Comparaison des Modèles et Architectures pour un Accès Mobile Restreint et Local au Web de Données

Mahamadou Toure, Fabien Gandon, Kaladzavi Guidedi, Christophe Guéret, Moussa Lô, Pascal Molli

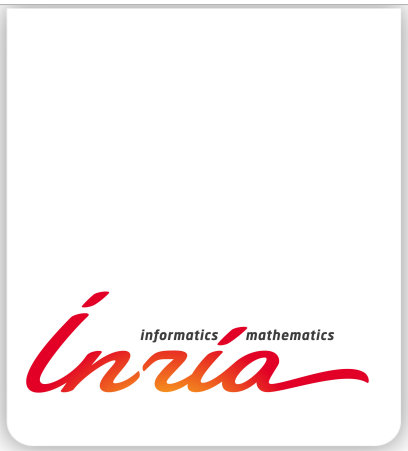
Comparaison des Modèles et Architectures pour un Accès Mobile Restreint et Local au Web de Données

Un état de l'art des architectures et solutions envisageables

Mahamadou Toure, Fabien Gandon, Kaladzavi Guidedi, Christophe Guéret, Moussa Lo, Pascal Molli

RESEARCH REPORT
N° 9121
Novembre 2017
Project-Teams WIMMICS

ISSN 0249-6399 ISRN INRIA/RR--9121--FR+ENG



Comparaison des Modèles et Architectures pour un Accès Mobile Restreint et Local au Web de Données

Un état de l'art des architectures et solutions envisageables

Mahamadou Toure, Fabien Gandon, Kaladzavi Guidedi,
Christophe Guéret, Moussa Lo, Pascal Molli

Équipes-Projets WIMMICS

Rapport de recherche n° 9121 — Novembre 2017 — 81 pages

Résumé : Ce document présente un état de l'art préalable à un projet de thèse qui vise à proposer des solutions permettant un accès local et restreint au Web de données. Nous avons exploré plusieurs mécanismes pertinents proposés dans la littérature, dédiés à des problématiques différentes et pouvant constituer des points importants pour nos futures contributions. Notre objectif principal étant de proposer des outils dédiés à des situations de mobilité dans des environnements ou l'accès internet est peu fiable, nous nous sommes intéressés particulièrement à comparer des approches (P2P, DHT, etc.) permettant de réaliser un scénario de fog/edge computing particulier qui est l'accès mobile restreint et local à des données liées contextuellement pertinentes et partagées. Dans cette optique, nous avons focalisé une partie de nos recherches bibliographiques sur les protocoles dits de « gossip » (ou protocoles épidémiques) qui s'avèrent bien adaptés au caractère dynamique des réseaux. L'aspect dynamique des pairs nous a aussi amené à considérer des solutions qui prennent en compte la localisation des pairs pour améliorer la qualité des services offerts.
Nous regardons de plus dans ce document, des solutions traitant de l'hétérogénéité sémantique dans des environnements pair-à-pair, notamment des mécanismes d'alignements d'ontologies qui permettent de profiter pleinement de la puissance des systèmes pair-à-pair sans imposer l'utilisation d'une ontologie commune à tous les pairs.
La réplication et le cache de données pouvant constituer une alternative importante à l'accès à des sources distantes, nous avons consacré la dernière partie de ce document aux solutions reposant sur ces mécanismes et offrant ainsi un accès local aux données.

Mots-clés : protocole de gossip, membership management, système pair-à-pair, RDF, cache décentralisé, accès mobile, accès local, géolocalisation

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

A Survey of the Models and Architectures for Restricted and Local Mobile Access for Web of Data

A state of the art of the architectures and the possible solutions

Abstract: This document presents a survey prior to a thesis project, which aims to propose solutions allowing local and restricted access to the Web of data. We have explored several relevant mechanisms proposed in the literature, dedicated to various problems and likely to constitute important points for our future contributions. Since our main objective is to offer tools dedicated to mobility situations in environments where internet access is unreliable, we were particularly interested in comparing approaches (P2P, DHT, etc.) allowing to realize a particular fog/edge computing scenario which is restricted and local mobile access to contextually relevant and shared related data. In this context, we have focused some of our research on so-called gossip protocols (or epidemic protocols) that are well suited to the dynamic nature of networks. The dynamic aspect of the peers also led us to consider the solutions that take into account the location of the peers in order to improve the quality of the services offered.
In addition, we looked at solutions dealing with semantic heterogeneity in peer-to-peer environments, including ontology alignment mechanisms that allow us to fully benefit from the power of peer-to-peer systems without imposing the use of a common ontology to all peers.
Data replication and caching can be an important alternative to accessing remote sources; we have devoted the last part of this document to solutions based on these mechanisms to supply a local access to data.

