Ceci est dû au gain de temps obtenu grâce à la communication directe entre les pairs. Dans le cas de l'architecture Client/Serveur, pour chaque agent qui va migrer d'une machine à une autre, le client doit contacter le serveur afin de récupérer l'adresse de destination. Afin d'illustrer cela, nous présentons dans la figure 6.13 et la figure 6.14 le nombre de messages échangés entre les différentes

machines selon les deux architectures pair-à-pair et Client/Serveur. Il est clair que dans l'architecture Client/Serveur, le nombre de messages échangés avec le serveur explose en fonction du nombre de clients. Le serveur devient rapidement un goulot d'étranglement et le réseau peut s'effondrer facilement. Notre approche permet de mieux gérer ces problèmes avec un meilleur équilibrage de charge du trafic échangé sur le réseau. Il offre de meilleures possibilités pour un passage à l'échelle.

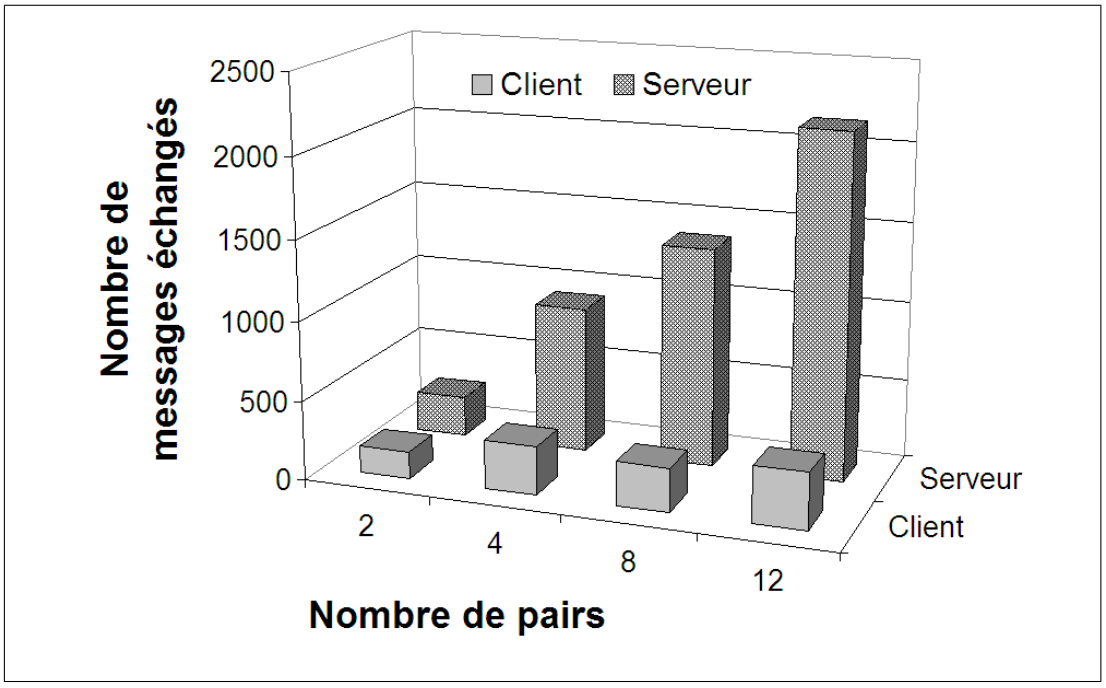



FIGURE 6.13 – Architecture Client/Serveur : nombre de messages échangés.

