Université de Reims Champagne-Ardenne
École Doctorale Sciences Technologies et Santé
Université de Cheikh Anta DIOP de DAKAR
École Doctorale Mathématiques et Informatique

Thèse réalisée en co-tutelle

présentée par **Thierno Ahmadou DIALLO**

pour l'obtention du grade de

Docteur de l'Université de Reims Champagne-Ardenne

DISCIPLINE : Informatique

sur le titre:
GRAPP&S, une solution totalement répartie pour le stockage de données & Services

Soutenue publiquement le 30 Novembre 2016 devant le jury composé de :

Président de jury	Mickaël KRAJECKI	Prof. à l'URCA
Rapporteurs	Hervé GUYENNET	Prof. à l'Université de Franche Comté, France
	Yahya SLIMANI	Prof. à l'Université de la Manouba,Tunisie
Examinateurs	Hubert NAACKE	MCF à l'Université Paris 6
	Florence LEVÉ	MCF à l'Université de Picardie Jule Vernes, Amiens
Directeur de thèse	Olivier FLAUZAC	Prof. à l'URCA
Co-directeur de thèse	Samba N'DIAYE	Maître de Conférences HDR à l' Université de Dakar
Co-encadrant de thèse	Luiz Angelo STEFFENEL	Maître de Conférences à l'URCA

“À la mémoire de mon père”
“À ma mère”
“À mes frères et sœurs”
“À ma famille mon épouse et mon fils Abdourahmane”

Remerciements

Je tiens tout d'abord à remercier chaleureusement mon directeur de thèse Olivier Flauzac d'avoir voulu me proposer ce sujet de thèse très intéressant. Merci infiniment de m'avoir supporté et fait confiance pendant ces années de thèse. Merci aussi pour les orientations, et la rigueur scientifique et d'avoir accepté le principe de la thèse en alternance entre le Sénégal et la France.

Mes remerciements vont à l'endroit de Samba N'DIAYE, Maitre de conférence HDR à l'université Cheikh Anta DIOP de DAKAR qui a bien voulu être le directeur de cette thèse pour la partie Sénégalaise de la cotutelle. Merci de guider mes premiers pas dans la recherche en m'encadrant lors de mon mémoire de MASTER2 Informatique à l'Université Cheikh Anta DIOP de DAKAR.

Je remercie très sincèrement Luiz Angelo STEFFENEL, mon co-directeur de thèse, pour sa disponibilité à tout moment pendant ces années de thèse. Les pistes qu'il m'a donné m'ont permis d'avancer dans mes recherches. Je te remercie de m'avoir grandement assister à la rédaction de ce manuscrit. Merci d'avoir procéder a de nombreuses relectures et corrections pour la perfection de ce manuscrite.

Merci également à mes deux rapporteurs *Hervé Guyennet* et *Yahya Slimani* respectivement professeur à l'Université de Besançon et professeur à l'Université de la Manouba Tinusie, d'avoir accepté de rapporter ma thèse, et qui malgré les imperfections qu'elle présentait, ne m'ont fait que des remarques constructives.

Je remercie le Professeur Mickael Krajecki de l'Université de Reims Champagne-Ardenne pour l'honneur qu'il me fait d'accepter de présider le jury de ma thèse. Également à ses collègues Hubert Naacké et Florence Levé d'avoir accepté de participer a ce même jury de cette thèse.

La recherche est aussi un travail d'équipe et je tiens tout d'abord à remercier le professeur *Ibrahima Niang* avec qui j'ai eu de nombreuses discussions, me permettant d'avancer dans mes travaux. Merci professeur *Niang* pour l'aide financière que vous m'aviez accordé au début de ma thèse. Merci également à mon camarade de bureau *Dr. Mandicou BA* qui a eu à me supporter durant ces dernières années, en plus de ses nombreux conseils surtout sur la dernière ligne droite de ma thèse. Sans oublier mes *collègues du département informatique de l'université Assane Seck de Ziguinchor*.

Le cadre de travail a été un facteur important du bon déroulement de ce travail thèse pendant mon séjour à REIMS. Pour cela je remercie le *professeur Mickael Krajecki*, Directeur du laboratoire *CReSTIC* et tous les membres de l'équipe *CReSTIC/SysCom* de m'avoir accueillie dans le laboratoire *CReSTIC* et de leur gentillesses.

Je remercie le Service de Coopération et d'action culturelle de l'ambassade de France au Sénégal pour son soutien financier qui m'a permis d'effectuer mes séjours de thèse en alternance entre le CReSTIC/SysCom de REIMS et EDMI de l'Université Cheikh Anta DIOP de Dakar. Je remercie également le CEA-MITIC, qui qui ma permis de voyager pour aller soutenir ma thése dans les meilleurs conditions.

Je tiens également à remercier tous ceux qui ont cru en moi. Plus particulièrement mes parents qui m'ont toujours soutenu dans mes études, mais aussi mes frères et sœurs. Je tiens enfin à remercier ma femme *Fatou Diedhiou*, qui a du supporter mes voyages d'etudes à Reims durant la thèse.

Table des matières

Remerciements **i**

Table des matières **iii**

Liste des figures **vii**

Liste des algorithmes **viii**

Liste des tableaux **ix**

Introduction Générale **1**

1 Modélisation des systèmes répartis **7**
1.1 Définitions et Modélisation 7
1.1.1 Graphe et système de communication 7
1.1.2 Topologies particulières 9
1.1.3 Topologie physique et topologie logique 13
1.2 Modèle d'exécution 14
1.3 Communications dans les systèmes répartis 14
1.3.1 Modèles de communication 15
1.3.2 Protocoles de communication 16
1.4 Synchronisme dans les systèmes répartis 17
1.4.1 Le modèle asynchrone 17
1.4.2 Le modèle synchrone 17
1.5 Algorithmes classiques 18
1.5.1 Algorithmes à vagues 18
1.5.2 Construction d'arbres 19
1.5.3 Algorithmes d'élection 19
1.5.4 Algorithmes de snapshots, détection de propriétés 20
1.6 Retour sur les algorithmes de routage 21
1.6.1 Spécification du problème du routage 21
1.6.2 Evaluation des algorithmes de routage 21
1.6.3 Classification des algorithmes de routage 22
1.6.4 Calcul de la meilleure route 23
1.6.5 Routages compacts 23

1.7 Conclusion 25

2 Les applications réparties **27**
2.1 Introduction 27
2.2 Gestion des communication 28
2.2.1 Le modèle *push* 28
2.2.2 Le modèle *pull* 28
2.2.3 Le modèle *proxy* 29
2.3 Modèles des applications réparties 29
2.3.1 Le modèle Client / Serveur 29
2.3.2 Modèles P2P 30
2.4 La gestion des données dans les applications réparties 33
2.4.1 Solutions de stockage orienté fichiers 33
2.4.2 Le stockage structuré : les bases de données 35
2.4.3 Stockage orienté document 37
2.5 Données et systèmes pair-à-pair(P2P) 39
2.5.1 Napster 39
2.5.2 Bittorrent 40
2.5.3 FastTrack et KaZaA 42
2.5.4 Gnutella 43
2.5.5 Freenet 44
2.6 Réseaux structurés décentralisés 46
2.6.1 Chord 47
2.6.2 Pastry 48
2.6.3 Kademlia 49
2.6.4 CAN 51
2.6.5 Extensions hiérarchiques 53
2.7 Conclusion 54

3 Architecture de *GRAPP&S* **55**
3.1 Introduction 56
3.2 Modèle et environnement 57
3.2.1 Quelques définitions 57
3.2.2 Communication et les réseaux overlay 57
3.3 Éléments de l'Architecture GRAPP&S 58
3.3.1 Notion de communauté 58
3.3.2 Data Manager (DM) 59
3.3.3 Structure logique d'un nœud DM 59
3.3.4 Gestion des données par les DM 61
3.3.5 Identification des données 62
3.4 Ressources manager *RM* 62
3.4.1 Structure logique nœud RM 63
3.4.2 L'indexation dans les *Ressources Manager* 64
3.4.3 Élection d'un *Ressource manager* 66
3.5 Communicator (*C*) 68

3.6 Adressage des nœuds de GRAPP&S . . . 69
3.6.1 Adressage à plat . . . 69
3.6.2 Adressage hiérarchique . . . 70
3.6.3 Système d'adressage adopté dans GRAPP&S . . . 71
3.7 Conclusion . . . 72

4 Routage et transfert d'informations dans GRAPP&S 73
4.1 Motivations et objectifs . . . 74
4.2 Primitives de GRAPP&S . . . 75
4.2.1 Retour sur la structuration de GRAPP&S . . . 75
4.2.2 Routage élémentaire . . . 76
4.2.3 Recherche de ressources . . . 77
4.2.4 Rappatriement des ressources . . . 83
4.3 Etude des performances des primitives . . . 86
4.3.1 Coût des communications . . . 86
4.3.2 Performance de la recherche . . . 88
4.3.3 Performance du Rapatriement des données . . . 91
4.4 Conclusion . . . 92

5 Perspective d'utilisation du framework GRAPP&S dans le domaine du e-learning 93
5.1 Le *E-learning* dans le cadre des pays en développement . . . 94
5.2 Les outils existants . . . 94
5.2.1 Solutions E-learning basées sur le Cloud . . . 94
5.2.2 Bureaux virtuels . . . 96
5.2.3 Groupware basé sur les systèmes Pair-à-Pair(P2P) . . . 97
5.3 Conception d'une application de *E-learning* basée sur GRAPP&S . . . 100
5.3.1 Partage de données . . . 102
5.3.2 Mise en œuvre de tableau blanc dans GRAPP&S . . . 103
5.3.3 Mise en œuvre de la vidéoconférence dans GRAPP&S . . . 103
5.4 Conclusion . . . 104

Conclusion Générale 105

Bibliographie 109

Table des figures

1.1 Graphe modélisant un systéme distribué 8
1.2 une arête *e=(u, v)* 9
1.3 structure en chaîne 10
1.4 structure en anneau 10
1.5 structure en arbre 11
1.6 Une grille 3 x 4 12
1.7 Un graphe complet à 5 sommets. 12
1.8 Un graphe complet à 5 sommets et un anneau logique 13
1.9 Un graphe complet à 5 sommets et un arbre logique 13
1.10 Principe de la fenêtre glissante 16
1.11 Principe de la communication par timer 16

2.1 Modéle de données Stockage de documents 37
2.2 Organisation de Napster 40
2.3 Organisation de Bittorrent 41
2.4 La recherche dans FastTrack 42
2.5 Architecture de Gnutella v0.4 44
2.6 Architecture de Gnutella0.6 45
2.7 Recherche de donnée dans Freenet inspiré de [CSWH01] 45
2.8 Recherche sequentielle dans Chord 47
2.9 Recherche avec les tables de voisinage de chaque Nœud 48
2.10 Table routage des Nœuds Pastry extrait dans [RD01a] 49
2.11 Acheminement des messages dans Pastry 50
2.12 Arbre binaire Kademlia 51
2.13 Organisation de l'overlay CAN 52
2.14 Insertion d'un nœud *i* au CAN 53

3.1 Organisation des nœuds de GRAPP&S sous forme de communautés 59
3.2 Architecture d'un nœud DM 60
3.3 Architecture d'un nœud RM 63
3.4 Un exemple de structure de numerotation à plat 69
3.5 Structure d'adresse hiérarchique 70

4.1 Recherche locale dans GRAPP&S 79
4.2 Recherche dans le réseau GRAPP&S 81

4.3 Rapartriement dans le cas d'un réseau local dans GRAPP&S 84
4.4 Rapatriement de données par une connexion routée 85

5.1 Illustration de la distribution des nœuds GRAPP&S sur un overlay Pastry . . . 107

Liste des Algorithmes

1 Methode de routage de base 77
2 Traitement des messages selon les rôles des nœuds 80
3 Recherche locale 82
4 Vérification de l'index RM 82
5 Diffusion de message aux RMs 83

Liste des tableaux

3.1 Table de routage d'un nœud dans un adressage à plat. 69
3.2 Table de routage du nœud 01.2 71

Introduction Générale

Contexte

L'augmentation du volume des données numériques est un fait avéré, à tel point que l'expression « déluge de données » revient fréquemment dans divers domaines scientifiques et dans le monde de l'entreprise. Cette augmentation des données est proportionnelle à l'accroissement continuelle du nombre de type de données(audio, vidéo, image, etc.) et des méta-données, etc.

Cette volumétrie des données est favorisée par l'essor des réseaux sociaux comme Facebook, eBay, etc (avec plusieurs de milliers de nouveaux fichiers qui sont archivés sur les serveurs de Facebook), les millions d'internautes, et l'état de connexion Internet permanente qui accentue les exigences des utilisateurs en terme de fiabilité, de rapidité et de volume de stockage[BA08]. La quantité de données produite est estimée selon l'institut IDC[idc14] a « 1,8 zettaoctets de données en 2011, soit 1,8 millier de milliards de gigaoctets ! » En 2014 l'IDC évalue le volume de données à 7 zettaoctets soit une progression de 47% chaque année depuis 2011. D'autres analyses prédisent que d'ici 2020, l'univers « numérique » sera 44 fois plus grand qu'en 2009. L'origine des ces données ne n'est plus uniquement réalisée par des humains, mais aussi des équipements tels que les capteurs, les caméras de surveillance, des sondes météo, des cartes bancaires, des télescopes constituent des mines d'informations(données).

Aujourd'hui, la capacité des outils de stockage a largement dépassé le téraoctet et il n'est plus rare pour les entreprises et les organismes de recherche de manipuler des volumes supérieurs au pétaoctet(Po, soit 1015 octets). Bonnel dans sa thèse [BA08], évalue le stockage des données produit par le Grand Collisionneur Hadronique(LHC) qui s'élève à 15 pétaoctets par an sur une pile de CD de 21 kilomètres de haut. En plus les données produites par l'application LHC devront être accessible pendant 15 ans [BA08]. Donc Les problèmes rencontrés avec la volumétrie des données se posent en termes de volume de stockage et de rapidité d'accès.

L'utilisation des supports USB, et disques durs permet de stocker au mieux quelques téraoctets de données. En plus du point de vue de la rapidité d'accès, les disques présentent une latence d'accès faible. Malgré l'évolution de la capacité de stockage des disques, le problème n'est donc pas tant de pouvoir loger les données que de pouvoir les accéder rapidement. Or, d'après [SC12] la vitesse de l'interface d'accès aux disques n' augmente que de 10% à 15% alors que la capacité de stockage passe de 70 à 80%. Donc la capacité par disque unitaire augmente donc plus rapidement que leur vitesse. Avec les applications citées précédemment, il est difficile de centraliser le stockage de grande quantité d'information.

C'est pour cette raison que les chercheurs et développeurs se sont tournés depuis longtemps vers le développement de solutions de stockage distribuée, afin de contourner ces limitations. La répartition des données entre différentes machines offre plusieurs avantages. Elle permet de

gérer de grandes quantités de données, elle est aussi plus résistante aux pannes, pour autant qu'une forme de redondance soit prise en compte. Si une machine tombe en panne, d'autres machines peuvent encore rester accessibles.

Plusieurs solutions de stockage distribué sont proposées pour répondre à l'augmentation des besoins de stockage, tout en offrant suffisamment de garanties pour assurer la consistance et la pérennité des données. Ainsi nous distinguons les données qui peuvent se présenter sous la forme d'un simple fichier, d'autres types de données qui ne sont pas des fichiers. Dans le cas ou les données sont des fichiers les solutions : SAN /NAS, les solutions Pair-à-Pairs, les Clouds constituent des choix de technologies qui offrent une grande capacité de stockage. Pour les données de type non fichiers les bases de données relationnelles, le NoSQL permettent de stocker de grosses quantités de données de l'ordre de plusieurs pétaoctets sur un réseau de plusieurs milliers de serveurs dédiés.

Les infrastructures SAN et NAS[GVM00] permettent de partager les données stockées sur les disques mis en commun. Les utilisateurs situés sur des serveurs distincts peuvent accéder en même temps et directement a une même donnée (fichiers) sur disques via les réseaux SAN et NAS mis en place. Les NAS et SAN proposent des méthodes robustes de stockage de grande quantité de données et facilitent le contrôle et l'accès aux données.

Le système de stockage en nuage(*cloud*)[Gro09] qui est en vogue, permet le stockage de grands volumes de données qui résulte d'abord de l' agrégation en une seule entité d'un ensemble de supports de stockage préexistants dispersés géographique sur différents sites. Ces sites abritent en général des technologies de traitement et de stockage de très grande taille. L'augmentation du volume de stockage de données n'est pas la seule motivation des *cloud*, il faut ajouter la dispersion géographiquement des données et l'augmentation du débit d'accès en To/s en cumulant des débits d'accès aux données sur disque.

Pour organiser les données dispersées sur différents sites, les bases de données distribuées de type relationnelle ou NoSQL ont été utilisés. Une base de données distribuée permet d'agréger plusieurs base de données logiquement liées et distribuées sur un réseau d'ordinateurs. Les bases de données NoSQL[Cat11a] par exemple Bigtable[CDG$^+$08] sont des systèmes de stockage distribués qui ont été conçus pour stocker de grosses quantités de données structurées(de l'ordre de plusieurs pétaoctets) sur un réseau de plusieurs milliers de serveurs dédiés.

L'utilisation de l'Internet comme moyen de stockage de données et l' augmentation du nombre d' utilisateurs ont poussé les chercheurs et développeurs à faire d'autre choix de stockage de grande quantité de données. Ces choix concernent les réseaux Pair-à-Pair(P2P) qui offrent des ressources potentiellement illimitées concernant le coût pour le stockage et l'échange décentralisé de données, la disponibilité et l' utilisation des ressources physiques. Dans les réseaux P2P les utilisateurs s'échangent des données sans passer par une entité centrale. Chaque utilisateur est à la fois client en demande de données et serveur en tant que fournisseur de données.

Contribution

La problématique traitée dans cette thèse porte sur le stockage et l'indexation unifiée de tous les formats de données tels que les fichiers, les flux de données, les requêtes sur les bases de données mais aussi l'accès à des services distants(des web services sur un serveur ou le cloud, ou des tâches de calcul dans un cluster HPC). Les données et services doivent être gérés de

façon transparente. De plus à notre problématique il faut ajouter la sécurité liée à la gestion des données et services.

Les solutions de stockages citées précédemment pour le stockage de grande quantité de données présentent des inconvénients liés à la vitesse et à la sécurité des données. En plus la pluspart de ces solutions ne permettent pas la gestion de données pouvant être représentés sous la forme de fichiers. Il est difficile pour ces solutions de manipuler des données qui ont d'autres formes de représentation comme les requêtes sur les bases de données, des flux de données(audio, vidéo) ou encore l'exécution de services.

Cette thèse s'intéresse à la gestion et le stockage unifié de tous les types de données et services. C'est dans cet objectif que nous présentons GRAPP&S(GRid APPlication & Services), une architecture multi-échelle pour l'agrégation de données et services de façon transparente. Ainsi que les mécanismes à mettre en place pour assurer la rapidité d'accès et la sécurité aux données et services. Elle s'intéresse également au caractère adaptatif du Framework GRAPP&S dans le domaine de l' enseignement pour le partage décentralisé de tout type de ressources éducatives.

Nos contributions peuvent être regroupées en trois grands points :

1. proposer une architecture multi-échelle pour la gestion des données et services que nous nommons GRAPP&S(Grid APPlication & Services). Ce Framework GRAPP&S gère de manière transparente les données de type fichier mais aussi les bases de données, les flux(audio, vidéo), les services Web et le calcul distribué grâce à l'utilisation de nœuds spécialisés « proxy data ». Ces « proxy data » permettent d'unifier l'accès à des ressources nombreuses et variées. GRAPP&S regroupe les données qui partagent une même propriété définie(même localisation, même autorité d'adminisatration, etc.) sous forme de communauté.
2. La deuxième contribution consiste à proposer un mécanisme de routage préfixé pour la recherche et l'accès aux données de GRAPP&S. En effet, si la recherche d'une information dans les middlewares Pair-à-Pair(P2P) varie selon l'architecture de ceux-ci, le transfert de données dépend généralement d'une connexion directe entre la source et l'intéressé. Le mécanisme de routage que nous proposons dans GRAPP&S permet d'effectuer le transfert des données par le chemin utilisé lors de la recherche, au cas où une connexion directe entre les nœuds n'est pas possible. Ce mécanisme contribue donc à relier des ressources très isolées mais aussi permet d'accroître la sécurité des réseaux, en offrant la possibilité de contrôler l'accès aux ressources.
3. Notre dernière contribution consiste à appliquer notre Framework GRAPP&S dans le domaine E-learning. En effet, GRAPP&S est une solution E-learning pour le partage décentralisé de tous les types de ressources grâce a son système d'indexation et une définition des types mime étendu des ressources. Il regroupe les différents référentiels(sources de données) locaux d'apprentissage de chacun des instituts(écoles, universités, les dépôts de laboratoires de recherche, etc.) sous forme « communauté ». Cela permettra un accès rapide à des ressources en raison de la proximité des consommateurs, contrairement à la plupart des solutions E-Learing basées sur le stockage en cloud ou les réseaux Pair-à-Pair lorsque les consommateurs sont pénalisés à cause de l'éloignement des ressources et une vitesse d'accès lente. Grâce à l'utilisation de communautés, nous pouvons intégrer tous les outils pour améliorer l'éducation dans une infrastructure unique, avec un coût

plus faible.

GRAPP&S à la possibilité de définir des politiques de sécurité pour la protection des ressources éducatives au sein d'une communauté, et les politiques d'accès entre les différentes communautés à travers la mise en place de *services level Agreements (SLAs)*. Pour ces raisons, GRAPP&S peut également être étendue à d'autres secteurs de l'école, par la création d'une communauté dans la partie administrative, séparé de la partie de l'éducation, avec des politiques d'accès et des règles de sécurité et de la protection des ressources.

Organisation du document

Ce manuscrit est organisé comme suit :

Le chapitre 1 introduit les systèmes distribués sous quelques uns de leurs aspects comme : la définition et la modélisation de ces systèmes distribués qui constituent le cadre général de nos travaux de recherche. Ensuite nous présentons quelques topologies régulières des systèmes distribués. Enfin nous présentons la rédaction d'algorithmes répartis, ainsi que différents algorithmes classiques, et notamment des algorithmes de routage.

Le chapitre 2 est consacré à l'état de l'art du document concernant les systèmes de stockage multi-échelle, comme les systèmes NAS/SAN, les SGBDs, les bases de données NoSQL, les systèmes cloud et les réseaux Pair-à-Pair(P2P), Pair-à-Pair hiérarchique. L'étude de chacun de ces systèmes se porte sur la structure de leurs topologies, leurs manières d'indexation de données et leurs mécanismes de recherche et de rapatriement de données. Par la suite nous allons faire une étude sur la généralité des techniques de routage comme : *le routage centralisé, le routage pair-à-pair, le routage hiérarchique, le routage préfixé.*

Le chapitre 3 présente la spécification de l'architecture GRAPP&S, les principaux éléments qui constituent l'architecture GRAPP&S et comment ces éléments s'interconnectent. Ces différents éléments sont essentiellement de trois types de nœuds : *Communicator, Ressource Manager et Data Manager*. Nous détaillerons dans ce chapitre le rôle de chaque nœud et comment ils peuvent être regroupés sous forme d' une entité autonome nommée *communauté*. Ainsi nous allons montrer dans ce chapitre le modèle d'adressage utilisé pour les nœuds de GRAPP&S qui sera nécessaire pour la recherche et l'accès aux informations dans notre système.

Le chapitre 4 détaille les principales opérations liées à la gestion, à la recherche et aux transferts d' informations dans GRAPP&S. La recherche et l'accès aux informations de GRAPP&S se fait par routage hiérarchique qui s'appuie sur les identités des différents nœuds de GRAPP&S pour définir les chemins par lesquels transitent les requêtes. Nous décrivons le principe de routage élémentaire utilisé dans GRAPP&S. Ensuite nous montrons comment construire les identités de ses nœuds. Nous décrivons dans ce chapitre le principe des différents algorithmes de recherche et de rapatriement de données puis nous donnons le formalisme de chacun de ces algorithmes. Àprés cela, nous évaluons différentes métriques de l'exécution des algorithmes de recherche ou de rapatriement de donnée. Ceci est réalisée en fonction des différentes stratégies

à adopter pour rétablir la communication lorsqu'un dispositif réseau empêchera le déroulement correct du routage dans l'arbre de GRAPP&S.

Le chapitre 5 est une perspective d'utilisation de GRAPP&S dans le domaine de l'enseignement E-learning. Il d'écrit d'abord les différentes solutions E-learning basées sur le *Cloud*, *les réseaux pair-à-pair*, les *bureaux virtuels* sous quelques uns de leurs aspects comme leur architecture, leur façon d'indexer les ressources et leur manière de recherche une information. Ensuite nous décrirons l'application E-learning basée sur GRAPP&S ainsi que les principales opérations dont la mise en œuvre est possible dans l'architecture GRAPP&S comme : l'interconnexion et le partage de ressources éducatives, la mise en œuvre de tableau blanc, de la vidéoconférence etc.

Enfin, nous clôturons nos travaux par la dernière partie du document avec des perspectives de développement de nos travaux.

Chapitre 1

Modélisation des systèmes répartis

Résumé. *Dans ce premier chapitre, nous allons donner la description du modèle théorique de structuration, d'échange, de calcul, et de synchronisation sur lequel nous nous appuyons pour concevoir les différentes solutions que nous allons présenter. Ce modèle ainsi défini nous permettra de formaliser les différents composants du système, ainsi que les différentes contraintes. Le modèle technique que nous utiliserons par la suite sera une extension du modèle théorique, modèle dans lequel seront intégrées les contraintes liées à la sécurité des systèmes, la hiérarchie des réseaux.*

Sommaire

1.1 Définitions et Modélisation 7
1.2 Modèle d'exécution 14
1.3 Communications dans les systèmes répartis 14
1.4 Synchronisme dans les systèmes répartis 17
1.5 Algorithmes classiques 18
1.6 Retour sur les algorithmes de routage 21
1.7 Conclusion 25

1.1 Définitions et Modélisation

Il existe de nombreuses définitions des systèmes distribués. Dans [Tel94] l'auteur définit un système distribué comme un *ensemble d'entités autonomes, ou unité de calcul, reliées par des canaux de communication.* Par ces canaux ils échangent des *messages* dans le but d'avoir des puissances de traitement ou de stockage, des disponibilités de services, un temps de réponse rapide etc. Ces entités peuvent être des processus ou des machines physiques (ordinateur par exemple).

Cette définition très large se doit d'être précisée et formalisée.

1.1.1 Graphe et système de communication

Définition 1.1. *Un* ***système distribué*** *est modélisé par un graphe $G = (V, E)$, les ensembles V et E correspondent respectivement aux sommets (ou nœuds) et aux arêtes ou arcs du graphe. Le graphe G peut être orienté ou non-orienté.*

Le graphe suivant 1.1 est un exemple de représentation d'un système distribué.

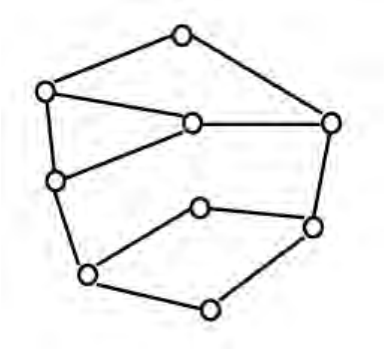


FIGURE 1.1 – Graphe modélisant un systéme distribué

Les nœuds du graphe représentent les unités de calcul du système, les arcs, ou arêtes du graphe représentent les liens de communication. Comme dans le cadre des graphes les liens de communication peuvent être orientés ou non orientés, fixant ainsi le « sens » des communications et des échanges.

Définition 1.2. *Un graphe G est dit* ***connexe*** *si pour deux sommets quelconques u et v du graphe, on peut trouver un chemin constitué d'arêtes consécutives qui relie u à v.*

Définition 1.3. *Un graphe G est dit* ***simple*** *si quels que soient u et v deux sommets, u est relié à v par au plus une arête.*

Définition 1.4. *Un graphe G est dit* ***sans boucle*** *si quel que soit u un sommet, il n'existe aucune arête (u, u).*

Les graphes que nous considèrerons dans la suite sont connexe, simple et sans boucles.

Définition 1.5. *Chaque* ***sommet*** *ou* ***nœud*** *est une entité autonome de calcul, de stockage, avec des moyens et des canaux de communications.*

Un sommet peut être, par exemple : un ordinateur, un équipement réseau (routeur, switch ...), un capteur, un objet communiquant ...

Les sommets peuvent ne posséder aucun identifiant, dans ce cas on parlera de *système anonyme*. Nous supposons que chaque sommet du graphe possède une identité unique. Cette hypothèse est tout à fait réaliste, dans un réseau classique, chaque interface de communication étant doté d'un identifiant physique unique : l'adresse MAC ; et chaque interface active est dotée d'un identifiant logique unique au sein du réseau dans lequel il se trouve : l'adresse IP.

L'activité d'un sommet, c'est-à-dire, celle qui correspond à l'exécution d'un algorithme, consiste en trois opérations élémentaires :

1. Envoyer des messages aux voisins `Send(message)` ;
2. Recevoir des messages `Receive(message)` ;
3. Effectuer des calculs locaux, permettant l'évaluation de toutes les opérations algorithmiques classiques `Compute()`.

Définition 1.6. *Une arête du graphe correspond à un* ***canal de communication*** *bidirectionnel entre deux entités de calcul voisines.*

Une arête $e= (u, v) \in E$ correspond alors à une paire $((u, i), (v, j))$ comme le montre la figure1.2.

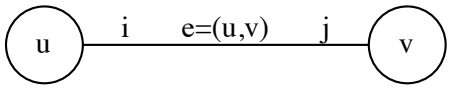


FIGURE 1.2 – une arête *e=(u, v)*

L'envoi d'un message de u vers v(resp. de v vers u) s'effectue alors en transmettant le message sur le port i(resp. sur le port j). Le sommet v(resp. u) reçoit le message en le récupérant sur le port j (resp. sur le port i). Nous supposons que les ports d'un même sommet ont des numéros distincts et qu'un sommet peut distinguer le port sur lequel il a reçu un message donné.

L'échange d'informations entre u et v peut se faire, comme nous le verrons dans la suite, selon deux principaux modèles : par lecture / écriture de mémoire partagée, ou par échange de messages explicites.

Définition 1.7. *Le* ***voisinage*** *d'un sommet u donné, est l'ensemble des sommets $v \in E$ auxquels ce dernier est connecté par une arête $(u, v) \in E$.*

Autrement dit, le voisinage correspond à l'ensemble des nœuds avec lesquels il est possible de communiquer directement.

Deux sommets u et v reliés par une arête dans le graphe G sont dit *adjacents*. L'ensemble des sommets adjacents à u définit aussi le voisinage de u ; il est noté par $N(u)$. Le degré de u, noté $deg(u)$, désigne le nombre de voisins de u.

1.1.2 Topologies particulières

Plusieurs types de graphes sont rencontrés dans les systèmes distribués. Le choix des graphes utilisés est fortement tributaire de leur caractère très hétérogène, leur besoin de passage à l'échelle, et de leur structuration.

Le choix d'une topologie particulière a un impact non négligeable sur les facilités de programmation et sur les performances des algorithmes et des programmes qui sont développés et implantés. Une topologie particulière permet d'exploiter des propriétés spécifiques comme l'orientation, la régularité du voisinage ... Ces propriétés simplifient grandement certaines étapes ou services liés à la conception d'algorithmes.

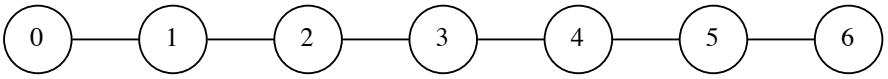


FIGURE 1.3 – structure en chaîne

Topologie en *Chaîne* On définit une chaîne de taille n comme un graphe connexe dont 2 nœuds sont de degré 1 et $n-2$ nœuds de degré 2. Une illustration d'une topologie en chaîne d'un système distribué est donnée dans la figure 1.3.

La topologie en chaîne est largement exploitée, on la retrouve notamment dans le cadre des architectures de type bus de communication, mais aussi des bus logiciels comme dans le cas de *Corba* [**?**, GGM00]

Elle permet des diffusions simples, ainsi qu'une grande simplicité des moyens utilisés pour le routage. Par contre, elle n'est pas tolérante aux fautes : une panne d'un nœud interne provoque la perte de la connexité du système.

Topologie en *Anneau* Un anneau est une chaîne reliée par ses extrémités. Il peut être orienté ou non. L'orientation peut se faire soit dans le cadre des liens de communications, soit de manière interne aux nœuds de communication en associant des identifiants de type `next` et `previous` aux deux voisins d'un nœud, c'est à dire aux ports de communication associés. Naturellement, dans ce cas le port `next` d'un nœud est relié au port `previous` de son voisin et inversement. Une illustration d'une topologie en chaîne d'un système distribué est donnée dans la figure 1.4.

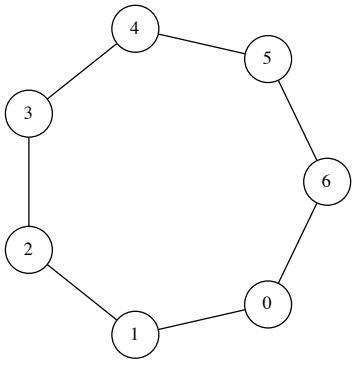


FIGURE 1.4 – structure en anneau

La topologie en anneau est classique, on la trouve aussi bien dans le cadre du *Token ring* REFERENCE, que dans les systèmes pair-à-pair comme Chord[SMK+01], Pastry[RD01a] ou

CONFIIT[FKS10].

La topologie en anneau permet une grande simplicité des opérations de routage de l'information, ainsi que de diffusion. Comme dans le cas de la chaîne, elle se montre non tolérante aux fautes, une seule panne d'un sommet transformant l'anneau en chaîne, deux pannes pouvant provoquer la perte de la propriété de connexité.

Topologie en *Arbre* Un arbre est un graphe connexe sans cycle. Un arbre peut aussi être défini comme un graphe connexe tel que pour tout couple de sommets (u, v), il existe une unique chaîne entre u et v. Les sommets d'un arbre sont appelés nœuds de l'arbre. L'un des nœuds est appelé la racine. On dit que l'arbre est *enraciné* (il est aussi appelé *arborescence*, la racine constitue le sommet le plus haut). La figure 1.5 illustre un système distribué sous forme d'un arbre.

On dit que v est un *descendant* d'un nœud u, si le nœud u est sur le chemin entre v et la racine de l'arbre. Si u et v sont reliés par une arête alors on dit que u est le père de v. Tout nœud qui ne possède pas d'enfant est une $feuille$.

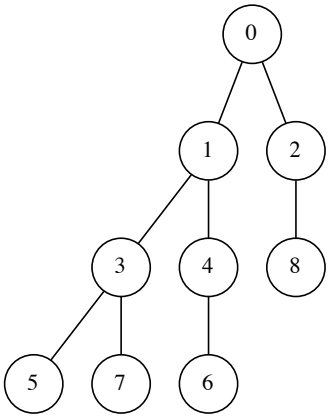


FIGURE 1.5 – structure en arbre

On trouve de nombreux systèmes organisés selon un arbre, on pourra citer pour les systèmes pair-à-pair *Napster* [FP99a], *Bittorrent* [Coh08a] et les nouvelles versions de *Gnutella* [KM02a]. La topologie en arbre est aussi largement utilisée dans le cadre des grilles de calcul comme dans les cas de $DIET$[CD06] et *Globus* [Fos05] qui ont une organisation hiérarchique, sous la forme d'un arbre.

La structure en arbre offre des propriétés intéressantes, notamment dans les cas de recherche et de diffusion. Elle permet notamment d'offrir des communications efficaces entre différents nœuds. Cette topologie se montre aussi peu tolérante face aux pannes qui peuvent impacter le système : la panne d'un nœud autre qu'un feuille implique la perte de la connexité du système.

Topologie en *Grille* Une grille K_{pXq} est un graphe composé de pXq sommets $v_{i,j}$ avec $1 \leq i \leq p$, $1 \leq j \leq q$, dans lequel deux sommets(ou nœuds) $v_{i,j}$ et $v_{k,l}$ sont adjacents si $|i - k|$

$+ |j - l| = 1$. Où p représente le nombre de lignes et q le nombre de colonnes. La figure 1.6 illustre un système distribué sous la forme d'une grille.

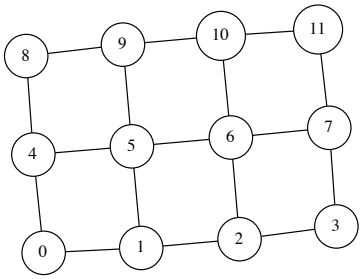


FIGURE 1.6 – Une grille 3 x 4

L'exploitation d'un grille permet de simplifier toutes les opérations de routage. En effet, comme dans le cas de l'anneau, il est possible d'orienter les liens de communication et de nommer les nœuds afin de rendre le routage implicite. Par contre la moindre panne d'un site détruit la structure, cette topologie est donc difficilement adaptable face aux changements topologiques.

Topologie *Graphe complet* Un graphe est dit *complet*, aussi appelé *clique* si pour toute paire $\{u, v\}$ de sommets, il existe une arête dont les extrémités sont u et v. Un graphe complet à n sommets est noté K_n. La figure1.7 illustre un graphe complet à 5 sommets.

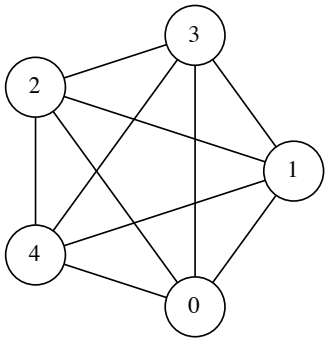


FIGURE 1.7 – Un graphe complet à 5 sommets.

la clique se montre particulièrememnt efficace face aux changements topologiques et aux pannes. Les communications sont efficaces car elles ne nécessitent que d'un seul échange de message pour tout échange ente deux sommets. Par contre, cette topologie implique une explosion du nombre de liens de communications, et de ports de communication en chaque nœud.

1.1.3 Topologie physique et topologie logique

L'ensemble des topologies présentées peuvent être physiques ou logiques. Dans le cas d'une topologie physique, le graphe de communication est organisé selon la topologie régulière définie. Les propriétés liées aux topologies peuvent donc être exploitées « directement » : dans le cas d'un anneau, chaque neud n'aura que 2 voisins ...

Il est aussi possible de construire une topologie *virtuelle* sur une topologie physique. dans ce cas, on parle de topologie *logique*. Une topologie logique est un donc sous-graphe $G' = \{V', E'\}$ de $G = \{V, E\}$ pour lequel $V = V'$ et $E' \subseteq E$, G' présente une topologie régulière. Il est donc possible de *plonger* un anneau sur un graphe quelconque, et ainsi, de bénéficier des propriétés des anneaux, alors que le graphe G d'origine ne présente aucun propriété particulière. La figure 1.8 montre le plongement d'un anneau virtuel sur un graphe quelconque.

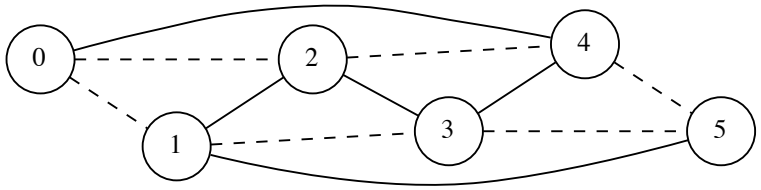


FIGURE 1.8 – Un graphe complet à 5 sommets et un anneau logique

L'utilisation des arbres comme structure logique est courante, avec le calcul de structures couvrantes des systèmes distribués. La figure 1.9 montre le plongement d'un arbre virtuel sur un graphe quelconque.

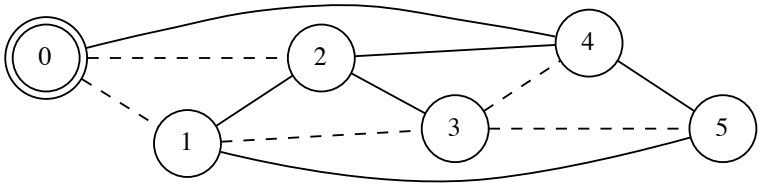


FIGURE 1.9 – Un graphe complet à 5 sommets et un arbre logique

1.2 Modèle d'exécution

Comme nous l'avons vu précédemment, un système distribué peut être représenté par un graphe. Une exécution distribuée quant à elle, peut être modélisée par un système de transitions.

Nous rappelons la définition d'un système de transitions donnés par Tel [Tel94]

Définition 1.8. *Un* ***système de transitions*** *est un triplet* $S = \{C, \rightarrow, I\}$ *tel que :*

1. C *est un ensemble fini ou infini d'états ;*
2. $\rightarrow$ *est une relation binaire de* C *dans* C *;*
3. I *est le sous-ensemble des configurations, ou états, initiales de* C*.*

Un algorithme distribué est donc totalement décrit par le système de transition qui lui est associé. La description d'un algorithme distribué est donc équivalente à la description du système de transitions correspondant.

La terminaison d'un algorithme distribué est conditionnée par la « découverte » d'un état terminal.

Définition 1.9. *Soit un système de transition* S*, un état* $\alpha \in C$ *est dit* ***terminal*** *si* $\forall \beta \in C$*, alors* $\neg(\alpha \rightarrow \beta)$*.*

Maintenant que nous avons décrit la notion de système de transition, nous pouvons, à l'aide de cette notion, donner la définition de l'exécution d'un algorithme distribué.

Définition 1.10. *Soit* $S = \{C, \rightarrow, I\}$ *un système de transition associé à l'exécution d'un algorithme* $\mathcal{A}$*, une* ***exécution*** *de* S *est une séquence maximale finie* $E = (\gamma_0, \gamma_1, \gamma_2 ...)$*, pour laquelle* $\gamma_0 \in I$ *et* $\forall i \geq 0, \gamma_i \rightarrow \gamma_{i+1}$*.*

Définition 1.11. *Un état* β *est dit* ***atteignable*** *depuis un état* α*, noté* $\alpha \rightsquigarrow \beta$*, s'il existe une séquence d'état* $(\gamma_0, \gamma_1, \ldots, \gamma_k)$ *telle que* $\gamma_0 = \alpha$ *quelque soit* $0 \leq i < k$*,* $\gamma_i \rightarrow \gamma_{i+1}$ *et* $\gamma_k = \beta$*.*

L'ensemble de ces définitions nous permettent donc de formaliser l'exécution d'un algorithme réparti. Afin de faire évoluer le système, il est donc nécessaire de disposer d'échanges et donc de proposer un modèle pour les communications.

