# Placement et stockage de l'information bio-inspiré : une approche orientée agent mobile

Hugo Pommier

▶ **To cite this version:**

Hugo Pommier. Placement et stockage de l'information bio-inspiré : une approche orientée agent mobile. Système multi-agents [cs.MA]. université de caen, 2010. Français. NNT : . tel-01076407

**HAL Id: tel-01076407**
**https://hal.science/tel-01076407**

Submitted on 22 Oct 2014

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.


**UNIVERSITÉ de CAEN/BASSE-NORMANDIE**

**U.F.R.** : Sciences

**ÉCOLE DOCTORALE** : SIMEM

**THÈSE**

présentée par

Hugo POMMIER

et soutenue

**le** 19 octobre 2010

en vue de l'obtention du

**DOCTORAT de l'UNIVERSITÉ de CAEN**

**spécialité : Informatique**

***(Arrêté du 7 août 2006)***

# Placement et stockage de l'information bio-inspiré : une approche orientée agent mobile

**MEMBRES du JURY**

| | | | |
|---|---|---|---|
| Mr Yves DEMAZEAU | Directeur de Recherche | LIG, Grenoble | (rapporteur) |
| Mr Cyrille BERTELLE | Professeur des Universités | Université du Havre | (rapporteur) |
| Mr Jean-Paul ARCANGELI | Maître de Conférence | Université de Toulouse | (examinateur) |
| Mr Laurent VERCOUTER | Assistant Professeur | Ecole des Mines de Saint-Etienne | (examinateur) |
| Mr Abdel-Illah MOUADDIB | Professeur des Universités | Université de Caen | (examinateur) |
| Mr François BOURDON | Professeur des Universités | Université de Caen | (directeur) |

Mis en page avec la classe thloria.

– – –

# Table des matières

# Table des figures

# Liste des tableaux

# Introduction

# 1 Contexte

Le rapport de l'homme à l'information a considérablement évolué en très peu de temps. Sans rentrer dans les prémices de l'informatique, durant les trente dernières années les médias de stockage sont passés de la disquette d'une capacité de 360 kilo-octets au disque Blu-Ray d'une capacité de 25 giga-octets soit environ 20000 disquettes. Parallèlement à l'augmentation des capacités de stockage a eu lieu le déploiement des réseaux informatiques, avec en point d'orgue, l'explosion de l'Internet. En observant le site [IWS, 2010] nous constatons que le nombre d'utilisateurs d'Internet a augmenté d'environ 400% au cours de la période 2000-2009, pour atteindre plus d'1,8 milliards d'utilisateurs. Cette croissance exceptionnelle s'est accompagnée d'améliorations techniques majeures notamment dans le débit des connexions réseaux. Nous sommes passés de modems possédant un débit de 33,6 et 56 kilo-bits par seconde, à l'Asymmetric Digital Subscriber Line ou ADSL possédant un débit pouvant atteindre plusieurs dizaines de méga-bits par secondes.

Ces évolutions ont modifié notre vision de l'information. Il y a de moins en moins de supports physiques, et toute information est accessible au travers d'Internet. L'étude d'IDC [IDC, 2009] à ce sujet est éloquente. En 2008, 487 milliards de giga-octets ont été créés et transmis par Internet. C'est l'équivalent de 19 milliards de disques Blu-Rays. IDC estime qu'en 2012, l'humanité sera responsable de la création de cinq fois plus d'information numérique.

C'est dans ce contexte de forte croissance et de dématérialisation des données que nous nous intéressons à la problématique de la gestion de l'information dans les réseaux. Les applications créées sur le modèle historique client/serveur, ne sont clairement plus adaptées aux volumes et aux contenus des données actuelles. Elles constituent des goulots d'étranglement évidents liés au nombre de connexions et au débit limité qu'elles proposent. En comparaison, les capacités de stockage et de calcul des machines connectées à l'Internet forment une quantité de ressources gigantesque. C'est l'utilisation de ces ressources disponibles qui est à la base des réseaux **pair-à-pair**. Les nœuds physiques qui auparavant ne faisaient qu'utiliser le service, sont désormais utilisés pour fournir le service.

Les systèmes pair-à-pair sont aujourd'hui largement répandus et leur utilisation va du partage de temps CPU [Seti@home, 1997] au partage de données [Napster, 1999, Clarke *et al.*, 2000, eMule, 2002], en passant par les communications [Baset et Schulzrinne, 2004] et le stockage de données [Rhea *et al.*, 2003]. Dans tous les cas une garantie d'authenticité et de sécurité des données est nécessaire. Ce besoin de robustesse se traduit par une tolérance aux fautes

accidentelles et intentionnelles. Les services proposés sur des réseaux dynamiques de type pair-à-pair doivent pouvoir résister à une défaillance technique ou humaine. Si une machine hébergeant des données tombe en panne ou si un utilisateur efface involontairement un fichier, l'existence de l'information ne doit pas être remise en cause. Le contenu d'une donnée se doit également d'être préservé en cas d'attaques. Une personne malveillante ne doit en aucun cas obtenir ou modifier le moindre renseignement sur la provenance, la destination, ou même le contenu de l'information. Enfin il est nécessaire de garantir la pérennité de l'information en la rendant persistante et utilisable à n'importe quel moment ou point du réseau. La **théorie de l'information** propose des mécanismes de fragmentation et de redondance assurant la disponibilité et la sûreté des données.

L'efficacité de la gestion de l'information dans un réseau repose également sur son placement et sa recherche. Les machines d'un réseau doivent s'accommoder de grandes quantités de données, peu structurées et tout cela dans un contexte de connexions de moins en moins figées voire peu sûres. Ces systèmes d'information exigent une nouvelle approche de la programmation sur laquelle pèsent de nouvelles contraintes liées à l'extrême variété et à la mobilité des équipements. Les concepts **d'agents mobiles et de systèmes multi-agents** présents dans l'intelligence artificielle distribuée apparaissent comme une solution facilitant la mise en œvre d'applications dynamiquement adaptables, et offrent un cadre générique pour le développement d'applications réparties sur des réseaux de grande taille.

Notre travail se situe à la croisée des trois domaines suivants :

- **Les réseaux pair-à-pair** pour les aspects liés à la gestion du réseau et à la décentralisation du service.
- **La théorie de l'information** pour assurer la disponibilité, la pérennité et la sûreté de l'information.
- **Les agents mobiles et les systèmes multi-agents** pour proposer des algorithmes de placements et de recherche de l'information adaptatifs.

## 2 Des nuées d'oiseaux au placement de l'information

Dans cette thèse nous présentons une approche originale consistant à utiliser des règles du vivant pour placer l'information dans un réseau décentralisé. L'information est représentée par une nuée en déplacement semblable à ce que l'on peut observer chez les oiseaux. À partir de cette modélisation nous proposons des méthodes de placement et de recherche de l'information robustes, "intelligentes", et totalement décentralisées.

## 3 Organisation du manuscrit

La première partie de cette thèse propose un état de l'art des différents domaines de recherche abordés. Dans le premier chapitre, nous revenons sur la problématique des réseaux pair-à-pair. Nous présentons les différentes évolutions de ces réseaux en nous intéressant plus parti-

culièrement à la construction et au maintien de leur structure. Le second chapitre aborde le cycle de vie de l'information dans un réseau. Nous introduisons dans un premier temps les mécanismes hérités de la théorie de l'information assurant la disponibilité et la sûreté des données. Dans un second temps, nous dressons un tableau des différentes méthodes issues des réseaux pair-à-pair pour placer l'information. Enfin nous nous intéressons aux techniques mises en œuvre pour rechercher une information dans un environnement décentralisé. Le dernier chapitre de l'état de l'art porte sur les notions d'agent et de système multi-agents. Nous abordons dans ce chapitre les définitions essentielles de cette thématique. Nous insistons sur deux notions au cœur de nos travaux, les systèmes multi-agents bio-inspirés, et les agents mobiles.

La seconde partie de cette thèse présente nos contributions. Le premier chapitre propose une modélisation en couches de la gestion de l'information à base d'agents dans un réseau pair-à-pair totalement décentralisé. Dans ce chapitre nous introduisons la couche réseau utilisée pour fournir un service totalement décentralisé et nous validons son fonctionnement par des résultats issus de simulations. Le second chapitre décrit le comportement bio-inspiré des agents utilisés dans notre approche. Nous détaillons les algorithmes nécessaires pour proposer un placement de l'information dynamique. Nous concluons ce chapitre par des simulations validant le comportement de nos agents et mesurant l'efficacité de nos algorithmes. Nous nous intéressons dans le troisième chapitre à la recherche d'information. Nous présentons deux algorithmes implantés dans des agents mobiles permettant de parcourir aléatoirement le réseau pour trouver une information. Nous comparons les résultats de ces deux approches au travers de différentes simulations. Enfin le quatrième et dernier chapitre de cette partie nous permet de sortir du cadre des simulations et de présenter une implémentation sur un réseau constitué de 100 machines physiques du modèle de l'information proposé dans cette thèse.

Enfin nous concluons à la validité du modèle proposé et nous présentons nos perspectives de recherche.

# Première partie

# État de l'art

# Introduction de la partie I

L'objet de cette première partie est d'introduire les concepts à la base du travail que nous présenterons par la suite. Nous débuterons, dans un premier chapitre, par définir les réseaux pair-à-pair en étudiant leur structure. Après avoir dressé un historique des applications les plus marquantes, nous nous attarderons sur les réseaux pair-à-pair décentralisés structurés en présentant les mécanismes liés à la construction du réseau. Enfin nous aborderons le cas des réseaux auto-organisés en détaillant le fonctionnement des protocoles épidémiques.

Le second chapitre de cette thèse traite du cycle de vie de l'information. Nous présenterons les techniques issues de la théorie de l'information pour assurer la disponibilité et la sûreté d'une donnée. Nous nous intéresserons ensuite à deux méthodes de placement de l'information dans les réseaux pair-à-pair. Nous terminerons ce chapitre par la présentation des approches utilisées dans le cadre de la recherche de fichiers.

Le troisième et dernier chapitre de cette partie est consacré aux agents et systèmes multi-agents. Nous définirons dans un premier temps la notion d'agent. Nous traiterons ensuite la dimension sociale de ces entités et nous discuterons de leur mise en interaction pour présenter les systèmes multi-agents. Nous nous attarderons sur le cas des systèmes multi-agents bio-inspirés dont le comportement est issu de l'étude du vivant. La dernière partie de ce chapitre établit le lien entre ce domaine et les réseaux pair-à-pair. Nous y étudierons plus en détails les agents mobiles et les différentes plates-formes utilisées pour la création d'applications réparties.

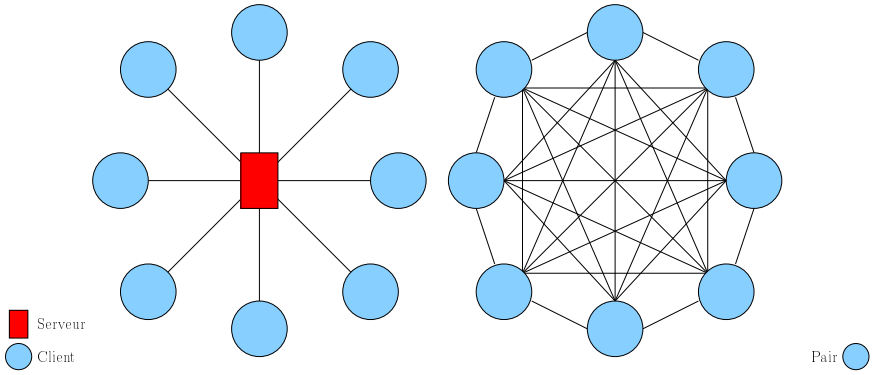
# Chapitre 1

# Structure des réseaux pair-à-pair

**Sommaire**



L'objectif de ce chapitre est de détailler la couche fondamentale d'une application de stockage répartie à savoir la construction et l'organisation du réseau. Dans une optique de décentralisation totale du service, nous nous sommes naturellement intéressés aux réseaux de types pair-à-pair, ou P2P. Ici nous introduisons le monde des réseaux pair-à-pair par le biais des différents modèles que l'on peut rencontrer dans la littérature. Nous débuterons donc ce chapitre par des définitions permettant de mieux comprendre la suite de cette partie, puis nous présenterons les premières applications P2P. Nous nous intéresserons ensuite aux réseaux décentralisés structurés, lieu de nombreuses études. Enfin nous finirons ce chapitre par une section consacrée aux réseaux auto-organisés, ou protocoles épidémiques.




(a) Modèle client/serveur (b) Modèle pair-à-pair

FIGURE 1.1 – Dans le modèle pair-à-pair, chaque nœud est un serveur potentiel

## 1.1 Présentation et définitions

Les réseaux pair-à-pair sont des systèmes d'information permettant à des utilisateurs de partager simplement des objets. Le plus souvent ces objets sont des fichiers, mais cela peut également être des ressources de calcul, des flux multimédia - appelé streaming -, ou des services. Skype [Baset et Schulzrinne, 2004], service de téléphonie par IP, est un bon exemple d'application reposant sur un réseau pair-à-pair. Par opposition au modèle client/serveur, les réseaux pair-à-pair ne nécessitent en théorie pas de point central dans le réseau et mettent directement en relation les utilisateurs entre eux. Les utilisateurs sont plus communément appelés **pairs**. La figure 1.1 montre la différence de structure entre les deux modèles.

Le service fourni dans le cadre d'une application client/serveur est clairement centralisé (figure 1.1(a)). Les faiblesses de ce modèle reposent sur la fiabilité et le passage à l'échelle du service. En effet si le serveur subit une panne ou une attaque, le service peut ne plus être fourni, ou de manière dégradé. Un serveur constitue également un goulot d'étranglement du fait du nombre de connexions limitées.

Dans le modèle pair-à-pair, chaque utilisateur contribue au bon fonctionnement du service. Ainsi chaque nœud est connecté non plus à un même serveur mais à un ensemble de pairs (figure 1.1(b)). Il est donc nécessaire que chaque membre du réseau possède le même niveau de responsabilité. Chaque pair doit alors pouvoir jouer le rôle de :

– serveur, lorsqu'il fournit une ressource ou un service à un autre pair,
– client, lorsqu'il demande une ressource ou un service.

Les premières applications pair-à-pair faisaient apparaître des disparités entre les nœuds. Ces applications dites de première génération, appartiennent aux modèles centralisés, semi-centralisés et hybrides. La seconde génération d'applications appartenant au modèle décentralisé non-structuré, vise à faire disparaître ces disparités.

La figure 1.1(b) montre un ensemble de pairs interconnectés. Cette organisation est virtuelle et construite sur le réseau physique, chaque lien représentant un canal de communication entre

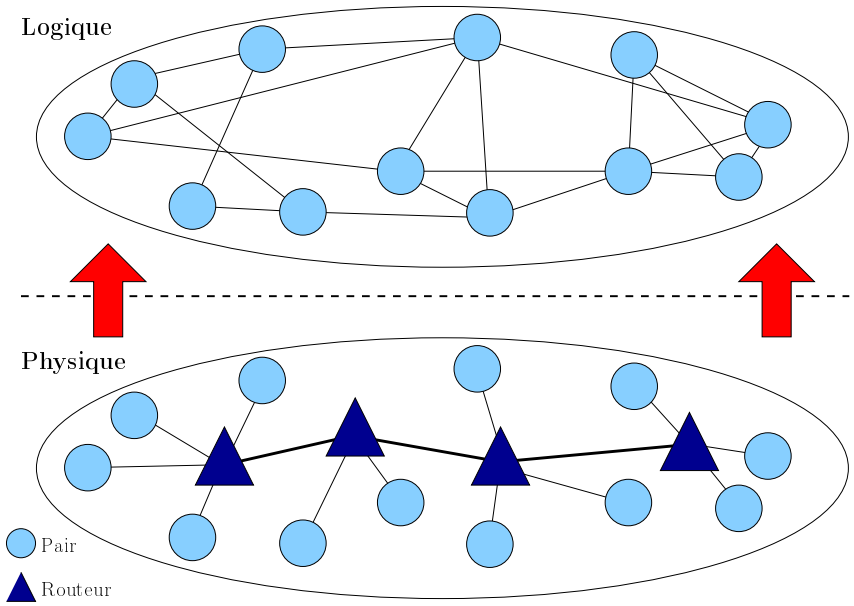



FIGURE 1.2 – Représentation d'un réseau logique ou overlay

deux nœuds pouvant passer par plusieurs liens physiques. Cette organisation est appelée **overlay** ou **réseau logique**.

**Définition 1** *On appelle overlay ou réseau logique l'interconnexion de pairs, représentée sous la forme d'un graphe, construite par une application pair-à-pair.*

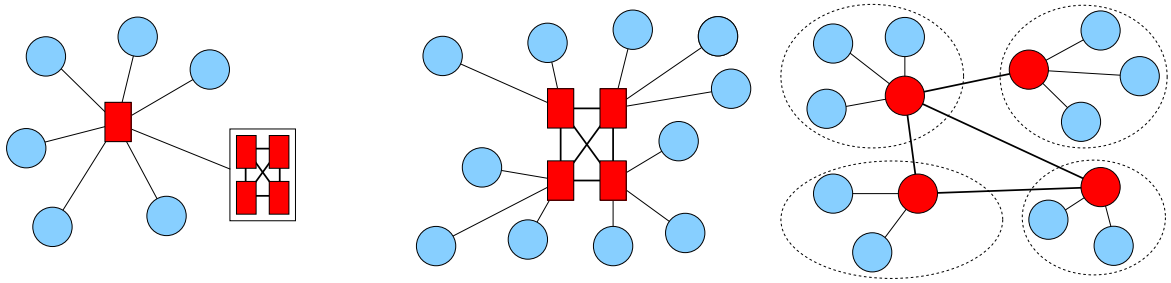
La figure 1.2 décrit la construction d'une telle structure. Les machines sont connectées sur le réseau physique. Une transposition des machines est effectuée dans un espace logique, chaque nœud possédant alors un voisinage de plusieurs pairs avec qui il peut communiquer.

L'évolution et la croissance des réseaux pair-à-pair a conduit la communauté scientifique à s'intéresser de près à la notion de réseau logique. Ainsi la construction d'application pair-à-pair repose sur des overlays possédant des propriétés héritées de la théorie des graphes. Ces propriétés portent essentiellement sur l'efficacité du routage de requêtes, sur le passage à l'échelle d'une application et sur la connexion et la déconnexion de pairs. Cette génération de réseaux pair-à-pair possède un comportement prouvé à l'aide de la théorie des graphes.

Une des propriétés forte du paradigme pair-à-pair est la dynamicité des nœuds. Chaque membre du réseau doit pouvoir se déconnecter ou se connecter au réseau de manière totalement transparente pour les autres utilisateurs. Ainsi une déconnexion non prévue ou une connexion simultanée de plusieurs pairs ne doit pas empêcher le service d'être fourni. Les propriétés de connexité du réseau logique doivent donc rester vérifiées au cours du temps.

## 1.2 Historique du P2P

Les modèles centralisés, semi-centralisés, hybrides et non-structurés sont les prémices des systèmes P2P. Les applications de partage de données appartenant à ces catégories ont démocratisé les réseaux pair-à-pair auprès du public. Nous allons maintenant présenter chacun des différents modèles au travers d'applications parmi les plus marquantes.



(a) Topologie utilisé dans Napster (b) Topologie utilisé dans eDonkey (c) Topologie utilisé dans KaZaA

FIGURE 1.3 – Modèle centralisé, semi-centralisé et hybride

### 1.2.1 Modèle centralisé

Napster [Napster, 1999] est la première application pair-à-pair grand public. Napster a été utilisé pour partager des fichiers audio de type mp3. Pour des raisons légales, Napster a du évoluer et se transformer en offre légale de téléchargement. Une des caractéristique de Napster est la topologie centralisée sous-jacente. La structure du réseau est la suivante :

- un serveur recense les fichiers stockés sur l'ensemble des pairs connectés,
- les clients se connectent au serveur pour connaître les fichiers présents sur le réseau.

Bien que nous soyons proche d'un modèle client/serveur, ici nous sommes bien en présence d'un modèle pair-à-pair. Ce sont les utilisateurs qui mettent à disposition les ressources. Ainsi un transfert de fichier se fera directement de pair à pair. Le serveur présent dans l'architecture permet simplement de fournir à un nœud émetteur d'une requête de recherche, la ou les sources possédant l'objet recherché. De cette manière le serveur ne stocke aucun fichier, mais permet la mise en relation des utilisateurs pour obtenir un fichier.

En réalité, il n'y a pas un mais plusieurs serveurs jouant le rôle d'index, chaque connexion étant répartie sur les serveurs par un méta-serveur. Cependant cette répartition est transparente pour un pair se connectant au service. Un utilisateur n'a pas le choix du serveur de connexion. La figure 1.3(a) montre la topologie de réseau employée par Napster.

Du fait du fonctionnement centralisé de Napster, le coût d'insertion et de suppression de pairs dans le réseau est faible. Il ne faut en effet qu'une seule opération pour se connecter ou se déconnecter au serveur. De même une machine ne répondant pas au bout d'un certain temps sera directement éliminée auprès du serveur. Le nombre maximum d'utilisateurs dépendant de la capacité des serveurs est clairement une limitation du modèle. Au delà d'un certain seuil

pouvant s'exprimer en fonction de la bande passante ou d'un nombre de connexions limité, il n'est plus possible d'ajouter des nœuds dans le réseau.

Ce type d'architecture centralisé peut être la cible d'attaques, empêchant l'accès au service, et ceci en dépit de la présence de plusieurs serveurs. La confidentialité des données et la protection de l'anonymat des utilisateurs ne peuvent également être garanties dans un tel système. Il est facile de mettre en place un contrôle des fichiers partagés et d'en identifier les propriétaires.

### 1.2.2 Modèle semi-centralisé

L'application eDonkey [eDonkey, 2000] est un logiciel de partage de fichiers de type assez similaire à Napster. Le protocole utilisé par eDonkey [Kulbak et Bickson, 2005], également appelé eDonkey, permet une interconnexion de serveurs de recherche. Ce protocole est à l'origine de nombreuses autres applications dont, la plus connue et encore utilisée de nos jours, eMule [eMule, 2002].

Ici l'idée d'un modèle centralisé n'est pas totalement abandonnée. Dans eDonkey, nous avons une interconnexion de serveurs, auxquels sont reliés les clients, appelés **feuilles**. Ces serveurs peuvent gérer un très grand nombre de feuilles, jusqu'à plus d'un million dans le cas d'eMule. L'interconnexion de serveurs n'a pas pour rôle de propager les requêtes des utilisateurs, mais sert essentiellement à maintenir une liste des serveurs disponibles. La figure 1.3(b) décrit une telle architecture.

Dans eDonkey, un serveur est responsable du recensement des fichiers stockés sur l'ensemble des pairs connectés. Pour connaître les fichiers présents sur le réseau les feuilles doivent envoyer leurs requêtes de recherche au serveur. Ici à la différence de Napster, un utilisateur peut choisir son serveur de connexion.

De la même manière que pour le modèle centralisé, le coût de connexion ou de déconnexion pour un nœud du réseau reste faible, une seule étape est nécessaire pour mettre à jour le réseau. A la différence de Napster, le goulot d'étranglement induit par l'utilisation d'un seul serveur est atténué ici. En effet, dans le cas où un serveur à atteint sa capacité maximale, un pair peut se connecter à un autre serveur.

### 1.2.3 Modèle hybride

Avec KaZaA [KaZaA, 2001], et à la différence des systèmes centralisés précédents, l'hétérogénéité des pairs est prise en compte. Dans ce modèle, des responsabilités sont attribuées à des pairs dans le réseau. Ces pairs sont appelés **super-nœuds** et se distinguent des feuilles par leur rôle plus important dans le réseau. Ils sont chargés d'effectuer la même charge de travail que les serveurs dans les modèles centralisés, et semi-centralisés.

KaZaA repose sur le protocole FastTrack [FastTrack, 2001]. Ce protocole est construit sur une interconnexion de super-nœuds. Ici les feuilles sont connectées à un seul super-nœud, chaque super-nœud pouvant accueillir 100 feuilles. L'avantage d'un tel réseau est de pouvoir s'affranchir

de serveurs, et ainsi de répartir les communications entre super-nœuds. La figure 1.3(c) montre une telle architecture.

Généralement, les super-nœuds sont des pairs possédant une bande passante plus élevée, permettant un acheminement plus rapide et plus fréquents des messages. Cependant le fait que des feuilles puissent se déclarer indépendamment super-nœud peut déséquilibrer l'ensemble et ainsi former un réseau principalement constitué de super-nœuds.

Dans une telle architecture, ce sont les super-nœuds qui jouent le rôle d'annuaire. À la connexion d'un nouveau pair, le super-nœud responsable reçoit la liste des fichiers partagés par la nouvelle feuille. De la même manière, les recherches effectuées dans le réseau le sont plus spécifiquement sur les super-nœuds.

Quand un utilisateur souhaite rejoindre le réseau, celui-ci se connecte à un super-nœud connu. Pour cela, il est nécessaire de recourir à un serveur recensant les super-nœuds actifs. Mis a part ce point de fonctionnement, le mécanisme est le même que pour les architectures centralisés et semi-centralisés.

### 1.2.4 Modèle décentralisé non structuré

Le modèle décentralisé non structuré marque les débuts de la décentralisation totale et d'une répartition équitable des fonctionnalités sur les nœuds. Ce modèle à pris la suite des modèles de première génération principalement pour faire face aux difficultés rencontrées par ces derniers, notamment en terme de confidentialité des données et d'anonymat. L'idée est de répartir l'ensemble des fonctionnalités sur l'ensemble des pairs, chaque pair possédant le même niveau de responsabilité. Ainsi les mécanismes de routage, de maintient, et de recherche de données, sont à la charge des pairs du réseau.

#### Gnutella

Gnutella [Gnutella, 2000] a été crée en 2000 par Frankel et Pepper. Une des particularités du protocole utilisé dans Gnutella est de n'avoir pas été publié à l'époque. Ainsi de multiples applications ont vu le jour en essayant de reproduire le comportement de Gnutella ce qui a conduit à de grandes incompatibilités.

Gnutella est une application appartenant au modèle décentralisé non-structuré. Ainsi chaque pair dans le réseau possède les mêmes responsabilités. Le réseau logique de Gnutella repose sur un graphe aléatoire. Chaque pair maintient une table de voisinage contenant les adresses de ses voisins dans le graphe. Dans un tel réseau, les liaisons sont symétriques. Ainsi si un pair $x$ possède un nœud $y$ dans son voisinage alors $y$ possède $x$ dans sa table de voisinage. Les tables de voisinage possèdent des bornes inférieures et supérieures en taille.

Pour se connecter, il est nécessaire de connaître un point d'entrée dans le réseau. Les caches d'adresses Gnutella présents sur le web sont chargés de répertorier les adresses des nœuds exécutant le protocole Gnutella. Les résultats de [GWCSR, 2010] montrent que le nombre de

requêtes sur de tels serveurs peut aller jusqu'à 300000 requêtes par heure. Avec autant de requêtes en si peu de temps, un utilisateur du réseau Gnutella ne peut pas fournir ce service.

Une autre solution consiste à utiliser une méthode dite de cache UDP. Quand un pair souhaite se connecter, il va dans un premier temps regarder les nœuds auquel il était connecté dans le passé, puis envoyer des pings parmi ceux-ci. Si un des nœuds répond par un pong, alors le pair souhaitant se connecter recevra une liste de machines présentes dans le réseau. L'avantage d'une telle méthode réside dans son faible coût.

#### Freenet

Freenet [Clarke *et al.*, 2000] a été crée pour préserver la confidentialité et l'anonymat des utilisateurs. Cette application utilise plusieurs outils cryptographiques pour parvenir à ces objectifs. Ici également l'overlay utilisé n'est pas structuré toutefois son évolutivité dans le temps fait qu'il peut être appelé réseau faiblement structuré.

Le réseau logique de Freenet est très proche des réseaux petit-monde [Milgram, 1967, Kleinberg, 1999, Kleinberg, 2000]. Un réseau petit-monde est vu comme un graphe connexe dans lequel deux sommets choisi aléatoirement sont reliés par 6 arêtes en moyenne. La distance moyenne entre deux pairs est alors d'environ 6 sauts. Cette structure suppose la présence de liens "longue distance" dans le réseau.

Lors de la connexion d'un nœud celui-ci se voit affecté un identifiant et se connecte à des pairs choisis aléatoirement. L'organisation des nœuds se fait progressivement dans le temps. Ici les nœuds font évoluer leur voisinage en choisissant des pairs dont leur identifiant est proche du leur.

La spécificité de Freenet est de pouvoir garantir l'anonymat des utilisateur. Pour cela Freenet utilise une méthode dite de routage en "oignon" [1] [Goldschlag *et al.*, 1999]. Lorsqu'un nœud $n_0$ souhaite envoyer un message à un pair destinataire $n_d$, $n_0$ détermine une route. Le message à envoyer est chiffré récursivement avec les clés publiques de chaque pair de la route en partant de l'arrivée. De cette manière le message est d'abord chiffré avec la clé publique de $n_{d-1}$ jusqu'à $n_1$. De plus à chaque étape l'identifiant du nœud chiffrant est associé pour déterminer simplement le prochain destinataire du message. Une fois l'"oignon" constitué le routage peut s'effectuer, chaque pair du chemin "pelant l'oignon" à l'aide de sa clé privé, et ne connaissant que son prédécesseur, et son successeur.

## 1.3 Modèle décentralisé structuré

Le but des applications adoptant un modèle décentralisé structuré est de reprendre les principes de décentralisation totale du réseau tout en apportant des garanties de fonctionnement, de passage à l'échelle et de performance. Ce sont des systèmes dont le fonctionnement peut être prouvé à partir de la théorie des graphes. Un des principaux objectifs de tels réseaux est de

1. onion routing

pouvoir fournir un mécanisme de recherche totalement décentralisé tout en équilibrant la charge entre les pairs et les liens. Pour y parvenir, un index décentralisé est construit sur l'ensemble des pairs. Cet index contient des entrées de la forme *(clé,objet)*. A chaque objet du réseau est associé une clé. Le mécanisme de recherche doit donc s'assurer que chaque pair puisse trouver une clé donnée, donc le nœud responsable de la clé. C'est pour cette raison, que la structure imposée doit garantir des propriétés sur l'établissement du voisinage des pairs.

### 1.3.1 Content Adressable Network

Le premier réseau exploitant les propriétés des graphes est CAN [Ratnasamy *et al.*, 2001], ou Content Adressable Network. Ici le réseau logique utilisé est semblable à un tore à $d$-dimensions. Les connexions sont non-orientées. Dans CAN les pairs possèdent un identifiant composé d'un intervalle pour chacune des $d$ dimensions. Le produit cartésien de ses coordonnées permet de définir une zone pour chaque nœud. La figure 1.4 permet d'observer la structure de l'overlay pour un espace à 2 dimensions et 5 nœuds.

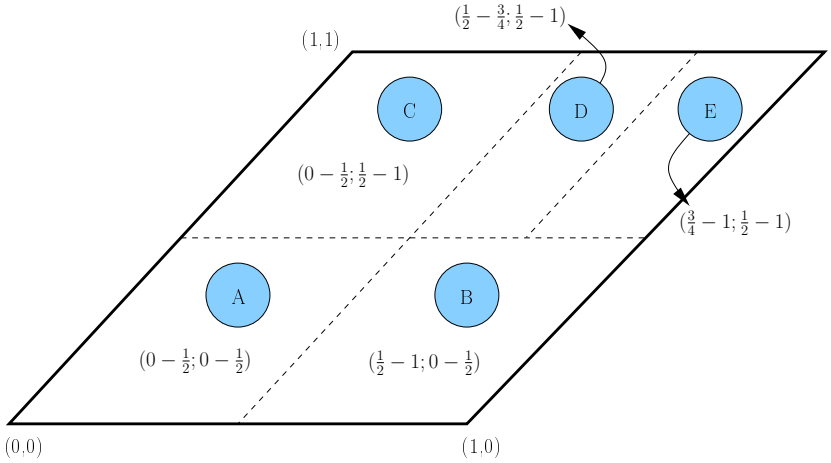
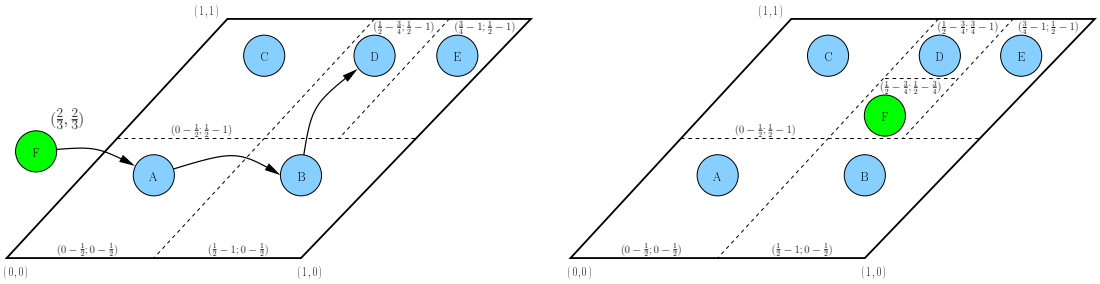



FIGURE 1.4 – Réseau CAN à 2 dimensions et 5 pairs

La figure précédente permet également de mettre en évidence le voisinage de chaque pair dans le tore. Un nœud possède un ensemble de pairs avec qui il partage ses frontières. Etant donné que chaque nœud est connecté en moyenne à 2 voisins dans chaque dimension, le degré moyen de chaque pair est $2d$.

Lorsqu'un pair souhaite rejoindre un réseau CAN, il va tout d'abord choisir un identifiant tiré aléatoirement sur $[0,1]^d$. Ensuite une requête de connexion est envoyée à cet identifiant. Une fois la demande reçue, le pair contacté va scinder sa région en deux parties égales pour en attribuer une au pair contactant. La figure 1.5 décrit un tel mécanisme. Le nœud $F$ choisit aléatoirement l'identifiant $(\frac{2}{3};\frac{2}{3})$. Un message est envoyé vers le nœud $D$ d'identifiant $(\frac{1}{2}-\frac{3}{4};\frac{1}{2}-1)$ responsable de cet identifiant (figure 1.5(a)). $D$ va donc scinder sa zone en deux parties égales, mettre à jour son identifiant, et attribuer la seconde zone au nœud $F$ rejoignant le réseau (figure 1.5(b)).




(a) Choix et routage d'un identifiant

(b) Séparation et attribution de la zone

FIGURE 1.5 – Protocole de connexion dans CAN

Le routage des messages dans CAN se fait de proche en proche suivant une politique de plus courte distance. Quand un nœud reçoit un message son comportement va être de le router vers le voisin possédant l'identifiant le plus proche de la destination. Cette proximité est obtenue par la distance de Manhattan.

### 1.3.2 Chord

Chord [Stoica *et al.*, 2003] a également été proposé en 2001. Ici c'est une structure logique en anneau qui a été retenue dans le but d'obtenir des bornes efficaces pour le degré moyen des pairs, le routage, et l'insertion de données dans le réseau. La figure 1.6 représente une telle structure.

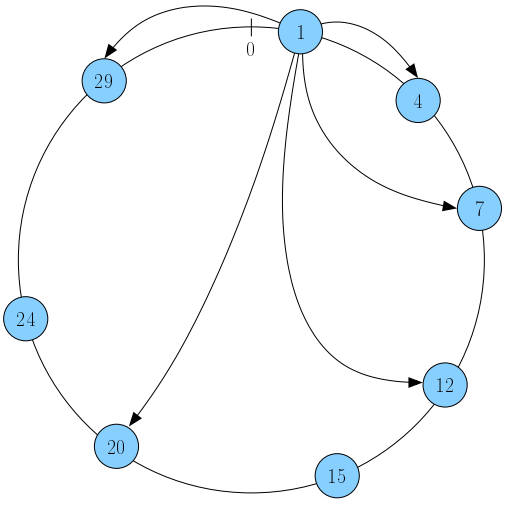



FIGURE 1.6 – Représentation logique de Chord

En réalité la structure logique est très proche d'un hypercube où les arêtes sont orientées. Le moyen de représenter cette structure est l'utilisation d'un anneau représentant l'espace de nommage $[O, 2^l]$ des objets et de nœuds bouclant en ses extrémités. Ici $l$ correspond au nombre

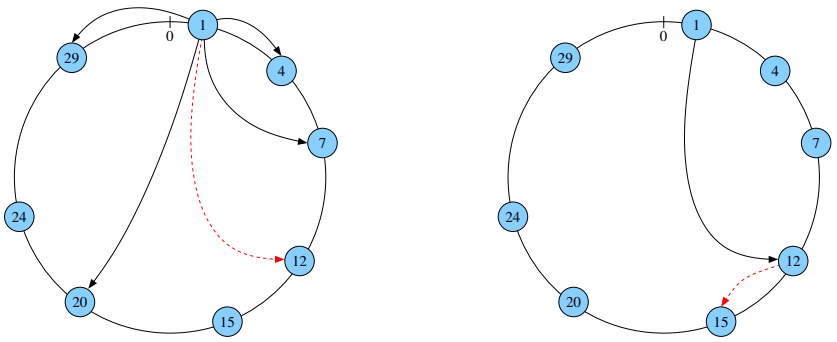
de bits d'une adresse logique dans le réseau. De manière générale $l$ est fixé à 160 et l'adresse logique est obtenue par hachage de l'adresse IP.

Le voisinage de chaque nœud $n$ du réseau est composé au minimum du successeur et du prédécesseur de $n$ sur l'anneau ($succ(n)$ et $pred(n)$) et d'un ensemble de pairs contenus dans une table (table finger) chargés d'accélérer le routage des informations. Cette table située sur un nœud $n$ contient les $k$ identités des pairs étant situés à au moins $2^k$ de $n$. Ainsi sur la figure 1.6 le pair d'identifiant 1 possède le nœud 4 (son successeur), le nœud 29 (son prédécesseur), et les nœuds 7, 12, et 20 pour voisins. Le degré moyen pour chaque pair dans Chord est de l'ordre de $O(\log n)$ pour un système possédant $n$ pairs actifs.

Pour entrer dans le réseau, un pair $x$ doit envoyer un message à destination du successeur de son identifiant ($succ(x)$). Une fois trouvé, $x$ demande le prédécesseur de $succ(x)$, à savoir $pred(succ(x))$. Une fois ces deux pairs trouvés, $x$ peut s'y insérer. Il ne reste plus qu'à construire la table finger de $x$ et de transférer les clés présentes sur $succ(x)$ comprises entre $pred(x)$ et $x$.

La déconnexion d'un nœud repose sur le transfert de clé. Si un pair $x$ vient à quitter le réseau, alors il doit prévenir son successeur et son prédécesseur pour mettre à jour les connexions dans l'anneau. $x$ doit également transférer les clés dont il est responsable à son successeur. Enfin les tables finger des pairs du réseau doivent être mises à jour. Dans le cas où $x$ tombe en panne et n'est donc pas dans la capacité de prévenir son successeur et son prédécesseur, il faut que les nœuds possédant $x$ dans leur voisinage puisse contacter $succ(x)$. Ceci peut-être effectué en conservant sur chaque nœud une liste de $r$ successeurs directs.

L'algorithme de routage de l'information repose sur la recherche de plus grand prédécesseur d'une donnée $d$. Etant donné que les pairs et les données partagent le même espace de nommage, et que un nœud est désigné responsable d'une clé si celui-ci est le premier identifiant supérieur à la donnée, il suffit pour un pair $x$ de regarder si la clé à router est comprise entre lui et son successeur. Si ce n'est pas le cas, il transmet la requête au plus grand prédécesseur de $d$ présent dans sa finger table. La figure 1.7 décrit un tel mécanisme.




(a) Routage de 1 vers 12

(b) Routage de 12 vers 15

FIGURE 1.7 – Routage de l'information dans Chord

Ici le pair d'identifiant 1 souhaite rechercher une donnée d'identifiant 14. Il va donc transmettre le message au nœud 12, plus grand prédécesseur de 14 dans sa table de voisinage (figure 1.7(a)). 14 étant compris entre 12 et $succ(12) = 15$, la ressource recherchée est sur le pair 15 (figure 1.7(b)).

### 1.3.3 Tapestry

Tapestry [Zhao *et al.*, 2001, Zhao *et al.*, 2004] présente également une structure proche de l'hypercube avec des arêtes orientées. De la même manière que pour les protocoles précédents, l'espace de nommage des objets et des pairs est identique. Ici les identifiants sont construits à partir d'un alphabet de taille $\beta$. Dans Tapestry c'est un alphabet hexadécimal qui est choisi donc $\beta = 16$. Chaque identifiant contient $l = \frac{\log n}{\log \beta}$ chiffres, avec $n$ le nombre de pairs du réseau. Dans Tapestry chaque pair $x$ contient une table de voisins de $\beta l$ entrées, chaque entrée possédant un suffixe commun avec $x$. La figure 1.8 montre un tel overlay.

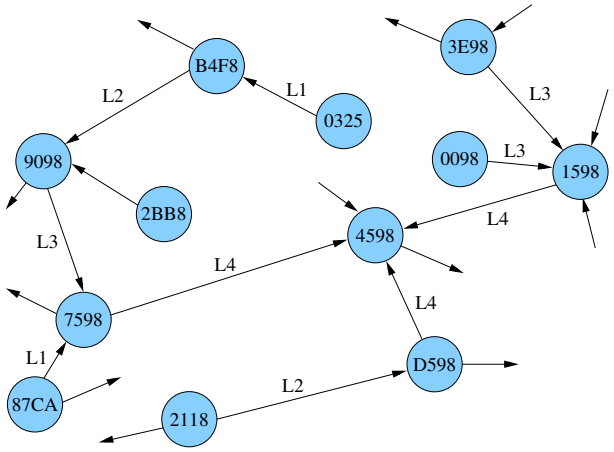


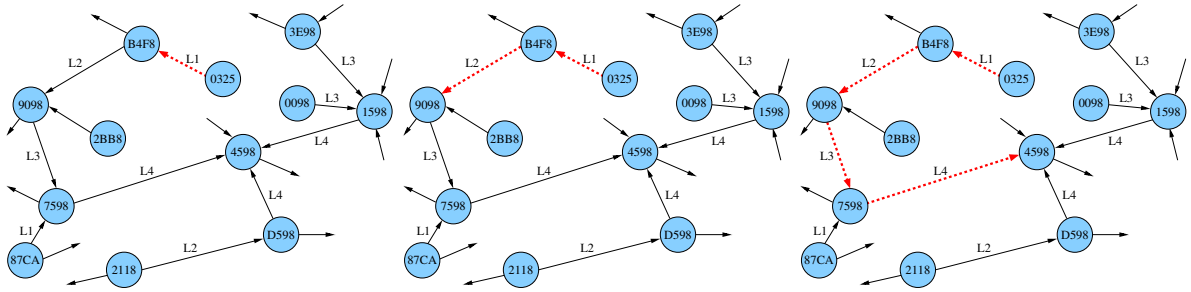

FIGURE 1.8 – Overlay utilisé dans Tapestry

Ici les arêtes sont annotées pour représenter le niveau de voisinage d'un nœud. Ainsi une étiquette $L1$ signifie que les deux pairs n'ont pas de suffixe communs. De la même manière, une étiquette $Lx$ signifie que deux pairs partagent un suffixe de taille $x - 1$. De plus chaque pair $x$ possède dans sa table de voisinage les identifiants des pairs ayant $x$ pour voisin. Le degré moyen de Tapestry est de l'ordre de $O(\beta \frac{\log n}{\log \beta})$.

Pour rejoindre un réseau Tapestry, un pair $x$ doit tout d'abord choisir un identifiant $id_x$. Une fois choisi, c'est l'algorithme de routage qui permet d'attribuer à $x$ ses voisins. $x$ envoie un message à destination de $id_x$ en passant par une passerelle $y$. A chaque étape du routage $x$ reçoit les identifiants des nœuds ayant un suffixe commun avec $y$. La dernière étape consiste pour $x$ à réorganiser ses voisins en fonction de leur proximité.

Le routage de l'information dans Tapestry est établi sur un algorithme suffixe. Cet algorithme consiste à router un message sur un pair possédant un suffixe commun avec la destination. La figure 1.9 montre les étapes pour acheminer un message du pair d'identifiant 0325 à destination

du nœud 4598. Le message est routé dans un premier temps au pair $B4F8$, possédant un suffixe commun 8 avec la destination (figure 1.9(a)). Puis le pair $B4F8$ envoie le message à 9098 pour se "rapprocher" de la destination (figure 1.9(b)). Le suffixe est alors équivalent à 98. On procède de la même manière pour acheminer le message au pair 4598 (figure 1.9(c)). Dans le cas où un pair ne trouve pas de nœud pour router le message, il va alors envoyer le message à un voisin possédant le même suffixe.




(a) Routage de 0325 à $B4F8$ (b) Routage de $B4F8$ à 9098 (c) Routage vers 4598

FIGURE 1.9 – Routage des messages dans Tapestry

Le départ de nœud s'appuie sur la republication de clé. Quand un pair $x$ quitte volontairement le réseau, il prévient les nœuds dont il est voisin. $x$ prévient également les pairs hébergeant les ressources dont il possède les clés pour que ces derniers republient les clés associées aux données. Dans le cadre d'un départ non volontaire, le système supprime les références d'objets si ceux-ci ne sont pas republiés au bout d'un certain temps. De plus les pairs assurent une réparation des liens de voisinage cassés.

### 1.3.4 Kademlia

La topologie de Kademlia [Maymounkov et Mazières, 2002] est proche de celle de l'hypercube. Ici la structure retenue pour représenter l'overlay est un arbre binaire préfixe. Comme pour les protocoles précédents les pairs et les objets du système partagent le même espace de nommage. Pour trouver un objet, il suffit de contacter le pair qui en est responsable. Ici un pair ou un objet est identifié sur $[O, 2^l[$, $l$ étant un paramètre du système. La structure d'arbre binaire adoptée permet d'assigner à chaque bit d'un identifiant un sous-arbre. Ce sous-arbre permet de constituer le voisinage d'un nœud. La figure 1.10 montre une telle structure.

Un pair $x$ conserve ses contacts dans des *k-buckets*, contenant donc $k$ nœuds $y$ tels que :

$$2^i < x \oplus y < 2^{i+1}$$

Dans un $k$-bucket un pair $x$ conserve $k$ nœuds $y$ situés à une distance comprise entre $2^i$ et $2^{i+1}$. Ces nœuds $y$ sont ordonnés du plus ancien au plus récent. Cette notion de distance a également été choisie dans Pastry [Rowstron et Druschel, 2001a]. La figure 1.11 montre les $k$-buckets pour le pair 0011. Ici seulement 2 pairs sont conservés.