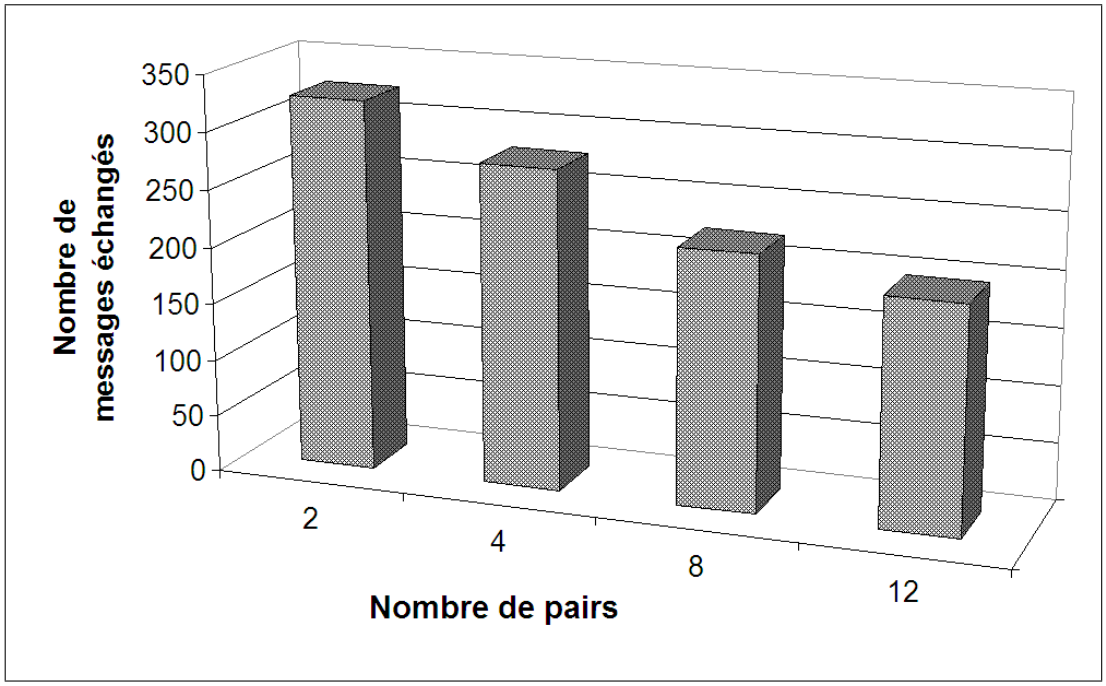



FIGURE 6.14 – Architecture pair-à-pair : nombre de messages échangés.

## 6.4 Conclusion

Dans ce chapitre, nous avons décrit les expérimentations réalisées au cours de nos travaux. Ceci nous a permis d'évaluer notre démarche. Nous avons tout d'abord présenté un problème déterministe qui est le calcul de $\pi$. Notre objectif est de vérifier la robustesse et la validité de notre démarche. Une première expérimentation réelle a été faite. Le cas où les pairs sont volatils a été testé. Nous avons testé notre plate-forme pour distribuer une simulation multiagent. Le cas de colonies de fourmi a fait l'objet d'une implémentation. Dans ce cas, on a considéré que les nœuds sont stables. Une comparaison a été faite avec l'approche Client/Serveur.

Les résultats de tests ont montré que notre plate-forme était bien adaptée et offre une solution pour distribuer des simulations. L'intérêt du pair à pair est justifié dans sa comparaison avec une architecture Client/Serveur classique. Ces

expérimentations ont également montré qu'avec *PHAC* on obtient de meilleurs temps de calculs en prétant une attention particulière à l'hétérogénéité des pairs, à leurs volatilités et au réseau de communication des pairs.

# 7

# Conclusion et perspectives

## Conclusion générale

La simulation distribuée étant un champ actif de la recherche, le but de cette thèse était d'explorer la voie « pair-à-pair » pour mener une simulation distribuée. Différents domaines ont recours à la simulation pour différentes raisons : budgétaire, sécuritaire ou autre. Il faut rappeler que durant ces dernières années, nous observons un intérêt grandissant pour le calcul scientifique d'une façon générale et plus spécialement des besoins en simulation de modèles de plus en plus grands et complexes.

Nous avons vu qu'il y a différentes classes et méthodologies de simulation et nous avons décris les intérêts de distribuer une simulation. Nous avons passé en revue l'importance de la gestion du temps dans la distribution des simulations. Le passage en revue des protocoles de simulation distribuée a montré l'évolution de leur maturité. Ces derniers se sont mûris au fur et à mesure des années. Le dernier en date, *High Level Architecture*, est basé sur une architecture de fédération où il y a un serveur central qui gère toute la simulation. Une telle architecture souffre de la faiblesse d'être centralisée. Le passage à l'échelle n'est pas possible. Les

machines participantes à la simulation doivent être déclarées au préalable. Les ressources à disposition de l'utilisateur restent limitées.

Ce besoin incessant de ressources a fait émerger différentes architectures utilisées par la communauté des scientifiques. Une des architectures les plus connues est la grille de calcul. Même si cette architecture est assez riche et offre des puissances de calcul non négligeables, la mise en œuvre d'applications tournant sous de telles architectures reste assez complexe. L'accessibilité est restreinte aux grandes firmes et laboratoires. L'accès est dans la majorité des cas payant. Les ressources étant partagées par toute la communauté, il faut organiser la réservation de créneaux horaires à l'avance. Le déploiement d'applications sur ce type d'architecture revient jusqu'ici aux chevronnés. La mise en œuvre d'applications distribuées par des physiciens, mathématiciens, biologistes ou autres est assez difficile. Une telle architecture est basée sur des clusters géographiquement distribués. La perte d'un cluster veut dire la perte d'un nombre important de machines, et donc d'énergie et de temps de calculs. Cette architecture repose sur le modèle Client/Serveur. Aucune communication n'est possible entre les nœuds. De ce fait, une telle architecture n'est pas recommandable pour le cas des systèmes multiagents où chaque agent ou entité doit être autonome et capable de communiquer avec les autres directement.

En considérant les différentes méthodologies et architectures existantes pour la simulation distribuée ont été présentées et quelques exemples de la littérature adoptant la distribution selon le protocole HLA ont été cités. Nous avons déduit que cette solution reste centralisée. Elle souffre de plusieurs points de faiblesses. Il nous semble que le modèle pair-à-pair est plus adapté à de telles problématiques.