1.3 Communications dans les systèmes répartis

L'exécution d'un algorithme dans les systèmes distribués dépend de la manière dont les différents sites communiquent. Deux modèles de communications principaux sont présents dans la littérature, le modèle à *mémoire partagée* et le modèle à *passage de messages*. Une fois le mode d'échange de données fixé, il est nécessaire de fixer les protocoles de communication que l'on va exploiter.

1.3.1 Modèles de communication

Le modèle à mémoire partagée

Dans ce modèle, les sommets ne communiquent pas directement par un canal de communication. Les échanges d'informations se font grâce à un espace mémoire partagé entre les différents flux d'exécution, les différents nœuds. Cet espace partagé est accessible à tous en lecture, à l'aide de la primitive `Read`, l'accès en écriture à l'aide de la primitive `Write` peut être ouvert à tous, ou réduit à certains sites.

L'utilisation de ce modèle impose de préciser les différentes notions vues précédemment. Dans le cas, les ports de communications associés aux nœuds sont des références vers des espaces partagés. De même la notion de voisinage évolue ; sont voisins deux nœuds u et v qui partagent l'accès à un même espace mémoire. On pourra remarquer qu'un même espace mémoire peut être accessible à un ensemble de nœuds.

Outre l'évolution du modèle précisé ci-dessus, une attention toute particulière doit être portée à la gestion des échanges. En effet, l'espace mémoire partagé présente toutes les propriétés et les contraintes liées aux partages de données :

— la nécessité de gérer les accès concurrents : si plusieurs accès peuvent se faire simultanément en lecture, il n'en va pas de même dans le cas d'une ou de plusieurs écriture ;
— l'assurance de la consommation des données : il est nécessaire de mettre en place un mécanisme spécifique permettant au nœud qui a écrit la donnée de savoir si elle a été lue par son destinataire, ou par lesquels des destinataires.

Le modèle à mémoire partagée est utilisée dans le cadre de la programmation *multi-threadée*, il ne correspond pas aux nécessités ni aux besoins de communication entre des stations réparties distantes. Dans ce second cas, on utilisera le modèle à passage de messages.

Le modèle à passage de messages (*message passing*)

Lorsqu'un sommet veut communiquer avec un autre, il doit lui envoyer un message qui transite via un support de communications. Les primitives exploitées dans le cadre de ce modèle sont `Send` et `Receive`.

L'utilisation de ce modèle nécessite de poser des hypothèses sur les canaux de communication :

— leur taille peut être bornée, finie, mais aussi non bornée ou infinie : le choix effectué influencera la capacité de communication, ainsi que les performances de l'algorithme. Des canaux de comunication à taille finie et bornée permettront un contrôle complet de la communication, et notamment permettront une gestion simplifiée des échanges, a contrario, des canaux de communication de taille infinie rendront le contrôle des communications plus complexe à mettre en œuvre ;
— le mode de transmission peut être *FIFO* (*First In First Out*) garantissant que les messages seront reçus dans l'ordre d'émission, a contrario, si les canaux ne présentent pas cette propriété, l'ordre des messages n'est pas respecté, il appatient donc à l'algorithme de gérer le séquencement des échanges. Certains modèles sont encore plus contraignants (causal, ordre total ...)

Si nous considérons des nœuds répartis, le modèle à passage de messages est celui qui se rapporche le plus de la réalité. Dans le cadre de cette thèse, nous considèrerons un modèle à

passage de messages, dont les canaux de communication sont *FIFO*, de taille finie mais non bornée.

1.3.2 Protocoles de communication

L'algorithmique distribuée étant basée sur l'exécution d'un calcul concurrent, les protocoles de communication y prennent un place particulière. En effet, l'exécution d'un algorithme distribué ne peut se faire correctement que si les communications entre les sites sont assurées. De l'efficacité du protocole de communication utilisé dépendront, en partie, les performances de l'algorithme. De plus, le protocole de communication mis en place permettra ou non l'implémentation de l'algorithme : un algorithme générant un nombre déraisonnable de messages ne pourra pas être implémenté car, bien que les opérations effectuées soient légales, les messages perturberont et satureront le réseau.

Les protocoles de communication peuvent être séparés en deux parties : les protocoles assurant la transmission d'informations d'une source vers tous les sites du réseau et les protocoles assurant la transmission cohérente d'informations entre deux sites, voisins ou non.

Source unique, destination unique L'écriture d'un protocole de communication garantissant la validité du transfert d'informations entre deux sites est indispensable. En effet, le résultat d'un algorithme distribué, dépend en grande partie de la validité des informations reçues, de l'assurance que tous les messages émis seront finalement reçus, que chaque message ne sera pris en compte qu'une et une seule fois, et que le contenu du message ne sera pas corrompu, ou que la corruption sera détectable.

— Le protocole de la «fenêtre glissante» (*Balanced sliding window protocol*) [Tel91] permet la transmission d'une série d'informations de façon sûre entre deux voisins. Dans ce protocole l'émetteur émet une série de données (la fenêtre) vers la destination. Le destinataire fait un accusé de réception. Ce protocole assure une transmission d'information sûre autorisant la perte de messages durant le transfert. Une application particulière de ce protocole est connu sous le nom de «protocole du bit alterné» (*alternate bit protocol*) [Lyn68].

FIGURE 1.10 – Principe de la fenêtre glissante

— Le protocole basé sur des «timers» [FW78] quant à lui, est capable d'assurer le transfert valide d'informations malgré la perte, la duplication ou le changement d'ordre des messages. Contrairement au protocole précédent, le protocole basé sur des «timer» nécessite l'ouverture d'une connexion. Une fois la connexion mise en place, chaque message est estampillé par une valeur permettant d'en connaître l'ordre à l'origine.

FIGURE 1.11 – Principe de la communication par timer

Les protocoles présentés permettent de transférer des informations en en assurant la cohérence. Ils ne permettent de pallier que certaines erreurs pouvant affecter les transmissions. Nous verrons par la suite, qu'il existe, selon certaines hypothèses des protocoles permettant de tolérer de façon bien plus efficace et plus complète, les fautes affectant la transmission des données.

1.4 Synchronisme dans les systèmes répartis

L'exécution d'une tache de calcul dans les systèmes distribués peut se faire suivant deux modes de synchronisation différents : par le modèle synchrone ou par le modèle asynchrone.

1.4.1 Le modèle asynchrone

Un système asynchrone est composé d'un réseau de communication connectant un ensemble fini de sites (ou nœuds) par des canaux bidirectionnels. La modélisation d'un tel système est donnée dans la section 1.1. Les échanges se font à l'aide de messages véhiculés par les canaux. Dans le système asynchrone, il n'y a pas d'horloge et un message envoyé depuis un site (ou nœud ou processus) arrive en un temps arbitraire, fini, mais non borné. Les liens de communication sont fiables, mais asynchrones, c-à-d., un message émis depuis un processus à l'ensemble de ses voisins arrive en un temps fini, mais non prédictible. Il faut noter que les communications ne sont pas nécessairement FIFO (*First In First Out*), c-à-d., les messages peuvent arriver dans un ordre différent que celui dans lequel ils ont été envoyés.

Un site peut recevoir des messages de ses voisins, exécuter des calculs locaux et envoyer des messages à ses voisins. Les événements correspondants sont : réception, événement interne, émission. Le délai de transfert des messages est fini, mais non prévisible. Les auteurs dans[CM98] présentent un modèle de diffusion asynchrone.

Le modèle asynchrone peut être comparé au fonctionnement de la poste : l'émetteur place une requête dans un file de message et ne se soucie plus de son acheminement. Le destinataire se connecte sur le file de message pour récupérer la requête. Le destinataire n'a pas besoin d'être présent au moment du dépôt de la requête. Nous voyons donc que ce modèle n'exige pas la disponibilité permanente des nœuds entre le moment où le message envoyé par l'émetteur est parti et le moment où il est arrivé à destination.

Il faut constater que dans un système asynchrone, les nœuds évoluent à des vitesses à priori indépendantes les unes des autres, les envois et réceptions de messages étant les seuls moyens de les synchroniser [AIR+88].

1.4.2 Le modèle synchrone

Le modèle synchrone diffère de l'asynchrone par le comportement des nœuds et des canaux de communications. Dans le modèle synchrone il n'existe pas nécessairement d'horloge globale. Les horloges locales de tous les nœuds sont synchronisées. Lorsqu'un message est émis par un nœud, il est réceptionné par son destinataire en un temps fixé. Tout calcul interne à un nœud est aussi effectué en un temps fixé au préalable.

Tout message dans un système synchrone émis par un nœud à temps t, est reçu et traité par un nœud voisin à temps $t + 1$.

Dans [AIR+88], l'auteur formalise un système synchrone ainsi :
Soit $E_i(t)$ l'ensemble fini des événements dont le nœud P_i est le siège à l'instant t :

$$\forall P_i, P_j \in X \ , \forall p \in N$$

$$\{(envoyer\ m\ à\ j) \in E_i(t)\} \leftrightarrow \{(recevoir\ m\ de\ i\ )\ \in E_j(t)\}$$

1.5 Algorithmes classiques

La littérature regorge d'algorithmes distribués pouvant être considérés comme des «briques de base» pour la construction d'algorithmes plus complexes. Nous donnons dans cette partie une liste, non exhaustive, d'algorithmes, ou de types d'algorithmes, qui sont souvent utilisés au cours de l'écriture d'algorithmes distribués. Ces derniers permettent la mise en place de structures de contrôle, ou de cadres d'exécutions, pour les algorithmes distribués.

1.5.1 Algorithmes à vagues

Les algorithmes à vagues sont à l'origine de la résolution de nombreux problèmes d'algorithmique distribuée. En effet, dans bien des cas, il est nécessaire d'être en mesure d'assurer un contrôle du système ou de certaines opérations effectuées par le système. Les algorithmes à vagues permettent d'assurer une telle tâche en facilitant la *diffusion* d'information de façon efficace à travers le réseau, la *synchronisation* des différents processus du réseau ...

Les algorithmes à vagues permettent la mise en œuvre de ces contrôles. Pour donner une définition des algorithmes à vagues, nous nous appuyons sur celle donnée par Tel dans [**?**] : les algorithmes à vagues contiennent une instruction, ou série d'instructions particulières, appelée décision. On associe un événement à cette instruction, cet événement représente l'obtention du résultat pour un problème donné.

Définition 1.12. *Un algorithme à vagues est un algorithme distribué vérifiant les trois propriétés suivantes :*

1. *(terminaison) Chaque exécution de l'algorithme se fait en un temps fini ;*
2. *(décision) chaque exécution de l'algorithme contient au moins un événement de décision ;*
3. *(dépendance) pour chaque exécution, chaque événement de décision est précédé par au moins un événement en chaque processus.*

Bien que certains algorithmes à vagues aient été étudiés seuls, dans la majorité des cas, de tels algorithmes ont été proposés dans le cadre de la résolution d'un problème particulier :
— synchronisation comme dans [Fin79] ;
— algorithme de collecte d'information et écho comme dans [Cha82, Seg83] ;
— algorithme dit de phase comme dans [Tel91, Pel90]. Dans le second article, l'auteur propose d'effectuer, en plus, le calcul du diamètre du réseau.

1.5.2 Construction d'arbres

Les algorithmes de construction d'arbres sur un réseau, forment une classe particulière des algorithmes à vagues. En effet, de nombreux algorithmes distribués se basent sur de tels algorithmes pour effectuer une tâche particulière : exclusion mutuelle, élection, émission d'une information depuis un site vers tous les autres sites du réseau ...

Il existe de nombreux algorithmes de construction d'arbres, ces algorithmes répondent à des besoins différents et construisent, sur le réseau, des arbres dont les propriétés sont différentes.

L'algorithme de Tarry [Tar95] permet la construction d'un arbre quelconque sur un réseau. L'obtention d'une telle structure permet la diffusion efficace de messages au travers du réseau ou d'assurer l'exclusion mutuelle.

La construction d'arbres respectant des propriétés particulières a aussi été étudiée :

— Cidon dans [Cid88] et Awerbuch dans [Awe85] proposent, entre autre, un algorithme de construction d'arbres en respectant la profondeur d'abord [1]. Ce type de construction est obtenu en effectuant, à l'aide d'un jeton unique, un parcours des sites de façon à ce que lorsqu'un site reçoit un jeton, s'il n'est pas encore dans l'arbre en construction, il relaie le jeton successivement à tous ses voisins, qui explorent à leur tour leur sous arbre, avant de le repasser au site qui lui a passé le jeton, son père dans l'arbre construit. Les algorithmes de construction d'arbres en profondeur d'abord, ou de parcours de jeton respectant la profondeur d'abord, permettent la sérialisation des actions des différents sites d'un réseau : comme il n'existe qu'un seul jeton, il est facile de mettre en place un mécanisme d'autorisation d'effectuer des actions pour les sites.

— la construction d'arbres en largeur d'abord [2] est un problème différent du précédent : dans le cas précédent les arbres sont construits à l'aide d'un jeton unique qui effectue un parcours dit en profondeur d'abord dans le réseau. Dans le cas d'une construction en largeur d'abord, la construction se fait par « construction » de niveaux successifs : la première étape est d'inclure les sites à distance 1 de la racine dans l'arbre, puis les sites à distance 2 ... La construction d'un tel arbre peut se faire concurremment : les sites du même niveau sont atteints en même temps par plusieurs jetons.
Il existe d'autres stratégies, moins communes, de constructions d'arbres couvrants sur un réseau. Ces stratégies sont basées sur des hypothèses différentes : construction d'arbres de poids minimum, construction d'arbre central ...

1.5.3 Algorithmes d'élection

Le problème de l'élection, ou de la désignation d'un leader, a été proposé par Lelann [LeL77]. Le principe général d'une élection est : à l'initiation de l'algorithme tous les sites ont le même état et, une fois l'algorithme terminé, un et un seul site est élu, ou désigné comme étant le leader.

Un algorithme d'élection respecte les propriétés suivantes :

1. chaque site possède localement le même algorithme ;
2. l'algorithme est décentralisé : il peut être initié par un ensemble de sites ;

1. Depth First Search
2. Breadth First Searche

3. l'exécution de l'algorithme mène toujours en une configuration dans laquelle un et un seul site est élu, alors que les autres sites sont dans l'état perdant.

Les premiers algorithmes d'élection proposés, le furent sur des anneaux. Dans le cadre de telles topologies, les algorithmes d'élection offrent de bonnes performances. Comme le précise Lavallée dans [Lav90] le problème de l'élection se rapporte à celui de la structuration : l'attribution d'un privilège à un processus unique nécessite dans bien des cas la mise en place d'une structure virtuelle particulière sur le réseau.

1.5.4 Algorithmes de snapshots, détection de propriétés

Par hypothèse, les différents sites des systèmes distribués ont une connaissance très limitée du réseau. Selon les modèles, chaque site ne connaît que son état local, ou au mieux, l'état de l'ensemble de ses voisins. L'évaluation de l'état du système ne peut donc se faire que localement.

Il est cependant nécessaire de connaître l'état actuel de l'algorithme : il faut pouvoir vérifier la validité d'une propriété ou encore tester la terminaison d'un algorithme distribué. Le test de terminaison ne peut être calculé par un site sans la mise en place d'un mécanisme spécifique de collecte d'informations. Le calcul de l'état des différentes propriétés globales d'un système distribué est présenté dans [Lav90] : par échange de message contenant l'état de chaque processus, tous les sites du système sont en mesure de calculer, en un temps fini, l'état global du système et plus particulièrement, d'en calculer la terminaison.

Il est aussi possible d'étudier la configuration globale en utilisant un algorithme de prise d'instantanés [3]. Ce type d'algorithmes permet de collecter les informations relatives à l'état de chaque site, mais aussi à l'état des différents liens de communication. En effet, l'étude de l'état d'un système ne peut se résoudre à l'étude de l'état local de chaque site : pour connaître l'état global du système, il est nécessaire d'avoir accès à l'état des canaux de communication.

Chandy et Lamport proposent dans [CL85] un algorithme de calcul de l'état global d'un système distribué. Par inondation du réseau par des jetons, on provoque la prise de l'instantané en chaque site. Tant qu'un site n'a pas reçu un jeton de la part de tous ses voisins, il enregistre l'ensemble des communications émises ou reçues de ce voisin. Cet algorithme s'applique dans le cas de réseaux *FIFO*.

Pour pallier le problème des communication *FIFO* Lay et Yang proposent dans [LY87], un algorithme de prise d'instantané pour des réseaux de type non *FIFO*. Contrairement au précédent, cet algorithme utilise un système de marquage, d'estampille, pour évaluer la date des messages, et donc créer un ordre pour ces messages.

Le calcul d'état global d'un système distribué sert à détecter les propriétés stables : lorsque l'état de chaque composant du système a été collecté, il est possible de réaliser :

— la détection de terminaison d'un algorithme distribué ;
— la détection de configurations bloquants [4] ;
— la détection de perte de jeton.

Les algorithmes de snapshot permettent donc d'avoir une vue globale du réseau. A partir de cette vue globale, on peut évaluer l'état du système, ce qui n'est pas possible à réaliser localement.

3. snapshot
4. deadlock

1.6 Retour sur les algorithmes de routage

Comme nous l'avons vu ci dessus, il existe des protocoles permettant d'assurer la transmission valide des informations. Mais, lorsque les informations doivent transiter entre deux sites non voisins, il faut assurer la mise en place d'une «couche» permettant le transit des informations au travers du réseau. Les algorithmes de routage permettent la mise en place de cette couche et la connaissance, selon les stratégies de routage, des chemins reliant les différents sites. Les protocoles de transmission assurent la validité locale du transport tandis que les algorithmes de routage assurent le transport des informations au travers du réseau.

1.6.1 Spécification du problème du routage

Il est possible d'appliquer plusieurs stratégies au calcul des chemins, des tables de routage, le long desquels les messages vont transiter. Ces stratégies dépendent des impératifs et des contraintes imposées par le réseau sur lequel on souhaite mettre en place les algorithmes.

Dans la plupart des cas, un réseau n'est pas complet. C'est à dire qu'un nœud n'est pas relié directement à tous les autres nœuds du réseau. Pour communiquer entre eux, les nœuds envoient donc des séries d'informations, des paquets, par l'intermédiaire d'un sous ensemble de nœuds du réseau qui relaient le message depuis sa source jusqu'à sa destination. Le routage peut être donc défini comme l'ensemble des opérations qui permettent d'assurer ce transfert des informations.

Pour assurer ce transfert d'informations, il faut naturellement sauvegarder en chaque site un ensemble de données relatives à la topologie du réseau, c'est à l'aide de ces données que les décisions de routage seront réalisées. Ces informations sont appelées *tables de routage*.

Dans la littérature [BG92, Tel94, Tan97], le problème du routage est divisé en deux parties distinctes :

1. **Le calcul des tables de routage :** ce calcul doit être fait lors de la première phase du protocole de routage pour permettre le transfert valide des informations. De plus, ces informations doivent être mises à jour lors des évolution topologiques que connaît le réseau. Le calcul des tables de routage revient, comme le présente Lavallée dans [Lav90], à l'établissement logique de communications entre des processus non contigus ;
2. **Le transfert des informations :** lorsqu'un message est prêt à être envoyé, il doit «voyager» à travers le réseau pour arriver à destination.

La mise en place d'un protocole de routage doit donc assurer la gestion de ces deux parties, pour permettre le transfert des informations au travers du réseau.

1.6.2 Evaluation des algorithmes de routage

L'évaluation de la performance d'un algorithme de routage peut se faire selon plusieurs critères :

1. **Correction :** un algorithme de routage doit assurer que chaque message arrive à destination en un temps fini ;
2. **Complexité :** l'algorithme de calcul des tables de routage doit occuper le moins possible le système, que ce soit au niveau du nombre de messages, que du temps de calcul et de la taille des données sauvegardées ;

3. **Efficacité :** l'algorithme de routage doit être en mesure de faire transiter les messages par les meilleurs chemins possibles. La notion de meilleur chemin possible devant, bien sur, être précisée en fonction du réseau et des hypothèses ;

   En plus de ces éléments de base, il existe d'autres critères relatifs à la capacité de « réaction » de l'algorithme face à l'évolution des différents paramètres du réseau :

4. **Robustesse :** l'algorithme est en mesure de recalculer les chemins connus des sites, dans le cas d'une modification topologique ;
5. **Adaptivité :** l'algorithme est en mesure d'éviter les congestions et la saturation de certains liens, nœuds, en favorisant dynamiquement l'utilisation de certains chemins moins sollicités que d'autres ;
6. **Equité :** l'algorithme est en mesure d'assurer le même service à chaque utilisateur du réseau.

L'évaluation des performances d'un algorithme de routage est donc une mesure de chacun des paramètres précédents. Certains de ces paramètres sont opposés, la mise en place d'un algorithme optimal dépendra du système ainsi que des objectifs initiaux.

1.6.3 Classification des algorithmes de routage

Tel propose dans [Tel94] une classification des algorithmes utilisant des tables de routage. Cette classification prend en compte la notion de meilleur chemin et, en fonction de la définition de ce meilleur chemin, regroupe les algorithmes en trois classes principales :

1. les algorithmes calculant les routes en terme de nombre de déplacements minimaux [5], pour lesquels le meilleur chemin possible est celui comportant le nombre minimum de sommets ;
2. les algorithmes de plus courts chemins (*shortest path*) qui calculent la taille d'un chemin en faisant la somme des valuations des liens présents dans ce chemin. Dans ce cas toutes les valuations associées aux liens sont statiques et positives ou nulles ;
3. les algorithmes assurant un temps de transfert minimum au transfert de messages [6] qui assignent une valuation dynamique aux liens de communication. Cette valuation évolue durant l'exécution de l'algorithme et le protocole de calcul de table de routage modifie dynamiquement les tables de routage pour permettre une exploitation efficace du réseau.

Cette classification ne prend en compte que les algorithmes calculant des tables de routage optimales au sens de la taille des chemins ou du temps de transfert. De plus, elle ne prend pas en compte le paramètre d'occupation mémoire ou de complexité en temps au niveau des sites du réseau.

Tanenbaum propose dans [Tan97] une autre classification des algorithmes de routage. Cette autre classification est basée sur « l'étendue » des connaissances nécessaires à chaque site pour assurer le calcul des tables de routage et le transfert des messages. Les algorithmes de routage sont classés en trois classes :

5. minimum hop
6. minimum delay algorithms

1. les algorithmes à connaissance globale : dans ces algorithmes, un ensemble de sites, a besoin de connaître des informations globales sur le réseau, informations sur la topologie, ou la structure complète du réseau par exemple ;
2. les algorithmes à connaissance distribuée : dans ces algorithmes les sites du réseau utilisent les informations d'un sous ensemble de sites du réseau pour effectuer le calcul de leurs tables de routage. Ce sous ensemble est le plus souvent l'ensemble des voisins. Dans ce cas, l'information «voyage» dans le réseau par relais successifs des différents sites à leurs voisins ;
3. les algorithmes à connaissance locale : dans ces algorithmes les sites n'ont besoin que de leur connaissance locale pour assurer le calcul des tables de routage, dans le cas où il y a tables de routage, et pour relayer les messages.

Selon les contraintes imposées : contraintes de temps de transfert, contraintes d'occupation mémoire, contrainte de capacité des liens de communication, plusieurs familles de protocoles de routage ont été définis : les algorithmes de routage respectant les chemins minimaux, les algorithmes de contrôle de flots, les algorithmes aléatoires de transfert d'information ...

1.6.4 Calcul de la meilleure route

Le calcul des plus courts chemins [7], consiste à rechercher pour chaque paire (i, j) de sites du réseau le plus court chemin les reliant. Ces algorithmes utilisent, généralement, deux structures permettant, dans un premier temps le calcul des chemins et le transfert des informations : les tables de routage. Classiquement la première table de routage est utilisée pour enregistrer les distances d'un site vers tous les autres et la seconde table enregistre le «voisin préféré», c'est à dire le voisin par lequel il faut passer pour atteindre un site particulier en garantissant que le chemin utilisé sera le plus court possible dans le réseau.

Toueg propose dans [Tou80] un algorithme distribué utilisant les informations connues en chaque site. Cet algorithme calcule les plus courts chemins dans un réseau, mais n'est pas en mesure de tolérer les fautes de lignes ou de sites : une fois les chemins calculés si un site, ou ligne, apparaît ou disparaît, il faut relancer l'algorithme pour permettre le recalcul des nouveaux chemins.

Merlin et Segall proposent dans [MS79] un algorithme basé sur le calcul, pour chaque destination, d'arbres, en largeur d'abord. Contrairement à l'algorithme de Toueg, l'algorithme de Merlin et Segall permet de supporter les modifications topologiques ainsi que les modifications du poids des arêtes.

Nous citerons enfin l'algorithme *Netchange* de Tajibnapis[Taj77] qui permet le calcul de tables de routage respectant les plus courts chemins et assurant le recalcul des chemins suites aux modifications topologiques touchant les liens comme les sites du réseau.

1.6.5 Routages compacts

Les algorithmes présentés ci dessus assurent le calcul de tables de routage au sens des plus courts chemins, en conservant en chaque site des tables de routage contenant explicitement chaque destination. Mais ces tables de routage occupent une grande place mémoire : dans le

7. shortest path computation

cas d'un algorithme de routage utilisant une table des distances et une table des voisins préférés, $n * log(n-1)$ pour la table des distances et $n * log(n)$ pour la table des voisins préférés. Pour pallier ce problème d'autres stratégies de calcul de chemins ont été élaborées : les stratégies de routage avec des tables compactes.

1. **Routages par intervalle**

   La première approche de réduction de la taille des tables de routage, proposée par Santoro et Khatib dans [SK85], est connue sous le nom de routage par intervalles. Le principe est d'étiqueter les nœuds d'un arbre en utilisant un parcours en profondeur. Ensuite, en chaque site, à chaque lien de communication, est associé un intervalle d'identités de sites. Par un lien on peut accéder à tous les sites compris dans l'intervalle qui lui est associé.

   La notion de routage par intervalle a été étendu aux réseau quelconques par Van Leeuwen et Tan [LT87] qui ont montré que pour tout réseau connexe, il existe un étiquetage par intervalle valide, en effet, chaque réseau connexe accepte au moins un arbre couvrant.

   Les algorithmes de routage par intervalle ne permettent de calculer des routages de type optimaux, au sens des plus courts chemins, que sur certaines topologies particulières comme les arbres, les anneaux, les grilles, les hypercubes. Par contre dans la majorité des cas les routages par intervalles ne permettent pas le calcul des chemins optimaux, au sens des plus courts chemins.

2. **Routage pré-fixés**

   Pour pallier les désavantages du routage par intervalle, Bakker, Van Leeuwen et Tan proposent un étiquetage d'arbre quelconques dans [BLT93]. En effet, l'utilisation d'un arbre en profondeur ne peut garantir de bonnes performances quant à la taille des chemins calculés. Cette méthode, qui permet d'accroître la performance permet, en plus, de gagner en robustesse : en effet, la modification d'un composant au réseau peut se faire sans recalculer l'ensemble de l'arbre.

   Cette méthode est basée sur l'utilisation d'une chaîne d'étiquetage des liens et des nœuds du réseau. La comparaison de la chaîne représentant l'identité de la destination et de la chaîne identifiant le site courant permet d'assurer le routage des informations au travers du réseau.

Routages aléatoires

On trouve aussi dans la littérature, un autre type de stratégies de routage : les stratégies aléatoires. L'algorithme aléatoire le plus connu est celui dit de «la patate chaude» [8] [Bar64] aussi appelé routage par déflection. Le principe de cet algorithme est très simple : il consiste à relayer automatiquement vers un voisin choisi aléatoirement, un message reçu, tant qu'il n'a pas atteint sa destination.

Cette méthode permet de s'affranchir de protocole de calcul de tables de routage et d'étude de ces tables de routage pour assurer le transfert des informations, ce qui permet, à la fois, un gain d'encombrement mémoire en ce qui concerne les tables de routage, et un gain au niveau de l'occupation des sites.

8. hot potatoe

Par contre, cette méthode ne permet pas de donner de résultats exacts concernant la distance parcourue par un message ou le temps de transfert d'une information entre deux sites. En utilisant cette stratégie de routage, il est impossible d'évaluer la distance parcourue par chaque message et contrairement aux algorithmes présentés ci-dessus, il n'y a aucune garantie sur l'utilisation de chemins sans boucles lors du transfert des messages.

1.7 Conclusion

Nous avons présenté dans ce premier chapitre, les fondememnts des systèmes distribués. Les éléments de modélisation présentés permettent la rédaction de méthodes et d'algorithmes de haut niveau. Dans la suite de cette thèse, nous utiliserons un modèle à passage de messages asynchrone, modèle le plus proche de la réalité des réseaux.

Chapitre 2

Les applications réparties

Résumé. *Dans le premier chapitre, nous nous sommes focalisés sur la modélisation des systèmes. Dans ce second chapitre, nous allons étudier plus particulièrement les solutions de mise en place des communications, ainsi que les applications qui peuvent être bâties autour de ces communications.*

Dans un premier temps, nous rappelerons les modes de gestion des communications, puis donnerons la modélisation des applications réparties. Ensuite, nous nous intéresserons à la gestion des données dans les systèmes répartis, et nous finirons ce chapitre par la définition du modèle que nous utiliserons dans la suite de cette thèse : le modèle pair-à-pair.

Sommaire

2.1 Introduction 27
2.2 Gestion des communication 28
2.3 Modèles des applications réparties 29
2.4 La gestion des données dans les applications réparties 33
2.5 Données et systèmes pair-à-pair(P2P) 39
2.6 Réseaux structurés décentralisés 46
2.7 Conclusion 54

2.1 Introduction

Dans le chapitre 1, nous avons présenté le modèle théorique sur lequel nous nous sommes basés pour effectuer ces travaux. Ce modèle théorique définit les différents aspects de l'architecture des systèmes à l'exécution des algorithmes, en passant par la gestion des communications.

Lorsque l'on passe de la phase de modélisation théorique à la phase de conception d'une application, on se base sur les modèles précédents mais on doit intégrer des éléments techniques externes qui vont notamment limiter les possibilités de comunication. C'est le cas des éléments de sécurité ou de hiérarchie des réseaux, qui bien que le graphe de communication général soit connexe, empêche dans certains cas certaines communications.

La prise en compte de ces éléments est parfaitement claire lorsque l'on étudie les modèles classiques des réseaux informatique, qu'il s'agisse du modèle OSI ou du modèle TCP/IP.

L'ensemble des éléments présentés dans ce chapitre se focalisent sur l'aspect développement des applications, et exploite les couches les plus hautes des différents modèles.

2.2 Gestion des communication

Comme nous l'avons dit précédemment, bien que le graphe d'interconnexion soit connexe, la communication entre deux entités peut très bien se faire de manière bi-directionnelle, unidirectionnelle, et même impossible de manière directe. Il est donc nécessaire de définir et de formaliser différents modes de communications qui permettront à deux nœuds de communiquer, malgré les limites imposées par la structure des réseaux et la sécurité.

Les modes de communication ainsi définis sont au nombre de 3 : le mode *push*, le mode *pull* et le mode *proxy*.

2.2.1 Le modèle *push*

Le mode *push* est le mode de communication le plus simple. Il permet à un nœud u qui souhaite transmettre une information à un nœud v d'envoyer directement cette information dans un message adressé à v. Dans le cas d'un mode *push* la communication n'est pas limitée dans le sens $u \to v$ par un dispositif réseau ou de sécurité.

Lorsque les deux sont en mesure de transmettre leurs messages selon le mode *push*, on parlera de mode *push* bidirectionnel. Si la communication en mode *push* ne peut se faire que dans un seul sens, on parlera de mode *push* unidirectionnel. En effet dans ce cas, u peut envoyer directement un message à v et v peut directement envoyer un message à u. Ce mode *push* bidirectionnel est classique si l'on considère deux machines d'un même réseau local, mais dès le moment ou les deux machines sont dans deux réseaux différents, ou à deux niveaux de hiérarchie différents, le mode *push* peut devenir au mieux unidirectionnel : la communication peut se faire dans un sens et non dans l'autre sens. C'est le cas de l'accès à un site web public depuis une machine positionnée dans un réseau local.

Le mode *push* est basé sur l'exploitation des deux primitives : `send` qui permet l'émission d'un message, `receive` qui en permet l'attente et la réception.

2.2.2 Le modèle *pull*

Comme nous venons de le préciser précédemment, il n'est pas toujours possible d'utiliser une communication de type *push* bidirectionnel. Si l'on se trouve dans un cas d'échange de type *push* unidirectionnel, si l'on considère u et v deux voisins, u à la possibilité d'envoyer une information à v directement, alors que v ne peut envoyer d'information à u. La communication $u \to v$ se fait donc selon le mode *push*, par contre, il est nécessaire de mettre un mécanisme particulier pour permettre un transfert des données de v à u : le mode *pull*.

Le mode *pull* n'est pas basé uniquement sur l'exploitation des primitives `send` et `receive`, mais sur la mise en place sur le nœud détenant les informations à envoyer d'un service de mise à disposition des données. A intervalle régulier, le destinataire des données invoquera ce service pour vérifier s'il n'y a pas des données qui lui sont destinées.

Donc si on considère u et v deux nœuds voisins, tels que $u \to v$ mais que $u \nrightarrow v$ alors u sera en charge de la gestion des données qu'il va transmettre à v, mais aussi des données qu'il ira

quérir à v. Ce mode permettra donc de réaliser le transfert logique des données dans les deux sens, en se pliant aux contraintes de communication.

Le mode *pull* est basé sur l'exploitation de deux primitives : `get` qui permet à un site d'aller chercher s'il existe un message à sa destination, et la primitive `provide` qui met à disposition les messages à destination du premier acteur.

2.2.3 Le modèle *proxy*

il existe aussi un cas de figure dans lequel les acteurs ne peuvent communiquer à l'aide du mode *push* dans aucun sens. Dans ce cas, si l'on souhaite transmettre des informations entre deux nœuds u et v, il est nécessaire de passer par un troisième acteur qui va servir de canal de communication « externe ». On parle dans ce cas d'un mode de communication *proxy*.

Dans le mode de communication *proxy* quand un nœud u souhaite envoyer une information à un nœud v, il « poste » cette information sur un proxy p (une file de messages par exemple). Périodiquement, comme dans le cas du mode de communication *pull* le nœud v va interroger p pour récupérer les données qui lui seraient destinées.

le modèle *proxy* permet donc de créer un voisinage « artificiel » entre deux nœuds u et v en exploitant un nœud accessible à la fois de u et de v.

2.3 Modèles des applications réparties

Dans la section 2.2, nous avons passé en revue les différents modèles d'échanges entre deux nœuds du réseau. En se basant sur les échanges effectués, et afin de réaliser des applications, il est nécessaire de définir des modèles de structuration des applications en attribuant aux différents acteurs des tâches spécifiques et des traitements à réaliser.

2.3.1 Le modèle Client / Serveur

Dans le modèle Client/Serveur, l'architecture est complètement centralisée. Elle est constituée d'un serveur sur lequel se connectent tous les clients. Le serveur fournit un service réseau à un ensemble de clients.

Plusieurs définitions du modèle Client/Serveur ont été proposée dans la littérature on citera notamment celle de Schollmeier [Sch01] qui définit le modèle Client/Serveur comme « un réseau distribué qui se compose d'un système supérieur performant, qui est le serveur, et de plusieurs systèmes de performance généralement plus faible, qui sont les clients. Le serveur est l'unité d'enregistrement central ainsi que le seul fournisseur de contenus et de services. Un client demande le contenu ou l'exécution de services, sans partager aucune de ses propres ressources ».

Le serveur a pour charge de mettre à disposition des services. Ces services peuvent être par exemple :

1. de la mise à disposition de fichiers, comme le font les serveurs FTP (*File Transfert Protocol*) ;
2. de la mise à disposition de contenu multimédia diffusé, comme les radios et télévisions en ligne ;

3. des sites qui donnent accès aux bases de données ou bien le service DNS (*Domain Name System*) qui traduit les noms de domaines en adresse IP.

Plusieurs applications reposent sur le modèle Client/Serveur parmi lesquels on peut citer les projets de calcul distribué SETI*@home*[ACK+02], BOINC[And04], NetSolve[SYAD05] et le système de calcul XtremWeb[FGNC01]. Dans ces systèmes tous les clients se connectent aux Serveurs pour récupérer ou stocker leurs données.

Pour répondre à la problématique de passage de l'échelle du modèle Client/Serveur quand la charge dépasse les capacités du serveur, les systèmes distribués pair-à-pair ont vu le jour.

2.3.2 Modèles P2P

Le modèle Pair-à -Pair propose de décentraliser les services et les mettre à disposition, en supprimant le serveur central et en mettant à contribution les clients pour qu'ils indexent leur propres ressources. Dans ce modèle chaque client devient un *fournisseur* potentiel de services. Dans cette architecture les clients appelés pairs sont identiques dans le sens où ils communiquent d'égal à égal, et ils jouent à la fois le rôle de client et serveur, on utilise généralement le terme de *servent*. Plusieurs définitions de système pair-à-pair ont été donné dans la littérature.

Selon Schollmeier[Sch01] une architecture de réseau distribué peut être désigné comme un réseau pair-à-pair (P2P), si les nœuds (ou participants) partagent une partie de leurs propres ressources matérielles (puissance, de capacité de stockage, la capacité de traitement etc.). Ces ressources partagées sont nécessaires pour fournir le service et le contenu offert par le réseau (par exemple, le partage de fichiers ou espaces de travail partagés pour la collaboration). Ils sont accessibles par d'autres nœuds (ou pairs) directement, sans passer des entités intermédiaires. Les participants d'un tel réseau sont donc des ressources fournisseurs (services et de contenus) ainsi que les ressources demandeurs (service et contenu).

Androutsellis and al.[ATS04] proposent un autre définition des systèmes pair-à-pair, ce sont des systèmes distribués qui consistent en une interconnexion de nœuds capables de s'auto-organiser en topologies réseau dont le but est de partager des ressources tel que du contenu, des cycles CPU, de l'espace de stockage et de la bande passante.

Quelle que soit la définition, les systèmes pairs-à-pairs que nous considèrerons dans la suite sont caractérisés par :

1. **Le passage à l'échelle :** grâce à sa structure totalement décentralisée, le modèle pair à pair peut interconnecter un grand nombre d'utilisateurs ;
2. **L'auto-organisation :** ce modèle s'adapte aux fautes (pannes) des nœuds et tolère la mobilité des nœuds (arrivées et départs des nœuds). Ces architectures pairs-à-pairs sont très résistantes aux pannes : même si une partie du réseau tombe en panne, ces architectures peuvent continuer de fonctionner.

Ces modèles pair-à-pair sont utilisés dans plusieurs types d'application. Dans le cadre de cette thèse, nous nous intéressons aux modèles pair-à-pair utilisés par le stockage, le partage de fichiers. Ainsi leurs modes de recherche, d'indexation et d'accès aux données.

Bien que la définition précédente soit assez explicite la structure des systèmes pairs-à-pairs permet de les classer en différentes familles. R. Schollmeier[Sch01] distingue deux modèles de réseaux pair-à-pair suivant leur architecture : le modèle pair-à-pair pur et le modèle pair-à-pair Hybride.

Modèle pair-à-pair pur

Selon Schollmeier[Sch01] une architecture pair-à-pair pure se définit ainsi « Une architecture de réseau distribué doit être classé comme un réseau pair-à-pair pur, si elle est d'abord un réseau pair-à-pair selon la définition 2.3.2 et deuxièmement si toute entité arbitraire peut être supprimée du réseau sans que le réseau subisse de perte de service ».

Dans les systèmes pair-à-pair purs les nœuds ont le même rôle et indexent leur propre contenu. Ce qui leurs confère un caractère totalement décentralisé.

Suivant les méthodes d'acheminement de messages et leurs topologies, dans [LCP$^+$] les auteurs distinguent le modèle pair-à-pair décentralisé structuré et le modèle pair-à-pair décentralisé non-structuré.

pair-à-pair pur non-structuré Les réseaux pair-à-pair pur non-structurés ne disposent d'aucune topologie d'interconnexion. Leur graphe de connexion se construit de manière pseudo-aléatoire. La construction du graphe aléatoire se fait ainsi :

— un nœud entre dans le réseau par l'intermédiaire d'un autre nœud déjà connecté ;
— une fois inséré, le nœud sonde de manière périodique son voisinage afin de maintenir et découvrir un certain nombre de connexions.

Donc les nœuds se connectent entre eux sans contraintes à leur entrée dans le réseau. L'absence de contraintes sur la topologie ne facilite pas la recherche d'une information. Il n'y a aucune contrainte entre la localisation d'une information et la topologie du réseau. La recherche d'une information dans de tel réseau se fait par inondation (diffusion).

L'inondation consiste à retransmettre un message dans tout le réseau : chaque nœud qui reçoit le message pour la première fois répète le message. Ainsi, de proche en proche, le message inonde le réseau.

Dans [RB13] l'auteur formalise la définition de la structure de diffusion ainsi : *Soit le système constitué de n nœuds. Chaque nœud n_i possède une vue aléatoire et partielle $d_i \subseteq n$ de l'ensemble du système.*

1. lorsqu'un nœud n_i veut propager une information à l'ensemble du système, il initie un nouvel identifiant d'information et envoie cette information à $k \subseteq d_i$ éléments tirés aléatoirement ;
2. lorsqu'un nœud n_i reçoit une information, s'il ne l'a pas déjà traitée, il la traite et la propage à $k \subseteq d_i$ éléments tirés aléatoirement.

pair-à-pair pur structuré La structure des réseaux pair-à-pair structurés s'inspire de graphes connus *a priori* comme les tores [RFH$^+$01a], les hypercubes [PRR99, RD01b, SMK$^+$01], les papillons [MNR02], les De Bruijn [FG06] ... ; qui contraint les relations de voisinage. Ces contraintes sont choisies pour alléger et accélérer les processus d'insertion et de recherche de données par un routage efficace qui limite le nombre de messages ainsi que la profondeur de la recherche.

L'adressage (ou clé) de chaque nœud est obtenu par hachage de son nom ou de son contenu dans le cas d'un système de gestion de données. Le hachage permet de répartir uniformément les clés sur les identifiants des nœuds. Les réseaux pair-à-pair structurés son dits des réseaux à contenu adressable.