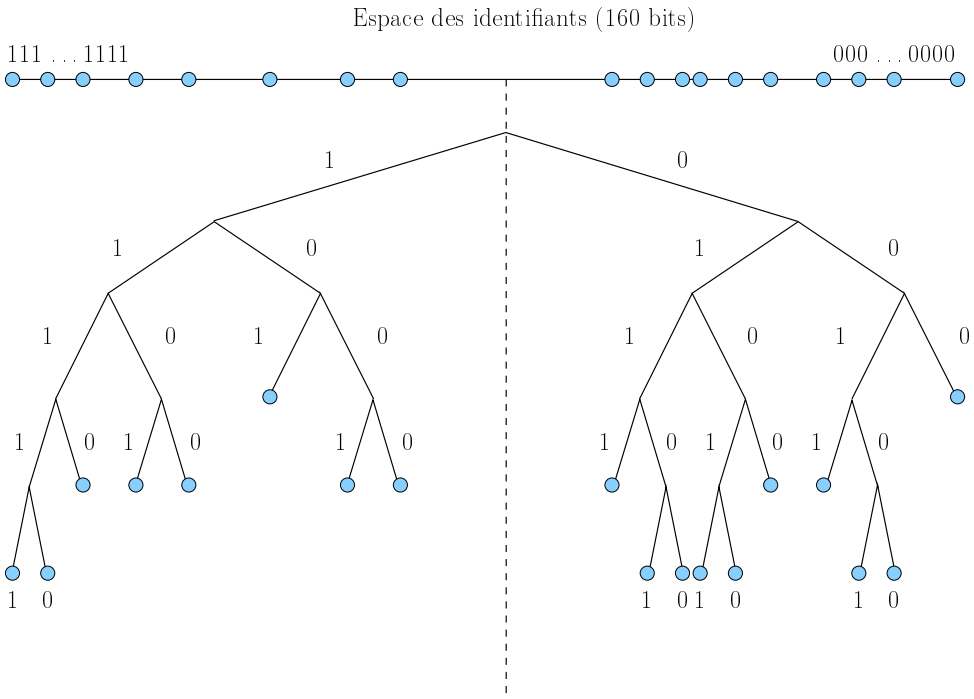



FIGURE 1.10 – Structure d'arbre binaire de Kademlia

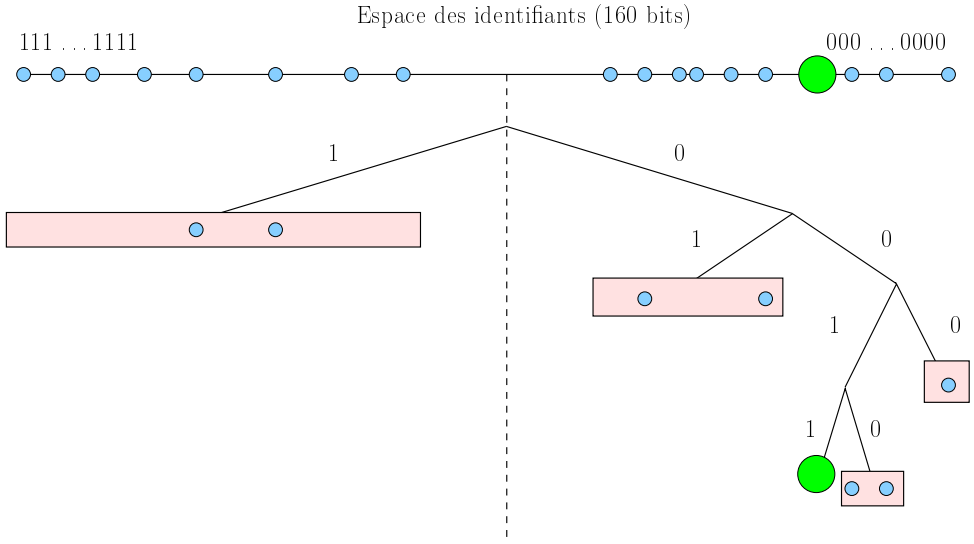
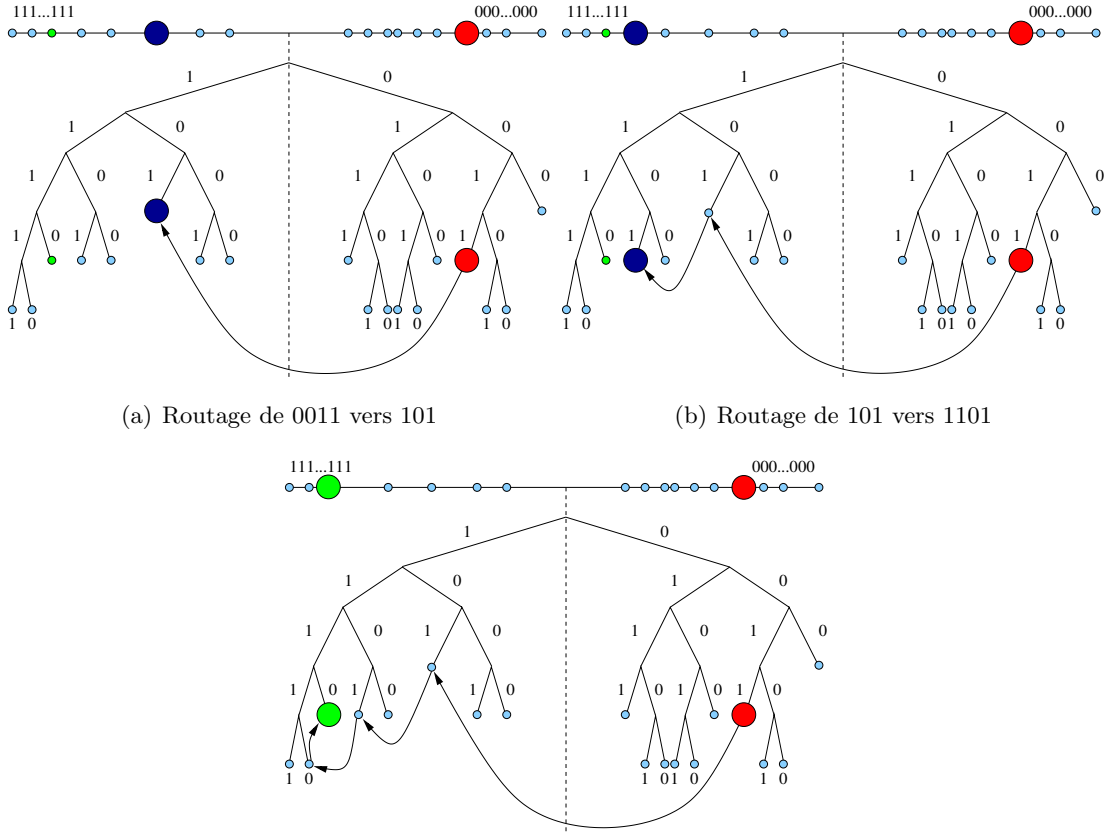



FIGURE 1.11 – 2 – *buckets* pour le pair 0011

L'utilisation des $k$-buckets facilite le routage des messages. Prenons le cas où le pair d'identifiant 0011 veut adresser un message au nœud 1110. 0011 consulte les $k$ plus proches nœuds dans ses $k$-buckets, qui consulteront à leur tour leur $k$ plus proches nœuds. La figure 1.12 décrit un tel mécanisme. 0011 envoie un message à 101 (figure 1.12(a)), qui envoie à son tour un message à 1101 (figure 1.12(b)). Le processus se répète jusqu'au pair destinataire 1110 en passant par le nœud 11110 (figure 1.12(c)).




(a) Routage de 0011 vers 101

(b) Routage de 101 vers 1101

(c) Routage vers 1110

FIGURE 1.12 – Routage de l'information dans Kademlia

Pour entrer dans le réseau, un nouveau pair va choisir aléatoirement son identifiant sur $[0, 2^l[$, puis par l'intermédiaire d'une passerelle qu'il va ajouter à son sous-arbre, $x$ va effectuer une recherche sur son identifiant. A chaque étape du routage, $x$ va ajouter à ses sous-arbres des voisins de plus en plus proches de lui. Ce mécanisme permet également de prévenir les nœuds contactés de l'arrivée de $x$ dans le réseau.

Pour le départ de pair du réseau, il n'y a pas eu d'opérations spécifiques de prévues. La re-publication des clés, et la découverte de nouveaux nœuds lors de leur arrivée permet de résoudre ce problème. Ici la publication d'une clé $c$ consiste à enregistrer $c$ sur les $k$ plus proches nœuds de $c$. La re-publication de $c$ est effectuée toutes les heures pour le responsable, et chaque 24 heures pour le propriétaire.

## 1.4 Réseaux auto-organisés : étude des protocoles épidémiques

Dans un contexte de décentralisation, et de fort dynamisme des nœuds, les protocoles épidémiques permettent la construction de réseaux logiques. Ils possèdent des caractéristiques proches des graphes aléatoires. Dans de tels protocoles, chaque pair joue un rôle dans la construction et le maintien du réseau, ces deux étapes reposant sur les connaissances des nœuds.

Chaque pair possède et publie des informations dans le temps, l'objectif étant de posséder à terme une connaissance complète des informations présentes dans le réseau. La publication de données prend la forme d'un échange mutuel périodique d'information entre deux nœuds. Les informations sont donc propagées dans le réseau de manière épidémique.

Les premiers protocoles épidémiques ont été introduits dans [Demers *et al.*, 1987] et utilisés pour la maintenance d'une base de données répartie. Ici les informations circulant dans le réseau sont des mises à jour de copies d'une base de données, l'objectif étant de posséder à terme une copie à jour de la base de données. Les résultats qui ont été obtenus ont permis de réduire grandement le trafic sur les connections entre nœuds et ceci avec de simples algorithmes.

### 1.4.1 Protocole épidémique générique

Un protocole épidémique repose sur un échange d'informations entre deux nœuds du réseau. Pour parvenir à cet échange, il est nécessaire d'effectuer plusieurs opérations pour un pair $p$ :

- choisir un pair avec qui échanger des informations,
- choisir les informations à envoyer,
- réceptionner les informations reçues, et,
- mettre à jour ses informations locales.

Ces opérations sont résumées dans les algorithmes 1. Deux processus distincts sont exécutés en parallèle sur chaque pair. Un processus actif initiant l'échange d'information, et un passif permettant une réponse au protocole actif. Ici nous avons pris l'exemple de deux pairs échangeant des informations sur leur voisinage respectifs. Un pair $p$ choisit un pair $p_{echange}$ dans le réseau. $p$ choisi un ensemble de voisins à envoyer à $p_{echange}$. Sur réception de la liste de voisins de $p$, $p_{echange}$ effectue les mêmes opérations de sélection d'informations puis d'envoi. Enfin $p$ et $p_{echange}$ mettent à jour leur voisinage en fonction des données échangées.

A partir de ces deux processus génériques simples, il est possible de proposer des algorithmes de construction de réseaux logiques totalement décentralisés de types non-structurés et structurés.

### 1.4.2 Construction de réseaux décentralisés non-structurés

#### Maintien du réseau

De nombreux algorithmes pour la construction et le maintien de réseaux non-structurés ont été proposés. Nous pouvons citer par exemple Lpbcast [Eugster *et al.*, 2003], Newscast [Voulgaris *et al.*, 2003], et Cyclon [Voulgaris *et al.*, 2005]. L'ensemble de ces algorithmes partagent les

**Algorithme 1** : Protocole épidémique générique

```
//Processus actif;
début
    tant que vrai faire
        attendre(nbCycles);
        p_echange ←selectionPair();
        aEnvoyer ←selectionEnvoi();
        envoyer aEnvoyer à p_echange;
        recu ← aEnvoyer de p_echange;
        voisinage ← selectionVoisinage(recu);
fin
```

```
//Processus passif;
début
    tant que vrai faire
        recu←aEnvoyer de p;
        aEnvoyer←selectionEnvoi();
        envoyer aEnvoyer à p;
        voisinage←selectionVoisinage(recu);
fin
```

propriétés des graphes aléatoires héritées du modèle d'Erdös et Rényi [Erdös et Rényi, 1960]. Leur fonctionnement repose sur l'algorithme générique précédent, leurs différences venant des différentes fonctions utilisées.

Dans Newscast [Voulgaris *et al.*, 2003] par exemple le choix du pair avec qui échanger des informations (*selectionPair()*) se fait aléatoirement alors que dans Cyclon [Voulgaris *et al.*, 2005] ce choix se fait suivant une politique Least Recently Used (LRU) permettant ainsi d'éliminer les pairs ne participant plus au réseau.

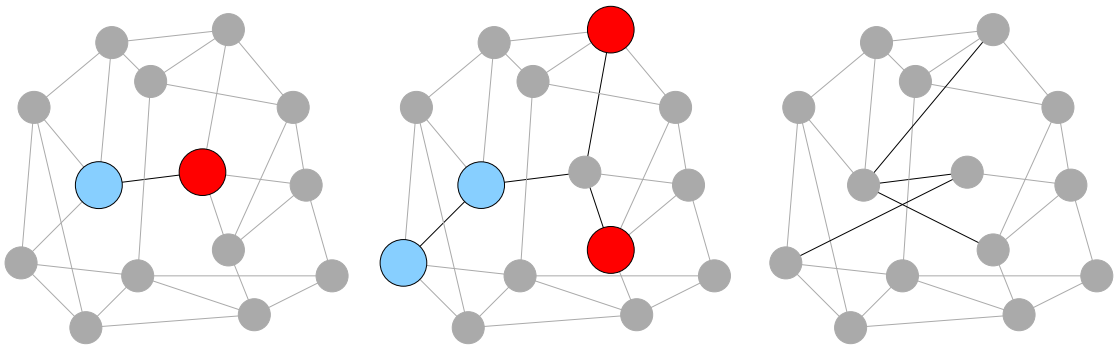
La méthode *selectionEnvoi()* de Newscast permet d'envoyer l'ensemble des données présentes dans le cache des pairs. C'est donc un cache de $2n + 1$ entrées qui est obtenu sur chaque pair exécutant le protocole. La méthode *selectionVoisinage()* permet de sélectionner les $n$ plus récentes entrées du cache.

Dans Cyclon, c'est un nombre aléatoire d'informations qui sont sélectionnées par la méthode *selectionEnvoi()*, puis envoyées par les pairs. Une fois reçues les informations sont agrégées aux données locales, puis la méthode *selectionVoisinage()* se charge de ramener le cache à une taille initiale $n$ en éliminant en priorité les informations qui ont été envoyées.

La figure 1.13 décrit le fonctionnement du protocole Cyclon. Le pair bleu choisit un pair suivant une politique LRU (figure 1.13(a)), puis sélectionne dans son voisinage n pairs aléatoirement. Ici c'est un seul pair ainsi que l'identifiant du pair bleu qui sont envoyés au pair vert, qui en réponse retourne deux identifiants pris aléatoirement dans son voisinage (figure 1.13(b)). Le protocole se termine par l'échange des liens entre les deux pairs (figure 1.13(c)).

### Ajout/Suppression de pairs

Dans les protocoles épidémiques l'ajout de nouveaux pairs dans le réseau possède également un fonctionnement assez simple. En effet il suffit pour un nouvel arrivant de se construire un cache de données contenant des adresses de nœuds du réseau. Dans Cyclon par exemple, l'opération consiste à envoyer un marcheur aléatoire dans le réseau pour construire un voisinage de taille



(a) *selectionPair()* (b) *selectionEnvoi()* (c) *selectionVoisinage()*

FIGURE 1.13 – Protocole épidémique appliqué à Cyclon

$c$ puis à exécuter les algorithmes 1. Dans Newscast, il suffit juste à un nouveau participant de construire son cache avec au moins un pair du réseau, puis d'exécuter le protocole épidémique précédent.

La suppression d'un pair dans Cyclon se fait naturellement. En effet, un nœud ne répondant pas à une demande d'échange est directement effacé du cache. De plus grâce à la politique LRU appliquée à l'étape de sélection du pair avec qui communiquer (*selectionPair()*), on a l'assurance que le nœud incriminé disparaisse dans la vue des autres pairs en $c - 1$ cycles au maximum.

Newscast présente le même mécanisme d'oubli au détail près que les pairs les plus anciens sont éliminées à l'étape de sélection du voisinage en fin de protocole, et donc par conséquent, un pair ne répondant plus sera éliminé progressivement de la vue des pairs.

### 1.4.3 Construction de réseaux décentralisés structurés

A partir des protocoles épidémiques génériques (Algorithme 1), il est également possible de construire des réseaux possédant une structure bien définie. C'est notament le cas dans T-man [Jelasity *et al.*, 2009] et Vicinity [Voulgaris et van Steen, 2005].

#### Construction d'une structure

La différence avec les précédents protocoles et T-man [Jelasity *et al.*, 2009] provient de l'utilisation d'une méthode de classement appelée *rank()*. Cette méthode permet de d'établir et de classer le voisinage d'un pair dans le réseau suivant ses préférences. Ainsi dans un tel protocole, l'étape d'envoi d'une vue d'un pair vers un autre est précédé d'un classement du voisinage. La méthode de classement est obtenue en fonction de la structure que l'on souhaite obtenir. Ainsi dans T-man, des exemples sont montrés avec une structure en anneau, en tore, et un graphe aléatoire où les nœuds ont un degré sortant de $k$.

Cette méthode de classement est beaucoup plus générique qu'une notion de distance. Dans le cas d'une structure en cercle, un nœud $x$ assigne à chaque membre du réseau $y$, un rang consistant à compter le nombre de saut pour atteindre $y$.

L'étape de sélection d'un pair repose également sur la méthode de classement. Un pair va choisir aléatoirement un nœud non pas dans l'intégralité de sa vue, mais sur les $\gamma$ premiers pairs de sa vue triée.

Dans Vicinity [Voulgaris et van Steen, 2005], les auteurs veulent maximiser le taux de réussite de la première recherche dans un réseau. Pour cela ils construisent un réseau de proximité sémantique à l'aide d'un protocole épidémique. Ainsi un pair maintient une liste de $l$ pairs sémantiquement proche, auquel il s'adresse en premier pour rechercher une donnée. Si le résultat n'est pas concluant, le pair effectue une recherche "classique" dans le réseau.

L'étape de sélection du pair à qui envoyer une vue de son voisinage est similaire à celle de Cyclon [Voulgaris *et al.*, 2005]. C'est une politique LRU qui est utilisée, et c'est donc le pair $x$ avec l'horodatage le plus ancien qui est choisi. La sélection des pairs à envoyer consiste à choisir les pairs les plus proches sémantiquement de $x$, puis de les envoyer à $x$. En réponse $x$ envoie les pairs les plus proches sémantiquement du pair initiateur du protocole.

Une fois les pairs reçus, la dernière étape consiste à conserver les $l$ pairs les plus proches parmi le voisinage courant et la sélection reçue.

Ces deux protocoles permettent de maintenir une structure dans un réseau. Toutefois T-man et Vicinity peuvent s'accomoder d'un second protocole épidémique responsable du maintien du réseau. En effet T-man et Vicinity permettent de construire des agrégats de pairs dans le réseau, et il est nécessaire de conserver des liaisons entre ces agrégats. Cette tâche est à la charge du second protocole épidémique. Dans Vicinity, c'est Cyclon qui est utilisé. Ainsi les opérations de sélection de pairs et de mise à jour du voisinage peuvent prendre en compte la vue héritée de Cyclon. Les preuves de la nécessité d'utiliser un second protocoles épidémiques pour le maintien du réseau sont présentes dans [Voulgaris, 2006].

## Conclusion du chapitre

Ce chapitre nous a permis d'exposer la première brique essentielle à une application de partage de fichier à savoir l'aspect réseau. Nous avons présenté ce qu'est un réseau pair-à-pair et nous en avons dressé un panorama nous permettant de mieux appréhender ce que doit être un réseau ainsi que les propriétés qu'il doit nécessairement avoir. Le cas des protocoles épidémiques est des plus intéressants et convient parfaitement à l'idée que nous nous faisons d'un réseau décentralisé. Il doit être léger dans le sens où le coût de sa construction et de son maintien ne doit pas influencer les performances globales du système. Le réseau doit également être autonome et ainsi être capable de s'organiser et de prendre en compte les fautes et pannes éventuelles de machines, sans en référer à une quelconque autorité. Les protocoles épidémiques remplissent parfaitement ces conditions.

# Chapitre 2

# Cycle de vie de l'information

**Sommaire**



Dans ce chapitre nous allons présenter les méthodes mises en œuvre dans les applications de stockage de données pour assurer le cycle de vie de l'information. Par cycle de vie nous entendons trois étapes fondamentales à savoir :

- Le découpage de l'information, où comment manipuler un fichier pour garantir la persistance de l'information.
- Le placement de l'information, où comment disperser les fragments générés pour garantir une répartition de charge efficace.
- La recherche de l'information, où comment retrouver une donnée disséminée dans un réseau.

## 2.1 Disponibilité de l'information

Le rôle de la disponibilité de l'information est d'assurer la persistance et l'accessibilité d'une information dans le temps. La disponibilité d'un fichier doit être assurée en respectant certains critères tels que :

- Le coût d'espace de stockage doit être le plus faible possible.
- La garantie de la confidentialité des données disponibles.
- Les opérations de lecture et d'écriture doivent être effectuées en utilisant le moins de ressources possibles. Les ressources se traduisent par le temps de calcul utilisé et par la quantité d'information transitant dans le réseau (bande passante).

La plupart des approches observées consistent à dupliquer une information [2] pour s'assurer qu'une donnée reste disponible dans le temps. Cette duplication peut prendre plusieurs formes et il est nécessaire de s'assurer que les critères précédents soient bien respectés. Pour comparer les différentes approches observées dans la littérature nous allons tout d'abord définir les notions essentielles pour la bonne compréhension des schémas suivants.

**Définition 2** *La probabilité de défaillance notée $P(def|nb_{pertes})$ désigne la probabilité de perdre un fichier dans un réseau connaissant le nombre de pertes concomitantes d'informations dupliquées $nb_{pertes}$.*

**Définition 3** *Le facteur de redondance désigne l'augmentation du stockage dans un réseau. Ce facteur est représenté par :*

$$r = \frac{|Stock|}{|Original|},$$

*avec $|Stock|$ le coût total des objets stockés, et $|Original|$ le coût de stockage de la donnée originelle.*

**Définition 4** *Le degré de réparation correspond au nombre de morceaux d'informations nécessaires pour reconstruire une donnée perdue. Ce degré est noté $d$.*

### 2.1.1 Réplication des données

La méthode la plus simple pour assurer la disponibilité d'une information est de la dupliquer entièrement. Cette méthode permet de faire face à différentes défaillances quelles soient humaines (effacement involontaire d'un fichier) ou matérielles (panne machine). Soit $Rep_F$ le nombre de réplications d'un fichier $F$. Les réplicats de $F$ sont stockés sur des pairs distincts dans le réseau. Ce système de réplication supporte un nombre de pertes inférieure à $Rep_F$. Si l'ensemble des réplicats de $F$ sont perdus alors ce dernier n'est plus accessible. Nous avons donc la probabilité de défaillance suivante :

$$P(def|nb_{Pertes}) = \begin{cases} 0 & nb_{Pertes} < Rep_F \\ 1 & nb_{Pertes} = Rep_F \end{cases}$$

Dans le cas de la réplication des données, le facteur de redondance est $r = Rep_F$, et le degré de réparation est $d = 1$, une seule donnée étant nécessaire pour en recréer une autre.

Parmi les applications utilisant la réplication de donnée pour assurer la pérennité d'une information nous pouvons citer Freenet [Clarke *et al.*, 2000], CAN [Ratnasamy *et al.*, 2001], et

2. ou des morceaux d'information

Chord [Stoica *et al.*, 2003]. Une telle démarche pour assurer la disponibilité n'est clairement pas efficace en terme d'espace de stockage de donnée [Weatherspoon et Kubiatowicz, 2002, Rodrigues et Liskov, 2005].

Pour illustrer ces propos prenons en exemple un fichier $f$ de taille 100Mo. Pour que $f$ puisse supporter 10 pertes concomitantes, il est nécessaire de le dupliquer 10 fois ($r = 10$). L'espace occupé dans le réseau est alors multiplié par 10 ce qui représente 1Go d'espace supplémentaire. Etant donné qu'une seule donnée est suffisante pour retrouver $f$, nous avons un degré de réparation égal à 1 ($d = 1$).

### 2.1.2 Partage de secret / Schéma à seuil

Le but du partage de secret est de partager un secret parmi un ensemble de participants de telle sorte que seul un sous-ensemble de participants, mettant en commun leurs informations, puisse reconstruire le secret. Chaque sous-ensemble de participants non-éligible à la reconstruction ne possède de connaissances sur le secret. Le partage de secret à été introduit parallèlement par Shamir [Shamir, 1979] et Blakley [Blakley, 1979]. Les schémas de Shamir et Blakley consistant à partager un secret parmi $n$ participants et dont la reconstruction impose la présence d'au moins $k$ d'entre eux, sont appelés des **$(\mathbf{k}, \mathbf{n})$-schémas à seuil**.

#### Schéma de Shamir

Shamir a proposé un algorithme permettant de partager un secret entre plusieurs utilisateurs. A partir d'un fichier $F$, $n$ fragments chiffrés sont créés de telle sorte que $k$ fragments parmi les $n$ sont au minimum nécessaires à la reconstruction du secret.

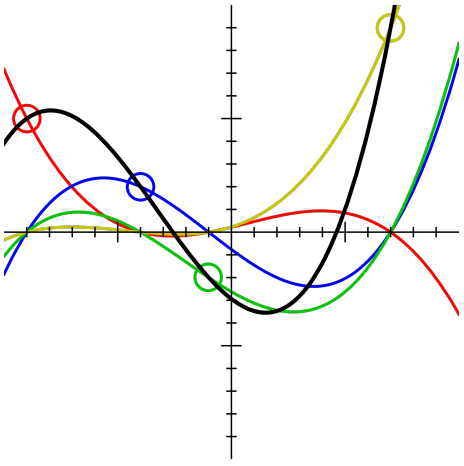


FIGURE 2.1 – Schéma de Shamir construit sur une interpolation de Lagrange

L'algorithme décrit dans [Shamir, 1979] repose sur l'utilisation d'une interpolation polynomiale de Lagrange. La figure 2.1 montre un exemple d'interpolation de Lagrange. Ici quatre points sont nécessaires pour définir une courbe cubique, soit un polynôme de degré 3. De manière

plus générale, $k$ points sont nécessaires pour définir un polynôme de degré $k-1$. Nous avons donc le théorème suivant :

**Théoreme 1** *Soit les $q$ points du plan $\{(x_1, y_1), \ldots, (x_q, y_q)\}$ avec $x_i \neq x_j$ pour $\forall i, j = 1, \ldots, q$. Il n'existe alors qu'un seul et unique polynôme $P(x)$ de degré $q-1$ tel que :*

$$P(x_i) = y_i, \forall i = 1, \ldots, q$$

Etant donné que la donnée est stockée sous forme d'une suite de bits, il est possible de représenter $F$ comme un nombre. Il est donc possible de choisir un polynôme $P(x)$ de degré $k-1$ tel que :

$$P(x) = c_0 + c_1 x^1 + \cdots + c_{k-1} x^{k-1} \text{ avec } c_0 = F$$

Pour diviser la donnée $F$ en morceaux $f_i$ il faut évaluer :

$$f_1 = P(1), \ldots, f_i = P(i), \ldots, f_n = P(n)$$

En procédant ainsi, l'objet $F$ peut-être reconstruit à partir d'un sous-ensemble de $k$ parmi $n$ valeurs de $f_i$. L'opération consiste à retrouver les coefficients de $P(x)$ par interpolation polynomiale et à évaluer $P(0) = F$. La connaissance de $k-1$ valeurs ne permet pas la reconstruction de $F$.

### Schéma de Blakley

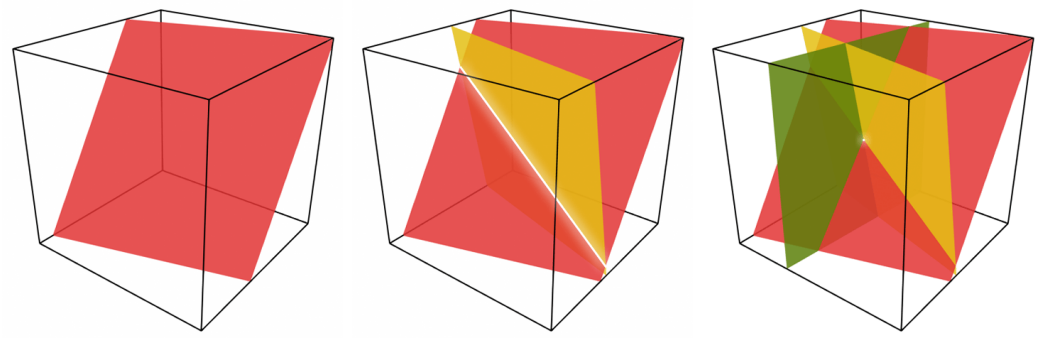
Le schéma de Blakley [Blakley, 1979], développé en même temps que celui de Shamir, est construit sur l'intersection d'hyperplans. Un secret est représenté par un point dans l'espace. Un hyperplan de dimension $k$ représente une part de secret. Sachant que $k$ hyperplans de dimension $k$ se recoupent en un seul point dans l'espace, pour retrouver un secret il faut réunir $k$ hyperplans. La figure 2.2 décrit un tel schéma dans un espace à trois dimensions. Dans la figure 2.2(a) le plan désigne une part de secret. La figure 2.2(b) montre que deux plans sont insuffisants pour reconstruire le secret. Dans la figure 2.2(c) trois plans se coupent en un seul point, le secret peut être reconstruit.

Un hyperplan dans un espace de dimension $k$ peut être défini par une équation linéaire de la forme :

$$a_1 x_1 + a_2 x_2 + \cdots + a_k x_k = b$$

Lors d'un partage de secret, un point $s$ est généré, où la première coordonnée de $s$ est le secret. Les autres valeurs sont fixées aléatoirement. Nous avons donc le vecteur $s = (s_1, s_2, \ldots, s_k)$ où $s_1$ est le secret. Soit $P$ l'ensemble des utilisateurs pouvant recevoir une part de secret. L'opération de construction du partage de secret consiste alors à construire $n$ équations linéaires d'hyperplans de dimension $k$ de la forme :

$$a_{i1} s_1 + a_{i2} s_2 + \cdots + a_{ik} s_k = y_i$$



(a) (b) (c)

FIGURE 2.2 – Schéma de Blakley construit sur une intersection de plans

où chaque participant $P_i$, $i = 1, \ldots, n$, reçoit lors du partage de secret la valeur $y_i$ associée aux coefficients $a_{i1}, \ldots, a_{ik}$. Les coefficients $a_{ij}$ peuvent être rendus publics.

La phase de reconstruction consiste à réunir $k$ participants parmi $n$. Ils peuvent ainsi former un système linéaire d'équations d'hyperplan et résoudre :

$$A_p x = y_p,$$

où $y_p$ est le vecteur de valeur de part de secret des participants. La première coordonnée de la solution est alors le secret.

### Généralisation

[Blakley et Meadows, 1985] ont proposé une généralisation des schémas à seuil en introduisant le schéma à rampe [3]. Un troisième paramètre $p$ est ajouté aux $(k, n)$-schémas à seuil pour fixer un nombre minimum de fragments ne révélant aucune information. On parle alors de $(\mathbf{p}, \mathbf{k}, \mathbf{n})$-**schémas à seuil**. Le tableau 2.1 recense différents schémas à seuil exprimés en fonction des paramètres $p$, $n$, et $k$.

| Paramètres | Nom |
|---|---|
| 1-1-$n$ | Réplication |
| 1-$k$-$n$ | Dispersion d'information |
| $p$-$k$-$n$ | Schéma à rampe |
| $k$-$k$-$n$ | Partage de secret |

TABLE 2.1 – Différents schémas à seuil

Ici, un (1,1,$n$)-schéma à seuil est un schéma de réplication. Chaque fragment révèle des informations sur la donnée encodée ($p$=1). Un seul fragment est nécessaire pour reconstruire la donnée initiale ($k$=1). $n$ fragments sont disponibles pour reconstruire le document originel.

3. Ramp scheme

La dispersion d'informations permet quant à elle de distribuer $n$ fragments, chacun contenant des informations sur la donnée initiale. Il est nécessaire de retrouver $k$ parts de secret pour reconstruire le secret. Le partage de secret au sens de Shamir [Shamir, 1979] et Blakley [Blakley, 1979] ne doit pas permettre d'avoir de connaissances sur le secret si $k$ parts de secret ne sont pas réunies. La réplication et le partage de secret sont les deux bornes d'un schéma à rampe.

Un $(p, k, n)$-schéma à seuil n'autorise donc qu'un nombre de pertes concurrentes restreint $nb_{Pertes} < n - k$. La probabilité de défaillance est donc la suivante :

$$P(def|nb_{Pertes}) = \begin{cases} 0 & nb_{Pertes} < n - k \\ 1 & nb_{Pertes} \geq n - k \end{cases}$$

Ce type de fragmentation a été mis en place dans l'architecture Fragmentation, Redondance et Dissémination (FRD [4]) [Fabre *et al.*, 1994] où un mécanisme de redondance des fragments à été ajouté pour améliorer la disponibilité de l'information. PASIS [Wylie *et al.*, 2000] utilise également un schéma à seuil pour assurer la sécurité des données.

Un schéma de fragmentation à seuil est relativement proche d'un schéma de réplication des données. Ici les parts de secrets générées possèdent une taille du même ordre que le secret initial. L'espace de stockage nécessaire étant également peu satisfaisant, il convient de s'intéresser aux schémas de fragmentation construits à l'aide de codes correcteurs, plus économes en espace de stockage.

### 2.1.3 Codes correcteurs / Codes d'effacement

Le principe d'un code correcteur en bloc est de diviser une donnée en $k$ blocs, et d'ajouter $n$ blocs de même taille dépendant des autres de telle sorte que $k$ éléments parmi les $k + n$ fragments générés soient nécessaires à la reconstruction de la donnée originelle. Un tel schéma de fragmentation est appelé un **$(\mathbf{k}, \mathbf{n})$-code correcteur** en bloc.

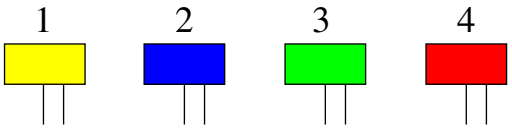
Dans la figure 2.3, une donnée est fragmentée en 4 éléments (figure 2.3(a)), puis à partir des 4 fragments générés, un 5$^{e}$ est construit (figure 2.3(b)). Si un des fragments vient à disparaître alors, il est possible de reconstruire l'information manquante à partir des fragments disponibles (figure 2.3(c) et figure 2.3(d)). Dans notre cas nous ne considérons que les erreurs consistant à reconstruire une donnée où des fragments d'information sont manquants. On parle alors de **code d'effacement**.

#### Propriétés

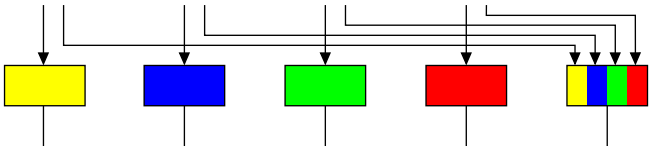
On peut définir le degré de réparation d'un $(k, n)$-code d'effacement par $d = k$. On appelle alors taux d'encodage $\tau$ :

$$\tau = \frac{k}{k + n}$$

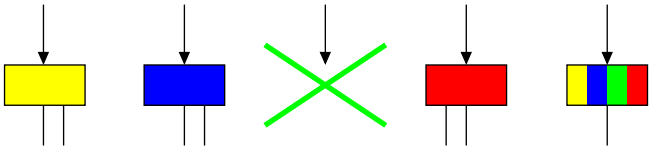
4. en anglais Fragmentation Redundancy and Scattering (FRS)



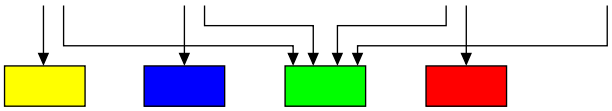

(a) Une donnée est découpée en 4 fragments



(b) Un fragment de redondance est construit



(c) Le fragment vert (3) est détruit ou inaccessible



(d) Le fragment vert (3) est reconstruit à partir des autres fragments

FIGURE 2.3 – Principe de fonctionnement d'un (4,1)-code correcteur

La probabilité de défaillance dans le réseau est définie de la manière suivante :

$$P(def|nb_{Pertes}) = \begin{cases} 0 & nb_{Pertes} < n \\ 1 & nb_{Pertes} \geq n \end{cases}$$

Il est facile de voir les avantages en terme d'espace de stockage des codes d'effacement. Reprenons notre exemple précédent portant sur un fichier $f$ de 100Mo. Si nous appliquons un $(10, 10)$-code d'effacement alors $f$ ne sera plus accessible à partir de 11 pertes de fragments concurrentes. $f$ va donc être découpée en 10 fragments de 10Mo, auxquels sont ajoutés 10 fragments également de 10Mo. Le taux d'encodage $\tau$ d'un tel code est donc $\tau = \frac{10}{10+10} = \frac{1}{2}$ et le facteur de redondance est de :

$$r = \frac{100 + (10 \times 10)}{100} = \frac{200}{100} = 2 = \frac{1}{\tau}$$

Le coût de stockage est donc multiplié par 2, et il est alors nécessaire d'occuper 100Mo d'espace supplémentaire dans le réseau. Le degré de réparation $d$ est de 10. Il est donc nécessaire de prendre 10 fragments parmi les 20 générés pour retrouver le fichier $F$.

Ici, utiliser un code d'effacement revient à diminuer par un facteur de cinq l'espace de stockage requis en comparaison avec la réplication simple des données. Dans le premier exemple, un

gigaoctet d'espace est requis pour assurer la pérennité des données alors que dans le cas des codes d'effacement seulement cent mégaoctets supplémentaires sont nécessaires.

Bien que le critère d'espace occupé dans le réseau soit primordial, il est crucial dans un système de stockage de données réparti que le degré de réparation soit le plus faible possible. Le degré de réparation d'un code correcteur étant supérieur à celui de la réplication des données beaucoup d'applications préfèrent dupliquer l'information [Clarke *et al.*, 2000, Kubiatowicz *et al.*, 2000, Adya *et al.*, 2002]. Toutefois il existe une catégorie de code correcteur plus intéressante en terme de réparation de données. Ce sont les **codes linéaires**. Dans un tel code, la dépendance entre blocs générés est obtenue par combinaison linéaire.

Soit $F$ l'ensemble des fragments initiaux $f_i$, $i = 1, \ldots, k$. Pour calculer chaque bloc encodé $b_i$, une fonction $G_i$, interprétée comme une combinaison linéaire des fragments initiaux $f_i$ est appliquée. Nous avons donc :

$$b_i = G_i(f_1, f_2, ..., f_k) = \begin{cases} f_i & i \leq k \\ \sum_{j=1}^{k} f_j g_{i,j} & k < i \leq k+n, g_{i,j} \in GF(2^q) \end{cases}$$

$GF(2^q)$ désigne un corps de Galois composé de $2^q$ éléments. Un corps de Galois est un ensemble d'éléments fermés sur eux-mêmes, l'addition et la multiplication de deux éléments du corps donnant toujours un élément du corps. $GF(2^q)$ est composé des éléments de $GF(2)$ (0 et 1) et contient des multiples des éléments de $GF(2)$. A l'aide des matrices suivantes, composées d'éléments de GF($2^q$) :

- $F_{k,1}$, vecteur de fragments initiaux,
- $B_{k+n,1}$, vecteurs de blocs encodés,
- $E_{k+n,k}$, matrice d'encodage.

nous pouvons donc donner une expression d'un code linéaire dans GF($2^q$) :

$$B = E \times F = \begin{pmatrix} e_{1,1} & e_{1,2} & \cdots & e_{1,k} \\ e_{2,1} & e_{2,2} & \cdots & e_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ e_{k,1} & e_{k,2} & \cdots & e_{k,k} \\ \vdots & \vdots & \ddots & \vdots \\ e_{k+n,1} & e_{k+n,2} & \cdots & e_{k+n,k} \end{pmatrix} \times \begin{pmatrix} f_1 \\ \vdots \\ f_k \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_{k+n} \end{pmatrix} \tag{2.1}$$

### Code de Reed-Solomon

On peut fixer la matrice d'encodage $E$ de plusieurs manières. En prenant $E = \begin{pmatrix} I \\ C \end{pmatrix}$ avec $I_{k,k}$ la matrice identité et $C_{n,k}$ une matrice de Vandermonde ($c_{i,j} = j^{i-1}$), nous avons à faire à un **code de Reed-Solomon** [Reed et Solomon, 1960, Plank, 1997]. L'utilisation de la matrice identité $I$ dans la matrice d'encodage $E$, implique que les $k$ premiers fragments encodés seront identiques aux originaux. Ainsi lors d'une phase de reconstruction il est nécessaire de reconstruire les fragments manquants $f_i$ pour $i \in [1, k]$ avant de pouvoir reconstruire un fichier.

L'inconvénient d'un code de Reed-Solomon réside dans le fait que les $k$ premiers fragments générés sont identiques aux originaux. Dès lors la confidentialité des fragments générés n'est pas complètement assurée.

Pour l'étape de reconstruction d'un tel code, supposons que exactement $k$ parts de fichiers soient perdues. On a la relation suivante :

$$E' \times F = B' \tag{2.2}$$

où la matrice $E'$ est composée de lignes de $E$, et le vecteur $B'$ d'éléments de $B$. Etant donné que $C$ est une matrice de Vandermonde, chaque sous-ensemble de lignes de $E$ est linéairement indépendant. Dès lors la matrice $E'$ est non-singulière et donc par conséquent inversible. En utilisant une élimination gaussienne dans 2.2, on peut retrouver les éléments de $F$ en faisant :

$$E'^{-1} \times B' = F \tag{2.3}$$

Dans le cas où moins de $k$ éléments sont perdus ou inaccessibles, la matrice $E^{'}$ formée de lignes de $E$ n'est plus carrée. Il suffit alors de choisir un sous-ensemble de $n$ lignes dans la matrice $E'$, pour obtenir une matrice carrée $n \times n$ notée $E^{''}$. Dès lors $E^{''}$ est inversible et par élimination gaussienne on peut retrouver les éléments de $F$. Un code de Reed-Solomon supporte donc jusqu'à $k$ pertes concurrentes.

Parmi les applications pair-à-pair de partage de fichiers, utilisant les codes linéaires, nous pouvons citer Glacier [Haeberlen *et al.*, 2005]. Dans [Haeberlen *et al.*, 2005] les auteurs utilisent une variante d'un code de Reed-Solomon construite à l'aide de matrices de Cauchy [Blömer *et al.*, 1995].

### Code linéaire aléatoire

Si la matrice $E$ est construite avec des coefficients aléatoires, nous mettons alors en évidence une seconde catégorie de codes linéaires appelés **codes linéaires aléatoires** [Ahlswede *et al.*, 2002, Li et Yeung, 2003, Acedański *et al.*, 2005, Dimakis *et al.*, 2006]. Ici un fichier est découpé en $k$ fragments de même taille, puis $(k+n)$ fragments sont générés en appliquant une combinaison linéaire aléatoire entre les fragments. De cette manière chaque fragment généré est une combinaison linéaire de plusieurs fragments. Ce sont ces $(k+n)$ fragments combinés aléatoirement qui sont stockés dans le réseau. Les coefficients de la matrice d'encodage sont stockés avec les fragments. Ici les fragments étant une combinaison linéaire de fragments, la confidentialité des données est préservée.

Comme pour un code de Reed-Solomon, l'étape de reconstruction repose sur l'inversion de matrice. Ici nous voulons reconstituer une donnée à partir de $k$ fragments encodés. Soit $B'$ le vecteur construit à partir de $k$ éléments encodés récupérés au travers d'un réseau. On peut construire une matrice $E'$ à partir des lignes de $E$ correspondantes aux éléments de $B'$. La construction du vecteur d'éléments encodé $B$ 2.1 nous permet d'avoir la relation suivante :

$$E' \times F = B \tag{2.4}$$

En inversant la matrice $E'$ et en multipliant $E'^{-1}$ avec le vecteur d'éléments récupérés $B'$, nous reconstruisons le vecteur de fragments initiaux $F$ :

$$E^{'-1} \times B' = F \tag{2.5}$$

Dans le cas des codes linéaires aléatoires, il existe des cas où la matrice $E'$ ne peut être inversée. Cependant il est possible de remédier à cette situation en choisissant un corps de Galois suffisamment grand [Ahlswede *et al.*, 2002, Li et Yeung, 2003, Acedański *et al.*, 2005]. Dès lors la probabilité d'inversion de la matrice $E$ peut être fixée arbitrairement proche de 1. Dans beaucoup de cas d'étude, un corps de Gallois de taille $2^{16}$ est suffisant [Duminuco et Biersack, 2009, Acedański *et al.*, 2005].

A la différence d'un code de Reed-Solomon, l'avantage d'un code linéaire aléatoire réside dans le fait qu'il n'est pas nécessaire de calculer le vecteur $B$ pour reconstruire un bloc de données perdu. Ici on peut seulement récupérer $k$ blocs encodés et les combiner entre eux. Cet apport est rendu possible par l'utilisation de coefficients aléatoires dans la matrice $E$. Le degré de réparation est quand à lui toujours de $d = k$.

#### Autres codes

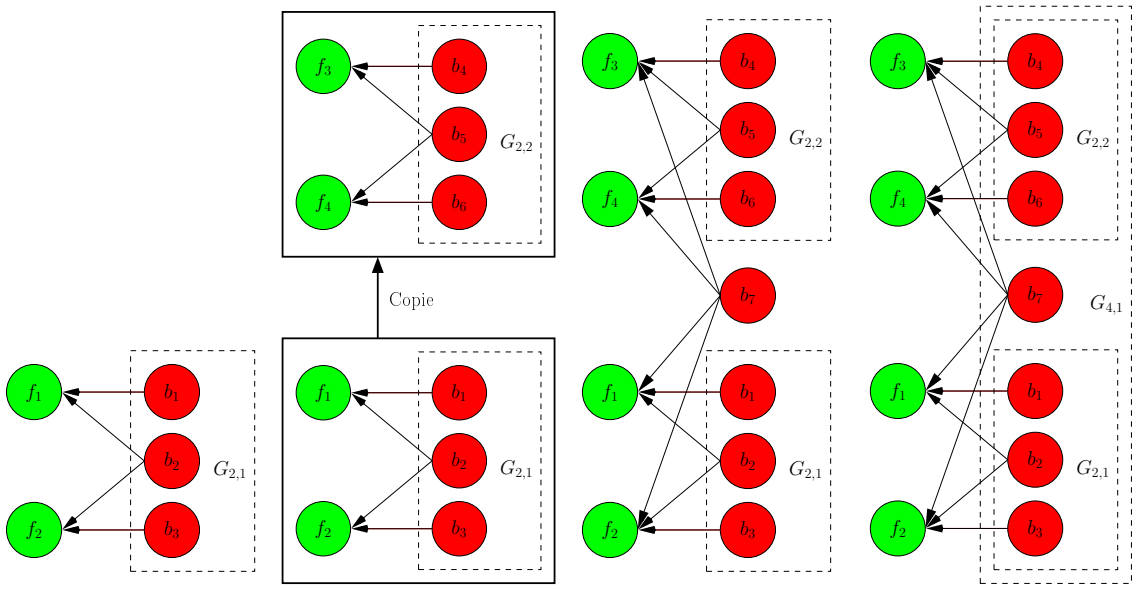
Il existe dans la littérature d'autres catégories de codes. Parmi ceux-ci on peut citer les **codes régénérants**. Ils sont considérés dans [Dimakis *et al.*, 2007, Duminuco et Biersack, 2009] comme étant une généralisation des codes d'effacement. Dans un code régénérant la taille des fragments générés est dépendante d'un indice $i$ que l'on fait varier en fonction de $k$, et d'un degré de réparation $d$ donné. Un $(k, n, d, i)$-code régénérant prend également en compte la quantité d'information transmise dans le cas d'une réparation. Une réparation consiste à reconstruire un fragment perdu en utilisant les fragments disponibles dans le réseau.