Le terme pair-à-pair fait référence à un modèle de réseau informatique où tous les nœuds (pairs) jouent à la fois le rôle de serveur et de client. Jusqu'ici, ce paradigme a été très utilisé dans les échanges de contenu et de fichiers. D'autres champs applicatifs commencent à émerger. L'avantage de ce paradigme est sa décentralisation. Il n'y a pas de serveur centralisé. Les pairs peuvent

communiquer directement entre eux ce qui offre de nouvelles possibilités à la simulation distribuée. La spécificité de tels réseaux est que les nœuds sont distribués géographiquement et ont des ressources hétérogènes et asymétriques. Les pairs sont hautement dynamiques puisque les propriétaires des pairs peuvent décider à tout moment de joindre ou de quitter le réseau. Ces nouvelles spécificités intrinsèques au réseau pair-à-pair offrent de nouvelles opportunités. Il faut en tenir compte et plus particulièrement des points de faiblesses qui peuvent paraître.

Cela nous a conduit à présenter une nouvelle taxonomie des systèmes de l'informatique répartie aidant les utilisateurs à arrêter leur choix quant à l'adoption du modèle pair-à-pair pour distribuer leurs applications. Notre plate-forme *PHAC* (*A P2P-based Highly Available Computing Framework*) est un environnement de programmation de haut niveau permettant de prendre en compte l'asymétrie des pairs et offrant à l'utilisateur un réseau adapté autant que possible à ses besoins. Les pairs sont choisis pour être utilisés lors de leurs périodes d'inactivités. Évidement, chaque pair est libre de joindre ou de quitter le réseau à sa guise. Un mécanisme de redondance des pairs est offert afin de ne pas perdre les calculs effectués. Le modèle de *PHAC* est le modèle pur. Aucun serveur n'est présent. Tous les nœuds sont considérés de la même manière. Lors de la phase d'initialisation de l'application, un pair maître va solliciter le réseau pour chercher des pairs disponibles. Il identifie chacun par son *daemon PHAC*. Après cette phase d'initialisation et de formation du réseau, l'utilisateur configure la décomposition de son application sur les différentes machines et lance les calculs. À partir de ce moment, il n'y a aucune distinction entre les différentes machines. Cependant, celle qui a lancé les calculs est celle qui récupérera le résultat.

Nous avons expérimenté la plate-forme sur un premier exemple déterministe et un second non déterministe. Les deux applications ont été modélisées selon une approche agent. Dans le premier exemple, nous avons calculé la énième décimale de $\pi$ en utilisant la formule de *Bailey-Borwein-Plouffe*. Le second exemple est une simulation de colonie de fourmis. Le test de différents scénarios a montré que les résultats expérimentaux obtenus sont encourageants. L'utilisation de notre

plate-forme permet d'obtenir des exécutions performantes sur des infrastructures pair-à-pair.

### Perspectives

Ces travaux ouvrent plusieurs perspectives. Tout d'abord, il serait intéressant d'étendre le champ applicatif à des exemples d'utilités industrielles (simulation numérique, réalité virtuelle...). De plus, il serait nécessaire de réaliser des tests d'exécution en grandeur nature et de tester le passage à l'échelle. Ceci peut être fait en diffusant à grande échelle notre plate-forme ou en essayant de l'interfacer avec des protocoles pair-à-pair habituellement pour l'échange de contenu et de fichiers. Cette voie reste ouverte à de futures études.

L'implémentation actuelle de PHAC ne prend pas en compte les déséquilibres de charge pouvant survenir pendant les calculs. En fait, la détection de tels cas est assez facile. Le plus difficile à quantifier est le prix à payer pour stopper une simulation, la redécouper et la relancer. Cela ouvre sur d'autres sujets de recherche relatifs à l'équilibrage de charge. Une des idées à prévoir est de répliquer les calculs sur différents pairs. A chaque pas de temps prédéfini, on peut imaginer une restructuration des pairs du *JobPeerGroup* basée sur le remplacement de chacun des pairs par le plus performant de ses répliquants. Le *JobPeerGroup* peut devenir dynamique en choisissant dans le temps les pairs offrant les meilleurs résultats.

Un autre point important influant sur les temps des calculs dans les simulations distribuées est la gestion du temps et la préservation de la causalité. Une étude comparative des différentes techniques de synchronisation sur une architecture pair-à-pair serait à envisager dans le futur.

# Bibliographie


Aberer, Karl and Manfred Hauswirth (2001). Peer-to-peer information systems : concepts and models, state-of-the-art, and future systems. In : *Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering*. ACM Press. Vienna, Austria. pp. 326–327.

Abramson, D., R. Sosic, J. Giddy and B. Hall (1995). Nimrod : A tool for performing parameterized simulations using distributed workstations. In : *Fourth IEEE International Symposium on High Performance Distributed Computing (HPDC-4 '95)*. Vol. D. Abramson, R. Sosic, J. Giddy, and B. Hall, "Nimrod : A Tool for Performing Parameterized Simulations Using Distributed Workstations" Proc. 4th IEEE Symp. on High Performance Distributed Computing, .. IEEE Computer Society. p. 112.

Adar, Eytan and Bernardo A. Huberman (2000). Free riding on gnutella. *First Monday*.