Les systèmes pair-à-pair purs structurés implémentent une Table de Hachage Distribuée (DHT), qui permet la publication et la recherche de clés. Cette recherche de clé s'effectue de manière distribuée dans le réseau des nœuds connectés, chaque clé étant associée à un nœud. Ainsi pour une clé k, la DHT fait correspondre cette clé avec un nœud du réseau. Les DHT disposent de deux primitives : *put(k, v) et get(k)*. La primitive *put(k,v)* permet de stocker la donnée v sur le nœud en charge de la clé k et la primitive get(k) permet de récupérer la donnée associée à la clé k en interrogeant le nœud responsable de la clé k. Les réseaux pair-à-pair pur structurés utilisent les DHT pour indexer les données en associant un $identifiant$ unique à chacune et utilisent le routage pour les rechercher en s'appuyant de leur identifiant. Les systèmes pair-à-pair pur les plus célèbres sont : Chord[SMK+01], Pastry[RD01a], Kademlia [MM02], CAN[RFH+01c].

Pour formaliser ces réseaux structurés, on associé l'espace de clés C à un graphe G=(V,E). Chaque nœud $v \in \zeta$ est responsable d'un espace de clés C_v selon son identifiant, tel que $\bigcup_{v \in V} C_v = \zeta$. Pour chaque clé, un arbre aléatoire $K - aire$ couvrant dans le réseau logique est enraciné au nœud responsable de la clé. Cet arbre permet le routage à partir d'un nœud quelconque dans l'arbre vers la clé, donc vers le nœud qui en est responsable.

L'acheminement de messages dans les réseaux pair-à-pair structurés vers une clé revient donc à router vers le nœud responsable de cette clé. Nous avons dit que les réseaux pair-à-pair structurés, utilisent une topologie connue a priori. L'acheminement de messages de ces systèmes pair-à-pair structurés est ainsi une adaptation du routage de messages dans la topologie de graphe utilisée. Ce routage permet ainsi à chaque nœud recevant un message de décider localement à quel voisin faire suivre le message. Le routage **R** dans un réseau structuré Pair–à-Pair est formalise ainsi [Gau06] : R est une application de **V x** C dans V, avec la contrainte pour un nœud v_0 qui envoie un message que le nœud destinataire $R(v_0, k) = v_i$ est :

— soit le nœud origine v_0
— soit un voisin v_v du nœud v_0 , c'est-à-dire que le message doit être transmis le long de l'arrêt(v_0, v_v).

Modèle pair-à-pair centralisé et Hybride

Schollmeier[Sch01] définit un modèle pair-à-pair hybride :« Une architecture de réseau distribué doit être classée comme un réseau pair-à-pair hybride, si elle est d'abord un réseau pair-à-pair selon la définition donnée précédemment et deuxièmement une entité centrale est nécessaire afin de fournir une partie des services de réseau offerts ».

Comme exemple du modèle pair-à-pair hybride, on peut considérer un groupe de serveurs qui maintient un index de tous les fichiers partagés par les pairs. Lorsqu'un pair veut partager un fichier, il envoie les caractéristiques de ce dernier au serveur auquel il est connecté pour être indexer. La recherche se fait en envoyant une requête au serveur. L'accès au fichier est fait par une connexion directe entre le demandeur et le possesseur. Le système Napster [FP99b] était un modèle pair-à-pair centralisé de partage de fichier. Dans Napster le transfert de fichier se fait uniquement sur une seule source, ce qui peut poser des problème de performance car lorsque l'émetteur tombe en panne, il faut à chaque fois émettre de nouvelles requêtes.

Le protocole Bittorrent[Coh08b] améliore le protocole de transfert en permettant un téléchargement multi-sources. Toujours dans le soucis d'améliorer le *modèle centralisé*, le système FasTrack[LKR06] crée une topologie sans serveurs basée sur une organisation hiérarchique des

pairs.

Après avoir étudié les modèles de communication, nous venons de présenter deux modèles de structuration des applications réparties. Afin de construire des applications, il nous faut maintenant nous intéresser aux différents moyens de stockage et de structuration de l'information.

2.4 La gestion des données dans les applications réparties

Les applications réparties exploitent des données qu'elles vont puiser dans différentes sources. Nous allons présenter les principales solutions de stockage d'informations, et de gestion des accès à ces informations.

2.4.1 Solutions de stockage orienté fichiers

Un NAS[GVM00] est un moyen stockage de données directement relié aux réseaux locaux(Ethernet, TCP/IP). Dans ce système les données qui sont des fichiers sont stockées dans des serveurs et sont partagées par l'ensemble des clients de bureau[Rie03] via le réseau local.

Le NAS fournit un accès aux fichiers en « mode fichiers » de façon transparent où le réseau local entre le client et le serveur est masqué. NAS fournit les protocoles CIFS[Bar97](Common Internet File System), ou NFS[PJS+94] pour accéder aux fichiers. Le protocole NFS utilise un modèle client/serveur et met en œuvre un protocole de réseau standard, qui permet le partage de fichiers pour les clients distants de manière transparente.

Le serveur de fichier de NAS est constitué de plusieurs lecteurs de disque performant. Les multiples disques durs sont organisés en RAID avec des données réparties sur les disques durs enfin d'agréger la capacité de stockage, les performances et la fiabilité[Rie03].

Le stockage et l'accès d'un fichier dans le système NAS est d'écrit dans [Den09] et se font ainsi :

1. quand un client veut stocker un fichier sur le NAS, il appelle un *write*(), qui invoque une demande au NAS via les appels de procédure distante(RPC). Les paramètres et les données seront envoyées vers le NAS comme des messages à travers la pile TCP/IP par le serveur RPC.

   Les messages sont alors transférés vers le serveur RPC du réseau à la mémoire cache du système via l'interface de la carte réseau et le bus de système du NAS. Le serveur RPC décompresse et transmet les messages du protocole de serveur NFS. Le protocole du serveur NFS transmet les données au système de fichiers local de disque (par exemple ext2/ext3) de NAS par le VFS. Les données seront stockées aux disques durs en passant au préalable par le gestionnaire de volume, puis par bus PCI, et le contrôleur de disque, et finalement stockées aux disques durs.
2. lorsqu'un client souhaite accéder à des données stockées dans le serveur NAS, il vérifie tout d'abord la mémoire cache. Si les données se trouvent dans le cache, la requête de lecture sera réalisée et les données correspondantes seront enveloppées dans des messages et ensuite envoyées au client par le serveur RPC via la pile TCP/IP immédiatement.

Le système NAS présente des avantages parmi lesquels on peut citer : sa facilité à mettre en place, son adaptabilité aux partages de fichiers et enfin son extensibilité qui se fait par un

ajout des capacités de stockage sans immobiliser le réseau. Cependant, la présence d'un serveur central pose plusieurs problèmes au système NAS. En effet, l'unicité du serveur constitue un risque qui compromet la disponibilité du service de données. En plus les protocoles utilisés tels que NFS et CIFS, sont inaptes à utiliser toute la puissance processeur/mémoire qui pourrait être fournie par un seul gros serveur, ce qui compromet la performance du service.

Au niveau de l'échelle, le serveur est incapable de traiter un grand débit agrégé à l'ensemble de ses clients (au-delà de 1Go/s). En plus la majorité des fichiers stockés dans le serveur NAS sont caractérisés par leurs petite taille[BHK+91] de quelques Ko, alors que maintenant on peut avoir des fichiers de très grande taille, pouvant atteindre le To.

Le renforcement des besoins en stockage et/ou en performance peut se faire en ajoutant des équipement de stockage de fichiers, offrant ainsi une volumétrie et une augmentation en débit d'accès. Mais les clients vont se retrouver face à plusieurs serveurs, même si les protocoles NFS et CIFS peuvent être utilisés sur des serveurs différents. Dans ce cas le client doit se connecter à plusieurs serveurs de données pour accéder à l'ensemble des données. Cela entraine une administration complexe.

Pour répondre aux questions de performance et de disponibilité du système NAS, certains auteurs proposent le système *NAS en cluster*. C'est un système de stockage à grande échelle qui impliquent de multiples nœuds NAS. Parmi les approches proposées, on citera notamment, RISC[Den08] qui divise les nœuds de stockage en plusieurs partitions pour faciliter la localisation d'accès aux données. Il y'a aussi CoStore[CNY02] qui est un cluster de stockage dit non-serveur, où tous les nœuds sont fonctionnellement symétrique et sans un contrôle centralisé. L'architecture CoStore a le potentiel de fournir une capacité de performance et de stockage évolutive avec une forte fiabilité et la disponibilité. Cependant, il n'est pas vraiment un système sans serveur[WZ08] parce que le répertoire racine du système de fichiers global est hébergé sur seulement une entité.

Le SAN[GVM00] est un réseau à haut débit, dédié au stockage qui relie les serveurs et les composants de stockage, à travers une infrastructure de communication (le plus souvent des commutateurs). On peut dire que le système de stockage SAN est une infrastructure de stockage centralisée. L'avantage de ce réseau est qu'il présente une bonne bande passante et un temps de latence faible, ce qui permet à une architecture SAN d'offrir des avantages tels que la haute performance, la disponibilité, et le partage facile des ressources.

L'infrastructure SAN utilise le protocole d'accès de type bloc pour accéder aux données (c'est le cas d'un disque directement connecté au serveur). C'est à dire un ordinateur qui dispose de son propre espace de stockage disque, connait l'emplacement des informations sur le disque local. Il peut donc accéder directement aux données sans passer par l'intermédiaire d'un fichier. Ce mode d'accès de type bloc est essentiellement utilisé pour les données structurées par les moteurs de base de données.

Le système SAN n'est peu adapté à des réseaux contenant des milliers de nœuds. En plus avec l'augmentation du volume de données actuelle, et leurs diversités font que les systèmes SAN comme NAS, sont inadaptés au stockage. En effet, les protocoles d'accès des systèmes NAS et SAN ne peuvent s'appliquer que sur des données structurées.

Les systèmes de stockages basés sur le réseau tels que NAS et SAN proposent des méthodes robustes de stockage de grande quantité de données et facilitent le contrôle et l'accès aux données. Cependant, avec la croissance soutenue de la demande d'accès client et les tailles de données qui existent actuellement, il est difficile pour les deux systèmes pré-cités de concevoir

un système, à grande échelle dynamique autonome et un système de stockage évolutif qui peut consolider les ressources de stockage distribué pour satisfaire à la fois la bande passante et de stockage de grande capacité[DWH+08]. En plus vient s'ajouter, le nombre de requêtes clients pouvant être traitées simultanément par le serveur, la charge induite sur le réseau, le nombre d'accès disque requis, entrainent un temps de réponse très long.

2.4.2 Le stockage structuré : les bases de données

Dans les cas où les données ne sont pas sous forme de fichiers, les technologies de bases de données telles que : les bases de données relationnelles, distribuées, ou NoSQL représentent des choix technologiques capables d'offrir un stockage à grande échelle. Dans les sections suivantes, nous allons détailler ces différentes technologies.

Base de données relationnelles

Les systèmes de base de données relationnelles(SGBDR) définissent des structures de données permettant de stocker des données dans ces structures, de modifier ces données, et de les consulter à l'aide de requêtes.

Les données sont structurées sous forme de tables. Le langage utiliser pour exécuter des requêtes sur les bases de données est souvent SQL. Le mode d'accès aux données est de type « bloc ». La machine qui héberge la base de données dispose de son propre espace de stockage disque, peut connaître l'emplacement des informations sur le disque local. Elle peut accéder directement aux données, sans passer par l'intermédiaire d'un fichier.

Actuellement la masse de donnée importante et leur diversité, font émerger auprès des SGBDR plusieurs préoccupations[SC11][SC12] : du point vue de la performance et du passage à l'échelle. En plus l'absence de classement possible et de relation entre ces données sont des points importants qui renforcent les difficultés de recourir aux bases de données relationnelle.

D'un point de vu performance aussi, pour une opération simple entrées / sorties qui nécessite une petite quantité de donnée, les SGBDR peuvent déployer des opérations complexes et des requêtes complexes qui ne sont pas nécessaires[SC11].

Certaines applications web sont plus préoccupées par leur évolutivité et la flexibilité de la structure des données que la cohérence. Toujours du point de vue de la performance, le temps de traitement dans les SGBDR est long et complexe à l'échelle. Ceci est du au rangement des données dans les tables. Ces difficultés citées précédemment font que les bases de données relationnelles deviennent inadaptées au stockage des données d'aujourd'hui caractérisées par une volumétrie importante et une diversité dans leurs formats. Notre objectif dans cette thèse consiste à stocker tous les formats de données de façon transparente aux utilisateurs.

Extension des SGBDR aux bases de données distribuées

Une base de données distribuée est définie comme une collection multiple de bases de données qui se trouvent dans des sites différents, logiquement liées et distribuées sur un réseau d'ordinateurs[Kos00]. L'intégration et la gestion des bases de données est assurée par des systèmes qui s'appellent SGBD distribué[CP84].

Ces systèmes s'appuient sur une intégration totale et un contrôle centralisé : toutes les données existantes sont intégrées en une base de données logique unique (la BD distribuée), et désormais gérées d'une manière cohérente par une seule autorité globale de contrôle.

Afin d'assurer la pérénité des données, celles-ci sont stockées sur au moins deux sites et indexées dans des tables a deux dimensions. Chaque table contient un nombre de colonnes définis dés la départ et des lignes identifiées de façon unique par une clé dans une table. L'identifiant peut être un entier, ou une chaine.

Les bases de données distribuées mettent l'accent sur des données structurées, ce qui fait que le langage d'interrogation de ces données est expressif. Cependant, le regroupement de plusieurs bases de données de format de données différents pose un problème d'hétérogénéité de schéma, ce qui peut rendre difficile l'exécution de requêtes de recherche portant sur des données dispersées dans des sites de schéma différent. En effet, les requêtes sont envoyées sur chaque site puis traduites en requêtes locales avant d'être routées sur les sites appropriés (stockant une portion des données manipulées).

Un SGBD distribué est structuré autour d'un serveur central qui localise les sources de données et du traitement des requêtes de façon transparente à l'utilisateur. En stockant les données sur plusieurs sites, la base de données distribuée peut fonctionner même si certains sites tombent en panne. Un second avantage des bases de données distribuées consiste à l'amélioration des temps de réponses des requêtes grâce à la parallélisation des traitements et un accès plus facile et rapide des données.

Ce pendant l'hétérogénéité des types de données appartenant à plusieurs utilisateurs d'un même domaine pose un problème. En effet, l'interrogation d'un ensemble de sources de données hétérogènes est difficile puisqu'une requête exprimée sur un type de données, ne pourra l'être sur un autre type de donnée.

Bases de données NoSQL

La technologie NoSQL est inspirée par les développeurs Web 2.0 et les communautés qui se rendent compte que la base de données relationnelle n'est pas adapté à la gestion en temps réel des sites de réseaux sociaux où les données sont volumineuses et hétérogènes. Cette nouvelle technologie se différencie des bases de données relationnelles : sur les interfaces, et dans la mise en œuvre. Les Bases de données NoSQL sont caractérisées par :

- leur extensibilité horizontale a chaque fois qu'on a besoin d'espace de stockage, il faut juste ajouter un nouveau support de stockage (serveur par exemple)
- une souplesse dans le structure des données. Les moteurs de base de données NoSQL n'utilisent pas de schémas prédéfinis à l'avance. Le NoSQL[HHLD11] regroupe donc de nombreuses bases de données qui ne se reposent donc plus sur une logique de représentation relationnelle. Ce qui fait que les NoSQL englobent le traitement des données qui ne peuvent pas être rêquêtées par SQL, mais également celle qui peuvent l'être.
- En plus la pluspart des NoSQL, privilégient la disponibilité des données et le passage à l'échelle, plutôt que la cohérence des données. Par exemple, NoSQL favorise le retour d'un résultat au lieu d'attendre l'ensembe des résultats.
- son coût faible.

Les auteurs [ABR14] affirment qu'il existe plusieurs technologies NoSQL et chacune présente un modèle de données et des API différentes. Certaines d'entres elles peuvent être déployées et

gérées dans des serveurs locaux, tandis que d'autres offrent leurs services clouds.

Les systèmes NoSQL sont différents les uns des autres sur leur modèle de données, sur les API, et sur leurs objectifs, mais ils ont presque tous une caractéristique commune à indexer leurs données. En effet ils indexent les éléments(données, objets) individuels, identifiés par des clés uniques en utilisant des structures différentes pour indexer les données. En outre ces structures, sont flexibles, en ce sens que les schémas des données sont souvent assouplies ou absents [ABR14].

En général, les systèmes NoSQL utilisent trois fonctionnalités(structures) pour indexer et modéliser les données[Cat11b] : stockage clé-valeur, la mise en œuvre BigTable, stockage de documents, qui sont rares dans la conception de base de données relationnelle traditionnelle[Cat11a].

2.4.3 Stockage orienté document

On peut dire que c'est une base de données (ou de collection) de *documents*, où un document est un ensemble de champs avec des valeurs associées. Chaque document est associé à un identifiant unique, pour l'indexation et pour la récupération lors de la recherche (voir Figure.2.1).

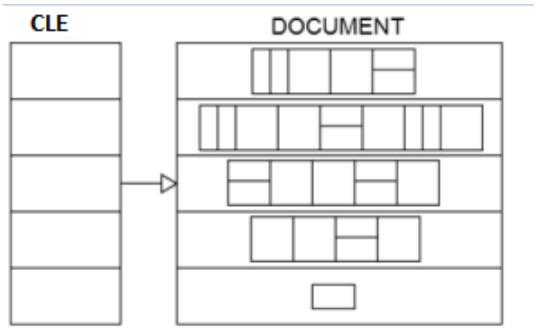


FIGURE 2.1 – Modéle de données Stockage de documents

Une collection contient un nombre illimité de *documents* et n'impose aucune contrainte sur la structure de ces derniers. Chaque document présente une clé *id* qui est unique au sein de la *collection* auquel il appartient. Les collection sont stockées dans une instance d'une base de données, chacune étant identifiée par un *nom*. Un exemple d'une telle base de données de documents est MongoBD[10g13]. Les documents sont identifiés à l'aide de la dénomination de la collection et de ID("id","valeur").

Compte tenu de la souplesse des collections sur la structure des documents, la banque de documents peut stocker n'importe quel type de documents. Ce système est favorable à un index secondaire et peut stocker plusieurs types de documents par base de données[Cat11b].

Cette technique organise les données sous forme de collection de documents au lieu de les structurer sous forme de tables avec des champs de taille uniforme pour chaque enregistrement.

La récupération d'un document au sein d'une collection se fait de façon individuel à l'aide de son identifiant ou bien suivant la configuration des champs du document[ABR14].

Stockage clés-valeurs :

Le système clé-valeur est similaire à un dictionnaire qui contient des données, dont chacune est identifiée par une clé unique. Étant donné que les données(ou valeurs) sont des tableaux d'octets non interprétés, qui sont complètement opaque dans le système, alors les clés sont le seul moyen de récupérer les données[HJ11].

Les valeurs peuvent être des éléments simples tels que les chaînes et les nombres entiers, ou des objets structurés. En ce qui concerne les clés, certains systèmes les considèrent comme des chaînes, tandis que d'autres permettent une structure hiérarchique, composé de plusieurs parties. Même si les clés sont justes des chaînes, ils imitent les structures hiérarchiques.

Le système de stockage clés-valeurs est complétement indépendant de la notion de schéma des données, et même des collections. Les données de toute nature peuvent être ajoutées à l'exécution sans entrer en conflit avec les autres données stockées et sans influencer la disponibilité du système. Le système clés-valeurs est utilisé pour les opérations simples comme l'insertion, la récupération, la suppression qui sont sur la base de critères seulement essentiels (par exemple se référent à des objets individuels et sont spécifiés par l'indication de la clé). Ainsi les opérations couvrant plusieurs objets ne sont pas du tout gérés.

Redis[Car13] est un exemple de système clés-valeurs. Dans Redis les clés sont des chaînes et les valeurs peuvent être de différent natures : chaîne, liste, ensemble triés, une liste hachage représentant des collections ordonnées de chaînes etc..

Les volumes des données qui se comptent en téraoctets et de leurs hétérogénéités sont devenues des difficultés que les bases de données relationnelles n'ont pas été en mesure de gérer. L'objectif principal des bases de données NoSQL est de proposer une alternative aux bases de données relationnelles.

Cependant les auteurs de [SBR+14], listent les limites des techniques d'indexations des données dans les NoSQL. Ces inconvénients sont : un accès lent avec de grands ensembles de données ; une incapacité à utiliser les valeurs obligatoires et les types de données ; la difficulté de l'intégrité référentielle (parce que la valeur d'un attribut est une chaîne sans contrainte comme une clé étrangère) ; la difficulté à assurer de manière cohérente les noms d'attributs ; et les requêtes complexes qui sont souvent nécessaires pour travailler avec les données parce que les fonctions intégrées de la base de données ne sont plus en mesure de comprendre les données, en déplaçant le fardeau de travail du moteur de base de données vers pour le développeur.

Selon les études de [Sto11], il y'a plusieurs bases de données NoSQL qui ont été mises en œuvre. Chacune de ces bases de données présentent des modèles de données différentes, des API différentes pour accéder aux données. Elles sont aussi différentes sur la gestion de la cohérence des données et aussi de la manière de garantir la durabilité des données.

Cette absence de norme standard pour accéder aux données (API) est problématique. Quelques systèmes SOS[ABR12] et ONDM[Cab13] ont été proposés pour cacher l'hétérogénéité des bases de données NoSQL et fournir un accès uniforme aux sources de données NoSQL.

Le Framework SOS[ABR12], est une plate-forme qui fournit une interface uniforme vers des systèmes NoSQL. SOS dispose des primitives tels que le stockage, la récupération, la suppression et la modification de données, inspirées de celles offerts par les bases de données NoSQL. Le système SOS est basé sur une approche de méta modélisation, en ce sens que les interfaces spécifiques des systèmes NoSQL sont mises en correspondance et avec en général en une commune.

SOS fournit une interopérabilité avec les différentes base de données NoSQL, permettant ainsi aux clients d'avoir une vue transparente des sources de données et un accès simple à travers la même interface API. Ainsi depuis une seule application on peut interagir avec plusieurs systèmes en même temps.

Le maintien d'une structure de données homogène est important, pour des questions d'indexation, de recherche par critère ou tout simplement pour des raisons de logique. L'accès aux données dans les NoSQL requiert toujours une entité pseudo centralisée capable d'agréger et de présenter les données. Cela n'exclut pas le traitement parallèle des requêtes.

2.5 Données et systèmes pair-à-pair(P2P)

Les systèmes pair-à-pair offrent des ressources potentiellement illimitées pour le stockage de grande quantité de données, grâce à l'association des ressources des pairs, et des mécanismes plus ou moins efficaces pour la recherche. Ces données concernent celles qui ont une représentation sous la forme de simple fichier, mais aussi des *flux*, des services (par exemple la téléphonie sur Skype) via le réseau Internet. Ces systèmes ont la particularité de pouvoir interconnecter des millions d'utilisateurs dans le but de leur faire partager une quantité importante de données.

Nous allons maintenant présenter différents systèmes pair-à-pair ainsi que leur manière de gérer les données.

2.5.1 Napster

Napster[FP99b] fait partie de la première génération de réseaux pair-à-pair apparue en 1999. Il a été crée pour partager des fichiers audio de format *mp*3 dans laquelle chaque pair pouvait mettre à disposition le contenu de sa bibliothèque musicale.

Dans cette architecture chaque pair qui arrive dans le réseau s'enregistre sur un répertoire centralisé en envoyant la liste de ses fichiers et son adresse IP. L'indexation des données dans le répertoire centralisé est faite en faisant une correspondance entre l'adresse du pair, les données et les méta-données.

La recherche dans Napster est très simple. Lorsqu'un client cherche un fichier, il passe une requête en inscrivant un mot-clé(contenant les caractéristiques du fichier recherché) au serveur centralisé auquel il est connecté. Le serveur central consulte son index et répond alors par une liste de nœuds(ip, fichier, metadonnées) actuellement connectés au service et dont les fichiers partagés correspondent au terme recherché. Dés lors, le client se connecte sur un des nœuds de la liste retournée pour entamer le transfert. La connexion s'arrête à la fin du transfert. Cette procédure de recherche dans Napster est schématisée dans la figure 2.2.

Cette architecture propose un mécanisme rapide de traitement des requêtes lors d'une recherche de fichier. En effet, la requête va directement vers le serveur central, en un saut, et ce serveur répond en un saut. Un simple va et vient vers l'index central est ainsi suffisant pour localiser un fichier. En plus Napster est résistant face à la mobilité des nœuds dans le réseau. En effet, les nouveaux nœuds, à leur arrivée, n'ont donc qu'à se connecter au serveur central, stable et connu, pour être insérés dans le système.

Á cause de son index central, le système Napster reste vulnérable aux pannes du serveur central.En effet, la panne du serveur central entraine l'expulsion du réseau de tous les nœuds

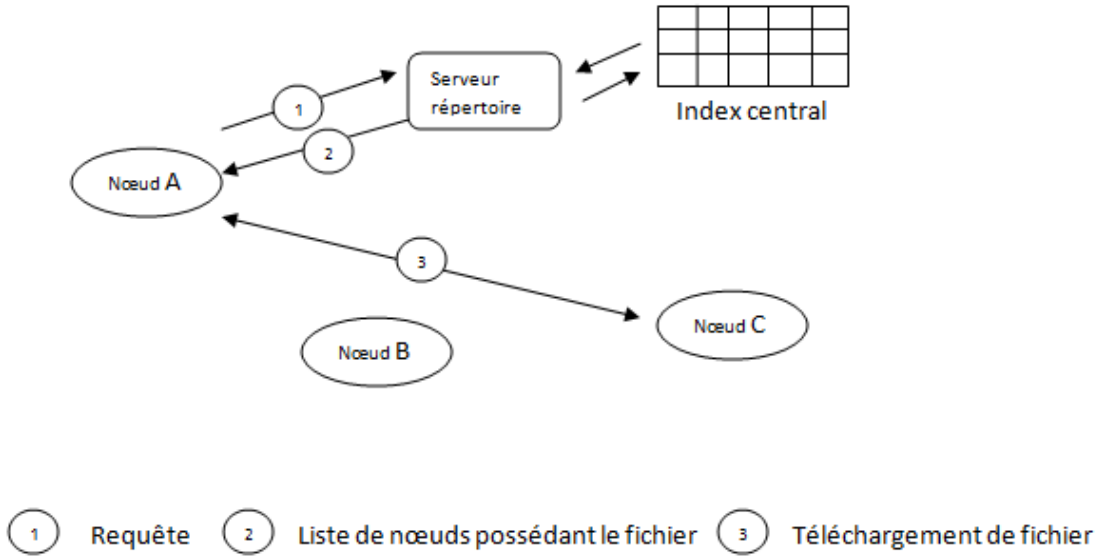


FIGURE 2.2 – Organisation de Napster

jusqu'alors connectés et entraine l'arrêt total de ce système.

Le système Napster montre ses limites dans la protection des données des utilisateurs. En effet, le problème d'anonymat se pose partout car chaque nœud est connu du serveur et des pairs sur lesquels il télécharge.

De plus, l'indexation centralisée des données est contraire à l'esprit des réseaux pair-à-pair, qui veut que la gestion des données soit décentralisée. Pour ce faire l'idée qui consiste à séparer la gestion de l'index de celle des données a été explorée et elle a conduit à la mise en place des systèmes avec des « super-nodes ».

2.5.2 Bittorrent

Bit torrent[Coh03] est un protocole pair-à-pair de transfert de fichiers, crée par Bram Cohen. Son but principal est uniquement la distribution de fichier et non pas la localisation de ce dernier.

La structure de Bittorrent est constituée de deux entités : (i) le *tracker* qui sert de serveur central qui regroupe les pairs intéressés par la distribution d'un seul fichier.(ii) le *torrent*, est un fichier qui contient les méta-données des fichiers que les utilisateurs souhaitent récupérer, ainsi l'adresse des *trackers* qui l'hébergent. Un fichier est mis à la disposition des utilisateurs par son propriétaire, lorsque le fichier *.torrent* est mis sur un serveur web ou sur une simple page web. Dans Bittorrent les fichiers sont découpés en un certains nombres de blocs et chaque bloc peut être téléchargé d'une source différente. Cette technique, connue sous le nim de *stripping*, a l'avantage de permettre le téléchargement parralèle et la réduction du volume global sur une seule machine. Son inconvénient est que des fichiers peuvent se retrouver incomplets car des morceaux manquent.

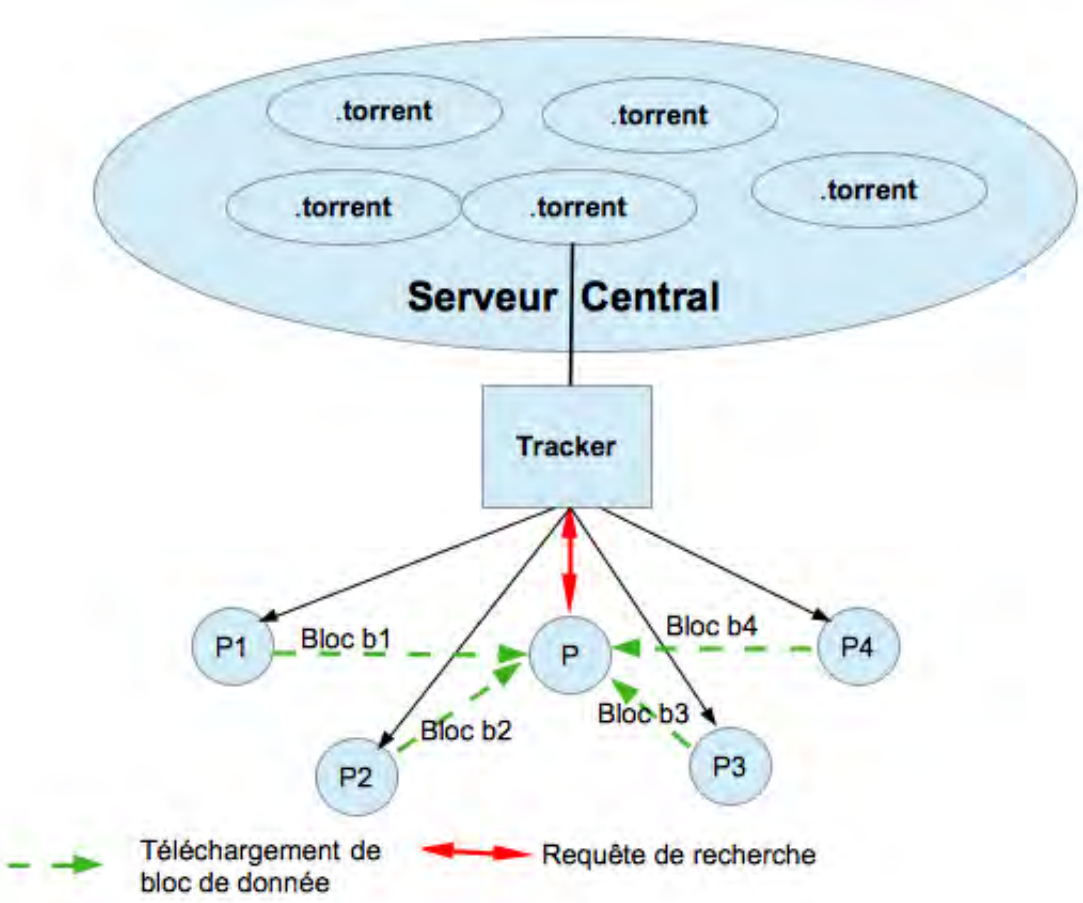


FIGURE 2.3 – Organisation de Bittorrent

Lorsqu'un pair p souhaite télécharger une donnée d, il récupère le fichier *.torrent* et se connecte au *tracker*. Ce dernier répond par une liste d'autres pairs p_i possédant le même fichier. Et le pair p peut maintenant télécharger en parallèle les blocs de données, en se connectant aux pairs p_i qui les possèdent. L'illustration d'une recherche d'un fichier dans Bittorrent est donnée dans la figure 2.3. Bittorrent permet un accès rapide des données grâce aux téléchargements multi-sources.

La présence et le rôle capital de l'entité *tracker*, fait de Bittorrent un réseau centralisé comme Napster. De ce fait Bittorrent partage les mêmes inconvénients que ce dernier : la panne du serveur central. En effet, la panne du *tracker*, ne permet plus de regrouper les pairs inscrits à ses *torrents*.

Des améliorations ont été apportés dans le protocole de Bittorrent[WH10] avec l'utilisation de Table de Hachage Distribuées(DHT) qui jouent le rôle de *trackers* décentralisés. Ces DHT sont utilisées pour la continuité du service lorsque le *tracker* tombe en panne. En plus les auteurs de CBT [YL08] proposent une architecture hiérarchique, pour améliorer les performances de Bittorrent en introduisant les « Super node » qui vont décharger le *tracker* du trafic réseau. Bittorrent est utilisé par plusieurs sites de distribution de contenu films, Série TV, musique ...

2.5.3 FastTrack et KaZaA

FastTrack[LKR06] est un protocole propriétaire crée en 2001 et implémenté dans les clients KaZaA[LKR], Morpheus, Grokster. Il présente une architecture hiérarchique de deux niveaux suivant le rôle des ses nœuds.

Au premier niveau on retrouve certains nœuds appelés *Super Node ou SN* qui supportent la charge d'indexation des autres nœuds qui se trouvent au deuxième niveau appelés *Nœud Ordinaire ou NO*. La structure de FastTrack est constituée d'une interconnexion de nœuds SN au premier niveau auxquels sont attachés les nœuds NO. Dans cette architecture les nœuds qui présentent plus de bande passante, plus de connexion, de mémoire etc., sont élus nœuds SN et reçoivent les requêtes de recherche des clients (NO) et assurent l'indexation de leurs contenu.

Lorsqu'un nœud NO rejoint le réseau FastTrack, il contact un SN et lui envoie ses données avec leurs méta données à indexer. Les méta-données contiennent notamment le code de hachage(hachage SHA-1). Donc les SN de FastTrack indexent le nom du fichier + son code Hache.

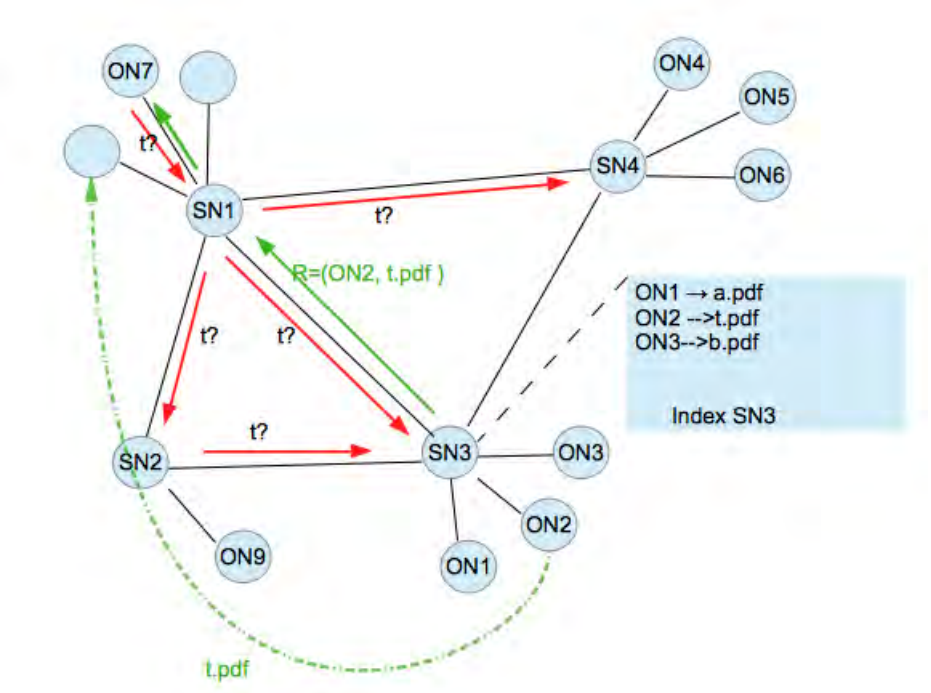


FIGURE 2.4 – La recherche dans FastTrack

Lorsqu'un nœud ON_i cherche une donnée, il adresse une requête de recherche à son SN. Celui-ci publie cette requête par inondation à tous les SN du réseaux d'interconnexion aux quels il est actuellement relié. Cette procédure continue jusqu'à trouver le SN qui a indexé la donnée recherchée. Une fois trouvé, la réponse permet et le transfert pair-à-pair entre les deux NO peut commencer.

Cette recherche est illustré dans la figure 2.4, où le nœud ON_7 cherche une donnée transmet la requête a son SN_1. Ce dernier envoie la requête par *inondation* à tous ses voisins SN via le réseau d'interconnexion qui les relie jusqu'à ce que le nœud SN qui à indexé la donnée soit

trouvée (ici le nœud SN. Le nœud SN_3 envoie la réponse qui contient l'identifiant du nœud NO qui héberge la donnée recherchée par le nœud NO_7 en suivant le chemin inverse à l'aller. Le nœud NO_7 peut maintenant entrer directement en contact avec le NO_2 pour le rapatriement.

Notons également qu'un champ TTL est introduit dans la requête de recherche ce qui permet de limiter l'inondation. En plus chaque SN connait toutes les données contenus dans les NO qui sont connectés à lui. De ce fait, un SN sait lorsqu'il reçoit une requête, s'il doit la rediriger vers un des nœuds reliés.

2.5.4 Gnutella

Gnutella[Rip01] est un protocole de partage de fichier qui a été développé en mars 2000 par Justin Frankel et Tom Pepper. Gnutella v0.4 est le premier réseau pair -à-pair décentralisé non-structuré dans lequel les pairs ont le même rôle et indexent leur contenu. C'est à dire chaque nœud est à la fois client, serveur et routeur. Le protocole de Gnutella est constitué et de 4 types de messages (*Ping*, *Pong*, *Query*, *Query hit*) émises par les servents(serveurs + client) à travers le réseau Gnutella.

Pour entrer dans le réseau de Gnutella, un client doit connaitre au moins un nœud déjà présent sur le réseau(servent), et à partir de cette connexion, il peut, par le moyen de messages *ping*, trouver les adresses d'autres servents.

Un servent qui reçoit une requête *ping* de connexion doit répondre avec avec un (ou plusieurs) *pong*, indiquant son adresse IP et son numéro de port, ainsi que le nombre et la quantité de données partagées, et ajoute le nœud initiateur à ses voisins, ce qui permet de créer le réseau logique. La taille des voisinage dans Gnutella n'est ni spécifiée ni bornée et le réseau logique résultant ne présente aucune structure caractéristique.

La recherche dans Gnutella s'effectue par inondation. Elle suit les étapes suivants : lorsqu'un client fait une requête de recherche de fichier, celle-ci est transmise à tous les servents auxquels l'émetteur est connecté. Ces derniers retransmettent la requête par inondation de nœuds en nœuds jusqu'à ce qu'elle atteigne un nœud possédant la ressource.

Lorsque la ressource est trouvée, son adresse est alors retournée au demandeur en suivant le long du chemin inverse. Le client n'aura plus qu'à télécharger le fichier à la source. Pour éviter une inondation et une saturation du réseau, chaque requêtes possède une durée de vie TTL (Time To Live), qui indique le nombre de fois que l'on peut retransmettre la requête. Un exemple de réseau Gnutella v0.4 est présenté dans la figure 2.5.

Lors du téléchargement, si la source de données est derrière un firewall, alors on utilise le mécanisme *push*. Avec Gnutella, le téléchargement est impossible si le client et la source sont tous derrières des firewalls.

La recherche dans Gnutella ne passe pas à l'échelle. En effet, le réseau est rapidement inondé par des messages requête qui croît exponentiellement avec la taille du réseau [CRB+03]. En plus le fait de fixer le champ TTL cause un manque de fiabilité dans les requêtes. En effet, celles-ci ne permettent pas d'atteindre la totalité du réseau surtout lorsque le diamètre du réseau est grand, ce qui va conduire à des résultats qui ne sont pas exacts.

Dans Gnutella, il n'y a pas de mise en commun de support de stockage des nœuds, puis les données sont directement accédées chez le propriétaire. Ce mécanisme de partage de données manque une garantie de pérennité des données, puisque la durée de vie de ces données est celle du nœud propriétaire.

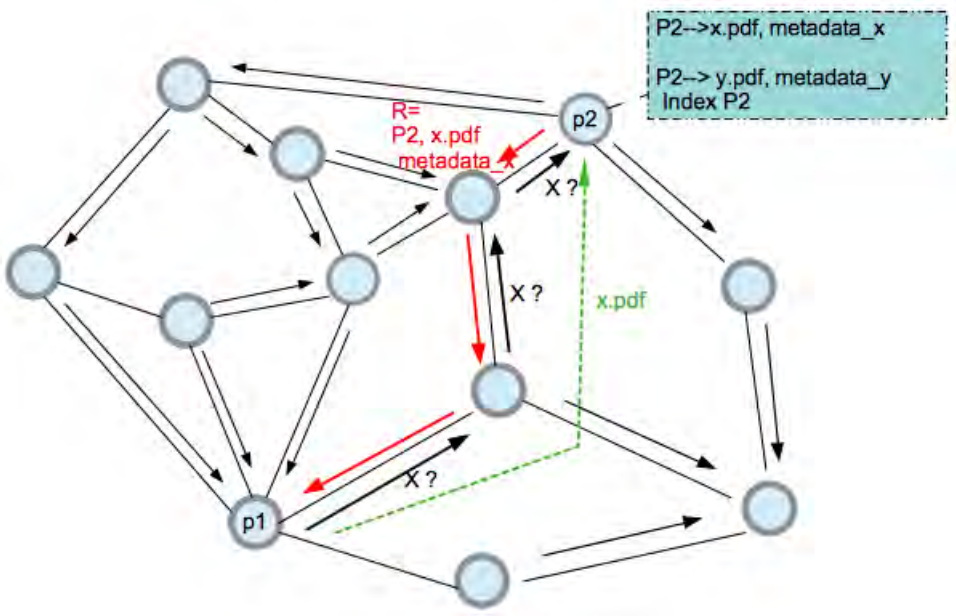


FIGURE 2.5 – Architecture de Gnutella v0.4

Pour améliorer le protocole de recherche de Gnutella, des mécanismes ont été proposés tels que : (*i*) remplacer l'inondation par des envois aléatoires effectués en parallèle, ce qui va diminuer le nombre de messages. (*ii*) L'utilisation des « Super serveurs » (nœud ayant beaucoup de connexions et ayant une puissance importante), qui se voient affecter des taches tels que le routage de requêtes.