Dans [Dimakis *et al.*, 2007] deux catégories de codes peuvent être distinguées. Un premier code régénérant appelé **Minimum Storage Regenerating Codes (MSR)** permet de minimiser l'espace de stockage dans le réseau. Dans ce cas la valeur de $i$ est fixée à 0. Un second cas de code régénérant appelé **Minimum Bandwidth Regenerating codes (MBR)** permet quant à lui de minimiser la bande passante lors d'une réparation. Ici la valeur de $i$ est fixée à $(k - 1)$.

Les auteurs dans [Duminuco et Biersack, 2009] ont montré que le coût de calcul des codes régénérants est trop élevé. C'est une des raisons pour laquelle il y a très peu d'implémentations de tels codes. C'est dans cet optique que les **codes hiérarchiques** ont été proposés [Duminuco et Biersack, 2008, Duminuco et Biersack, 2010]. Les codes hiérarchiques sont des codes permettant un juste milieu entre la réplication et les codes d'effacement en terme de coût de reconstruction tout en proposant un coût de calcul moindre.

Le principe de construction d'un code hiérarchique repose tout d'abord sur la construction d'un premier code aléatoire. Les fragments ainsi créés forment un groupe. Un code hiérarchique

consiste à dupliquer ce groupe de fragments plusieurs fois en ajoutant, une fois les duplications de groupe effectuées, des fragments supplémentaires. Ces fragments supplémentaires sont issus d'une combinaison linéaire aléatoire des fragments originaux. L'ensemble des groupes dupliqués et des fragments ajoutés forment un code hiérarchique. Il est possible de répéter cette opération de duplication et d'ajout de fragments plusieurs fois pour ajouter des degrés dans la hiérarchie du code. La figure 2.4 décrie la construction d'un $(4, 3)$-code hiérarchique.




(a) Construction d'un $(2, 1)$-code hiérarchique. (b) Duplication de $G_{2,1}$. (c) Ajout d'un bloc $b_7$. (d) Construction d'un $(4, 3)$-code hiérarchique.

FIGURE 2.4 – Construction d'un $(4, 3)$-code hiérarchique

La figure 2.4(a) montre la construction d'un $(2, 1)$-code linéaire aléatoire. Un groupe $G_{2,1}$ de 3 blocs $b_1, b_2, b_3$ combinaisons linéaires aléatoire de 2 fragments initiaux $f_1, f_2$ est alors formé. Dans la figure 2.4(b), le groupe $G_{2,1}$ est dupliqué pour former 2 groupes. A ces 2 groupes, un fragment combinaison linéaire aléatoire des fragments initiaux est ajouté (figure 2.4(c)). Le groupe $G_{4,1}$ constitué des deux groupes $G_{2,1}$ et $G_{2,2}$, et du bloc $b_7$ est un $(4, 3)$-code hiérarchique (figure 2.4(d)).

Les résultats dans [Duminuco et Biersack, 2010] montrent qu'un code hiérarchique nécessite un nombre de réparations plus important qu'un code de type Reed-Solomon. Cependant le nombre de blocs transmis pour une réparation est nettement moins élevée. Il en résulte dès lors un gain en terme de bande passante dans le réseau. Ici une réparation consiste à reconstruire un bloc si celui-ci est toujours indisponible après un certain temps.

### Conclusion

Il existe de nombreuses méthodes pour assurer la pérennité de l'information dans le temps. Toutes ne sont pas équivalentes et certaines n'assurent pas par exemple la confidentialité des

données (cas de la réplication simple). Pour construire une application de stockage de donnée sécurisée il convient de choisir un schéma offrant le meilleur rapport entre la confidentialité des données, l'espace occupé, et le coût global nécessaire aux opérations de réparation et de reconstruction. Dans cette optique les codes hiérarchiques constituent une réponse prometteuse. A l'heure actuelle et pour nos besoins, un code d'effacement de type Reed-Solomon nous semble adapté.

## 2.2 Placement de l'information

Le découpage des données est une étape fondementale pour assurer la disponibilité de l'information dans un système de stockage de données distribué. Une fois un fichier coupé en plusieurs morceaux, il convient de placer sur plusieurs pairs les fragments générés. Il est évident que la stratégie de placement de l'information a une importance non-négligeable sur les performances globales d'une architecture distribuée [Douceur et Wattenhofer, 2001]. Cette étape est donc cruciale et étroitement liée à l'étape de fragmentation.

Dans le cadre d'une application répartie, le placement de l'information doit respecter plusieurs critères essentiels :

- **Performance :** Le critère de performance porte principalement sur l'accessibilité des données et les opérations de maintenance associées. Le placement des fragments doit être réalisé de telle manière que la recherche et la reconstruction de données doit se faire le plus efficacement et rapidement possible. La réparation de fragments inaccessibles du à des départs, temporaires ou non, doit également être efficace.
- **Répartition de charge :** Il convient de répartir la charge équitablement entre l'ensemble des pairs dans le réseau.
- **Robustesse :** Le placement de l'information choisi doit avoir un impact négligeable sur les performances globale du système, sur l'équilibrage de charge, et doit donc proposer un coût de fonctionnement minimum.

On peut distinguer plusieurs méthodes de dissémination des fragments. La première consiste à disséminer aléatoirement dans un réseau l'ensemble des fragments. La seconde consiste à utiliser les propriétés de routage du réseau logique sous-jacent pour placer de manière structurée l'information. Nous allons voir maintenant les différentes méthodes au travers d'applications de stockage de fichiers construites au dessus d'un overlay.

### 2.2.1 Placement aléatoire

Un premier placement décentralisé de l'information consiste à disséminer les fragments générés suivant une distribution aléatoire sur l'ensemble des pairs du réseau. Ce placement aléatoire est décrit dans la figure 2.5.

Ici un pair du réseau (le pair de couleur rouge), distribue aléatoirement dans le réseau les fragments d'information qu'il a généré sur un ensemble de pair de couleur verte. Il existe

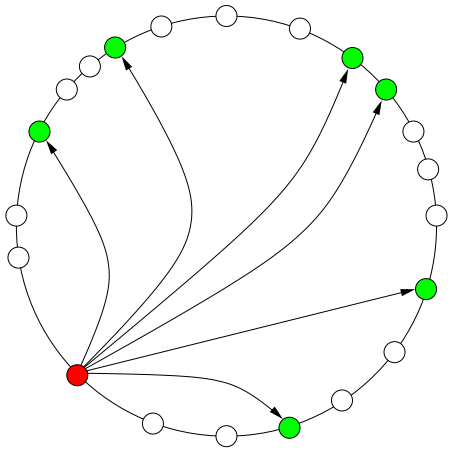


FIGURE 2.5 – Placement aléatoire de l'information

principalement deux méthodes pour parvenir à un tel schéma de placement de l'information. Une première consiste à avoir une vue globale d'un réseau puis de disséminer les fragments dessus [van Renesse, 2004, Ghemawat *et al.*, 2003, Fabre *et al.*, 1994]. Cette connaissance globale peut se faire soit au travers d'un serveur indépendant du réseau observé, soit sur chaque machine du réseau considéré. La seconde méthode consiste à disperser l'information sans connaissance a priori sur le réseau et ses propriétés. C'est notamment le cas dans [Rhea *et al.*, 2003].

#### Fonctionnement

Dans [van Renesse, 2004], la construction de la fonction pseudo-aléatoire repose sur une fonction de hachage prenant en paramètres un identifiant de fragment et un identifiant de pair du réseau. Soit $D$ une donnée découpée en $d_1, \ldots, d_k$ morceaux, et $p_1, \ldots, p_q$ les $q$ pairs d'un réseau. Nous avons donc :

$$rang_i = HASH(D, p_i), i \in [1, q]$$

A chaque pair $p$ du réseau est donc associée une valeur *rang* retournée par la fonction de hachage. Les $k$ fragments d'information issus de $D$ sont placés sur les $k$ pairs possédant le plus petit rang. Etant donné que la fonction de hachage retourne des résultats très différents pour chaque bloc, les nœuds devant accueillir un fragment d'information sont distribués pseudo-aléatoirement suivant l'espace de nommage du réseau.

Dans GFS [Ghemawat *et al.*, 2003] chaque fichier est découpé en chunks, et pour assurer la disponibilité de l'information, plusieurs copies de ces morceaux sont stockées. GFS repose sur un réseau constitué de plusieurs grappes de machines (clusters) où sont stockés les chunks.

A la différence de la méthode précédente où chaque membre connaît l'ensemble du réseau, ici cette connaissance est réduite à un seul élément chargé d'assurer le placement des fragments de données. Cette entité appelée "master" et présente dans chaque cluster de machines, est responsable du placement des chunks. Ce placement se fait suivant des critères d'espace disque

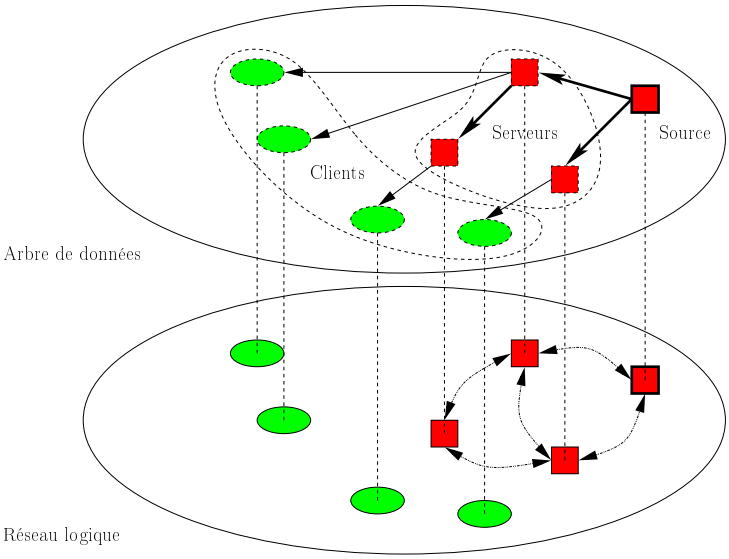



FIGURE 2.6 – Placement de l'information utilisant un arbre

et de bande passante. Le placement doit également répartir les chunks de telle sorte qu'une panne machine ou réseau ne puisse empêcher l'accès à une donnée.

Dans FRD [Deswarte *et al.*, 1991, Fabre *et al.*, 1994], le placement repose sur la dissémination aléatoire de fragments dans un réseau de serveurs de stockage. Une fois les fragments générés, il convient de les répartir sur ces serveurs. Cette répartition se fait suivant un algorithme pseudo-aléatoire prenant en compte l'espace occupé sur les serveurs. Un second paramètre de l'algorithme consiste à éviter de placer sur un même serveur plusieurs fragments issus d'une même source. De cette manière, les auteurs se prémunissent de comportements malveillants cherchant à prendre le contrôle d'un serveur pour obtenir des informations sur une donnée à partir de plusieurs fragments. L'objectif de la dissemination est donc d'offrir un stockage "sécurisé" tout en répartissant efficacement la charge du réseau en terme d'espace occupé. Ainsi pour un attaquant il est plus difficile de cibler des parties d'un réseau en fonction de l'espace occupé.

Contrairement aux précédentes applications, Oceanstore [Rhea *et al.*, 2003] n'utilise pas de connaissances à priori du réseau pour répartir l'information. Ici la dissémination repose sur l'utilisation d'un arbre dont la racine est la réplique primaire d'un objet. Les répliques secondaires désignent quant à elles les nœuds de l'arbre de dissémination. Les objectifs de cette méthode de dissémination est de pouvoir fournir un placement de l'information qui soit non statique et non fourni par une entité ayant une connaissance globale du réseau tout en prenant en compte les contraintes de capacités des serveurs de stockage et la bande passante utilisée pour stocker les répliques d'information.

La figure 2.6 montre une telle structure pour le placement de l'information. La racine de l'arbre est la source d'une donnée. Des répliques secondaires sont stockées sur des serveurs reposant sur un réseau logique de type Tapestry (cf section 1.3.3). Les clients, feuilles de l'arbre, hébergent quant à eux des caches de répliques.

L'algorithme présent dans [Chen *et al.*, 2002] permet de construire un tel arbre de répliques dans un réseau de type Tapestry. Prenons le cas où un nouveau client $c$ veut obtenir un objet $o$ dans le réseau. Il doit donc rejoindre en tant que feuille un arbre existant pour l'objet $o$. Le but est d'optimiser le choix d'un parent pour $c$ dans l'arbre.

Pour rentrer dans l'arbre, $c$ contacte un serveur $s$. Le choix d'un parent doit se faire parmi $s$, le parent de $s$, les frères de $s$, et les fils de $s$. Parmi ces possibilités, $c$ choisit celui avec la charge courante la moins élevée. Toutefois si aucun des choix possibles ne remplit les conditions de charge et de latence, alors $s$ essaie de placer une réplique de l'objet $o$ sur un serveur $s'$ présent sur le chemin entre $s$ et $c$. Ce serveur $s'$ doit satisfaire les conditions de charges, et de distance sur l'overlay. $s'$ devient alors le parent de $c$ et de $s$. Cet algorithme glouton permet de répartir équitablement la charge entre les serveurs et de réduire le nombre de répliques d'un objet dans le réseau tout en respectant les contraintes de charge et de latence.

#### Avantages / Inconvénients

Chacune des méthodes précédentes aboutissant à un placement aléatoire de l'information prennent naturellement en compte une répartition efficace du stockage dans le réseau. C'est une des forces d'un tel schéma de dissémination, et c'est ce qui ressort des différentes applications utilisant une telle méthode.

Dans [Qiao *et al.*, 2005] et [Giroire *et al.*, 2009] c'est l'aspect réparation de donnée qui est mis en avant. Une telle approche favorise grandement la réparation de fragments perdus. Cela se traduit par une récupération en parallèle, à différents endroits du réseau, des fragments nécessaires à la reconstruction de celui perdu. Ici nous n'avons pas l'effet goulot d'étranglement présent dans l'approche chaîné (section 2.2.2).

Les auteurs de [van Renesse, 2004] proposent même un mécanisme de réparation plus rapide et efficace en associant à chaque pair du réseau une mesure de disponibilité. Cette mesure consiste à effectuer un ping à intervalle régulier sur les machines du réseau. Le but d'une telle démarche est de détecter les fragments indisponibles plus rapidement. Cela offre également un placement de l'information plus "sûr" en plaçant les fragments sur des pairs ayant, par exemple, une disponibilité supérieure à 90%.

Le fait qu'une ou plusieurs entités possèdent la connaissance globale du réseau, ne semble pas être un choix pertinent. Dans le cas où cette connaissance est réduite à un serveur, c'est le concept même d'application distribuée qui est remis en cause. Une attaque ciblée sur un serveur rend le service fourni indisponible. Pour pallier à ce problème, une pratique courante consiste à posséder des serveurs de secours dans le cas de pannes ou d'attaques. Dans [Ghemawat *et al.*, 2003], plusieurs serveurs sont responsables du placement de l'information dans chaque cluster. Dans [Fabre *et al.*, 1994], ce n'est pas un serveur qui est responsable de la dissémination mais un ensemble de machines. Toutefois cela ne résout pas le problème de la centralisation du service, et cela ne fait que déporter le problème à une échelle moindre.

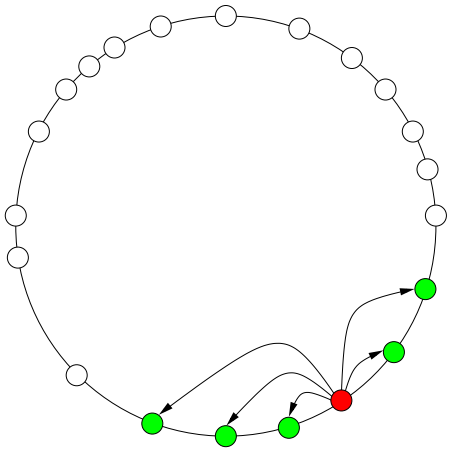


FIGURE 2.7 – Placement de l'information chaîné

Le cas d'Oceanstore est différent. Ici aucune entité centrale n'est responsable du placement des données. Celles-ci sont placées suivant des critères de charge réseau sur un ensemble de serveurs. Ici il n'est pas pris en compte le cas du rééquilibrage de charge. Il serait plus confortable ici de disposer d'un mécanisme permettant de replacer dynamiquement les informations au cours du temps similaire à celui présent dans GFS. Dans GFS le master a pour rôle d'examiner dans le temps la répartition des chunks. Il peut ainsi procéder à une nouvelle répartition des chunks en observant leur placement dans le cluster. Le nouveau placement des données se fait suivant les même critères que précédemment à savoir optimiser l'espace disque, et la charge réseau.

### 2.2.2 Placement chaîné

Une seconde méthode de placement de l'information consiste à s'appuyer sur les propriétés de localité dans un réseau. De manière générale, les fragments d'information sont placés autour d'un nœud désigné responsable de l'information. La figure 2.7 décrit un tel placement de l'information.

Ici un pair rouge est chargé de stocker un ensemble de données, et de les distribuer sur les nœuds situés autour de lui (pairs de couleur verte). Cette méthode de dissémination repose sur l'utilisation d'un réseau logique efficace. De manière générale, les fragments sont stockés sur les pairs consécutifs les plus proches "logiquement" et non physiquement.

Nous allons maintenant étudier plus en détail ce placement de l'information au travers de trois applications, PAST [Rowstron et Druschel, 2001b], CFS [Dabek *et al.*, 2001], et Glacier [Haeberlen *et al.*, 2005].

#### Fonctionnement

Dans une méthode de placement chaîné, $k$ morceaux d'information doivent être placés localement autour d'un pair désigné responsable. Une fois les $k$ morceaux d'information obtenus, la première étape consiste donc à désigner un pair responsable.

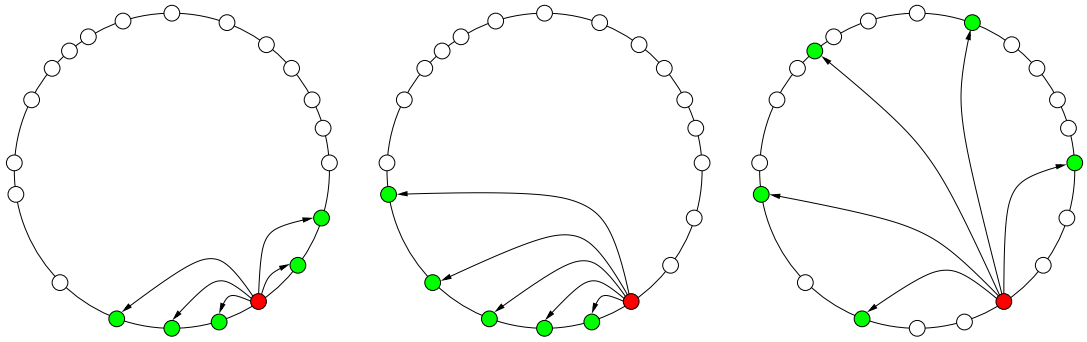
Dans PAST [Rowstron et Druschel, 2001b], une application de stockage de fichiers reposant sur le réseau logique Pastry [Rowstron et Druschel, 2001a], les identifiants de fichiers et de nœuds partagent le même espace de nommage. Après avoir généré l'identifiant $ID_d$ d'une donnée $d$ par l'utilisation d'une fonction de hachage, un message est envoyé à destination de cet identifiant. Quand le pair $p$ le plus proche de $ID_d$ est trouvé, il est désigné responsable de la donnée $d$. Ici la distance utilisée est similaire à celle de Kademlia à savoir le ou exclusif (section 1.3.4).

Dans Cooperative File System (CFS) [Dabek *et al.*, 2001] le choix du pair responsable d'une donnée $d$ repose sur Chord [Stoica *et al.*, 2003] (cf section 1.3.2). Un identifiant $ID_d$ est généré pour $d$, puis un message est envoyé à cet identifiant. Le pair responsable de cet identifiant, soit le premier pair successeur de $ID_d$ sur l'anneau, est désigné responsable pour $d$.

Un mécanisme similaire est également présent dans Glacier [Haeberlen *et al.*, 2005]. Ici, le placement des données repose sur une méthode permettant de stocker les $k$ répliques sur $k$ nœuds équidistants. Ainsi une fois un pair responsable désigné, le placement assure que $k-1$ répliques se situe sur des nœuds équidistants du pair responsable.

Ce qui différencie PAST de CFS est la méthode de placement des $k$ répliques de $d$. Dans PAST ce sont les $k-1$ pairs les plus proches du nœud responsable qui vont héberger les répliques de $d$. Dans CFS ce sont les $k-1$ pairs successeur du nœud responsable de $d$ dans l'anneau induit par le réseau logique Chord qui sont choisis.

La figure 2.8 montre les 3 méthodes de placements proposées par PAST, CFS et Glacier appliquées à un réseau structuré en anneau [5].



(a) PAST (b) CFS (c) Glacier

FIGURE 2.8 – Placement chaîné des répliques d'un fichier $f$

### Avantages/Inconvénients

Une telle méthode de dissémination semble être contradictoire avec une répartition efficace des données. Toutefois, il existe dans PAST deux politiques pour parvenir à une distribution équilibrée des informations. La première concerne le placement des $k-1$ répliques de $d$. Un pair $p_A$ peut refuser d'héberger une réplique de $d$. Ce refus peut-être motivé par un espace de

5. Cette représentation a été adoptée par soucis de clarté.

stockage restreint comparé à celui d'un de ses voisins. Auquel cas il va décider de contacter un nœud $p_B$ parmi ses voisins ne possédant pas de répliques de $d$, et ayant un espace de stockage "meilleur" que le sien, et lui demander de stocker une réplique. $p_A$ va alors conserver un lien vers $p_B$ associé à la réplique.

La seconde politique concerne l'insertion de la donnée $d$. La requête pour placer $d$ dans le réseau peut échouer. Cet échec se produit dans le cas où le pair responsable ne possède pas suffisamment d'espace pour stocker $d$ où quand il n'y a pas assez de pairs pour stocker les $k-1$ répliques de $d$. Il est alors nécessaire de reformuler une demande d'insertion de $d$ en modifiant les paramètres de la fonction de hachage dans le but de trouver un autre nœud responsable pour $d$. Si au bout de 3 essai, $d$ n'est toujours pas stocké dans le réseau, il convient alors de modifier les opérations sur $d$, à savoir fragmenter $d$ ou modifier le nombre de répliques.

Les résultats présents dans [Rowstron et Druschel, 2001b] montrent une utilisation de la quasi totalité des pairs du réseau (98%), tout en conservant un taux d'échec d'insertion de donnée faible (inférieur à 5%). Ces résultats confirment une bonne répartition du stockage de données sur l'ensemble des machines du réseau.

Le fait de désigner un seul membre du réseau comme étant responsable d'une donnée induit des problèmes de répartition de charge, notamment dans leur accès. Dans le cas d'un fichier populaire, il est évident qu'un seul nœud ne peut pas traiter l'ensemble des requêtes. Pour répartir la charge réseau plus efficacement, les auteurs de CFS utilisent un mécanisme de cache. Ainsi quand un client recherche une donnée $d$, une requête est envoyée. Cette dernière est routée dans le réseau jusqu'au nœud responsable de $d$. Une fois trouvé, le client place alors une copie de $d$ sur les pairs parcourus lors du routage de la recherche. Pour ne pas surcharger les pairs de copies de blocs, chaque nœud maintient un cache de taille fixée, dans lequel les copies sont remplacées suivant une politique LRU (Last Recently Used). Ce mécanisme de cache est différent et complémentaire de celui de la réplication des données chargé d'assurer la pérennité des informations.

Enfin tel placement influence la fiabilité des données dans le temps. Dans le cas où un des $k$ pairs n'est plus disponible, il est nécessaire de reconstruire les données manquantes. Ce processus doit être au maximum parallélisé pour pouvoir prendre le moins de temps possible. Ce constat est confirmé dans [Qiao *et al.*, 2005] et [Giroire *et al.*, 2009] où le temps pour reconstruire une donnée manquante est beaucoup plus grand que pour une approche aléatoire. Cependant l'un des avantages d'une telle approche réside dans le nombre limité de placements possibles pour un ensemble de $k$ données. Ce faible nombre permet d'augmenter la résistance aux pannes simultanées dans le réseau.

### Conclusion

Il est difficile de choisir une méthode de placement. Chacune possède ses avantages et ses inconvénients. C'est pour cela que différentes approches ont été proposées pour combiner les deux modèles. Dans Kinésis [MacCormick *et al.*, 2009] par exemple il n'y a pas un mais $r$ réseaux

logiques, chacun possédant une fonction permettant d'assigner à une réplique un unique pair. Ainsi si une donnée possède $k < r$ répliques alors les $k$ répliques seront placés sur $k$ réseaux logiques. En fonction du choix de $k$ et $r$ les résultats obtenus montrent une bonne répartition de charge tout en permettant une réparation en parallèle efficace. Toutefois, l'augmentation du nombre de réseau logique semble être un handicap pour la prise en compte d'une telle dissémination dans un réseau.

Une autre méthode reposant sur l'utilisation d'un Processus Décisionnel de Markov (PDM) [6] [Beynier et Mouaddib, 2009] à également été proposée. Ici chaque serveur du réseau exécute un MDP permettant de décider si il doit stocker un fichier ou non. Les fichiers sont stockés suivant des critères de pertinence (intérêt de stocker un fichier en fonction de ceux déjà présents), de charge réseau, et de ressources restantes (espace disque disponible). Bien que les résultats présentés soient prometteurs, le trop petit nombre de machines misent en œuvre contraste les résultats. Pour évaluer plus précisément un tel placement de l'information, il faudrait envisager son déploiement à une plus grande échelle.

## 2.3 Recherche de l'information

La dernière étape dans le cycle de vie de l'information consiste à retrouver l'information une fois sa dissémination effectuée. Cette étape est fortement dépendante de la structure du réseau utilisé. Une comparaison des différents modèles de réseaux pair-à-pair permet déjà d'observer différentes techniques de recherche. Nous allons maintenant voir ces différentes méthodes de recherche de données en fonction des propriétés du réseau sous-jacent.

### 2.3.1 Modèle centralisé, semi-centralisé, et hybride

Dans Napster, application construite sur un modèle centralisé, la recherche de données est un mécanisme assez simple. Celui-ci repose sur l'utilisation de serveurs répertoriant les objets mis à disposition. Quand un utilisateur se connecte, le serveur stocke dans un index les fichiers partagés et les associe à l'adresse IP de l'utilisateur. De cette manière quand un utilisateur recherche un fichier, le serveur renvoie les adresses IP où il peut trouver la ressource.

C'est le même principe de fonctionnement qui est observé dans les architectures semi-centralisées. Dans eDonkey, la première étape pour une feuille souhaitant récupérer une donnée, est de s'adresser au serveur auquel elle est connectée. La recherche effectuée par le serveur porte alors sur les fichiers partagés par les utilisateurs présents dans le réseau. De la même manière que pour Napster, le mécanisme de recherche est économique et ne prend pas plus de deux sauts pour être satisfait.

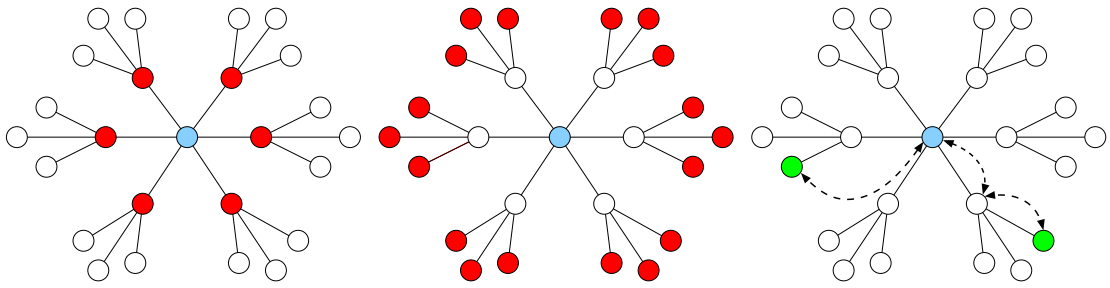
Enfin, dans les applications dites hybrides telles que KaZaA, une recherche effectuée par une feuille est envoyée à un ou plusieurs super-nœuds. Cette recherche est alors propagée à l'ensemble

6. Markov Decision Process (MDP)

des pairs connectés aux super-nœuds. Cette méthode de recherche permet de diminuer le nombre de messages vers les feuilles tout en augmentant la charge des super-nœuds.

### 2.3.2 Flooding et expanding ring

Dans le cas d'une décentralisation totale du système, il est nécessaire d'avoir une méthode de recherche efficace. Gnutella possède un mécanisme de recherche par flooding ou inondation. Ainsi un nœud va diffuser une requête de recherche bornée à l'ensemble de ses voisins. Une fois que les voisins ont reçu la requête de recherche, ils la diffusent à leur tour à leur voisins en décrémentant le nombre de sauts (TTL[7]) restant. Dans Gnutella [Gnutella, 2000] la borne maximum est fixée à 7 sauts. La figure 2.9 décrit une telle méthode. Le pair bleu lance le mécanisme de recherche. Il envoie à l'ensemble de ses voisins une requête (figure 2.9(a)). Si aucun résultat n'est trouvé, la requête est propagée aux voisins 2.9(b). En fonction de l'environnement (présence de pare-feu ou non), deux méthodes de retour à l'expéditeur sont possibles. Soit l'expéditeur est prévenu directement, soit le message remonte le long du chemin parcouru (figure 2.9(c)).



(a) TTL = 1 (b) TTL = 2 (c) Envoi de la réponse

FIGURE 2.9 – Mécanisme de recherche dans Gnutella

Cette méthode de recherche bien qu'efficace dans le pourcentage de réussite, est très coûteuse en terme de messages envoyés et de nœuds parcourus [Lv *et al.*, 2002, Zeinalipour-Yazti *et al.*, 2004]. Une première solution consiste à régénérer une requête en augmentant le TTL à chaque recherche infructueuse. Cette méthode est appelée "expanding ring". Ainsi un nœud envoie une requête avec un TTL = 1 à l'ensemble de ses voisins. Etant donné que le TTL est fixé à 1, la requête n'est pas propagée dans le réseau. Si la réponse est négative, alors le pair initiateur régénère une requête avec un TTL fixé à 2. Les recherches sont effectuées jusqu'à l'obtention d'une réponse positive. Cette méthode permet d'obtenir des résultats en terme de succès similaire à l'inondation tout en diminuant le nombre de messages dans le réseau [Lv *et al.*, 2002].

Une autre solution pour améliorer la méthode d'inondation consiste à propager les requêtes à un sous-ensemble de voisins. Ce sous-ensemble peut être choisi suivant plusieurs paramètres. Par exemple dans [Yang et Garcia-Molina, 2002] c'est un critère de résultats dans le passé qui est choisi. Les pairs ayant fourni le plus de réponses positives sont choisis en priorité. Dans

7. Time To Live

[Zeinalipour-Yazti *et al.*, 2004] les auteurs se sont intéressés aux contenu même des nœuds pouvant satisfaire la requête de recherche. Dans tout les cas les résultats montrent une baisse importante du nombre de requêtes et un pourcentage de réussite proche de 90%.

### 2.3.3 Marches aléatoires

Pour faire face aux problèmes de la réplication des messages des méthodes précédentes, une approche consiste à utiliser des marches aléatoires, ou chaîne de Markov. Un marcheur aléatoire est une requête parcourant aléatoirement le réseau. Ainsi un nœud effectuant une recherche dans un réseau va générer $k$ marcheurs et les envoyer sur $k$ voisins. Une fois envoyés, les $k$ marcheurs vont choisir aléatoirement des pairs sur lesquels se déplacer. Un marcheur possède, comme dans le cas de l'inondation, un TTL. Si un résultat est trouvé avant que le TTL atteigne 0, alors la réponse est envoyée au pair initiateur de la recherche et les marcheurs s'arrêtent.

Beaucoup d'études ont cherché à optimiser les différents paramètres pour les marcheurs aléatoires. Ainsi dans [Lv *et al.*, 2002] les auteurs ont fait varier le nombre de marcheurs aléatoires ($k$). Les résultats montrent une diminution notable du nombre de messages par nœuds pour 32 marcheurs.

Dans [Bisnik et Abouzeid, 2005] les auteurs ont réglé le nombre de marcheurs $k$ ainsi que leur TTL $T$ à partir de la popularité $p$ d'un fichier. Ici $p$ désigne la probabilité qu'un pair du réseau possède une ressource donnée, c'est-à-dire le ratio entre le nombre de nœuds possédant la ressource et le nombre de pairs total du réseau. Ils ont alors exprimé le coût induit dans le réseau ainsi que le délai attendu pour une recherche en fonction des paramètres $k$, $T$, et $p$. Il résulte de cette modélisation un algorithme de recherche où chaque nœud du réseau maintient une table possédant les paramètres $k$ et $T$ en fonction de $p$. Ainsi un pair effectuant une recherche sur une ressource possédant une popularité donnée consultera la table et initiera une recherche à partir de $k$ marcheurs possédant un TTL de $T$. Les résultats en comparaison avec une méthode de marcheur aléatoire "pure" [8], montrent une diminution de la charge réseau, tout en conservant un taux de succès d'une recherche élevé (supérieur à 95%).

Les auteurs de [Ronasi *et al.*, 2007] proposent une méthode "intelligente" pour rechercher une information dans un réseau. Cette méthode inspirée du comportement animal, et plus précisément des fourmis, propose de déposer des traces de fichiers dans le réseau pour améliorer les mécanismes de recherche. Pour cela les pairs contiennent une table des fichiers qu'ils hébergent et des fichiers hébergés par leur voisin. Pour obtenir cette connaissance, chaque pair utilise une méthode d'inondation bornée pour propager sa liste de fichiers. La borne utilisé appelée *Maxseed* est associée à chaque fichier. Ce couple $(Maxseed, IDfichier)$ est stocké sur chaque nœud parcouru et décrémenté à chaque saut. Ainsi un marcheur aléatoire effectuant une recherche va être orienté vers le pair possédant le *Maxseed* le plus élevé pour un fichier donné. En fonction de la valeur de *Maxseed*, nous observons une diminution du nombre de sauts liée à une augmen-

8. $k$ et $T$ sont fixés et constants

tation du nombre de messages. De plus, ce mécanisme permet de diminuer le trafic réseau en comparaison des autres méthodes de recherche telles que l'inondation ou l'expanding ring.

Enfin les travaux dans [Gkantsidis *et al.*, 2006] prouvent l'influence de la stabilité de la topologie sur les mécanismes de recherche. Ainsi une méthode à base de marcheurs aléatoires se comporte mieux qu'une méthode d'inondation notamment dans le cas où la topologie reste stable entre deux recherches consécutives. Pour un nombre de messages fixé, deux marcheurs aléatoires vont emprunter des chemins différents et ont donc plus de chance de découvrir de nouvelles ressources comparativement à une méthode d'inondation. Pour aboutir à de tels résultats les auteurs ont mesuré le nombre moyen de *hits* (nombre de sources distinctes trouvées), et la probabilité de défaillance correspondant au cas où aucune source n'est trouvée. La première expérience permet de comparer la méthode d'inondation et la marche aléatoire dans un réseau statique. Les deux méthodes ont été initialisées pour utiliser le même nombre de messages. Les résultats sont très similaires entre les deux méthodes, environ 9 *hits* trouvés. La seconde expérience met en jeu un réseau dynamique. Plus la topologie change entre deux recherches consécutives, plus la méthode d'inondation se révèle efficace. D'un autre côté la performance de la marche aléatoire n'est pas affectée par les changements dans le réseau.

### Conclusion

Les méthodes de recherche dans les réseaux pair-à-pair ont été largement étudiés. Les marcheurs aléatoires offrent des performances meilleures que les techniques à base d'inondation, notamment en terme de messages dans le réseau. Toutefois ces résultats s'obtiennent en réglant finement les paramètres des marcheurs [Bisnik et Abouzeid, 2005], à savoir leur nombre, et leur durée de vie (TTL). De plus le comportement et les performances des marcheurs peut-être influencé par la popularité des fichiers stockés dans le réseau [Bisnik et Abouzeid, 2005, Gkantsidis *et al.*, 2006], mais également par la topologie du réseau logique [Gkantsidis *et al.*, 2006].

## Conclusion du chapitre

Nous avons vu dans ce chapitre comment est traité une donnée de son arrivée dans le réseau à sa récupération. La première étape de la vie d'une donnée est sa création. Ce processus requiert des techniques suffisamment robustes et légères pour pouvoir assurer une longue durée de vie à une donnée. Ainsi un fichier va être découpé, chiffré, dupliqué pour assurer sa disponibilité dans le temps.

Une fois ces opérations effectuées, il convient de placer les morceaux d'information dans le réseau. Cette dissémination doit être la plus efficace possible pour pouvoir permettre une collecte des fragments et une reconstruction de la donnée le plus efficacement possible.

Enfin, la dernière étape consiste à retrouver les fragments en ayant le coût le moins élevé possible. Cette étape est fortement lié au placement de l'information. Si l'ensemble des fragments est disséminé aléatoirement dans le réseau alors, il est nécessaire d'en trouver un sous-ensemble

pour reconstruire une donnée. Dans le cas où le placement de l'information suit une politique chaînée, trouver un des fragments permet de contacter plus rapidement les autres morceaux situés sur le voisinage du pair responsable.

Nous avons vu qu'il est difficile de devoir choisir une méthode de placement plutôt qu'une autre. Notre approche consiste à proposer une dissémination dynamique de l'information permettant une adaptation de l'information aux conditions changeantes du réseau. Le placement de l'information n'est alors plus considéré comme un placement définitif mais adaptatif. Pour cela nous nous appuyons sur les concepts d'agent, et de Systèmes Multi-Agents (SMA), introduits dans le chapitre suivant.

# Chapitre 3

# Agents et systèmes multi-agents

**Sommaire**



Les deux chapitres précédents nous ont permis de détailler les deux premiers domaines de notre thèse. Nous nous intéressons ici au troisième domaine, celui ayant attrait à l'intelligence artificielle distribuée. Le but de ce chapitre est de présenter dans un premier temps les notions d'agents ainsi que les notions sur lesquelles un agent repose. Nous introduirons dans un second temps le concept de système multi-agents (SMA). Nous nous attarderons sur le cas des systèmes multi-agents bio-inspirés. Enfin nous terminerons ce chapitre en étudiant le cas des agents logiciels et plus particulièrement celui des agents mobiles dans les réseaux.

## 3.1 Qu'est ce qu'un agent ?

La littérature propose beaucoup de définitions différentes pour la notion même d'agent. Nous nous attacherons ici à recenser les plus importantes et les plus connues, pour proposer notre propre notion d'agent.

### 3.1.1 Définitions

Cette question peut paraître simple mais étant donné le nombre de définitions différentes, il est légitime de se poser la question. Nous allons essayer de faire le tour de cette interrogation tout en essayant de garder l'essence même de ce qui peut caractériser un agent pour essayer d'en donner une définition.

La première définition que nous donnons d'un agent est celle donnée par Wooldridge [Wooldridge, 2002] :

**Définition 5** *Un agent est un système informatique qui est situé dans un environnement, et qui est capable d'actions autonomes dans cet environnement dans le but d'atteindre les objectifs qu'il s'est fixé.*

Dans cette définition, un agent est caractérisé comme une entité autonome, capable de se fixer des objectifs, et d'agir pour essayer de les atteindre, les actions d'un agent prenant place dans un environnement. Ici Wooldridge défini clairement un agent comme une entité informatique dotée d'une capacité de prise de décision pour évoluer dans un environnement. Stuart Russell et Peter Norvig étendent cette définition d'agent dans [Russell et Norvig, 2009], avec la définition suivante :

**Définition 6** *Un agent est une entité qui peut-être vue comme percevant son environnement à travers des capteurs et qui agit sur cet environnement à travers des effecteurs.*

Ici c'est la notion de percevoir et de modifier l'environnement dans lequel l'agent est situé qui est primordiale. La perception et la modification de l'environnement au travers de capteurs et d'effecteurs sont deux des principes clés d'un agent. Il doit être capable d'observer l'environnement dans lequel il se situe, de l'analyser, puis de le modifier. La figure 3.1 montre la boucle perception-réflexion-action d'un agent.

Le meilleur exemple d'agent au sens de Stuart et Russell est un robot autonome utilisé dans des missions spatiales. Un tel agent peut par exemple :

- prendre des photos (rôle des capteurs permettant de percevoir l'environnement),
- prendre la décision d'envoyer les photos en fonction des ressources restantes (réflexion),
- se déplacer ou déplacer un objet dans l'environnement (rôle des effecteurs).

Jacques Ferber propose dans [Ferber, 1995] une définition essayant d'englober l'ensemble des caractéristiques requises pour un agent :

**Définition 7** *On appelle agent une entité physique ou virtuelle :*

- *qui est capable d'agir dans un environnement,*
- *qui peut communiquer directement avec d'autres agents,*
- *qui est mue par un ensemble de tendances (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'elle cherche à optimiser),*
- *qui possède des ressources propres,*

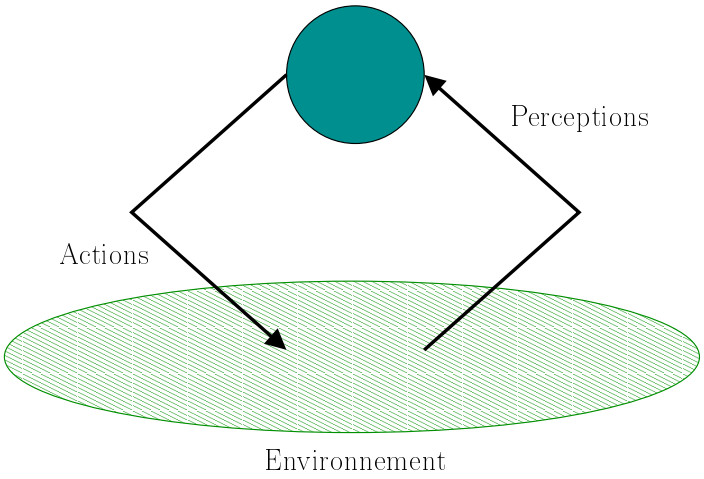



FIGURE 3.1 – Définition d'un agent

- *qui est capable de percevoir (mais de manière limitée) son environnement,*
- *qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune),*
- *qui possède des compétences et offre des services,*
- *qui peut éventuellement se reproduire,*
- *dont le comportement tend à satisfaire ses objectifs, en tenant compte des ressources et compétences dont elle dispose, et en fonction de sa perception, de ses représentations et des communications qu'elle reçoit.*

Cette définition, très complète, liste les capacités nécessaires pour définir un agent. De la même manière que pour Stuart et Russell, ici un agent est une entité physique ou informatique, évoluant dans un environnement, et agissant dans l'objectif d'atteindre des buts. On peut noter que cette définition est donnée dans le cadre de la conception de système multi-agents. De fait un agent peut posséder des capacités étendues telles que la communication ou le clonage.

De ces définitions nous pouvons retenir deux mots clés essentiels à la définition d'un agent : **environnement** et **autonomie**.

### Environnement

Un environnement est le lieu où un agent évolue, observe, interagit. Pour caractériser et définir plus finement un environnement, Russell et Norvig ont proposé l'ensemble de propriétés suivant. Un environnement peut être :

- **totalement observable** ou **partiellement observable**. Le fait qu'un environnement soit totalement observable permet à un agent de posséder l'ensemble des connaissances du monde dans lequel il est. Dans le cas où un agent possède qu'une connaissance limitée du monde, on parle d'environnement partiellement observable.

- **déterministe** ou **stochastique**. Un environnement est dit déterministe quand la probabilité de passer d'un état $s$ à un état $s'$ quand un agent effectue une action $a$ est 0 ou 1. Nous avons donc $P(s, a, s') \in \{0, 1\}$. Dans le cas d'un environnement stochastique $P(s, a, s') \in [0, 1[$. Ici à partir d'un même état $s$, et d'une même action $a$ il est possible d'arriver dans différents états.
- **épisodique** ou **séquentiel**. Un environnement épisodique est un environnement où pour un agent le couple perception, exécution d'une action constitue un épisode. Ainsi le vécu d'un agent est découpé en épisode indépendant entre eux, le résultat d'une action ne dépendant que de l'épisode courant. Dans un environnement séquentiel, l'exécution d'une action à une influence sur l'ensemble des actions à venir.
- **statique** ou **dynamique**. Un environnement dynamique peut être modifié à n'importe quel moment, même pendant la prise de décision d'un agent. De fait la décision d'un agent peut ne plus être en accord avec une situation initiale. Un environnement statique est un environnement qui ne subit pas de modifications dans le temps.
- **discret** ou **continu**. Un environnement est discret s'il existe un nombre fini de perceptions et d'actions. Dans le cas contraire, on dit qu'il est continu.

D'après ces propriétés on peut facilement voir que la réflexion d'un agent est étroitement liée à l'environnement dans lequel il se situe. Plus un agent aura à sa disposition des informations fiables et en quantité, plus la décision résultant de sa réflexion sera appropriée à une situation donnée [Wooldridge, 2002].

### Autonomie

Nous avons vu qu'un agent plongé dans un environnement est capable d'observer et, à partir de ses perceptions, de prendre une décision. Cette décision est prise en toute autonomie. Pour Russell et Norvig cette décision est uniquement basé sur les connaissances acquises par l'agent au cours du temps, et ne nécessite aucun savoir lié à la conception de l'agent. Initialement un agent ne possède donc aucun savoir et doit donc acquérir de la connaissance au cours du temps pour améliorer son comportement.

La définition d'autonomie apporté par Wooldridge et Jennings est plus générale. Un agent est dit autonome quand ses décisions sont prises sans l'intervention d'un tiers. Le terme tiers peut désigner, le concepteur de l'agent (au quel cas on se rapporte à la définition de Russell et Norvig), un centre de contrôle présent dans le système, ou bien même un autre agent.

L'autonomie est un des facteurs qui permet de différencier un agent d'un programme informatique. Un agent ne possède pas de connaissances héritées de sa conception, et acquière de l'expérience dans le temps à partir de ses propres décisions liées aux observations de l'environnement.

La prise de décision est étroitement liée aux notions d'environnement et d'autonomie. Toutefois cela n'est pas suffisant pour décrire précisément le mécanisme de décision d'un agent.

Nous allons donc nous intéresser à la mécanique interne d'un agent et à la représentation des connaissances pour un agent.

### 3.1.2 Plusieurs modèles d'agents

Nous allons exposer plusieurs modèles d'agents. Ces modèles permettent de définir le degré de réflexion d'un agent, et permettent ainsi de fournir une aide à la conception d'agent en fonction des besoins.

#### Réactifs

La **réactivité** est le processus qui, à partir d'une observation, aboutit à une action. Un agent est dit réactif, s'il répond de manière opportune aux changements de son environnement. Les changements observés sont de l'ordre du stimulus. Un tel type d'agent n'a donc pas besoin de représentation complexe de son environnement et de lui-même. Pour caractériser un tel agent, Brustoloni [Brustoloni, 1991] prend l'exemple d'un thermostat dans une pièce. Le but d'un thermostat est de réguler la température d'une pièce. Ici l'agent thermostat va seulement se contenter de réagir en fonction des stimuli perçus (température). Il n'a aucune compréhension de lui-même et de l'environnement dans lequel il est plongé.

Un agent réactif possède donc un comportement "très simple". Toutefois ces agents peuvent être capable d'actions de groupe complexes et structurées. Le meilleur exemple d'agent réactif est une fourmi. Alexis Drogoul décrit dans sa thèse [Drogoul, 1993], au travers d'une approche multi-agents, un phénomène de sociogénèse dans un groupe de fourmis. Au fur et à mesure que cette société de fourmi évolue, une organisation sociale entre les fourmis se met en place. L'évolution de la société et des interactions sociales entre les fourmis permet alors de réaliser des activités complexes dirigées vers un objectif global. Ces activités sont alors des propriétés émergentes de cette société.