Andersen, David, Hari Balakrishnan, Frans Kaashoek and Robert Morris (2001). Resilient overlay networks. *ACM SIGOPS Operating Systems Review* **35**(5), 131–145. 502048 0163-5980 http ://doi.acm.org/10.1145/502059.502048 ACM Press.

Ascape (web). http ://www.brook.edu/es/dynamics/models/ascape/default.htm.

Avouris, Nicholas M. and Les Gasser (2003). *Distributed Artificial Intelligence : Theory and Praxis*.

Baccelli, François and Miguel Canales (1992). Parallel simulation of stochastic petri nets using recurrence equations. In : *Proceedings of the 1992 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*. Newport, Rhode Island, United States. pp. 257–258.

Bailey, David H., Jonathan M. Borwein, Peter B. Borwein and Simon Plouffe (1997). The quest for pi. *MATHINT : The Mathematical Intelligencer*.

Banks, Jerry, John S. Carson, Barry L. Nelson and David M. Nicol (2000). *Discrete-Event System Simulation*. Prentice Hall ; 3 edition.

Barkai, David (2001*a*). *Peer-to-Peer Computing : Technologies for Sharing and Collaborating on the Net*. Intel Press.

Barkai, David (2001*b*). Technologies for sharing and collaborating on the net. In : *First International Conference on Peer-to-Peer Computing (P2P'01)*. pp. 13–28.

Baxter, J. and R. T. Hepplewhite (1996). Broad agents for intelligent battlefield simulation. In : *the 6th Computer Generated Forces and Behavioural Representation*.

Bitcomet (web). http ://www.bitcomet.com/.

Bond, Alan H. and Les Gasser (1998). *Readings in Distributed Artificial Intelligence*.

Bouché, Jean-Paul (1997). La simulation interactive distribuée applications dans le domaine de la défense. In : *Première conférence francophone de MOSIM'97 : Modélisation et simulation des systèmes de production et de logistique* (Hermes, Ed.). Rouen, France. pp. 125–133.

Bousquet, François, Innocent Bakam, Hubert Proton and Christophe Le Page (1998). Cormas : Common-pool resources and multi-agent systems. In : *Proceedings of the 11th International Conference on Industrial and Engineering Applications of Artificial In telligence and Expert Systems* (Springer-Verlag, Ed.). pp. 826–837.

Buyya, Rajkumar (2002). Convergence characteristics for clusters, grids, and p2p networks.

Cabani, Adnane, Mhamed Itmi and Jean Pierre Pécuchet (2005*a*). Improving collaborative jobs in p2p networks. In : *12th IEEE International Conference on Electronics, Circuits and Systems (ICECS'05)*. Gammarth, Tunisia. pp. 135–138.

Cabani, Adnane, Mhamed Itmi and Jean-Pierre Pécuchet (2005*b*). Multi-agent distributed simulation : Discussions and prototyping a p2p architecture. In : *Computer Simulation Conference (SCSC'05)*. Philadelphia, Pennsylvania, United States. pp. 281–286.

Cabani, Adnane, Mhamed Itmi and Jean-Pierre Pécuchet (2007*a*). Distributed multiagent simulation on p2p architecture. In : *The 11-th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications*. pp. 76–79.

Cabani, Adnane, S. Ramaswamy, M. Itmi, S. Al-Shukri and J.P. Pécuchet (2006). Distributed computing systems : P2p versus grid computing alternatives. In : *International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering (IEEE CISSE'06)*. Springer.

Cabani, Adnane, Srini Ramaswamy, Mhamed Itmi and Jean-Pierre Pécuchet (2007*b*). Phac : A p2p-based environment for distributed collaborative applications. *The International Journal of Intelligent Control and Systems, (IJICS)*.

Cabani, Adnane, Srini Ramaswamy, Mhamed Itmi and Jean-Pierre Pécuchet (2007*c*). Phac : an environment for distributed collaborative applications on p2p networks. In : *4th International Conference on Distributed Computing and Internet Technology (ICDCIT'07)*.

Cappello, F. and S. Djilali (2004). Calcul global, desktop grids et xtremweb. In : *Ecole thématique sur la Globalisation des Ressources Informatiques et des Données : Utilisation et Services - GridUse'2004*.

Chaib-draa, B. (1995). Industrial applications of distributed ai. *Commun. ACM* **38**(11), 49–53.

Chandy, K. and R. Sherman (1989). Space-time and simulation. In : *the SCS Multiconference on Distributed Simulation*. pp. 53–57.

Chandy, K. M. and J. Misra (1981). Asynchronous distributed simulation via a sequence of parallel computations.

Chandy, K. M., V. Holmes and J. Misra (1979). Distributed simulation of networks. *Computer Networks* **3**(1), 105–113.

Chavez, A. and P. Maes (1996). Kasbah : An agent marketplace for buying and selling goods.

Chen, G. and B. Szymanski (2002*a*). Lookahead, rollback and lookback, searching for parallelism in discrete event simulation. In : *The Proceedings of the 2002 Summer Computer Simulation Conference*.