Key-words: gossip protocol, membership management, peer-to-peer system, RDF, decentralised cache, mobile access, local access, Geolocation

Table des matières

Liste des tableaux

Table des figures

1 Introduction

1.1 Contexte général

L'accès mobile au Web de données a fait l'objet de plusieurs études ces dernières années, mais il subsiste encore le besoin de concevoir et de mettre à disposition des utilisateurs des outils leur permettant d'accéder ou de contribuer aux données, en situation de mobilité, avec des ressources limitées (mémoire de stockage, processeur, RAM, etc.) et aussi dans les cas où l'accès internet est limité ou peu fiable.

Nous nous intéressons en particulier aux cas où des utilisateurs souhaitent accéder ou contribuer à des données socio-culturelles notamment locales et ceci malgré les limitations des infrastructures. Un scénario typique, par exemple, serait celui d'échanges d'informations à propos de rencontres sportives, d'événements culturels ou de commerces et services locaux dans une zone où l'accès premier à Internet se fait par les mobiles mais où la connectivité à Internet n'est pas fiable ou les ressources technologiques limitées.

1.2 Objectifs

Notre objectif principal est de proposer des modèles, protocoles et algorithmes pour des applications mobiles permettant aux utilisateurs d'accéder, de partager et de publier des données dans un environnement où les ressources matérielles sont limitées, dans le but de co-construire localement et de manière automatique des entrepôts de triplets de données RDF distribués représentant une base de connaissance socioculturelle partagée sur le web sémantique.

Une partie importante de nos recherches consistera à mettre en place une application en réseau distribuée avec un accès très limité à Internet qui permettra aux utilisateurs d'échanger des informations. Pour construire un tel réseau, nous nous focaliserons particulièrement sur les protocoles de communication basés sur les spéculations (gossip-based or epidemic protocols). Dans ce type de protocole, lorsqu'un nœud (un utilisateur) souhaite envoyer une information via le réseau, il sélectionne aléatoirement t nœuds parmis ses voisins (t est un paramètre de configuration appelé fanout) et leur envoie le message. Après réception du message pour la première fois, chaque nœud répète cette procédure [72]. Ces protocoles ont une approche intéressante dans le sens où ils sont hautement résilients (ils ont un niveau intrinsèque de redondance leur permettant de masquer les défaillances au niveau du réseau et des nœuds) et extensibles (la charge est distribuée à travers tous les nœuds dans le système). Nous nous intéresserons donc à étudier et concevoir des protocoles dans la continuité de cette famille mais dédiés au partage et à la maintenance de bases de connaissances locales en RDF.

Les aspects liés aux notions de context, profil et interface de l'utilisateur sont très importants pour améliorer la qualité du système.

- L'accès (visualisation et contribution) des données de la base de connaissance peut prendre en compte la notion de contexte de l'utilisateur qui permettra à l'application d'être la plus précise possible dans ses réponses aux interactions avec l'utilisateur. La notion de contexte représente toutes les informations pouvant être utilisées pour caractériser la situation d'une entité (un utilisateur dans notre cas). Une entité est une personne, une place, ou un objet qui est considéré pertinent pour l'interaction entre un utilisateur et une application, ces derniers y compris [103]. Le caractère omniprésent des appareils mobiles

(exemple : tablette, Smartphone, GPS) est en partie la raison de l'utilisation massive des informations contextuelles dans les applications mobiles modernes.
- La prise en compte du profil des utilisateurs est utile aussi bien pour la mise en place des vues et fédérations dans l'architecture P2P que pour la sélection des données à proposer.

Ces aspects n'étant pas couverts par cet état de l'art, ils feront l'objet d'une autre étude.

La géolocalisation participe aussi au profilage de l'utilisateur pour pouvoir définir une distance appropriée de communication directe entre deux utilisateurs (notion de voisinage) ou pour sélectionner des données les plus immédiatement pertinentes.

Enfin les contraintes de connectivité demandent des mécanismes de cache intelligents pour les données anticipant les besoins des utilisateurs locaux en cas de problèmes de déconnexion.