#### Cognitifs

Les agents cognitifs ont souvent été mis en opposition aux agents purement réactifs. Un agent cognitif est un agent complexe possédant une structure interne lui conférant une capacité de réflexion que ne possède pas un agent réactif. Ainsi un agent cognitif possède une représentation explicite de l'environnement et des autres agents, une mémoire, et un but explicite. Les questions qu'un agent cognitif doit se poser sont :

- Qu'est-ce que je veux faire ? Quel est mon but ?
- Que vois-je ? Quelles informations je perçois de mon environnement ?
- Où suis-je ? Dans quel état je me situe ?

De ces interrogations découle une phase de réflexion conduisant au choix d'une action à effectuer.

On dit aussi que ces agents sont des agents intentionnels dans le sens où ils possèdent explicitement leurs buts et les plans leurs permettant de les accomplir.

### 3.1.3 Cas des agents logiciels

Nous avons vu qu'un agent est une entité autonome évoluant dans un environnement. Cette définition s'applique particulièrement bien aux agents physiques tels que les robots qui peuvent au travers de capteurs et d'effecteurs interagir avec leur environnement. Il existe une seconde catégorie d'agents pour lesquels les notions d'environnement, de ressources, et d'interactions sont plus floues. Ce sont les agents logiciels. Nous allons maintenant présenter le concept d'agent logiciel, et nous intéresser plus particulièrement à la notion de mobilité pour un agent logiciel.

#### Définitions

Par définition, un agent mobile est un agent logiciel ou virtuel, n'ayant par conséquent pas de représentation physique. Un agent logiciel est constitué de code s'exécutant dans un environnement informatique. La frontière entre un agent logiciel et un simple programme informatique est mince. Toutefois il est assez simple de voir qu'un agent informatique est plus évolué qu'un programme informatique. Là où un programme s'exécute sur une machine et produit un résultat, un agent logiciel est capable de s'adapter aux conditions changeantes de l'environnement (ressources CPU, mémoire disponible, bande passante, etc) pour réaliser le but qu'il s'est ou qui lui a été fixé.

L'adaptation d'un agent logiciel passe par des capacités de **réplication**, de **terminaison**, de **communication** et de **déplacement**. Derrière ces trois termes se cachent les propriétés suivantes :

- Réplication : un agent logiciel peut décider de se dupliquer sur un site distant pour, par exemple, distribuer un calcul trop important.
- Terminaison : un agent logiciel peut décider d'arrêter son exécution sans intervention extérieure.
- Communication : un agent logiciel est capable de communiquer par l'intermédiaire de messages avec d'autres agents logiciels.
- Déplacement : un agent logiciel possède la capacité de se déplacer dans l'environnement dans lequel il est situé. Dans le cas de ressources devenues insuffisantes sur une machine, un agent logiciel peut choisir de se déplacer sur un autre dispositif informatique pour poursuivre son exécution.

#### Conclusion

Un agent est un concept très riche pouvant être définit de plusieurs façons. Cependant, on retrouve assez facilement des concepts communs pour expliciter un agent. Ainsi un agent est une entité dotée de capacités de réflexion (plus ou moins importantes), évoluant dans un environnement avec lequel il peut interagir par le biais d'effecteurs. De plus un agent peut-être doté de modules lui permettant d'établir des communications. Dans notre cas, un agent correspond à la définition précédente, au détail près qu'il s'agit d'un agent mobile donc logiciel et que son environnement est une interconnexion de machines.

## 3.2 Systèmes Multi-Agents

Le concept d'agent prend tout son sens, non pas dans le cas où un seul agent est plongé dans un environnement, mais quand plusieurs agents évoluent et interagissent en même temps dans un même environnement. Un tel système est appelé **système multi-agents** ou **SMA**.

### 3.2.1 Définitions

De la même manière que pour la notion d'agent, il existe une multitude de définition d'un système multi-agents. Nous allons essayer d'en dégager les plus importantes pour, dans les sections suivantes, expliciter les caractéristiques d'un système multi-agent.

Jacques Ferber propose dans [Ferber, 1995] la définition suivante pour un SMA :

**Définition 8** *On appelle système multi-agent (ou SMA), un système composé des éléments suivants :*

- *Un environnement E, c'est-à-dire un espace disposant généralement d'une métrique.*
- *Un ensemble d'objets O. Ces objets sont situés, c'est-à-dire que, pour tout objet, il est possible, à un moment donné, d'associer une position dans E. Ces objets sont passifs, c'est-à-dire qu'ils peuvent être perçus, créés, détruits et modifiés par les agents.*
- *Un ensemble A d'agents, qui sont des objets particuliers ($A \subseteq O$), lesquels représentent les entités actives du système.*
- *Un ensemble de relations R qui unissent des objets (et donc des agents) entre eux.*
- *Un ensemble d'opérations Op permettant aux agents de A de percevoir, produire, consommer, transformer et manipuler des objets de O.*
- *Des opérateurs chargés de représenter l'application de ces opérations et la réaction du monde à cette tentative de modification, que l'on appellera les lois de l'univers.*

Ici Ferber propose de situer dans un environnement composé d'objets modifiables, des agents dotés de capteurs et d'effecteurs unis entre eux par des relations. Dans cette définition c'est clairement le couple agent/environnement qui est au cœur d'un SMA.

La notion de relation entre objets aperçue dans la définition précédente est étendue dans l'approche proposée par Nicholas R. Jennings [Jennings, 2000]. Ici c'est la notion d'interaction qui est centrale dans un SMA.

La figure 3.2 représente un SMA composé d'agents interagissant entre eux, et regroupés à l'aide de relations organisationnelles. A cela Mickaël Wooldridge [Wooldridge, 2002] ajoute des sphères d'influences, distinctes des relations de haut-niveau, représentant l'influence des agents sur l'environnement. Dans le cas où des sphères d'influences se chevauchent alors, cela traduit des relations d'ordre physique entre les agents. Wooldridge prend l'exemple de deux robots devant passer par une même porte. Ils ne peuvent pas le faire simultanément, il est donc nécessaire de posséder des relations de haut-niveau permettant par exemple de déterminer un ordre de passage.

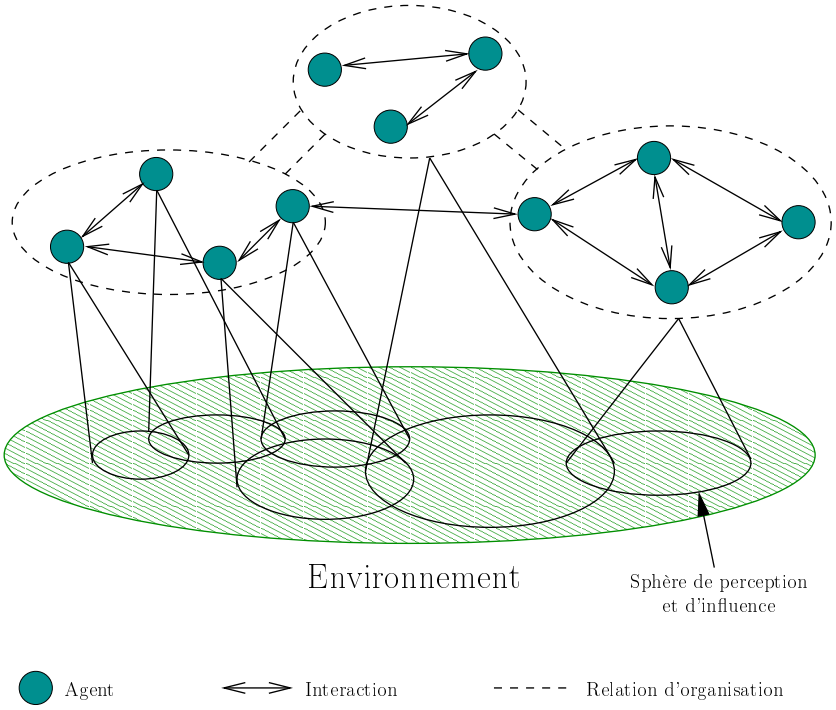



FIGURE 3.2 – Représentation d'un SMA [Jennings, 2000]

Yves Demazeau [Demazeau, 1995, Demazeau, 1997] décrit dans l'approche Voyelle les composantes majeures d'un SMA. Ainsi nous pouvons trouver dans un SMA, des **A**gents, un **E**nvironnement, des **I**nteractions entre les agents, et au moins une **O**rganisation. Ceci peut être résumé par l'équation suivante :

$$MAS = A + E + I + O \tag{3.1}$$

A cette équation Jacques Tisseau [Tisseau, 2001] propose d'ajouter le point de vue de l'**U**tilisateur. De cette façon le concepteur d'un SMA, ou un utilisateur quelconque peut participer et intervenir directement dans le SMA. L'utilisateur est alors considéré comme un agent à part entière dans le SMA.

Au travers des différentes définitions apportées dans la littérature, nous pouvons clairement observer qu'un SMA repose sur les concepts d'agent, d'environnement, d'interaction, et d'organisation. Nous avons déjà introduit dans la section précédente les notions d'agent et d'environnement. Nous allons maintenant nous intéresser aux différentes interactions et organisations pouvant exister entre les agents.

### 3.2.2 La notion d'interaction

#### Classification des interactions

Jacques Ferber [Ferber, 1995] propose la définition suivante pour une interaction :

**Définition 9** *Une interaction est une mise en relation dynamique de deux ou plusieurs agents par le biais d'un ensemble d'actions réciproques.*

L'exemple de Wooldridge sur les robots est un bon exemple d'interaction. Deux robots souhaitant passer simultanément au travers d'une même porte trop étroite. Cette situation conduit les deux robots à avoir une interaction pour décider des actions à effectuer en fonction de leurs buts respectifs, et des ressources disponibles. A partir de cette définition, Ferber propose le tableau 3.1 pour ordonner les différents types d'interactions.

| **Buts** | **Ressources** | **Compétences** | **Situation** |
|---|---|---|---|
| Compatibles | Suffisantes | Suffisantes | **Indifférence** |
| Compatibles | Suffisantes | Insuffisantes | **Coopération** |
| Compatibles | Insuffisantes | Suffisantes | |
| Compatibles | Insuffisantes | Insuffisantes | |
| Incompatibles | Suffisantes | Suffisantes | **Antagonisme** |
| Incompatibles | Suffisantes | Insuffisantes | |
| Incompatibles | Insuffisantes | Suffisantes | |
| Incompatibles | Insuffisantes | Insuffisantes | |

TABLE 3.1 – Classification des différentes interactions au sens de Ferber

L'**indifférence** caractérise une situation où les agents sont totalement indépendants. Ici chaque agent peut atteindre ses objectifs en complète indépendance vis-à-vis des autres agents. La **coopération** nécessite que les agents agissent de concert pour atteindre leurs buts. C'est le cas quand des agents possèdent des buts compatibles, mais que soit leurs ressources soit leurs compétences viennent à manquer. Enfin l'**antagonisme** traduit le fait que les agents sont en situation de compétition (les buts sont incompatibles).

### La communication entre agents

Le meilleur moyen pour faire interagir plusieurs agents est la communication. Cette communication peut-être **directe** ou **indirecte**.

Dans le cadre d'une communication directe, un message est directement adressé à un ou plusieurs agents. Cet envoi de message peut être à son tour réalisé de façon **synchrone**, ou **asynchrone**. Une communication directe synchrone nécessite que la communication soit bien établie et que le message envoyé soit effectivement arrivé au destinataire pour obtenir une réponse. Le meilleur exemple de communication synchrone est une conversation téléphonique. A l'inverse la communication asynchrone permet une séparation distincte dans le temps des phases d'envoi de message/réponses au message. Le meilleur exemple de communication asynchrone est un forum de discussion.

La communication indirecte suppose que les agents ne possèdent pas de moyens pour communiquer directement. Dans ce cas, les agents peuvent utiliser leur environnement pour établir

une communication avec les autres agents. Ainsi un agent peut laisser des traces dans l'environnement pour donner des informations sur son état ou ses actions [9]. Un exemple d'environnement par lequel la communication peut s'établir est la mémoire vive d'un ordinateur. Les processus peuvent communiquer entre eux en déposant des informations sur cette mémoire volatile.

Le meilleur exemple de communication indirecte par l'environnement est le dépôt de phéromones des fourmis (cf. section 3.2.4).

### 3.2.3 La notion d'organisation

Pour des raisons d'efficacité d'un système multi-agents, il est nécessaire de regrouper les interactions entre agents au sein d'une organisation en y définissant les relations entre les agents. Ce regroupement d'interactions pour former un tout est repris dans la définition d'organisation apportée par Jacques Ferber[Ferber, 1995] :

**Définition 10** *Une organisation peut être définie comme un agencement de relations entre composants ou individus qui produit une unité, ou système, doté de qualités inconnues au niveau des composants ou individus. L'organisation lie de façon interrelationnelle des éléments ou événements ou individus divers qui dès lors deviennent les composants d'un tout. Elle assure solidarité et solidité relative, donc assure au système une certaine possibilité de durée en dépit de perturbations aléatoires.*

Pour Edmund Durfee [Durfee *et al.*, 1987], une organisation spécifie un ensemble de responsabilités à long-terme et des cadres d'interactions pour les agents. Ces informations supplémentaires doivent guider les décisions locales des agents et ainsi permettre d'améliorer le comportement des agents en fournissant une stratégie globale. Pour étayer leur définition, les auteurs prennent pour exemple un réseau de nœuds chacun pouvant résoudre un ou une partie d'un problème. Ils représentent l'organisation par un ensemble de structures de données appelé "zones d'intérêt" d'un nœud. Chaque zone représente alors les différents paramètres qu'un nœud est susceptible d'utiliser. En ajoutant un mécanisme de communication de haut-niveau sur la coordination, les résultats montrent que les nœuds sont capables de tirer parti de l'ensemble des connaissances disponibles, qu'elles soient locales ou non, pour former une équipe la plus performante possible.

Bryan Horling et Victor Lesser recensent dans [Horling et Lesser, 2005] les différentes formes d'organisations existantes. La première, et sans doute une des plus connues, est la **hiérarchie**. Dans une hiérarchie, les agents sont organisés sous la forme d'arbre, la racine étant l'agent ayant le plus de responsabilités et pouvant donner des ordres aux agents situés en dessous de lui dans l'arbre. L'**holarchie** est quand à elle une hiérarchie d'holons, où chaque holon est à la fois tout et une partie de l'organisation. Par exemple, un organisme est un holon composé de cellules qui sont des holons composés de molécules qui sont des holons composés d'atomes ... Les **coalitions** sont des groupes d'agents possédants une faible durée de vie, formés pour apporter une réponse adéquate à un problème bien précis. Il n'y pas nécessairement de hiérarchie dans une

9. La communication indirecte est une communication asynchrone

coalition. Les **équipes** sont le résultat des décisions d'agents qui ont pour but de travailler sur un objectif commun [Durfee *et al.*, 1987]. A la différence d'une coalition, ici les agents cherchent en priorité à satisfaire le but global plutôt que leur intérêt personnel. Les **congrégations** permettent d'organiser les agents en fonctions de leurs compétences dans le but de faciliter de futures collaborations. Ici les agents cherchent à maximiser leur utilité à long terme.

Les **sociétés** sont des espaces ouverts, où les agents peuvent aller et venir comme bon leur semble. On dispose ainsi d'une organisation composée d'agents hétérogènes. La société peut dès lors être vue comme un lieu d'échanges entre les agents. Une société est également constituée de règles imposées sur le comportement des agents. L'organisation sous la forme de **fédérations** permet de constituer des groupes d'agents possédant en leur sein un représentant. Ce représentant est le seul moyen d'échange avec l'extérieur du groupe, et les autres groupes. L'organisation de **marché** repose sur le principe des enchères. Les agents peuvent placer des enchères, ou mettre en enchères des ressources ou des tâches. Pour s'assurer du bon fonctionnement général, des agents jouent le rôle d'adjudicateur. L'organisation **matricielle** relâche la contrainte *un agent possède un unique supérieur* de la hiérarchie en autorisant plusieurs supérieurs à posséder une influence sur les actions d'un agent. Enfin les organisations **composées** sont celles possédant plusieurs types d'organisations, chacune ayant un but bien précis. Le tableau 3.2 recense l'ensemble des avantages/inconvénients des organisations que nous avons évoqués précédemment.

| **Paradigme** | **Caractéristiques principales** | **Avantages** | **Inconvénients** |
|---|---|---|---|
| Hiérarchie | Décomposition | Couramment applicable<br>Passage à l'échelle | Fragile<br>Goulot d'étranglement |
| Holarchie | Décomposition avec autonomie | Exploite l'autonomie des unités fonctionnelles | Devoir d'organiser les Holons<br>Prédictabilité des performances |
| Coalition | Dynamique<br>Orienté but | Exploite le fait d'être nombreux | Bénéfices à court terme potentiellement inférieur au coût de construction |
| Equipe | Cohésion au niveau du groupe | L'équipe peut traiter des problèmes qu'aucun agent ne pourra faire seul<br>Centré tâche | Augmente les communications |
| Congrégation | Longue durée de vie<br>Orienté utilité | Faciliter la découverte d'agents | Les ensembles peuvent être trop restrictifs |
| Société | Système ouvert | Services publiques<br>Conventions bien définies | Potentiellement complexe<br>Nécessité de capacités liées à la société |
| Fédération | Agents intermédiaires | Services de traduction intermédiaires<br>Facilite un pool d'agents dynamiques | Les intermédiaires sont des goulots d'étranglement |
| Marché | Compétition à travers les prix | Allocation<br>Utilité accrue par la centralisation<br>Équité accrue par les enchères | Potentielles collusions<br>Comportements malicieux<br>Complexité de la décision d'allocation |
| Matrice | Managers multiples | Partage de ressources<br>Multiplication des agents influencés | Conflits potentiels |
| Composé | Organisations concurrentes | Exploite les avantages de plusieurs organisations | Augmente la sophistication<br>Inconvénient de différentes organisations |

TABLE 3.2 – Tableau comparatif des différentes formes d'organisation [Horling et Lesser, 2005]

Les organisations précédentes sont fournies généralement lors de la conception d'un SMA. Dans certains cas, la structure du système émerge des comportements. Ce phénomène est l'**auto-organisation**. Cette forme d'organisation, opposée à une organisation centralisée, dispose de deux niveaux d'observation. Le premier, appelé niveau microscopique, est le niveau où ont lieu les interactions. Le second, appelé niveau macroscopique, est celui où une structure émerge. Nous retrouvons cette notion de niveau dans la définition issue de [Camazine *et al.*, 2001] :

**Définition 11** *L'auto-organisation est un processus dans lequel l'organisation au niveau global d'un système émerge seulement des nombreuses interactions issus des composants bas niveau du système. De plus, les règles spécifiant les interactions entre les composants du système, sont exécutées seulement en utilisant les informations locales, sans références à l'organisation globale.*

Les exemples d'auto-organisation sont nombreux, et sont le plus souvent issus de systèmes biologiques. C'est ce que nous allons étudier en nous intéressant aux SMA bio-inspirés.

### 3.2.4 Systèmes Multi-Agents bio-inspirés

Les concepteurs de SMA ont souvent puisé leur inspiration dans le domaine biologique. Beaucoup de travaux ont été inspirés par les animaux et plus particulièrement par leur comportement et les organisations régissant différentes espèces. Les deux types d'animaux les plus étudiés par la communauté sont les fourmis, et les oiseaux. Les fourmis ont notamment été étudiés pour leur comportement et leur organisation sous la forme de société animale. Les oiseaux ont été étudiés pour leur capacité à se déplacer en groupe durant leur vol. Cette capacité est appelée **flocking**. Nous allons maintenant faire un tour d'horizon des travaux portant sur ces deux thématiques.

#### Cas des fourmis

Alexis Drogoul dans sa thèse [Drogoul, 1993] s'est attaché à reproduire le comportement social d'une colonie de fourmis. Au travers de la plate-forme MANTA, il a ainsi pu reproduire le fonctionnement d'une colonie de fourmis. Pour cela chaque fourmi a été modélisé comme un agent réactif possédant donc un comportement de type stimulus/réponse. Les expériences menées portent sur la naissance d'une société entre les agents (sociogénèse). Ici les agents représentent les œufs, les larves, les cocons, les fourmis ouvrières, et les reines. Pour compléter le modèle la nourriture, la lumière et l'humidité ambiante ont été modélisées.

Les premières expériences valident le comportement observé dans la réalité, et voient l'apparition d'une société de fourmis fidèle à celles observées par les spécialistes. On peut ainsi retrouver une spécialisation des fourmis conduisant à une division du travail au sein de la colonie, alors que ce comportement n'est aucunement présent à la conception des agents. Les agents ne possèdent aucune représentation de la séparation du travail, et cette dernière ne nécessite aucun contrôle centralisé. Ce comportement est donc totalement émergeant. Une organisation de type société se met donc en place progressivement au fur et à mesure que de nouvelles fourmis apparaissent.

Tout les choix effectués reposent sur des études éthologiques très complètes et sur l'observation attentive par des éthologues du fonctionnement de fourmilières. De ces observations une modélisation fidèle des agents et de l'environnement jouant un rôle clé dans la vie sociale d'une fourmilière a été obtenue.

Une seconde série d'expériences permet de mettre en évidence l'apport d'une telle approche pour la résolution collective de problèmes. Un des exemples les plus connus et les plus étudiés est le problème des robots fourrageurs [Steels, 1989, Goss et Deneubourg, 1991, Simonin, 2001], terme emprunté aux colonies de fourmis. Un tel problème consiste pour un ensemble de robots au comportement simple évoluant dans un environnement incertain, à trouver des ressources et à les ramener à une base spatiale [10]. Dans le cas des fourmis ce problème est résolu par l'utilisation de phéromones. Une phéromone est une trace chimique volatile laissée dans l'environnement par une fourmi agissant comme un messager. Dès lors les fourmis sont capables d'établir une communication indirecte entre elles par le biais de l'environnement. Cette modification indirecte de l'environnement est appelé *stigmergie* [Grassé, 1959].

Dans le cadre du problème de la collecte de ressources, les fourmis cherchent aléatoirement de la nourriture (ressources). Une fois l'objectif atteint la fourmi rentre au nid (base spatiale) en déposant des phéromones le long du chemin de retour. Ainsi après avoir déposé au nid la nourriture collectée, la fourmi peut retourner sur le site fournisseur de nourriture. Ce comportement permet de définir au cours du temps un plus court chemin entre la source de nourriture et le nid.

Il existe de nombreux autres problèmes basée sur des algorithmes à base de fourmis. Nous pouvons par exemple citer le problème du voyageur de commerce [Colorni *et al.*, 1991], ou de la classification de données [Monmarché, 2000].

### Flocking, Swarming, Schooling

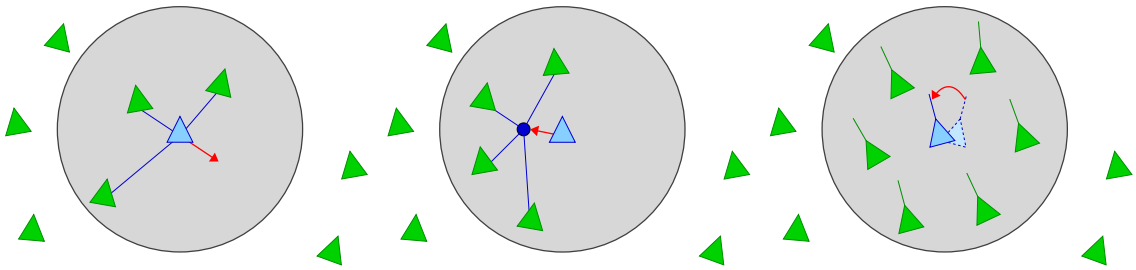
La seconde catégorie d'animaux à la quelle nous nous intéressons est celle des oiseaux. Depuis plusieurs années, et ceci dans beaucoup de domaines, de multiples travaux ont cherché à reproduire le comportement de groupe observé durant le vol des oiseaux. C'est ce que l'on désigne par **flocking**.

Le principe du flocking a été introduit par Craig W. Reynolds en 1987 [Reynolds, 1987] sous le nom de **Boids**. Le but est de fournir des règles simples aux agents pour aboutir à un comportement émergeant de groupe. Le modèle de base à été conçu autour de trois règles :

- *séparation :* deux agents ne peuvent pas rentrer en collision entre eux (figure 3.3(a)),
- *cohésion :* un agent doit rester à proximité de ses voisins (figure 3.3(b)),
- *alignement :* les agents doivent aller dans la même direction et à la même vitesse (figure 3.3(c)).

C'est la combinaison de ces trois règles simples qui permettent à un groupe d'agents de se déplacer sous la forme d'une nuée sans qu'aucun membre ne rentre en collision avec un autre.

10. Il est souvent pris l'exemple de robots évoluant sur la planète Mars.



(a) Règle de séparation (b) Règle de cohésion (c) Règle d'alignement

FIGURE 3.3 – Règles de flocking [Reynolds, 1987]

D'après Craig W. Reynolds le flocking est donc un comportement collectif émergeant, résultant d'interactions locales entre les agents.

L'un des domaines où l'on peut rencontrer le plus de modèles s'inspirant du flocking est la robotique. Dans [Gervasi et Prencipe, 2004] chaque véhicule autonome est équipé de capteurs permettant d'observer la position des autres véhicules et possède une vue de son monde lui permettant de s'orienter. Ici l'approche retenue met en jeu un agent particulier, un leader chargé d'orienter le déplacement de la nuée de robots. L'utilisation d'un leader dans une nuée est clairement une faiblesse du modèle. Dans le cas où le leader n'est plus opérationnel, il est nécessaire que le reste des robots continue à se comporter en groupe. C'est à partir de ce constat que les auteurs dans [Canepa et Potop-Butucaru, 2007] ont proposé un modèle reposant sur l'élection d'un leader. A la différence de [Gervasi et Prencipe, 2004], les auteurs de [Tanner *et al.*, 2003a] ont montré que la forme du nuage d'agents n'est pas forcément figée, et peut même être dynamique [Tanner *et al.*, 2003b]. Il n'est donc pas nécessaire lors de la création d'un groupe d'agents d'assigner une forme précise au nuage, ou une position à chaque membre [Gervasi et Prencipe, 2004].

Dans [Hayes et Dormiani-Tabatabaei, 2002], c'est une approche sans leader qui a été retenue. Les auteurs se servent seulement de deux comportements pour reproduire un mouvement de groupe émergeant :

- éviter les obstacles à savoir les objets de l'environnement ou les agents,
- se déplacer en suivant la vélocité du centre de la nuée.

Ici chaque agent est capable de calculer, à partir de la position des agents dans son voisinage, un vecteur représentant le centre de gravité. Ainsi les robots autonomes sont capables de s'aligner suivant le centre de gravité de la nuée estimée.

Dans [Olfati-Saber, 2006], l'approche sans leader est obtenue en représentant les agents à l'aide un graphe de proximité où chaque agent possède une sphère d'influence. Pour modéliser l'organisation des agents au sein d'une nuée, l'auteur a utilisé une structure en treillis. Cette modélisation d'une nuée permet également de définir l'évitement d'obstacles. Cette propriété est obtenue par l'utilisation d'agents virtuels résultat de la projection d'agents de la nuée sur les obstacles de l'environnement. Cette méthode permet de déclarer un agent "voisin" d'un

obstacle, et donc par conséquent de sa projection, lorsque sa sphère d'influence est rompue par un obstacle.

Dans [Lindhé *et al.*, 2005] la méthode utilisée pour obtenir une nuée de véhicules autonomes est basée sur des partitions de Voronoï. Une partition de Voronoï est une décomposition d'un espace métrique déterminée par les distances à un ensemble d'objets de l'espace. Un diagramme de Voronoï partitionne donc un espace en régions, chacune d'entre elles réunissant les points qui sont plus proches d'un objet que de tous les autres. Ainsi à partir du voisinage d'un véhicule, obtenu à l'aide de capteurs, et de l'utilisation de partitions de Voronoï un algorithme de coordination entre véhicules et d'évitement d'obstacle est construit.

De nombreux autres domaines d'application se sont intéressés au flocking. Nous pouvons citer par exemple la classification de données [Monmarché *et al.*, 2002] ou bien encore la visualisation et modélisation de données [Proctor et Winter, 1998].

## 3.3 Agents mobiles et SMA dans les réseaux

#### Mobilité forte / Mobilité faible

Les premières apparitions d'agents mobiles viennent de Telescript [White, 1994], une architecture permettant à des processus de se déplacer dans un réseau dans l'objectif de travailler sur des ressources locales. Dans le cas de Telescript les processus se déplacent avec leur pile d'exécution. Ainsi lorsqu'un processus ce déplace, il passe par un état non-actif et peut reprendre le cours normale de son exécution. Dans MOOREA [Dillenseger *et al.*, 2002], nous trouvons un mécanisme similaire. Lors d'une demande de mobilité, l'état de l'agent est sauvegardé (pile d'exécution et données), puis réstauré après déplacement, et ce quelque soit l'instant dans l'activité de l'agent. Nous parlons alors de **mobilité forte**.

L'inconvénient majeur de la mobilité forte est son coût important et non négligeable. Pour se déplacer et reprendre le cours de l'exécution d'un processus il est nécessaire d'en capturer l'état. Cette étape coûteuse est difficile à réaliser dans un environnement hétérogène.

Pour apporter une réponse à ces problèmes, il est possible de déplacer un agent, non pas avec son unité d'exécution, mais seulement avec ses données et caractéristiques internes. De cette manière un agent est capable de se déplacer sur un site distant, mais n'est pas en mesure de reprendre son exécution là ou elle s'est arrêtée. Nous parlons alors de **mobilité faible**.

### 3.3.1 Apports et inconvénients des agents mobiles

L'utilisation d'agents mobiles dans un réseau large échelle permet dans un premier temps d'améliorer les performances globales du réseau. Ainsi dans [Sahai et Morin, 1998], il est montré au travers de la plate-forme agents mobiles MAGENTA et d'un exemple de gestionnaire de réseau, que l'utilisation d'agents mobiles permet de réduire de manière significative l'utilisation de la bande passante en comparaison avec une approche client/serveur. Des résultats du même ordre sont observés avec la plate-forme **D'Agents** [Gray *et al.*, 2002]. Ici c'est une application

de collecte d'informations qui a été choisi pour illustrer le gain de bande passante. Dag Johansen a montré dans [Johansen, 1998] que l'utilisation d'agents mobiles, toujours en comparaison avec le modèle client/serveur, permet également un gain en terme de temps de réponse. Pour cela il s'est basé sur des applications de rapatriement d'images et de videos construites grâce à la plateforme TACOMA [Johansen *et al.*, 1995]. Enfin Salah El Falou [El Falou, 2006] a étudié les performances des agents mobiles en tant qu'entité de communication dans le cadre de la recherche sur Internet. L'approche proposée, reposant sur l'utilisation de Processus Décisionnel de Markov, permet de réduire conjointement le délai d'attente et le trafic sur le réseau.

Dans une application répartie, les performances globales du systèmes sont améliorées si la charge de travail est correctement répartie. Dans la majeure partie des cas, l'allocation de tâche se fait statiquement, et est basée sur des connaissances *a priori*. Une fois les tâches allouées et débutées, celles-ci ne peuvent faire l'objet d'une réallocation. Du fait de leur mobilité et de leur adaptabilité, les agents mobiles peuvent fournir une allocation dynamique. Ainsi dans [Cao *et al.*, 2003] les auteurs ont montré les apports d'une telle approche, en observant simplement la charge de serveurs placés dans un cluster et dans un réseau large échelle. Dans [Bredin *et al.*, 1998] la répartition de charge se fait au travers d'enchères. Les agents utilisent une monnaie électronique et effectuent leurs transactions au travers d'une banque de confiance.

Dans [Johansen, 2004], Dag Johansen tente d'expliquer pourquoi les agents mobiles ne sont pas plus utilisés de nos jours. Parmi les raisons évoquées, on peut trouver tout à d'abord la trop grande variété de plate-forme agents mobiles. Il existe de multiples plates-formes de développement pour agents mobiles. Chacune d'entre elles possèdent des qualités et il devient dès lors compliqué pour un développeur de devoir choisir une plate-forme adéquate. Nous reviendrons sur ce point dans la section 3.3.2. Le second grief contre les agents mobiles, est le problème de la sécurité. Les agents mobiles sont impliqués dans deux catégories d'attaques [Loureiro, 2001] :

- Un agent mobile peut avoir un comportement malveillant et attaquer son site d'exécution et les ressources liées.
- Les données internes (codes, données, états ...) peuvent être modifiées par un site malveillant.

Il existe de nombreuses solutions pour se prémunir d'attaques émanant d'agents mobiles. La plus connue est le recours à un **bac à sable** pour l'exécution de l'agent. L'exécution de l'agent se fait dans un espace restreint où l'agent n'a pas accès aux ressources essentielles et vitales d'une machine. Cette technique est notamment utilisée dans le cadre d'applications développées en Java [Fuggetta *et al.*, 1998].

Pour garantir le fonctionnement et le comportement d'un agent, il est possible de s'intéresser à sa structure interne et plus particulièrement à son code. Ainsi à sa création il est possible de **signer numériquement** le code d'un agent mobile. Ainsi lors d'un déplacement, un agent est capable de s'identifier auprès du site récepteur de destination. Authenticode [Authenticode, 1996] est un exemple de signature de code permettant de signer des applets Java ou des contrôles Active X. L'utilisateur de telles applications peut alors vérifier l'identité de l'auteur du logiciel

et l'intégrité de ce dernier. La seconde opération possible sur le code d'un agent est de fournir des preuves de conformité. Cette méthode appelée **Proof Carrying Code (PCC)** [Necula et Lee, 1996, Necula et Lee, 1998] permet au concepteur d'un agent, d'y inclure des fonctions d'évaluation qui seront exécutées une fois l'agent arrivé sur un site destination. Généralement ces fonctions permettent de retourner le niveau d'autorité et les privilèges accordés à l'agent. Si les preuves de l'agent ne correspondent pas à la politique de sécurité mise en place sur un site, ce site, peut décider de ne pas exécuter l'agent.

Du fait de sa mobilité un agent est sensible aux attaques émanant d'un site malveillant. Il est alors nécessaire de pouvoir garantir la confidentialité et l'intégrité des éléments internes à un agent mobile. Ainsi un site malveillant peut par exemple regarder le contenu d'un agent afin d'en extraire des informations importantes. Il peut être tentant également pour un site malveillant de modifier le comportement d'un agent afin de récupérer des informations issues du futur parcours de l'agent. Nous pouvons trouver des techniques de réputation permettant d'évaluer le comportement de sites dans un réseau. Dans [Songsiri, 2006] Suphithat Songsiri présente **MTrust**, un modèle basé sur la réputation permettant des interactions plus sûres entre les agents et les sites. Pour décider de son futur déplacement, un agent utilise une sélection de sites à visiter et une fonction permettant de calculer un niveau de confiance associé. Pour calculer le niveau de confiance l'auteur se base sur un réseau bayésien et sur les rapports d'expériences ou *feedbacks* des sites. Il existe également des mécanismes d'authentification reposant sur des méthodes cryptographiques pour s'assurer du comportement d'hôtes distants. Dans [van 't Noordende *et al.*, 2009], le mécanisme permettant de s'assurer du comportement de sites distants repose sur un schéma de communications sécurisées. Il est possible d'inclure à leur architecture agent, un schéma d'authentification à base de clés publiques entre les sites. Une fois un site authentifié au près d'un autre, les transactions pour envoyer/recevoir un agent peuvent alors se dérouler.

### 3.3.2 Plate-formes agents mobiles

Il existe un grand nombre de plate-forme pour le développement et le déploiement d'agents mobiles. La majeure partie repose sur le langage Java. Ceci n'est pas un hasard. En effet Java de part sa conception et son recours à une machine virtuelle, permet une grande adaptabilité des applications dans un parc de machines hétérogènes. Il est donc en théorie possible pour un agent mobile de se déplacer sur une machine dotée d'un système d'exploitation différent de celui nécessaire à sa conception. Nous nous attacherons ici à présenter les plates-formes les plus connues et les plus utilisées reposant sur langage d'Oracle.

#### JADE

JADE (Java Agent DEvelopment Framework) est un intergiciel permettant une programmation multi-agents. Comme son nom l'indique JADE est développé en Java. La principale caractéristique de JADE est son respect de la norme FIPA [Bellifemine *et al.*, 1999]. JADE offre en plus une interface graphique pour faciliter le développement et l'administration.

L'élément central dans JADE est le *container*. Un container désigne un site exécutant une machine virtuelle Java. La plate-forme JADE correspond alors à un ensemble de containers reliés à un *main-container*. Il est possible de relier plusieurs plate-forme JADE entre elles en établissant un lien entre main-container. Toutefois ce lien autorise uniquement l'envoi de messages entre main-container, et ne permet en aucun cas la migration d'agents entre plate-forme.

La communication entre agents se fait par l'envoi de messages asynchrones respectant la norme ACL. Ce langage entre agents, dont la syntaxe est similaire à KQML, permet une définir :

- un ensemble d'actes de communication de base.
- une sémantique détaillée de ces primitives, définies par une ontologie.
- un ensemble de messages prédéfinis que tout agent doit être en mesure de traiter.

JADE propose également un service de pages jaunes permettant le référencement des services fournis par les agents. L'utilisation des ontologies présentes dans les messages permet une description de ces services.

L'un des atouts de JADE est la communauté active fournissant une aide non négligeable aux développeurs. Cependant le caractère centralisé de JADE, de par l'utilisation de main-container, et la mobilité limitée des agents aux liaisons inter plates-formes font que JADE ne possède pas les caractéristiques nécessaires à la conception d'une application répartie, robuste et autonome.

#### Aglets

L'API Aglets [Lange *et al.*, 1997], développée initialement par IBM en 1997, puis maintenu par la communauté open source depuis 2001, est une des plates-formes agents mobiles les plus connues avec JADE. Une des force de Aglets est son respect du standard **MASIF** [Milojicic *et al.*, 1998]. MASIF est un ensemble de définitions et d'interfaces permettant une interopérabilité pour les SMA composés d'agents mobiles.

Aglets permet une gestion des messages synchrones et asynchrones, par une méthode dite de passage de message. Chaque agent, ou aglet, possède un gestionnaire de messages déterminant l'ordre de traitement des messages. Pour communiquer un agent s'adresse à un proxy. Ce proxy sert d'abstraction pour s'adresser à un agent distant. Ainsi un message est envoyé à un proxy qui se charge de transmettre le message au gestionnaire.

Les agents dans aglets sont vus comme de simples threads, ce qui peut poser des problèmes dans le traitement des messages. Un agent ne prend pas, par exemple, de messages entrants tant qu'il n'a pas fini l'exécution de sa tâche courante. De même deux agents se trouveront en situation de blocage mutuel, si ils s'envoient respectivement un message synchrone au même moment. Chaque agent n'est dès lors plus capable de traiter de messages entrant tant que l'autre agent n'a pas reçu son message. L'activité d'Aglets semble aujourd'hui au point mort, la dernière version du kit de développement datant de 2002.

### JavAct

JavAct [Arcangeli *et al.*, 2004] est un intergiciel Java pour la création et la programmation d'applications concurrentes, réparties et mobiles. JavAct a été développé à l'Institut de Recherche en Informatique de Toulouse (IRIT). Le but de JavAct est de fournir des mécanismes simples pour la programmation d'agents mobiles adaptables reposant sur la notion d'acteur. JavAct, reposant sur la bibliothèque Java standard, étend le paquetage RMI et permet une abstraction de la gestion des threads.

Ici les agents sont des entités communiquant par messages, dont le comportement décrit la réponse à adopter sur réception d'un message. Ainsi sur réception d'un message, un agent peut :

- créer dynamiquement de nouveaux agents,
- envoyer des messages aux acteurs qu'il connaît,
- changer de comportement.

Le fait de changer de comportement pour un agent correspond à définir le comportement à adopter pour le prochain message reçu. Le comportement est privé à chaque agent et contient les données et fonctions de l'acteur ce qui constitue son état. Ainsi concevoir un agent mobile avec JavAct consiste à programmer ses comportements.

Chaque acteur [Agha, 1986] possède une boite aux lettres qui contient les messages que l'agent reçoit. Cependant le problème de la mobilité des agents soulève le problème de l'acheminement des messages. Dans JavAct, quand un agent se déplace, le protocole suivant est respecté :

- l'agent crée sur le site courant un proxy dont le rôle est de relayer les messages,
- l'agent se déplace sur le site distant.

Cette méthode pose toutefois le problème de la fiabilité du fonctionnement des machines, et de la rupture de la chaîne de liens pouvant exister dans un réseau.

Enfin Javact ne fournit pas de mécanisme pour la validité des applications. Les auteurs soulèvent le problème des messages orphelins [Dagnat *et al.*, 2000] :

- un message est dit *orphelin de sûreté* quand un agent ne peut plus prendre en compte un message qu'il doit traiter après un changement de comportement.
- Un message est dit *orphelin de vivacté* quand un agent ne peut pas traiter un message, car celui-ci est bloqué dans l'attente d'un message qu'il n'a pas reçu.

Pour éviter de tomber dans ces cas d'erreur, il convient donc d'être prudent lors de la conception des comportements d'agent, et plus particulièrement sur le traitement des messages.

### Autres plates-formes

Etant donné qu'il existe un grand nombre de plates-formes agent mobile, et qu'elles ne sont pas obligatoirement basées sur le langage Java, nous allons ici simplement dresser la liste des plates-formes que l'on peut trouver en fonction de leurs caractéristiques. Le tableau 3.3 recense les plates-formes agents mobiles parmi les plus connues en fonction du langage utilisé, du type de communication adopté par les agents, et de la date de dernière version de la plate-forme. On

trouve également dans ce tableau les références essentielles des plates-formes. Cet état des lieux à été réalisé à partir de [Trillo *et al.*, 2007, Cubat dit Cros, 2005, Outtagarts, 2009].

Pour conclure ce chapitre Nous allons maintenant décrire le fonctionnement de deux applications mettant en jeu des agents mobiles ayant un comportement emprunté au monde animal, et évoluant dans un réseau informatique.

### 3.3.3 Agents mobiles et SMA dans un réseau P2P

#### Anthill - Partage de fichiers avec Gnutant

La première des applications que nous introduisons est **Anthill** [Özalp Babaoglu et Montresor, 2002]. Anthill est un intergiciel pour le développement d'applications pair-à-pair basées sur les sytèmes multi-agents et la programmation évolutionniste. L'architecture de anthill est composée de fourmis pouvant se déplacer sur un ensemble de nids interconnectés. Ici les nœuds du réseau (nids) sont le lieu d'exécution d'applications réparties, telles que le calcul distribué ou le partage de fichiers, et constituent une interface entre les utilisateurs et le réseau en mettant à disposition des services tels que la recherche de fichier par exemple.

Ces services sont représentés par des fourmis, agents autonomes capables de voyager dans le réseau. En réponse à une requête une ou plusieurs fourmis sont générées et assignées à une tâche particulière. Pour accomplir leur but, les fourmis peuvent interagir avec les nids visités. Un nid est composé de trois entités. Le **programmeur de fourmis** permet la création de fourmis suite aux requêtes de l'utilisateur. Les fourmis ainsi générées peuvent se déplacer de nids en nids jusqu'à ce qu'elles accomplissent leurs tâches. Le programmateur de fourmis fourni également un bac à sable pour limiter les ressources disponibles aux fourmis et interdire l'exécution d'actions potentiellement dangereuses (cf section 3.3.1). La **couche de communication** permet la découverte de nouveaux nids, de gérer la topologie du réseau, et les mouvements des fourmis entre nids. Les nids offrent leurs ressources aux fourmis les visitant par l'intermédiaire du **manager de ressources**. Chaque manager de ressources est associé à un ensemble de politiques pour gérer les ressources.

Une application de partage de fichiers appelée **Gnutant** à été développée en utilisant Anthill. Le but de Gnutant est la création d'un index distribué contenant les URL des fichiers partagés, et les informations de routage nécessaires à l'orientation des requêtes de recherche. Lors de l'insertion d'un nouveau fichier dans le répertoire partagé d'un utilisateur, une fourmi est émise pour prévenir les autres nids de la présence du nouveau fichier. Celle-ci contient l'identifiant du fichier, une URL, et une collection de mot-clés. La recherche de fichier engendre également la création d'une fourmi. À la différence des fourmis précédentes, celles-ci ne contiennent qu'un seul mot-clé. Ces fourmis exploitent les informations de routage laissées par les autres fourmis dans le but de trouver le plus court chemin menant à la localisation du fichier.

Le stockage des chemins dans Gnutant repose sur le routage de mot-clés hashés. Ici on associe à la valeur de hashage de chaque mot-clé un ensemble de nids pouvant héberger les fichiers associés. Ainsi, lors de la visite d'un nid, la fourmi de recherche regardera les chemins

| **Plate-forme** | **Langage** | **Communications synchrones/asynchrones** | **Mobilité** | **Dernière version** | **Références** |
|---|---|---|---|---|---|
| JADE | Java | Non/Oui | Faible | 2010 | [TILAB, 2000] [Bellifemine *et al.*, 1999] |
| Aglets | Java | Oui/Oui | Faible | 2002 | [Aglets, 1997] [Lange *et al.*, 1997] |
| JavAct | Java | Oui/Oui | Forte | 2008 | [JavAct, 2004] [Arcangeli *et al.*, 2004] |
| Telescript | Telescript | Oui/Non | Forte | 1997 | [White, 1994] |
| MOOREA | Java | Oui/Non | Forte | 2002 | [Dillenseger *et al.*, 2002] |
| TACOMA | Multiple | Oui/Oui | Faible | 2002 | [TACOMA, 1995] [Johansen *et al.*, 1995] |
| D'agents | Multiple | Oui/Oui | Forte | 2002 | [D'Agents, 1995] [Gray *et al.*, 2002] |
| ARA | Tcl | Oui/Non | Forte | 1997 | [ARA, 1997] [Peine et Stolpmann, 1997] |
| Voyager | Multiple | Oui/Oui | Forte | 2010 | [Voyager, 1997] |
| Grasshopper | Java | Oui/Oui | Faible | 2003 | [Grasshopper, 1995] |
| Tryllian | Java | Oui/Oui | Forte | 2005 | [Tryllian, 2005] |
| SPRINGS | Java | Oui/Oui | Faible | 2008 | [SPRINGS, 2008] [SPRINGS, 2008] |
| PIAX | Java | Oui/Oui | Faible | 2009 | [PIAX, 2009] |

TABLE 3.3 – Plates-formes agents mobiles

stockés pouvant être associés avec le mot-clé qu'elle transporte. Si une correspondance exacte est trouvée alors la fourmi choisira un nid parmi l'ensemble correspondant. Sinon un nid avec un mot-clé hashé le plus proche est choisi.

Les expérimentations menées portent sur le taux de réussite d'une recherche de fichier, et sur le nombre de sauts minimum pour trouver une ressource dans le réseau. Les résultats montrent dans le premier cas, que le taux de réussite augmente significativement dans le temps, et dans le second cas une chute du nombre de saut minimum dans le temps. Ces résultats montrent clairement l'apport des fourmis dans le routage de l'information. Il a été également montré dans [Montresor *et al.*, 2003] qu'à partir des mêmes fourmis, il est possible de leur donner un comportement leur permettant de répartir la charge du réseau équitablement entre les pairs.