Ces « Super serveurs » vont former un réseau entre eux et acceptent la connexion de nœuds simples et deviennent leurs points d'accès au réseau. Les « Super serveurs » assurent l'indexation des données et le traitement des requêtes(Figure 2.6). Le réseau pair-à-pair tel Gnutella 0.6[Kli02] utilise cette approche.

2.5.5 Freenet

Freenet[CSWH01], est un réseau pair-à-pair dont la structure est totalement décentralisée. Les nœuds du réseau possèdent une table de routage dynamique contrairement à Gnutella. Les entrées de cette table de routage associent un identifiant avec un nœud. le principe de stockage dans Freenet est qu'un fichier est stocké en plusieurs exemplaire, à des emplacements déterminés par sa clé. Il existe deux types de clés dans Freenet :

- clés de bas niveau(Content Hash Key : CHK), utilisées pour stocker les fichiers. Elle est obtenue par le Hash-code sur le contenu du fichier,
- clés de haut niveau(Signed Subspace Key : SSK), utilisées pour stocker des descripteurs (listes de CHK). Une SSK est obtenue à partir de de descriptifs des fichiers à rechercher exemple : mots-clés.

Le stockage d'un fichier dans Freenet se fait par palier en suivant les étapes suivantes :

1. Le propriétaire du fichier envoie à un nœud la clé(CHK ou SSK) et un TTL(nombre maximum de sauts, également nombre maximum de copies).

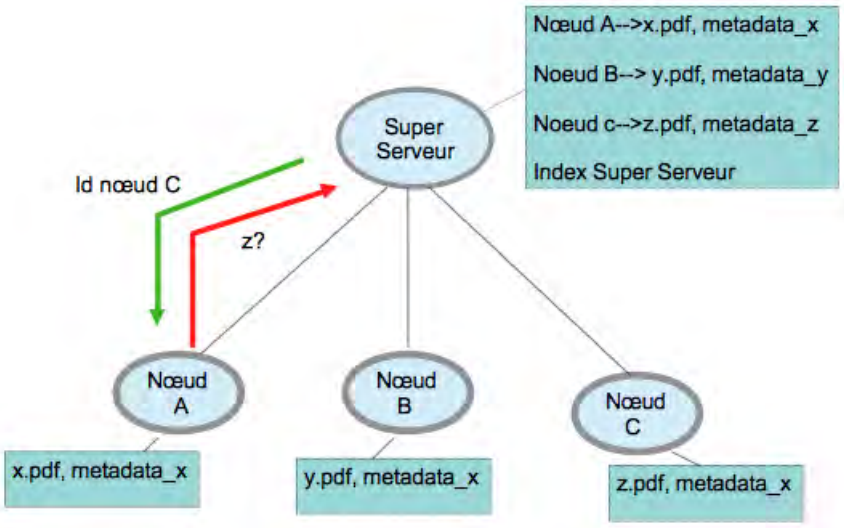


FIGURE 2.6 – Architecture de Gnutella0.6

2. Le nœud vérifie si la clé existe déjà, si oui, alors il y'a collision (le fichier ou le descripteur est déjà présent dans le réseau). Si non, le nœud retransmet au nœud possédant la clé la plus proche dans la table de routage. Si le TTL est atteint sans collision, un message Ok est renvoyé, si ce message Ok est reçu par le propriétaire, alors il envoie le fichier, qui est stocké sur tous les nœuds du parcours précédent.

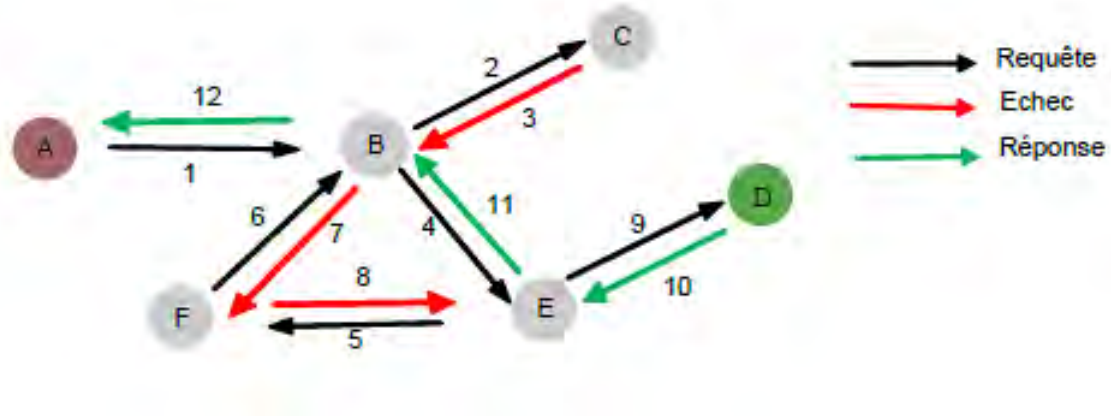


FIGURE 2.7 – Recherche de donnée dans Freenet inspiré de [CSWH01]

La recherche d'un fichier dans Freenet(voir figure2.7) se fait à partir de sa clé(toute ressource est identifiable à partir d'une clé unique), de manière totalement distribuée. Ainsi lorsqu'un utilisateur recherche un fichier : il envoie la clé et un TTL à un nœud du réseau, si la clé est présente dans sa table de routage, alors il renvoie le fichier demandé.

Si la clé n'est pas présente, le nœud transmet la clé à celui possédant l'identifiant du fichier la plus proche dans la table de routage. On continue ce procédé jusqu'à ce que le fichier soit trouvé ou bien le paramètre TTL soit atteint. Lorsqu'une boucle dans le routage est détectée ou qu'il n'y a plus successeurs disponibles sur une branche de l'arbre, alors on revient en arrière et on repart d'un autre voisin.

Au niveau du stockage de donnée l'architecture Freenet est limité par la capacité de stockage finie des ses nœuds. En effet lorsqu'un fichier arrive pour être stocké est qu'il n'existe plus d'espace de stockage dans le nœud, alors les fichiers peu demandé sera supprimé pour laisser la place au nouveau fichier, ce qui fait que Freenet ne garantit pas la pérennité ses données.

Au niveau de la recherche de données Freenet est plus robuste que Gnutella, utilisant le routage à la place de l'inondation. Cependant le temps de recherche dans Freenet peut être long, ce qui oblige Freenet à être utilisé dans une communauté restreinte d'utilisateurs, ce qui limite le passage à l'échelle de ce réseau.

2.6 Réseaux structurés décentralisés

Dans les solutions pair-à-pair étudiées dans les sections précédentes 2.5.5 et 2.5.4, le stockage de données n'est pas leur premier objectif. En plus leurs systèmes de recherche de données ne passent pas à l'échelle à cause de l'inondation pour Gnutella et d'un temps long lors d'une recherche dans Freenet. Pour améliorer ces points des réseaux pair-à-pair décentralisés structurés ont été proposés et s'appuient sur des Tables de Hachage Distribuées(DHT).

Les DHT permettent de réaliser d'une part le stockage d'une donnée dans le réseau à l'emplacement de son identifiant, et d'autre part la récupération d'une donnée dans le réseau en connaissant son identifiant. Une DHT est une structure qui associe une clé à une ressource. Chaque clé de la table est le résultat d'une fonction de Hachage appliquée à un élément de la ressource, par exemple son nom ou son contenu.

La fonction de hachage ne garantit pas que pour deux ressources différentes les clés qui seront générées le seront aussi. Un mécanisme est ajouté au hachage pour éviter ce problème de collision. L'unicité de la clé permet d'identifier et de retrouver de manière fiable la ressource à la quelle elle est associée. Une clé dans les DHT est générée par les fonctions de hachage telle que SHA-1[**?**].

Les DHT sont caractérisées par : l'auto-organisation, le passage à l'échelle, et la résistance à la mobilité des nœuds du réseau.

— L'auto-organisation ;
— Le passage à l'échelle ;
— Résistance à la mobilité des nœuds.

L'emplacement du stockage des fichiers est entièrement déterminé par une seule opération qu'est la clé(fonction : cle $->$ nœud). Nous citons quelques exemples de réseaux DHT : Chord[SMK+01], CAN[RFH+01c], Kademlia[MM02] et Pastry[RD01a], que nous étudierons dans les différentes sections qui suivent.

2.6.1 Chord

Chord[SMK+01] est une infrastructure de stockage et de routage. Les clés et les nœuds reçoivent un identificateur de m bits. L'identificateur d'un nœuds « nodeID » est obtenu en appliquant la fonction de hachage sur son adresse IP. La clé d'une ressource par exemple un fichier « key(fichier) » est obtenue en appliquant la fonction de hachage sur son contenu. L'espace des identifiants « IDs » est organisé en anneau et ordonné en modulo 2^m.

Le principe consiste à stocker, de façon totalement distribuée, l'information (« contenu fichier », *adresseIP*) pour chaque ressource, qui associe au nom d'une ressource l'adresse IP du nœud qui possède la ressource. Le stockage d'un fichier dans Chord se fait ainsi : un fichier de clé k est stocké au premier nœud A dont l'identifiant est égal ou supérieur à celui de la clé k(nodeID(A) supérieure à k mod 2^m).

La recherche d'une clé dans Chord peut se faire de manière séquentielle. En effet, dans Chord chaque nœud doit connaitre son successeur. Étant donné une clé, on parcourt la suite des nœuds jusqu'à trouver une paire de nœuds qui encadre la clé. Le dernier est le nœud cherché (voir figure 2.8). Cette méthode de recherche est simple mais, peu rapide surtout dans réseaux de grande taille où on risque de parcourir tous les nœuds du réseau.

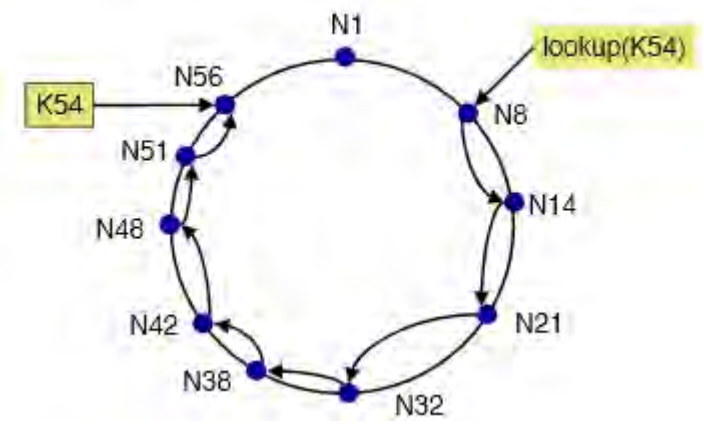


FIGURE 2.8 – Recherche sequentielle dans Chord

Pour accélérer la recherche dans Chord, on fait maintenir dans chaque nœud de l'anneau Chord une *table de voisinage* appelé $finger$. Cette table de voisinage permet de faire des sauts dans l'espace de recherche guidés par la distance. Le voisinage de k est le premier nœud qui suit$(n + 2^(k-1))$ mod 2^m, $1 \leq k \leq m$. Le principe de cette recherche rapide dans Chord est : (i) lorsqu'on veut récupérer une clé k, on regarde si la clé est entre le nóeud de départ et son successeur sinon, on cherche dans la table des voisinages le nœud le plus proche précédant la clé. (ii) On réitère cette procédure de recherche à partir de ce nœud, jusqu'à trouver le prédécesseur absolu de k.(voir figure 2.9).

Les performances significatives de Chord sont évaluées ainsi : (i) lors d'une recherche, le nombre de nœuds à interroger pour la recherche du nœud successeur d'une clé dans un réseau de N nœuds est $O(logN)$. (ii) Les entrées des tables de voisinages sont en $O(logN)$. Le nombre moyen de messages pour réaliser une recherche est de l'ordre de $1/2 * O(logN)$.

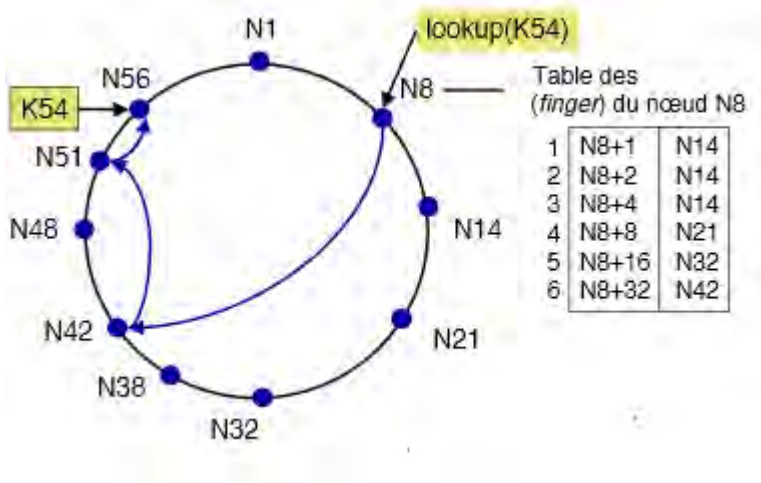


FIGURE 2.9 – Recherche avec les tables de voisinage de chaque Nœud

Cependant Chord utilise la notion du *successeur le plus proche* pour envoyer un message ce qui risque un parcourt de tout l'espace d'adressage de l'anneau au pire des cas. En plus Chord un routage rigide, il ne connait pas la notion de *successeur*, ce qui fait qu'il n'est très adaptées à un environnement très dynamique. Ceci constitue une limite de Chord.

La DHT Chord est utilisée dans l'application de gestion de données comme CFS(Coopération File System)[DKK+01] qui est un système de stockage de fichiers distribué.

2.6.2 Pastry

Pastry[RD01a] est une infrastructure de stockage et de routage. Il utilise les mêmes principes de base que Chord(voir section **??**). Les identifiants des nœuds « nodeID » sont choisis aléatoirement et codés sur 128 bits, alors que les identifiants des fichiers « Key(fichier) » sont obtenus en hachant leur contenu grâce à la fonction SHA-1 sur 160 bits.

Un fichier d'identifiant k(ou sa clé) est stocké sur le premier nœud ayant l'identifiant immédiatement supérieur à K(ou bien le nœud ayant l'identifiant le plus proche numériquement).

Le routage dans Pastry est plus efficace que celui de Chord. En effet, chaque nœud Pastry dispose d'une table de routage « incrémentale » et des voisins $logiques(leafset)$ et des voisins $physique$(voir figure 2.10). Le $leafset$, contient L voisins dans l'espace logique ; et sert à donner au nœud une vision précise de sa localité. La table de routage, est une table de préfixe plus précise pour les nœuds proches dans l'espace virtuel .

Le principe de routage de Pastry consiste à propager le message vers un nœud partageant un plus grand préfixe commun avec la ressource.

La recherche d'une ressource d'identifiant ID_k se fait par palier en suivant les étapes suivantes : (i) le nœud n vérifie d'abord si la ressource ID_k est dans ses voisins logiques. Si oui, alors n transmet directement ID_k au nœud responsable. (ii) Sinon, n cherche dans sa table de routage un nœud partageant un préfixe plus long avec ID_k pour lui transmettre le message.(iii) Si n ne trouve pas de nœud satisfaisant, alors il transmet le message à un nœud partageant un préfixe de même taille que lui avec ID_k, mais numériquement plus proche.

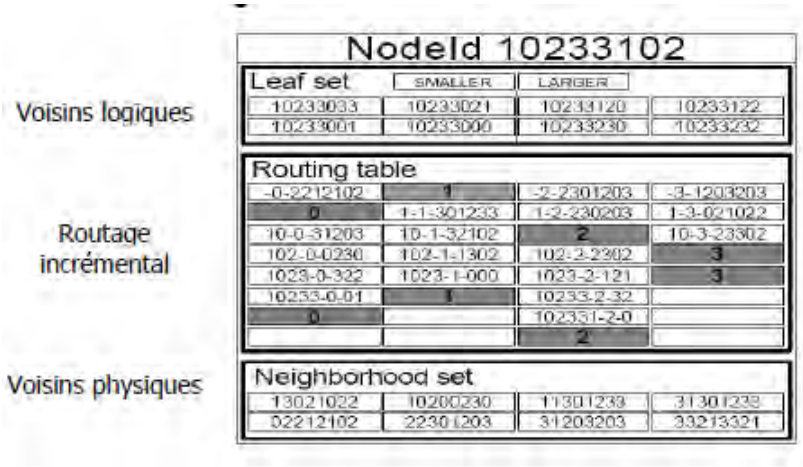


FIGURE 2.10 – Table routage des Nœuds Pastry extrait dans [RD01a]

Un exemple de routage dans Pastry est illustré dans la figure 2.11. Le nœud 322 envoie un message a la ressource 2106. 322 résout d'abord le premier chiffre et envoie a 231 en utilisant son entrée $2x$, 231 résout le deuxième chiffre et envoie à 211 en utilisant $21x$, enfin, 211 résout le dernier chiffre et transmet à 210 en utilisant $210x$ (dans Pastry, les identifiants de ressources sont plus longs que les identifiants de nœuds).

Les performances de Pastry peuvent être résumés ainsi : une structure totalement distribuée avec une répartition équitable des charges entre les nœuds. Pastry présente un système de routage efficace(en terme de messages) à cause de la présence de $leafset$ dans chaque nœud qui garantit la progression des requêtes de recherche vers la ressource. La complexité de l'algorithme de recherche dans Pastry est $Olog(N)$ où N est le nombre de nœud.

Le système Pastry peut être adapté pour des applications pair-à-pair tel que le stockage par exemple PAST[DR01] grâce a son algorithme de routage efficace et résistant aux pannes des nœuds, pour chercher les données.

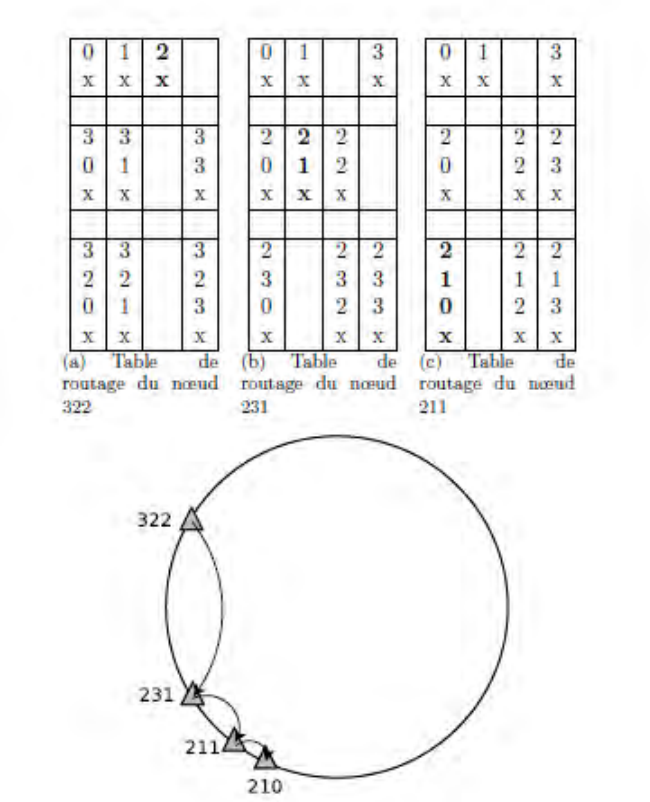
Dans PAST, une clé k est enregistrée sur n nœuds d'identifiants les plus proches, oú n est un paramètre du système assurant qu'un message routé vers une clé trouve l'un des n plus proche nœud de cette clé.

2.6.3 Kademlia

Kademlia[MM02], est un protocole P2P qui se base sur les DHT. L'identifiant des ressources(key) et des nœuds($nodeID$) sont codés sur 160 bits définissant leur place dans l'espace d'adressage logique du système.

Ces identifiants sont utilisés par Kademlia pour rechercher des données indexées. Kademlia dispose d'une fonction symétrique XOR, qui permet d'évaluer la distance d entre deux nœuds A et B sur l'espace d'adressage du système. Ainsi d est égale à A XOR B ou B XOR A. L'avantage de cette fonction métrique permet à Kademlia d'avoir un protocole de routage plus souple que Chord.

Le stockage d'une donnée se fait sur les n nœuds qui ont les identifiants les plus proches de key selon la métrique XOR.



(a) Table de routage du nœud 322

0 x	1 x	**2 x**	
3 0 x	3 1 x		3 3 x
3 2 0 x	3 2 1 x		3 2 3 x

(b) Table de routage du nœud 231

0 x	1 x		3 x
2 0 x	**2 1 x**	2 2 x	
2 3 0 x		2 3 2 x	2 3 3 x

(c) Table de routage du nœud 211

0 x	1 x		3 x
2 0 x		2 2 x	2 2 3 x
2 1 0 x		2 1 2 x	2 1 3 x

FIGURE 2.11 – Acheminement des messages dans Pastry

Le principe de recherche d'une donnée consiste à trouver un nóeud dont l'identifiant($nodeID$) est proche de key. Pour cela, chaque nœud n va diviser l'arbre en une série de sous-arbres ne le contenant pas. Ces subdivisions de l'arbre sont appelées des *buckets*. Figure 2.12 montre un exemple de représentation en arbre qu'un pair d'identifiant 0011 se fait du réseau. Les zones en cercle représentent les *buckets*.

Ces zones sont les tables de voisinage d'un pair qui maintient pour chaque *bucket* un ensemble de pairs connectés dans cette zone. Dans l'exemple (voir Figure 2.12) du nœud 0011, les sous-arbres encerclés sont composés de tous les nœuds dont les préfixes sont respectivement 1, 01, 000 et 0010.

La recherche d'une clé key à partir d'un nœud n d'identifiant $nodeID$ se fait par routage.

Le principe d'indexation des ressources se fait ainsi : chaque nœud est responsable d'une liste d'informations(liste de couple < key, ressource> de telle sorte que les identifiants key soient proches de son $nodeID$. La fonction symétrique XOR, permet d'évaluer la distance d entre deux nœuds (A et B) sur l'espace d'adressage du système. Ainsi d est égale à A XOR B ou B XOR A. L'avantage de cette fonction XOR permet Kademlia, de maintenir un routage efficace par rapport à Chord. En effet, Kademlia peut contacter n'importe quel nœud à l'intérieur d'un intervalle respectant une certaine distance XOR, lá ou Chord utilise la notion de plus successeur.

Le protocole Kademlia fournit quatre primitives : *PING STORE*, *FIND_NODE*, et *FIND_VALUE*.

— PING : Teste la connectivité d'un nœud dans le réseau ;

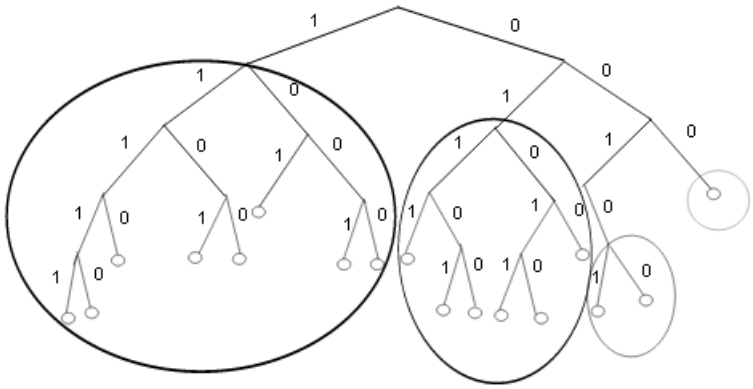


FIGURE 2.12 – Arbre binaire Kademlia

— STORE : permet de stocker une donnée sur les n nœuds dont les identifiants trouvés sont plus proches de l'identifiant de la donnée *key* selon la métrique XOR ;
— *FIND_ NODES :* retourne l'identifiant $nodeID$ des nœuds connus pour être les plus proches de l'identifiant ciblé ;
— *FIND_ VALUES :* retourne la position d'une valeur.

Le protocole Kademlia est incorporé dans plusieurs applications pair-à-pair de partage de fichier, notamment Kad[SENB07] qui utilise Kademlia pour indexer ses données. Il y'a aussi le réseau eDonkey[HBMS04a] et BitTorrent qui utilisent la DHT Kademlia pour faire face aux problèmes liés à leurs éléments centralisés.

Le protocole Kademlia a connu un sucés populaire grâce à son routage souple par rapport à celui de Chord qui est rigide. En plus grace à son routage efficace et simple à mettre en œuvre. La complexité de la recherche dans Kademlia est de $O(log(N))$ avec N, le nombre total de nœuds dans le réseau.

2.6.4 CAN

Le système CAN(Content Addressable Network)[RFH$^+$01b] est un *overlay* qui réalise une DHT. Un réseau *overlay* forme une topologie de réseau logique à la couche application qui n'est pas liée à la couche de la topologie du réseau.

La structure logique de CAN est un espace cartésien de d dimensions basé sur des coordonnées virtuelles. Cette structure logique est divisée en zones qui sont administrées par les nœuds participant dans CAN. Chaque nœud gère une zone distincte au sein de l'espace global. Ce dernier sera responsable du stockage et de la gestion de toutes les données correspondantes à sa zone. La figure2.13 montre un espace de coordonnées à deux dimensions qui est divisé en 10 zones. Les numéros dans l'espace de coordonnées indiquent quel nœud gère quelle zone.

Les opérations de base exécutées par CAN sont l'insertion, la recherche et suppression du couple(clé, valeur). Les nœuds de CAN stockent les couples *(clé, valeur)*. Le réseau *overlay* est utilisé pour faire stocker les couples en faisant un mappage de la clé *key* sur un point dans l'espace de coordonnées.

Pour stocker une donnée dans CAN, on calcul d'abord sa clé *key* par une fonction de hachage, qui fournit en sortie une valeur de hachage de longueur fixe. Cette valeur de hachage

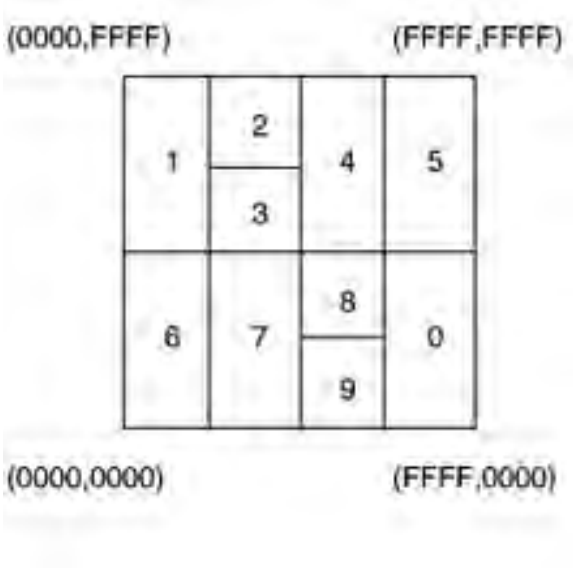


FIGURE 2.13 – Organisation de l'overlay CAN

est en fait les coordonnées d'un point dans l'espace de d dimensions. Le tuple *clé, valeur* est alors stocké sur le nœud qui gère la zone dans laquelle le point est situé.

Pour récupérer un couple de clé k et une valeur V, un nœud peut appliquer la même fonction de hachage déterministe sur K pour mapper K sur un point P(x,y) dans l'espace de coordonnées et ensuite récupérer le couple de la zone dans laquelle P est situé. avec x=h(K_1) et y=h(k_2).

La recherche d'une clé est basée sur un algorithme de routage glouton. Chaque message de requête inclut le point de destination comme adresse de destination souhaitée. Chaque nœud maintient une liste de voisins qui entourent sa propre zone. Cette liste de voisins agit en tant que table de routage et est utilisée pour les données d'acheminement.

Lorsqu'on veut faire une recherche dans CAN, la requête est acheminée à travers le réseau logique jusqu'à ce qu'elle atteigne le nœud qui gère la zone P. Les messages sont routés dans la topologie du réseau *overlay* en transmettant aux voisins dont les coordonnées sont les plus proches de la destination.

L'ajout d'un nouveau nœud p dans CAN suit les étapes suivantes :

1. le nœud p choisit aléatoirement un point de contact dans l'espace d'adressage ;
2. la zone dans laquelle le point de contact se trouve se divise en deux, l'ancien occupant p' dans le zone devient le responsable d'une moitié et le moitié restant devient sous la responsabilité de p.
3. pour que p participe au routage, une distribution uniforme des points de contact est admissible afin de maintenir des zones de taille égale, ce qui est important parce que l'acheminement efficace dépend d'une taille de la zone proche de l'ensemble des zones.

Par exemple dans la Figure 2.14, le pair i se connecte au CAN dans la zone initialement affectée au pair 4. 4 envoie à i les données qu'il doit prendre en charge. Lors du départ d'un pair, les opérations inverses sont effectuées.

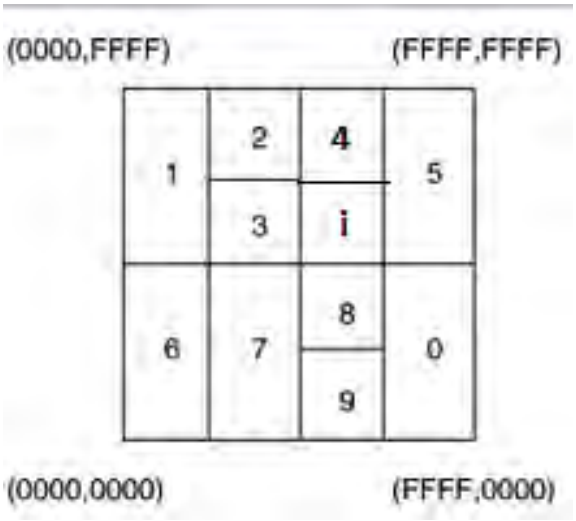


FIGURE 2.14 – Insertion d'un nœud i au CAN

Quand une panne est détectée par un nœud voisin, ce dernier récupère l'espace dont était responsable le nœud défaillant, met à jour ses tables de routage et envoie un message à ses voisins, pour s'assurer que leurs tables de routage ont bien été mises à jour.

En termes de performances, dans CAN, chaque pair maintient une table de voisinages de taille $2d$ et la complexité de routage d'une clé vers un nœud est de l'ordre de $O(d.N^1/d)$ avec N le nombre total de pairs. La mise à jour des voisinages lors du départ et de l'arrivée d'un pair impliquent $2d$ pairs voisins.

2.6.5 Extensions hiérarchiques

Les solutions pair-à-pair étudiées précédemment sont tous à design horizontale c'est-à-dire elles ont tous une architecture plane. Pour les améliorer différentes architectures hiérarchique P2P ont été proposée dans [PDQ+07], [YJZ08], [GEBF+03].

Dans [YJZ08], l'auteur propose une architecture hiérarchique à deux niveaux. Au niveau inférieur il regroupe les pairs d'une même région, organisés sur un anneau de Chord et coordonnés par un super nœud. Au niveau supérieur se trouvent les super- nœuds de chaque région. Le système [YJZ08] propose un algorithme de recherche régionale basé sur les super-nœuds, qui gardent une table de routage bidirectionnelle dans le but de réduire efficacement la redondance de la table de routage originale Chord. Toutefois, même si l'architecture passe à l'échelle, elle ne résiste pas dans un environnement dynamique.

Dans [GEBF+03]les auteurs ont proposé un modèle hiérarchique de DHT (HDHT), où les pairs sont organisés en groupes. L'objectif des HDHTs est d'améliorer l'architecture plane DHT conventionnelle, par une exploitation des ressources hétérogène des pairs, la mise en cache des infrastructures, la transparence et l'autonomie de différentes parties du système, afin de rendre la recherche de clés plus efficace tout en produisant moins de trafic.

Les travaux effectués dans [YJZ08], [GEBF+03] reposent sur une architecture hiérarchique à deux niveau. Leur objectif est d'améliorer efficacement les performances tels que la latence lors d'une recherche et générer moins de trafic réseau. Cependant, ces architectures montrent

leur limite dans les environnements dynamiques à cause de l'instabilités nœuds.

Une alternative est proposée dans [PDQ+07], où les auteurs proposent HP2P, une architecture hiérarchique hybride à deux niveaux, combinant DHT et systèmes P2P non structuré. HP2P utilise dans son premier niveau Chord et au deuxième niveau Kazaa, et procède par inondation lors d'une recherche.

Ceci fait de HP2P un système robuste car elle combine les avantages des DHT et des systèmes non structurés, mais reste aussi limité par leurs inconvénient, comme la dépendance aux ressources de même type (fichiers uniquement) et le nombre de messages exponentiel généré lors d'une recherche par inondation.

Face à ces travaux, nous sommes motivés à proposer la spécification d'une architecture hiérarchique plus générique, qui permet le stockage de tous les formats de données tout en conservant les avantages vis-à-vis de la performance réseau.

— du point de vu de la gestion des données, les solutions P2P non structurés offrent une solution intéressante pour la mise en place d'une communauté virtuelle P2P de partage de donnée. En effet, ils offrent une meilleure expressivité des requêtes donc capable de supporter des requêtes complexes. En plus les systèmes P2P non structurés ont un coût de maintenance faible du réseau en cas de la mobilité des nœuds par rapport aux systèmes P2P structuré.
— Du point de vu de la recherche, les systèmes P2P structurés, offrent un système efficace routage des messages. Ils constituent en générale la couche de routage des application de stockage de données(par exemple PAST qui utilise Pastry comme couche de routage).

2.7 Conclusion

Dans ce chapitre, nous avons étudié des solutions multi-échelle pour la gestion des données. Dans le cas ou les données se présentent sous la forme d'un fichier, nous avons étudié les technologies pair-à-pair, les NAS et les SAN, sous quelques uns de leurs aspects : structuration, façon d'indexer, et les protocoles de recherche utilisés. Dans le cas ou les données ne sont pas des fichiers nous avons étudié les solutions de stockage telles que les bases de données relationnelles, les bases de distribuées et les bases de données NoSQL concernant les mêmes aspects cités précédemment.

Nous avons constaté dans l'étude de ces différentes solutions que des inconvénients se posent sur le recherche de donnée. si certaines solutions comme les bases de données requiert toujours une entité centralisé qui agrège les requêtes de recherche. Les autres solutions comme pair-à-pair présente des inconvénients liés à la vitesse d'accès aux données qui sont lentes et des problèmes de sécurités.

Pourtant de toutes les constatations soulevées dans l'étude des différentes solutions de ce chapitre, nous nous sommes proposer une architecture multi-échelle pour le stockage unifiée de tous les formats de données(sous forme de fichier ou non fichier etc..) de façon transparente aux utilisateurs. La description de notre architecture est faite dans le chapitre 3.

Notre second proposition par rapport aux constatations faites sur les techniques de recherche est mise en place de protocoles de recherche et d'accès aux données qui ne dépendent pas d'une connexion directe entre les différents nœuds. Il est toujours possible par routage de faire une recherche ou un accès aux données dans notre architecture(voir chapitre 4).

Chapitre 3

Architecture de *GRAPP&S*

Résumé. *Ce chapitre présente GRAPP&S (Grid APPlication & Services), une spécification d'une architecture multi-échelle pour le stockage et l'indexation unifiée de différents formats de données tels que les fichiers, les flux de données, les requêtes sur les bases de données mais aussi l'accès à des services distants(des web services sur un serveur ou le cloud, ou des tâches de calcul dans un cluster HPC). L'architecture de GRAPP&S est constituée de trois types de nœuds ayant chacun des rôles différents. Ces nœuds sont regroupés sous forme de communautés(réseaux locaux) grâce aux principes multi-échelle.*

Les données sont présentées de façon transparente à l'utilisateur par l'intermédiaire de proxies(un exemple de nœuds de GRAPP&S) spécifiques à chaque type de donnée. La transparence offerte par les proxies porte sur la localisation des sources de données, le traitement des requêtes, la composition des résultats mais aussi la gestion de la consistance des données. De plus, l'architecture de GRAPP&S a été conçue pour permettre l'interconnexion de différentes communautés et l'application de politiques de sécurité et confidentialité, tant à l'intérieur d'une communauté qu'entre les différentes communautés.

Les travaux présentés dans ce chapitre ont fait l'objet de trois publications

- Thierno Ahmadou Diallo, Olivier Flauzac, Luiz Angelo Steffenel, Ibrahima Niang et Samba NDiaye : Grapp&s data grid : Une approche de type grille et système pair-à-pair pour le stockage de données. *In Revue URED (Université-Recherche-Développement), Presses Universitaires de l'Université Gaston Berger de Saint-Louis du Sénégal*. 2012.
- Thierno Ahmadou Diallo, Olivier Flauzac, Luiz Angelo Steffenel, Ibrahima Niang et Samba NDiaye : Grapp&s data grid : Une approche de type grille et système pair-à-pair pour le stockage de données. *In Colloque National sur la Recherche en Informatique et ses Applications (CNRIA'2012)*. Thiès/Bambey, Sénégal, April 25-27 2012.
- Thierno Ahmadou Diallo, Samba NDiaye, Olivier Flauzac et Luiz Angelo Steffenel : Grapp&s, une architecture multi-échelle pour les données et le services. *In Proceedings of 9èmes Journées Francophones Mobilité et Ubiquité (Ubi-mob 2013)*. Nancy, France, June 5-6 2013.

Sommaire

3.1 Introduction **56**
3.2 Modèle et environnement **57**
3.3 Éléments de l'Architecture GRAPP&S **58**

3.4 Ressources manager *RM* 62
3.5 Communicator (*C*) 68
3.6 Adressage des nœuds de GRAPP&S 69
3.7 Conclusion 72

3.1 Introduction

La gestion de données à grande échelle est un problème récurrent autant dans les domaines scientifiques que dans le monde de l'entreprise. Malgré les constantes avancées en matière de capacité des mémoires et disques, l'utilisation d'un seul dispositif de stockage n'est plus une option car l'accès concurrent, la fiabilité, la consommation énergétique et le coût sont des obstacles au développement des systèmes. Pour cette raison les chercheurs et développeurs se sont tournés depuis longtemps vers le développement de solutions de stockage distribué, afin de contourner ces limitations.

Á travers différentes stratégies, des solutions distribuées proposent des solutions transparentes à l'augmentation des besoins de stockage, tout en offrant suffisamment de garanties pour assurer la consistance et la pérennité des données. Aujourd'hui, l'utilisation de solutions de stockage de fichiers sur NAS/SAN ou cloud est devenu aussi courante que l'utilisation de disques ou clés USB, une fois que les ressources potentiellement illimités offerts par ces solutions présentent plusieurs avantages en ce qui concerne le coût, la disponibilité et l'utilisation des ressources physiques.

Cependant, ces solutions peuvent aussi présenter des inconvénients liés à la vitesse d'accès et à la sécurité des données. La solution a ses inconvénients est encore loin d'être garantie et dépend majoritairement des solutions propriétaires proposées par les fournisseurs des services de stockage. Un autre aspect à considérer est la compatibilité entre les systèmes : si certaines APIs rendent la manipulation des fichiers relativement simple, il est moins évident l'intégration d'autres représentations de données telles que les requêtes sur une base de données, des flux de données ou encore l'exécution de services.

C'est dans le but de proposer une architecture unifiée pour les données et les services que nous présentons GRAPP&S(*GRid APPlications and Services*), une architecture multi- échelle pour l'agrégation de données et services. Ce Framework à été conçu de manière à intégrer de manière transparente les données de type fichier mais aussi les bases de données, les flux(audio, vidéo), les services Web et le calcul distribué. À travers une structuration hiérarchique basée autour du concept de « communauté », GRAPP&S permet l'intégration de sources d'information disposant de protocoles d'accès hétérogènes et des règles de sécurité variées.

Le Framework GRAPP&S est une solution de stockage générique, qui utilise des nœuds spécifiques appelés « proxies » qui permettent d'unifier l'accès à des ressources variées telles que des fichiers, des flux, des services(FTP, mail) ou des données créées à la volée par des tâches de calcul ou des capteurs.

La suite de ce Chapitre est organisée comme suit : la section 3.2 d'écrit la spécification du Framework GRAPP&S en ce qui concerne le modèle de graphe utilisé par notre Framework, certaines définitions générique. Enfin nous présentons les principaux éléments qui constituent l'architecture GRAPP&S et comment ces éléments s'inter-connectent. La section 3.3.2 d'écrit le rôle spécifique du nœud *Data Manager* noté DM, sa structure logique et la gestion des données

par ces nœuds DM. La section 3.4 détaille le rôle des nœuds *Ressource Manager* noté RM et sa structure logique. On fera une discussion sur les possibilités d'indexation de données que le nœud RM peut adopter. En fin nous allons proposer un algorithme d'élection d'un nouveau RM en cas de panne. La section 3.5 d'écrit le rôle du nœud *Communicator* noté *c*. Dans le section 3.6, nous allons discuter des modèles d'adressage que les éléments de l'architecture GRAPP&S peut adopter. Finalement, la section 3.7 propose nos conclusions pour ce chapitre.

3.2 Modèle et environnement

L'architecture que nous proposons dans ce chapitre exploite le modèle que nous avons défini dans les chapitres précédents.

3.2.1 Quelques définitions

Dans un premier temps, nous donnons la définition des différents composants élémentaires, ainsi que des différents objets et structures que nous exploiterons par la suite.

Définition 3.1. *Un **nœud** est défini comme étant une capacité de calcul, de stockage, avec des moyens et des canaux de communications.*

Définition 3.2. *Une **donnée brute** est un flux d'octets qui peut être sous différentes formes : une base de données objet ou relationnelle, un fichier(texte et hypertexte, XML), un flux(video, audio, VoIP), des fichiers P2P, des requêtes de base de données ou des résultats issus d'un calcul ou d'un service.*

Définition 3.3. *Un réseau overlay ou réseau logique relie virtuellement les nœuds participants d'un système pair-à-pair, au dessus d'un réseau physique[BYL08]. Il fournit la connectivité, le routage et l'acheminement de messages entre les nœuds.*

Un réseau logique est utilisé comme support au déploiement d'application réseau ou pour fournir une structure de routage qui n'est pas accessible à partir du réseau physique sous-jacent.