Chen, Gilbert and Boleslaw K. Szymanski (2002*b*). Lookback : a new way of exploiting parallelism in discrete event simulation. In : *Proceedings of the sixteenth workshop on Parallel and distributed simulation* (IEEE Computer Society, Ed.). Washington, D.C., USA. pp. 153–162.

Committee, The DIS Steering (1994). The dis vision, a map to the future of distributed simulation. Technical report. University of Central Florida.

Cormas (web*a*). http ://cormas.cirad.fr/.

Cormas (web*b*). http ://cormas.cirad.fr/fr/applica/applica.htm.

Cougaar (web). http ://www.cougaar.org/.

Coulouris, Georges, Jean Dollimore and Tim Kindberg (2006). *Distributed Systems : Concepts and Design*. Addison-Wesley.

Davies, J., T. Manion, R. Rao, J. Miller and X. Zhang. (2006). Introduction to windows peer-to-peer networking. White paper. Microsoft.

Deangelis, D. L., D. K. Cox and C. C. Coutant (1980). Cannibalism and size dispersal in young-of-the-year largemouth bass : Experiment and model. *Ecological Modelling* **8**, 133–148.

Deffuant, Guillaume, Frédéric Amblard, Gérard Weisbuch and Thierry Faure (2002). How can extremism prevail ? a study based on the relative agreement interaction model. *The Journal of Artificial Societies and Social Simulation, JASSS*.

Dingledine, Roger, Michael J. Freedman and David Molnar (2001). The free haven project : distributed anonymous storage service. In : *International workshop on Designing privacy enhancing technologies : design issues in anonymity and unobservability*. Springer-Verlag New York, Inc.. Berkeley, California, United States. pp. 67–95.

Doyen, Guillaume (2005). Supervision des réseaux et services pair à pair. PhD thesis. Université Henri Poincaré - Nancy I.

Drogoul, A., B. Corbara and S. Lalande (1995). Manta : New experimental results on the emergence of (artificial) ant societies. In : *Artificial Societies : The Computer Simulation of Social Life* (N. Gilbert Conte and R., Eds.). pp. 190–211. UCL Press : London.

Druschel, Peter and Antony Rowstron (2001). Past : A large-scale, persistent peer-to-peer storage utility.

Duvvuri, V., P. Shenoy and R. Tewari (2000). Adaptive leases : A strong consistency mechanism for the world wide web. In : *Proc. of the IEEE Infocom'00*. Israel.

emule (web). http ://www.emule-project.net/.

Enembreck, Fabricio and Jean Paul Barthès (2004). Mais - un système multi-agents pour la recherche d'information sur web. *Document Numérique* **8**, 83–106.

Falou, Salah El and François Bourdon (2006). Agent mobile et recherche d'information sur le web : Une solution basée sur le mdp. In : *15ème congrès francophone AFRIF-AFIA Reconnaissance des Formes et Intelligence Artificielle*.

Ferber, Jacques (1999). *Multi-Agent Systems : An Introduction to Distributed Artificial Intelligence*. Addison-Wesley.

Ferscha, A. and S. K. Tripathi (1994). Parallel and distributed simulation of discrete event systems. Technical report. University of Maryland.

Folding (web). http ://folding.stanford.edu/.

Foster, Ian and Adriana Iamnitchi (2003). On death, taxes, and the convergence of peer-to-peer and grid computing. *Lecture Notes in Computer Science* **2735**, 118–128.

FreeHaven (web). http ://www.freehaven.net/.

freenet (web). http ://freenet.sourceforge.net/.

Fujimoto, R. (1990). Parallel discrete event simulation. *Communications of the ACM* **33**(10), 31–53.

Fujimoto, R. (1998). Time management in the high level architecture. *Simulation Special Issue on High Level Architecture* **71**(6), 388–400.

Fujimoto, R. M. (2001). Parallel and distributed simulation systems. In : *Proceedings of the Winter Simulation Conference*. Vol. 1. pp. 147–157.

Fujimoto, Richard and Richard M. Weatherly (1996). Time management in the dod high level architecture. In : *Workshop on Parallel and Distributed Simulation*. pp. 60–67.

Fujimoto, Richard M. (1999). Exploiting temporal uncertainty in parallel and distributed simulations. In : *Proceedings of the thirteenth workshop on Parallel and distributed simulation*. Atlanta, Georgia, United States. pp. 46–53.

Galland, Stephane, Nicolas Gaud and Abder Kouka (2006). Towards a multi-agent model of the decisional subsystem of distributed industrial systems : an organizational and formal approach. *Service Systems and Service Management, 2006 International Conference on* **2**, 859–865.

Gardner, Mathematical Games Martin (1970). The fantastic combinations of john conway's new solitaire game life. *Scientific American* **223**(4), 120–123.

Garg, Vijay K. (2004). *Concurrent and Distributed Computing in Java*. John Wiley & Sons.

Genome (web). http ://genomeathome.stanford.edu/.

Gilbert, Nigel and Klaus G. Troitzsch (2005). *Simulation for the Social Scientist*. 2 ed.. Open University Press.