**Ecureuils et peer-to-peer**

La seconde application est basée sur le comportement adaptif des écureuils [Camorlinga *et al.*, 2004]. La motivation de cette approche repose sur l'utilisation des DHT dans beaucoup d'applications P2P entraînant des inconvénients en terme d'allocation de ressources et de mauvaise utilisation des capacités de chaque pair. Pour répondre à cette problématique les auteurs ont proposé un système composé d'écureuils pouvant allouer des ressources en réponse à une demande, le but étant d'obtenir un équilibrage des ressources. Ici les demandes de ressources sont modélisées par des noisettes que les écureuils doivent stocker dans des caches représentant les ressources. Chaque nœud du réseau peut posséder plusieurs caches. Le comportement d'un écureuil peut être défini par :

- enterrer des noisettes dans une zone géographique proche de son nid,
- investir plusieurs nœuds du réseau aléatoirement avant de décider où mettre les noisettes,
- décider de ne pas stocker sa nourriture si d'autres écureuils sont autour, et
- travailler en petites équipes avec ceux qui lui sont familiers.

Le premier comportement d'un écureuil consiste à allouer des ressources proches de son nœud de départ. Un écureuil va donc prendre des noisettes sur son site de départ et va chercher aléatoirement un autre endroit (qui peut être le sien) pour les cacher. Si l'endroit choisi est plein (plus d'espace sur chaque cachette de l'endroit), alors l'écureuil va sur un autre endroit et regarde la disponibilité. Ce cycle se répète jusqu'a ce qu'un endroit convenable soit trouvé ou que l'écureuil meurt en cherchant un endroit.

Le second comportement est similaire au premier à la différence qu'un écureuil est capable de "renifler" plusieurs nœuds avant de se déplacer pour stocker les noisettes. Dans ce cas, la meilleure place sera choisie en fonction des performances locales. Si tout les endroits reniflés sont pleins, alors l'écureuil va sur une autre place. Ce cycle se répète jusqu'à ce qu'un endroit soit trouvé, ou que l'écureuil meurt.

Enfin le troisième comportement enrichi les algorithmes précédents en permettant aux écureuils de ne renifler que les endroits non visités par d'autres écureuils. La place choisie est la meilleure qui n'a pas été visitée par des étrangers. Si tous les endroits sont pleins ou visités par des

étrangers, alors l'écureuil va sur une autre place. Ce cycle se répète jusqu'à ce qu'un endroit soit trouvé, ou que l'écureuil meurt.

Les expérimentations permettent de tester les trois comportements. Ainsi les auteurs ont étudié le nombres d'étapes nécessaires pour arriver à une convergence du système, ainsi que l'écart type portant sur le pourcentage de noisettes stockées en fonction de la place dans les caches. Les résultats montrent que le second comportement est le plus performant, mais surtout que la convergence s'accélère en fonction de l'augmentation du nombre d'écureuils.

## Conclusion du chapitre

Nous avons présenté dans ce chapitre une des notions fondamentales en intelligence artificielle, la notion d'agent. Il se dégage d'un agent deux composantes essentielles, l'environnement et l'autonomie. Nous avons également étudié et défini ce qu'est un système multi-agent en nous intéressant aux notions d'interaction et d'organisation. À partir de ce cadre, nous avons pu porter notre attention sur le cas des SMA bio-inspirés et de leurs applications, notamment dans le cadres des réseaux pair-à-pair.

Nous avons également au cours de ce chapitre insisté sur la notion d'agent logiciel, et plus particulièrement sur le cas des agents mobiles. Dans le but de développer une application de gestion de l'information distribué reposant sur l'utilisation d'un SMA, nous avons fait un tour d'horizon des différentes plates-formes existantes. Leur grand nombre, leurs caractéristiques, l'activité de la communauté sous-jacente, ainsi que leur actualité récente rend ce choix extrêmement difficile.

# Conclusion de la partie I

Nous avons introduit dans cette partie les composantes majeures de notre travail. Dans un premier temps nous avons présenté les mécanismes nécessaires à l'obtention d'une couche réseau totalement décentralisée. Les réseaux pair-à-pair et plus particulièrement le domaine des protocoles épidémiques proposent des solutions pertinentes et efficaces pour la construction et le maintien d'un réseau décentralisé.

Dans un deuxième temps nous avons décrit les étapes de la vie d'une donnée. À la création d'une information, il est possible de lui appliquer une étape de fragmentation lui assurant ainsi sa disponibilité. Une fois fragmentée, l'information est disséminée dans le réseau. Il existe peu d'applications permettant un placement de l'information dynamique. Par dynamique nous entendons le fait de pouvoir déplacer l'information pour équilibrer la charge, ou pour améliorer sa disponibilité dans le temps. Dans beaucoup de cas le placement se fait en fonction de l'état du réseau, et non pas en fonction de son évolution. Il existe des solutions permettant de replacer les informations dans le réseau, mais ces solutions utilisent des serveurs créant dès lors des points de faiblesse évidents.

Pour pallier cette difficulté, nous nous sommes penchés sur le domaine de l'intelligence artificielle distribuée et plus particulièrement sur celui des systèmes multi-agents. Notre approche consiste à représenter l'information sous la forme d'un système multi-agents où chaque membre est un agent mobile. Chaque agent mobile contient un fragment d'information issu d'une étape de fragmentation. Pour proposer une répartition de charge efficace et décentralisée, nous utilisons les capacités de stigmergie des agents. Ainsi un agent est capable de marquer ses déplacements en déposant des phéromones dans le réseau. À partir de cette mesure, nous pouvons déterminer les zones du réseau ayant eu le moins de fréquentation.

Le second aspect emprunté au monde biologique que nous utilisons dans notre approche, est le comportement de groupe des oiseaux. Nous proposons d'intégrer à nos agents des règles de flocking, leur permettant de se déplacer sous la forme d'une nuée. Le but d'un tel comportement est de faciliter la recherche et la collecte d'information. Il suffit de contacter un des membres de la nuée pour retrouver l'ensemble d'une information.

# Deuxième partie

# Contributions

# Introduction de la partie II

Notre approche consiste à utiliser un système multi-agent composé d'agents mobiles pour la gestion de l'information dans un réseau. Dans notre modèle, chaque agent héberge une partie d'un fichier préalablement fragmenté et est capable de se déplacer de manière totalement autonome dans le réseau. L'objectif principal du recours aux agents mobiles et de pouvoir prendre en compte les modifications de l'environnement pour proposer un placement de l'information intelligent. Ainsi chaque agent peut observer le réseau et déterminer un site de stockage, indépendamment de tout élément extérieur à lui-même.

Dans un premier chapitre nous expliquerons comment représenter au sein d'une même architecture les différents aspects étudiés dans l'état de l'art. Ce chapitre détaillera la couche réseau nécessaire au bon fonctionnement d'un application répartie. Nous étudierons plus en détails les mécanismes assurant la construction et le maintien du réseau.

Dans un second temps, nous nous intéresserons aux agents composant notre système et présenterons notre gestion bio-inspirée de l'information. Nous présenterons le placement d'une donnée reposant essentiellement sur l'établissement de règles de flocking et sur les interactions avec l'environnement. À partir de ces déplacements nous proposerons des algorithmes dédiés à la répartition de charge, et à l'évaluation de la confiance dans le réseau.

Le troisième chapitre est consacrée à la recherche de l'information. Le but est d'étudier le comportement des techniques entrevues dans le chapitre 2 dans un contexte de forte mobilité. Nous avons vu que les méthodes à base de marche aléatoires étaient de manière générale plus performantes que les méthodes à base d'inondation. Nous présenterons donc deux algorithmes de recherche basé sur un parcours aléatoire du réseau. Le premier est une marche aléatoire classique. Le second algorithme utilise les capacités de stigmergie des agents pour marquer et parcourir aléatoirement le réseau.

Enfin nous terminerons cette partie en présentant une application déployée sur 100 machines physiques construite à partir de notre modèle. Ce chapitre aura pour but de sortir du cadre des simulations et de proposer des résultats permettant de valider nos résultats expérimentaux issus de diverses simulations.

# Chapitre 4

# Vers une gestion décentralisée de l'information à base d'agents

**Sommaire**



Ce premier chapitre a pour but d'introduire nos contributions et plus particulièrement de présenter l'approche retenue pour concevoir une application de stockage de fichiers totalement décentralisée. Dans une première partie nous modélisons le problème du stockage de données dans un réseau pair-à-pair. Une fois les composantes essentielles de notre modèle détaillées, nous nous intéressons à la gestion du réseau qui se trouve à la base de notre architecture. Enfin nous conclurons ce chapitre par l'évaluation des performances et caractéristiques de cette couche réseau.

## 4.1 Modélisation de notre approche

Notre objectif dans cette thèse est de proposer une application de stockage de fichier totalement décentralisée, reposant sur un réseau pair-à-pair autonome et utilisant les méthodes

issues des codes d'effacement pour garantir la sécurité et la confidentialité des données. A ces deux domaines, nous proposons d'encapsuler l'information fragmentée dans des agents mobiles capables de se mouvoir dans le réseau sans l'intervention d'un tiers.

La mobilité de ces derniers associée à leur comportement doivent nous permettre de proposer des mécanismes contribuant à garantir la sécurité et la sûreté de l'information et de proposer un placement de l'information autonome prenant en compte l'évolution de l'environnement.

C'est à partir de ces constats que nous avons proposé une modélisation de la gestion de l'information distribuée permettant de synthétiser ces objectifs [Pommier et Bourdon, 2009]. L'architecture souhaitée doit proposer :

- Un réseau autonome nécessitant un faible nombre d'opérations pour assurer son établissement et son maintien.
- Un mécanisme de fragmentation de l'information assurant la disponibilité et l'intégrité des données.
- Un placement des fragments totalement décentralisé prenant en compte les contraintes liées aux machines du réseau.
- Une méthode de recherche également décentralisée permettant de retrouver efficacement l'ensemble ou une partie des fragments nécessaires à la reconstruction d'une donnée.

Pour cela nous nous appuyons sur une structure en couches. La première couche de notre modèle concerne le réseau. Les composantes essentielles de cette couche doivent permettre une gestion du réseau efficace en terme de maintient de l'overlay et d'insertion et de départ de machines. La seconde couche doit fournir l'ensemble des services nécessaires pour assurer la disponibilité de l'information. Nous avons retenu un code de type Reed-Solomon pour assurer les étapes de fragmentation et de reconstruction des données.

Le rôle des agents mobiles est de proposer un placement de l'information qui soit le plus efficace possible et de fournir un mécanisme de recherche peu coûteux. L'ensemble des agents est contenu dans la troisième et dernière couche. À partir de cette couche les agents doivent être capable d'interagir avec le pair sur lequel ils se situent. La figure 4.1 montre une telle modélisation.

### 4.1.1 Couche réseau

Le but de la couche réseau est de gérer la topologie du réseau en communiquant avec les pairs. Pour cela nous nous appuyons sur un protocole permettant une diffusion épidémique efficace. Ce n'est donc pas à proprement parlé un protocole épidémique, mais plutôt un protocole établissant l'ensemble des propriétés nécessaires pour propager efficacement une rumeur dans un réseau.

L'ensemble des informations issues de cette couche sont stockées et partagées avec la couche application. C'est donc à partir des informations envoyées et reçues que chaque pair met à jour ses données. C'est ici que nous trouvons les fonctionnalités de connexion/déconnexion au réseau, et d'établissement pertinent d'un voisinage pour un nœud. L'ensemble de ces fonctionnalités sera décrit dans la suite de ce chapitre.

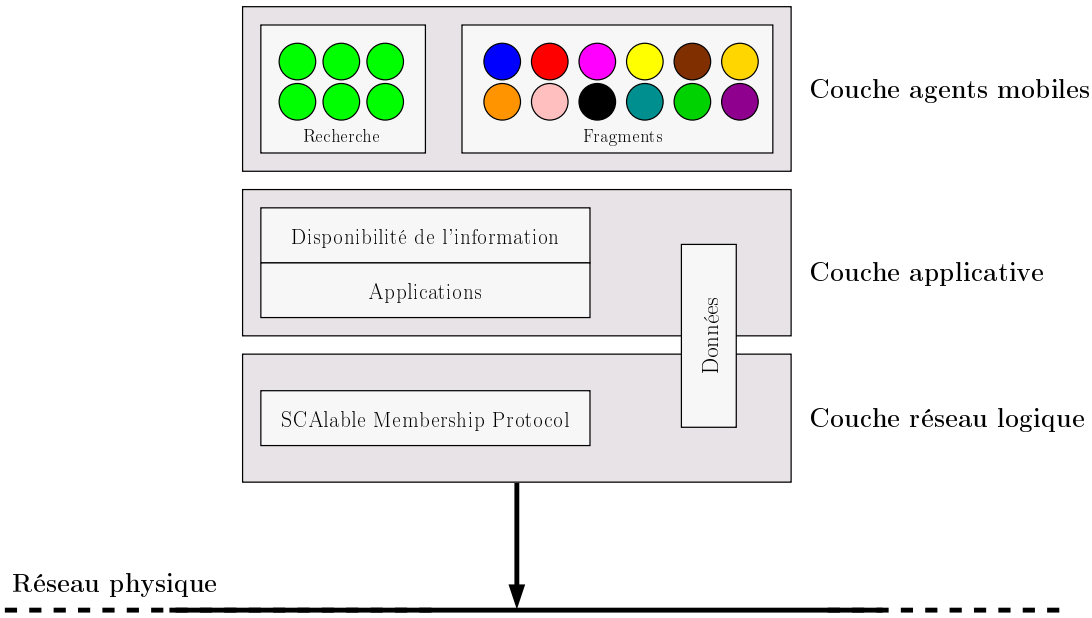



FIGURE 4.1 – Modélisation en couches de notre approche

### 4.1.2 Couche application

C'est ici que la disponibilité de l'information est assurée. La théorie des codes d'effacements et plus particulièrement un code de type Reed-Solomon (section 2.1.3) peut être utilisé. Avec un tel code, il est possible de partager un fichier en divisant un fichier en $k$ parts de taille identique, puis d'y ajouter $n$ fragments issus des $k$ fragments. La principale propriété de ces codes est de pouvoir reconstruire la donnée originelle à partir de $n$ fragments récoltés.

Les paramètres $k$ et $n$ sont des facteurs importants possédant une influence non négligeable sur la performance et la robustesse de l'architecture. Toutefois, cette thèse ne cherche pas à étudier comment fixer les paramètres $k$ et $n$. A partir de ce point nous considérons alors que la taille d'une nuée est la somme de $k$ et de $n$.

C'est également dans cette couche que nous trouvons diverses applications pouvant définir le comportement de chaque pair du réseau. Ainsi nous verrons dans le chapitre suivant comment nous pouvons définir des méthodes permettant de répartir la charge du réseau et d'établir un indice de confiance entre les pairs.

Le rôle de cette couche est également de stocker et de partager avec la couche réseau les informations relatives aux agents. Les informations issues de la couche réseau sont nécessaires pour nos différents algorithmes. Chaque pair va donc contenir :

- les informations relatives au réseau,
- les informations nécessaires aux déplacements des agents,
- la liste des agents présents.

### 4.1.3 Couche agents mobiles

On peut trouver deux types d'agents mobiles dans notre architecture. Les **agents fragments** transportant les fragments d'information et les **agents de recherche**. Les agents fragments servent au transport des données fragmentées issues de la couche application. Nous faisons correspondre alors à chaque agent un fragment. Le but de chaque agent fragment est de pouvoir se déplacer dans le réseau de manière autonome et d'ainsi adapter son placement en fonction de l'environnement.

Pour pouvoir gérer les déplacements de nos agents de manière décentralisée nous avons choisi de maintenir une cohésion entre les agents en appliquant les règles de flocking énoncées par Reynolds [Reynolds, 1987]. Ainsi nos agents sont regroupés sous la forme d'une nuée se déplaçant dans un réseau. Chaque agent possède également la capacité de se déplacer de pair en pair en déposant des phéromones. Cette approche doit nous permettre de proposer un placement de l'information prenant en compte l'évolution de l'environnement, et plus particulièrement la charge réseau ainsi que les pannes ou fautes pouvant survenir, tout en proposant une recherche des fragments efficace.

Les agents de recherche permettent de localiser un fichier en vue d'être consulté. Leur objectif est de trouver quelques membres de la nuée recherchée pour déplacer l'ensemble du nuage sur le site ayant effectué la recherche. Pour cela une méthode de marche aléatoire, ou chaîne de Markov, peut être utilisée.

Les capacités et propriétés des agents fragments et de recherche seront abordées dans le chapitre suivant.

## 4.2 Couche réseau

Le but de cette couche est de fournir un réseau logique superposé au réseau physique qui soit le plus léger possible, autonome, et complètement distribué. Les principales opérations d'une telle couche concerne la construction et le maintient du réseau logique. Nous avons vu dans le chapitre 1 que différentes solutions existent pour mettre en place un overlay de machines. L'étude des solutions à base de protocoles épidémiques nous a conduit à nous intéresser à SCAMP [11] [Kermarrec *et al.*, 2003, Ganesh *et al.*, 2003]. SCAMP est un protocole de construction de groupe multicast permettant le support de la diffusion épidémique. Ce protocole construit un graphe aléatoire orienté, tolérant aux fautes.

Ici la tolérance aux fautes est exprimée en terme de connexité du graphe, ce qui signifie que le réseau est considéré comme sûr si il reste connexe dans le temps. Les résultats présentés dans [Kermarrec *et al.*, 2003] montrent qu'il existe les mêmes seuils de connexité sur les graphes aléatoires orientés et les graphes aléatoires non orientés. Erdös et Rényi [Erdös et Rényi, 1960] ont montré qu'un graphe aléatoire non-orienté est connexe avec une probabilité $p = e^{-e^{-c}}$ s'il possède un degré $d$ égal à $\log(n) + 2c$, pour $c$ une constante fixée, et $n$ le nombre de sommets.

11. SCAlable Membership Protocol

Ici nous allons voir que le degré du graphe construit par SCAMP converge vers $(c+1)\log n$ avec $c$ une constante de tolérance aux fautes et $n$ le nombre de pairs présents dans le graphe. A partir de ce degré, le graphe généré reste connexe si la probabilité de perte d'arc reste inférieure à $\frac{c}{c+1}$. Bien qu'emprunté au domaine du multicast, SCAMP construit un réseau pair-à-pair qui est alors vu comme un graphe non orienté par la couche agent.

Ici le voisinage d'un pair est exprimé en fonction de sa **PartialView** ($PV$) et de sa **InView** ($IV$). Ces deux vues permettent de stocker dans le cas de la $PV$, l'ensemble des successeurs d'un pair (arcs sortants), et dans le cas de l'$IV$, l'ensemble des prédécesseurs d'un pair (arcs entrants).

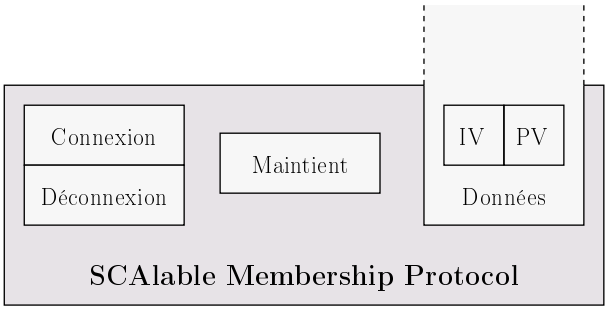



FIGURE 4.2 – Couche réseau logique

La figure 4.2 recense les éléments essentiels du protocole SCAMP à savoir l'insertion et le départ d'un pair et le maintien de la connexité du réseau.

### 4.2.1 Arrivée d'un pair

Lorsqu'un pair $p$ souhaite rejoindre le réseau, il doit tout d'abord contacter un nœud jouant le rôle de passerelle. Le choix de ce pair, appelé *contact*, est crucial pour le respect des garanties de convergence du graphe [Ganesh *et al.*, 2003]. Le mécanisme d'**indirection** proposé par les auteurs, permet de parcourir une chaine de Markov à partir d'un point d'entrée du réseau jusqu'au pair *contact*.

#### Construction de la chaîne de Markov

Pour entrer dans le réseau, $p$ va envoyer un message de type `subscribe`. Ce message va être propagé jusqu'au pair *contact*. Le message `subscribe` va passer d'un pair $i$ à un pair $j$, tel que $j \in PV_i$, avec une probabilité $w_{ij}$. Nous avons donc pour tout $i$ :

$$\sum_{j \in PV_i} w_{ij} = 1 \tag{4.1}$$

Dans l'équation 4.1, pour un pair $i$, la somme des poids de toutes les arêtes sortantes est égale à 1. Chaque pair possède également une condition similaire sur ses arêtes entrantes. Nous

avons donc alors :

$$\sum_{i \in IV_j} w_{ij} = 1 \tag{4.2}$$

Une matrice $W$ de poids $w_{ij}$ satisfaisant l'équation 4.1 est appelée matrice stochastique. Si elle satisfait également l'équation 4.2 alors $M$ est appelée doublement stochastique. Si une matrice irréductible est doublement stochastique alors, une chaîne de Markov associée à la matrice de transition $M$ possède une distribution uniforme comme unique distribution stable.

La matrice de transition est répartie sur chaque pair qui sont alors vu comme les états de la chaîne de Markov. Les pairs associent donc à chaque extrémité d'arc un poids. Par exemple pour un pair $i$, $w_{ij}$ (resp. $w_{ji}$) correspond au poids associé à l'arc $(i, j)$ (resp. $(j, i)$). Pour que les poids de chaque extrémité des arcs soient identiques, chaque pair va périodiquement normaliser les poids associés aux pairs connus puis leurs propager cette valeur. Ce mécanisme de mise à jour permet de résoudre le problème lié à l'arrivée d'un nouveau nœud. En effet quand un pair $j$ arrive dans la $PV$ (resp. $IV$) d'un pair $i$, alors $i$ initie la valeur de $w_{ij}$ (resp. $w_{ji}$) à la moyenne des poids de cette vue.

### Parcours de la chaîne de Markov

Une fois la chaîne de Markov construite, $p$ peut donc faire une demande d'insertion par un message `subscribe` auprès d'un pair $i$ responsable de l'initialisation du marcheur aléatoire. Le nombre de sauts du marcheur est fixé à $2 * Card(PV_i)$ et est décrémenté après chaque saut. Un pair $i$ choisit alors un pair $j$ à qui propager la demande avec une probabilité $w_{ij} = \frac{w_{ij}}{w_{out}}$ avec $w_{out} = \sum_{j \in PV_i} w_{ij}$.

Cette étape est répétée jusqu'à ce que le nombre de saut du marcheur soit nul. Une fois la marche terminée, le dernier nœud parcouru est le pair *contact*.

### Insertion du pair

Une fois le pair *contact* trouvé, $p$ doit construire son voisinage, à savoir ses tables $IV$ et $PV$, tout en respectant les propriétés de SCAMP. La première étape pour $p$ consiste à ajouter le pair *contact* parmi ses arcs sortant ($PV$). Puis $p$ demande au pair *contact* de propager son arrivée par l'intermédiaire de messages `forward`. L'objectif des messages `forward` est de créer le nombre exact d'arcs au départ de $p$ pour maintenir les propriétés de connexité de SCAMP.

La première diffusion de messages `forward` est effectuée sur les voisins du pair *contact*. Sur réception d'un tel message, un nœud $x$ va ajouter $p$ à sa $PV$ avec une probabilité $p = \frac{1}{Card(PV_x)+1}$. Dans le cas ou $p$ n'est pas ajouté, le message est propagé à un voisin choisi aléatoirement. Cette première diffusion permet d'atteindre une convergence moyenne du degré de $p$ vers $\log(n)$. Une seconde diffusion de message `forward` est effectuée par le pair *contact*. Celle-ci est effectuée sur $c$ voisins pris aléatoirement parmi $PV_{contact}$. Ainsi le degré de $p$ converge vers $(c + 1)\log(n)$.

### 4.2.2 Départ d'un pair

Dans le cas d'un départ planifié d'un pair $p$ (qui n'est donc pas la conséquence d'une faute), il est nécessaire d'avoir un mécanisme permettant de reconnecter certains éléments de $IV_p$ avec certains éléments de $PV_p$. Ce mécanisme doit s'assurer que le degré de chaque nœud reste constant [Ganesh *et al.*, 2003].

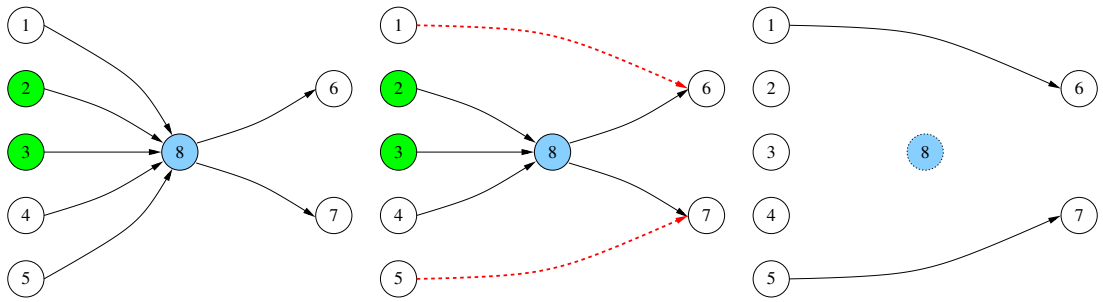
Lors de la déconnexion un pair $p$ va choisir un sous-ensemble de nœuds à déconnecter $P \subseteq IV_p$ tel que :

$$Card(P) = \begin{cases} c+1 & siCard(IV_p) \geq c+1 \\ Card(IV_p) & sinon \end{cases} \tag{4.3}$$

D'après l'équation 4.3, il y a donc au plus $c+1$ pairs à déconnecter. Si $P^c$ est le complémentaire de $P$ dans $IV_p$ alors il est nécessaire de reconnecter les éléments d'un sous-ensemble $Q_1 \subseteq P^c$ avec les éléments d'un sous-ensemble $Q_2 \subseteq PV_p$ tel que :

$$Card(Q_1) = Card(Q_2) = min(Card(P^c), Card(PV_p)) \tag{4.4}$$

La figure 4.3 montre un tel processus en 3 étapes. Dans la figure 4.3(a), le pair 8 choisi $c+1$ pairs présents dans sa table $IV$ à déconnecter. Ici ce sont les nœuds 2 et 3. Dans la figure 4.3(b), le pair 8 reconnecte autant de voisins que possible. Ici nous avons $Card(PV) < Card(IV) - (c+1)$ donc $Card(Q_1) = Card(Q_2) = Card(PV)$. Par conséquent $Q_1 = \{1, 5\}$ et $Q_2 = \{6, 7\}$. Enfin dans la figure 4.3(c), le nœud 8 se déconnecte des pairs 2, 3, et 4 en leur envoyant un message de déconnexion.




(a) Choix des pairs à déconnecter (b) Reconnection de pairs (c) Déconnecxion du pair 8

FIGURE 4.3 – Processus de déconnexion du pair 8 avec $c = 1$

### 4.2.3 Auto-réparation

Le graphe construit par le protocole SCAMP est connexe. Il est cependant possible qu'à la suite d'une faute ou d'une panne, un pair $p$ se retrouve isolé. Tout les éléments de $IV_p$ sont alors inaccessibles et $p$ ne peut plus recevoir d'information. La détection de l'isolation est faite en ayant recours à un mécanisme similaire au ping. Périodiquement, tout nœud va envoyer un message aux éléments de sa $PV$. Ceci permet aux pairs de connaître l'état de leurs prédécesseurs.

Les pairs ayant reçus un tel message, répondent en envoyant un message contenant leur état. Lorsqu'un nœud s'aperçoit qu'il est isolé, il se déconnecte de sa $PV$ et initialise une nouvelle souscription au réseau SCAMP depuis une porte d'entrée bien connue ou d'un élément de son ancienne $PV$.

## 4.3 Expérimentations sur la couche réseau

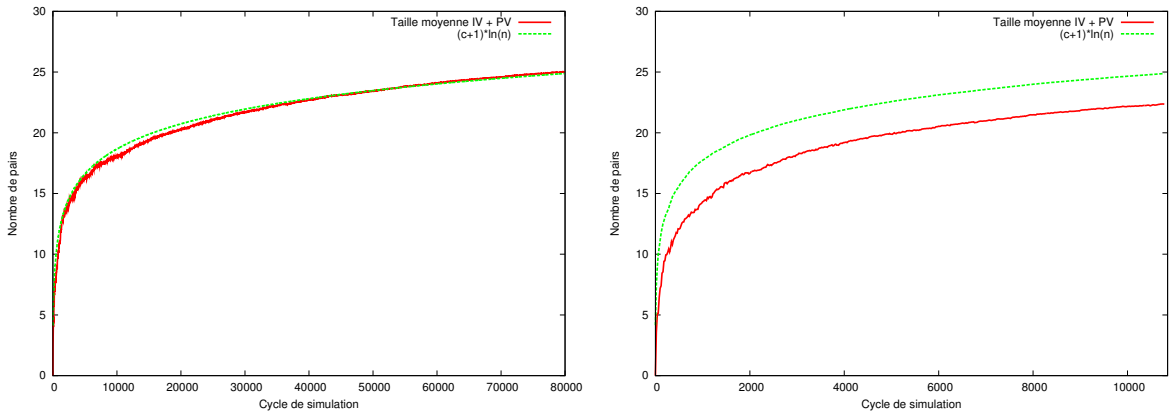
Benoît Romito [Romito, 2009] a pu valider les comportements attendus de la couche réseau. Pour y parvenir, SCAMP a été implémenté sur le simulateur oRis [Harrouet, 2000]. Ces expérimentations portent :

- sur la convergence du degré moyen de chaque pair,
- sur la convergence du poids de chaque pair,
- et sur les conséquences liées au départ de plusieurs nœuds.

### 4.3.1 Convergence du degré moyen

La première série d'expériences consiste à observer le comportement de chaque pair du réseau au cours du temps. Ainsi nous avons mesuré le degré moyen de chaque nœud lors de la construction d'un réseau de 4000 machines. Nous avons également fait varier le délai de connexion entre deux pairs. Le but d'une telle variation dans le temps est de montrer l'influence de multiples connexions simultanées sur la construction du réseau.

La figure 4.4 décrit le degré moyen des pairs du réseau pour une constante $c$ fixée à 2. Le graphique 4.4(a) montre le degré moyen avec une connexion espacée dans le temps. Le graphique 4.4(b) décrit le degré moyen des pairs du réseau avec un délai raccourci entre deux arrivées de pairs. Ici un temps de connexion espacé correspond à la connexion d'un pair tous les 20 cycles de simulation et un temps raccourci correspond à la connexion d'un pair tous les 3 cycles de simulation.



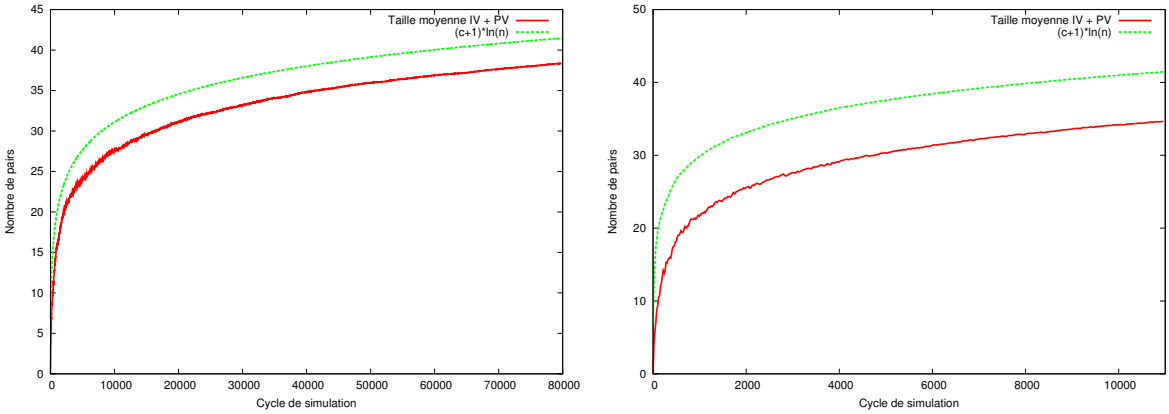

(a) Connexion tous les 20 cycles

(b) Connexion tous les 3 cycles

FIGURE 4.4 – Convergence du degré moyen de chaque pair avec c = 2

Nous pouvons observer sur le graphique 4.4(a) une convergence du degré moyen vers la valeur de $(c+1)\log(n)$. Cette convergence valide bien le comportement attendu de l'algorithme SCAMP. La figure 4.4(b) décrit la convergence du degré moyen de chaque pair lorsque les connexions successives des pairs se font plus rapidement. Ici les résultats sont moins nets que précédemment, et la convergence n'est plus aussi flagrante que sur la figure 4.4(a).

Les graphiques 4.5(a) et 4.5(b) montrent les résultats concernant un réseau construit avec une constante $c$ équivalente à 4. Dans le cas où les connexions sont espacées dans le temps, la convergence vers $(c+1)\log(n)$ se fait. Dans le cas où les connexions sont rapprochées nous nous retrouvons dans le cas similaire à $c = 2$ où la convergence n'est pas aussi nette.




(a) Connexion tous les 20 cycles

(b) Connexion tous les 3 cycles

FIGURE 4.5 – Convergence du degré moyen de chaque pair avec c = 4

Ces résultats peuvent s'expliquer par le fait que le nombre d'arcs créés est dépendant du nombre d'arcs précédents. La preuve de la convergence du degré de chaque nœud a pour origine l'équation suivante où $E[M_n]$ désigne le nombre d'arcs du réseau quand $n$ machines sont connectées :

$$E[M_n] = E[M_{n-1}] + \frac{E[M_{n-1}]}{n-1} + c + 1 \tag{4.5}$$

Dans l'équation 4.5 le nombre d'arêtes espérées quand $n$ machines sont connectées, représenté par $E[M_n]$, est égal à la somme des arêtes au rang précédent ($E[M_{n-1}]$) avec le degré moyen attendu pour un pair $\frac{E[M_{n-1}]}{n-1} + c + 1$.

En cas de connexion rapprochée de plusieurs pairs, de nombreux messages `forward`, responsables de l'insertion des pairs dans le réseau, sont propagés. Il est alors possible qu'un pair rejoigne le réseau alors que des précédents messages `forward` n'ont pas fini leur parcours. Le nombre d'arcs espérés n'a alors pas la bonne valeur ce qui propage une erreur au cours du temps. Ce cas de figure n'est pas si rare. Prenons le cas d'une diffusion en streaming d'un événement à un horaire donné. Les clients du service vont se connecter en masse peu avant l'horaire de l'évènement, ce qui nous ramène au cas de figure précédent.

### 4.3.2 Convergence du poids

La figure 4.6 montre la convergence de la somme des poids $w_{ij}$ et $w_{ji}$ de chaque pair. La courbe représentant la somme des $w_{ji}$ est superposée à la courbe représentant la somme des $w_{ij}$. Nous pouvons remarquer ici que les deux sommes convergent bien vers 1 ce qui valide bien le comportement du mécanisme de indirection dans le réseau. De plus cette convergence est obtenue rapidement, la somme des poids oscillant faiblement autour de la valeur 1 en environ 1000 cycles de simulation.

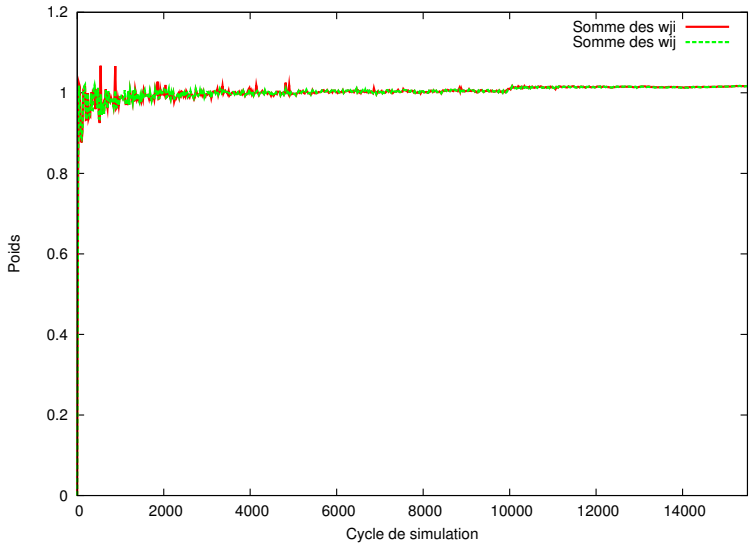



FIGURE 4.6 – Somme des poids de chaque pair

### 4.3.3 Départ de plusieurs pairs

Enfin la dernière série de simulations concerne la déconnexion simultanée d'un grand nombre de pairs du réseau. La figure 4.7 montre les résultats obtenus dans un réseau constitué de 500 nœuds dans lequel une perte de 290 pairs a été introduite à un temps équivalent à 10000 cycles de simulation. Ici la courbe pointillée verte représente le degré vers lequel la taille des vues de chaque pair doit converger, la courbe pleine rouge la taille moyenne des vues de chaque pair, et la courbe pointillée bleue le seuil limite de connexité. Nous pouvons remarquer que la perte simultanée de plus de la moitié des pairs du réseau ne permet pas de franchir le seuil de connexité. La proportion théorique de nœuds que l'on peut déconnecter dans un tel réseau est égale à $\frac{2}{3} * 500 \approx 333$. Cependant, nous observons que la convergence réelle du degré a été plus faible que la valeur théorique de $(c+1)\log(n)$. il ne faut donc pas déconnecter 333 pairs mais moins. Avec une perte de 290 machines le réseau reste connexe et résiste bien à une perte de nœuds supérieure à la moitié des machines connectées.

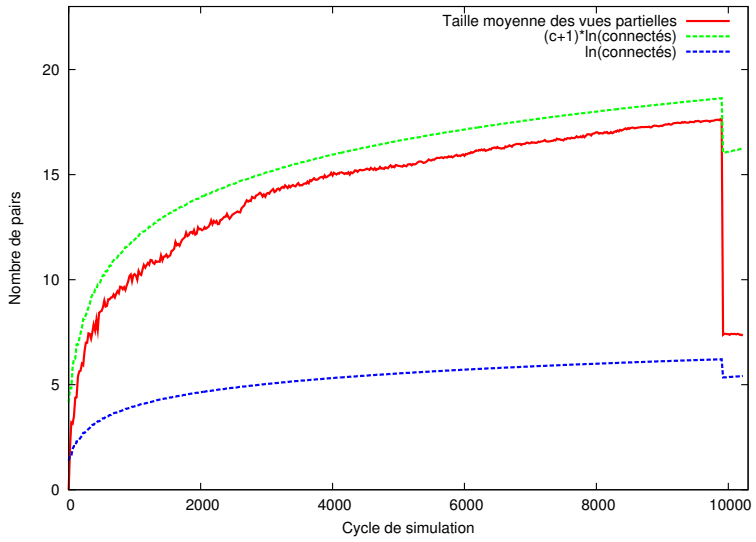



FIGURE 4.7 – Perte de 290 pairs dans un réseau de 500 pairs

## Conclusion du chapitre

Ce chapitre nous a permis de poser les bases nécessaires pour proposer une application de stockage de fichiers construite à partir d'agents mobiles. Nous avons défini une modélisation en couche et nous avons décrit la couche fondamentale à savoir la couche réseau. Pour valider notre modèle nous avons implémenté un réseau de type SCAMP, et nous avons vérifié son comportement à partir de simulations. Les résultats montrent les convergences attendues pour le degré moyen et pour la somme des poids des vues de chaque pair sous l'hypothèse que les connexions des nœuds soient suffisamment espacées dans le temps.

Dans notre cas, cela peut ne pas être dérangeant. En effet, à partir de ce point nous allons considérer que notre modèle prend place dans un monde fermé et non pas ouvert comme peut l'être de la diffusion en streaming d'un évènement, le but étant de faire évoluer nos agents dans un réseau préalablement construit où le degré de chaque pair est déjà établi.

# Chapitre 5

# Placement bio-inspiré de l'information

**Sommaire**



Nous avons vu dans le chapitre 2 que le placement de l'information dans les réseaux pair-à-pair ne propose pas de mécanismes permettant une adaptation dynamique aux conditions changeantes de l'environnement. Par conditions changeantes, nous entendons pannes machines ou comportements non-attendus. Notre approche consiste à utiliser des agents mobiles, capables de se déplacer en toute autonomie, pour héberger des fragments d'information. Pour faciliter la recherche de fichiers tout en conservant leur degré d'autonomie, nous utilisons des règles de flocking. Ce comportement issu du monde animal repose sur le déplacement en groupe des oiseaux. Nous en présentons une adaptation aux réseaux pair-à-pair. Le but est de maintenir les agents sous la forme d'une nuée, permettant ainsi d'en faciliter la collecte. En effet il suffit de contacter un ou plusieurs membres du groupe d'agents pour accéder à l'ensemble des fragments constituant un document. Nous verrons au cours de ce chapitre un second aspect emprunté aux animaux. Nos agents sont également capables de déposer des phéromones lors de leurs

déplacements. Cela nous permet d'enrichir le comportement des agents pour dans un premier temps répartir efficacement la charge dans le réseau, et dans un second temps proposer un algorithme évaluant le comportement des pairs dans le réseau.

## 5.1 Règles de flocking

Pour décentraliser une application de stockage de donnée, nous allons utiliser un système multi-agents où les fragments d'information sont des agents mobiles cognitifs, capables de prendre leurs propres décisions. Cette cognition doit permettre à nos agents de choisir le meilleur endroit de stockage dans le réseau. Il est toutefois nécessaire de "contrôler" la mobilité de nos agents. Pour cela nous allons utiliser des règles de flocking, semblables à celles proposées par Craig Reynolds [Reynolds, 1987], s'inspirant du déplacement en nuée des oiseaux. La motivation de Reynolds était de trouver des règles simples que chaque agent pouvait suivre pour reproduire le vol en formation des oiseaux. Trois règles ont été identifiées :

1. **Cohésion :** les agents veillent à ne pas être trop éloignés de leur voisinage.
2. **Séparation :** les agents veillent à ne pas être trop proches de leur voisinage.
3. **Alignement :** les agents gardent la même direction que leur voisinage.

Dans ce modèle, le flocking n'est pas propre à chaque agent mais est un comportement émergeant des interactions entre les membres du groupe. Ces règles mettent en évidence deux notions essentielles, à savoir le voisinage et la distance.

### 5.1.1 Notion de voisinage

Nous avons étudié au cours du chapitre précédent la construction du réseau logique composant notre architecture. Cette construction est basée sur deux vues, la $InView$ et la $PartialView$, notées respectivement $IV$ et $PV$. Dans le cas de nos agents, nous considérons que ces deux vues composent le voisinage d'un agent. Nous avons donc la définition du voisinage suivante :

**Définition 12** *Le voisinage d'un pair $x$ nommé $V_x$ est l'union de la PartialView et de la InView de $x$ soit : $V_x = PV_x \cup IV_x$.*

### 5.1.2 Notion de distance

Pour appliquer les règles de flocking précédemment décrites, il est nécessaire d'avoir une estimation de la distance entre deux pairs du réseau. Il est possible d'utiliser une métrique similaire à celle utiliser dans Kademlia [Maymounkov et Mazières, 2002] basée sur l'opérateur XOR.

**Définition 13** *La distance $d$ entre un pair $x$ et un pair $y$ est donnée par : $d(x, y) = x \oplus y$.*

Cette métrique possède les propriétés suivantes en plus de l'inégalité triangulaire :

- $d(x,x) = 0$,
- $d(x,y) > 0$ si $x \neq y$,
- $\forall x, y : d(x,y) = d(y,x)$
- $\forall x, \forall \Delta > 0$, il y a un et un seul $y$, tel que $d(x,y) = \Delta$.

La dernière propriété, appelée unidirectionalité, permet de s'assurer que toutes les requêtes dans le réseau vers un nœud $y$ convergeront vers le même pair, indépendemment du nœud émetteur de la requête.

**Exemple 1** *Soient deux pairs* $p_1$ *d'idenifiant* $id_{p_1} = 1001$ *et* $p_2$ *d'identifiant* $id_{p_2} = 1011$. *La distance* $d(p_1, p_2)$ *est donnée par :* $d(p_1, p_2) = id_{p_1} \oplus id_{p_2} = 1001 \oplus 1011 = 0010$. *Il existe donc un seul et unique pair situé à une distance* 0010 *de* $p_1$, *qui est* $p_2$.

Dans une telle métrique la localité n'est pas prise en compte. Deux pairs proches logiquement peuvent ne pas l'être physiquement. Il est possible de baser la mesure de distance sur la latence existante entre deux pairs du réseau. Le calcul de la latence entre deux pairs s'obtient par le calcul du temps d'aller-retour que met un paquet pour transiter entre deux machines. Cette mesure est appelée round-trip delay time (RTT) [12].

Le RTT est défini comme le temps de transit d'un signal dans un circuit fermé. La figure 5.1 montre l'établissement d'un RTT pour deux pairs $A$ et $B$.

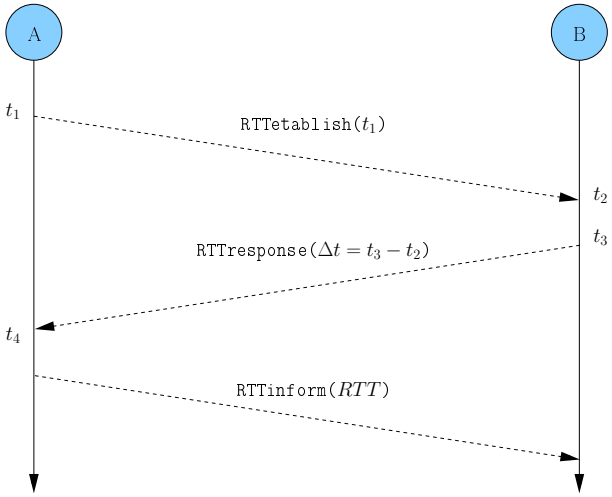



FIGURE 5.1 – Etablissement d'une RTT entre $A$ et $B$

$A$ initie une demande de RTT à destination de $B$. $A$ sauvegarde le temps $t_1$ auquel la demande est émise. Une fois la demande reçue par $B$ au temps $t_2$, $B$ envoie au temps $t_3$ à $A$ le temps $\Delta t = t_3 - t_2$ mis pour traiter la demande. Une fois le message reçu au temps $t_4$, $A$ calcule $RTT = t_4 - t_1 - \Delta t$, le temps mis par le message pour effectuer un aller-retour entre $A$ et $B$.

12. Le RTT entre deux pairs n'est pas une métrique et viole dans certaines conditions l'inégalité triangulaire. Toutefois cette incidence n'a pas été étudiée et ne semble pas porter à conséquence sur notre modèle.

### 5.1.3 Algorithme de déplacement

Après avoir défini ce qu'est un voisinage pour un agent, et la représentation de la distance que nous utilisons, nous allons adapter les règles de flocking à notre modèle. Pour que les agents se déplacent sous la forme d'une nuée ils doivent respecter les règles suivantes :

**Règle 1** *Un pair ne peut stocker qu'un seul agent par document (règle de séparation).*

**Règle 2** *Les agents se déplacent dans le but de se rapprocher des agents les plus éloignés de leur voisinage (règle de cohésion associée à une distance métrique).*

La première condition se rapporte à la règle de séparation des règles de Reynolds. Ici, il est impossible d'avoir deux agents sur un même pair du réseau [13]. La seconde condition porte sur la règle de cohésion. Les agents ont pour objectif de se rapprocher des agents observés les plus éloignés. Il est donc nécessaire d'avoir une estimation de la distance à laquelle les agents se situent. Pour les raisons de localité précédentes nous utilisons une distance basée sur le RTT.