3.2.2 Communication et les réseaux overlay

Le modèle de communication présenté dans le chapitre 1 est suffisamment générique pour ne pas inférer sur la manière dont les messages sont effectivement livrés, se limitant uniquement à la définition des propriétés de communication bidirectionnelles entre deux sommets. Pour cette raison, l'architecture que nous poposons peut s'appuyer sur n'importe quel réseau de communication *overlay* qui garantit une communication bidirectionnelle fiable entre deux sommets, et qui permet d'explorer différents chemins de communication pour chaque arête dirigée (réseau routé). Ceci donne une plus grande liberté d'implémentation et d'adaptation à l'environnement d'exécution, vu que les opérations *send/receive* peuvent être implémentées de différentes façon, selon les capacités de communication des nœuds. Dans ce cas, trois scénarios principaux peuvent être considérés : mode **Push**, mode **Pull**, et enfin en mode **Proxy**(voir chapitre 1).

Dans ces trois scénarios, il est toujours possible d'établir un voisinage direct ou partiel entre les processus, ce qui est compatible avec le modèle par graphes connectes dirigés.

3.3 Éléments de l'Architecture GRAPP&S

Dans cette section nous allons présenter notre Framework GRAPP&S qui est une architecture hiérarchique pour le stockage et l'indexation unifiée de tous les types de données et Services. Ce Framework, est une solution de stockage générique qui utilise des acteurs spécialisés qui permettent de se détacher des différentes sources de données tels que : un stockage sur disque local, serveur FTP, ou sur un serveur Mail Gmail par exemple. Les proxy/data permettent la gestion des données à savoir lecture et écriture de façon transparente a l'utilisateur, comme pour le système.

3.3.1 Notion de communauté

Le Framework GRAPP&S dispose des propriétés telles que la gestion de la connexité de réseaux locaux appelés *communautés*, qui permet l'intégration de multiple sources de données divers avec des protocoles de gestions hétérogènes. Afin de présenter notre architecture, nous introduisons dans un premier temps quelques notations.

Définition 3.4. *Une* ***communauté*** *notée C_i est une entité autonome, qui regroupe des nœuds qui peuvent communiquer et qui partagent une propriété particulière.*

Les propriétés qui permettent de réunir des nœuds en communautés sont par exemple :

1. même localisation géographique : L'agrégation des nœuds qui se trouvent dans un même pays, ou une même région, ou en zone plus réduite. Ce regroupement peut concerner aussi des nœuds d'un laboratoire de recherches dans une université ;
2. même autorité d'administration : dans ce cas il s'agit de l'agrégation des nœuds appartenant à une même entreprise. Ces différents nœuds se trouvent dans des sites géographiques différents ;
3. même domaine d'application, qui concerne les nœuds qui peuvent être regroupés dans une même machine physique. C'est l'exemple d'un particulier chez lui qui souhaite héberger une communauté de GRAPP&S.

Une communauté GRAPP&S se constitue à partir de trois types de composants :
— le *Communicator* noté C ;
— le *Ressource Manager* noté RM ;
— le *Data Manager* noté DM.

Ces composants sont organisés de façon hiérarchique. L'interconnexion entre différentes communautés C se fait grâce à des liens de voisinage point-à-point entre les processus communicator(c).

la figure 3.1 donne une vision globale de l'architecture proposée.

Nous allons maintenant décrire chaun des composants, en exposant ses attributions, ainsi que les inter-connexions et communications associées. Nous reviendrons sur l'urbanisation des communautés après avoir détaillé les différents composants que sont les *Communicator*, *Ressource Manager* et les *Data Manager*

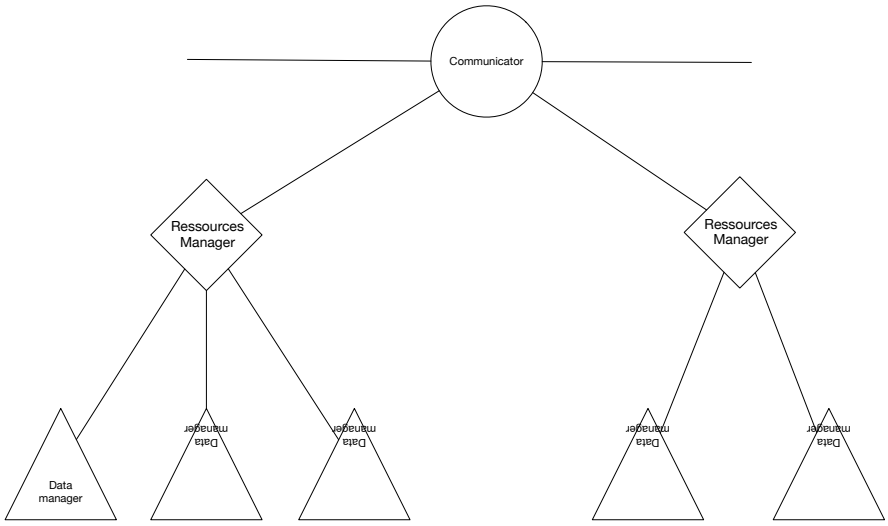


FIGURE 3.1 – Organisation des nœuds de GRAPP&S sous forme de communautés

3.3.2 Data Manager (DM)

Les nœuds de type *Data manager* (DM), sont des acteurs spécialisés qui interagissent avec les sources de données. Un des objectifs de ces nœuds est de permettre d'accéder aux données de manière transparente, sans que l'utilisateur ou le processus qui exécute cet accès ne se soucie du support, ou du protocole de stockage, de cette donnée. Dans la suite, nous utiliserons aussi le terme de *proxy* ou *proxy de données* quand nous évoquerons un nœud de type *Data manager*.

3.3.3 Structure logique d'un nœud DM

Un acteur de type *Data Manager* est constitué d'un ensemble de composants (détaillés dans la figure 3.2). Ces composants sont architecturés en couches et offrent des fonctionnalités spécifiques : l'interface utilisateur, le gestionnaire de requêtes, l'adaptateur, le gestionnaire de communications et la source de données.

Adaptateur ou Interface Proxy

Les nœuds DM peuvent se différencier suivant le type de données et de langage de requêtes qu'ils gèrent. Les adaptateurs ou Interfaces Proxy leur permettent de participer au réseau de partage de données en traduisant les requêtes dans un langage compréhensible par les sources de données. En fonction de la source de données interrogées, les adaptateurs ou proxy font appel à une méthode de conversion qui convertit le résultat dans un format approprié et compréhensible et afin d'envoyer les données traduites.

Cet adaptateur est relié à la source de données par un protocole de connexion spécifique au type de donnée, par exemple JDBC, ODBC, FTP, etc.

Cet élément est en fait un élément central de notre spécification, car il permet à des requêtes génériques, générées par un utilisateur ou par un processus, d'être exploitées sur des plate-formes

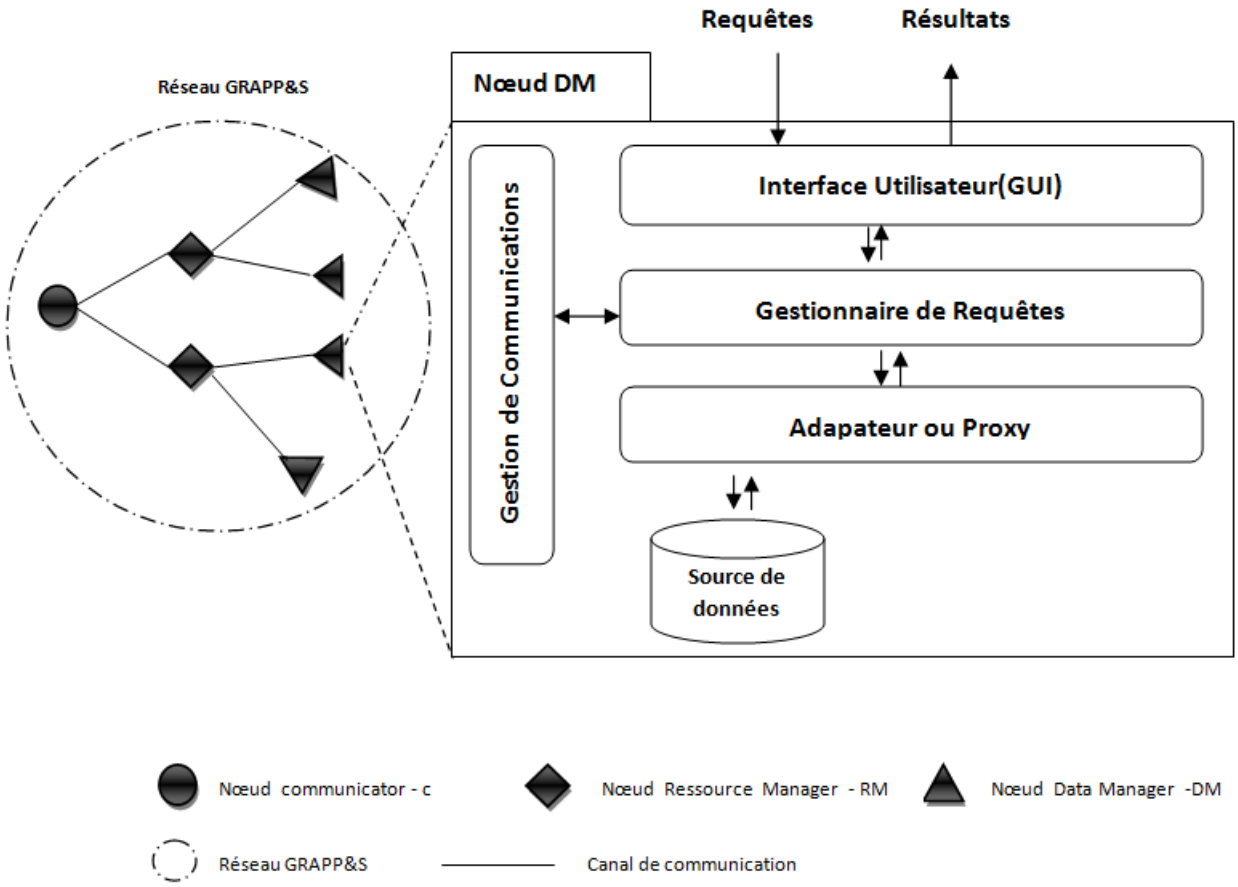


FIGURE 3.2 – Architecture d'un nœud DM

ou des solutions de stockage tout à fait hétérogènes. Ainsi la recherche d'un fichier pourra aussi bien être traduite en requête SQL, en lecture d'un système de fichiers ou en recherche parmi une collection de mails.

Gestionnaire de Requêtes

Ce composant permet d'exprimer des requêtes locales ou globales dans l'ensemble d'une communauté de GRAPP&S. Les requêtes locales sont exécutées sur le DM local sur lequel interagit l'utilisateur et les requêtes globales seront routées vers le père du nœud DM originaire de la requête, pour une diffusion à la communauté dans laquelle appartient ce DM ou bien aux autres communautés de GRAPP&S.

Interface graphique utilisateur ou API

L'interface utilisateur (ça peut être aussi une API) permet à un utilisateur d'importer des données, mais aussi de formuler plus facilement des requêtes locales sur ses données ou globales dans l'ensemble du réseau de GRAPP&S. Les données et les nœuds sont transparents

à l'utilisateur, c'est-à-dire, ce dernier n'a pas du tout connaissance des nœuds distants et qu'il formule des requêtes sur son nœud local *Data Manager-(DM)*.

Gestionnaire de communication

Ce module permet au nœud *Data Manager-(DM)* de communiquer avec le nœud *Ressource Manager* (RM) auquel il est connecté. Cette communication est assurée par des services tels que RMI, des protocoles liés aux webservices et des protocoles réseaux des différentes couches TCP, $HTTP$. Le nœud DM dispose des protocoles de connexion spécifique au type de donnée, par exemple $JDBC$, $ODBC$, FTP etc.

On pourra noter que notre spécification présente la particularité d'être générique au niveau des données et des moyens de stockage de ces données, mais aussi au niveau des protocoles exploités. Chacun des composants décrits dans cette section peut être redéveloppé à loisir, permettant ainsi d'intégrer de nouveaux protocoles d'échanges et de communication entre les différents acteurs.

Source de données

Chaque nœud *Data Manager-(DM)* interagit avec une seule source de données permettant de gérer des données telles que : base de données objet, base de données relationnelle, des documents XML, Texte, multi-média, etc., des requêtes de base de données, des données issus des capteurs ou encore des services *cloud*, des flux vidéo, audio, de la VoIP etc. Chaque nœud DM est indépendant des autres nœuds DM, et accède à différentes sources de données.

3.3.4 Gestion des données par les DM

Les processus *Data Manager-(DM)* sont des services qui interagissent avec les sources de données, qui peuvent être sur différents supports tels que les bases de données objet ou relationnelle, les services *cloud* (par exemple *Dropbox*), les serveurs Mail (Gmail par exemple), les serveurs FTP et Webdav, des disques ...

La gestion de ces sources de données différentes se fait de manière transparente à l'utilisateur grâce à l'utilisation de « proxy » ou adaptateurs adaptés aux différentes sources de données et relié à ceux-ci par des protocoles de connexion spécifiques aux types de données par exemple JDBC, ODBC, FTP, CIFS, NFS, etc.

Le rôle d'un proxy est celui de traducteur de requête, ainsi que de traducteur de résultats. Ainsi les opérations liés au traitement d'une requêts sont les suivantes :

1. à la réception d'une requête à destination d'une source de donnée, le proxy procède au préalable à une opération de traduction de la requête dans un langage compréhensible par la source si celle-ci est une application.
2. Selon le type de source interrogée, le proxy peut faire usage des plugins pour fournir une réponse.

Afin d'exploiter au mieux les ressources, il est nécessaire d'envisager des solutions de mappage des processus sur les sources de données. Deux possibilités de mappage des processus DM sur les sources de données peuvent être envisagés :

1. chaque source de données de GRAPP&S est associée a un service DM. Cela est justifié par sa simplicité à mettre en œuvre lors de la phase de développement. On utilisera les concepts d'héritage et de polymorphisme ;
2. un service DM gère plusieurs sources de données. Dans ce cas le service DM est obligé de maintenir une table d'indexation des différents types de données qu'il aura à gérer. En plus, les requêtes traitées par le seul nœud DM peuvent nécessiter la manipulation d'un nombre important de sources de données. Le service DM est chargé de traiter toutes les réponses des différentes sources de données et de les unifiés, afin de fournir une réponse globale à la requête posée par l'utilisateur. Ceci peut nécessiter un temps de réponse plus long des services DM.

Remarque 3.1. *dans un contexte du web, le nombre de sources peut être très important. S'il faut mapper ces dernières par un seul DM, alors le stockage de la table d'indexation sera relativement coûteuse vu que la puissance de stockage des adaptateurs est limitée.*

Remarque 3.2. *L'unicité d'un service DM pour gérer plusieurs sources de données, devient un point critique à la suite de la disparition du DM. S'il tombe en panne alors toutes les sources qu'il gérait deviennent inaccessibles.*

3.3.5 Identification des données

La manipulation de données hétérogènes, dans un environnement de stockage hétérogène, nécessite de qualifier les données de manière générique. En effet, il n'est pas possible, comme dans le cas des réseaux P2P, de désigner une donnée par juste une clé (signature, *hash*, nom de fichier), ou comme dans le cas d'une base de données de désigner cette donnée par une requête. Il nous faut donc proposer une solution d'identification et de désignation des données. Pour cela nous proposons de nous inspirer de la notion de type MIME.

En effet, la nature d'une donnée web est définie de manière unique par son type MIME. Chaque type de donnée de GRAPP&S sera identifié de manière unique grâce à l'identifiant du nœud DM, auquel s'ajoute une extension contenant des métas informations (par exemple les tag, la signature ...) et le type MIME des données.

Ceci permet de franchir la barrière du simple « nom de fichier », et peut donc faire cohabiter des données statiques (fichiers), des données dynamiques (requêtes sur une base de données, résultats d'un calcul) et des données à caractère temporaire (flux voix ou vidéo, état d'un capteur, etc.).

La définition de ce type MIME étendu répond donc aux besoins de généricité que nous nous imposons : généricité des types de données, ainsi que des supports et architecture de stockage.

3.4 Ressources manager *RM*

Les composants de type *Ressource Manager RM* assurent l'indexation et l'organisation des données et des services dans une communauté de GRAPP&S. Ils reçoivent les requêtes des utilisateurs et assurent leur pré-traitement. Les nœuds *RM* participent à la recherche de données dans la communauté. Pour des fins de tolérance aux fautes et performances, les informations indexées par les RM peuvent être répliquées et/ou partiellement distribuées(DHTs,

par exemple). Afin de rendre plus performante la coordination des nœuds *RM*, nous préconisons l'élection d'un *RM* maître, qui assurera la gestion principale d'un ensemble de nœuds de type *DM*. En cas de panne de ce *RM* maître, une nouvelle élection permettra de désigner le nœud *RM* qui récupèrera les nœuds *DM* devenus « orphelins ».

Dans la suite nous présentons d'abord la structure logique et le rôle des processus *Ressources Manager* (*RM*). Nous aborderons dans un second temps les différentes possibilités d'indexation qu'un nœud RM peut implémenter. Dans un troisième temps nous détaillerons la méthode employée pour l'élection d'un nœud *RM* à la suite d'une panne de ce dernier afin de garantir la reconstruction du réseau et la continuité du service.

Nous supposons que tous les nœuds *RM* ont des architectures et rôles identiques. Ils peuvent se différencier sur leur capacité de calcul, de bande passante, de nombre de connexions etc.

3.4.1 Structure logique nœud RM

La structure logique de chaque nœud *RM* dans GRAPP&S est composée des éléments suivants : un index local, le gestionnaire de requête, composé du module de routage et du moteur de recherche des requêtes, le gestionnaire de communications. La structure globale en couches d'un *Ressource Manager* est présenté à la figure 3.3.

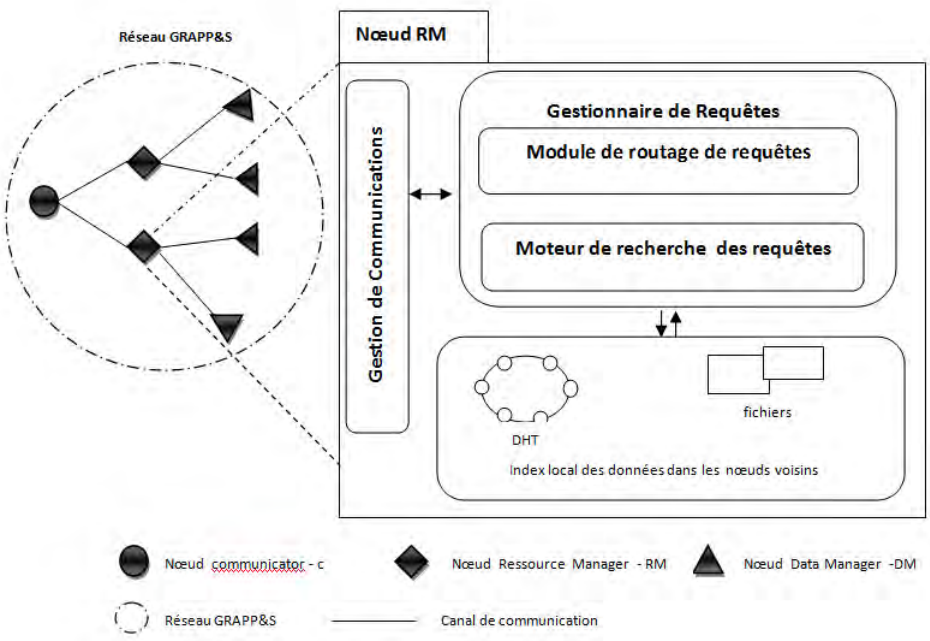


FIGURE 3.3 – Architecture d'un nœud RM

Index local

Ce composant indexe les données et les services dans une communauté de GRAPP&S. L'indexation peut se faire avec une Table de Hachage Distribuée(DHT) ou bien avec un fichier selon taille et les caractéristiques des données concernées. Dans le module index local, on fait une

correspondance entre les identifiants des nœuds DM, et un registre contenant des informations et le type $MIME$ étendu des données, que nous avons présenté précédemment.

L'indexation se fait donc de manière étendue, prenant en compte l'hétérogénéité des types de données, des supports et des architetures de stockage.

Gestionnaire de Requêtes

Le gestionnaire de requêtes est constitué d'un *module de vérification de l'index* et d'un *module de routage de requêtes*.

Le module de vérification de l'index permet, une fois que le nœud RM reçoit la requête, de vérifier dans l'index du RM, s'il existe un nœud DM qui détient l'information recherchée, quel que soit sont support de stockage.

Le module de routage de requêtes permet d'assurer le transfert et le routage des requêtes vers les nœuds DM en réponse à la requête reçue.

Il est aussi possible, dans le cas d'une redondance des index, de router la requêtes directement à un RM voisin dont on connait tout ou partie de l'index.

Si la requête n'aboutit pas localement, le nœud RM route vers son *Communicator* associé afin qu'elle soit transmise à d'autres RM ou à d'autres communautés de GRAPP&S.

Gestionnaire de Communications

Ce composant assure la communication d'un nœud RM avec ses voisins directs, qui peuvent être des nœuds DM, des nœuds RM dont il connait les informations (dans le cas de la redondance des index) ou le nœud C. Cette communication est assurée par des services RMI, des protocoles associés aux webservices et des protocoles réseaux de différentes couches TCP, HTTP.

3.4.2 L'indexation dans les *Ressources Manager*

L'un des rôles principaux d'un nœud RM est d'assurer l'indexation des données dans une communauté de GRAPP&S. Nous allons proposer plusieurs solutions d'indexation, la modularité de notre système permettant, comme dans le cas des proxy de données, d'implémenter différentes solutions ou de redévelopper à posteriori des méthodes spécifiques d'indexation. L'exploitation d'un modèle en couche permet de définir les points de contact des différents couches, et donc de pouvoir les redéfinir.

L'indexation locale

Le principe de base d'un index est d'associer à la clé d'une ressource son adresse relative dans le fichier à l'aide d'une « table des matières » du fichier. Ce mode de placement et d'organisation, est appelé index[Gar99].

Dans le type d'indexation locale, l'index est stocké sur un nœud. Lors d'une opération de récupération de donnée, la demande de recherche se fait par inondation ou par marche aléatoire[LCC+02]. Dans le cas de l'inondation la requête est diffusée à tous les nœuds, tandis que le cas de l'approche aléatoire la requête est transmise de nœud à nœud jusqu'à la découverte des données, en vérifiant régulièrement si l'origine de la requête veut continuer la recherche afin d'arrêter la marche une fois que la requête a trouvé une réponse.

L'avantage de l'index local est que chaque nœud peut évaluer la pertinence entre la requête de recherche et les données qui sont dans son index sans consulter les autres nœuds. Il décide s'il doit transmette la requête de recherche ou non.

Cependant que ce type d'index est qu'il génère trop de messages lors d'une recherche du fait chaque nœud est impliqué dans le traitement de chaque requête. Ce trop plein de messages limite son passage à l'échelle. Le système Gnutella[Rip01] est sur une variante de l'index local. En effet, dans Gnutella, chaque pair indexe ses propres fichiers. Cette connaissance n'est pas partagée ou déléguée à d'autres pairs.

Des solutions tendant à améliorer le routage des requêtes dans le cas d'index local ont été proposées. Ces solutions sont basées sur l'utilisation d'heuristiques[YGM02] où par l'introduction de nœuds présentant plus de connexions[ALPH01].

L'indexation centralisée

L'index est centralisé lorsqu'il réside dans un serveur qui se charge de mettre en relation directe tous les utilisateurs connectés. L'avantage de centraliser l'index, est la connaissance de l'ensemble des ressources partagées par les pairs à un moment donné. Le serveur qui joue le rôle d'annuaire doit être :

— puissant pour indexer l'ensemble des ressources et parcourir rapidement l'index afin de répondre aux nombreuses requêtes des pairs.
— disponible car sans lui les pairs ne peuvent localiser les ressources et aucun service n'est rendu.

L'index des fichiers partagés peut être centralisé dans un seul serveur. Par exemple c'est le cas le Napster[FP99b], ou comme dans le cas de Bittorrent [Coh03], l'annuaire centralisé peut être hébergé sur des serveurs différenciés.

L'indexation centralisée peut aussi être implémentée sur des serveurs distribués. Chaque serveur stocke l'annuaire centralisé à la manière de Naspter(voir section.2.2). Le contenu mis à disposition par l'ensemble des pairs est indexé par des clusters de serveurs distincts bien connus. Ce type d'indexation centralisé est implémenté par le réseau eDonkey[HBMS04b]. La recherche de données dans eDonkey est d'abord effectuée sur le serveur auquel est connecté le pair. Si cette recherche est infructueuse, elle peut être étendue aux autres serveurs du cluster jusqu'à trouver des sources.

L'avantage d'héberger l'index sur des serveurs différenciés est que la panne d'un serveur ne pénalise pas tout le système.

La simplicité pour diffuser les ressources et les récupérer en font des réseaux fortement adaptés à la diffusion de ressources diverses. Ces réseaux sont fragiles en deux points : la mise à disposition initiale de la ressource ne se fait que sur une seule machine et la fragilité du principe d'indexation qui repose sur un ensemble de nœuds pour identifier l'ensemble des ressources du système.

L'indexation distribuée

L'une des approches d'indexation les plus couramment utilisés pour les systèmes de P2P est le Distributed Hash Tables(DHT). Dans la DHT, une table de correspondance avec des paires de clé et la valeur est maintenue, de sorte que les pairs dans le réseau P2P peuvent effectivement récupérer la valeur qui correspond à une clé donnée. Les tables de hachage distribuées sont

une catégorie de réseaux décentralisés structurés qui fournissent deux primitives, `put` et `get`. Soit une donnée v possédant une clé k définie sur le même espace d'adressage que les pairs, la primitive put(k,v) stocke la donnée v sur le pair en charge du sous-espace contenant k. La clé k d'une valeur v est obtenue grâce à la fonction de hachage h(v)= k. Il y a deux façons de stocker les données. On peut le faire directement quand la valeur des données est stockée directement par le pair responsable de leurs clés associées. Une autre alternative est de stocker des pointeurs là où les valeurs des données sont véritablement stockées.

Pour rechercher une donnée avec les DHT : étend donné une clé k, on peut localiser efficacement le nœud qui stocke la clé avec au maximum $logN$ messages. N étend le nombre de nœuds du réseau. La primitive get(k) récupère la donnée associée à la clé k en interrogeant le pair responsable du sous-espace contenant k.

Toutefois les DHT imposent un enregistrement sur la base d'une valeur unique de clé. De plus, il détruit la localité des données. La destruction de la localité compromet le traitement efficace des requêtes complexes. Une requête complexe est toute requête qui ne se contente pas de rechercher une données unique, mais de rechercher un ensemble de données à partir d'une ou de plusieurs identifiants. Exemple de requêtes complexes : les requêtes par intervalles, les requêtes de similarité et les requêtes skyline, etc...

Pour que les DHT puissent prendre en compte les requêtes complexes, il faut concevoir un système d'index efficace. Pour cela des techniques d'indexation sont proposées dans la littératures. Certaines construisent un index au dessus de la DHT et sont appelées des over-DHT, d'autres construisent un arbre B+ sur une DHT. Et enfin, certaines posent le système d'index sur la DHT.

Les différentes techniques présentées précédemment peuvent être exploitées dans le cadre de nos *Ressource Manager*. Le choix qui serait effectué pour implémenter une ou l'autre technique doit être guidé, lors de l'implémentation, par différentes contraintes ou différents objectifs :

— le volume et le nombre des données à indexer ;
— l'efficacité de la solution proposée ;
— la dynamicité des nœuds du système ;
— les ressources disponibles des *Ressources Manager*.

3.4.3 Élection d'un *Ressource manager*

Vu le rôle important des nœuds de type *Ressource manager* dans l'indexation et la recherche d'informations dans GRAPP&S, il est important de toujours rendre la disponibilité de ce nœud RM pour éviter l'arrêt de services offert par GRAPP&S. Pour cela nous étudions dans cette section les possibilités de tolérance aux pannes des nœuds RM.

Détection de la panne d'un nœud RM

Les nœuds peuvent subir des déconnexions volontaires ou involontaires. Comme le cas des déconnexions volontaires est simple à gérer (un message de déconnexion est émis à destination du voisinnage, qui a le temps de mettre en place un mécanisme de copie / basculement de RM), nous nous concentrons ici sur les déconnexions involontaires.

Dans notre architecture, entre deux niveaux hiérarchiques, les pannes peuvent être détectées soit par des messages périodiques de type *Pull* (aussi connu comme *heartbeat*), ou encore en

s'appuyant sur un mécanisme propre au middleware *overlay*. Pour les nœuds appartenant à un même niveau hiérarchique, la surveillance peut aussi se faire grâce à un mécanisme de passage de *jeton* « de service ». Cela permet non seulement l'allégement du mécanisme de détection (il suffit de surveiller son prédécesseur et son successeur) comme permet la diffusion rapide des informations à l'ensemble des nœuds.

Pour la mise en place d'un mécanisme générique de détection de défaillances, nous préconisons une procédure en deux étapes. Tout d'abord, chaque nœud dispose d'une liste de voisins $\{ N_1,...,N_n\}$ composée des nœuds en contact direct (par exemple, un RM_i est en contact avec son c, ses DMs et ses voisins RM_{i-1} et RM_{i+1}). Á cette liste de voisins est associée une liste de temporisateurs d'attente$\{ta_1,...,ta_n\}$.

Si aucun message du nœud N_k n'est reçu jusqu'à l'expiration du temps ta_k, une suspicion de défaillance est levée et doit être vérifiée auprès d'un deuxième nœud qui est aussi en contact direct avec le nœud suspect. Ainsi, si la suspicion concerne le nœud c, un nœud RM_i interroge son voisin direct RM_{i+1} avec un message jeton initialisé à **faux**. Si RM_{i+1} a reçu un message du nœud c avant l'expiration de son temps d'attente ta_c, RM_{i+1} modifie la valeur du jeton à **vrai** et retourne le message jeton à son émetteur RM_i. Ceci signifie(indirectement) que le nœud c n'est pas déconnecté et le nœud RM_i peut envoyer à nouveau un message au nœud c. Si par contre RM_{i+1} n'a pas été contacté récemment par le nœud c, il fera suivre un jeton la valeur **faux** qui, grâce au passage du jeton, alertera tous les nœuds $RM\{ RM_1,..,RM_n\}$ de la défaillance de c.

De manière similaire, si un nœud c suspecte un nœud RM_k, il peut demander confirmation à RM_{k+1}. Évidemment, cette procédure générique peut s'adapter aux différentes situations telles qu'un nœud qui contient un RM et plusieurs DM. Dans ce cas, le mécanisme de détection peut être allégé pour mieux répondre aux caractéristiques du nœud.

À la suite de la confirmation d'une défaillance, les nœuds concernés doivent (i) mettre à jour leurs informations (liste de voisin, tables d'index, etc.) et éventuellement (ii) procéder à l'élection d'un nouveau RM qui prendra en charge les éventuels DM orphelins.

Algorithme d'élection d'un nouveau RM

Dans l'optique de gérer les déconnexions(pannes) des nœuds dans notre architecture, nous présentons dans cette section les algorithmes de gestion des pannes dans notre système. En effet vu le caractère dynamique et volatile des réseaux informatiques, il est important de choisir un algorithme d'élection qui soit le plus léger et réactif possible.

L'élection d'un nœud peut être nécessaire en deux situations : soit pour remplacer un nœud défaillant et garantir la continuité du service (par exemple, lors de la panne d'un nœud c), mais aussi pour simplifier la coordination entre les nœuds de même type, avec par exemple l'élection d'un RM qui agirait comme « super-node » pour l'indexation de données et services.

Il faut noter que la connaissance préalable des nœuds du niveau supérieur n'est pas obligatoire, vu que différentes techniques permettent d'obtenir les identifiants des autres nœuds. La méthode la plus simple consiste à utiliser directement la hiérarchie de GRAPP&S : indépendante du middleware de communications, il est aisé de remonter la hiérarchie GRAPP&S et de contacter d'autres nœuds (grâce au routage du rèseau *overlay*). Il suffit donc de remonter les niveaux ou de contacter d'autres nœuds dont on récolté les identifiants (ceux dont on a reçu des requêtes récemment, par exemple).

Vu que le problème de la reconnexion au reste de la communauté peut être traité de manière plus ou moins simple au sein de la propre architecture GRAPP&S, il est intéressant de se pencher sur les algorithmes d'élection eux-mêmes. Dans GRAPP&S, nous préconisons un algorithme d'élection distribué inspiré des protocoles de routage OSPF et IS-IS[Ora90, BMO05, M$^+$98]. En effet, les nœuds GRAPP&S disposent d'un identifiant unique qui peut être utilisé de manière systématique par ces algorithmes d'élection.

Le choix entre les algorithmes des IS-IS ou d'OSPF est plus lié aux préférences d'implémentation et à l'hétérogénéité des nœuds. En effet, l'algorithme d'élection de IS-IS est de type déterministe, où l'élu est toujours le nœud avec le plus grand identifiant(appelé DIS- Designated IS). Ce mécanisme est simple à implémenter et ne requiert pratiquement aucun échange d'informations car les nœuds disposent déjà d'une liste avec les identifiants de leurs voisins, il ne resterait que le coût associé à la prise de fonctions d'un nœud élu à un rôle différent de celui qu'il occupait précédemment. L'inconvénient de cette technique est qu'un réseau avec un fort taux de volatilité peut occasionner des élections à répétition, soit lors de la déconnexion du leader, soit lors de la connexion d'un nœud avec un identifiant prioritaire.

Dans les cas où la volatilité risque d'impacter la performance du réseau, il est possible d'utiliser un mécanisme non-déterministe comme celui d'OSPF. Dans ce type d'algorithme, plus conservateur, le choix d'un leader(DR- Designated Router) n'est nécessaire que si le leader actuellement en place disparaît. Ainsi, l'entrée de nouveaux nœuds dans la communauté a un impact moins important sur le fonctionnement du réseau.

3.5 Communicator (C)

Le nœud *communicator-(c)* joue un rôle essentiellement lié au transport d'informations et à l'interconnexion entre différentes communautés. Ceci implique la coopération et la coordination entre ses fournisseurs de services (les communautés), ceci permet à un utilisateur de notre *framework* d'avoir accès à une architecture multi-échelle.

Le nœud *Communicator* (C) est le point d'entrée de la communauté, il assure sa sécurité vis-à-vis de l'extérieur, grâce à l'établissement de règles d'échanges et de rejet, de type *ACL* avec les autres communautés. De même, le nœud *communicator* coordonne la sécurité intérieure de la communauté, et peut modifier ses politiques d'accès grâce à des décisions prises au sein de la communauté. nous considérons donc, qu'au dessus des solutions de sécurité offertes par le réseau *overlay*, le communicateur joue le rôle de pare-feu des communautés GRAPP&S, en assure la sécurité.

Le nœud communicator c coordonne la communication des nœuds RM et des DM à l'intérieur d'une communauté ou entre différentes communautés. En effet, lorsqu'un nœud RM_x veut envoyer un message de recherche à un autre RM_y, situé dans une même communauté ou bien même de communauté différente, le nœud communicator c va servir de passerelle, qui se charge de le faire parvenir aux nœuds de destination.

Le rôle d'un nœud de type *Communicator* est primordial, assurant le relai des requêtes et, comme nous le verrons par la suite, le routage des messages inter-communautés et la sécurité des différentes communautés. Sa structure est très simple, pouvant être assimilé à une solution de filtrage.

3.6 Adressage des nœuds de GRAPP&S

Chaque nœud de notre architecture, détient au moins une adresse dans le système sous-jacent. il peut aussi détenir une adresse spécifique dans le réseau *overlay* sur lequel nous nous appuyons. Bien que titulaire de deux adresses, il est difficile, dans notre système d'exploiter ces adresses, car la construction des communautés ne respecte pas de logique propre ni au réseau sous-jacent, ni au réseau *overlay*. Il est donc nécessaire, de définir un adressage logique propre à GRAPP&S. Nous allons dans la suite, étudier les différentes possibilité d'adressage que nous pouvons exploiter dans GRAPP&S.

3.6.1 Adressage à plat

Dans ce type d'adressage chaque entité du réseau possède une adresse unique sans aucune règle de structuration dans le réseau. La figure3.4 représente un réseau avec une structure d'adressage à plat.

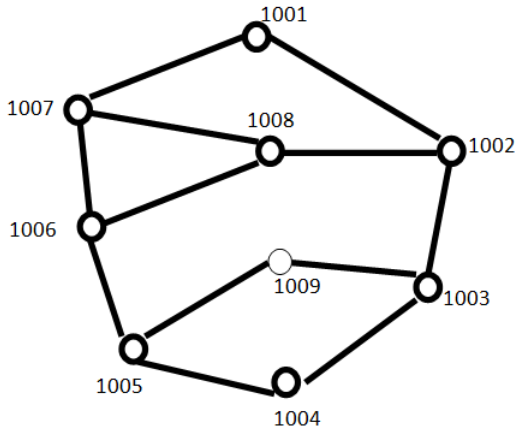


FIGURE 3.4 – Un exemple de structure de numerotation à plat

Destination	Route	Sauts
1001	1003-1002-1001	3
1002	1003-1002	2
1003	1003	1
1004	1003-1004	2
1005	1005	1
1006	1005-1006	2
1007	1005-1006-1007	3
1008	1003-1002-1008	3

TABLE 3.1 – Table de routage d'un nœud dans un adressage à plat.

Pour que les nœuds puissent communiquer, il faut que chacun maintient une table de routage qui contient les routes et le nombre de saut qui le séparent des autres nœuds. Le tableau 3.1 illustre la table de routage du nœud d'identifiant 1009. Le protocole de communication se fait en point à point(Unicast), et dans certains cas par groupe (Mutilcast) ou par inondation (Broadcast).

La structure d'adressage à plat a cependant l'inconvénient de générer des tables de routage qui augmentent linéairement avec la taille du réseau et inutilisable. Par exemple, dans le réseau comme Internet, cette technique d'adressage est inutilisable à cause de la taille de la table de routage qu'on aurait à gérer. Le système d'adressage hiérarchique permet de surmonter ces problèmes.

3.6.2 Adressage hiérarchique

Dans un système d'adressage hiérarchique, l'adresse de chaque nœud est composée de plusieurs parties correspondant à des adresses de réseau, de sous-réseaux et finalement des machines dans le réseaux. Ces différentes parties correspondent aux découpages du réseau en parties adjacentes. Dans ce système, un seul nœud d'un groupe(réseau local), appelé aussi *gateway* gère une table de routage sur un ensemble de nœuds « proches ». Le tableau 3.2, illustre la table de routage du nœud 01.2 (gateway). Les autres nœuds du réseau local n'ont qu'à connaître cette nœud pour communiquer. Lorsqu'un nœud veut envoyer un message à un autre, situé sur un réseau différent, il envoie son message à la passerelle, qui se charge de le faire parvenir au réseau de destination. La figure 3.5 illustre un exemple d'adressage hiérarchique. Le réseau est découpé en deux clusters, numérotés 01 et 02. Dans chaque cluster il existe un nœud *gateway*.

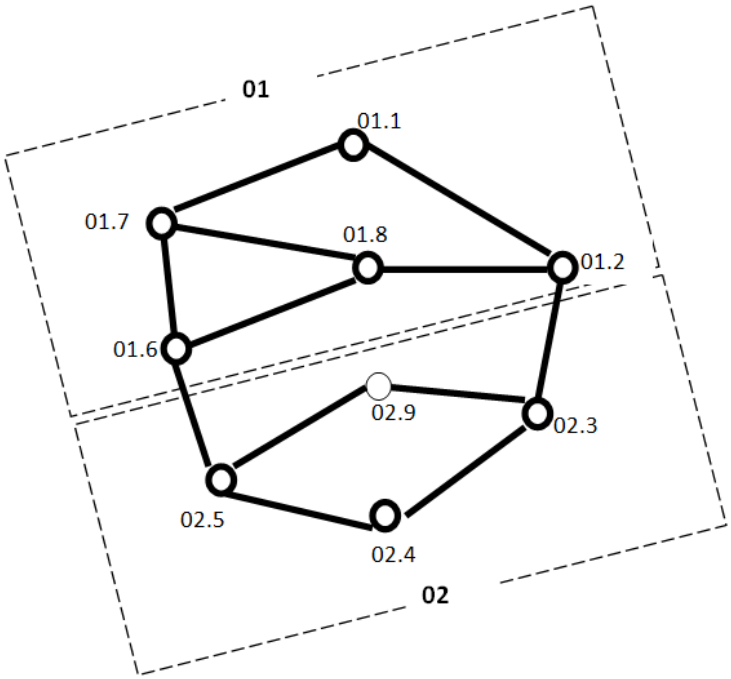


FIGURE 3.5 – Structure d'adresse hiérarchique

Destination	Nœuds suivant
01.2	01.2
01.8	01.8
02.x	02.3

TABLE 3.2 – Table de routage du nœud 01.2

L'avantage d'un tel adressage du réseau est de réduire le nombre d'entrées dans la table de routage dans les réseaux de grande taille. En effet, elle exploite la notion de « localité » des communications. En ce sens beaucoup de communications ont lieu entre des nœuds qui sont relativement proches entre eux. Cependant, le problème de cette organisation est de trouver la meilleure partition du réseau en clusters de sorte à minimiser la taille des tables, ou bien de trouver une passerelle qui supporte le trafic engendré.

3.6.3 Système d'adressage adopté dans GRAPP&S

GRAPP&S adopte un système d'adressage hiérarchique pour ses différents nœuds. En effet, dans le système d'adressage à plat, le volume d'informations à stocker dans les nœuds du réseau augmente de façon considérable avec la taille du réseau, amenant un routage ralentit voir même inconsistant et une explosion des tables de routage.

Pour garantir le service de routage, il faut proposer une solution stable et pouvant fonctionner dans les grands réseaux, formés d'une interconnexion de nombreux groupes et avec des groupes comportant plusieurs membres.

Pour cela le système GRAPP&S adapte pour ses nœuds un mécanisme d'adressage hiérarchique pour optimiser le routage des requêtes. Ce mécanisme de routage hiérarchique empêche l'inondation du réseau en définissant les chemins par lesquels transitent les requêtes. Ces requêtes traversent des nœuds de rôles différent qui partagent le trafic sur diffèrent liens du réseau. Les nœuds disposent d'un modèle d'adressage hiérarchique, qui facilite la montée de l'arbre de GRAPP&S pour atteindre les autres membres du réseau. Le système d'adressage de GRAPP&S est construit en respectant la hiérarchie de l'arbre où l'identifiant de chaque nœud est préfixé par celui de son nœud parent. Les détails de cette construction seront données dans le chapitre suivant.