Gnutella (web). http ://www.gnutella.com/.

Goldspink, Chris (2002). Methodological implications of complex systems approaches to sociality : Simulation as a foundation for knowledge. *Journal Artificial Societies and Social Simulation*.

Gong, Li (2001). Jxta : A network programming environment. *IEEE Internet Computing* **5**(3), 88–95.

Gourdon, Xavier and Pascal Sebah (2003). N-th digit computation. preprint, http ://numbers.computation.free.fr/Constants/Algorithms/nthdigit.pdf.

Gradecki, Joseph D. (2002). *Mastering JXTA : Building Java Peer-to-Peer Applications*. John Wiley & Sons.

Graham, R.L. (2001). Traditional and non-traditional applications. *Peer-to-Peer Networks*.

Greenberg, A., B. Lubachevsky and I. Mitrani (1990). Unboundedly parallel simulation via recurrence relations. In : *ACM SIGMETRICS Conference in Measurement and Modeling of Computer Systems* (Association for Computing Machinery, Ed.).

Grimm, Volker, Tomasz Wyszomirski, David Aikman and Janusz Uchmanski (1999). Individual-based modelling and ecological theory : synthesis of a workshop. *Ecological Modelling* **115**(2-3), 275–282.

Groove (web). http ://office.microsoft.com/en-us/groove/.

Gruer, Pablo, Vincent Hilaire, Jaroslaw Kozlak and Abder Koukam (2003). A multi-agent approach to modeling and simulation of transport on demand problem. In : *Artificial intelligence and security in computing systems* (Kluwer Academic Publishers, Ed.). pp. 119–126.

Guttman, Robert H. and Pattie Maes (1998). *Cooperative Information Agents II Learning, Mobility and Electronic Commerce for Information Discovery on the Internet.*

Halepovic, Emir and Ralph Deters (2005). The jxta performance model and evaluation. *Future Generation Computer Systems* **Volume 21**(Issue 3), 377–390.

Hausheer, David, Nicolas C. Liebau, Andreas Mauthe, Ralf Steinmetz and Burkhard Stiller (2003). Token-based accounting and distributed pricing to introduce market mechanisms in a peer-to-peer file sharing scenario.

ICQ (web). http ://www.icq.com/.

IEEE (1996). *HLA Time Management Design Document V1.0.*

IEEE-Std.1516.1 (2000). Ieee standard for modeling and simulation (m&s) high level architecture (hla) - framework and rules.

IEEE-Std.1516.2 (2000*a*). *IEEE standard for modeling and simulation (M&S) high level architecture (HLA)-object model template (OMT) specification.* IEEE Std 1516.2-2000.

IEEE-Std.1516.2 (2000*b*). Ieee standard for modeling and simulation (m&s) high level architecture (hla)-object model template (omt) specification.

IEEE-Std.1516.3 (2003). *Recommended Practice for High Level Architecture Federation Development and Execution Process (FEDEP).* IEEE Std 1516.3-2003.

Itmi, Mhamed, Adnane Cabani and Jean-Pierre Pécuchet (2004). Simulation distribuée et gestion du temps. In : *5ème conférence francophone de MOdélisation et SIMulation, MOSIM'04*. Vol. 2. Nante, France. pp. 1013–1019.

Jabber (web). http ://www.jabberfr.org/.

Jefferson, D. and H. Sowizral (1985). Fast concurrent simulation using the time warp mechanism. In : *Proceedings of the SCS Distributed Simulation Conference*. pp. 63–69.

Jennings, N. R., K. Sycara and M. Wooldridge" (1998). A roadmap of agent research and development. *Journal of Autonomous Agents and Multi-Agent Systems* **1**(1), 7–38.

JXTA-v2.3.x (2005). Jxta v2.3.x : Java programmer's guide.

Kazaa (web). http ://www.kazaa.com/.

Kitano, Hiroaki, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda and Eiichi Osawa (1997). Robocup : The robot world cup initiative. In : *Proceedings of the first international conference on Autonomous agents*. ACM Press. pp. 340–347.

Kratkiewicz, Gary, Amelia Fedyk and Daniel Cerys (2004). Integrating a distributed agent-based simulation into an hla federation. In : *Simulation Interoperability Workshop (SIW) Logistics and Enterprise Models Forum*.

Kubiatowicz, John, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells and Ben Zhao (2000). Oceanstore : An architecture for global-scale persistent storage.

Kuhl, Fred, Judith Dahmann and Richard Weatherly (1999). *Creating Computer Simulation Systems : An Introduction to the High Level Architecture*.

Lamport, Leslie (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM* **27**(7), 558–565.

Lan, J., X. Liu, P. Shenoy and K. Ramamritham (2002). Consistency maintenance in peer-to-peer file sharing networks. In : *Proc. of 3rd IEEE Workshop on Internet Apps*.

Lawson, Barry G. and Steve Park (2000). Asynchronous time evolution in an artificial society mode. *Artificial Societies and Social Simulation*.