1.3 Contraintes

Plusieurs contraintes peuvent être prises en compte :
- nécessité de consulter et contribuer localement à des données socio-culturelles (événements, activités, services, etc. locaux)
- connexion Internet peu fiable et limitée
- ressources matérielles limitées (mémoire de stockage, autonomie en énergie, processeur, RAM, coûts)
- haute disponibilité des données localement pertinentes
- mobilité, arrivées et départs des contributeurs

1.4 Verrous

Le projet de thèse motivant cet état de l'art propose d'explorer les spécificités (avantages et contraintes) du partage mobile de connaissances. Plus précisément nos recherches porteront sur la question suivante : Comment permettre l'accès en lecture et écriture à des données du Web localement pertinentes dans un environnement limité en ressources et en accès Internet ?
Cette question sera traitée à travers plusieurs sous-questions :
- **Pair-à-pair pour Web de données :** Comment mettre en place une topologie pair-à-pair décentralisée permettant l'échange de données aux formats du Web sémantique avec un accès Internet limité ?
- **Caching local intelligent :** Comment maintenir une bonne disponibilité des données pertinentes localement malgré l'arrivée, le départ et les déconnexions du réseau des pairs et de la connexion Internet ?
- **Accès mobile :** Comment faciliter l'accès en recherche et en contribution des utilisateurs depuis des terminaux mobiles limités ?

1.5 Glossaire

Cette section rassemble quelque termes clefs du domaine en donnant les termes Anglais et Français ainsi que les explications associées.

Terme (EN)	Terme (FR)	Définition
Gossip-based protocols, Epidemic protocols	Protocol de « gossip » Protocole épidémique	Ce sont les protocoles qui agissent comme suite. Lorsqu'un nœud souhaite envoyer une information via le réseau, il sélectionne aléatoirement t nœuds dans le système (t est un paramètre de configuration appelé fanout) et leur envoie le message. Après réception du message pour la première fois, chaque nœud répète cette procédure [72].
Context	Contexte	Toute information pouvant être utilisée pour caractériser la situation d'une entité. Une entité est une personne, un lieu ou un objet considéré comme pertinent pour l'interaction entre un utilisateur et une application, y compris l'utilisateur et les applications elles-mêmes.
Push		Dans le modèle "Push", l'expéditeur peut livrer le trafic à volonté à un récepteur, qui ne peut qu'accepter passivement le trafic, comme dans les systèmes de messagerie électronique SMTP [35].
Pull		Dans le modèle "Pull", le récepteur peut décider si et quand il souhaite récupérer des données, comme les systèmes HTTP d'accès au web [35].
Push-Pull		C'est une combinaison des deux modèles « push » et « pull » .
Peer	Pair	Entité de réseau avec laquelle un nœud effectue des opérations d'envoie et de réception de message.
View	Vue	La vue d'un nœud indique un ensemble de nœuds dans le système avec lesquels le nœud concerné peut interagir directement.
Neighbor	Voisin	
Neighborhood	Voisinage	
Random Peer Sampling	Échantillonnage aléatoire de pairs	
Shuffling		Processus de brassage des vues.
Membership		Processus d'adhésion des nœuds.
fanout		Paramètre de configuration indiquant le nombre de cibles auxquels le message de gossip est envoyé.

edge computing	edge computing	Edge computing est une architecture de technologie d'information distribuée (IT) dans laquelle les données client sont traitées à la périphérie du réseau, aussi près que possible de la source d'origine. . Au lieu de faire la majeure partie du traitement dans un serveur centralisé, chaque périphérique du réseau jouerait son rôle dans le traitement de l'information.
fog computing	fog computing	Le fog computing est un scénario où un grand nombre de dispositifs hétérogènes (sans fil et parfois autonomes) ubiquitaires et décentralisés communiquent et coopèrent potentiellement entre eux et avec le réseau pour effectuer des tâches de stockage et de traitement sans intervention de tiers. Ces tâches peuvent être destinées à prendre en charge les fonctions réseaux basiques ou les nouveaux services et applications qui s'exécutent dans un environnement sandbox (en copie locale).[125]
Single-master	Single-master	Une seule copie primaire pour chaque objet répliqué.
Multi-master	Multi-master	Plusieurs sites contiennent une copie primaire du même objet.