A partir des règles précédentes, nous avons établi un algorithme permettant à un groupe d'agent l'exécutant en parallèle de former une nuée [Pommier et Bourdon, 2009]. Cet algorithme permet à un fragment $f_{doc}$ issu de la fragmentation d'un document *doc* de se déplacer d'un pair $x$ vers un pair $p$ appartenant au voisinage $V_x$. Dans cet algorithme le déplacement des agents se fait en fonction d'une distance d'éloignement entre deux pairs. Cette distance $\lambda$ nous permet de détecter les éventuelles violations des règles. Nous pouvons alors donner la définition suivante :

**Définition 14** *Soit $\lambda$ la distance d'écartement entre deux pairs $x$ et $y$ pour appliquer les règles de flocking. Au delà de cette valeur la règle de cohésion est violée, en deçà la règle de séparation est violée.*

La première étape de l'algorithme 2 consiste pour un agent à interroger le pair sur lequel il se situe pour pouvoir déterminer quelles sont ses possibilités de déplacement. Pour cela nous construisons deux ensembles, la liste des pairs possédant des fragments issus du même document que $f_{doc}$ appelée *PairsOccupés*, et une liste complémentaire de la première nommée *PairsLibres*. L'établissement de la liste des pairs susceptibles d'héberger $f_{doc}$ est alors obtenue en retirant de $PairsLibres$ les nœuds violant la règle de cohésion avec les pairs contenus dans $PairsOccupes$. La dernière étape de l'algorithme consiste à choisir un site de déplacement parmi les pairs restants.

## 5.2 Dépôt de phéromones et applications

Le vol des oiseaux n'est pas le seul aspect inspiré par les sciences de la nature et de la vie que nous prenons en compte dans notre modèle. Nous nous intéressons également à la capacité d'interaction avec leur environnement des fourmis, appelée stigmergie. Ce mécanisme permet

13. Cette règle pourrait être assouplie en particulier dans le cas de répliques de fragments

**Algorithme 2** : Déplacement d'un fragment $f_{doc}$ d'un pair $x$ vers un pair $p$

**Entrées** : Un pair $x$, $V_x$ le voisinage de $x$, $doc$ un document, $f_{doc}$ un fragment $\in doc$

**Sorties** : Un pair $p$ où stocker $f_{doc}$

```
// Récupération des pairs occupés
```
$PairsOccupés \leftarrow$ pairs $\in V_x$ possédant des fragments $\in doc$;

**pour chaque** $y \in PairsOccupés$ **faire**

```
    // Suppression des pairs violant la règle de cohésion
```
    **si** $d(x, y) < \lambda$ **alors**
        Retirer $y \in PairsOccupés$;

```
// Récupération des pairs libres
```
$PairsLibres \leftarrow$ pairs $\in V_x$ possédant des fragments $\notin doc$;

**pour chaque** $y \in PairsOccupés$ ***et*** $z \in PairsLibres$ ***et*** $y \in V_z$ **faire**

```
    // Application de la règle de cohésion
```
    **si** $d(y, z) > \lambda$ **alors**
        Retirer $z \in PairsLibres$;

```
// Choix d'un pair où se déplacer
```
Choisir aléatoirement un pair $p \in PairsLibres$;

à une fourmi de communiquer indirectement diverses informations avec d'autres fourmis. Dans notre cas les agents utilisés vont marquer leurs déplacements d'un pair à un autre en mettant des phéromones sur leurs points de départ et d'arrivée.

**Définition 15** *Soit $x$, et $y$ deux pairs tel que $y \in V_x$. On note $\phi_{xy}$ le taux de phéromones entre le pair $x$ et le pair $y$ stocké sur le pair $x$.*

La figure 5.2 montre une vue générale de notre modèle de déplacement. Quand un document est inséré dans le réseau, les fragments sont générés puis disséminés sous forme de nuée. Chaque fragment interagit avec son environnement en déposant des phéromones sur les sites parcourus. Dans la figure 5.2 l'ajout de phéromones est noté $\phi{+}{+}$. Quand, par exemple, un agent se déplace du pair $A$ vers le pair $B$, il dépose des phéromones sur le pair $A$ ($\phi_{AB} + +$), et sur le pair $B$ ($\phi_{BA} + +$).

### 5.2.1 Répartition de charge

La première utilisation du dépôt de phéromones est l'observation de la charge réseau. Très simplement un fort taux sur une relation de voisinage indique un passage important de fragments. Ainsi pour choisir un déplacement, un agent choisira de préférence un chemin vers un pair possédant un taux de phéromones le plus bas possible.

Ici, il faut bien remarquer que le niveau de phéromones localisé sur chaque pair du réseau désigne l'activité sur les liens entre les pairs et non sur les pairs eux-mêmes. Pour obtenir l'activité

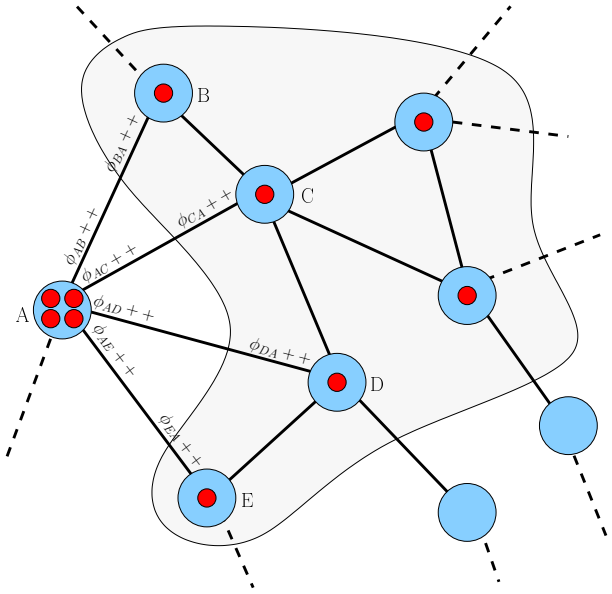



FIGURE 5.2 – Vue d'ensemble de notre modèle

globale d'un pair il est nécessaire de faire la somme des phéromones sur les liaisons incidentes à ce pair puis de propager cette valeur aux pairs présents dans le voisinage. Cette méthode bien que plus représentative de l'activité du réseau, nécessite un envoi de message supplémentaire. Une possibilité pour réduire ce coût est d'utiliser l'envoi des messages de maintien des poids des arcs dans le réseau. Dans notre cas, nos agents basent leurs déplacements sur leurs observations locales, et ne sont donc capables d'observer que les phéromones déposées sur les liaisons entre les pairs.

Pour prendre en compte le taux de phéromones dans le réseau, il est nécessaire d'adapter l'algorithme 2 de déplacement. L'étape de sélection d'un pair respectant les règles 1 et 2 doit se faire en prenant en compte un niveau minimal de phéromones. Pour cela, nous ajoutons la règle suivante :

**Règle 3** *Les agents se déplacent sur les liens vers les pairs ayant eu le moins de passages, en observant les phéromones déposées.*

### 5.2.2 Mesure de confiance

En plus de la répartition de charge, nous pouvons utiliser les phéromones déposées pour déterminer le comportement d'un pair dans le réseau. Pour cela nous observons le déplacement des agents entre les pairs. Si un déplacement de fragment est correct alors le niveau de phéromones entre deux pairs doit être identique. En effet lors du déplacement d'un agent, celui-ci dépose la même quantité de phéromones sur le pair de départ et celui d'arrivée.

**Propriété 1** *Soit $x$ et $y$ deux pairs tels que $y \in V_x$. Le niveau de phéromones stocké sur le pair $x$ pour $y$ est égal au niveau de phéromones stocké sur le pair $y$ pour $x$. Nous avons donc :*

$$\phi_{xy} = \phi_{yx}$$

En évaluant la propriété 1, les pairs d'un réseau sont donc capables d'évaluer le comportement des membres de leur voisinage. A partir de cette évaluation nous pouvons définir une mesure de confiance destinée à l'identification des pairs possédant un comportement inattendu.

**Définition 16** *Soit $x$, et $y$ deux pairs. On note $\tau_{xy}$ la confiance accordée par le pair $x$ au pair $y$ stockée sur le pair $x$.*

Un comportement inattendu peut prendre la forme de pannes machines ou de modifications volontaires du fonctionnement d'un pair dans le but par exemple d'attirer des agents. Dans notre modèle, un fragment se déplace en prenant en compte un faible niveau de phéromones. Un comportement malveillant peut donc se traduire par la modification artificielle des taux de phéromones dans le but d'attirer un document.

Nous avons proposé un algorithme [Pommier et Bourdon, 2008] basé sur le problèmes des généraux byzantins [Wagner, 2003, Zuba, 2004] permettant de mesurer la confiance qu'un pair accorde aux éléments de son voisinage et par conséquent de détecter des comportements malveillants. Cet indice est construit à l'aide des phéromones déposées le long du réseau par les agents en déplacement.

Pour établir la confiance d'un pair $y$, un pair $x$ va comparer le niveau de phéromones $\phi_{xy}$, avec celui stocké sur $y$, $\phi_{yx}$. Si $x$ observe une différence, alors il demande à deux voisins de $y$ présents également dans son voisinage de valider le comportement de $y$. L'avantage d'un tel algorithme est de pouvoir, pour un pair, quantifier la confiance qu'il accorde à ses voisins, simplement en observant l'environnement modifié par le déplacement des fragments. Une fois cet indice de confiance établi, un agent doit le prendre en compte pour effectuer son déplacement. Un agent respecte donc la règle suivante :

**Règle 4** *Les agents se déplacent sur les pairs ayant le plus haut niveau de confiance.*

### 5.2.3 Application au déplacement

Le déplacement d'un agent consiste donc à choisir parmi les pairs voisins d'un site ceux qui ne violent pas les règles de flocking. Une fois obtenue cette liste de pairs, il convient d'en choisir un suivant la confiance observée et les phéromones déposées. Une décision prenant en compte plusieurs critères doit donc être prise. C'est un problème de décision dit multi-critère (MCDM [14]), dans lequel l'opérateur max ne peut pas être appliqué [Galand et Perny, 2006] :

$$\max_{MCDM} \{(\phi_{x1}, \tau_{x1}); (\phi_{x2}, \tau_{x2}); \ldots; (\phi_{xi}, \tau_{xi})\}$$

14. Multi-Criteria Decision Making problem

**Algorithme 3** : Etablissement de la confiance d'un pair $x$ vers un pair $y$

**Entrées** : Deux pairs $x$ et $y$, $V_x$ le voisinage de $x$, $\phi_{xy}$ le niveau de phéromones de $y$ stocké sur $x$

**Sorties** : $\tau_{xy}$ la confiance de $y$ stockée sur $x$

attendre(temps de cycle);

`// Récupération et comparaison des niveaux de phéromones pour` $y$

$x$ demande $\phi_{yx}$ à $y$;

**si** $\phi_{xy} \neq \phi_{yx}$ **alors**

`// Choix de deux voisins de` $y \in V_x$

$p_1 \leftarrow$ pair $\in V_y$ et $\in V_x$;

$p_2 \leftarrow$ pair $\in V_y$ et $\in V_x$ et $\neq pair_1$;

`// Récupération des niveaux de phéromones`

$x$ demande $\phi_{p_1y}$ et $\phi_{p_2y}$ à $p_1$ et à $p_2$;

$x$ demande $\phi_{yp_1}$ et $\phi_{yp_2}$ à $y$;

`// Comparaison des niveaux de phéromones pour` $y$

`// Mise à jour de la confiance pour` $y$

**si** $\phi_{yp_1} \neq \phi_{p_1y}$ *ou* $\phi_{yp_2} \neq \phi_{p_2y}$ **alors**

$x$ diminue $\tau_{xy}$;

**sinon**

$x$ augmente $\tau_{xy}$;

**sinon**

$x$ augmente $\tau_{xy}$;

où $x$ est un pair, $\phi_{xi}$ est le niveau de phéromones pour un pair $i$, et $\tau_{xi}$ la confiance associée. Considérons le problème à deux critères suivant :

**Exemple 2** *Soit un agent souhaitant se déplacer vers un pair distant. Comment choisir une destination ne dégradant aucun des deux critères parmi les pairs $p_1$, $p_2$, et $p_3$ suivants : $p_1 = (0, 50)$, $p_2 = (20, 51)$ et $p_3 = (35, 35)$ ? Appliquer l'opérateur* max *sur le second critère de confiance reviendrait à choisir le pair $p_2$ dégradant le premier critère.*

Dans notre modèle nous avons donc normalisé le niveau de phéromones et le niveau de confiance. Une somme pondérée a pu alors être utilisée :

$$\max_{MCDM} = \max_{i}(\alpha_\phi \phi_{xi} + \alpha_\tau \tau_{xi})$$

où $\alpha_\phi, \alpha_\tau$ sont les poids de chaque critère. Dans le cas de l'exemple 2, en fixant le poids $\alpha_\phi$ à $-1$ et le poids $\alpha_\tau$ à 1, le pair $p_1$ aurait été choisi. Nous pouvons donc proposer l'algorithme 4, adaptation de l'algorithme 2 pour le déplacement d'un agent.

**Algorithme 4** : Déplacement d'un fragment d'un pair $x$ vers un pair $p$

**Entrées** : Un pair $x$, $V_x$ le voisinage de $x$, *doc* un document, $f_{doc}$ un fragment $\in$ *doc*
**Sorties** : Un pair $p$ où stocker $f_{doc}$

```
// Récupération des pairs issus des règles de flocking
```
$PairsLibres \leftarrow$ Algorithme 2;

```
// Normalisation des critères et calcul du score
```
**pour chaque** $y \in PairsLibres$ **faire**
  Normaliser $\phi_{xy}$;
  Normaliser $\tau_{xy}$;
  $Score \leftarrow \alpha_\phi \phi_{xy} + \alpha_\tau \tau_{xy}$;

```
// Choix d'un pair où se déplacer
```
Choisir $p \in PairsFocking$ avec max $Score$;

Ici nous avons repris l'algorithme 2 et nous avons supprimé l'étape de sélection d'un pair $p$ pour la remplacer par une sélection reposant sur une décision multi-critère. Ainsi parmi les nœuds respectant les règles de flocking, nous normalisons le niveau de phéromones et la confiance puis nous calculons une somme pondérée représentant le *score* d'un pair. Le choix de $p$ consiste alors à choisir un pair possédant un *score* maximal parmi ceux présents dans $PairsLibres$.

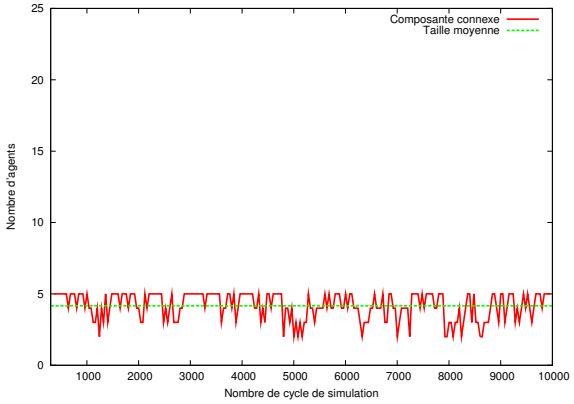
## 5.3 Expérimentations

Les expérimentations suivantes, menées sous oRis [Harrouet, 2000], nous ont permis d'évaluer le comportement de nos agents responsables du placement de l'information, ainsi que de valider les différents algorithmes mis en jeu. Pour cela nous avons mesuré :

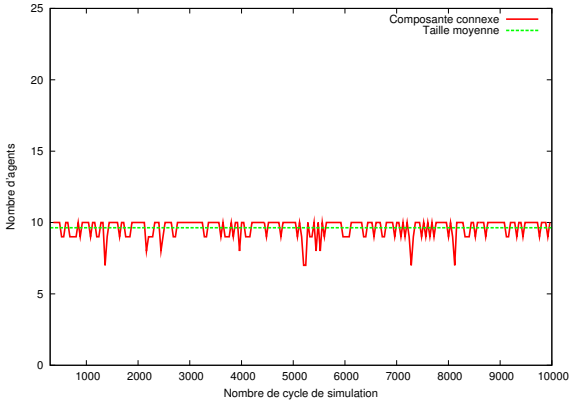
– La connexité d'une nuée en déplacement pour valider le fait que nos agents se déplacent en groupe dans le temps.
– La répartition des phéromones dans le temps et la couverture du réseau en terme de pairs parcourus pour observer la répartition de charge.
– Le niveau de confiance dans le réseau et les niveaux de phéromones associés aux pairs dont le comportement est suspicieux pour évaluer l'algorithme de confiance et l'implication sur une nuée.

### 5.3.1 Évaluation des règles de flocking

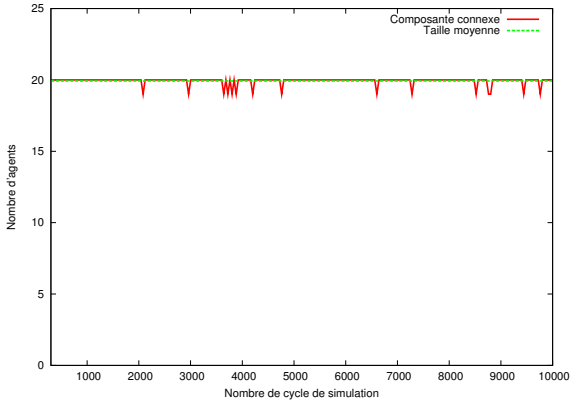
Pour valider le fait que nos agents se déplacent sous la forme d'une nuée nous en avons mesuré la connexité. La première série d'expériences consiste à simuler le comportement d'une nuée composée de 5, 10, et 20 fragments dans un réseau constitué de 100 pairs. La seconde série consiste à évaluer le comportement des ces mêmes nuées dans un réseau de 400 pairs. Les figures 5.3 et 5.4 montrent les résultats pour des réseaux composés respectivement de 100 pairs et de 400 pairs.




(a) Nuée de 5 fragments




(b) Nuée de 10 fragments



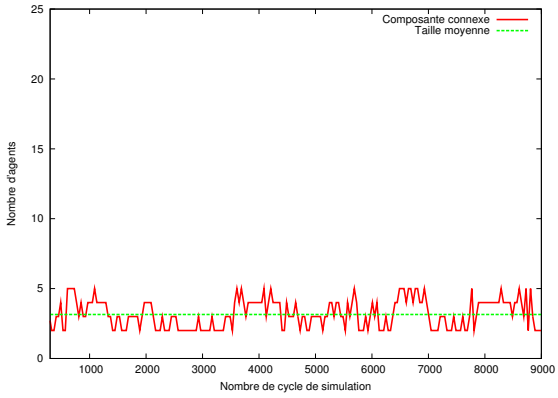

(c) Nuée de 20 fragments

FIGURE 5.3 – Connexité d'une nuée dans un réseau de 100 pairs

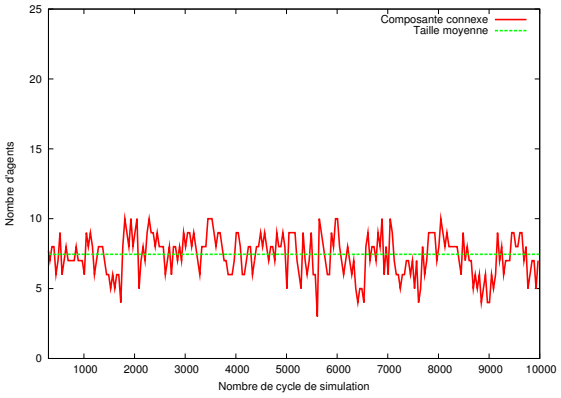
Dans le cas où une nuée est constituée de 5 éléments, nous pouvons voir sur la figure 5.3(a) que la plus grande composante connexe est en moyenne de taille 4. Nous pouvons également remarquer que la plus grande composante connexe peut être de taille 2 puis, les cycles suivants, être de taille maximale. Les règles jouent ici pleinement leur rôle à savoir qu'un fragment n'est jamais véritablement décroché de la nuée.

Dans le cas où une nuée est composée de 10 éléments, les résultats exposés dans la figure 5.3(b) montrent un meilleur maintien de la taille de la nuée. Ici la moyenne se rapproche de la taille maximale et se situe à 9,6. Nous pouvons remarquer que dans de rares cas plus de 4 agents sont isolés de la nuée principale.

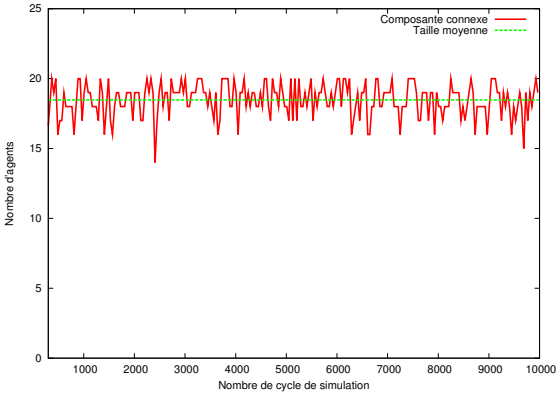
La figure 5.3(c) montre la situation pour une nuée de 20 fragments. Ici nous pouvons observer une très faible perte de fragments et la moyenne se confond avec la taille maximale. Ce phénomène d'augmentation du degré de cohésion avec la taille de la nuée provient de la grandeur du réseau utilisée et par conséquent du nombre de voisins pour chaque pair. Étant donnée la taille de la nuée, le nombre de voisins permettant à un agent de se déplacer est restreint. De fait il devient difficile pour un agent de violer les règles de flocking. Nous allons maintenant étudier ces trois nuées de 5, 10, 20 éléments dans un réseau de 400 machines.




(a) Nuée de 5 fragments




(b) Nuée de 10 fragments




(c) Nuée de 20 fragments

FIGURE 5.4 – Connexité d'une nuée dans un réseau de 400 pairs

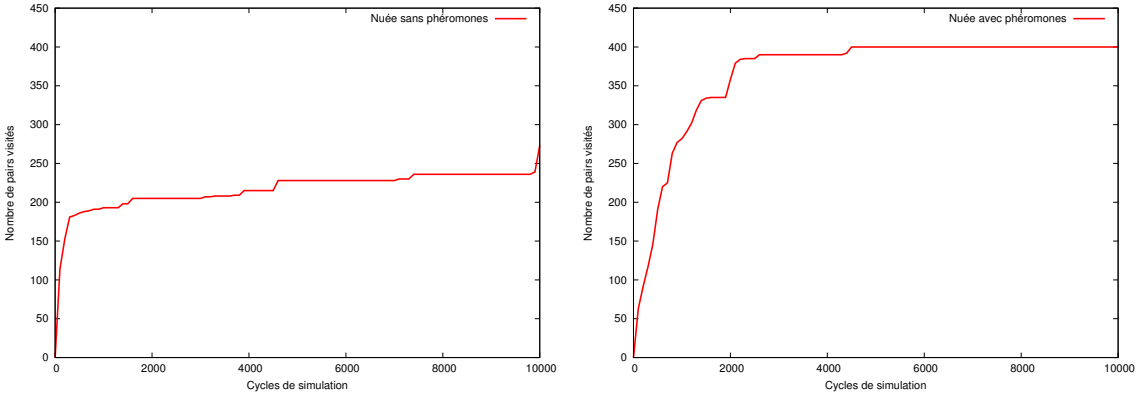
Les résultats présentés dans la figure 5.4(a) montrent une nuée composée en moyenne de 3 éléments. Ici dans très peu de cas, la nuée possède une composante connexe de taille maximale. Bien que la nuée puisse se reformer, ces résultats indiquent une séparation en plusieurs sous-groupes. Nous sommes dans le cas opposé au cas précédent. Ici l'augmentation de la taille du voisinage d'un pair laisse trop de possibilités de déplacement pour un fragment. Le même constat s'établi pour une nuée de 10 éléments. Sur la figure 5.4(b) la moyenne est supérieure à 7 et dans très peu de cas la nuée se déplace sous une forme de taille maximale.

Enfin, nous pouvons observer dans la figure 5.4(c) le comportement pour une nuée de 20 éléments. Ici les résultats sont meilleurs que dans les cas précédents. Le nombre moyen d'éléments de la nuée se situe autour de 18, et la majeure partie du temps la nuée se déplace sous la forme d'un nuage de 20 fragments.

Ces expérimentations permettent de valider le comportement de groupe de nos agents. Dans tous les cas qui ont été observés, les règles de flocking permettent à une nuée ayant une composante connexe inférieure à la taille maximale de se recomposer et ainsi retrouver sa taille d'origine. Toutefois il convient d'en relativiser l'efficacité. En effet les résultats précédents montrent une dépendance entre la connexité d'une nuée et la taille du réseau. Cette dépendance est héritée du réseau sous-jacent qui a été construit et dont le degré de chaque pair dépend du nombre de pairs dans le réseau.

### 5.3.2 Répartition de charge

Pour évaluer la répartition de charge dans le réseau, nous avons mesuré le nombre de cycles de simulation nécessaires à une nuée pour couvrir la totalité du réseau. Nous avons effectué les mesures pour une nuée de 20 éléments déposant des phéromones, et pour une nuée de taille équivalente n'en utilisant pas. Le réseau est constitué ici de 400 pairs.




(a) Sans phéromones

(b) Avec phéromones

FIGURE 5.5 – Couverture du réseau pour une nuée de 20 fragments

La première courbe (figure 5.5(a)) montre une couverture pour une nuée ne déposant pas de phéromones. Le second graphique dans la figure 5.5(b) montre la couverture pour une nuée

déposant des phéromones. Nous pouvons voir clairement l'apport des phéromones. Quand celles-ci sont prises en compte dans le déplacement des agents, la totalité du réseau est parcourue par la nuée en environ 2000 cycles de simulation. Dans le cas où il n'y pas d'interactions avec l'environnement les agents ne cherchent pas à emprunter les liens de voisinage possédant peu de phéromones.

Pour évaluer plus précisément la répartition de charge, nous avons mesuré le nombre de fragments qui ont été stockés par les pairs du réseau dans le cas où une nuée dépose des phéromones. La figure 5.6 montre une telle répartition dans le réseau.

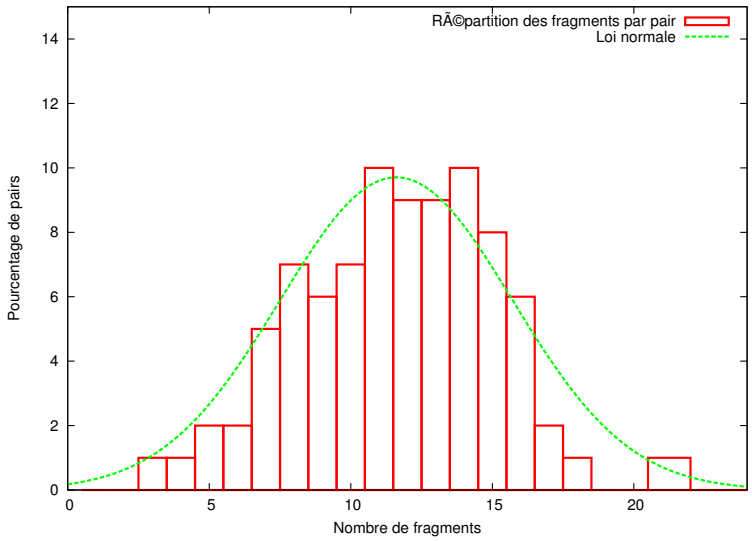



FIGURE 5.6 – Nombre de fragments stockés par pair dans un réseau de 400 nœuds

Nous avons représenté ici le pourcentage de pairs du réseau en fonction du nombre de fragments hébergés. Nous pouvons observer que la moyenne se situe à environ 12 fragments hébergés pour un écart type proche de 4. Cette courbe met en évidence le fait que la répartition de charge dans le réseau est homogène. Nous pouvons observer ici qu'environ 75% des pairs du réseau ont hébergé entre 8 et 16 fragments.

Les cas où peu de pairs ont hébergé peu de fragments peuvent s'expliquer par le fait que l'expérimentation s'arrête une fois que l'ensemble des pairs ont été parcourus. Dès lors le dernier pair parcouru n'aura vu par exemple qu'un seul fragment, et ses voisins auront également vu un nombre moindre d'agents. Nous avons un cas similaire avec les pairs ayant possédé beaucoup de fragments. Ici cela est du à la dissémination initiale d'une nuée. Un seul pair génère l'ensemble des fragments, il est donc nécessaire que peu de pairs situés dans le voisinage du nœud générateur stockent à intervalle de temps réduit plusieurs fragments pour que les règles de flocking se mettent en place.

L'histogramme obtenu semble respecter une distribution suivant une loi normale asymétrique, ce qui semble se confirmer à la vue de la courbe gaussienne associée. Nous avons mesuré les coefficients d'asymétrie et d'aplatissement. Le coefficient d'asymétrie est obtenu par le calcul du

"skewness" et le coefficient d'aplatissement par le calcul du "kurtosis". Nous pouvons étudier la normalité de la distribution en évaluant le rapport de ces mesures sur l'erreur type. Le tableau 5.1 contient les résultats de nos mesures effectuée avec le logiciel de statistique R.

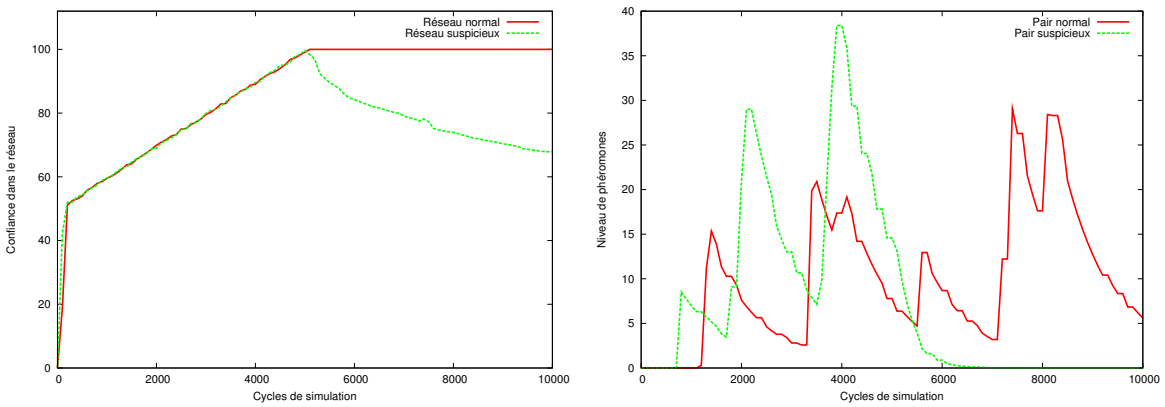
| Asymétrie | Aplatissement | Erreur type | $\frac{\text{Asymétrie}}{\text{Erreur type}}$ | $\frac{\text{Aplatissement}}{\text{Erreur type}}$ |
|---|---|---|---|---|
| 0.6476469 | 1.873071 | 0.7101878 | 0.9119375 | 2.6374305 |

TABLE 5.1 – Statistiques descriptives de la distribution

A la lecture de ces résultats il est difficile de conclure. Le rapport du coefficient d'asymétrie sur l'erreur type est compris entre -2 et 2. Nous ne pouvons donc pas rejeter la normalité. Cependant, la valeur du coefficient d'aplatissement sur l'erreur type, supérieure à 2, indique que la normalité peut être rejetée.

### 5.3.3 Mesure de confiance

Les expériences suivantes vont nous permettre d'évaluer et de mettre en évidence les pairs possédant un comportement inattendu. Dans ces simulations nous avons traduit ce comportement par l'action de modifier artificiellement ses phéromones. Cela se traduit par une évaporation artificiellement accélérée des phéromones. Pour mesurer notre algorithme de confiance nous introduisons 10% de pairs possédant un comportement anormal. Nous déclenchons la malveillance des pairs après le 5000$^{e}$ pas de simulation.




(a) Mesure de la confiance

(b) Taux de phéromones

FIGURE 5.7 – Évaluation de la confiance dans un réseau de 400 pairs

La confiance dans le réseau est visible sur la figure 5.7(a). La courbe rouge pleine montre la confiance dans un réseau composé de pairs ayant un comportement normal. Ici chaque pair possède un niveau de confiance initialisé à une valeur de 50. Dans le cas où le comportement malveillant des pairs est déclenché, la confiance générale du réseau diminue. Une fois ce comportement détecté par l'algorithme de confiance, la nuée ne passe plus par les pairs incriminés. Le

niveau de phéromones sur ces pairs, visible sur la figure 5.7(b) diminue alors jusqu'à atteindre le seuil minimal.

Ces résultats sont confirmés également sur la couverture du réseau. Sur la figure 5.3.3 nous pouvons observer que seulement 85% du réseau ont été parcourus par une nuée. Ici nous remarquons une différence de 5% entre le nombre de machines parcourues et le nombre de machines malveillantes. Cette différence s'explique par le fait que les pairs malveillants sont placés aléatoirement dans le réseau ce qui peut conduire à l'apparition de zones inaccessibles.

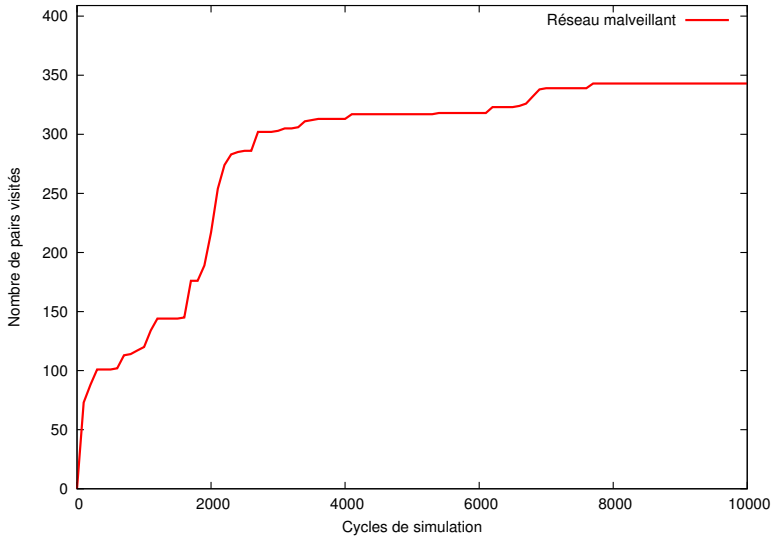



FIGURE 5.8 – Couverture d'un réseau de 400 pairs possédant 10% de pairs malveillants

La conséquence de cette mesure de confiance, est l'émergence de l'évitement d'obstacle. Avec une telle mesure, une nuée a la capacité d'éviter un ou des pairs possédant un comportement suspicieux. Un tel schéma peut se trouver dans le cas d'attaques coordonnées par exemple. Plusieurs pairs peuvent s'entendre et ainsi tenter une attaque en coalition.

## Conclusion du chapitre

Nous avons présenté au cours de ce chapitre un placement de l'information reposant sur des agents mobiles capables de prendre leurs propres décisions. Le comportement de ces agents est emprunté au monde animal et plus particulièrement aux oiseaux et aux fourmis. Les règles de flocking mises en place permettent à un groupe d'agents de se déplacer sous la forme d'une nuée dont les membres hébergent un fragment d'information issu d'une étape de fragmentation préalable. Le second aspect biologique est le dépôt de phéromones. A partir de cette trace laissée dans l'environnement, nous avons établi une règle de répartition de charge complémentaire au déplacement, et nous avons construit un algorithme permettant d'établir la confiance de chaque pair du réseau.

L'ensemble de ces mécanismes, a été implémenté puis testé au travers du simulateur oRis. Les résultats ont permis de valider notre méthode de placement de l'information reposant sur le déplacement en groupe des agents. Ce placement dynamique et décentralisé, est capable de s'adapter aux conditions changeantes de l'environnement grâce à l'algorithme de confiance. De plus ce placement permet de respecter une répartition de charge homogène sur l'ensemble du réseau.

Une fois cette gestion de l'information établie, il est nécessaire de proposer un mécanisme de recherche efficace capable de contacter un ou plusieurs membre d'une nuée, prenant en compte la mobilité de la nuée. C'est cette problématique que nous allons étudier dans le chapitre suivant.

# Chapitre 6

# Recherche de l'information mobile

**Sommaire**



Notre modèle de placement de l'information permet à un groupe d'agents de se déplacer sous la forme d'une nuée. Pour retrouver un document stocké sous la forme d'une nuée en mouvement, il est nécessaire de proposer une méthode de recherche qui soit efficace même en cas de forte mobilité. Nous avons vu dans le chapitre 2.3.2 que les méthodes à base d'inondation, bien qu'efficaces, sont beaucoup trop coûteuses.

Pour retrouver un groupe d'agents nous nous appuyons sur deux algorithmes. Le premier repose sur une marche aléatoire ou chaîne de Markov, et le second utilise les capacités d'adaptation à l'environnement et de stigmergie des agents. Cet algorithme permet aux agents responsable de la recherche de documents de déposer des phéromones pour marquer leurs déplacements. L'objectif de ces deux algorithmes est de proposer un mécanisme de recherche totalement décentralisé, permettant de trouver un ou plusieurs éléments d'une nuée donnée. Les règles de flocking précédemment mises en place, assurant une connexité entre les agents stockant des morceaux d'information, doivent ainsi permettre de récupérer à moindre coût l'ensemble ou une partie des fragments d'un document puis de le reconstruire.

## 6.1 Marcheur aléatoire

Nous avons vu dans le chapitre 2.3 que les méthodes de recherche reposant sur des marcheurs aléatoires obtenaient de bons résultats en terme de nombre de messages et de nombre de sauts pour localiser une ressource. Nous avons voulu soumettre cette approche dans un cadre de forte mobilité des données. Nous proposons dans cette partie un algorithme de marche aléatoire pour rechercher une nuée de fragments. Le principe de l'algorithme consiste pour un agent de recherche à choisir aléatoirement un voisin sur qui se déplacer. Un marcheur aléatoire peut être vu comme une chaîne de Markov, processus stochastique dont l'état futur ne dépend que du présent et pas des choix passés.

### 6.1.1 Connexité du réseau

La première condition pour construire un marcheur aléatoire est de respecter la condition de connexité du réseau. La construction du réseau sous-jacent (chapitre 4) permet de voir ce dernier comme un graphe connexe où chaque pair possède un degré de $(c+1)\log(n)$. Cette propriété de connexité du réseau est essentielle. Un agent de recherche a ainsi accès à l'ensemble des pairs du réseau à partir de son état courant. Une fois cette condition établie, nous pouvons construire l'ensemble des déplacements possibles pour un agent de recherche de la manière suivante :

**Définition 17** *Soit $dep_x$ l'ensemble des déplacements d'un agent de recherche à partir d'un pair $x$. Nous avons :*

$$dep_x = V_x \cup x$$

Ici pour un pair $x$ donné nous avons pris comme possibilité de déplacement l'ensemble du voisinage de $x$ noté $V_x$. Pour rappel $V_x$ désigne le voisinage d'un pair $x$ comme étant la réunion des vues $IV$ et $PV$ (cf. définition 12). Nous avons également ajouté $x$ à la liste des déplacements. Cet ajout permet de prendre en compte le cas où l'agent peut décider de rester sur place.

### 6.1.2 Probabilité de transition

La seconde étape pour la construction d'une chaîne de Markov concerne l'établissement des probabilités de transition entre les différents sommets du réseau. Pour parcourir le réseau un agent de recherche va passer d'un pair $x$ à un pair $y$ appartenant à la liste des déplacements possibles $dep_x$ (cf. definition 17) avec une probabilité $P_{xy}$. Nous avons donc :

$$\sum_{y \in dep_x} P_{xy} = 1 \tag{6.1}$$

Le réseau SCAMP possédant des propriétés similaires à celles présentes dans les graphes aléatoires [Erdös et Rényi, 1960] nous avons une convergence du degré de chaque pair vers $(c+1)\log(n)$. La probabilité $P_{xy}$ pour qu'un marcheur passe d'un pair $x$ à un pair $y \in dep_x$ est :

$$P_{xy} = \frac{1}{Card(dep_x)} = \frac{1}{(c+1)\log(n)+1} \tag{6.2}$$

Ici nous avons une distribution uniforme sur l'ensemble des déplacements, chaque état pouvant être atteint avec la même probabilité. Pour compléter la construction de la marche aléatoire, nous faisons l'hypothèse que le degré des nœuds dans le réseau est constant en $(c+1)\log(n)$ dans le temps. Cette condition est nécessaire pour établir la symétrie entre les probabilités de transition :

$$P_{xy} = P_{yx} \tag{6.3}$$

L'équation 6.3 permet de s'assurer que la matrice des probabilités de transition est symétrique et par conséquent que la distribution uniforme est stationnaire.

### 6.1.3 Algorithme de déplacement

Une fois établi l'ensemble des propriétés nécessaires à la création d'une chaîne de Markov, nous pouvons proposer un algorithme permettant à un agent de parcourir les pairs du réseau suivant une marche aléatoire.

**Algorithme 5** : Agent de recherche suivant une marche aléatoire situé sur un pair $x$

**Entrées** : *doc* l'identifiant d'un document recherché, *emetteur* le pair émetteur de l'agent, $V_x$ le voisinage d'un pair $x$

**Sorties** : Un pair $y$ où se déplacer

```
// Vérification de la présence du fichier doc recherché
```
**si** $verif(doc) == true$ **alors**
  Donner *emetteur* à $f_{doc}$;
**sinon**
  `// Construction de` $dep_x$
  $dep_x \leftarrow V_x$;
  $dep_x \leftarrow x$;
  `// Assignation des probabilités`
  **pour chaque** $y \in dep_x$ **faire**
    $P_{xy} = \frac{1}{Card(dep_x)+1}$;
  Choisir $y \in dep_x$ avec la probabilité $P_{xy}$;
  Se déplacer sur le pair $y$;

La première étape de l'algorithme consiste à vérifier si un des membres de la nuée recherchée est présent sur le pair parcouru. La méthode $verif(doc)$ vérifie la présence de fragments issus d'un document d'identifiant *doc*. Si la présence d'un fragment $f_{doc}$ est avérée alors l'agent de recherche fournit l'adresse du pair émetteur de la recherche à l'agent possédant $f_{doc}$. Dans le cas où le nœud parcouru n'héberge pas de ressource appartenant à *doc*, nous construisons l'ensemble des déplacements possibles $dep_x$ contenant l'ensemble des voisins de $x$, ainsi que $x$ lui-même. Une fois cette opération effectuée il est nécessaire d'assigner à chaque pair $y$ contenu dans $dep_x$ une

probabilité de transition $P_{xy}$ comme définie précédemment. Enfin l'agent de recherche exécutant cet algorithme va choisir un pair $y$ avec une probabilité $P_{xy}$ pour se déplacer dessus.

L'algorithme 5 permet à un agent de se déplacer dans l'ensemble du réseau et d'en atteindre n'importe quel nœud. Ici un agent choisit aléatoirement un membre du voisinage d'un pair et vérifie si la ressource recherchée est présente dessus. Etant donnée qu'une marche aléatoire ne prend pas en compte les choix passés d'un agent il est possible qu'un agent parcourt plusieurs fois les mêmes pairs ralentissant de fait sa couverture du réseau.

Nous proposons un second algorithme utilisant les capacités d'interaction avec l'environnement des agents. Cet algorithme autorise un agent à déposer des phéromones indiquant son passage. Nous souhaitons conserver dans cet algorithme l'aspect aléatoire de la recherche tout en marquant les zones déjà parcourues.

## 6.2 Parcours de réseau utilisant des phéromones

Le but de l'algorithme de recherche que nous présentons ici est de proposer une couverture du réseau plus rapide que dans le cas d'une marche aléatoire. Cet algorithme repose sur l'utilisation de phéromones. Un agent de recherche va parcourir aléatoirement le réseau en marquant les pairs parcourus par l'intermédiaire d'un dépôt de phéromones.

### 6.2.1 Dépôt et évaporation de phéromones

Pour construire un tel algorithme nous utilisons les capacités de stigmergie des agents. Le but est de faire déposer par chaque agent une trace de phéromones sur les pairs parcourus. Le comportement d'un agent consiste alors à choisir un pair possédant le moins de phéromones pour effectuer son prochain déplacement. Nous pouvons donc proposer la définition suivante :

**Définition 18** *Soit $x$ et $y$ deux pairs. Un agent de recherche marque son déplacement d'un pair $x$ vers un pair $y$ en déposant des phéromones sur le pair $y$. On note $\rho_y$ le niveau de phéromones de recherche stocké sur le pair $y$.*

Cette définition des phéromones est différente de celle utilisée dans la section précédente (cf définition 15). Ici les agents de recherche marquent les sommets visités à la différence des agents hébergeant des fragments qui déposaient des phéromones sur les liens. Cette méthode de dépôt doit permettre à un agent de couvrir plus rapidement le réseau comparativement à une nuée. La couverture accélérée du réseau s'explique par le fait que les agents déposant des phéromones sur les liens peuvent revenir sur des pairs déjà parcouru. Le marquage se fait sur le moyen d'accéder à un pair, et non sur le pair. Les agents de recherche que nous proposons effectuent ce marquage sur les pairs.

Du fait de la mobilité de la nuée, un agent de recherche peut parcourir l'ensemble du réseau sans pouvoir trouver une nuée recherchée. Il est donc nécessaire de permettre à un agent de parcourir à nouveau les pairs déjà visités. Le marquage de sommets des agents doit donc être

temporaire et fournir une indication sur la date de passage d'un agent. Pour cela nous fixons le taux d'évaporation des phéromones en fonction de la taille du réseau. Ainsi quand un agent dépose une unité de phéromones sur un pair, il ne doit rester aucune trace de phéromones sur ce pair au bout d'un nombre de cycles équivalent à la taille du réseau. Nous pouvons donc définir :

**Définition 19** *Soient $x$ un pair et $n$ le nombre de pairs du réseau. Nous avons le taux d'évaporation des phéromones :*

$$evapo = \frac{\rho_x}{n} \tag{6.4}$$

Ainsi à chaque temps dans le réseau les niveaux de "phéromones de recherche" sur chaque pair sont décrémentés de *evapo*. De cette manière il ne reste aucune trace du passage d'un agent de recherche au bout d'un temps équivalent à $n$ sauts dans le réseau.

### 6.2.2 Choix des déplacements

Pour prendre en compte le niveau de phéromones de recherche dans le déplacement d'un agent, nous sélectionnons les pairs possédant le moins de phéromones. Cette sélection s'effectue dans la liste des déplacements possibles à partir d'un pair $x$. Le but est d'extraire de $dep_x$, construit suivant la definition 17 l'ensemble des pairs possédant un niveau de phéromones minimum.

**Définition 20** *Soit $dep_x$ l'ensemble des possibilités de déplacement d'un agent de recherche. Nous pouvons construire l'ensemble $depPhero_x$ tel que : $depPhero_x = y \in dep_x$ avec $\rho_y$ minimum.*

Le fait de choisir les pairs possédant un niveau de phéromones minimum a deux implications. Tout d'abord cela permet pour un agent d'éviter de parcourir les pairs plusieurs fois et donc d'améliorer en terme de sauts la couverture du réseau. Ensuite cela permet de prendre en compte le cas où plusieurs agents de recherche pour un même fichier sont générés. Ainsi un agent est capable de détecter les pairs ayant vu le passage d'un agent du même type que lui récemment. Une fois l'ensemble *depPheo* construit nous pouvons donc y choisir un pair aléatoirement pour le déplacement d'un agent de recherche.

### 6.2.3 Algorithme de déplacement

Nous proposons l'algorithme suivant pour prendre en compte le dépôt de phéromones dans le déplacement d'un agent de recherche.