Lees, Michael, Brian Logan and Georgios Theodoropoulos (2007). Distributed simulation of agent-based systems with hla. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* **17**(3), 25.

Lees, Michael, Brian Logan, Ton Oguara and Georgios K. Theodoropoulos (2004). Hla agent : Distributed simulation of agent-based systems with hla. In : *International Conference on Computational Science*. pp. 907–915.

Lees, Michael, Brian Logan, Ton Oguara and Georgios Theodoropoulos (2003). Simulating agent-based systems with hla : The case of sim agent - part ii. In : *Proceedings of the 2003 European Simulation Interoperability Workshop*.

Lefèvre, Valéry, Yann Pollet, Sylvie Philipp and Sylvie Brunessaux (1996). Un systeme multi-agents pour la fusion de données en analyse d'images. *Revue Traitement du Signal* **13**, 100–111.

LimeWire (web).

Logan, Brian and Georgios Theodoropoulos (2001). The distributed simulation of multi-agent systems. *Agents in Modeling and Simulation Special Issue of the Proceedings of the IEEE* **89**(2), 174–186.

Luke, Sean, Claudio Cioffi-Revilla, Liviu Panait and Keith Sullivan (2004). Mason : A new multi-agent simulation toolkit. In : *Proceedings of the 2004 SwarmFest Workshop*.

Mason (web). http ://cs.gmu.edu/ eclab/projects/mason/.

Mazouzi, Smaine, Zahia Guessoum, Fabien Michel and Mohamed Batouche (2007). *A Multi-agent Approach for Range Image Segmentation with Bayesian Edge Regularization*. Chap. Advanced Concepts for Intelligent Vision Systems, pp. 449–460. Lecture Notes in Computer Science.

Metcalfe, R. (1995). Metcalfe's law : A network becomes more valuable as it reaches more users. *Infoworld*.

Michel, Fabien, Pierre Bommel and Jacques Ferber (2002). Simulation distribuée interactive sous madkit. In : *10ème Journées Francophones pour l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents, JFIADSMA'02*. pp. 175–178.

Milojicic, Dejan S., Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins and Zhichen Xu (2002). Peer-to-peer computing. External HPL-2002-57. HP.

Misra, J. (1986). Distributed discrete-event simulation. *ACM Computing Surveys* **18**(1), 39–65.

Mobidyc (web). http ://www.avignon.inra.fr/mobidyc/.

Moduleco (web). http ://www-eco.enst-bretagne.fr/ phan/moduleco/.

Morse, Katherine L. (1996). Interest management in large scale distributed simulations. Technical report. Department of Information and Computer Science, University of California, Irvine.

Mouaddib, Abdel-illah (2004). Co-operative scheduling for a resource-bounded multiagent planning system. *Journal of Experimental & Theoretical Artificial Intelligence* **16**(2), 57–71.

MSN (web). http ://get.live.com/messenger/.

Napster (web). http ://www.napster.com/.

Nodine, Mariam, WilliamBohrer and Anne Hee Hiong Ngu (1999). Semantic brokering over dynamic heterogeneous data sources in infosleuth. In : *International Conference on Data Engineering, ICDE'99*. pp. 358–365.

OceanStore (web). http ://oceanstore.sourceforge.net/.

Olson, Lance (2001). .net p2p : Writing peer-to-peer networked apps with the microsoft .net framework. *MSDN Magazine*.

Orcutt, Guy H. (1957). A new type of socio-economic system. *The Review of Economics and Statistics* **39**(2), 116–123.

P2PWG (2001). Bidirectional peer-to-peer communication with interposing firewalls and nats.

Page, Ernest H. (1994). Simulation modeling methodology : Principles and etiology of decision support.

Panait, Liviu and Sean Luke (2004). Ant foraging revisited. In : *the Ninth International Conference on the Simulation and Synthesis of Living Systems*.

Pandurangan, Gopal, Prabhakar Raghavan and Eli Upfal (2003). Building low-diameter p2p networks. *IEEE Journal on Selected Areas in Communications* **21**(6), 995–1002.

Parker, Miles (2001). What is ascape and why should you care ?. *The Journal of Artificial Societies and Social Simulation, JASSS*.

Perich, Filip (2004). On Peer-to-Peer Data Management in Pervasive Computing Environments. PhD thesis. UMBC.

Phan, Denis (2002). Régimes et changements structurels. Essais d'économie historique. PhD thesis. Université d'Orléans.

Prietula, Michael J., Kathleen M. Carley and Les Gasser (1998). *Simulating organizations : computational models of institutions and groups*. MIT Press.

Pruski, A., Y. Morere and M. Ennaji (2003*a*). Le fauteuil intelligent vahm-3 : Architecture, commande et premiers résultats. *Revue JESA*.

Pruski, A., Y. Morere and M. Ennaji (2003*b*). A multiagent control structure for an intelligent wheelchair. *Journal AMSE Modelling, Measurement, Control*.

Queau, Philippe (1986). *Eloge de la simulation, de la vie des langages à la synthèse des images*.

Rao, A. S. and M. P. Georgeff (1995). Bdi agents : from theory to practice.