Un nœud *Communicator* (C) dispose d'un identifiant unique (ID) à partir duquel on va construire les identités des autres nœuds RM et DM de la communauté. L'identifiant du nœud C devient alors le préfixe de tous les nœuds de sa communauté. L'avantage de cet adressage préfixé est de respecter l'organisation hiérarchique des nœuds, qui va nous permettre de remonter facilement l'arbre hiérarchique et de définir les chemins par lesquels transitent les messages. Le nœud *Communicator* ne stocke pas de données ni d'indexation, mais c'est le point d'entrée de sa communauté. Les nœuds *Ressource Manager* disposent eux aussi d'un identifiant unique, composé d'un préfixe : l'identifiant du *Communicator* associé, et d'un identifiant spécifique. Le procédé est reproduit pour les *Data Manager* qui voient leur identifiant préfixé par celui du *Ressource Manager*.

Remarque 3.3. *l'adressage proposé respecte la structure logique hiérarchique de l'architecture de GRAPP&S.*

Le modèle d'adressage de GRAPP&S garantit l'unicité des nœuds et en plus il doit être indépendant des adresses physiques, des nœuds. De ce fait deux nœuds de GRAPP&S, tournant sur n'importe quelles machines du réseau peuvent communiquer entre elles sans se préoccuper de l'architecture physique sous-jacent.

La gestion des identifiants de chaque niveau doit garantir l'unicité des adresses produites, il faut absolument éviter que deux *Ressource Manager* reliés au même *Communicator* ne disposent de la même adresse. Afin d'assurer l'unicité des identifiants, on peut concevoir cet identifiant à partir d'informations spécifiques au processus qui s'exécute : numéro de série du processeur de la machine hôte, adresse MAC de la machine hôte, PID du processus en exécution ... Un calcul mathématique prenant ces différents paramètres permettra d'obtenir un identifiant unique.

3.7 Conclusion

Dans ce chapitre nous avons présenté notre architecture GRAPP&S(GRid APPlications & Services), une architecture multi-échelle pour l'agrégation de données et services. Nous avons décrit le modèle de GRAPP&S, comme la spécification des composants, les différentes possibilités d'indexation des données qu'un nœud RM peut adopter, des mécanismes de détection de pannes et d'élection d'un nœud RM.

Avec l'utilisation des Data Managers, des proxies spécialisés pour l'adaptation et le traitement de différents types de données, il est possible à GRAPP&S d'intégrer de manière transparente aussi les données de type fichier que les bases de données, les flux(audio, vidéo), les services Web et le calcul distribué. Ces données de GRAPP&S sont identifiées par l'identifiant du nœud DM qui la gère, auquel s'ajoute une « extension de type MIME » contenant des informations (par exemple : tag, signature, etc.) et le type MIME des données. Ceci permet de franchir la barrière du simple « nom de fichier », et peut donc faire cohabiter des données statiques(fichiers), des données dynamiques(requêtes sur une base de données, résultats d'un calcul) et des données à caractère temporaire(flux voix ou vidéo, état d'un capteur, etc.).

Différentes stratégies de nommage de nœuds sont étudiées dans ce chapitre avec leurs avantages et inconvénients. Dans GRAPP&S, un système d'adressage hiérarchique des nœuds est adopté. L'organisation hiérarchique de GRAPP&S et son système d'adressage hiérarchique définissent les chemins par lesquels transitent les requêtes de recherche, ce qui empêche l'inondation des liens du réseau.

Le chapitre suivant va concerner les protocoles de routage dans GRAPP&S en s'appuyant sur le système d'adressage hiérarchique pour rechercher et accéder aux données du Framework GRAPP&S.

Chapitre 4

Routage et transfert d'informations dans GRAPP&S

Résumé. *Ce chapitre présente les différentes primitives qui exploitent l'architecture de GRAPP&S, architecture présentée au chapitre précédent.*

Dans un premier temps, nous présentons un mécanisme de routage préfixé pour la recherche et l'accès aux données de GRAPP&S. Ce schéma de routage est basé sur le nommage hiérarchique des différents nœuds de GRAPP&S, ainsi que sur le test des accès possibles entre les différents pairs. La recherche de ressource, ainsi que le rappatriement des ressources, sont également présentés et sont basés sur cet algorithme de routage.

Dans une seconde partie, nous étudions les performances des différentes opérations que sont la recherche et le transfert de données. Cette étude prend en compte les différentes configurations possibles des communications.

Les travaux présentés dans ce chapitre ont fait l'objet d'une publication :

- Thierno Ahmadou Diallo, Olivier Flauzac, Luiz Angelo Steffenel & Samba Ndiaye : Routage préfixé dans grapp&s. *In Proceedings of Colloque Africain sur la Recherche en Informatique et en Mathématiques Appliquées (CARI'2014), pages 239-246.* Saint Louis, Sénégal, October 20-23 2014.

Sommaire

4.1 Motivations et objectifs 74
4.2 Primitives de GRAPP&S 75
4.3 Etude des performances des primitives 86
4.4 Conclusion 92

4.1 Motivations et objectifs

L'Internet moderne est un support majeur du partage de données. Les nouveaux services (les flux vidéos haute définition, de la VoIP) et besoins des utilisateurs qui émergent de l'Internet posent des problèmes d'extensibilité pouvant remettre en question la robustesse des architectures actuelles.

Pour améliorer la robustesse des services offerts par les réseaux, il est nécessaire de développer de nouvelles solutions d'acheminement des messages. Le routage est un ensemble de mécanismes qui désigne, non seulement le calcul des différents routes ou chemins que l'information peut suivre, mais aussi, le transfert des informations selon cette route, ou ces routes, définie(s) ou non à l'avance. En résumé, il s'agit des techniques d'acheminement d'un message depuis un nœud source vers un nœud destination, d'un nœud générateur de données à un nœud consommateur de ces mêmes données.

Dans un système Pair-à-Pair (P2P), les ressources sont distribuées entre tous les pairs. Afin d'accéder à une ressource, les nœuds doivent d'abord chercher ces ressources, obtenant ainsi les coordonnées des nœuds qui les détiennent. Généralement, une clé identifie une ressource dans un système P2P et, selon la manière dont les systèmes P2P organisent ces clés, les mécanismes de recherche et accès peuvent varier.

Dans les P2P non-structurés, comme par exemple Gnutella [Rip01], l'absence d'une organisation des clés (ou d'un nœud d'index) force l'utilisation d'une méthode par inondation lors de la découverte des ressources. Toutefois, la réponse à une requête de recherche peut emprunter un chemin direct vers le nœud demandeur, si cela est possible, c'est à dire si aucune contrainte telle qu'un pare-feu ou une configuration réseau particulière n'empêche un tel envoi.

L'inondation des messages de recherche contribue à impacter fortement le fonctionnement du réseau, voir à risquer de saturer ce dernier. Des alternatives telles que celles proposées par les systèmes comme KaZaA[LKR], Gnutella0.6 [KM02b] reposent sur des *super-pairs*, des nœuds intermédiaires qui permettent la concentration des requêtes et l'élagage de l'arbre de diffusion.

Les réseaux P2P structurés tels que Chord[SMK+01] et Pastry [RD01a], par contre, sont basés sur les DHT(*Distributed Hash Table*). Dans ce cas, l'algorithme de recherche associe un pair p à une clé de ressource k, de manière à ce que ce pair p agisse comme l'index de cette ressource. Ainsi, lorsqu'un nœud q recherche une ressource, il envoie la requête au pair r (parmi la liste de pairs connus à q) avec l'identifiant (ID) le plus proche de k, selon une certaine métrique. Lorsqu'un nœud r reçoit une requête, soit il détient la clé de la ressource et peut renseigner l'adresse où cette ressource est disponible, soit il renvoie le message à un nœud avec un ID plus proche à celui de la clé recherchée. Avec ce mécanisme, la complexité de la recherche dans un réseau P2P structuré de N nœuds est $O(logN)$. Donc, l'avantage d'un système P2P structuré est son efficacité dans la recherche, mais il est plus fragile à la volatilité des nœuds, car cela pourrait entraîner des erreurs de table de routage.

Une caractéristique commune à tous les réseaux P2P, structurés ou pas, est qu'ils ont tous un *design* horizontal, avec un routage à plat (non hiérarchique). Dans ces réseaux, tous les pairs sont identiques dans le sens où tous les pairs utilisent les mêmes règles pour déterminer le routage d'une requête. Cette approche est très différente de celle de l'Internet, où le routage hiérarchique est utilisé.

4.2 Primitives de GRAPP&S

Dans le chapitre précédent, nous avons présenté l'architecture générale de GRAPP&S. Afin d'exploiter cette spécification, il est nécessaire de proposer un ensemble de primitives en mesuse d'exploiter notre architecture. Nous allons maintenant présenter ces différentes primitives.

Nous allons donc proposer notre méthode de routage hiérarchique basée sur l'arbre des comunautés. Ce mécanisme de routage hiérarchique empêche l'inondation du réseau en définissant les chemins par lesquels transitent les requêtes.

Les requêtes traversent des nœuds de rôles différent qui partagent le trafic sur diffèrent liens du réseau. Les nœuds disposent d'un modèle d'adressage hiérarchique, qui facilite la « montée » de l'arbre pour atteindre les autres les membres du réseau. Un autre apport de notre protocole est que le routage ne dépend pas d'une interconnexion point à point entre les différents nœuds, il suffit juste de contacter son père dans la hiérarchie grâce à son propre identifiant pour atteindre la destination.

4.2.1 Retour sur la structuration de GRAPP&S

Dans le chapitre précédent, nous avons donné le principe de l'adressage, nous allons maintenant en donner la teneur.

Dans l'architecture de GRAPP&S, chaque nœud a un identifiant(ID) unique. Les adresses IP ou MAC [1] ne sont pas des identifiants suffisamment précis car ils ne permettent pas d'identifier de manière unique les différents nœuds qui peuvent résider sur une même machine (par exemple, un RM et plusieurs DM).

De plus, l'utilisation des adresses IP ou MAC ne garantit pas une identité unique : les adresses IP privées peuvent être réutilisées, et les adresses MAC peuvent être logiciellement modifiées. On pourra noter que le développement des machines virtuelles accroit le risque de collision d'adresses MAC : chaque administrateur attribue une adresse MAC aux interfaces réseau virtuelles, ceci sans concertation ni protocole spécifique normalisé de création d'adresse.

Nous proposons pour notre système, de nous inspirer de la solution proposée par JXTA[ATA+03] : exploiter une chaine de 128 bits. Chaque nœud dispose ainsi d'une chaîne unique ID_local, sous la forme *"urn :nom-communaute :uuid :chaine-de-bit"*.

L'expression de l'adressage hiérarchique se fait par la concaténation des identifiants(IDs) sous forme de préfixe, c'est-à-dire :

— l'identifiant(ID) du nœud c_i est équivalent à son identifiant local qui ID_local ;
— l'identifiant du nœud RM_i(ID_{RM}) est formé par ID_c_i/ID_RM_i ;
— et l'identifiant du nœud DM_i (ID_{DM}) prend la forme $ID\_c_i/ID\_RM_i/ID\_DM_i$.

L'avantage d'un modèle d'adressage propre à GRAPP&S est que cela le rend indépendant du modèle d'adressage du réseau *overlay* sur lequel GRAPP&S est implémenté. Ainsi, deux communautés GRAPP&S implémentées sur des *middlewares* différents : FreePastry[DEG+01] et Phex[phe14] par exemple, seront toujours compatibles, une fois la connexion établie entre leurs *communicator*. En plus l'adressage hiérarchique permet de remonter facilement la structure de GRAPP&S, en définissant les chemins par lesquels transitent les requêtes. Ce qui va permettre d'empêcher l'inondation des liens de GRAPP&S.

1. Media Access Control

Cette hiérarchie, et ce système de nommage, permettent donc de se passer de tables de routage : le nom même d'un nœud en indique la position.

4.2.2 Routage élémentaire

Toutes nos primitives, qu'il s'agisse de recherche de ressource ou de transfert des données se basent sur un schéma commun et stadard de routage au sein de notre architecture. Nous pouvons donc exploiter un schéma de routage hiérarchique, adapté à GRAPP&S.

Une communauté de GRAPP&S est un arbre T, dont les sommets correspondent à ses différents composants. Si on considère T avec une numérotation des sommets de T suivant un DFS [2][FG01], par construction on obtient donc :

- pour chaque sommet X d'identifiant Id_X l'adresse de X est constituée par la chaine binaire représentant Id_X concaténé par les chaines binaires des sommets parents de X dans T. L'adresse de chaque sommet X est donc constituée par une chaîne binaire $PATH_X$ et d'un entier $Lpath_X$ représentant la longueur de la chaine ;
- pour tout sommet X dans l'arbre T, on considère la variable $Masque_F$ le masque constitué par une chaîne binaire de $Lpath_X + 8$ bits dont les $Lpath_X$ premiers bits sont tous à 0 et les 8 derniers bits sont à 1. Par exemple pour un sommet X tel que $Lpath_X$= 16, son masque sera $Masque_X$= 00000000. 00000000. 11111111.

Cette construction permet d'obtenir les propriétés suivantes :

1. soit Y un sommet de l'arbre T, si Y est un descendant de X dans T, alors $PATH_X$ est une sous chaîne de $PATH_Y$ et en appliquant un ET logique entre $Masque_X$ = 00000000.00000000.11111111 et $PATH_Y$ on a l'identifiant de Y.
2. si Y n'est pas descendant de X dans T, alors il faut passer par le père de X pour aller en Y. Soit *Ancestor*(**X, Y**) la fonction qui pour tout couple de sommet retourne VRAI ou FAUX suivant que X est parent de Y dans l'arbre T ou pas. L'architecture de GRAPP&S est hiérarchique et constituée de trois niveaux. Ce qui limite la taille des adresses des nœuds, à une taille raisonnable, même dans un réseau de grande taille.

La méthode d'expédition des messages dans notre système hiérarchique est donc inspirée de celle proposé par [BvLT93]. L'algorithme de routage de base (voir Algorithme 1), permet la transmission de messages entre deux sommets, en connaissant l'identifiant du nœud source et celui du nœud destinataire, et évidemment, sans avoir à calculer de tables de routage.

Les fonctionnalités principales de l'architecture de GRAPP&S sont la recherche et l'accès aux données et services. La recherche dans GRAPP&S exploitent le routage hiérarchique qui s'appuie sur les identifiants des nœuds(adressage préfixé) qui définissent les chemins par lesquels transitent les requêtes de recherches.

Comme nous allons le voir par la suite, l'accès aux données et services ne dépend pas d'une connexion directe entre les nœuds. Dans GRAPP&S, il est toujours possible de rappatrier une information par routage dans le cas ou la connexion directe entre les nœuds est impossible.

2. Depth First Search

Algorithme 1 : Methode de routage de base

```
1  m : message a envoyer à la destination ;
2  local_address : adresse du nœud local ;
3  destination : nœud destination ;
4  add : Adresse d'un nœud ;
5  Procédure Route (m [, destination])
6  si destination ≠ ∅ alors
7  |  si canReach (destination) alors
8  |  |  add ← getAdd (destination) ;
9  |  |  Send(m, add, local_address) ;
10 |  sinon
11 |  |  add ← getNextHop (local_address, destination) ;
12 |  |  Send(m, add, local_address) ;
13 |  finsi
14 sinon
15 |  add ← father ;
16 |  Send (m, add, local_address) ;
17 finsi
```

4.2.3 Recherche de ressources

Parmi les fonctionnalités principales de GRAPP&S, il y a la recherche et l'accès aux données. Cette section décrit notre algorithme de recherche de données dans une communauté de GRAPP&S.

Les différents nœuds qui interviennent dans cette recherche disposent de quelques fonctions de base, certaine exposées précédemment, sur lesquelles nous allons nous appuyer pour définir nos algorithmes :

Send permet à un nœud source d'envoyer un message à un nœud destination. Cependant, cette primitive nécessite une connexion directe entre les nœuds, raison pour laquelle elle est associée à la primitive *Route* ;

Route permet à un nœud source de transmettre un message à un nœud quelconque, soit en l'envoyant directement grâce à *Send*, soit en relayant l'information à un nœud avec le mécanisme de routage préfixé. Afin d'obtenir une adresse dans le mécanisme de routage préfixé, on utilise une méthode $getNextHop()$ qui effectue la comparaison des adresses et applique les masques de sous-réseau afin d'obtenir l'adresse du prochain nœud en direction de la destination.

Receive qui permet de traiter les messages reçus, selon leurs types et selon le rôle joué par le nœud (voir algorithme 2).

Recherche_locale qui permet à un DM de répondre à une recherche de ressource.

Lors de la recherche, les messages peuvent être de deux types :

1. message de recherche de type :

$$< "req", contenu, source >$$

2. message de réponse de type :

$$< "reply", contenu, source >$$

Quand un client cherche une donnée sur GRAPP&S, il entre en contact avec un proxy DM_i, qui envoie une requête Y contenant les caractéristiques de la donnée et le type `req`. Donc la requête Y prend la forme $< "req", contenu, source >$. Comme le message n'a pas un destinataire précis, la recherche dans une communauté de GrAPP&S se fait par paliers, en respectant l'organisation hiérarchique du réseau.

La procédure de recherche est comme suit :

1. $DM_i \in C_i$ envoie la requête Y a son nœud $RM_i \in C_i$ grâce à $Route()$ (voir algorithme1).
2. RM_i reçoit la requête Y selon son type (voir algorithme2) et vérifie grâce à la méthode $Recherche\_locale(m)$ (voir algorithme 3) si dans son index il y'a un voisin DM qui contient la donnée recherchée

   (a) si oui, alors le nœud RM_i retourne au nœud DM_i une liste de nœuds $\{DM\}$ qui contiennent l'information recherchée, sous la forme d'un message Y' avec le type `reply`. La forme de la requête Y' est $< "reply", contenu, source >$.

   A ce niveau de réussite de la recherche, on peut dire qu'on a une recherche locale similaire à Napster[FP99b], FastTrack[LKR06], BitTorrent[Coh08b].

   L'illustration est donnée dans la Figure 4.1 où le nœud $DM_i \in C_i$ d'identifiant AAA envoie la requête Y de type `req` a son nœud $RM_i \in C_i$ grâce à la fonction $Route()$. Le nœud RM_i d'identifiant AA réplique en envoyant un message de type $reply$ contenant l'identifiant du nœud DM_y qui contient la donnée recherchée.

   (b) sinon, le nœud RM_i fait suivre la requête à son nœud $c_i \in C_i$ pour une diffusion aux autres nœuds $RM_k \in C_i$ tel que les $RM_k \neq RM_i$(Algorithme 5). Chacun de ces nœuds RM_k exécutent la méthode $Recherche\_locale$(m) (voir algorithme 3) pour vérifier si dans son indexe local il y'a un voisin DM qui contient la donnée recherchée,
      — Lorsqu'un un nœud $RM_k \in C_i$ trouve une correspondance, une réponse estampillée `reply` de ce message avec la forme $< "reply", contenu, source >$ sera retournée au nœud DM_i émetteur en prenant le chemin inverse à aller grâce à $Route()$(voir Algorithme1).

3. Si aucun des nœuds $RM_k \in C_i$ ne trouve une correspondance, et si la communauté C_i est connectée à d'autres communautés, alors le c_i fait suivre la requête selon les politiques d'accès définies et mises, ou négociées, en place dans chaque communauté.

   À cette étape de recherche, on fait une recherche par inondation comme Gnutella[Rip01], inondation sur l'arbre des communautés. Cette recherche par inondation est illustrée dans la Figure 4.2 ou le nœud communicator c_i d'identification A diffuse dans la communauté C_i à tous les $RM_k \in C_i$ le message Y de type Req. Le nœud RM_k d'identifiant $BB \in C_j$ trouve une correspondance et répond au nœud DM_i d'identifiant AAA en prenant le

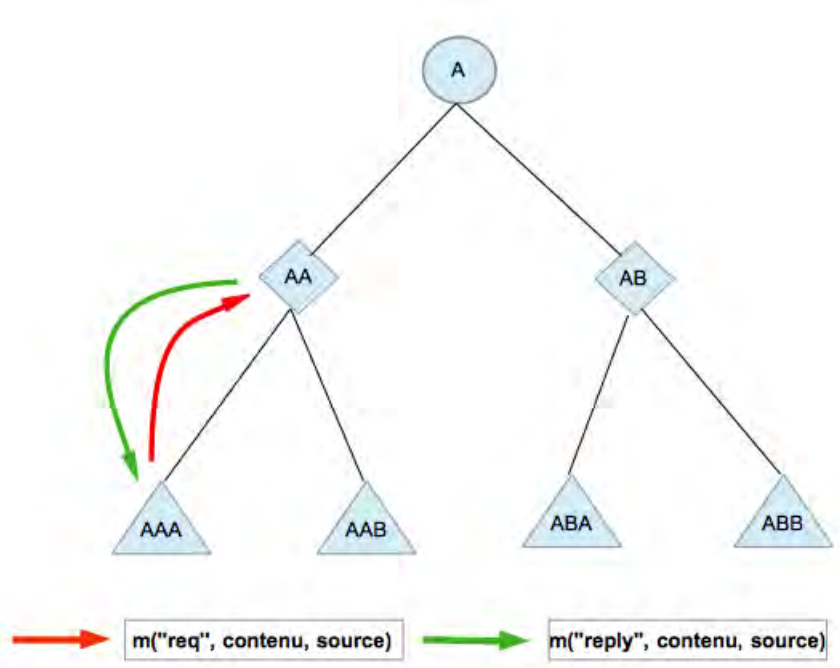


FIGURE 4.1 – Recherche locale dans GRAPP&S

chemin inverse avec un message de type *reply* contenant l'identifiant du nœud DM_y possesseur de la donnée. On pourra toutefois notter que l'inondation est limitée à la structure des communautés GRAPP&S, et non à la structuration du réseau pair-à-pair sous jacent.

Voici donc maintenant les différents algorithmes des fonctions présentées précédemment. D'abord la gestion spécifique des messages en fonction des différents types de nœuds.

Algorithme 2 : Traitement des messages selon les rôles des nœuds

```
1  Type m : message
2  ; Type :rôle du nœud(DM, RM, c);
3  Type m.type : type de message(req | reply | get | data);
4  Procédure Receive( m)
5  si (role=DM) alors
6  |  si (m.type=reply) alors
7  |  |  m'←("get", m.data, local_address)
8  |  |  Route(m', m.source);
9  |  finsi
10 |  si m.type = get alors
11 |  |  m'←("data", data, local_address)
12 |  |  Route(m', m.source) ;
13 |  finsi
14 finsi
15 si (role=RM) alors
16 |  si m.type=req alors
17 |  |  si recherche_locale(m)=FAUX alors
18 |  |  |  si origin=child alors
19 |  |  |  |  Route(m, father) ;
20 |  |  |  finsi
21 |  |  finsi
22 |  sinon
23 |  |  Route(m, m.destination) ;
24 |  finsi
25 finsi
26 si (role=C) alors
27 |  si m.type=req alors
28 |  |  Diffusion(m) ;
29 |  sinon
30 |  |  Route(m, m.destination) ;
31 |  finsi
32 finsi
```

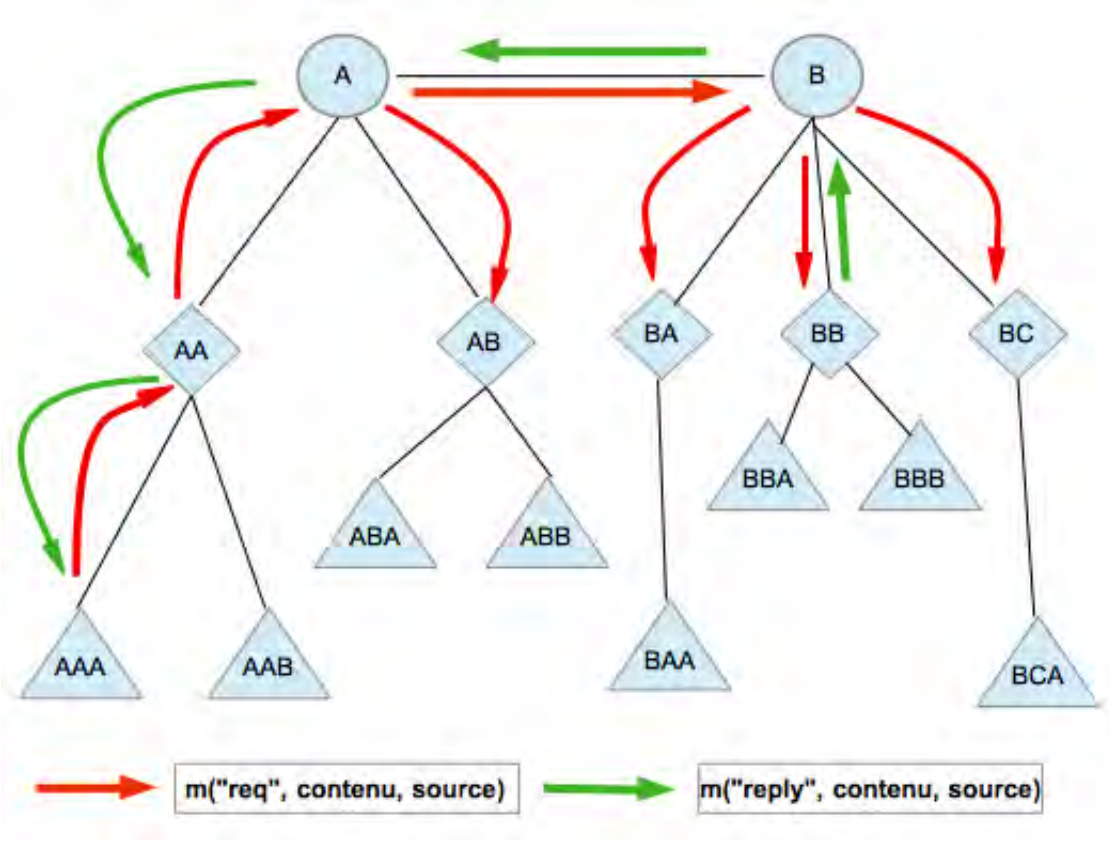


FIGURE 4.2 – Recherche dans le réseau GRAPP&S

Puis l'algorithme de Recherche locale d'une ressource.
Maintenant l'algorithme de test et de vérification de l'index du *Ressource Manager*

Algorithme 3 : Recherche locale

```
1  Type m : message de recherche ;
2  Type m.source : adresse du DM qui a émis le message ;
3  Type m' : message de réponse ;
4  Type local_address : adresse du nœud local ;
5  Fonction Recherche_locale ( m) : booléen
6  listeDM ← RM.Verif_of_RM(m.data)
7  si listeDM ≠ ∅ alors
8  |  pour (chaque DM_k de RM tel que DM_k ∈ listeDM) faire
9  |  |  m'← ("reply", m.data, DM_k)
10 |  |  Route(m', m.source) ;
11 |  fin
12 |  return VRAI
13 sinon
14 |  return FAUX
15 finsi
```

Algorithme 4 : Vérification de l'index *RM*

```
1  Type Index_of_RM[0...n] : index du nœud RM ;
2  Type X ←{} : liste de DM ;
3  Type TypeMime : Type MIME de l'information ;
4  Type TypeMimeRec : Critère de recherche ;
5  Fonction Verif_of_RM(TypeMimeRec) : X ←{}
6  pour (i allant de 0 à n) faire
7  |  if Index_of_RM[i].TypeMime=TypeMimeRec then
8  |  |  X1 ← DM_i ;
9  |  end
10 fin
11 return X ;
```

Enfin l'algorithme assurant la diffusion des messages aux *Ressource Manager*

Algorithme 5 : Diffusion de message aux RMs

```
1 Type m : message a diffuser ;
2 Type origin : adresse du RM qui a transmis le message ;
3 Type local_address : adresse du nœud local ;
4 Procédure Diffusion( m, origin)
5 pour (chaque fils RM_k de c_i tel que RM_k ≠ origin) faire
6 |  Route(m, RM_k)
7 fin
```

Ce mécanisme de recherche hiérarchique empêche l'inondation des liens du réseau. La hiérarchisation permet de définir des chemins par lesquels transitent les requêtes et comme la connectivité logique de notre architecture est par définition de $(n-1)$, il suffit d'appliquer un algorithme de type PIF[BDPV99] (Propagation of Information Feeback) pour agréger les requêtes et réduire le nombre de messages d'une recherche.

4.2.4 Rappatriement des ressources

En cas de réussite lors de la recherche, le client (DM_x) obtient l'identifiant du nœud DM_y qui détient la donnée. Pendant l'étape de transfert de données, les messages peuvent prendre deux formes différentes

— message de demande de contenu

$$< get, contenu, source >$$

— message de transfert de contenu

$$< data, contenu, source >$$

Lorsqu'un nœud reçoit un message de type $< data, contenu, source >$, s'il n'est pas la cible de ce message, alors il retransmet le message. Si ce dernier lui est destiné alors il décide soit de visualiser le contenu du message ou bien de le stocker grâce a une primitive $Save()$ sur un support tels que : sur disque, sur un service *cloud* du type *Dropbox* ou *Seafile*, sur un service de messagerie du type de *Gmail* etc.

Dans ce cas, il a deux possibilités pour entrer en contact avec DM_y et récupérer la donnée : (i) par une connexion directe ; (ii) par une connexion routée si la connexion directe n'est pas possible. Dans les deux cas, on fera appel à la primitive $Route()$. Si la connexion directe est possible, l'envoi se fera par $Send()$, dans le cas contraire il se fera par le mécanisme de routage hiérarchique.

Remarque 4.1. *Comme la ressource a été trouvée par une recherche et que sa localisation a été retournée au site ayant initié la requête, il existe donc un chemin bidirectionnel exploitant les liens de l'architecture hiérarchique de GRAPP&S, utilisable entre le requérant et la ressource.*

Cas d'une connexion directe possible

Si une connexion directe est possible entre le nœud requérant et le possesseur de la ressource, le nœud DM_x envoie directement la requête de demande $< get, contenu, source >$ au nœud DM_y par une simple méthode $Send()$(voir Algorithme 1). Ce dernier DM_y répond par le message de transfert $< data, contenu, source >$ en suivant le chemin inverse à l'aller(voir Algorithme 1).

C'est l'exemple des nœuds qui se trouvent dans le même sous réseau. Une illustration est donnée dans la Figure 4.3, ou le nœud DM_x d'identifiant AAA envoie directement au nœud DM_y d'identifiant AAB, une requête pour accéder au donnée. Ce dernier répond en suivant le chemin inverse. Alors le nœud DM_x peut visualiser la donnée ou bien le stocker sur un support tel qu'un disque, sur un service cloud, sur Gmail etc., grâce à la fonction $save()$.

Remarque 4.2. *Deux nœuds d'identifiant ABC et DEF peuvent très bien faire parti du même réseau, et faire parti, comme leurs identifiants le montrent de deux communautés GRAPP&S. Dans ce cas, une connexion direct sera alors possible.*

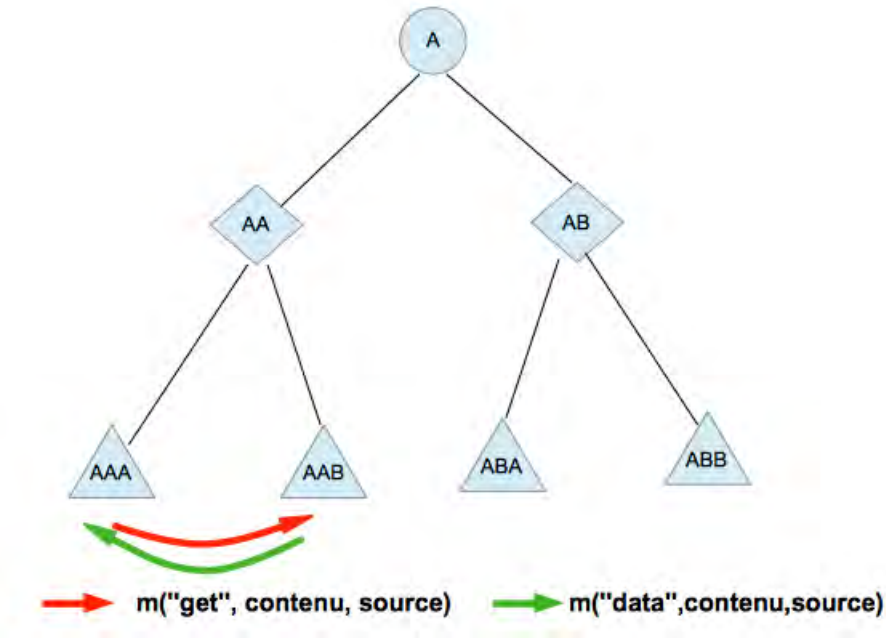


FIGURE 4.3 – Rapartriement dans le cas d'un réseau local dans GRAPP&S

Pour une connexion directe impossible

Si la connexion directe n'est pas possible, alors l'envoie de message se fait par routage hiérarchique dans GRAPP&S. Le mécanisme de routage hiérarchique est apliqué, et donc la ressource suivra le chemin emprunté par la requête de recherche et sa réponse associée.

Dans ce cas, les opérations réalisées sont les suivantes

1. le client DM_x envoie une requête de type `get` à son RM, qui va retransmettre le message `get` grâce à sa primitive *Route*() vers son parent le nœud communicator C_i.
2. le nœud C_i, selon le routage préfixé(voir Algorithme 1), envoie la requête `get` vers le nœud RM qui a indexé la donnée.
3. Ce dernier, par routage préfixé(voir Algorithme 1), fait suivre la requête `get` vers le nœud DM_y responsable de la donnée.
4. le nœud DM_y, qui détient la donnée peut la transmettre avec la primitive *Route*()(voir Algorithme 1), grâce à un message de type `data` en suivant le chemin inverse à l'aller. Alors le nœud DM_x peut visualiser la donnée ou bien la stocker sur un support quelconque, grâce à la fonction *save*().

La Figure 4.4 illustre cette connexion routée, oú le nœud DM_x d'identifiant AAA envoie le message de demande `get` en utilisant sa primitive `Route()` qui remonte l'arbre pour télécharger la donnée gérée par le nœud DM_y d'identifiant BBA. Ce dernier le nœud DM_y de la même manière, lorsqu'il reçoit un message de type `get`, le nœud DM_y qui détient la donnée peut la transmettre avec la primitive *Route*(), grâce à un message de type `data`.

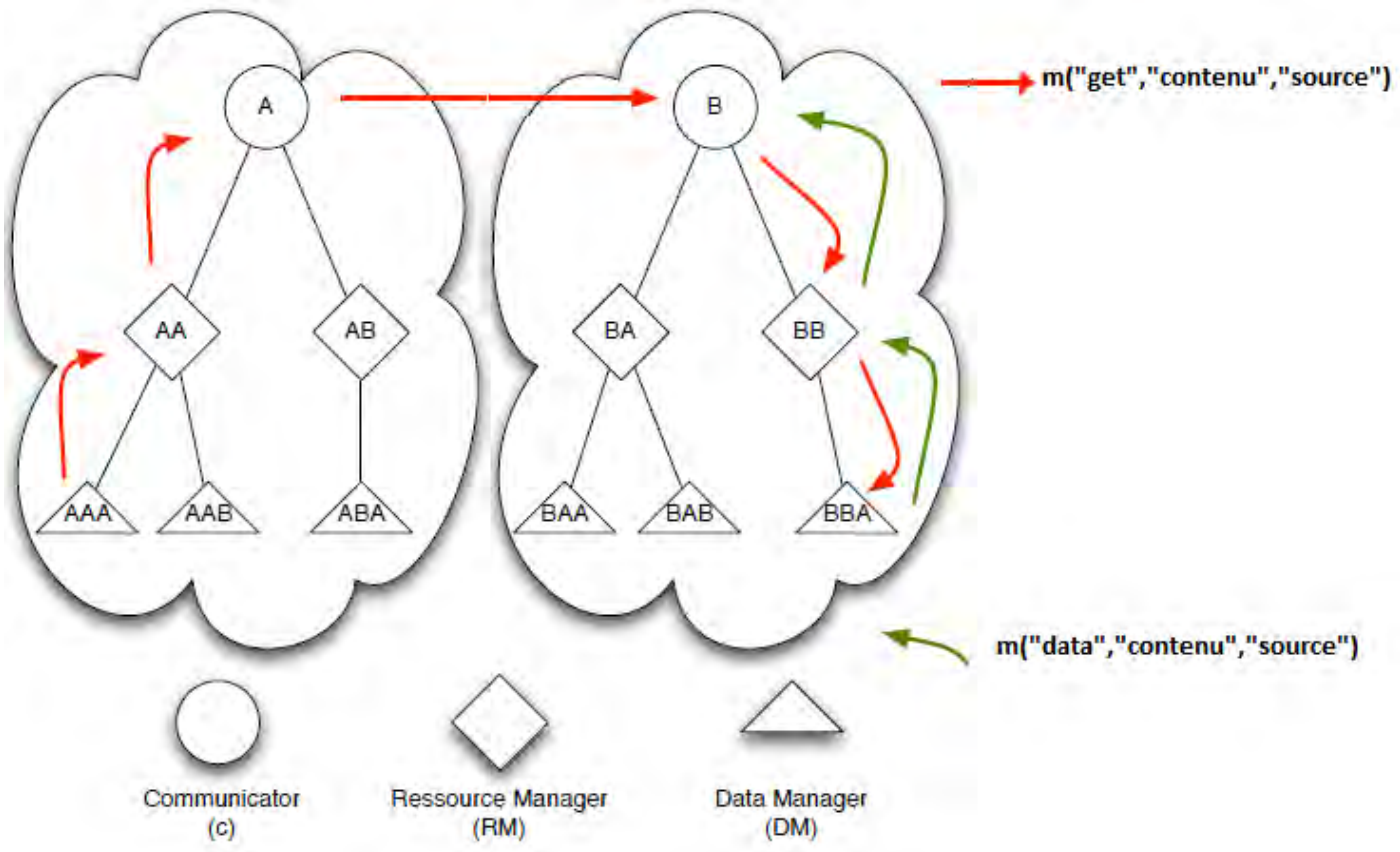


FIGURE 4.4 – Rapatriement de données par une connexion routée

4.3 Etude des performances des primitives

Nous allons dans cette section, réaliser une étude des performances des différentes primitives, en fonction des types de communication entre les acteurs, et des positions relatives du requérant et de la ressource.

4.3.1 Coût des communications

Le problème de communication bidirectionnelle est absolument crucial dans tous les systèmes Pair-à-Pair. Le *framework* GRAPP&S dont les nœuds acceptent des connexions externes peut être confronté aux limites de connexions liées aux pare-feux ou aux configurations réseau particulières. Ce problème concerne d'abord le système de recherche dans GRAPP&S. En effet, il peut se trouver qu'un nœud soit protégé par un pare-feux, ou dans un réseau privé, dans ce cas toute requête en direction de ce dernier est bloquée. Ce qui va bloquer le système de recherche de données mis en œuvre dans GRAPP&S.

Pour contourner le problème des pare-feux des techniques sont utilisées par les solutions Pair-à-Pair telle que Gnutella[Rip01] qui prévoit un mécanisme qui permet à l'émetteur de la requête de recontacter le possesseur du document par le système de mise en relation en lui demandant d'exécuter lui-même une requête http, de type `put`, en direction du destinataire. Ce mécanisme appelé *push* inverse les rôles pour la transmission du document. Bien évidemment si l'émetteur de la requête du document est lui-même protégé par un pare-feu, ou dans un réseau privé, la transmission ne sera pas possible. C'est là une des limites intrinsèques des systèmes de transport de documents par connexion directe.

Dans tous les cas de figure, communications bi-directionnelles, uni-directionnelles ou communications directes impossibles entre les acteurs, GRAPP&S offre des mécanismes permettant d'assurer la communication et le routage des messages, depuis la source jusqu'à la destination. Les modes *pull*, *push*, et le mode « file de messages », permettent de gérer toutes les configurations de la sécurité et du réseau.

Voici donc l'évaluation des différents coûts de communications entre deux nœuds en fonction de leurs relations de communication.

Dans la suite, nous utiliserons les notations suivantes :

— t_i : temps de connexion d'un nœud i ;
— t_{m_i} : le timeout du nœud i ;
— $traitement_i$: temps de traitement du nœud i ;
— $traitement - file$: temps de traitement d'une file de message.

Dans la suite, nous donnons selon les différentes configurations, ainsi que les différents types de connexion, connecté et non connecté, T l'estimation du temps de communication nécessaire. Au mode connecté est associé un mode de communication synchrone, au mode non connecté un mode asynchrone, nécessitant la lecture régulière d'un registre local de réception.

Coût d'un échange en environnement de type $A \leftrightarrow B$

Dans cette configuration, chaque acteur peut joindre l'autre de manière directe. Le coût est évalué en mode connecté et non connnecté.

C'est évaluation concerne la communication dans le sens $A \rightarrow B$.

— En mode connecté

$$T = t_A + traitement_B + t_B$$

— En mode non connecté

$$T = t_A + t_m B + traitement_B + t_B + t_m A$$

Coût d'un échange en environnement fermé $A \nleftrightarrow B$

Dans ce cas, aucun acteur ne peut joindre l'autre directement. Pour établir la communication, on utilise un proxy ou file de message. Le coût est évalué en mode connecté et non connnecté.

— En mode connecté

$$T = t_A + traitement - file + t_B + traitement_B + t_B + traitement_{file} + t_A$$

— En mode non connecté

$$T = t_A + t_m B + t_+ traitement_{file} + t_B + traitement_B + t_B + t_m A + t_A + traitement - file + t_A$$

Remarque 4.3. *dans ce cas, le coût de la communication augmente largement, car chaque acteur est à la fois à l'initiative de l'émission d'une information, mais aussi de la réception d'une information qu'il doit aller chercher dans la file de message ou proxy.*

Coût d'un échange dans un environnement de type $A \to B$

Un acteur peut joindre l'autre directement, mais la réciproque n'est pas vraie. L'initiative de toute communication est donc laissée au nœud ayant la capacité de communication. Le coût est évalué en mode connecté et non connnecté.

L'évaluation proposée concerne la communication dans le sens $B \to A$ car dans le sens $A \to B$ l'émission d'un message ne pose aucun problème.