1.6 Méthode de bibliographie

1.6.1 Interrogations sur le domaine

Nous identifions ici quelques questions relatives aux différents objectifs que nous voulons considérer dans ce rapport pour guider notre travail d'état de l'art. Chacune de ces questions est traitée dans les différents chapitres du document.
Q1 Quels sont les travaux dédiés à des systèmes de communication reposant sur des protocoles de gossip ?
Q2 Quels sont les effets majeurs de l'utilisation des protocoles de gossip sur un système dédié à l'échange de données ?
Q3 Comment les protocoles de gossip permettent-ils l'échanges de données sémantiques ?
Q4 Comment la géolocalisation peut-être prise en compte pour les échanges de données sémantiques ?
Q5 Quelles sont les solutions existantes pour un accès local à des données ?
Q6 Comment le cache de données peut-il améliorer le processus d'accès et d'interrogation des sources de données ?
Q7 Quelles sont les méthodes d'évaluations utilisées dans le domaine ?
Q8 Quelles sont les perspectives de recherches les plus pertinentes ?

1.6.2 Mots clefs

Nous indiquons ici les mots clés et les expressions et synonymes utilisés pour la recherche d'articles à considérer pour réaliser le rapport.

Mots clefs : semantic web, gossip protocol, système pair-à-pair, cache décentralisé, accès mobile, géolocalisation.

Expressions de recherches : “Semantic profil” ou “clustering sémantique” ou “Échange de données sémantiques” et “gossip protocol” ou “random peer sampling” ou “membership management” et "système pair-à-pair" ou "peer-to-peer membership" et "access mobile" ou "accès locale" ou "données locales" ou "réplication de données" et "cache décentralisé" ou "cache client" ou "cache local" ou "cache distribué" et "géolocalisation" ou "peer localisation" ou "routage géographique".

1.6.3 Sources

Nous indiquons ici les sources utilisées pour construire cet état de l'art.

- ◇ DBLP : dblp.uni-tier.de
- ◇ HAL : hal.archives-ouvertes.fr
- ◇ ACM Digital Library : dl.acm.org
- ◇ IEEExplore : ieeexplore.ieee.org
- ◇ Mendeley : www.mendeley.com
- ◇ Scholar : scholar.google.com
- ◇ CiteSeer : citeseerx.ist.psu.edu

1.6.4 Critères de choix des documents

Nous regardons ici les différents critères considérés lors de nos recherches de documents pour réaliser ce rapport. Nos choix sont justifiés par l'orientation des papiers par rapport aux critères d'inclusion et d'exclusion ci-dessous.

Critères d'inclusion :

- Papiers traitant de la conception de protocole de gossip
- Papiers traitant du problème de l'échantillonnage aléatoire
- Papiers proposant des systèmes d'échange de données qui reposent sur des protocoles de gossip
- Papiers traitant de l'échange de données sémantiques sur des architectures pair-à-pair
- Papiers traitant de l'accès local à des sources de données
- Papiers écrits en français ou en anglais
- Papiers publiés dans des conférences ou des journaux, papiers courts, papiers de workshop et rapports de recherche qui traitent des points cités précédemment.

Critères d'exclusion :

- Papiers ne traitant ni de la communication pair-à-pair ni de l'accès local à des données.
- Thèses de doctorats, slides de présentations, études non publiés
- Papiers traitant des techniques du web sémantique mais non appliquées sur des architectures pair-à-pair
- Papiers qui présentent des mécanismes de gossip mais non dédiés à l'échange de données dans des environnements physiques
- Papiers qui proposent des solutions d'accès local sur des appareils non mobiles
- Papiers traitant de protocoles de gossip sur des architectures pair-à-pair centralisées.

1.7 Exemple de scénario

Pour illustrer notre scénario, considérons l'événement du festival International de Jazz de Saint Louis. C'est un événement annuel où des milliers de personnes se rencontrent pour célébrer