La première étape de l'algorithme 6 consiste pour un agent de recherche a déposer des phéromones sur le site sur lequel il se situe. Dans un second temps l'agent va vérifier la présence du fichier recherché d'identifiant *doc* par l'intermédiaire de la méthode $verif(doc)$. Si le résultat retourné est vrai alors l'agent de recherche fournit l'adresse du pair émetteur de la recherche à l'agent hébergeant un fragment issu de *doc* noté $f_{doc}$. Dans le cas où le nœud parcouru n'héberge

**Algorithme 6** : Agent de recherche déposant des phéromones situé sur un pair $x$

**Entrées** : *doc* l'identifiant d'un document recherché, *emetteur* le pair émetteur de l'agent, $V_x$ le voisinage d'un pair $x$

**Sorties** : Un pair $y$ où se déplacer

`// Dépôt de phéromones`

Incrémenter $\rho_x$;

`// Vérification de la présence du fichier` *doc* `recherché`

**si** $verif(doc) == true$ **alors**

  Donner *emetteur* à $f_{doc}$;

**sinon**

  `// Construction de` $dep_x$

  $dep_x \leftarrow V_x$;

  $dep_x \leftarrow x$;

  `// Demande des phéromones de recherche` $\rho$

  **pour chaque** $p \in dep_x$ **faire**

    Demander et stocker $\rho_p$;

  `// Construction de` $depPhero_x$

  **pour chaque** $p \in dep_x$ **faire**

    $depPhero_x \leftarrow p$ avec $\rho_p$ minimum;

  Choisir aléatoirement $y \in depPhero_x$;

  Se déplacer sur le pair $y$;

pas de ressources appartenant à *doc*, nous construisons l'ensemble des déplacements possibles, $dep_x$ à partir du pair $x$. Une fois $dep_x$ construit, un agent doit récupérer le niveau de phéromones de recherche $\rho$ de chaque pair contenu dans $dep_x$. Nous pouvons donc dès lors extraire les pairs possédant un niveau de phéromones minimum dans $depPhero_x$. C'est à partir de l'ensemble de pairs $depPhero_x$ que l'agent de recherche va établir son prochain déplacement. La dernière étape consiste alors à choisir aléatoirement un pair $y$ appartenant à $depPhero_x$.

Le but de l'algorithme présenté ici est de proposer une méthode proche d'une marche aléatoire utilisant les capacités de stigmergie des agents dans le but d'accélérer la recherche de nuée. Cet algorithme n'est pas et ne peut pas être considéré comme une marche aléatoire. Bien que le choix d'un pair pour se déplacer soit aléatoire, les probabilités de transition entre les pairs ne sont pas symétriques.

## 6.3 Expérimentations

Les expériences suivantes permettent d'évaluer les algorithmes de recherche précédents. Nous avons implémenté sur le simulateur oRis les deux comportements de recherche et nous avons mesuré en fonction de la taille d'une nuée et de sa vitesse de déplacement :

- La couverture du réseau, permettant de voir la différence de comportement entre les deux algorithmes.
- Le nombre de sauts effectués par un agent pour retrouver un document avec et sans phéromones de recherche.
- L'efficacité des deux algorithmes de recherches. Nous avons borné le nombre de sauts obtenus dans chaque cas, et nous avons mesuré le pourcentage de réussite de la recherche.

Pour chaque paramètre d'expérimentation nous avons généré 100 recherches successives à intervalle de temps régulier dans un réseau composé de 400 pairs.

### 6.3.1 Couverture du réseau

La figure 6.1 montre la couverture du réseau pour les deux algorithmes de recherche. La couverture est exprimée en nombre de pairs différents parcourus en fonction du nombre de cycles de simulation. La courbe pleine rouge décrit les résultats pour un marcheur aléatoire, et la courbe pointillée verte décrit les résultats pour un agent déposant des phéromones.

Les résultats confirment le fait que l'algorithme utilisant des phéromones permet une couverture du réseau plus rapide en comparaison avec un algorithme reposant sur une marche aléatoire. Une marche aléatoire nécessite plus de 2000 sauts pour parcourir l'ensemble du réseau. Un agent suivant l'algorithme à base de phéromones aura vu la totalité des pairs en moins de 700 cycles de simulation. Une fois ce constat établi nous pouvons nous intéresser aux résultats comparatifs entre les deux algorithmes concernant la recherche de nuées.

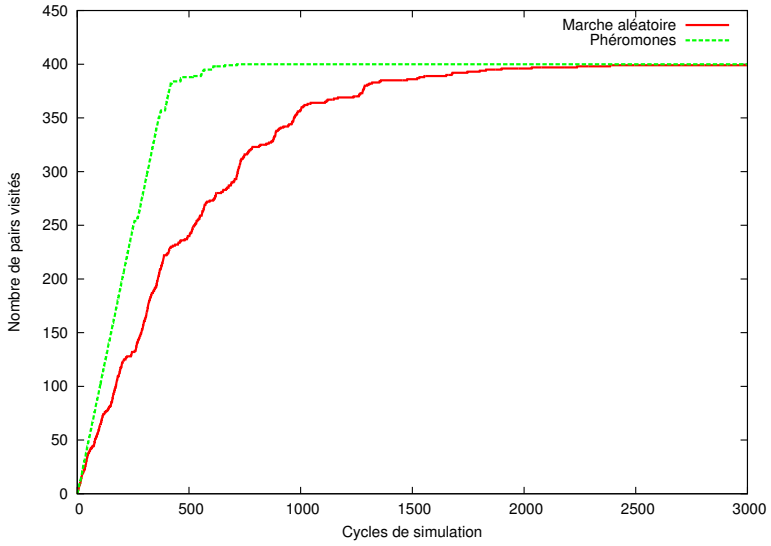


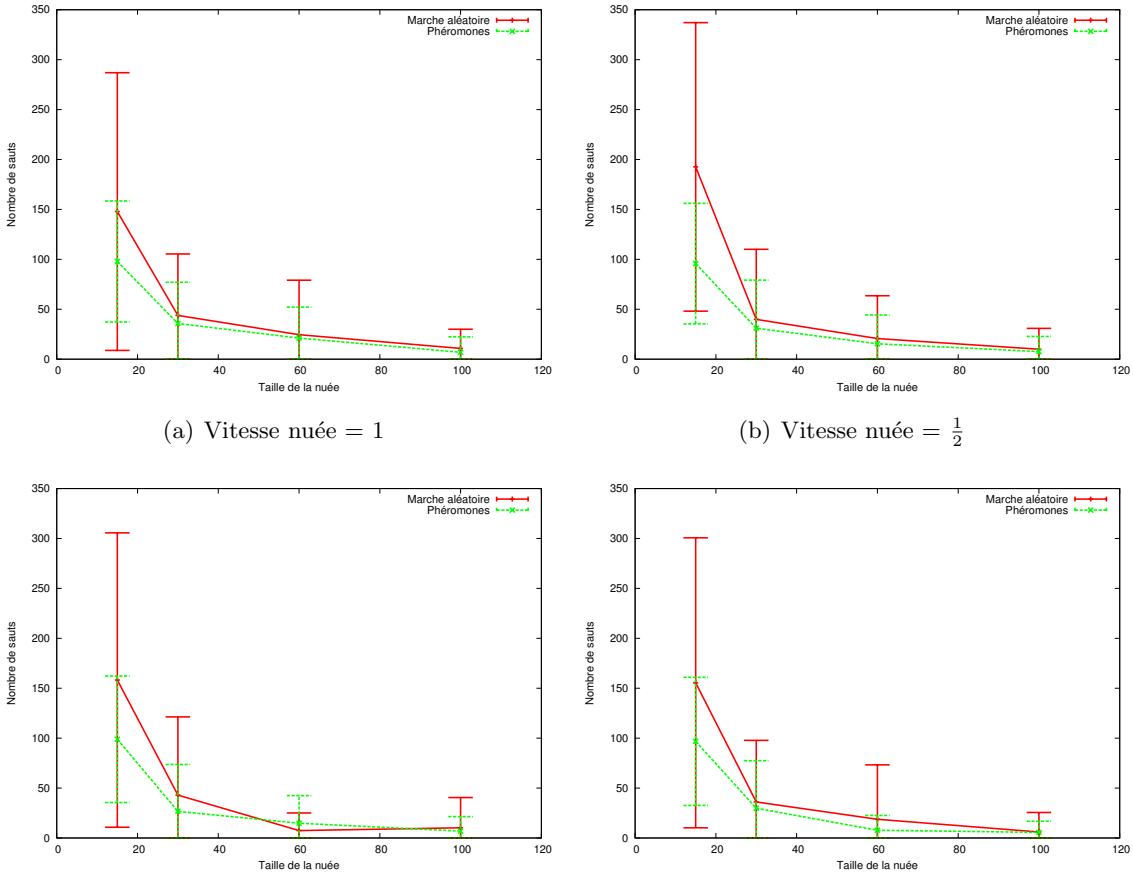

FIGURE 6.1 – Couverture du réseau des agents de recherche

### 6.3.2 Nombre de sauts

La figure 6.2 permet d'observer le nombre de sauts nécessaires pour retrouver une nuée en fonction de sa vitesse de déplacement. Chaque courbe décrit le nombre moyen de sauts effectués ainsi que l'écart type pour chaque comportement d'agent suivant la taille d'une nuée. Nous avons effectué des simulations pour des nuées de taille 15, 30, 60, 100 se déplaçant successivement à des vitesses 1, $\frac{1}{2}$, $\frac{1}{5}$, et $\frac{1}{10}$. Une vitesse $\frac{1}{2}$ signifie qu'un agent de la nuée se déplace un cycle de simulation sur 2. Les agents de recherche se déplacent à chaque cycle de simulation. La courbe pleine rouge montre les résultats pour une méthode à base de marche aléatoire, et la courbe pointillée verte décrit le comportement de l'agent utilisant des phéromones.

La courbe 6.2(a) montre les résultats lorsque les agents d'une nuée se déplacent à chaque cycle de simulation. Pour une nuée de taille 15, il faut en moyenne 97 sauts dans le réseau pour retrouver une nuée dans le cas où un agent utilise des phéromones et 148 dans le cas où l'agent suit une marche aléatoire. Le gain ici est proche de 35%. L'écart type est également nettement plus faible lorsqu'un agent utilise des phéromones. Dans le cas où la taille de la nuée passe à 30 éléments l'écart entre les méthodes de recherche diminue et le gain à utiliser une méthode à base de phéromone chute à 19%. Cette chute s'amplifie pour une nuée de taille 60, où le gain est de l'ordre de 13%. Enfin dans le cas où la nuée est de taille 100 les deux méthodes fournissent des résultats similaires aux environ de 7 sauts en moyenne.

Ce comportement se retrouve sur les figures 6.2(b), 6.2(c), et 6.2(d). Nous avons un écart substantiel pour des nuées de taille 15, et une diminution de cette différence corrélée avec une augmentation de la taille d'une nuée. Le fait de retrouver le même comportement sur les quatre variations de vitesse semble indiquer que le paramètre de vitesse ne joue aucun rôle sur la recherche de fichier quelque soit la méthode utilisée. Ces propos sont confirmés par les courbes 6.3(a), 6.3(b), 6.3(c), 6.3(d) suivantes.



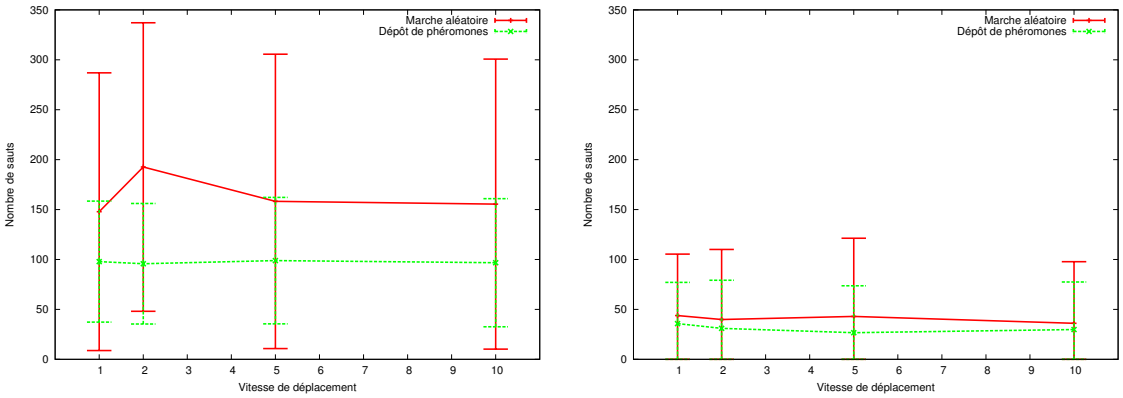

(a) Vitesse nuée = 1

(b) Vitesse nuée = $\frac{1}{2}$

(c) Vitesse nuée = $\frac{1}{5}$

(d) Vitesse nuée = $\frac{1}{10}$

FIGURE 6.2 – Comparaison des méthodes de recherche en fonction de la vitesse de déplacement




(a) Taille nuée = 15

(b) Taille nuée = 30

(c) Taille nuée = 60

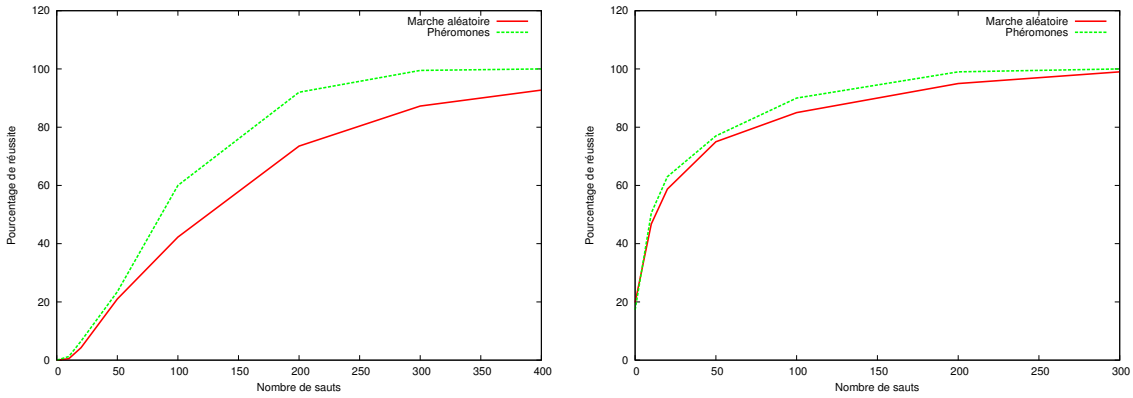
(d) Taille nuée = 100

FIGURE 6.3 – Comparaison des méthodes de recherche en fonction de la taille d'une nuée

La courbe 6.3(a) montre le nombre de sauts moyens et l'écart type en fonction de la vitesse de déplacement d'une nuée de 15 éléments. Nous pouvons observer que la vitesse ne fait quasiment pas varier le nombre de sauts et l'écart type quelque soit la méthode de recherche utilisée. La remarque précédente concernant l'influence de la vitesse de déplacement de la nuée est ici confirmée. L'ensemble des courbes obtenues montrent très peu de variations dans le nombre moyen de sauts et de l'écart type. Enfin l'écart de performance entre les deux algorithmes de recherche est confirmé pour une nuée de taille 15 (figure 6.3(a)).

### 6.3.3 Efficacité de la recherche

Pour mesurer l'efficacité de la recherche nous avons borné le nombre de sauts d'un agent de recherche et nous avons mesuré le pourcentage de réussite pour trouver une nuée en mouvement. Nous avons effectué les mesures sur des nuées de taille 10, 15 et 30. Étant donné le faible nombre de sauts moyen pour des nuées de taille 60 et 100 nous n'avons pas jugé utile de nous y intéresser.




(a) Taille nuée = 15

(b) Taille nuée = 30

FIGURE 6.4 – Performance des algorithmes de recherche

Dans le cas où la nuée est de taille 15 (figure 6.4(a)), l'algorithme utilisant des phéromones permet un gain significatif de performance. Pour 100 sauts l'écart entre les deux méthode de recherche est proche de 20%. Cette écart se confirme pour une borne fixée à 200 sauts. L'agent déposant des phéromones retrouvera une nuée avec 92% de réussite, et le marcheur aléatoire 74%. Dans le cas où la nuée est de taille 30 les deux algorithmes proposent des performances similaires, et à partir de 50 sauts nous approchons les 75% de réussite. Pour une borne à 200 sauts, les deux méthodes de recherche proposent un pourcentage de réussite supérieur à 95%.

Ces résultats permettent également de confirmer le fait que le mécanisme de recherche utilisant des phéromones est plus performant dans le cas de nuées de petite taille. L'écart de performance entre la marche aléatoire et le dépôt de phéromones est suffisamment conséquent pour envisager une méthode hybride de recherche. Dans le cas où des nuées de petite taille sont générées, il est préférable d'utiliser un agent déposant des phéromones dans le réseau, et dans

le cas où des nuées de taille supérieure à 20 par exemple, nous utiliserons une marche aléatoire plus économique en nombre de messages.

## Conclusion du chapitre

Nous avons présenté dans ce chapitre deux algorithmes de recherche. Le premier est une marche aléatoire et le second propose un déplacement dans le réseau basé sur l'utilisation de phéromones. Les expérimentations précédentes ont permis de montrer la faible influence de la vitesse de déplacement sur les agents recherche. Le second objectif était d'évaluer leur efficacité. Bien que l'algorithme utilisant des phéromones couvre le réseau plus rapidement qu'une marche aléatoire, les deux algorithmes proposent des résultats assez proches. Nous avons cependant pu remarquer qu'en fonction de la taille d'une nuée le nombre de sauts moyens pour retrouver un document varie.

Dans le cas d'une nuée de petite taille il est préférable d'utiliser un agent de recherche déposant des phéromones. Pour des nuées de taille 30 et supérieure nous préférerons utiliser une méthode à base de marche aléatoire, proposant des performances très proches de l'algorithme utilisant la stigmergie des agents et générant moins de messages dans le réseau.

# Chapitre 7

# De la simulation à l'application

**Sommaire**



Les chapitres précédents nous ont permis de tester nos différents algorithmes au travers du simulateur oRis. Au cours de cette thèse nous souhaitions sortir du cadre limitatif des simulations et observer le comportement de nos agents dans un environnement réel. Nous avons donc développé une application totalement décentralisée permettant d'évaluer nos algorithmes. L'objectif de ce chapitre est donc de présenter et d'introduire les premiers résultats obtenus dans le cadre d'une application déployée sur un réseau de l'ordre d'une centaine de machines. La première partie de ce chapitre sera consacrée à la présentation de la plate-forme d'évaluation construite sur le modèle établi dans le chapitre 4 et plus particulièrement aux détails d'implémentation des couches réseau et agents. Dans un second temps nous présenterons l'interface graphique et les fonctionnalités associées à notre outil. Enfin dans une troisième partie nous nous intéresserons aux résultats obtenus au travers de cette application.

## 7.1 Vue d'ensemble de la plate-forme

La plate-forme que nous proposons suit la modélisation introduite dans le chapitre 4. Nous avons donc implémenté une couche réseau reposant sur l'algorithme SCAMP, sur laquelle évoluent

des agents mobiles développés grâce au framework JavAct [Arcangeli *et al.*, 2004]. Le but de cette application est de valider notre approche à base de règles de flocking et non de proposer une solution de stockage d'information en tant que telle. Par le biais de cette plate-forme nous voulions vérifier la faisabilité d'une telle approche dans un environnement réel. Cette application constitue donc une étape préliminaire importante dans la conception d'une architecture de stockage de données répartie. Nous allons maintenant détailler les éléments majeurs constituant cette plate-forme.

### 7.1.1 Couche réseau

Nous avons réalisé avec Benoît Romito et Bruno D'Auria [D'Auria, 2009] la formalisation et l'implémentation des algorithmes nécessaires à l'établissement du protocole SCAMP. La plate-forme a été réalisée en JAVA et utilise le protocole de communication UDP nécessitant la gestion de l'acquittement des messages.

#### Structure d'un pair

Un pair dans le réseau est caractérisé par son adresse IP. Nous avons vu qu'un pair possédait plusieurs tables à sa disposition pour établir son voisinage. Un pair $x$ est donc composé :

- d'une table $IV$ (InView) répertoriant les adresses IP des nœuds connaissant $x$.
- d'une table $PV$ (PartialView) répertoriant les adresses IP des nœuds connus de $x$.

A chaque pair présent dans les tables $IV$ et $PV$ nous associons un poids nécessaire au mécanisme d'arrivée d'un pair dans le réseau ainsi qu'un RTT fournissant une estimation de la distance à laquelle se situent les pairs. La figure 7.1 décrit une telle structure.

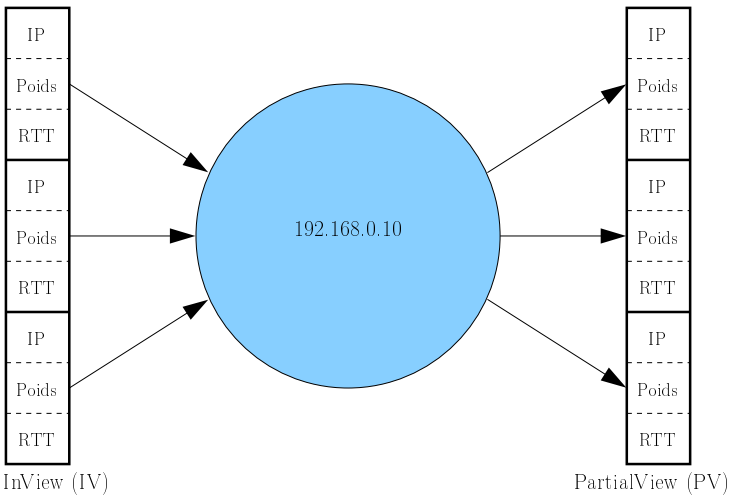



FIGURE 7.1 – Structure d'un pair d'adresse IP 192.168.0.10

### Messages pour la connexion/déconnexion d'un pair

Pour qu'un pair rejoigne le réseau, plusieurs étapes sont nécessaires. Il faut dans un premier temps faire une demande de souscription à un pair connu. Cette demande est ensuite propagée pour trouver le pair *contact* responsable de l'insertion du nœud. Pour rappel cette étape est basée sur une marche aléatoire et nécessite la propagation des poids associés aux arcs. Enfin la dernière étape consiste à établir les relations de voisinage entre les pairs. L'étape de déconnexion consiste pour un pair à prévenir les nœuds présents dans son voisinage puis de relier certains pairs de sa table $IV$ avec des membres de sa table $PV$. Pour réaliser l'ensemble des algorithmes nécessaires à l'établissement du réseau nous avons donc besoin de plusieurs types de messages :

- Le message `subscribe` effectuant la demande de souscription au réseau.
- Le message `updateweight` permettant la mise à jour locale du poids d'un arc et provoquant la modification de ce dernier dans la table du destinataire.
- Le message `forward` propageant l'identifiant d'un pair souhaitant rejoindre le réseau. Ce message fait suite au message `subscribe`.
- Le message `unsubscribe` correspondant à une demande de déconnexion. Lorsqu'un tel message est reçu par un pair, ce dernier supprime l'émetteur de sa $PV$.
- Le message `inform` envoyé dans le cadre d'une déconnexion aux membres de la table $IV$ devant effectuer une reconnexion avec des pairs issus de la table $PV$. Sur réception d'un tel message le pair ajoute l'adresse IP contenue dans le message à sa table $PV$.

### Messages pour le maintien du réseau

Pour le maintien du réseau nous utilisons un message `hello`. Le but de ce message est d'éviter l'isolement de pairs dans le réseau. Ainsi quand un pair émet un tel message il informe ses successeurs dans le réseau qu'ils ne sont pas isolés. Ce message, envoyé périodiquement à l'ensemble des éléments de la $PV$, permet au pair émetteur de détecter une éventuelle défaillance d'un des membres de son voisinage. Enfin ce message autorise un pair à notifier à un de ses successeurs qu'il est devenu son parent.

Chaque destinataire d'un tel message doit obligatoirement l'acquitter. Si un pair $x$ envoie périodiquement $n$ messages `hello` qui n'ont pas été acquittés par un pair $y$ alors le pair $x$ considère que le pair $y$ n'est plus présent dans le réseau et le déconnecte en le supprimant de sa table $PV$.

Tout nœud du réseau est également capable de détecter la présence d'un de ses parents. Chaque pair associe un *timer* à chaque membre de la table $IV$. Quand le *timer* expire cela signifie qu'un pair parent n'a pas envoyé de message `hello` dans les délais et il se retrouve donc supprimé de la table $IV$. Nous considérons un pair isolé quand sa table $IV$ est vide.

Enfin, la réception par un pair $x$ d'un message `hello` venant d'un élément n'étant pas présent dans sa table $IV$ entraîne l'ajout de l'émetteur dans cette dernière. Cela signifie que l'émetteur du message vient juste d'ajouter $x$ à sa table $PV$.

#### Messages pour l'estimation de la distance

Nous avons vu dans le chapitre 5 que nous pouvons utiliser un RTT pour estimer la distance entre deux pairs dans le réseau. Pour réaliser cette mesure nous nous basons sur les messages `RTTestablish`, `RTTresponse`, et `RTTinform`. Lorsqu'un pair souhaite connaître son RTT avec un pair de son voisinage, il va envoyer un message `RTTestablish`. Ce message contient la date $t_1$ d'envoi. Sur réception d'un tel message, un pair enregistre la date de réception $t_2$, et envoi un message `RTTresponse` contenant $\Delta_t = t_3 - t_2$, avec $t_3$ la date d'émission du message. Sur réception au temps $t_4$ d'un message `RTTresponse`, un pair calcule le RTT correspondant à $t_4 - t_1 - \Delta_t$. Il ne reste plus qu'a propager cette valeur par l'intermédiaire d'un message `RTTinform` contenant la valeur RTT calculée. La figure 5.1 présente dans le chapitre 5 montre un tel échange de message.

Nous avons recensé l'ensemble des messages nécessaires pour la gestion de la couche réseau dans le tableau 7.1.

### 7.1.2 Couche agents mobiles

Pour construire l'algorithme de flocking tel qu'il a été énoncé dans le chapitre 5 nous complétons la structure d'un pair par l'ajout des niveaux de phéromones et de confiance associés à chaque pair du voisinage. L'agent respectant un tel algorithme est appelé `agentFlock`.

Pour identifier les agents, et distinguer deux agents appartenant à deux nuées différentes, nous fournissons deux identifiants à chaque agent. Le premier est l'identifiant propre de l'agent, et le second l'identifiant de la nuée que l'agent représente.

Pour se déplacer un agent `agentFlock` doit tout d'abord récupérer les informations stockées sur le pair sur lequel il se situe puis il doit déposer des phéromones sur le lien par lequel il vient d'arriver pour marquer son précédent déplacement. Une fois ces étapes préliminaires effectuées, l'agent peut envisager de trouver un nouveau lieu de stockage.

Pour cela un agent va interroger les pairs de son voisinage composé des tables *IV* et *PV* pour récupérer les informations nécessaires à l'établissement des règles de flocking. Le but est de construire les ensembles *PairsOccupés* et *PairsLibres* présents dans l'algorithme 2 contenant respectivement les pairs occupés par un agent issu de la même nuée et les pairs n'en possédant pas. Cette construction se fait par l'intermédiaire d'un message `getAgents` prenant en paramètre l'identité de la nuée de l'agent.

La dernière étape de l'algorithme de flocking repose sur le respect de la règle de cohésion. Pour cela nous utilisons un message `getPeerFlock` déterminant les pairs issus de *PairsLibres* ne violant pas la règle de cohésion avec les pairs présents dans *PairsOccupés*.

Il ne reste alors plus qu'à un `agentFlock` a choisir parmi les pairs obtenus par `getPeerFlock` un pair possédant le meilleur score suivant les critères de phéromones et de confiance. Le tableau 7.1 liste les différents messages utilisés par un agent de type `agentFlock`.

| Nom du message | Contenu | Destinataire | Objectif |
|---|---|---|---|
| ***Réseau : connexion d'un pair*** | | | |
| `subscribe` | IP d'un pair entrant | $p \in PV$ | Trouver le pair *contact* |
| `forward` | IP d'un pair entrant | $p \in PV$ | Etablir le voisinage du pair entrant |
| `updateweight` | poids de $p \in IV \cup PV$ | $p \in IV \cup PV$ | Mise à jour du poids des arcs |
| ***Réseau : déconnexion d'un pair*** | | | |
| `unsubscribe` | IP de l'émetteur | $p \in IV$ | Demande de déconnexion |
| `inform` | $p_1 \in PV$ | $p_2 \in IV$ | Connexion d'un pair $p_1 \in IV$ avec un pair $p_2 \in PV$ |
| ***Réseau : maintien du réseau*** | | | |
| `hello` | - | $p \in PV$ | Détection de pairs isolés<br>Établissement du lien de parenté avec un pair |
| ***Réseau : estimation de la distance*** | | | |
| `RTTestablish` | temps $t$ | $p \in IV \cup PV$ | Demande de $RTT$ au temps $t_1$ |
| `RTTresponse` | temps $t$ | émetteur de `RTTestablish` | Réponse au temps $\Delta_t$ |
| `RTTinform` | $RTT$ | émetteur de `RTTresponse` | Propagation de $RTT$ |
| ***Agents : déplacement*** | | | |
| `getAgents` | $id_{nuee}$ | $p \in V_x$ | Construire *PairsLibres*<br>Construire *PairsOccupés* |
| `getPeerFlock` | *PairsOccupés* | $p \in$ *PairsLibres* | Construire la liste des déplacements possibles |

TABLE 7.1 – Messages pour la gestion du réseau et des agents

## 7.2 Interface graphique

Pour visualiser et vérifier le fonctionnement des couches réseau et agent, nous avons enrichi avec Benoît Romito notre plate-forme d'une interface graphique. C'est au travers de cette interface que nous pouvons interagir avec les pairs du réseau et les agents. Nous allons maintenant détailler les principales fonctionnalités offertes par cette interface graphique.

### 7.2.1 Connexion

L'interface que nous proposons repose sur différents onglets. La gestion du réseau est centrée sur les onglets **Options**, **Logs viewer**, et **Nodes launcher**. Le premier onglet, l'onglet **Options**, recense l'ensemble des paramètres nécessaires pour établir la connexion des pairs. Nous y trouvons les délais entre l'arrivée de deux pairs dans le réseau, ainsi que les paramètres liés à l'initialisation de la machine virtuelle Java. L'onglet **Nodes launcher** contient la liste des machines à connecter au réseau. La figure 7.2 propose une capture d'écran de l'onglet **Nodes launcher**.

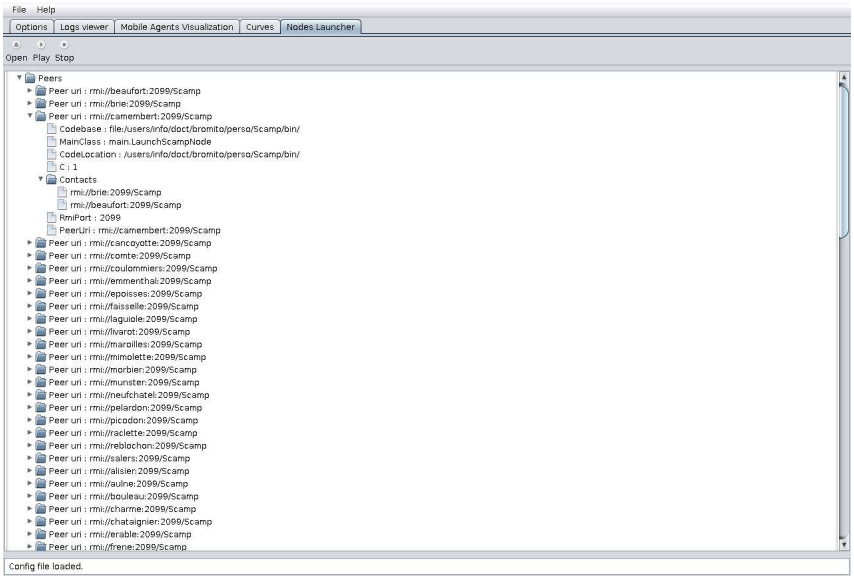



FIGURE 7.2 – Capture d'écran de l'onglet Nodes launcher

Pour fournir la liste des pairs, nous avons recours à un fichier au format xml recensant l'ensemble des caractéristiques d'un nœud. Chaque pair, identifié par un *Uniform Resource Identifier* (URI) et un numéro de port sur lequel est exécuté la machine virtuelle, possède une liste de pairs pouvant jouer le rôle de passerelle pour la connexion. Pour vérifier la construction du réseau, chaque pair utilise un fichier de log. L'onglet **Logs viewer** permet de les consulter.

### 7.2.2 Visualisation

La capture présentée dans la figure 7.3 montre l'onglet **Curves** contenant le graphique des propriétés du réseau. Ici sont représentées les courbes concernant le degré moyen du réseau, le seuil minimal de connexité en $(c)\log(n)$, ainsi que la valeur théorique de connexité en $(c+1)\log(n)$.

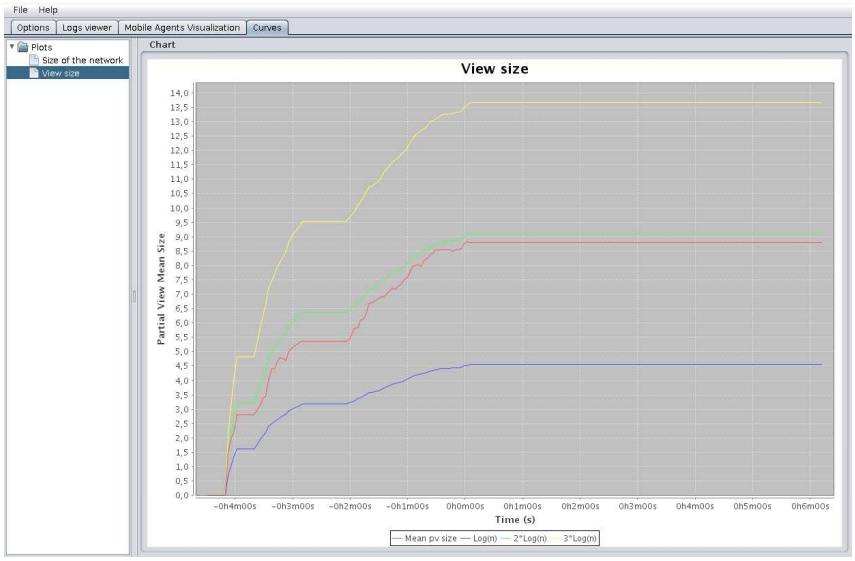



FIGURE 7.3 – Capture d'écran de l'onglet Curves

Nous proposons également une visualisation du réseau et des agents mobiles évoluant dessus. Une telle vue se situe dans l'onglet **Mobile Agents Visualization**, et est présentée dans les figures 7.4 et 7.5.

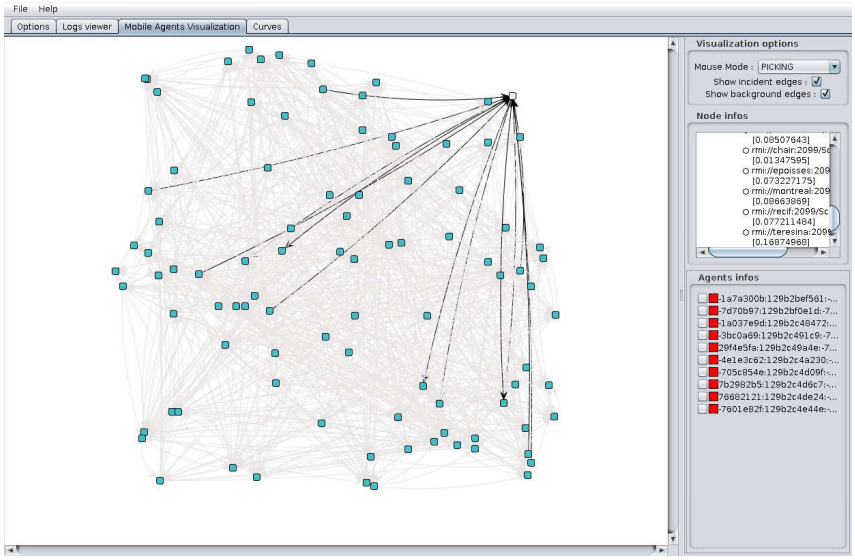



FIGURE 7.4 – Capture d'écran de l'onglet Mobile Agents Visualization - Vue réseau

En cliquant sur un pair il est possible d'obtenir et d'afficher les informations stockées dessus. Une telle action met également en évidence les relations de voisinage représentées par des flèches noires (figure 7.4). Ces informations sont complétées par la présence d'un cadre contenant les données liées au voisinage. Cette partie visible sur la partie droite de la figure 7.4 contient les adresses URI des nœuds ainsi que leur poids, et leur RTT. Pour faciliter la représentation du réseau, il est également possible de sélectionner un pair et de le déplacer.

L'onglet **Mobile Agent Visualization** offre également la possibilité de visualiser les agents et leurs déplacements dans le réseau. Sur la figure 7.5 les agents sont représentés par des ronds rouges. Il est possible d'observer les caractéristiques des agents, tels que leurs identifiants et l'identifiant de la nuée à laquelle ils appartiennent dans le coin inférieur droit de l'interface graphique.

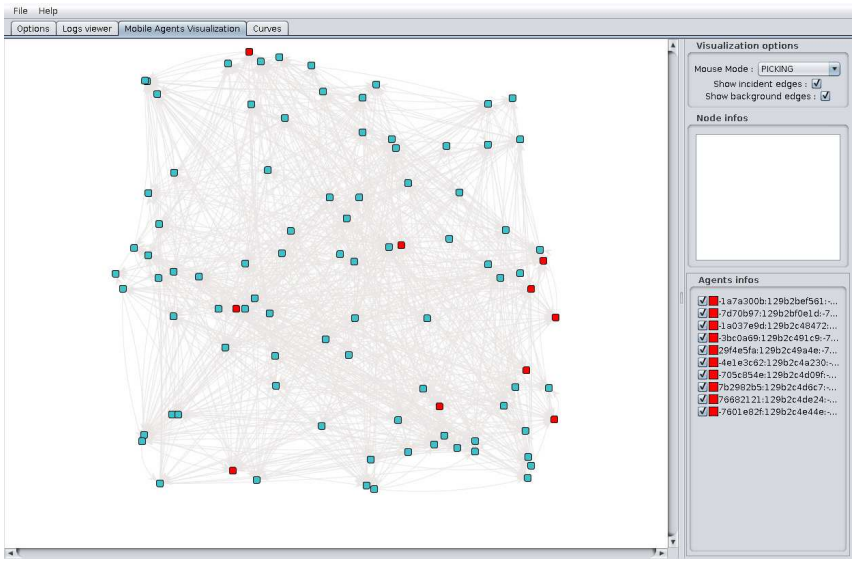



FIGURE 7.5 – Capture d'écran de l'onglet Mobile Agents Visualization - Vue agent

## 7.3 Résultats expérimentaux

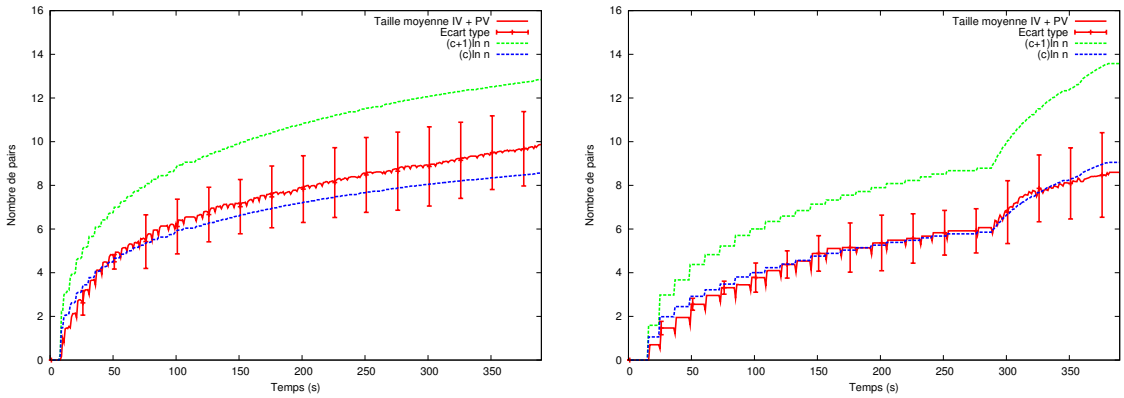
Le but de notre application est de vérifier la validité de notre approche dans un contexte sortant du cadre des simulations. Nous avons déployé des agents sur un réseau de type SCAMP composé de 100 machines physiques du département informatique de l'Université de Caen Basse-Normandie. Nous avons ainsi pu mesurer :

- Le degré moyen du réseau obtenu pour vérifier les propriétés de construction du graphe.
- La connexité d'une nuée en déplacement pour valider les règles de flocking.
- La couverture du réseau en nombre de machines parcourus pour observer la répartition de charge et la mobilité de la nuée.

Les résultats suivants ont été publiés dans [Pommier *et al.*, 2010a, Pommier *et al.*, 2010b].

### 7.3.1 Couche réseau

La première série d'expérimentations consiste à évaluer la construction du réseau. La figure 7.6 montre le degré moyen du réseau suivant deux configurations. La première, représentée sur la figure 7.6(a), montre une construction où une centaine de pairs rejoignent le graphe linéairement répartis dans le temps. La figure 7.6(b) décrit le cas où un pic de connexion à lieu au temps $t = 290s$. Ici seulement 25 nœuds se sont connectés, puis 75 pairs rejoignent le réseau simultanément. Sur les deux figures, les courbes pointillées vertes représentent la valeur théorique du degré attendu soit $(c+1)\log(n)$, les courbes pleines rouges la taille moyenne des tables $IV$ et $PV$ ainsi que l'écart type, et les courbes pointillées bleues le seuil limite de connexité soit $(c)\log(n)$. Chaque expérience a été répétée 10 fois.




(a) Connexion linéaire

(b) Pic de connexion à $t = 290s$

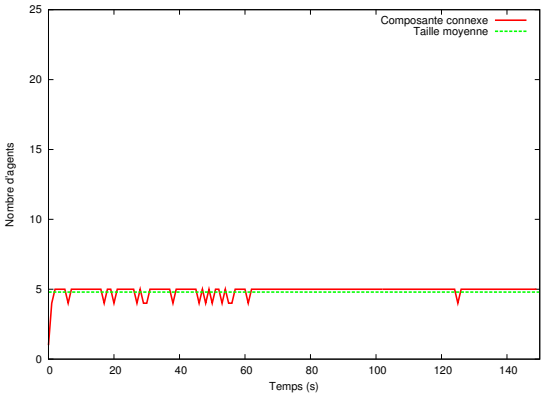
FIGURE 7.6 – Convergence du degré moyen d'un réseau de 100 pairs

À la vue de la figure 7.6(a), nous pouvons observer que pour 100 nœuds, le degré moyen du graphe reste contenu entre $(c+1)\log(n)$ et $(c)log(n)$. Bien que le degré moyen reste au-dessus du seuil de connexité la convergence n'est pas nette. Pour remédier à cette situation il faudrait utiliser un plus grand nombre de machines. Le résultat observé sur la figure 7.6(b) vérifie ce qui a été postulé dans le chapitre 4. Le degré moyen du graphe en cas de pic est plus faible que le degré moyen sans présence de pic. La construction du graphe a donc clairement un impact sur son degré moyen.

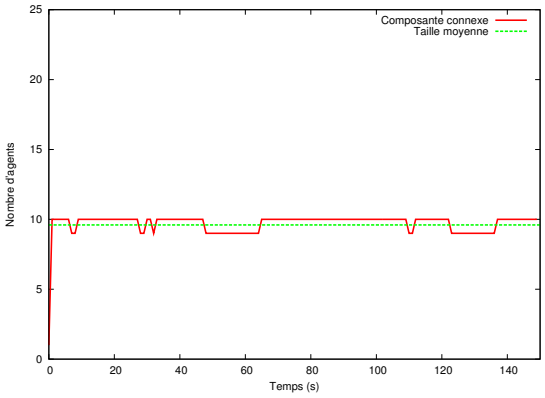
### 7.3.2 Évaluation des règles de flocking

Nous avons déployé des agents mobiles dans un réseau construit avec SCAMP pour mesurer la taille et la cohésion de la nuée générée. La constante $c$ pour SCAMP a été fixée à 3, chaque agent dépose 2 unités de phéromones à chaque déplacement, et le taux d'évaporation à été fixé à $0,1\%$ toutes les secondes. Nous avons disséminé des nuées de 5, 10, et 20 agents. La figure 7.7 montre les résultats dans un réseau de 50 machines et la figure 7.8 décrit les résultats dans un réseau de 100 machines. Dans les figures 7.7 et 7.8, les courbes décrivent le nombre de fragments

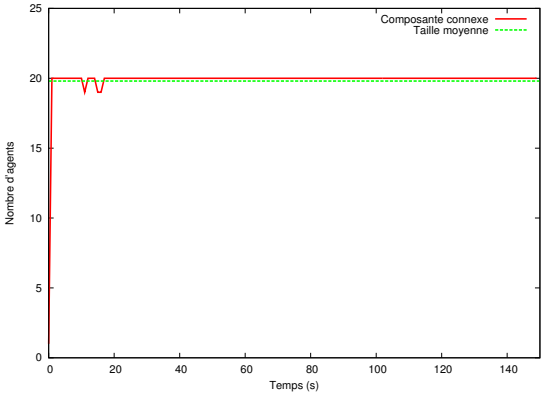
non-isolés d'une nuée au cours du temps. Un agent non-isolé est un agent possédant au moins un fragment dans son voisinage.




(a) Nuée de 5 fragments




(b) Nuée de 10 fragments

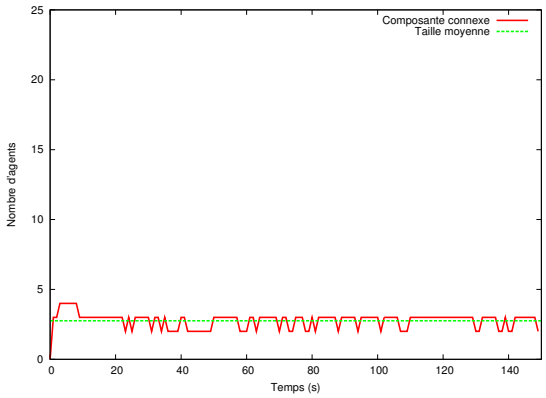



(c) Nuée de 20 fragments

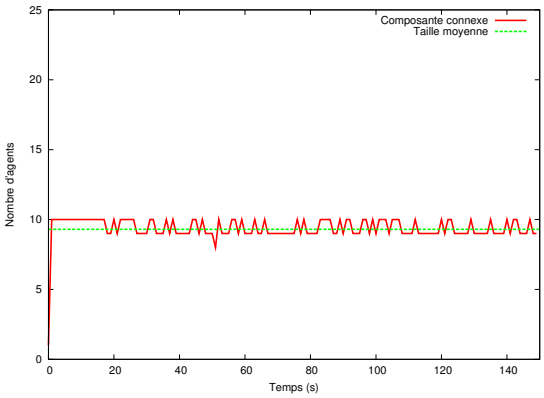
FIGURE 7.7 – Connexité d'une nuée dans un réseau de 50 pairs

Dans le cas où le réseau est constitué de 50 pairs, nous pouvons voir que la nuée reste connexe, quelque soit sa taille. Pour 5 fragments (figure 7.7(a)) seulement un fragment peut se retrouver isolé et la moyenne de la plus grande composante connexe est supérieure à 4. Le même phénomène est observable pour 10 et 20 agents (figures 7.7(b) et 7.7(c)). Ceci s'explique par la taille réduite du réseau construit entraînant un faible nombre de voisins pour un pair. Nous pouvons constater ici que les règles de flocking fonctionnent, un agent ne restant jamais isolé dans le temps.