Ratnasamy, Sylvia, Paul Francis, Mark Handley, Richard Karp and Scott Shenker (2001). A scalable content-adressable network. In : *ACM SIGCOMM 2001*. San Diego, California, United States. pp. 161–172.

Reed, David P. (1999). That sneaky exponential beyond metcalfe's law to the power of community building. http ://www.reed.com/Papers/GFN/reedslaw.html.

Renesse, R. V., Y. Minsky and M. Hayden (1998). A gossip-style failure detection service. Technical report.

RePast (web). http ://repast.sourceforge.net/.

RoboCup (web). http ://www.robocup.org/.

Rowstron, Antony and Peter Druschel (2001). Pastry : Scalable, distributed object location and routing for large scale peer-to-peer systems. *Lecture Notes in Computer Science* pp. 329–351.

Rus, Daniela, Clifford Stein and Rong Xie (2001). Scheduling multi-task multi-agent systems. In : *Proceedings of the Fifth International Conference on Autonomous Agents*. ACM Press. pp. 159–160. Poster abstract.

Said, Lamjed Ben, Thierry Bouron and Alexis Drogoul (2002). Agent-based interaction analysis of consumer behavior. In : *Proceedings of the first international joint conference on Autonomous agents and multiagent systems : part 1* (ACM Press, Ed.). Bologna, Italy. pp. 184–190.

Saroiu, Stefan, P. Krishna Gummadi and Steven D. Gribble (2002). A measurement study of peer-to-peer file sharing systems.

Schoder, Detlef and Kai Fischbach (2003). Peer-to-peer prospects. *Commun. ACM* **46**(2), 27–29.

SearchLing (2000). Peer-to-peer networking.

Shardanand, Upendra and Pattie Maes (1995). Social information filtering : algorithms for automating word of mouth. In : *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM Press/Addison-Wesley Publishing Co.. Denver, Colorado, United States. pp. 210–217.

Skype (web). http ://www.skype.com/.

Sloman, Aaron (1998). What's an ai toolkit for ?

Srivatsa, Mudhakar, Bugra Gedik and Ling Liu (2006). Large scaling unstructured peer-to-peer networks with heterogeneity-aware topology and routing. *IEEE Trans. Parallel Distrib. Syst.* **17**(11), 1277–1293. Student Member-Mudhakar Srivatsa and Member-Bugra Gedik and Senior Member-Ling Liu.

Stoica, Ion, Robert Morris, David Karger, Frans Kaashoek and Hari Balakrishnan (2001). Chord : A scalable peer-to-peer lookup service for internet applications. In : *ACM SIGCOMM 2001*. ACM Press. San Diego, California, United States. pp. 149–160.

Sunaga, Hiroshi, Toshiyuki Oka, Kiyoshi Ueda and Hiroaki Matsumura (2005). P2p-based grid architecture for homology searching. In : *Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing (P2P'05) - Volume 00*. IEEE Computer Society. pp. 148–149.

Tedeschi, C. (2003). Prise en main de jxta. Technical report.

Vanbergue, Diane and Alexis Drogoul (2002). Approche multi-agent pour la simulation urbaine.

Vanbergue, Diane, Jean-Pierre Treuil and Alexis Drogoul (2000). Modelling urban phenomena with cellular automata. *Advances in Complex Systems* **3**(1-4), 127–140.

Vercouter, Laurent (2001). Une gestion distribuée de l'ouverture dans un système multi-agent. In : *Actes des Journées Francophones IAD et SMA*. Montréal, Canada. pp. 177–188.

Vincent, R., B. Horling, T.Wagner and V. Lesser (1998). Survivability simulator for multi-agent adaptive coordination. In : *the International Conference on Web-Based Modeling and Simulation 1998 (WMC'98)*.

Waldman, Marc, Aviel D. Rubin and Lorrie Faith Cranor (2000). Publius : A robust, tamper-evident, censorship-resistant, web publishing system. In : *Proc. 9th USENIX Security Symposium*. pp. 59–72.

Weiss, Gerhard (1999). *Multiagent Systems : a Modern Approach to Distributed Artificial Intelligence*. MIT Press.

WEISS, Gerhard (2000). *Multiagent systems : a modern approach to distributed artificial intelligence*.

Wilson, Brendon J. (2002). *JXTA*. New Riders Publishing.

Wooldridge, Michael and Nicholas R. Jennings (1995). Intelligent agents : Theory and practice. *Knowledge Engineering Review*.

Yin, J., L. Alvisi, M. Dahlin and C. Lin (1999). Hierarchical cache consistency in a wan. In : *Proc. of the USENIX Symp. on Internet Technologies*. Boulder, CO.

YM ! (web). http ://messenger.yahoo.com/.

Zeigler, Bernard (1976). *Theory of modeling and simulation.* John Wiley and Sons.

Zhao, B. Y., J. D. Kubiatowicz and A. D. Joseph (2001). Tapestry : An infrastructure for fault-tolerant wide-area location and routing. Tech. report ucb/csd-01-1141. UC Berkeley.