— En mode connecté (TCP)

$$T = t_B + t_B + traitement_B + t_B$$

— En mode non connecté (UDP)

$$T = t_B + traitement_B + t_m B + t_B + traitement_{file} + t_m A + t_A + t_A$$

Remarque 4.4. *Dans ce cas, le mode non connecté exploite dans le sens de communication non autorisé une file de messages.*

Remarque 4.5. *Naturellement, si aucune file de message n'est disponible, il n'est pas possible de communiquer.*

4.3.2 Performance de la recherche

Notre étude va être menée selon 4 hypothèses de communications. Si on considère A et B deux nœuds tels que A est père de B dans GRAPP&S, les cas d'études sont :

1. organisation des communications bidirectionnelles :

$$A \longleftrightarrow B$$

2. organisation des communications directes impossible entre les acteurs

$$A \nleftrightarrow B$$

3. organisation des communications unidirectionnelle de type *push* ;

$$A \longrightarrow B$$

4. organisation des communications unidirectionnelle de type *pull* ;

$$A \longleftarrow B$$

Recherche interne

Voici donc les évaluations des coûts liés à une recherche interne.

1. Communications bi-directionnelles : $A \longleftrightarrow B$
   - En mode connecté
     Pour la montée $DM \rightarrow RM \rightarrow C$

$$T = t_DM + traitement_{RM} + t_{RM} + traitement_c$$

     Pour la descente $C \rightarrow RM \rightarrow DM$

$$T = t_c + traitement_{RM} + t_{RM} + traitement_{DM}$$

   - En mode non connecté
     Pour la montée $DM \rightarrow RM \rightarrow C$

$$T = t_DM + t_{mRM} + traitement_{RM} + t_{RM} + t_mc + traitement_c$$

     Pour la descente $C \rightarrow RM \rightarrow DM$

$$T = t_c + t_mRM + traitement_{RM} + t_{RM} + t_mDM$$

2. Communications directes impossible entre les acteurs : $A \nleftrightarrow B$
   - En mode connecté
     Pour la montée $DM \rightarrow RM \rightarrow C$

$$T = t_{DM} + traitement_{file} + t_{RM} + traitement_{RM} + t_{RM} + traitement_{file} + t_c + traitement_c$$

Pour la descente $C \rightarrow RM \rightarrow DM$

$$T = t_c + traitement_{file} + t_RM + traitement_{RM} + t_{RM} + traitement_{file} + t_DM + traitement_{DM}$$

— En mode non connecté
Pour la montée $DM \rightarrow RM \rightarrow c$

$$T = t_DM + t_mRM + t_RM + traitement_{file} + t_RM + traitement_{RM} + t_{RM} + t_{mc} + t_c + traitement_{file} + t_c + traitement_c$$

Pour la descente $C \rightarrow RM \rightarrow DM$

$$T = t_c + t_{mRM} + t_{RM} + traitement_{file} + t_{RM} + traitement_{RM} + t_{RM} + t_{mDM} + t_{DM} + traitement_{file} + t_{DM} + traitement_{DM}$$

3. Communications uni-directionnelle de type $push : A \longrightarrow B$
— En mode connecté
Pour la montée $DM \rightarrow RM \rightarrow C$

$$T = t_DM + traitement_RM + t_RM + traitement_c$$

Pour la descente $C \rightarrow RM \rightarrow DM$

$$T = t_c + traitement_{file} + t_{RM} + traitement_{RM} + t_RM + traitement_{file} + t_DM + traitement_{DM}$$

— En mode non connecté
Pour la montée $DM \rightarrow RM \rightarrow C$

$$T = t_DM + t_mRM + traitement_RM + t_{RM} + t_mc + traitement_c$$

Pour la descente $C \rightarrow RM \rightarrow DM$

$$T = t_c + traitement_{file} + tmRM + t_{RM} + traitement_{RM} + t_{RM} + traitement - file + t_mDM + t_{DM} + traitement_DM$$

4. Communications uni-directionnelle de type *pull* : $A \longleftarrow B$

   (a) En mode connecté
   Pour la montée $DM \rightarrow RM \rightarrow C$

   $$T = 2 * t_{RM} + traitement_{RM} + 2 * t_c + traitement_c$$

   Pour la descente $C \rightarrow RM \rightarrow DM$

   $$T = t_c + t_{RM} + traitement_{RM} + t_DM + traitement_{DM}$$

   (b) En mode non connecté
   Pour la montée $DM \rightarrow RM \rightarrow C$

   $$T = 2 * t_{RM} + t_{mRM} + traitement_{RM} + 2 * t_c + t_{mc} + traitement_c$$

   Pour la descente $C \rightarrow RM \rightarrow DM$

   $$T = t_c + tm_{RM} + t_{RM} + traitement_{RM} + t_{mDM} + traitement_{DM}$$

Recherche inter-communauté

La recherche inter-communauté consiste à étendre les résultats précédents, en ajoutant la communication entre les différents nœuds communicator. Afin de rendre le plus générique possible notre solution, nous considérons que les communicateurs communiquent entre eux par l'intermédiaire d'une file de messages.

Remarque 4.6. *Le cas de la file de messages est un cas défavorable en ce qui concerne le coût des communications et nous permet d'obtenir une borne supérieure.*

— En mode connecté

$$T = t_c + traitement_{file} + t_c + traitement_c + t_c + traitement_{file} + t_c$$

— En mode non connecté

$$T = t_c + t_{mc} + t_c + traitement_{file} + t_c + traitement_c + t_c + t_{mc} + t_c + traitement_{file} + t_c$$

On peut aisément, par extension, évaluer le nombre de messages à partir des diféfretes évaluations du temps, en affectant 0 aux temps de traitement.

4.3.3 Performance du Rapatriement des données

Les impératifs de communication impactent aussi le rapatriement de données dans GRAPP&S. Il peut arriver qu'un nœud DM qui détient des données ou bien qu'un réseau local formé de RM, des DM soit protégé par un pare-feu, ou dans une architecture réseau incompatible avec des communications directes. Alors toute requête en direction de ce nœud ou ce réseau sera bloquée. Le Framework GRAPP&S implémente des mécanismes pour contourner ce problème.

Ce contournement a un coût. Nous allons évaluer ce coût selon les possibilités de connexion entre deux nœuds. Le rapatriement de données se fait suivant deux possibilités : soit par une connexion directe ou par une connexion routée.

1. dans le cas où la connexion est directe entre les deux nœuds concernés respectivement le demandeur et le possesseur du document, le temps de communication est égal à :

$$T = tc_{DM_x} + tc_{DM_y} + t_{traitement_{DM_y}}$$

dans l'expression précédente, tc_{DM_x}, tc_{DM_y} sont les temps de connexion respectivement du nœud DM_x sur le nœud DM_y et du nœud DM_y sur DM_x, $t_{traitement_{DM_y}}$ est le temps de traitement du nœud DM_y.
Si on considère que tous les temps sont les mêmes $tc_{DM_x} = tc_{DM_y} t_{traitement_{DM_y}} = t$ on obtient en nombre de messages :

$$T = 3 * t$$

2. dans le cas où la connexion directe est impossible, on procède par routage hiérarchique
   - si la communication entre tous les nœuds est de type $A \leftrightarrow B$:
     - pour un transfert local on aura le nombre de messages `get` : 2*t et le nombre de messages `data` égal à $2 * t$
     - pour un transfert dans la communauté de GRAPP&S on obtient le nombre de messages `get` égal à $4 * t$ et le nombre de messages `data` égal à $4 * t$.
   - si la communication entre tous les nœuds de l'arbre est de type $A \nleftrightarrow B$:
     - pour un transfert local on obtient le nombre de messages `get` égal à $6 * t$ et le nombre de messages `data` égal à $6 * t$.
     - Pour un transfert dans la communauté de GRAPP&S on obtient le nombre de messages `get` égal à $12 * t$ et le nombre de messages `data` égal à $12 * t$
   - si la communication est uni-directionnelle, de type $A \rightarrow B$ ou $A \leftarrow B$
     (a) Pour le cas $A \rightarrow B$
     pour un transfert local on obtient le nombre de messages `get` égal à $6 * t$ et le nombre de messages `data` égal à $2 * t$.
     pour un transfert dans la communauté de GRAPP&S on obtient le nombre de messages `get` égal à $12 * t$ et le nombre de messages `data` égal à $4 * t$.
     (b) Pour le cas $A \leftarrow B$
     Pour un transfert local on obtient le nombre de messages `get` égal à $8 * t$ et le nombre de messages `data` égal à $2 * t$.
     pour un transfert dans la communauté de GRAPP&S on obtient le nombre de messages `get` égal à $16 * t$ et le nombre de messages `data` égal $0 * t$.

4.4 Conclusion

Dans ce chapitre nous avons présenté les différentes primitives de GRAPP&S : la recherche et l'accès aux données. L'organisation hiérarchique de GRAPP&S et son système d'adressage hiérarchique définissent les chemins par lesquels transitent les requêtes de recherche. Ce qui évite l'inondation des liens du réseau. L'accès aux données de GRAPP&S ne dépend pas uniquement d'une connexion directe entre les nœuds. Il est toujours possible par routage de rapatrier les données par le chemin utilisé lors de la recherche, au cas où une connexion directe entre les nœuds n'est pas possible.

Nous avons évalué le coût des algorithmes de recherche et de rapatriement de données.

Le Framework GRAPP&S pourrait servir à connecter les instituts d'enseignement, intéressés par le partage de ressources à travers une mise en commun des données éducatives de chaque institut sous forme de communauté.

Face aux problématiques des solutions de partage basées sur le cloud où le stockage des données est distant et la vitesse d'accès lente, GRAPP&S devient une solution décentralisée qui permettra un accès rapide aux ressources du fait de la proximité des données et des consommateurs. Ce domaine d'application de GRAPP&S qu'est le E-learning sera étudié dans le chapitre 5.

Chapitre 5

Perspective d'utilisation du framework GRAPP&S dans le domaine du e-learning

Résumé. *Dans ce chapitre, nous étudions le domaine d'application du Framework GRAPP&S qui est E-learning. GRAPP&S se présente comme une solution E-learning décentralisée pour la gestion et le partage de tout type de ressources éducatives. Ces ressources éducatives se présentent sous forme de fichiers, de flux de données(vidéo, VoIP, audio), des données issues des capteurs, des services distants (cloud, messagerie), etc.*

GRAPP&S regroupe les ressources éducatives de chaque institut sous forme de « communauté » et permet l'interconnexion et le partage entre ces différentes communautés. Les ressources éducatives sont gérées de façon transparente à travers des proxies spécialisés pour chaque ressource de chaque communauté.

L'utilisation de la communauté de GRAPP&S peut être étendue au sein d'un institut. Par exemple, on peut avoir une sous communauté pour la partie administrative, séparée de la partie éducation, avec des règles de sécurité et de protection des ressources. Le Framework GRAPP&S définit des politiques de sécurité pour la protection des données, tant au sein de la communauté et entre les différentes communautés.

Les travaux présentés dans ce chapitre ont fait l'objet de deux publications :

— Thierno Ahmadou Diallo, Olivier Flauzac, Luiz Angelo Steffenel, Samba NDiaye & Youssou DIENG : GRAPP&S : A Distributed Framework for E-learning Resources Sharing *5th International IEEE EAI Conference on e ?Infrastructure and e ?Services for Developing Countries*. Blantyre, Malawi, novembre 2013.

— Thierno Ahmadou Diallo, Olivier Flauzac, Luiz Angelo Steffenel, Samba NDiaye & Youssou DIENG : GRAPP&S, a Peer-to-Peer Middleware for Interlinking and Sharing Educational Resources *EIA, Transactions on Industrial Networks and Intelligent Systems, 2(3), mai 2015.*

Sommaire

5.1 Le *E-learning* dans le cadre des pays en développement 94
5.2 Les outils existants 94
5.3 Conception d'une application de *E-learning* basée sur GRAPP&S 100
5.4 Conclusion 104

5.1 Le *E-learning* dans le cadre des pays en développement

Comme le précise l'UNESCO dans ses *Principes directeurs pour l'inclusion*, l'intégration pédagogique des Technologies de l'information et de la communication (TIC) à tous les niveaux d'enseignement améliore notablement la qualité des systèmes éducatifs. L'apprentissage en ligne ()*E-learning*), peut non seulement contribuer à consolider une planification et une gestion de l'éducation démocratique et transparente, mais aussi à amplifier l'accès à l'apprentissage, améliorer la qualité et assurer l'inclusion.

Grâce aux opportunités qu'elles offrent en termes d'utilisation ou d'adaptation, notamment dans les environnements où les ressources sont insuffisantes, les ressources éducatives libres constituent une excellente occasion d'atteindre l'objectif d'une éducation de qualité pour tous. On se pose alors une question légitime : pourquoi le niveau d'utilisation des TIC à des fins éducatives est encore si bas dans les pays en développement ?

La réponse à cette question se trouve entre autres sur des problèmes techniques ou d'infrastructure qui doivent être surmontés. Les plus courants sont les questions telles que le manque de fiabilité des connexions à Internet et des lignes téléphoniques, la lenteur de l'accès aux sites Internet due à l'étroitesse de la bande passante et le nombre limité d'ordinateurs connectés, etc. Un défi majeur est donc de réduire au maximum les contraintes technologiques afin d'augmenter la participation des pays en développement, ce qui constitue les motivations de notre Framework GRAPP&S.

5.2 Les outils existants

Aujourd'hui, le Web est passé d'un simple référentiel de documents à un puissant moyen de communication. Ce changement a apporté des améliorations dans le domaine de l'éducation avec l'apparition des applications Web qui comprennent souvent des sessions de collaboration[AJ13].

Cette collaboration est devenue beaucoup plus souple et efficace avec le développement rapide de l'Internet. Les technologies basées sur le Web ont permis aux gens qui sont situés dans des endroits différents de communiquer les uns avec les autres de façon synchrone et asynchrone, qui peuvent alors soutenir des activités d'apprentissage collaboratif (travail en commun). Les technologies de travail collaboratif sont destinées à aider les groupes de personnes qui travaillent ensemble.

Dans le domaine de l'enseignement, les solutions collaboratives permettent aux enseignants et aux apprenants de partager des données, de communiquer par courriel, chat-rooms, de partager des espaces de travail, des forums, etc..

5.2.1 Solutions E-learning basées sur le Cloud

Avec l'avènement de l'Internet, la pluspart des médias se présentent sous la forme numérique, qu'il s'agisse de livres, courriers, les flux vidéo(vidéoconférence, chaine de télévision), flux audio comme la Voix sur IP, bref toutes les diffusions de flux continus. Tous ces médias numériques se mettent au service de l'éducation.

Pour rendre l'éducation plus efficace, un nombre croissant d'écoles ont mis en place des plate-formes *e-learning*. Ces plate-formes appelées Learning Management Systems(LMS)[LT09] sont des systèmes basés sur le web qui permettent aux instructeurs et/ou aux étudiants de partager des matériaux de cours, de soumettre ou de retirer des documents et de communiquer en ligne. Il existe plusieurs LMS, environnement d'apprentissage en ligne. Dans [WPY11], les normes les plus utilisées sont présentées. Deux approches se distinguent : celles qui mettent l'accent soit sur une communication entres les différents acteurs[WWG07], ou celle qui privilégient un enseignement interactif et les pratiques d'apprentissage [LT09].

Les LMS réunissent l'ensemble des acteurs de l'apprentissage autour des différents éléments tels que les tableaux inter-actifs et des plateformes comme de mise à disposition de contenu et d'activités. Malgré tous les avantages des LMS, l'interopérabilité entre les différents outils est une préoccupation majeure. Des systèmes LMS individuels ne peuvent pas échanger et partager des objets d'apprentissage directement avec d'autres LMS[WPY11]. En effet il n'existe hélas pas de mécanisme de partage fiable entre plate-forme de E-Learning isolé !

Pour palier ces problèmes, certaines solutions E-learning ont été proposées. Le cloud computing est le point commun de tous les LMS et est un acteur incontournable du monde éducatif[GPJA+14]. Le cloud computing a rendu possible la conception d'une nouvelle génération d'environnements virtuels : des environnements qui ne dépendent pas des distances et des localisations géographiques. Les offres de cloud computing peuvent être assez simples et sont diverses : bureau virtualisé qui relie les différents acteurs de l'éducation géographiquement dispersé, stockage de données, et le courrier électronique, suite d'applications de bureau, paquet de sécurité, et outils de collaboration [HN11].

Dans[ONH10], l'auteur présente le concept « Écosystème E-learning avec une intégration des fonctionnalités Web 2.0 ». Il reprend les environnements de type LMS en ajoutant quelques caractéristiques comme : la configuration en temps réel et l'utilisation des ressources, l'allocation à la demande des ressources et une meilleure gestion des défaillances matérielles. Cet écosystème est aussi déroulé dans [DZY+09] : le cloud est la base de l'écosystème E-learning. Il démontre la fiabilité, la flexibilité, la rentabilité, l'auto-régulation et la qualité de service offert par le cloud dans le concept de l'écosystème E-learning.

Les auteurs de[WPY11] tentent de résoudre le problème de l'interopérabilité des objets entre les différents systèmes LMS individuels. Ils proposent une architecture E-learning basée sur le cloud qui indexe les méta-données des différents objets des LMS et les transforme pour accomplir l'échange des ces différents systèmes E-learning.

Par extension, même les laboratoires de recherche sont virtualisés. Les auteurs dans [SAH+09] décrivent comment l'université North Carolina State University(NCSU) a virtualisé des laboratoires informatiques(VCL) appliquée sur cloud computing. Cela a permis une maintenance des ressources rentables et de puissance de calcul. Les auteurs ont également mentionné que l'échelle et le niveau d'utilisation sont importants pour fournir des services VCL rentables.

L' accès aux technologies mobiles sans fil est de plus en plus facile, et les technologies mobiles(PDA, Tablettes, Smartphone, etc.) sont adoptées dans le domaine éducatif. Le *M-Iearning*, ou l'apprentissage mobile, est un domaine qui implique les technologies de réseau sans fil et l'informatique mobile pour permettre à l'apprentissage de se produire à tout moment et à n'importe quel lieu, tout en augmentant considérablement la liberté de l'élève[SMQ11]. Cependant, les auteurs listés dans [SMQ11] exposent des difficultés à réaliser le *M-learning* avec ces technologies mobiles à cause des ressources limitées ou réduites des appareils mobiles,

et les investissements coûteux en terme de matériels et de logiciels. En plus, les appareils mobiles sont livrés avec une variété de systèmes d'exploitation incompatibles, ce qui provoque de graves problèmes d'interopérabilité [FSA12].

Pour apporter des réponses, des solutions *M-learnig* basées sur le cloud ont été proposées, ne nécessitant que l'usage d'un navigateur WEB, quel que soit le système d'exploitation associé. On peut notamment citer [Li10] [SMQ11] [CLHX10].

Dans[SV11], les auteurs proposent un système *Mobile Live Video* qui s'occupe de la conception et la mise en œuvre de l'apprentissage intéractif sur des appareils mobiles combinés avec la nouvelle technologie de cloud computing. Les tutotats des instructeurs sont en direct, filmés par des webcam installés dans la salle. Le film de la conférence est découpé en plusieurs épisodes et diffusé en streaming dans le serveur cloud. Une fois qu'une demande est reçue de l'étudiant, pour accéder à la vidéo en direct via son terminal mobile, elle est *bufferisée* à partir du serveur cloud dans la mémoire locale du dispositif mobile et diffusée.

Il est bien évident que le système cloud apporte beaucoup d'efficacité dans le système éducatif. Il aiderait les universités ou instituts d'enseignement à partager et diffuser les connaissances chez les élèves, les enseignants et les chercheurs. Mais l'utilisation du Cloud Computing dans le système éducatif présente de nombreux risques et limitations comme énumérés précédemment. De plus toutes les applications ne sont pas gérées dans les clouds.

Il existe des solutions *E-learning* basées sur le cloud qui tentent de palier le problème de sécurité en indiquant aux établissements d'enseignement de construire leur propre cloud privé pour stocker les données sensibles. D'autre préconisent le chiffrement des données sensible lors du transfert.

Enfin, nous sommes confrontés à un autre problème dans l'adoption des systèmes cloud : la vitesse d'accès aux infrastructures de cloud computing. Celle-ci peut être un facteur critique, amplifié par l'absence d'une connexion Internet stable qui peut affecter les méthodes de travail dans certaines régions isolées. En effet, avec les clouds, les ressources pédagogiques et les consommateurs sont distants, ce qui peut poser des problèmes d'accès et de qualité de service.

Les systèmes E-learning basés sur les réseaux pair-à-pair présentent une architectures « à plat » dans le sens ou tous les nœuds sont pareils ou ont un même rôle. Les données éducatives ne sont pas stockées en local. En effet, si l'on utilise une DHT comme outil de structuration, l'identifiant k d'une donnée peut être stocké sur une machine distante. Comme dans le cas des *clouds* l'accès distant peut limiter la qualité. De plus, le problème de sécurité est présent : tout le monde peut accéder aux données dans les systèmes pair-à-pair. Les réplications des données éducatives dans les systèmes pair à pair font qu'il est difficile de connaître les machines détentrices de ces dernières. Il est nécessaire de plonger sur -le système pair-à-pair utilisé une couche spécifique de sécurité et d'accès.

5.2.2 Bureaux virtuels

Afin de s'affranchir des problèmes de compatibilité de systèmes et d'outils, des solutions basées sur des bureaux virtuels sont proposées. Les auteurs de [LN02] [Mot07][dSR06] proposent de telles solutions, en offrant aux personnels étudiants des services qui facilitent l'enseignement, l'apprentissage et les tâches administratives liées à l'éducation. L'approche de base est collaborative, et même peut-on dire hybride, combinant une variété de dispositifs mobiles et non mobiles via une variété de techniques de transmission(filaires et sans fil).

Ces bureaux virtuels fonctionnent sur Internet pour les appareils mobiles et fixes complètent l'environnement E-learning en offrant des services tels que des alertes d'événements (des SMS, des messageries, des forums ...), répertoire pour partager des documents textes ou vidéos, calendriers et autres services ou ressources classiques de campus.

Les deux systèmes combinent la technologie basée sur un navigateur pour accéder au serveur central qui stocke toutes les données partagées dans ces bureaux virtuels pour enrichir l'expérience d'apprentissage de l'élève, et le soutien aux apprenants issu de l'apprentissage conversationnel. Ces campus virtuels utilisent des protocoles standards pour le partage des données.

La plupart de ces bureaux virtuels sont pris en charge par les applications client-serveur traditionnelles [GB] qui permettent le partage de données et de services entre les membres de la communauté éducative.

Dans de tels systèmes, les ressources partagées sont centralisées sur les serveurs et les membres de la communauté virtuelle (clients) y accèdent via des protocoles de demande. Tout est fait sur le côté serveur tandis que du côté client, juste une interface est nécessaire.

Cela porte les limites importantes telles que le manque de tolérance de l'évolutivité (le passage à l'échelle) et de la faute du serveur central, un manque de performance et les goulots d'étranglement ainsi que des coûts élevés dans l'acquisition, le développement et le maintien de ces applications.

En effet, dans les grandes universités qui utilisent ces bureaux virtuels, l'approche centralisée présente un coût très élevé et dans certains cas, prohibitif (par exemple comme lors de l'acquisition vidéo professionnelle et de l'équipement audio pour la distribution multimédia) et pas facile à maintenir, exigeant des tâches complexes, telles que la distribution intelligente de la charge ainsi que la synchronisation et la coordination des ressources.

La gestion au sein d'une institution de l'architecture technique peut s'avérer complexe : le serveur présente le gros défaut d'être un point de défaillance critique, le dimensionnement de ce serveur peut s'avérer complexe, et l'évolutivité de celui-ci se doit d'être garantie. Pour résoudre les inconvénients, des bureaux virtuels qui reposent sur un modèle client serveur, les instituts d'enseignements peuvent adopter l'approche décentralisée, notamment les systèmes E-learning basés sur les réseaux pair-à-pair. Les auteurs dans [BA03] discutent de l'utilisation des technologies pair-à-pair dans le domaine éducatif.

5.2.3 Groupware basé sur les systèmes Pair-à-Pair(P2P)

Les systèmes pair à pair de gestion des connaissances sont couramment utilisés dans l'éducation [Zhu02]. Ils fournissent les fonctionnalités de recherche, en utilisant un espace de travail, le partage de fichiers, la distribution de contenu et de communication synchrone.

Les élèves peuvent collaborer, échanger, partager, distribuer et diffuser les données, les uns avec les autres. Ces outils permettent de simuler l'environnement d'apprentissage réel de l'interaction entre pairs, la communication, le partage de matériel d'apprentissage et le travail de groupe afin que les élèves puissent apprendre en groupe dans cet environnement E-learning Pair-à-Pair.

Plusieurs travaux de recherche ont discuté des avantages de l'utilisation de technologies P2P dans l'éducation. Un de ces avantages est l'utilisation des technologies P2P pour le partage des ressources d'apprentissage.

Les auteurs de [XP10] proposent un groupware P2P. Leur modèle de réseau P2P est basé sur l'utilisation de Super-pair et de simple *pairs* qui intéragissent avec les sources de données (bases de données). Dans ce modèle, le réseau se compose de plusieurs *Peergroups*. Les pairs de chaque *Peergroup* sont connectés à un seul Super-pair.

La communication locale se fait entre les pairs dans un *Peergroup* et c'est plus fréquent, et les communications globales moins fréquentes se font entre les Super-pairs. Les Super-pairs servent en tant que coordinateurs de l'ensemble du réseau alors que les autres pairs représentent les membres du groupe *Peergroups*. Les Super-pairs seront généralement fixés sur les ordinateurs en réseau alors que les pairs peuvent être fixés sur ordinateurs ou mobiles, ou bien des dispositifs à ressources limitées.

Pour réaliser le travail collaboratif, les *Peergroups* forment un groupe des pairs qui souhaitent partager des ressources. Les informations sur les processus du groupe et l'accomplissement des tâches selon le flux de travail de groupe sont distribuées et répliquées à toute ou une partie des pairs de la communauté *Peergroups*(la distribution des données est transparente pour les membres du groupe). Les opérations de pairs et les informations de contrôle sont communiquées par eux à leur Super-pair. Le Super-pair est responsable de la gestion de la réplication et de la cohérence de l'information au sein de son *Peergroup* dans son ensemble.

Chaque Super-pair définit les règles d'accès aux données de chaque *Peergroups* basé sur les rôles(par exemple en lecture seule, lecture/ écriture, privé). Ces règles permettent un Super-pair pour spécifier si les objets et les ressources peuvent être consultés par d'autres super-pairs, en dehors de la *Peergroup*. Des privilèges d'accès similaires peuvent être définis sur les pairs, ce qui permet aux pairs de spécifier les objets et les ressources pouvant être consultées par d'autres membres du groupe.

Les auteurs utilisent dans leurs approches les opérations et services primitifs qui sont intégrés dans le middleware P2P pour partager l'accès des ressources. Les ressources partagées se résument à de simples fichiers stockés dans des bases de données.

Dans [EH12], les auteurs proposent un prototype de groupware Pair-à-Pair pour le travail coopératif et collaboratif dans le domaine de l'enseignement. Leur outils CSCL (Computer Supported Collaborative Learning), montre un scénario de salle de classe virtuelle typique avec flux de travail prévu entre les participants. L'architecture clé du prototype implique au moins trois composantes principales, à savoir : composante session de synchronisation, composante d'authentification, et la composante de partage de documents. Ces composantes sont extensibles et mises en œuvre en utilisant des connecteurs de services qui permettent l'intégration des services Web multiples qui sont disponibles dans l'Internet pour soutenir le scénario conçu.

Le prototype dispose des connecteurs de services cloud pour fournir au groupware une capacité d'accès à des ressources externes dans le réseau Internet ou d'un cloud. En outre, ces connecteurs de service peuvent également être partagés entre pairs dans le cadre de leurs ressources partagées. Il existe deux types de pairs : les pairs dont la puissance est élévée et les pairs moins puissants. Le partage de ressources entre ces pairs se fait via cloud : le pair puissant a la capacité d'accéder directement à un service cloud qui est nécessaire pour le pair moins puissant. Le pair moins puissant consomme le service de cloud à travers le pair plus puissant qui assure le service de proxy dans le cadre de ses ressources partagées.

Les auteurs dans [ZGL+07] et [JYY+05] proposent de combiner les technologies P2P, Grille et E-learning pour le développement d'environnements collaboratifs virtuels puissants, efficaces, évolutives, portables et polyvalents. Dans [ZGL+07] les auteurs proposent Environnement col-

laboratif virtuel géographique(CVGE) qui peut être utilisé dans le milieu universitaire. La plate-forme de la grille est utilisée pour soutenir le système de CVGE en assurant l'indexation des informations, le service management, la sécurité et le management des méta-données. Quant à la plate-forme P2P, elle est utilisée dans la couche, service pour assurer un haut rendement de transfert de service dans CVGE. Les services assurer par la plate-forme P2P sont : la collaboration vidéo(vidéoconférence), les messageries instantanée, les protocoles de recherche et de partage de fichiers et enfin les scènes de visualisation du terrain.

En fait, la plate-forme P2P est seulement logique et beaucoup de ses fonctions sont intégrées dans la plate-forme de la grille et représentées comme des services Grille. Enfin, sur la base de la plate-forme P2P et Grille, un certain nombre de technologies ont été intégrées et de nombreux services conçus pour faciliter le traitement et l'analyse application. L'accès aux services P2P se fait de manière transparente via la couche application qui fournit des fonctions pour les différentes applications telles que la vidéo conférence, la messagerie instantanée, la recherche de fichiers et des applications sécurisées pour les utilisateurs.

D'autres technologies basées sur les réseaux pair à pair ont été proposées pour améliorer l'efficacité des plateformes E-learning. Ainsi, dans [DSR10], les auteurs ont proposés LOP2P, une architecture qui permet l'interopérabilité aux différents établissements d'enseignement et de partager leurs objets référentiels d'apprentissage, afin de créer des cours à l'aide de systèmes de gestion de l'apprentissage (LMS). Le système LOP2P a une philosophie de partage d'objets d'apprentissage libre, qui est très similaire aux systèmes Pair-à-Pair de partage de fichiers. La principale différence est que l'architecture propose des LO(Learning Object) entre institutions plutôt que sur les ordinateurs personnels, à travers les LMS. Une observation importante est que, dans cette architecture, chaque institution connectée est considérée comme un pair. Donc, dans LOP2P il n'y a pas de structure centrale qui est maintenue comme les architectures basées sur le cloud, ce qui nécessiterait des coûts de maintenance élevés. Il y'a le système EduLearn [PSK09] qui est une architecture P2P qui offre un meilleur cadre pour l'interopérabilité et le partage des objets d'apprentissage(LO).

L'intégration des données distribuées sur des plateformes E-learning hétérogènes est possible grâce à l'utilisation des services hétérogènes (API) comme OAI-PMH ou SQI [DYG$^+$12]. Cependant, les méthodes normalisées pour résoudre les hétérogénéités entre les terminologies utilisées par fournisseurs de données ou de services distincts ne sont pas disponibles. Par conséquent, l'interopérabilité et l'évolutivité des applications E-Learning actuels est limitée. En plus le processus d'intégration est coûteux [YDL$^+$11] car les différents référentiels E-learning sont isolés les uns des autres et basés sur des normes de mise en œuvre différentes.

Dans[NEXC12] les auteurs utilisent les avantages des réseaux P2P pour permettre aux petits groupes d'élèves de travailler et d'avoir à disposition les ressources d'apprentissage nécessaires à la réalisation d'un projet commun, ainsi que d'être en mesure de communiquer immédiatement à tous les membres du groupe ces ressources. Cela aide grandement le groupe à prendre des décisions fondées sur les ressources disponibles et les informations sur ce qui se passe dans le groupe. Leur approches est basée sur la bibliothèque JXTA de sun Microsytems, qui comprend des protocoles de gestion de groupe de pair.

Les réseaux P2P caractérisés par une mobilité des nœuds, les auteurs utilisent la réplication pour assurer la disponibilité des données. Le scénario principal est celui dans lequel chaque nœud stocke une copie locale des fichiers qu'il gère, ainsi que d'un certain nombre de répliques d'autres fichiers à partir d'autres pairs.

La réplication P2P est particulièrement utile pour les systèmes P2P de groupware et le travail d'équipe, en augmentant la disponibilité et l'accès à toutes les informations que l'équipe gère, soutenant ainsi toute une gamme de possibilités pour accélérer, améliorer et rendre plus productif le travail d'équipe.

Les systèmes P2P apportent beaucoup d'avantages dans le système éducatif.En effet,ils construisent un environnement de collaboration décentralisé qui passe à l'échelle contrairement aux bureaux virtuels reposant sur le modèle client serveur inadapté dans les instituts de grande taille en nombre d'apprenants. En plus, la communication directe entre pairs peut non seulement réduire le temps de réponse aux demandes des apprenants mais aussi augmenter le nombre d'interaction entre pairs. Toutefois les systèmes E-learning basés sur les réseaux pair-à-pair présentent beaucoup de limites.

D'abord l'utilisation des technologies P2P dans le E-leraning ne permet pas le partage de données présentées sous la forme d'un simple fichier. Il est moins évident de partager des données qui ont d'autres formes de représentation comme les requêtes de bases de données, les données générées par les capteurs de toutes sortes par exemple les données environnementales nécessaires aux prévisions météorologiques ou à l'observation de l'activité sismique, ou bien les données générées lors d'une campagne d'exploration du sous sol à la recherche de gisements.

Du point de vue de la communication et du routage, les systèmes P2P sont principalement destinés à soutenir la communication explicite pair-à-pair, tandis que la collaboration de groupe devrait soutenir le mode plusieurs-à-plusieurs. En outre, les problèmes inhérents aux systèmes de P2P comme leur ampleur, la dynamique et l'hétérogénéité doivent être traités. Pour ce qui est du routage les systèmes E-learning basés sur les réseaux pair-à-pair il présente une architecture plate, dans le sens où tous les nœuds sont pareils ou ont même rôle. Ce qui fait que le système de recherche et de transfert de données se fait par un routage à plat qui génére trop de messages, surtout dans les réseaux de grande dimension.

Come nous l'avons précisé précédemment, les données éducatives dans les systèmes E-learning basé sur les technologies P2P ne sont pas stockées en local. Les auteurs [PAI01] avancent qu'un système E-learning efficace doit être préoccupé par le temps, par l'intensité du travail, et des ressources matérielles impliquées dans la gestion des environnements d'apprentissage en ligne.

En plus, s'ajoute le problème de sécurité car tout le monde peut accéder aux données dans les systèmes pair-à-pair lors de leur transfèrement. Les réplications des données éducatives dans les systèmes pair à pair comme dans [NEXC12] font qu'il est difficile de connaitre les machines détentrices de ces dernières.

5.3 Conception d'une application de *E-learning* basée sur GRAPP&S

Le Framework GRAPP&S est une solution E-learning pour le partage décentralisé de tous les types de ressources grâce a son système d'indexation et à la définition définition étendue des types des ressources. Il regroupe les différents référentiels (sources de données) locaux d'apprentissage de chacun des instituts (écoles, universités, les dépôts de laboratoires de recherche ...) dans une « communauté », ce qui peut favoriser les objectifs de réutilisation et de partage des ressources éducatives sans impliquer de les dupliquer dans les locaux d'apprentissage.

Grâce à l'utilisation de communautés, nous pouvons promouvoir les objectifs de réutilisation et de partage des ressources éducatives. En étendant la couverture à différentes ressources pédagogiques (fichiers, vidéos, données dans une base de données, même un flux vidéo par exemple lors de la souscription à la chaîne de télévision pour l'apprentissage des langues), nous pouvons intégrer tous les outils pour améliorer l'éducation dans une infrastructure unique, avec un coût plus faible.

Selon cette approche, des défis majeurs doivent être envisagées pour garantir l'interopérabilité sur le Web :

1) la décentralisation : une solution décentralisée pour le partage de tous les types de ressources éducatives, non seulement les fichiers(texte/xml), mais aussi des bases de données, des flux(vidéo, audio), et ressources de services à distance tels que les services Internet et de l'informatique distribuée, tous intégrés pour l'utilisation ou la fourniture de données ;
2) la simplicité d'accès : un moyen très simple de connecter un grand nombre d'instituts intéressés dans le partage des ressources. Ceci est obtenu par l'agrégation des ressources de chaque établissement d'enseignement sous la forme d'une « communauté ». Cela permettra un accès rapide à des ressources en raison de la proximité des consommateurs, contrairement à la plupart des solutions E-Learing basées sur le stockage en cloud ou les réseaux pair-à-pair lorsque les consommateurs sont pénalisés à cause de la vitesse d'accès lente ;
3) la gestion de la sécurité : GRAPP&S a la possibilité de définir des politiques de sécurité pour la protection des ressources éducatives au sein d'une communauté, et les politiques d'accès entre les différentes communautés à travers la mise en place des accords de niveau de service (SLA). Pour ces raisons, GRAPP&S peut également être étendue à d'autres domaines de l'école , par la création d'une communauté dans la partie administrative, séparé de la partie de l'éducation, avec des politiques d'accès et des règles de sécurité et de protection des ressources.

Comme décrit dans le chapitre 3, une communauté du Framwork GRAPP&S contient un processus *Comunicator* noté C et au moins un processus de *Resource Manager* noté RM et un processus *Data Manager* noté DM et ces processus sont hiérarchiquement organisés dans la Communauté.

Le processus C (voir chapitre 3, Section 3.5) permet de relier les communautés des instituts intéressés par le partage de ressources. Il assure la sécurité de sa communauté vers l'extérieur, par l'établissement des SLAs avec les autres communautés. Le communicateur définit également les règles de sécurité (accès) pour la protection des ressources éducatives à l'intérieur de la communauté(par exemple, la communauté de l'administration peut voir les données sur le système éducatif, mais pas l'inverse, grâce à ces règles d'accès définies par le communicator).

Le processus RM (voir chapitre 3, Section 3.4) assure l'indexation et l'organisation des données, et plus particulièrement des ressources, et services dans la communauté. Chaque donnée est identifiée de manière unique grâce à l'identifiant du nœud DM auquel s'ajoute une extension contenant des informations et le type MIME des données. Ceci permet de franchir la barrière du simple « nom de fichier », et peut donc faire cohabiter des données statiques(fichiers), des données dynamiques(requêtes sur une base de données, résultats d'un calcul) et des données à caractère temporaire(flux voix ou vidéo, état d'un capteur, etc.). Les nœuds reçoivent les requêtes des utilisateurs et assurent leur pré-traitement. Les nœuds RM participent à la recherche

de données dans la communauté.

Les processus *DM* (voir chapitre 3, section3.3.2) sont des proxies spécialisés qui interagissent avec les différents référentiels (sources de données) locaux d'apprentissage de chacun des instituts(écoles, universités, les dépôts de laboratoires de recherche, etc.) dans une « communauté ». Ces référentiels peuvent être : des bases de données ; des serveurs : FTP, webdav, mails ... ; des supports de stockage disque ou bien des services internet : cloud par exemple Dropbox, messageries Gmail ...

Le Framework GRAPP&S dispose de plusieurs fonctionnalités qui sont le partage de données, la mise en œuvre de la vidéoconférence et le partage du tableau blanc.

5.3.1 Partage de données

Le partage de données éducatives dans GRAPP&S se fait en utilisant les protocoles étudiés dans le chapitre 4. Le client qui accède à une donnée partagée dans GRAPP&S peut disposer des droits de lecture seule, ou bien des droits d'écriture sur la donnée.

Cas de la lecture seule

Lorsqu'un client Y veut lire une donnée dans une communauté de GRAPP&S, il commence d'abord par chercher le nœud DM_x qui souhaite partager ses ressources. Cette recherche dans une communauté de GRAPP&S se fait par paliers, de manière à respecter l'organisation hiérarchique du réseau. La procédure de recherche est décrite en détail dans le chapitre 4 et peut être résumée ainsi :

1 Un nœud $DM \in C_i$ envoie la requête à son nœud $RM_i \in (C_i)$
2 RM_i vérifie dans son index s'il y a parmi ses voisins un DM qui contient la donnée recherchée
3 Si oui, alors le nœud RM_i retourne au nœud DM_i une liste de nœuds DM qui contiennent l'information recherchée
4 Sinon, le nœud RM_i fait suivre la requête soit directement aux voisins $RM_k \in C_i$ (si le mécanisme de communication le permet), soit à son nœud $c_i \in C_i$ pour retransmission aux autres $RM_k \in C_i$
5 quand un nœud $RM_k \in CM_i$ trouve la ressources recherchée, alors la requête sera retournée au nœud DM_i émetteur en suivant le chemin inverse
6 Si la donnée recherchée n'est pas dans la communauté C_i, alors le nœud c_i fait suivre la requête vers d'autres communautés C_j.

En cas de réussite,le client obtient l'identifiant du nœud DM_x responsable de la donnée. Dans ce cas, le client a deux possibilités pour accéder à la donnée, soit par connexion directe, soit par une connexion routée. Dans le cas de la connexion directe, le client fait une requête directe au nœud DM_x afin d'accéder à la donnée. Comme le client peut se trouver dans un autre réseau qui ne permet pas l'accès directe à DM_x, la connexion peut se faire par routage interne dans GRAPP&S(voir chapitre 4, Section 4.2.4).

Cas de modification des données

Un client peut modifier les données partagées. La modification opérée porte sur la donnée elle-même. Cela signifie que l'on peut insérer, supprimer ou modifier une donnée dans la source.

Pour modifier une donnée, on suppose que le client à déjà obtenu l'identifiant du nœud DM possesseur du document grâce aux protocoles de recherche étudiés dans le chapitre 4.

Grâce à l'application de modification, le client dispose d'un message supplémentaire qui contient le type de mise à jour à savoir ajout, suppression de ressources, et modification. Le nœud DM possesseur du document dispose d'une liste appelée *logger* qui enregistre les dernières modifications, le *log*, etc.

Notre architecture présuppose que les capacités de stockage des nœuds DM sont limitées. De ce fait, le nœud DM détenteur de la donnée à modifier peut accorder ou non certaines tâches de calcul. Par exemple, un nœud DM peut accepter ou non que sa donnée soit modifiée ou bien lui en rajouter une nouvelle donnée.

5.3.2 Mise en œuvre de tableau blanc dans GRAPP&S

Dans [CL12], les auteurs proposent IMS une application de partage de documents basée sur l'application de tableau blanc. L'architecture de IMS est constituée d'une centrale de contrôle qui est composée de plusieurs serveurs qui distribuent des session aux clients pour modifier le tableau blanc en utilisant des protocoles de communication SIP.