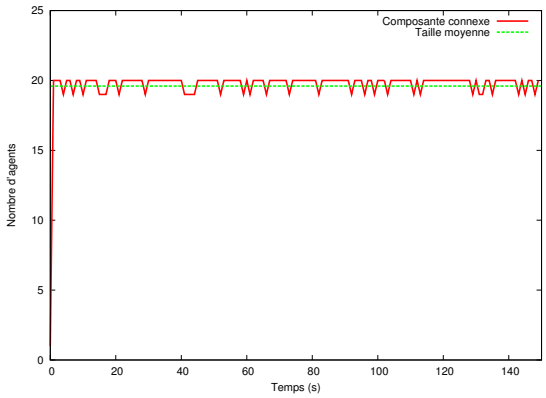
Pour un réseau de 100 machines, la connexité d'une nuée est moins stable. Les résultats visibles sur la figure 7.8(a) décrivent le comportement d'une nuée de 5 agents. Ici la nuée ne se déplace sous sa taille maximale qu'un faible intervalle de temps à la dissémination initiale des fragments, puis se scinde en plusieurs parties, la plus grande composante étant en moyenne de taille inférieure à 3. Cette moyenne indique la perte de 1 à 2 fragments. Pour une nuée de 10 agents (figure 7.8(b)), les résultats sont meilleurs et indiquent une plus forte cohésion. La courbe




(a) Nuée de 5 fragments




(b) Nuée de 10 fragments




(c) Nuée de 20 fragments

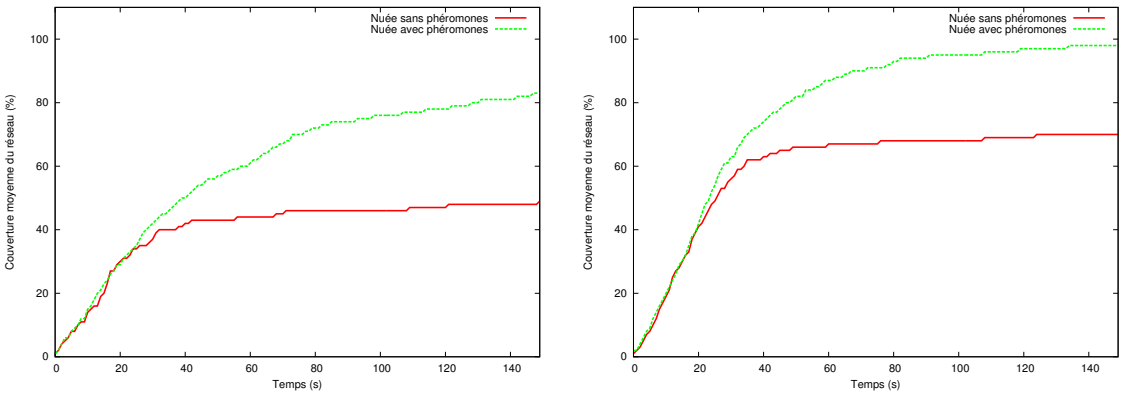
FIGURE 7.8 – Connexité d'une nuée dans un réseau de 100 pairs

oscille ici entre 9 et 10 fragments, pour une moyenne de connexité supérieure à 9. Enfin pour une nuée de taille 20 (figure 7.8(c)), la connexité est élevée et les agents se déplacent sous la forme d'un groupe connexe complet la majeure partie du temps.

Ces résultats nous permettent de valider le comportement de nos agents. Dans la figure 7.7, les nuées constituées de 5, 10, et 20 fragments restent groupées dans le temps et la comparaison des figures 7.7 et figures 7.8 montre que la taille du réseau influence clairement la cohésion d'une nuée. Ces résultats confirment ceux observés dans le chapitre 5. Le comportement de groupe est donc dépendant de la taille du réseau et par conséquent du nombre de voisins d'un pair.

### 7.3.3 Répartition de charge

Nous avons testé la couverture du réseau et la figure 7.9 montre les résultats pour des nuées de 10 et 20 agents sur une même instance d'un réseau composé de 100 pairs. Les courbes représentent le pourcentage de machines ayant été visitées en fonction du temps, chaque fragment se déplaçant toutes les secondes.




(a) Nuée de 10 fragments

(b) Nuée de 20 fragments

FIGURE 7.9 – Couverture moyenne du réseau

Sur les figures 7.9(a) et 7.9(b) la courbe pleine rouge décrit la couverture du réseau lorsque les phéromones sont désactivées et la courbe pointillée verte lorsque les agents déposent des phéromones. Dans les deux cas, le fait d'utiliser des phéromones permet de parcourir plus de machines dans le réseau, et permet une plus grande mobilité de la nuée. Pour une nuée de 10 agents (figure 7.9(a)) seulement 50% du réseau est parcouru par le groupe d'agent quand les phéromones sont désactivées. Cette valeur monte à plus de 80% quand nous activons la capacité de stigmergie des agents. Nous pouvons également remarquer que ces résultats sont assez proches de ceux obtenus lors des simulations précédentes. Les résultats sont similaires pour une nuée de 20 éléments (figure 7.9(b)) avec une couverture supérieure à 70% sans phéromones et proche de 100% avec phéromones.

## Conclusion du chapitre

Pour valider notre proposition de placement de l'information dynamique prenant en compte les capacités d'interaction avec l'environnement des agents, nous avons proposé une plate-forme permettant un déploiement d'agents mobiles dans un réseau pair-à-pair. Nous avons implémenté l'ensemble des mécanismes nécessaires à la construction et au maintien du réseau, et nous avons utilisé la librairie JavAct pour implanter le comportement de nos agents mobiles. Les résultats obtenus confirment ceux obtenus dans les chapitres 4 et 5. Nous avons confirmé le fait que le degré moyen du réseau est sensible aux pics de connexion. Nous avons également vu que la connexité d'une nuée est vérifiée dans le temps et que la mobilité des agents permet de parcourir l'ensemble du réseau. Ces résultats valident notre approche reposant sur l'algorithme de flocking que nous avons proposé dans le chapitre 5.

# Conclusion de la partie II

Dans cette partie, nous avons présenté nos contributions liées à la gestion de l'information répartie. Nous avons défini dans le chapitre 4 une modélisation en couches organisant l'ensemble des éléments nécessaires à un placement de l'information sous la forme de nuée. Nous avons également présenté dans ce chapitre la couche réseau de notre architecture établie grâce aux algorithmes du protocole SCAMP. Les expérimentations menées valident les convergences attendues pour le degré moyen du réseau et pour la somme des poids des vues de chaque pair.

Le chapitre 5 introduit la notion de placement de l'information "bio-inspiré". Nous avons adapté au monde des réseaux pair-à-pair les règles de flocking énoncées par Reynolds. Chaque agent prend en compte dans son déplacement des contraintes de cohésion et de séparation pour se déplacer sous la forme d'une nuée. Nous utilisons les résultats issus des algorithmes de répartition de charge et d'évaluation de la confiance construits à partir des dépôts de phéromones des agents pour proposer un algorithme de déplacement. Les résultats observés ont validé notre approche. La cohésion d'une nuée est vérifiée dans le temps à la nuance près qu'elle est dépendante de la taille du réseau. La répartition de la charge s'effectue bien sur l'ensemble des pairs du réseau, et la détection de machines malveillante fonctionne.

Au chapitre 6, nous avons abordé la problématique de la recherche de fichiers. Nous avons proposé et implémenté deux types d'agent mobile responsables de la recherche d'une nuée. Le premier agent suit une marche aléatoire, et le second utilise un dépôt de phéromones pour couvrir le réseau plus rapidement. À partir des expériences menées, nous pouvons remarquer que les deux approches proposent des résultats assez similaires. Cependant, dans le cas de nuée de petite taille, les résultats indiquent une meilleure performance de l'algorithme utilisant des phéromones. Il apparaît également que la vitesse de déplacement des agents dans le réseau ne joue aucun rôle dans les performances des deux algorithmes.

Pour valider notre placement de l'information dans un contexte dépassant le cadre des simulations, nous avons déployé notre architecture sur un réseau constitué d'une centaine de machines physiques. Les résultats présents dans le chapitre 7 confirment ceux obtenus lors des simulations précédentes. Nous pouvons tout d'abord remarquer que la construction du réseau possède une influence sur le degré moyen. Ensuite les expériences montrent que la connexité d'une nuée, bien que celle-ci soit établie, est dépendante de la taille du réseau. Enfin la répartition de charge s'effectue bien sur l'ensemble des pairs du réseau.

# Troisième partie

# Conclusion et perspectives

# Chapitre 8

# Conclusion

## 8.1 Bilan

Dans cette thèse nous avons proposé une approche originale consistant à utiliser des règles issues du vivant pour placer l'information dans un réseau informatique. Nous avons centré nos travaux sur trois axes, les réseaux pair-à-pair, la théorie de l'information, et les systèmes multi-agents bio-inspirés composés d'agents mobiles. Chaque axe constitue un élément que nous avons organisé dans une architecture en couche.

#### Placement de l'information

Nous considérons l'information comme un système multi-agents. Chaque membre héberge un morceau de donnée issu d'une étape de fragmentation assurant la disponibilité et la sûreté de l'information. Chaque fragment est alors vu comme une entité autonome capable de prendre ses propres décisions. Pour faciliter la recherche d'information, nous avons donné à nos agents des règles leur permettant de se déplacer sous la forme d'une nuée similaire à celle observée chez les oiseaux. Ces règles sont basées sur l'utilisation d'une distance. Ainsi un agent va chercher à se déplacer dans l'objectif de se rapprocher d'agents éloignés sans pour autant violer une distance minimale de cohésion.

À partir du comportement de nos agents et de leur capacité d'interaction avec l'environnement, nous avons construit des algorithmes permettant une répartition de charge, et évaluant la confiance des pairs du réseau. La force de ces algorithmes réside dans leur caractère décentralisé et sur le fait qu'ils sont uniquement obtenu grâce à la capacité de stigmergie des agents. Ainsi chaque agent est capable de déposer dans le réseau des phéromones indiquant son passage. La répartition de charge consiste alors pour un agent à emprunter les chemins possédant le moins de phéromones. L'algorithme de confiance que nous avons établi est construit sur la propriété que deux pairs reliés par une relation de voisinage possèdent un niveau équivalent de phéromones.

Les différentes expérimentations ont validé l'ensemble des algorithmes proposés, et ont mis en lumière la dépendance de la taille d'une nuée à la taille du réseau.

#### Recherche de l'information

Nous avons proposé deux algorithmes de recherche permettant de contacter un membre d'une nuée pour retrouver la totalité d'un document. Le premier agent de recherche que nous avons construit repose sur une marche aléatoire, également appelée chaîne de Markov. Le second agent se déplace quant à lui aléatoirement dans le réseau et utilise ses capacités de stigmergie en déposant des phéromones pour marquer son passage.

Au cours de nos expérimentations, l'algorithme utilisant les phéromones s'est avéré plus efficace qu'un marcheur aléatoire dans le cas de nuées de petite taille. Pour des nuées de taille supérieure, les performances des deux algorithmes sont similaires.

#### Application à un réseau physique

Nous avons construit une plate-forme reposant sur un réseau totalement décentralisé, où les opérations nécessaires à la connexion/déconnexion des pairs et au maintien de la structure sont peu coûteuses. L'ajout d'agents mobiles à cette plate-forme nous a permis d'implémenter le déplacement en nuée.

Nous avons déployé cette application sur un réseau physique d'une centaine de machines. Les résultats obtenus confirment dans leur ensemble la validité de notre approche précédemment observée sous simulateur.

## 8.2 Perspectives

Les résultats obtenus ont soulevé certaines hypothèses qu'il nous semble judicieux d'étudier.

#### Paramétrage de la taille de la nuée

**La connexité de la nuée est dépendante de la taille du voisinage** Nous avons pu voir qu'une nuée de petite taille plongée dans un réseau de grande taille adoptait un comportement chaotique et n'était pas en mesure de conserver sa cohésion. Ce phénomène est du à une taille de voisinage trop élevée, les choix de déplacements pour un agent étant alors nombreux. Il semblerait intéressant d'étudier la relation de dépendance entre la taille d'une nuée et la taille du voisinage d'un pair.

**Le choix d'une méthode de recherche dépend de la taille de la nuée** Lors de la recherche de fichiers nous avons pu remarquer que l'algorithme utilisant un dépôt de phéromones est plus efficace qu'une marche aléatoire dans le cas où la nuée est de petite taille. Nos observations expérimentales laissent supposer l'existence d'un seuil à partir duquel les deux méthodes sont équivalentes. Une étude approfondie permettrait de déterminer ce seuil et d'en évaluer l'impact sur le placement de l'information.

### Impacts sur la sûreté du modèle

**Rendre la reconstruction d'informations plus robuste** Dans le cadre de la reconstruction de fichiers, il est nécessaire d'accéder aux éléments de la nuée. Les mécanismes usuels se basent sur des données statiques or les éléments du réseau sont dynamiques. Notre approche basée sur la mobilité prend en compte les évolutions du réseau. Nous pensons que notre méthode pourrait être appliquée avec succès à la reconstruction en déplaçant les fragments sur des pairs aux performances plus élevées.

# Bibliographie

# Bibliographie


[Acedański *et al.*, 2005] Acedański, S., Deb, S., Médard, M. et Koetter, R. (2005). How good is random linear coding based distributed networked storage. *In the 1st Workshop on Network Coding, Theory, and Applications*, NETCOD '05, Riva del Garda, Italy.

[Adya *et al.*, 2002] Adya, A., Bolosky, W. J., Castro, M., Cermak, G., Chaiken, R., Douceur, J. R., Howell, J., Lorch, J. R., Theimer, M. et Wattenhofer, R. (2002). Farsite : Federated, available, and reliable storage for an incompletely trusted environment. *In the 5th Symposium on Operating Systems Design and Implementation*, OSDI '02, Boston, MA, USA.

[Agha, 1986] Agha, G. (1986). *Actors : a model of concurrent computation in distributed systems*. MIT Press, Cambridge, MA, USA.

[Aglets, 1997] Aglets (1997). http://aglets.sourceforge.net/.

[Ahlswede *et al.*, 2002] Ahlswede, R., Cai, N., Li, S.-Y. R. et Yeung, R. W. (2002). Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216.

[ARA, 1997] ARA (1997). http://wwwagss.informatik.uni-kl.de/Projekte/Ara/index_e.html.

[Arcangeli *et al.*, 2004] Arcangeli, J.-P., Hennebert, V., Leriche, S., Migeon, F. et Pantel, M. (2004). Javact 0.5.0 : principes, installation, utilisation et développement d'applications. Rapport technique, IRIT/2004-5-R, Toulouse.

[Authenticode, 1996] Authenticode (1996). http://msdn.microsoft.com/fr-fr/library/ms172240.aspx.

[Baset et Schulzrinne, 2004] Baset, S. A. et Schulzrinne, H. (2004). An analysis of the skype peer-to-peer internel telephony protocol. Rapport technique, Departement of Computer Science, Columbia University.

[Bellifemine *et al.*, 1999] Bellifemine, F., Poggi, A. et Rimassa, G. (1999). JADE - a FIPA-compliant agent framework. *In the 4th International Conference on the Practical Applications of Intelligent Agents*, PAAM-99, pages 97–108, London, UK. The Practical Application Company Ltd.

[Beynier et Mouaddib, 2009] Beynier, A. et Mouaddib, A.-I. (2009). Decentralized decision making process for document server networks. *In the 1st International ICST Conference on Game Theory for Networks*, GameNets '09, Istanbul, Turquie. IEEE.

[Bisnik et Abouzeid, 2005] Bisnik, N. et Abouzeid, A. (2005). Modeling and analysis of random walk search algorithms in p2p networks. *In the 2nd International Workshop on Hot Topics in Peer-to-Peer Systems*, HOT-P2P '05, pages 95–103, La Jolla, CA, USA. IEEE Computer Society.

[Blakley, 1979] Blakley, G. R. (1979). Safeguarding cryptographic keys. *In the AFIPS National Computer Conference*, NCC '79, pages 313–317, Monval, NJ, USA. AFIPS Press.

[Blakley et Meadows, 1985] Blakley, G. R. et Meadows, C. (1985). Security of ramp scheme. *In Proceedings of CRYPTO 84 on Advances in cryptology*, CRYPTO '84, pages 242–268, Santa Barbara, CA, USA. Springer-Verlag.

[Blömer *et al.*, 1995] Blömer, J., Kalfane, M., Karpinski, M., Karp, R., Luby, M. et Zuckerman, D. (1995). An xor-based erasure-resilient coding scheme. Rapport technique TR-95-048, International Computer Science Division, UC Berkeley, Berkeley, CA, USA.

[Bredin *et al.*, 1998] Bredin, J., Kotz, D. et Rus, D. (1998). Market-based resource control for mobile agents. *In the 2nd International Conference on Autonomous Agents*, AGENTS '98, pages 197–204, Minneapolis, MN, USA. ACM.

[Brustoloni, 1991] Brustoloni, J. C. (1991). Autonomous agents : Characterization and requirements. Rapport technique CMU-CS-91-204, School of Computer Science, Carnegie Mellon University, Pittsburg, PA, USA.

[Camazine *et al.*, 2001] Camazine, S., Franks, N. R., Sneyd, J., Bonabeau, E., Deneubourg, J.-L. et Theraula, G. (2001). *Self-Organization in Biological Systems*. Princeton University Press, Princeton, NJ, USA.

[Camorlinga *et al.*, 2004] Camorlinga, S., Barker, K. et Anderson, J. (2004). Multiagent systems for resource allocation in peer-to-peer systems. *In the Winter International Synposium on Information and Communication Technologies*, WISICT '04, pages 1–6, Cancun, Mexico. Trinity College Dublin.

[Canepa et Potop-Butucaru, 2007] Canepa, D. et Potop-Butucaru, M. G. (2007). Stabilizing flocking via leader election in robot networks. *In the 9th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, SSS 2007, pages 52–66, Paris, France. Springer.

[Cao *et al.*, 2003] Cao, J., Sun, Y., Wang, X. et Das, S. K. (2003). Scalable load balancing on distributed web servers using mobile agents. *Journal of Parallel and Distributed Computing*, 63(10):996–1005.

[Chen *et al.*, 2002] Chen, Y., Katz, R. H. et Kubiatowicz, J. D. (2002). Dynamic replica placement for scalable content delivery. *In the 1st International Workshop on Peer-to-Peer Systems*, IPTPS '02, pages 306–318, Cambridge, MA, USA. Springer.

[Clarke *et al.*, 2000] Clarke, I., Sandberg, O., Wiley, B. et Hong, T. W. (2000). Freenet : A distributed anonymous information storage and retrieval system. *In Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, Berkeley, CA, USA.

[Colorni *et al.*, 1991] Colorni, A., Dorigo, M. et Maniezzo, V. (1991). Distributed optimization by ant colonies. *In the 1st European Conference on Artificial Life*, ECAL'91, pages 134–142, Paris, France. MIT Press.

[Cubat dit Cros, 2005] Cubat dit Cros, C. (2005). *Agents Mobiles Coopérants pour les Environnements Dynamiques*. Thèse de doctorat, Institut National Polytechnique de Toulouse.

[Dabek *et al.*, 2001] Dabek, F., Kaashoek, M. F., Karger, D., Morris, R. et Stoica, I. (2001). Wide-area cooperative storage with cfs. *In the 18th ACM Symposium on Operating Systems Principles*, SOSP'01, pages 202–215, Chateau Lake Louise, Banff, Canada. ACM.

[D'Agents, 1995] D'Agents (1995). http://www.tacoma.cs.uit.no/.

[Dagnat *et al.*, 2000] Dagnat, F., Pantel, M., Colin, M. et Sallé, P. (2000). Typing concurrent objects and actors. *L'Objet*, 6(1):83–106.

[D'Auria, 2009] D'Auria, B. (2009). Implémentation d'une couche de transport épidémique sur une architecture pair-à-pair pour la dissémination d'agents mobiles. Rapport de stage DUT, Institut Universitaire Technologique de Caen Basse-Normandie, Caen, France.

[Demazeau, 1995] Demazeau, Y. (1995). From interactions to collective behaviour in agent-based systems. *In the 1st European Conference on Cognitive Science*, pages 117–132, Saint Malo, France.

[Demazeau, 1997] Demazeau, Y. (1997). Steps towards multi-agent oriented programming. *In the 1st International Workshop on Multi-Agent Systems*, IWMAS'97, Boston, MA, USA.

[Demers *et al.*, 1987] Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D. et Terry, D. (1987). Epidemic algorithms for replicated database maintenance. *In the 6th annual ACM Symposium on Principles Of Distributed Computing*, PODC '87, pages 1–12, Vancouver, Canada. ACM.

[Deswarte *et al.*, 1991] Deswarte, Y., Blain, L. et Fabre, J.-C. (1991). Intrusion tolerance in distributed computing systems. *In IEEE Symposium on Security and Privacy*, pages 110–121.

[Dillenseger *et al.*, 2002] Dillenseger, B., Tagant, A.-M. et Hazard, L. (2002). Programming and executing telecommunication service logic with moorea reactive mobile agents. *In the 4th International Workshop Mobile Agents for Telecommunication Applications*, MATA 2002, pages 48–57. Springer.

[Dimakis *et al.*, 2007] Dimakis, A. G., Godfrey, B., Wainwright, M. J. et Ramchandran, K. (2007). Network coding for distributed storage systems. *In the 26th IEEE International Conference on Computer Communications*, INFOCOM '07, pages 2000–2008, Anchorage, AK, USA. IEEE.

[Dimakis *et al.*, 2006] Dimakis, A. G., Prabhakaran, V. et Ramchandran, K. (2006). Decentralized erasure codes for distributed networked storage. *IEEE/ACM Transactions on Networking*, 14(SI):2809–2816.

[Douceur et Wattenhofer, 2001] Douceur, J. R. et Wattenhofer, R. P. (2001). Competitive hill-climbing strategies for replica placement in a distributed file system. *Lecture Notes in Computer Science*, 2180:48–62.

[Drogoul, 1993] Drogoul, A. (1993). *De la simulation multi-agent à la résolution collective de problèmes*. Thèse de doctorat, Université de Paris VI.

[Duminuco et Biersack, 2008] Duminuco, A. et Biersack, E. (2008). Hierarchical codes : How to make erasure codes attractive for peer-to-peer storage systems. *In the 8th International Conference on Peer-to-Peer Computing*, P2P '08, pages 89–98, Aachen, Germany. IEEE.

[Duminuco et Biersack, 2010] Duminuco, A. et Biersack, E. (2010). Hierarchical codes : A flexible trade-off for erasure codes in peer-to-peer storage systems. *Peer-to-Peer Networking and Applications*, 3(1):52–66.

[Duminuco et Biersack, 2009] Duminuco, A. et Biersack, E. W. (2009). A practical study of regenerating codes for peer-to-peer backup systems. *In the 29th International Conference on Distributed Computing Systems*, ICDCS '09, pages 376–384, Montreal, Canada. IEEE.

[Durfee *et al.*, 1987] Durfee, E., Lesser, V. et Corkill, D. (1987). Coherent Cooperation Among Communicating Problem Solvers. *IEEE Transactions on Computers*, C-36(11):1275–1291.

[eDonkey, 2000] eDonkey (2000). http://www.edonkey2000-france.com.

[El Falou, 2006] El Falou, S. (2006). *Programmation répartie, optimisation par agent mobile*. Thèse de doctorat, Université de Caen Basse-Normandie, Caen, France.

[eMule, 2002] eMule (2002). http://www.emule-project.net/.

[Erdös et Rényi, 1960] Erdös, P. et Rényi, A. (1960). On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, 5:17–61.

[Eugster *et al.*, 2003] Eugster, P. T., Guerraoui, R., Handurukande, S. B., Kouznetsov, P. et Kermarrec, A.-M. (2003). Lightweight probabilistic broadcast. *ACM Transactions on Computer Systems*, 21(4):341–374.

[Fabre *et al.*, 1994] Fabre, J.-C., Deswarte, Y. et Randell, B. (1994). Designing secure and reliable applications using fragmentation-redundancy-scattering : An object-oriented approach. *In the 1st European Dependable Computing Conference*, EDCC '94, pages 21–38, Berlin, Germany. Springer.

[FastTrack, 2001] FastTrack (2001). http://developer.berlios.de/projects/gift-fasttrack/.

[Ferber, 1995] Ferber, J. (1995). *Les systèmes multi-agents : vers une intelligence collective*. InterÉditions, Paris, France, 1ère édition.

[Fuggetta *et al.*, 1998] Fuggetta, A., Picco, G. P. et Vigna, G. (1998). Understanding code mobility. *IEEE Transactions on Software Engineering*, 24:342–361.

[Galand et Perny, 2006] Galand, L. et Perny, P. (2006). Search for compromise solutions in multiobjective state space graphs. *In the 17th European Conference on Artificial Intelligence*, ECAI 2006, pages 93–97, Riva del Garda, Italy. IOS Press.

[Ganesh *et al.*, 2003] Ganesh, A. J., Kermarrec, A.-M. et Massoulié, L. (2003). Peer-to-peer membership management for gossip-based protocols. *IEEE Transactons on Computers*, 52(2):139–149.

[Gervasi et Prencipe, 2004] Gervasi, V. et Prencipe, G. (2004). Coordination without communication : the case of the flocking problem. *Discrete Applied Mathematics*, 144(3):324–344.

[Ghemawat *et al.*, 2003] Ghemawat, S., Gobioff, H. et Leung, S.-T. (2003). The google file system. *In the 19th Symposium on Operating Systems Principles*, SOSP '03, pages 29–43, New York, NY, USA. ACM.

[Giroire *et al.*, 2009] Giroire, F., Monteiro, J. et Pérennes, S. (2009). P2p storage systems : How much locality can they tolerate ? *In the 34th IEEE Conference on Local Computer Networks*, LCN 2009, pages 320–323, Zurich, Switzerland. IEEE.

[Gkantsidis *et al.*, 2006] Gkantsidis, C., Mihail, M. et Saberi, A. (2006). Random walks in peer-to-peer networks : algorithms and evaluation. *Performance Evaluation*, 63(3):241–263.

[Gnutella, 2000] Gnutella (2000). http://rfc-gnutella.sourceforge.net.

[Goldschlag *et al.*, 1999] Goldschlag, D., Reed, M. et Syverson, P. (1999). Onion routing for anonymous and private internet connections. *Communications of the ACM*, 42:39–41.

[Goss et Deneubourg, 1991] Goss, S. et Deneubourg, J.-L. (1991). Harvesting by a group of robots. *In the 1st European Conference on Artificial Life*, ECAL'91, pages 195–204, Paris, France. MIT Press.

[Grasshopper, 1995] Grasshopper (1995). http://cordis.europa.eu/infowin/acts/analysys/products/thematic/agents/ch4/ch4.htm.

[Grassé, 1959] Grassé, P.-P. (1959). La reconstruction du nid et les coordinations interindividuelles chez bellicositermes natalensis et cubitermes sp. la théorie de la stigmergie : Essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, 6(1):41–84.

[Gray *et al.*, 2002] Gray, R. S., Cybenko, G., Kotz, D., Peterson, R. A. et Rus, D. (2002). D'agents : applications and performance of a mobile-agent system. *Software – Practice & Experience*, 32(6):543–573.

[GWCSR, 2010] GWCSR (2010). http://gcachescan.jonatkins.com/.

[Haeberlen *et al.*, 2005] Haeberlen, A., Mislove, A. et Druschel, P. (2005). Glacier : Highly durable, decentralized storage despite massive correlated failures. *In the 2nd Symposium on Networked Systems Design and Implementation*, NSDI '05, pages 143–158, Boston, MA, USA. USENIX.

[Harrouet, 2000] Harrouet, F. (2000). *oRis : s'immerger par le langage pour le prototypage d'univers virtuels à base d'entités autonomes*. Thèse de doctorat, Université de Bretagne Occidentale, Brest, France.

[Hayes et Dormiani-Tabatabaei, 2002] Hayes, A. T. et Dormiani-Tabatabaei, P. (2002). Self-Organized Flocking with Agent Failure : Off-Line Optimization and Demonstration with Real Robots. *In the 2002 IEEE International Conference on Robotics and Automation*, ICRA 2002, pages 3900–3905, Washington, DC, USA. IEEE.

[Horling et Lesser, 2005] Horling, B. et Lesser, V. (2005). A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*, 19(4):281–316.

[IDC, 2009] IDC (2009). La quantité d'informations numériques explose. http://www.ginjfo.com/Publics/Actualites/La-quantite-d\OT1\textquoterightinformations-numeriques-explose-3952.html.

[IWS, 2010] IWS (2010). Internet world stats : Usage and population statistics. http://www.internetworldstats.com/stats.htm.

[JavAct, 2004] JavAct (2004). http://www.javact.org/.

[Jelasity *et al.*, 2009] Jelasity, M., Montresor, A. et Babaoglu, Ö. (2009). T-man : Gossip-based fast overlay topology construction. *Computer Networks*, 53(13):2321–2339.

[Jennings, 2000] Jennings, N. R. (2000). On agent-based software engineering. *Artificial Intelligence*, 117(2):277–296.

[Johansen, 1998] Johansen, D. (1998). Mobile agent applicability. *Personal and Ubiquitous Computing*, 2(2):57–67.

[Johansen, 2004] Johansen, D. (2004). Mobile agents : Right concept, wrong approach. *In the 5th IEEE International Conference on Mobile Data Management*, MDM 2004, pages 300–301. IEEE Computer Society.

[Johansen *et al.*, 1995] Johansen, D., van Renesse, R. et Schneider, F. B. (1995). An introduction to the tacoma distributed system. Rapport technique, Departement of Computer Science, University of Tromsø.

[KaZaA, 2001] KaZaA (2001). http://www.kazaa.com/fr/index.htm.

[Kermarrec *et al.*, 2003] Kermarrec, A.-M., Massoulié, L. et Ganesh, A. J. (2003). Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(3):248–258.

[Kleinberg, 1999] Kleinberg, J. M. (1999). The small-world phenomenon : An algorithmic perspective. Rapport technique 99-1776, Department of Computer Science, Cornell University.

[Kleinberg, 2000] Kleinberg, J. M. (2000). Navigation in a small world. *Nature*, 406(6798).

[Kubiatowicz *et al.*, 2000] Kubiatowicz, J. D., Bindel, D., Chen, Y., Czerwinski, S. E., Eaton, P. R., Geels, D., Gummadi, R., Rhea, S. C., Weatherspoon, H., Weimer, W., Wells, C. et Zhao, B. Y. (2000). Oceanstore : An architecture for global-scale persistent storage. *In the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '00, pages 190–201, Cambridge, MA, USA.

[Kulbak et Bickson, 2005] Kulbak, Y. et Bickson, D. (2005). The emule protocol specification. Rapport technique, DANSS Lab, School of Computer Science and Engineering The Hebrew University of Jerusalem.

[Lange *et al.*, 1997] Lange, D. B., Oshima, M., Karjoth, G. et Kosaka, K. (1997). Aglets : Programming mobile agents in java. *Worldwide Computing and Its Applications*, 1274/1997: 253–266.

[Li et Yeung, 2003] Li, S.-Y. R. et Yeung, R. W. (2003). Linear network coding. *IEEE Transactions on Information Theory*, 49(2):371–381.

[Lindhé *et al.*, 2005] Lindhé, M., Ögren, P. et Johansson, K. H. (2005). Flocking with obstacle avoidance : A new distributed coordination algorithm based on voronoi partitions. *In the 2005 IEEE International Conference on Robotics and Automation*, ICRA 2005, pages 1785–1790, Barcelona, Spain. IEEE.

[Loureiro, 2001] Loureiro, S. (2001). *Mobile code protection*. Thèse de doctorat, Ecole National Supérieure des Télécommunications / Institut Eurécom.

[Lv *et al.*, 2002] Lv, Q., Cao, P., Cohen, E., Li, K. et Shenker, S. (2002). Search and replication in unstructured peer-to-peer networks. *In the 16th International Conference on Supercomputing*, ICS '02, pages 84–95, New York, NY, USA. ACM.

[MacCormick *et al.*, 2009] MacCormick, J., Murphy, N., Ramasubramanian, V., Wieder, U., Yang, J. et Zhou, L. (2009). Kinesis : A new approach to replica placement in distributed storage systems. *Transactions on Storage*, 4(4):1–28.

[Maymounkov et Mazières, 2002] Maymounkov, P. et Mazières, D. (2002). Kademlia : A peer-to-peer information system based on the xor metric. *In the 1st International Workshop on Peer-to-Peer Systems*, IPTPS '02, pages 53–65, Cambridge, MA, USA. Springer-Verlag.

[Milgram, 1967] Milgram, S. (1967). The small world problem. *Psychology Today*, 2:60–67.

[Milojicic *et al.*, 1998] Milojicic, D. S., Breugst, M., Busse, I., Campbell, J., Covaci, S., Friedman, B., Kosaka, K., Lange, D. B., Ono, K., Oshima, M., Tham, C., Virdhagriswaran, S. et White, J. (1998). MASIF : The OMG mobile agent system interoperability facility. *In the 2nd International Workshop on Mobile Agents*, MA'98, pages 50–67, Stuttgart, Germany.

[Monmarché, 2000] Monmarché, N. (2000). *Algorithmes de fourmis artificielles : applications à la classification et à l'optimisation*. Thèse de doctorat, Université de Tours.

[Monmarché *et al.*, 2002] Monmarché, N., Guinot, C. et Venturini, G. (2002). Fouille visuelle et classification de données par nuage d'insectes volants. *RSTI-RIA-ECA : Méthodes d'optimisation pour l'extraction de connaissances et l'apprentissage*, 6:729–752.

[Montresor *et al.*, 2003] Montresor, A., Meling, H. et Babaoglu, O. (2003). Messor : load-balancing through a swarm of autonomous agents. *In the 1st International Workshop on Agents and Peer-to-Peer Computing*, AP2PC'02, pages 125–137, Bologna, Italy. Springer-Verlag.

[Napster, 1999] Napster (1999). http://www.napster.com.

[Necula et Lee, 1996] Necula, G. C. et Lee, P. (1996). Safe kernel extensions without run-time checking. *In the 2nd Symposium on Operating Systems Design and Implementation*, OSDI '96, pages 229–243, Seattle, WA, USA.

[Necula et Lee, 1998] Necula, G. C. et Lee, P. (1998). Safe, untrusted agents using proof-carrying code. *Mobile Agents and Security*, 1419/1998:61–91.

[Olfati-Saber, 2006] Olfati-Saber, R. (2006). Flocking for multi-agent dynamic systems : Algorithms and theory. *IEEE Transactions on Automatic Control*, 51(3):401–420.

[Outtagarts, 2009] Outtagarts, A. (2009). Mobile agent-based applications : a survey. *International Journal of Computer Science and Network Security*, 9(11):331–339.

[Özalp Babaoglu et Montresor, 2002] Özalp Babaoglu, H. M. et Montresor, A. (2002). Anthill : A framework for the development of agent-based peer-to-peer systems. *In the 22th International Conference on Distributed Computing Systems*, ICDCS '02, pages 15–22, Vienna, Austria. IEEE.

[Peine et Stolpmann, 1997] Peine, H. et Stolpmann, T. (1997). The architecture of the ara platform for mobile agents. *In the 1st International Workshop on Mobile Agents*, MA '97, pages 50–61, Berlin, Germany. Springer-Verlag.

[PIAX, 2009] PIAX (2009). http://www.piax.org/en/.

[Plank, 1997] Plank, J. S. (1997). A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software – Practice & Experience*, 27(9):995–1012.

[Pommier et Bourdon, 2008] Pommier, H. et Bourdon, F. (2008). Data mobility in peer-to-peer system to improve robustness. *In the 7th International Workshop on Agents and Peer-to-Peer Computing*, AP2PC 2008, Estoril, Portugal.

[Pommier et Bourdon, 2009] Pommier, H. et Bourdon, F. (2009). Agents mobiles et réseaux pair-à-pair : Vers une gestion sécurisée de l'information répartie. *Revue d'Intelligence Artificielle*, 23(5-6):697–718.

[Pommier *et al.*, 2010a] Pommier, H., Romito, B. et Bourdon, F. (2010a). Bio-inspired data placement in peer-to-peer networks. *In the 6th International Conference on Web Information Systems and Technologies*, WEBIST 2010, Valencia, Spain. Springer.

[Pommier *et al.*, 2010b] Pommier, H., Romito, B. et Bourdon, F. (2010b). Placement de l'information bio-inspiré dans les réseaux pair-à-pair. *In 11èmes Journées Doctorales en Informatique et Réseaux*, JDIR 2010, Sophia-Antipolis, France.

[Proctor et Winter, 1998] Proctor, G. et Winter, C. (1998). Information flocking : Data visualisation in virtual worlds using emergent behaviours. *Virtual Worlds*, 1434/1998:168–176.

[Qiao *et al.*, 2005] Qiao, L., Wei, C. et Zheng, Z. (2005). On the impact of replica placement to the reliability of distributed brick storage systems. *In the 25th International Conference on Distributed Computing Systems*, ICDCS '05, pages 187–196, Columbus, OH, USA. IEEE.

[Ratnasamy *et al.*, 2001] Ratnasamy, S., Francis, P., Handley, M., Karp, R. et Schenker, S. (2001). A scalable content-addressable network. *In the annual conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '01, pages 161–172, San Diego, CA, USA. ACM.

[Reed et Solomon, 1960] Reed, I. S. et Solomon, G. (1960). Polynomial Codes Over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304.

[Reynolds, 1987] Reynolds, C. W. (1987). Flocks, herds and schools : A distributed behavioral model. *In the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH'87, pages 25–34, Anaheim, CA, USA. ACM.

[Rhea *et al.*, 2003] Rhea, S. C., Eaton, P. R., Geels, D., Weatherspoon, H., Zhao, B. Y. et Kubiatowicz, J. D. (2003). Pond : the oceanstore prototype. *In the 2nd USENIX conference on File And Storage Technologies*, FAST '03, pages 1–14, San Francisco, CA, USA. USENIX.

[Rodrigues et Liskov, 2005] Rodrigues, R. et Liskov, B. (2005). High availability in dhts : Erasure coding vs. replication. *In the 4th International Workshop on Peer-to-Peer Systems*, IPTPS '05, pages 226–239.

[Romito, 2009] Romito, B. (2009). Fléxibilité et adaptabilité des protocoles de sécurité dans les architectures P2P : une approche agents. Mémoire de Master Recherche, Université de Caen Basse-Normandie, Caen, France.

[Ronasi *et al.*, 2007] Ronasi, K., Firooz, M. H., Pakravan, M. R. et Avanaki, A. N. (2007). An enhanced random-walk method for content locating in p2p networks. *In the 27th International Conference on Distributed Computing Systems Workshops*, ICDCSW '07, pages 21–24, Toronto, Canada. IEEE Computer Society.

[Rowstron et Druschel, 2001a] Rowstron, A. et Druschel, P. (2001a). Pastry : Scalable, decentralized object location and routing for large-scale peer-to-peer systems. *In the IFIP/ACM International Conference on Distributed Systems Platforms*, Middleware 2001, pages 329–350, Heidelberg, Germany. ACM.

[Rowstron et Druschel, 2001b] Rowstron, A. et Druschel, P. (2001b). Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. *In the 18th ACM Symposium on Operating Systems Principles*, SOSP'01, pages 188–201, Chateau Lake Louise, Banff, Canada. ACM.

[Russell et Norvig, 2009] Russell, S. et Norvig, P. (2009). *Artificial Intelligence : A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, USA, 3ème édition.

[Sahai et Morin, 1998] Sahai, A. et Morin, C. (1998). Mobile agents for enabling mobile user aware applications. *In the 2nd International Conference on Autonomous Agents*, pages 205–211, Minneapolis, MN, USA.

[Seti@home, 1997] Seti@home (1997). http://setiathome.ssl.berkeley.edu/.

[Shamir, 1979] Shamir, A. (1979). How to share a secret. *Commun. ACM*, 22(11):612–613.

[Simonin, 2001] Simonin, O. (2001). *Le modèle satisfaction-altruisme : coopération et résolution de conflits entre agents situés réactifs, application à la robotique*. Thèse de doctorat, Université Montpellier II.

[Songsiri, 2006] Songsiri, S. (2006). Mtrust : A reputation-based trust model for a mobile agent system. *Autonomic and Trusted Computing*, 4158/2006:374–385.

[SPRINGS, 2008] SPRINGS (2008). http://sid.cps.unizar.es/SPRINGS/.

[Steels, 1989] Steels, L. (1989). Cooperation between distributed agents through self-organisation. *In the 1st European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, MAAMAW'89, pages 175–196, Amsterdam, Netherlands. Elsevier Science B.V.

[Stoica *et al.*, 2003] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F. et Balakrishnan, H. (2003). Chord : a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32.

[TACOMA, 1995] TACOMA (1995). http://www.tacoma.cs.uit.no/.

[Tanner *et al.*, 2003a] Tanner, H. G., Jadbabaie, A. et Pappas, G. J. (2003a). Stable flocking of mobile agents, part i : Fixed topology. *In the 42nd IEEE Conference on Decision and Control*, CDC 2003, pages 2010–2015, Maui, Hi, USA. IEEE.

[Tanner *et al.*, 2003b] Tanner, H. G., Jadbabaie, A. et Pappas, G. J. (2003b). Stable flocking of mobile agents, part ii : Dynamic topology. *In the 42nd IEEE Conference on Decision and Control*, CDC 2003, pages 2016–2021, Maui, Hi, USA. IEEE.

[TILAB, 2000] TILAB (2000). Jade. http://http://jade.tilab.com/.

[Tisseau, 2001] Tisseau, J. (2001). Virtual reality — in virtuo autonomy —. Habilitation à diriger des recherches, Université de Rennes 1.

[Trillo *et al.*, 2007] Trillo, R., Ilarri, S. et Mena, E. (2007). Comparison and performance evaluation of mobile agent platforms. *In the 3rd International Conference on Autonomic and Autonomous Systems*, ICAS '07, pages 41–46, Athens, Greece. IEEE.

[Tryllian, 2005] Tryllian (2005). http://www.tryllian.org.

[van Renesse, 2004] van Renesse, R. (2004). Efficient reliable internet storage. *In the 1st Workshop on Dependable Distributed Data Management*, WDDDM '04, Florianopolis, Brazil.

[van 't Noordende *et al.*, 2009] van 't Noordende, G., Overeinder, B. J., Timmer, R. J., Brazier, F. M. T. et Tanenbaum, A. S. (2009). Constructing secure mobile agent systems using the agent operating system. *International Journal of Intelligent Information and Database Systems*, 3(4):363–381.

[Voulgaris, 2006] Voulgaris, S. (2006). *Epidemic-based Self-Organization in Peer-to-Peer Systems*. Thèse de doctorat, VU University, Amsterdam, Netherlands.

[Voulgaris *et al.*, 2005] Voulgaris, S., Gavidia, D. et van Steen, M. (2005). Cyclon : Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2):197–217.

[Voulgaris *et al.*, 2003] Voulgaris, S., Jelasity, M. et van Steen, M. (2003). A robust and scalable peer-to-peer gossiping protocol. *In the 2nd International Workshop on Agents and Peer-to-Peer Computing*, AP2PC 03, Melbourne, Australia.

[Voulgaris et van Steen, 2005] Voulgaris, S. et van Steen, M. (2005). Epidemic-style management of semantic overlays for content-based searching. *In the 11th International Euro-Par Conference*, Euro-Par 2005, pages 1143–1152, Lisbon, Portugal. Springer.

[Voyager, 1997] Voyager (1997). http://recursionsw.com/index.html.

[Wagner, 2003] Wagner, L. D. (2003). Byzantine agreements in secure communication. *In the 5th Operations Research Conference*, ASOR 2003.

[Weatherspoon et Kubiatowicz, 2002] Weatherspoon, H. et Kubiatowicz, J. (2002). Erasure coding vs. replication : A quantitative comparison. *In the 1st International Workshop on Peer-to-Peer Systems*, IPTPS '02, pages 328–338, Cambridge, MA, USA. Springer.

[White, 1994] White, J. E. (1994). Telescript technology : The foundation for the electronic marketplace. Rapport technique, General Magic, Inc.

[Wooldridge, 2002] Wooldridge, M. J. (2002). *An introduction to MultiAgent systems*. John Wiley & Sons, New-York, NY, USA, 1st édition.

[Wylie *et al.*, 2000] Wylie, J. J., Bigrigg, M. W., Strunk, J. D., Ganger, G. R., Kiliççöte, H. et Khosla, P. K. (2000). Survivable information storage systems. *IEEE Computer*, 33(8):61–68.

[Yang et Garcia-Molina, 2002] Yang, B. et Garcia-Molina, H. (2002). Improving search in peer-to-peer networks. *In the 22th International Conference on Distributed Computing Systems*, ICDCS '02, pages 5–14, Vienna, Austria. IEEE.

[Zeinalipour-Yazti *et al.*, 2004] Zeinalipour-Yazti, D., Kalogeraki, V. et Gunopulos, D. (2004). Information retrieval techniques for peer-to-peer networks. *Computing in Science and Engineering*, 6:20–26.

[Zhao *et al.*, 2004] Zhao, B. Y., Huang, L., Stribling, J., Rhea, S. C., Joseph, A. D. et Kubiatowicz, J. D. (2004). Tapestry : A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53.

[Zhao *et al.*, 2001] Zhao, B. Y., Kubiatowicz, J. D. et Joseph, A. D. (2001). Tapestry : An infrastructure for fault-tolerant wide-area location and routing. Rapport technique UCB/CSD-01-1141, International Computer Science Division, UC Berkeley, Berkeley, CA, USA.

[Zuba, 2004] Zuba, D. (2004). Apports d'une approche multi-agents pour un stockage de fichiers sécurisés. Mémoire de Master Recherche, Université de Caen Basse-Normandie, Caen, France.

## Résumé

Cette thèse aborde la problématique de la gestion de l'information dans les réseaux pair-à-pair. L'objectif de notre approche est de fournir des mécanismes de placement et de recherche de l'information dynamiques, intelligents, robustes, et totalement décentralisés.

Nous proposons une architecture en couches composée d'un réseau pair-à-pair responsable de la décentralisation du service, d'applications héritées de la théorie de l'information pour assurer la disponibilité et la sûreté des données, et d'agents mobiles transportant l'information. Nous proposons un algorithme reposant sur le déplacement en nuée des oiseaux pour placer l'information dans un réseau. Nous associons à chaque déplacement d'agent un dépôt de phéromones permettant une répartition de charge efficace, et l'établissement d'une mesure de confiance envers les pairs du réseau. Nous enrichissons ce modèle en proposant deux agents de recherche de données. Le premier agent suit une marche aléatoire également apellée chaîne de Markov. Le second agent se déplace aléatoirement et utilise ses capacités d'interaction avec l'environnement en déposant des phéromones pour marquer les pairs déjà parcourus. Enfin, nous présentons une application déployée sur un réseau physique permettant l'observation du comportement de nos agents et la validation de nos algorithmes.



**Mots-clés:** Réseaux pair-à-pair, placement de l'information, recherche de l'information, agents mobiles, systèmes multi-agents bio-inspirés.


---


**Discipline :** Informatique

**Laboratoire :** Groupe de Recherche en Informatique, Image, Automatique et Instrumentation de Caen - UMR 6072, Université de Caen/Basse-Normandie, France