Dans notre architecture, un tableau blanc est considéré comme une ressource partagée et modifiée. Une mise en œuvre d'un tableau blanc dans GRAPP&S se fait ainsi dans GRAPP&S : on dispose d'un centre de contrôle de session comme dans [CL12] qui est assuré dans GRAPP&S par un nœud Data Manager DM spécial. Chaque client du tableau est contrôlé par le centre de contrôle de session, chaque session de tableau blanc est attribué d'un identifiant(URI), à travers lequel le client peut prendre part à une session de tableau blanc. Le nœud DM spécial qui fait office de serveur dans le centre de contrôle de session fournit aux clients des stratégies de session pour garantir le couplage de dialogue. Le centre de contrôle de session est principalement responsable du contrôle et la gestion de la conversation.

5.3.3 Mise en œuvre de la vidéoconférence dans GRAPP&S

La vidéoconférence est un dispositif qui permet à un ou plusieurs utilisateurs(interlocuteurs) de s'échanger des informations(audio ou vidéo) en temps réel. Il est aussi possible d'échanger des documents à partir d'un ordinateur. Pour réaliser une vidéoconférence dans GRAPP&S, on utilise un nœud Data Manager(DM) spécial($DM_{special}$) qui fait office d'annuaire dans laquelle se trouve les identités de tous les autres nœuds DM sur lesquels se trouvent les utilisateurs qui souhaitent participer à la vidéoconférence. Lorsqu'un utilisateur sur DM_x veut entrer en contact avec un autre sur DM_y pour une diffusion de flux(audio ou vidéo), il envoie une requête au $DM_{special}$ pour récupérer l'identifiant DM_y.

Le nœud $DM_{special}$ vérifie dans son annuaire. S'il connait l'identifiant du nœud DM_y alors il le retourne au nœud DM_x. Á partir de ce moment le nœud DM_x peut avoir une relation(audio ou vidéo) avec le nœud DM_y.

Le nœud $DM_{special}$ peut disposer des fonctions en plus d'assurer la gestion de l'annuaire, il permet un enregistrement ou stockage de fichiers audio ou vidéo après conférence. En plus, le nœud $DM_{special}$ permet de faire des enregistrements direct de fichiers audio ou vidéo.

5.4 Conclusion

Dans ce chapitre, nous avons présenté une application E-learning basée sur GRAPP&S, qui est une solution décentralisée pour la gestion et le partage de ressources éducatives de tout les formats grâce à son système d'indexation et une définition des types Mime étendus à des données. Il regroupe les différents référentiels(sources de données) locaux d'apprentissage de chacun des instituts(écoles, universités, les dépôts de laboratoires de recherche, etc) dans une « communauté ». Une « communauté » de GRAPP&S est un réseau hiérarchique qui permet une mise en commun des ressources qui partagent une même propriété définit : même localité géographique, même autorité d'administration.

L'avantage d'une communauté de GRAPP&S permet une proximité des données et les utilisateurs(étudiants, enseignants, administratifs, etc.). GRAPP&S permet un environnement de travail collaboratif qui permet de mettre en œuvre des outils tels que les tableaux blancs qui sont pour notre Framework des ressources modifiées et de la mise en œuvre de la vidéoconférence.

Conclusion générale

Dans cette thèse, nous avons proposé dans le chapitre 3, le Framework GRAPP&S, une solution multi-échelle pour la gestion unifiée des données et Services. Ce Framework a été conçu pour intégrer de manière transparente les données de type fichier mais aussi les bases de données, les flux(audio, vidéo), les services Web et le calcul distribué.

L'architecture GRAPP&S regroupe les données et services qui partagent une même propriété définie : même autorité d'administration(exemple des bases de données métiers, les serveurs d'une structure entreprise éparpillée dans plusieurs sites, etc.), par une même localisation(par exemple les données et services d'un même pays, ou même région, les données d'une salle de travaux pratiques, etc.) sous forme d'un concept appelé *Communauté*.

Une *Communauté* est une entité *autonome* constituée de trois types de nœuds , un nœud *communicator*(c) , au moins un nœud *Ressource Manager(RM)* et au moins un nœud *Data Manager(DM)* organisés de façon hiérarchique, ce qui permet l' intégration de sources d'information disposant de protocoles d'accès hétérogènes et des règles de sécurité variées.

Le nœud c permet l'interconnexion de différentes *Communautés* et définit les règles de sécurité à l'intérieur de sa *Communauté* grâce à l'établissement de *Services Level Agreements (SLAs)*. Les nœuds *RM* assurent l'indexation des données dans la *Communauté* et enfin les DM sont des *proxy* qui interagissent avec les sources de données et permettent un accès unifié à des données et services multiples et variés dans GRAPP&S.

Nous avons présenté dans le chapitre 4 les principales fonctionnalités de GRAPP&S, c'est-à-dire la recherche et l'accès aux données. Nous avons proposé un mécanisme de routage préfixé pour la recherche de données de GRAPP&S. Ce mécanisme de routage s'appuie sur l' organisation hiérarchique de GRAPP&S et sur son système d' adressage hiérarchique des nœuds qui définissent les chemins par lesquels transitent les requêtes de recherche. Ce qui empêche l'inondation des liens du réseau.

Ensuite, nous avons proposé un protocole pour l'accès aux données de GRAPP&S qui ne dépend pas d'une connexion directe entre les nœuds, comme c'est le cas de la plupart des réseaux P2P. Dans GRAPP&S , il est toujours possible par routage de rapatrier les données par le chemin utilisé lors de la recherche, au cas où une connexion directe entre les nœuds n'est pas possible. De plus, l'exécution des mécanismes de recherche et d'accès aux données de GRAPP&S nécessite au pire des cas $O(n)$ messages pour chaque nœud du Framework.

Dans le contexte des réseaux P2P, la présence des pare-feux cause des problèmes. Ces problèmes concernent le système de recherche et le système de transfert de données dans GRAPP&S. Nous avons proposé des mécanismes de type *pull, proxy, push* implémentés par GRAPP&S pour contourner le problème des pare-feux afin de faire fonctionner le service de recherche et le service de transfert de données.

Dans le chapitre 5, nous avons proposé une application E-learning basée sur GRAPP&S. Notre Framework devient une solution décentralisée pour le partage de tous les types de ressources éducatives, non seulement les fichiers (texte /xml), mais aussi des bases de données, des flux (vidéo, audio), et ressources de service à distance tels que les services Internet et de l'informatique distribuée.

Le système connecte un grand nombre d'instituts intéressés dans le partage des ressources. Ceci est obtenu par l'agrégation des ressources de chaque établissement d'enseignement sous la forme d'une « communauté ». Cela permettra un accès rapide à des ressources en raison de la proximité des consommateurs, contrairement à la plupart des solutions E-Learing basées sur le stockage en cloud ou les réseaux pair à pair lorsque les consommateurs sont pénalisés à cause de la vitesse d'accès lente.

GRAPP&S a la possibilité de définir des politiques de sécurité pour la protection des ressources éducatives au sein d'une communauté et les politiques d'accès entre les différentes communautés à travers la mise en place d' accords de niveau de service (SLA). Pour ces raisons, GRAPP&S peut également être étendue à d'autres domaines (secteur) de l'école, par la création d'une communauté dans la partie administrative, séparée de la partie de l'éducation, avec des politiques d'accès et des règles de sécurité et de la protection des ressources.

Perspectives

Suite à ces travaux, différentes perspectives s'ouvrent.

Développement d'un Prototype sur Pastry

Le prolongement de ces travaux peut être envisagé dans plusieurs directions. Nous devons développer un prototype de l'architecture GRAPP&S. Ce prototype utilise le réseau overlay P2P FreePastry afin d'inter-connecter les différents nœuds de l'architecture GRAPP&S, comme illustré en la Figure 5.1.

L'utilisation de FreePastry est purement pratique car l'overlay Pastry[RD01a] se charge de toute interconnexion de bas niveau (ouverture de sockets, détection de défaillances, etc.), permettant ainsi aux développeurs de se concentrer sur la coordination des nœuds GRAPP&S (élection, indexation des données et services, traitement de requêtes, recherche, etc.).

L'utilisation d'un *design pattern* comme Façade permet un développement modulaire où la couche *overlay* peut être facilement remplacée. En effet, les contraintes de performance liées à la gestion des nœuds et au transfert des données peuvent être trop imposantes pour FreePastry, si la communauté est censée comporter un nombre élève de nœuds ou si le réseau présente un taux de volatilité important.

En plus de démontrer l'application de l'architecture GRAPP&S et la mise en place de tests de scalabilité, ce prototype permettra aussi l'analyse de l'impact de la localisation des nœuds par rapport à la gestion de la hiérarchie. En effet, le fonctionnement de GRAPP&S repose sur une structuration hiérarchique pour le routage, la recherche et l'interconnexion de nœud distants. Si le placement des nœuds a un impact sur la performance de l'architecture, il devra être pris en compte aussi lors de l'élection des RMs et C.

Il nous semble important d'étudier les contraintes de performance liées à la gestion des

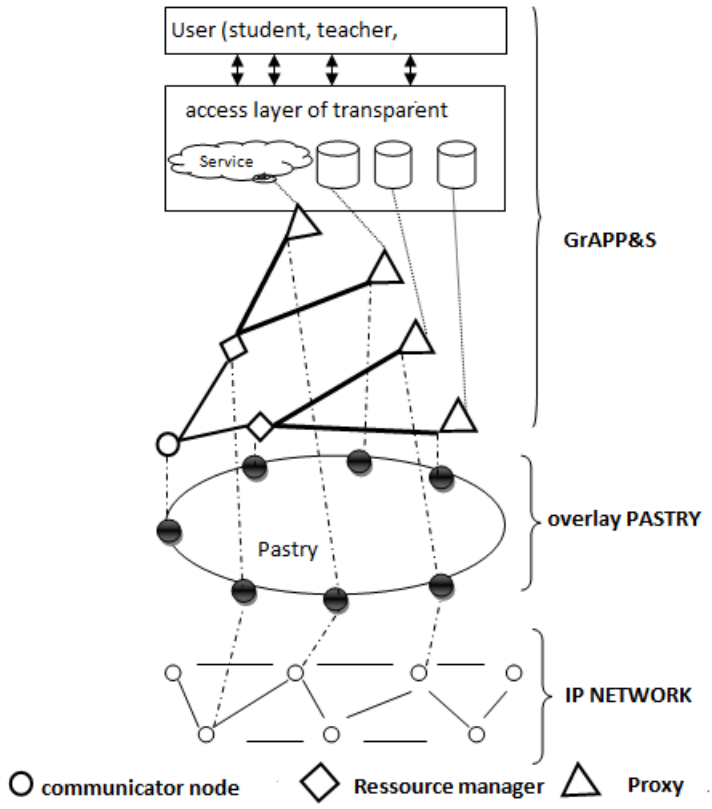


FIGURE 5.1 – Illustration de la distribution des nœuds GRAPP&S sur un overlay Pastry

nœuds et au transfert des données qui peuvent être trop imposantes pour FreePastry, si la communauté est censée comporter un nombre élevé de nœuds ou si le réseau présente un taux de volatilité important.

Redistribution de Ressources avec GAIA

Si dans un premier moment GRAPP&S agit comme un médiateur pour la localisation de ressources stockées dans les différents DMs, il dispose aussi d'un potentiel d'évolution grâce à la redistribution de ressources à travers l'ordonnanceur GAIA. GAIA a pour but l'intégration et l'éventuelle réorganisation des ressources entre les différents DMs, selon des critères tels que la fréquence d'utilisation, la proximité vis-a-vis des clients, le besoin de réplication, le coût de stockage et, bien sûr, la faisabilité d'une telle distribution.

En effet, le projet GAIA concerne le développement d'un ordonnanceur qui pourra gérer le stockage de certains types de données afin d'augmenter l'efficacité d'accès aux éléments les plus utilises, tout en réduisant les coûts de stockage des données moins critiques. Grâce à GAIA, GRAPP&S pourra mieux gérer l'intégration de services de stockage « lents » comme Amazon Glacier [1], qui pressentent un coût réduit mais aussi des politiques d'accès plus restrictives que les mécanismes de stockage traditionnels. Le prototype GRAPP&S servira à des tests de passage à

1. http ://aws.amazon.com/fr/glacier/

l'échelle et comme plateforme pour des services avancés tels que les grilles de sécurité[FNRS12], et l'optimisation du stockage avec l'ordonnanceur GAIA.

Tolérance aux fautes, Passage à L'échelle

Sur cette même thématique de performance, nous proposons d'étudier la gestion de la mobilité des nœuds par des algorithmes pour construire et maintenir notre réseau hiérarchique GRAPP&S. En effet, le passage à l'échelle et la tolérance aux fautes font partie de la spécification du Framework GRAPP&S. A l'aide de techniques issues des systèmes distribués et des réseaux pair-à-pair(P2P), les implémentations de GRAPP&S doivent permettre l'interconnexion d'un grand nombre de ressources(de l'ordre de quelques milliers de machines, comme par exemple dans une grille de calcul), tout en gardant une gestion totalement décentralisée et des hauts niveaux de disponibilité et réactivité face aux défaillances.

Ces algorithmes doivent offrir de très bonnes performances : prévoir un temps de latence maximale pour accéder aux données, de proposer une technique d'indexation des données et services propre à GRAPP&S. Cette technique peut être une extension des types MIMES des données de GRAPP&S.

Bibliographie

[10g13] 10GEN : http ://www.mongodb.org/, 2013.

[ABR12] Paolo ATZENI, Francesca BUGIOTTI et Luca ROSSI : Uniform access to non-relational database systems : The sos platform. *In Advanced Information Systems Engineering*, pages 160–174. Springer, 2012.

[ABR14] Paolo ATZENI, Francesca BUGIOTTI et Luca ROSSI : Uniform access to nosql systems. *Information Systems*, 43(0):117 – 133, 2014.

[ACK+02] David P ANDERSON, Jeff COBB, Eric KORPELA, Matt LEBOFSKY et Dan WERTHIMER : Seti@ home : an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[AIR+88] Michel ADAM, Philippe INGELS, Michel RAYNAL *et al.* : Algorithmes distribués synchrones et systèmes répartis asynchrones : concepts, mises en oeuvre et expérimentations. 1988.

[AJ13] R. ALADRAJ et M. JOY : Security and collaborative groupware tools in education : A case study at the university of bahrain. *In e-Learning "Best Practices in Management, Design and Development of e-Courses : Standards of Excellence and Creativity" , 2013 Fourth International Conference on*, pages 421–426, May 2013.

[ALPH01] Lada A ADAMIC, Rajan M LUKOSE, Amit R PUNIYANI et Bernardo A HUBERMAN : Search in power-law networks. *Physical review E*, 64(4):046135, 2001.

[And04] David P ANDERSON : Boinc : A system for public-resource computing and storage. *In Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 4–10. IEEE, 2004.

[ATA+03] Bernard Traversat AHKIL, Bernard TRAVERSAT, Ahkil ARORA, Mohamed ABDELAZIZ, Mike DUIGOU, Carl HAYWOOD, Jean christophe HUGLY, Eric POUYOUL et Bill YEAGER : Project jxta 2.0 super-peer virtual network. Rapport technique, 2003.

[ATS04] Stephanos ANDROUTSELLIS-THEOTOKIS et Diomidis SPINELLIS : A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys (CSUR)*, 36(4):335–371, 2004.

[Awe85] B. AWERBUCH : A new distributed depth first search algorithm. *INF. Proc. Lett.*, 1985.

[BA03] Kenneth A BERMAN et Fred S ANNEXSTEIN : An educational tool for the 21st century : peer-to-peer computing. *In Proc. of Ohio Learning Network Conference*, 2003.

[BA08] Nicolas Bonnel et Jacques Achille : *Adapnet : stratégies adaptatives pour la gestion de données distribuées sur un réseau pair-a pair*. Thèse de doctorat, Lorient, 2008.

[Bar64] P. Baran : On distributes commmunication networks. *IEEE Transactions on Communications*, 12:1–9, 1964.

[Bar97] Andy Barnhart : The common internet file system. *Software Development*, 5(1):75–77, 1997.

[BDPV99] Alain Bui, Ajoy Kumar Datta, Franck Petit et Vincent Villain : State-optimal snap-stabilizing pif in tree networks. *In Self-Stabilizing Systems, 1999. Proceedings. 19th IEEE International Conference on Distributed Computing Systems Workshop on*, pages 78–85. IEEE, 1999.

[BG92] D. Bertsekas et R. Gallager : *Data networks*. Prentice Hall, 1992.

[BHK+91] Mary G Baker, John H Hartman, Michael D Kupfer, Ken W Shirriff et John K Ousterhout : Measurements of a distributed file system. *In ACM SIGOPS Operating Systems Review*, volume 25, pages 198–212. ACM, 1991.

[BLT93] EM. Bakker, J. Van Leeuwen et RB. Tan : Prefix routing schemes in dynamics networks. *Computer Networks an ISDN Systems*, 1993.

[BMO05] Manav Bhatia, V Manral et Y Ohara : Is-is and ospf difference discussions. *Retrieved Jan*, 9:2007, 2005.

[BvLT93] Erwin M Bakker, Jan van Leeuwen et Richard B Tan : Prefix routing schemes in dynamic networks. *Computer Networks and ISDN Systems*, 26(4):403–421, 1993.

[BYL08] John Buford, Heather Yu et Eng Keong Lua : P2p networking and applications. 2008.

[Cab13] Luca Cabibbo : Ondm : an object-nosql datastore mapper. *Faculty of Engineering, Roma Tre University. Retrieved June 15th*, 2013.

[Car13] Josiah L. Carlson : *Redis in Action*. Manning Publications Co., Greenwich, CT, USA, 2013.

[Cat11a] Rick Cattell : Scalable sql and nosql data stores. *SIGMOD Rec.*, 39(4):12–27, mai 2011.

[Cat11b] Rick Cattell : Scalable sql and nosql data stores. *ACM SIGMOD Record*, 39(4):12–27, 2011.

[CD06] Eddy Caron et Frédéric Desprez : Diet : A scalable toolbox to build network enabled servers on the grid. *International Journal of High Performance Computing Applications*, 20(3):335–352, 2006.

[CDG+08] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes et Robert E Gruber : Bigtable : A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[Cha82] EJ. Chang : Echo algorithm : Depth parallel operations on general graphs. *IEE Transaction Software Eng.*, 1982.

[Cid88] I. Cidon : Yet an other distributed depth-first-search algorithm. *Information Processing Letter*, 1988.

[CL85] KM. Chandy et L. Lamport : Distributed snapshot determining global states of distributed systems. *ACM transaction computing system*, 3:63–75, 1985.

[CL12] Hao Cheng et Weimin Lei : Design and implementation of document-sharing based on ims. *In Computational Intelligence, Communication Systems and Networks (CICSyN), 2012 Fourth International Conference on*, pages 173–175, July 2012.

[CLHX10] Xuefei Chen, Jing Liu, Jun Han et Hongyun Xu : Primary exploration of mobile learning mode under a cloud computing environment. *In 2010 International Conference on E-Health Networking Digital Ecosystems and Technologies (EDT)*, volume 2, pages 484–487, 2010.

[CM98] Israel Cidon et Osnat Mokryn : Propagation and leader election in a multihop broadcast environment. *In Distributed Computing*, pages 104–118. Springer, 1998.

[CNY02] Yong Chen, Lionel M Ni et Mingyao Yang : Costore : A storage cluster architecture using network attached storage devices. *In Parallel and Distributed Systems, 2002. Proceedings. Ninth International Conference on*, pages 301–306. IEEE, 2002.

[Coh03] Bram Cohen : Incentives build robustness in bittorrent. 2003.

[Coh08a] Bram Cohen : The bittorrent protocol specification, 2008.

[Coh08b] Bram Cohen : The bittorrent protocol specification, 2008.

[CP84] Stefano Ceri et Giuseppe Pelagatti : *Distributed databases principles and systems*. McGraw-Hill, Inc., 1984.

[CRB+03] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham et Scott Shenker : Making gnutella-like p2p systems scalable. *In Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 407–418. ACM, 2003.

[CSWH01] Ian Clarke, Oskar Sandberg, Brandon Wiley et Theodore W Hong : Freenet : A distributed anonymous information storage and retrieval system. *In Designing Privacy Enhancing Technologies*, pages 46–66. Springer, 2001.

[DEG+01] Peter Druschel, Eric Engineer, Romer Gil, Y Charlie Hu, Sitaram Iyer, Andrew Ladd *et al.* : Freepastry. *Software available at http ://www. cs. rice. edu/CS/Systems/Pastry/FreePastry*, 2001.

[Den08] Yuhui Deng : Risc : A resilient interconnection network for scalable cluster storage systems. *Journal of Systems Architecture*, 54(1):70–80, 2008.

[Den09] Yuhui Deng : Deconstructing network attached storage systems. *Journal of Network and Computer Applications*, 32(5):1064 – 1072, 2009.

[DKK+01] Frank Dabek, M Frans Kaashoek, David Karger, Robert Morris et Ion Stoica : Wide-area cooperative storage with cfs. *ACM SIGOPS Operating Systems Review*, 35(5):202–215, 2001.

[DR01] Peter Druschel et Antony Rowstron : Past : A large-scale, persistent peer-to-peer storage utility. *In Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on*, pages 75–80. IEEE, 2001.

[dSR06] Gisele Trentin da Silva et Marta Costa Rosatelli : Adaptation in educational hypermedia based on the classification of the user profile. *In Intelligent Tutoring Systems*, pages 268–277. Springer, 2006.

[DSR10] Rafael De Santiago et André Raabe : Architecture for learning objects sharing among learning institution : Lop2p. *Learning Technologies, IEEE Transactions on*, 3(2):91–95, 2010.

[DWH+08] Yuhui Deng, Frank Wang, Na Helian, Sining Wu et Chenhan Liao : Dynamic and scalable storage management architecture for grid oriented storage devices. *Parallel Computing*, 34(1):17–31, 2008.

[DYG+12] Stefan Dietze, Honq Qing Yu, Daniela Giordano, Eleni Kaldoudi, Nikolas Dovrolis et Davide Taibi : Linked education : interlinking educational resources and the web of data. *In Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 366–371. ACM, 2012.

[DZY+09] Bo Dong, Qinghua Zheng, Jie Yang, Haifei Li et Mu Qiao : An e-learning ecosystem based on cloud computing infrastructure. *In Advanced Learning Technologies, 2009. ICALT 2009. Ninth IEEE International Conference on*, pages 125–127, July 2009.

[EH12] F.A Ekadiyanto et A Hunger : Prototype development towards hybrid peer-to-peer framework of distributed environment for cooperative and collaborative work. *In Computer and Communication Engineering (ICCCE), 2012 International Conference on*, pages 344–348, July 2012.

[FG01] Pierre Fraigniaud et Cyril Gavoille : Routing in trees. volume 2076 de *Lecture Notes in Computer Science*, pages 757–772. Springer Berlin Heidelberg, 2001.

[FG06] Pierre Fraigniaud et Philippe Gauron : D2b : A de bruijn based content-addressable network. *Theoretical Computer Science*, 355(1):65–79, 2006.

[FGNC01] Gilles Fedak, Cecile Germain, Vincent Neri et Franck Cappello : Xtremweb : A generic global computing system. *In Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pages 582–587. IEEE, 2001.

[Fin79] SG. Finn : Resynch procedures and fail-safe network protocol. *IEEE Tansaction Commununication*, 1979.

[FKS10] Oliver Flauzac, Michael Krajecki et Luiz-Angelo Steffenel : Confiit : a middleware for peer-to-peer computing. *The Journal of Supercomputing*, 53(1):86–102, 2010.

[FNRS12] Olivier Flauzac, Florent Nolot, Cyril Rabat et Luiz-Angelo Steffenel : Grid of security : a decentralized enforcement of the network security. *Manish Gupta, John Walp, and Raj Sharman (Eds.), Threats, Countermeasures and Advances in Applied Information Security, IGI Global*, pages 426–443, 2012.

[Fos05] Ian Foster : Globus toolkit version 4 : Software for service-oriented systems. *In Network and parallel computing*, pages 2–13. Springer, 2005.

[FP99a] Shawn Fanning et S Parker : Napster, 1999.

[FP99b] Shawn Fanning et S Parker : Napster, 1999.

[FSA12] Heba Fasihuddin, Geoff Skinner et Rukshan Athauda : A holistic review of cloud-based e-learning system. *In Teaching, Assessment and Learning for Engineering (TALE), 2012 IEEE International Conference on*, pages H1C–6. IEEE, 2012.

[FW78] JG. Fletcher et RW. Watson : Mechanisms for a reliable timer-based protocol. *Computer Network*, 2:271–290, 1978.

[Gar99] G. Gardarin : *Bases de données : objet et relationnel*. Eyrolles, 1999.

[Gau06] Philippe Gauron : *Efficient interconnection and routing for decentralized searching in peer-to-peer systems*. Theses, Université Paris Sud - Paris XI, septembre 2006.

[GB] Elena Gaudioso et Jesus G Boticario : Towards web-based adaptive learning communities.

[GEBF+03] L Garcés-Erice, EW Biersack, PA Felber, KW Ross et G Urvoy-Keller : Hierarchical peer-to-peer systems. *In Euro-Par 2003 Parallel Processing*, pages 1230–1239. Springer, 2003.

[GPJA+14] Francisco Josï¿½ García-Peñalvo, Mark Johnson, Gustavo Ribeiro Alves, Miroslav Minović et Miguel Ángel Conde-González : Informal learning recognition through a cloud ecosystem. *Future Generation Computer Systems*, 32(0):282 – 294, 2014.

[Gro09] Robert L Grossman : The case for cloud computing. *IT professional*, 11(2):23–27, 2009.

[GVM00] Garth A Gibson et Rodney Van Meter : Network attached storage architecture. *Communications of the ACM*, 43(11):37–45, 2000.

[HBMS04a] Oliver Heckmann, Axel Bock, Andreas Mauthe et Ralf Steinmetz : The edonkey file-sharing network. *GI Jahrestagung (2)*, 51:224–228, 2004.

[HBMS04b] Oliver Heckmann, Axel Bock, Andreas Mauthe et Ralf Steinmetz : The edonkey file-sharing network. *GI Jahrestagung (2)*, 51:224–228, 2004.

[HHLD11] Jing Han, E Haihong, Guan Le et Jian Du : Survey on nosql database. *In Pervasive computing and applications (ICPCA), 2011 6th international conference on*, pages 363–366. IEEE, 2011.

[HJ11] R Hecht et S Jablonski : Nosql evaluation : A use case oriented survey. *In Cloud and Service Computing (CSC), 2011 International Conference on*, pages 336–341. IEEE, 2011.

[HN11] Benjamin Hirsch et Jason WP Ng : Education beyond the cloud : Anytime-anywhere learning in a smart campus environment. *In Internet Technology and Secured Transactions (ICITST), 2011 International Conference for*, pages 718–723. IEEE, 2011.

[idc14] it-expertise.com/idc/. *In Home Page*. 2014.

[JYY+05] Hai Jin, Zuoning Yin, Xudong Yang, Fucheng Wang, Jie Ma, Hao Wang et Jiangpei Yin : Apple : a novel p2p based e-learning environment. *In Distributed Computing-IWDC 2004*, pages 52–62. Springer, 2005.

[Kli02] R KLINGBER : Gnutella protocol development : Gnutella 0.6, 2002.

[KM02a] Tor KLINGBERG et Raphael MANFREDI : The gnutella protocol specification v0. 6. *Technical specification of the Protocol*, 2002.

[KM02b] Tor KLINGBERG et Raphael MANFREDI : The gnutella protocol specification v0. 6. *Technical specification of the Protocol*, 2002.

[Kos00] Donald KOSSMANN : The state of the art in distributed query processing. *ACM Computing Surveys (CSUR)*, 32(4):422–469, 2000.

[Lav90] I. LAVALLÏ¿½E : *Algorithmique parallï¿½le et distribuï¿½e.* Hermï¿½s, 1990.

[LCC+02] Qin LV, Pei CAO, Edith COHEN, Kai LI et Scott SHENKER : Search and replication in unstructured peer-to-peer networks. *In Proceedings of the 16th international conference on Supercomputing*, pages 84–95. ACM, 2002.

[LCP+] Eng Keong LUA, J CROWCROFT, M PIAS, R SHARMA et S LIM : A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys & Tutorials, IEEE*, 7(2):72–93.

[LeL77] G. LELANN : Distributed systems : towards a formal approach. *In Information PROCESSING '77*, 1977.

[Li10] Jian LI : Study on the development of mobile learning promoted by cloud computing. *In Information Engineering and Computer Science (ICIECS), 2010 2nd International Conference on*, pages 1–4. IEEE, 2010.

[LKR] Jian LIANG, Rakesh KUMAR et Keith W ROSS : Understanding kazaa.

[LKR06] Jian LIANG, Rakesh KUMAR et Keith W ROSS : The fasttrack overlay : A measurement study. *Computer Networks*, 50(6):842–858, 2006.

[LN02] F. LEHNER et H. NOSEKABEL : The role of mobile devices in e-learning first experiences with a wireless e-learning environment. *In Wireless and Mobile Technologies in Education, 2002. Proceedings. IEEE International Workshop on*, pages 103–106, 2002.

[LT87] J. Van LEEUWEN et RB. TAN : Interval routing. *Computer Journal*, 1987.

[LT09] Steven LONN et Stephanie D. TEASLEY : Saving time or innovating practice : Investigating perceptions and uses of learning management systems. *Computers & Education*, 53(3):686 – 694, 2009.

[LY87] TH. LAI et TH. YANG : On distributed snapshots. *Inf. Proc.Letter*, 1987.

[Lyn68] WC. LYNCH : Reliable full-duplex file transmission over half-duplex telephones lines. *Communications of the Association of the Computing Machinery*, 11(6):407–410, 1968.

[M+98] John MOY *et al.* : Ospf version 2, rfc 2328. *Ascend Communications Inc*, 1998.

[MM02] Petar MAYMOUNKOV et David MAZIERES : Kademlia : A peer-to-peer information system based on the xor metric. *In Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[MNR02] Dahlia MALKHI, Moni NAOR et David RATAJCZAK : Viceroy : A scalable and dynamic emulation of the butterfly. *In Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 183–192. ACM, 2002.

[Mot07] Luvai F. Motiwalla : Mobile learning : A framework and evaluation. *Computers & Education*, 49(3):581 – 596, 2007.

[MS79] PM. Merlin et A. Segall : A fail safe distributed routing protocol. *IEEE Tansaction Communication*, 1979.

[NEXC12] A Navarro-Estepa, F. Xhafa et S. Caballe : A p2p replication-aware approach for content distribution in e-learning systems. *In Complex, Intelligent and Software Intensive Systems (CISIS), 2012 Sixth International Conference on*, pages 917–922, July 2012.

[ONH10] Shimaa Ouf, Mona Nasr et Yehia Helmy : An enhanced e-learning ecosystem based on an integration between cloud computing and web2. 0. *In Signal Processing and Information Technology (ISSPIT), 2010 IEEE International Symposium on*, pages 48–55. IEEE, 2010.

[Ora90] David Oran : Osi is-is intra-domain routing protocol. 1990.

[PAI01] G. Piccoli, R. Ahmad et B. Ives : Web-based virtual learning environments : A research framework and a preliminary assessment of effectiveness in basic it skills training. *MIS Quarterly : Management Information Systems*, 25(4):401–426, 2001. cited By (since 1996)399.

[PDQ+07] Zhuo Peng, Zhenhua Duan, Jian-Jun Qi, Yang Cao et E Ertao Lv : Hp2p : A hybrid hierarchical p2p network. *In Digital Society, 2007. ICDS'07. First International Conference on the*, pages 18–18. IEEE, 2007.

[Pel90] D. Peleg : Time-optimal leader election in general networks. *J. Parallel and Distributed Computing*, 1990.

[phe14] *http ://www.phex.org/mambo/*. Thèse de doctorat, home page 2014.

[PJS+94] Brian Pawlowski, Chet Juszczak, Peter Staubach, Carl Smith, Diane Lebel et Dave Hitz : Nfs version 3 : Design and implementation. *In USENIX Summer*, pages 137–152. Boston, MA, 1994.

[PRR99] C Greg Plaxton, Rajmohan Rajaraman et Andrea W Richa : Accessing nearby copies of replicated objects in a distributed environment. *Theory of Computing Systems*, 32(3):241–280, 1999.

[PSK09] Lakshmi Sunil Prakash, Dinesh Kumar Saini et Narayana Swamy Kutti : Integrating edulearn learning content management system (lcms) with cooperating learning object repositories (lors) in a peer to peer (p2p) architectural framework. *ACM SIGSOFT Software Engineering Notes*, 34(3):1–7, 2009.

[RB13] Benoit Romito et François Bourdon : Stockage décentralisé adaptatif. autonomie et mobilité des données dans les réseaux pair-à-pair. *Revue d'Intelligence Artificielle*, 27(6):765–796, 2013.

[RD01a] Antony Rowstron et Peter Druschel : Pastry : Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *In Middleware 2001*, pages 329–350. Springer, 2001.

[RD01b] Antony Rowstron et Peter Druschel : Pastry : Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *In Middleware 2001*, pages 329–350. Springer, 2001.

[RFH+01a] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp et Scott Shenker : *A scalable content-addressable network*, volume 31. ACM, 2001.

[RFH+01b] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp et Scott Shenker : *A scalable content-addressable network*, volume 31. ACM, 2001.

[RFH+01c] Sylvia Ratnasamy12, Paul Francis, Mark Handley, Richard Karp12 et Scott Shenker : A scalable content-addressable network. 2001.

[Rie03] Erik Riedel : Storage systems : Not just a bunch of disks anymore. *Queue*, 1(4):32–41, 2003.

[Rip01] Matei Ripeanu : Peer-to-peer architecture case study : Gnutella network. *In Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, pages 99–100. IEEE, 2001.

[SAH+09] H.E. Schaffer, S.F. Averitt, M.I. Hoit, A. Peeler, E.D. Sills et M.A. Vouk : Ncsu's virtual computing lab : A cloud computing solution. *Computer*, 42(7):94–97, July 2009.

[SBR+14] Mark Scott, Richard P. Boardman, Philippa A. Reed, Tim Austin, Steven J. Johnston, Kenji Takeda et Simon J. Cox : A framework for user driven data management. *Information Systems*, 42(0):36 – 58, 2014.

[SC11] Michael Stonebraker et Rick Cattell : 10 rules for scalable performance in'simple operation'datastores. *Communications of the ACM*, 54(6):72–80, 2011.

[SC12] Y. Serra et G. Chesnot : *Cloud computing, big data, parallélisme, hadoop : Stockage de données du futur*. Bases de données, stockage. Vuibert, 2012.

[Sch01] Rüdiger Schollmeier : [16] a definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. *In Peer-to-Peer Computing, IEEE International Conference on*, pages 0101–0101. IEEE Computer Society, 2001.

[Seg83] A. Segall : Distributed network protocols. *IEE Trans. Inform.Theory*, 1983.

[SENB07] Moritz Steiner, Taoufik En-Najjary et Ernst W Biersack : A global view of kad. *In Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 117–122. ACM, 2007.

[SK85] N. Santoro et R. Khatib : Labelling and implicit routing in networks. *Computer Journal*, 1985.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek et Hari Balakrishnan : Chord : A scalable peer-to-peer lookup service for internet applications. *In ACM SIGCOMM Computer Communication Review*, volume 31, pages 149–160. ACM, 2001.

[SMQ11] Qin Shuai et Zhou Ming-Quan : Cloud computing promotes the progress of m-learning. *In Uncertainty Reasoning and Knowledge Engineering (URKE), 2011 International Conference on*, volume 2, pages 162–164, Aug 2011.

[Sto11] Michael Stonebraker : Stonebraker on nosql and enterprises. *Commun. ACM*, 54(8):10–11, 2011.

[SV11] S.M. Saranya et M. Vijayalakshmi : Interactive mobile live video learning system in cloud environment. *In Recent Trends in Information Technology (ICRTIT), 2011 International Conference on*, pages 673–677, June 2011.

[SYAD05] Keith Seymour, Asim YarKhan, Sudesh Agrawal et Jack Dongarra : Netsolve : Grid enabling scientific computing environments. *Advances in Parallel Computing*, 14:33–51, 2005.

[Taj77] WD. Tajibnapis : A correctness proof of a topology information maintenance protocol for a distributed computer network. *Communications of the Association of the Computing Machinery*, 20:477–485, 1977.

[Tan97] AS. Tanenbaum : *Computer Networks*. Prentice Hall, 1997.

[Tar95] G. Tarry : Le probl ?me des labyrinthes. *Nouvelles annales de math ?mathique*, 1895.

[Tel91] G. Tel : *Topics in Distributed Algorithms*. Cambridge University Press, 1991.

[Tel94] Gerard Tel : *Introduction to distributed algorithms*. Cambridge university press, 1994.

[Tou80] S. Toueg : An all-pairs shortest-path distributed algorithm. Rapport technique RC 8327, IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, 1980.

[WH10] Scott Wolchok et J Alex Halderman : Crawling bittorrent dhts for fun and profit. *In Proceedings of the 4th USENIX conference on Offensive technologies*, pages 1–8. USENIX Association, 2010.

[WPY11] Chun-Chia Wang, Wen-Chang Pai et Neil Y. Yen : A sharable e-learning platform based on cloud computing. *In Computer Research and Development (ICCRD), 2011 3rd International Conference on*, volume 2, pages 1–5, March 2011.

[WWG07] R.E. West, G. Waddoups et C.R. Graham : Understanding the experiences of instructors as they adopt a course management system. *Educational Technology Research and Development*, 55(1):1–26, 2007.

[WZ08] Wenfeng Wang et Yuelong Zhao : A novel network storage scheme : intelligent network disk storage cluster. *In Networking, Sensing and Control, 2008. ICNSC 2008. IEEE International Conference on*, pages 142–147. IEEE, 2008.

[XP10] F. Xhafa et A Poulovassilis : Requirements for distributed event-based awareness in p2p groupware systems. *In Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*, pages 220–225, April 2010.

[YDL+11] Hong Qing Yu, Stefan Dietze, Ning Li, Carlos Pedrinaci, Davide Taibi, Nikolas Dovrolls, Teodor Stefanut, Eleni Kaldoudi et John Domingue : A linked data-driven & service-oriented architecture for sharing educational resources. 2011.

[YGM02] B. Yang et H. Garcia-Molina : Improving search in peer-to-peer networks. *In Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 5–14, 2002.

[YJZ08] Huayun Yan, Yunliang Jiang et Xinmin Zhou : A bidirectional chord system based on base-k finger table. *In Computer Science and Computational Technology,*

2008. ISCSCT '08. International Symposium on, volume 1, pages 384–388, Dec 2008.

[YL08] Jiadi Yu et Minglu Li : Cbt : A proximity-aware peer clustering system in large-scale bittorrent-like peer-to-peer networks. *Computer Communications*, 31(3):591–602, 2008.

[ZGL+07] Jun Zhu, Jianhua Gong, Weiguo Liu, Tao Song et Jianqin Zhang : A collaborative virtual geographic environment based on {P2P} and grid technologies. *Information Sciences*, 177(21):4621 – 4633, 2007.

[Zhu02] A knowledge flow model for peer-to-peer team knowledge sharing and management. *Expert Systems with Applications*, 23(1):23 – 30, 2002.

GRAPP&S, UNE SOLUTION TOTALEMENT RÉPARTIE POUR LE STOCKAGE DE DONNÉES & SERVICES

Résumé. *Le stockage des données est un point crucial du développement des applications, et plus particulièrement des applications réparties. Les problématiques liées au stockage sont multiples : assurer la pérennité des données, en assurer l'identification et l'indexation, en garantir la recherche et l'accès, et le cas échéant le rappatriement afin d'être exploité par les différents acteurs de l'application. Il faut de surcroit concevoir des méthodes permettant de garantir l'efficacité de toutes des propriétés et opérations, dans un environnement multi-hétérogène, tant au niveau des formats des données, des protocoles d'échange, des systèmes et des applications.*

Dans cette thèse, nous proposons GRAPP&S, une architecture multi-echelle pour le stockage et l'indexation unifiée de différents formats de données tels que les fichiers, les flux de données, les requêtes sur les bases de données mais aussi l'accès à des services distants (des web services sur un serveur ou le cloud, ou des tâches de calcul dans un cluster HPC). GRAPP&S offre et orchestre une architecture hiérarchique de routage pour accéder aux différents types de données, et permet l'interconnexion de différentes communautés(réseaux locaux) grâce aux principes des systèmes multi-échelle. Les données sont présentées de façon transparente à l'utilisateur par l'intermédiaire de proxies spécifiques à chaque type de donnée. Le routage hiérarchique de GRAPP&S permet de s'affranchir de tables de routage, et d'éviter l'utilisation de solutions d'inondation pour la recherche de données.

Enfin, nous proposons d'exploiter GRAPP&S dans le cadre particulier du E-Learning. Notre solution permettra de fédérer, à moindre coût, des ressources pédagogiques réparties sur plusieurs organisme, et d'en assurer l'exploitation par des apprenants.

Mots-clés : Systèmes répartis, Applications répartis , Systèmes Pair-à-Pair, Routage préfixés, Proxies, Adressage hiérarchique, E-learning

---

GRAPP&S, SOLUTION FULLY DISTRIBUTED FOR DATA STORAGE & SERVICES

Abstract : Data storage is a crucial point for applications development and particularly for distributed applications. There are many issues related to data storage : the insurance of sustainability, identification and indexing of the data, the warranty of searching and accessing the data, and possibly the fetching of the data for applications use. So, there is a need to design methods for ensuring effectiveness of all of these properties and operations in heterogeneous environments, both in terms of data formats, exchange protocols, and applications.

In this thesis, we propose GRAPP&S (Grid APPlication & Services), a multi-scale framework for an unified storage and indexing of both data and services. Indeed, GRAPP&S was designed to support different data formats such as files, stream or database queries, but also to support the access to distant services (web services, cloud services or HPC computing services, for example). GRAPP&S coordinates an hierarchical routing architecture that allows data indexing and access, thanks to a multi-scale network of local-area communities. Transparent access is provided by a network of specialized proxies, which handle most aspects related to data location, request handling, data preprocessing or summary and also data consistence.

Finally, we exploit GRAPP&S in the particular context of E-Learning. Our solution will reduce the cost of merging distributed educational resources over several organizations and ensure the exploitation of learners..

keywords : distributed systems, distributed applications, Peer-to-Peer Systems, Prefix Routing, Proxies, Hierarchical Addressing, E-learning

---

Centre de Recherche en STIC (CReSTIC) équipe Systèmes Communicants (SysCom EA 3804), URCA, BP 1039, 51687 Reims Cedex 2, France
Laboratoire de Mathématiques et Informatiques (LMI) équipe Bases de Données et Data Maning, UCAD, BP 5005 , Dakar -fann, Sénégal