la musique Jazz et profiter des différents concerts et prestations culturelles qui se tiennent à cet effet. Durant cet événement le besoin permanent d'accès et de partage d'informations liées aux programmes et à leurs contenus représente un point important pour l'amélioration de la qualité du festival.
Khadim, Thierno et Guirane sont un groupe d'amis venus spécialement assister au festival. Khadim est intéressé par les Jazzmans Africains et le fleuve. Thierno et Guirane quant à eux sont spécialement venus pour les prestations des différents quintets sur la place Faidherbe. Leurs téléphones mobiles sont dotés chacun de l'application dédiée à cette événement qui repose sur le modèle que nous proposons.
A leur arrivée à la gare routière, leurs appareils mobiles détectent le réseau wifi de la zone et s'y connectent automatiquement. Thierno et Guirane sont alors intégrés dans des clusters situés dans la zone par rapport à leurs points d'intérêts. Quant à Khadim, étant le seul intéressé par le Jazz Africain, il est intégré dans un cluster vide pour ce point d'intérêt et dans un autre cluster relatif au fleuve.
Chacun souhaitant avoir les informations par rapport à ses points d'intérêt, ils lancent des requêtes relatives à ces derniers via l'application. Thierno et Guirane reçoivent des informations identiques puisqu'ils appartiennent aux mêmes clusters. Khadim aussi reçoit les informations sur des festivités se tenant tout près du fleuve. Grâce au mécanisme de routage de requêtes au niveau des super-pairs, il reçoit également des informations sur le Jazz Africain provenant d'un cluster couvert par le super-pair situé au niveau de l'hôtel de ville.
Au même moment, leur camarade Sammuel venant de la France pour assister au festival, débarque à l'aéroport de Dakar. Le téléphone de Sammuel est aussi doté de l'application mobile dédiée au festival. Le téléphone est alors automatiquement connecté au réseau wifi couvrant la zone et porté par le super-pair de l'aéroport. Sammuel paramètre son application en précisant qu'il est aussi particulièrement intéressé par le Jazz Africain. La zone étant dépourvue en ce moment de personne avec les mêmes intérêts, Sammuel est intégré dans un cluster vide. Il décide cependant de faire une requête pour voir la programmation selon son point d'intérêt. Grâce au mécanisme de routage sur les super-pairs, sa requête est alors routée vers le super-pair de la gare routière de Saint Louis. Il reçoit alors les informations issues du cache de Khadim qui se trouve dans cette zone.
Connaissant bien un des Jazzman Africain dont le nom a été mal écrit sur le système, il décide de modifier cette information. L'opération de mise à jour est alors envoyée vers le super-pair de l'aéroport. Grâce au mécanisme de réplication optimiste implémenté sur le système sous-jacent de l'application mobile, les super-pairs du village artisanal et de Saint Louis reçoivent l'opération de mise à jour et l'appliquent de manière asynchrone. Son ami Khadim décide quelques minutes plus tard de rafraîchir les données qui lui sont présentées et se rend compte que le nom du Jazzman a effectivement changé.

2 Communications pair-à-pair et échanges de données sémantiques

En partant des questions **Q2** et **Q3**, nous nous intéressons ici aux solutions reposant sur des protocoles de gossip. Nous regardons les enjeux et principes de fonctionnement de ces protocoles. Les protocoles de « gossip » sont très adaptés à la conception de système de communication pair-à-pair du fait notamment de leur variabilité dimensionnelle, de leur facilité de déploiement, mais surtout de leur résistance aux défaillances des réseaux et processus.
Ils ont été appliqués avec succès dans les systèmes à grande échelle. En dehors de la traditionnelle application bien connue pour la diffusion de l'information, ces protocoles ont été appliqués pour le calcul d'agrégats (exemple : moyenne, variance, minima, maxima, etc.) [71][82][66], l'équilibrage de charge [64], la gestion de réseau [126]. La propriété commune de ces protocoles est que, périodiquement, chaque nœud du système échange des informations avec certains de ses pairs.

Un protocole de gossip suppose cependant l'existence d'un mécanisme d'adhésion sous-jacent, une composante fondamentale, qui fournit à chaque nœud une connaissance globale ou partielle du système. Ceci consiste en une liste d'identifiants ou de profils de nœuds communément appelée vue. Les coûts en terme de mémoire et de trafic réseau pour assurer une connaissance globale du système étant élevés dans un environnement dynamique où les nœuds quittent et rejoignent le réseau de manière continue, nous nous intéresserons ici aux mécanismes fournissant des connaissances partielles. Ces mécanismes peuvent être classés en deux familles [72] : les mécanismes d'adhésion basiques et les mécanismes d'adhésion à deux overlays. Nous détaillerons ces deux mécanismes dans la suite. Plusieurs protocoles de « gossip » ont ensuite été implémentés audessus de ces mécanismes d'adhésion.

2.1 Protocoles de gossip à mécanismes d'adhésion basiques

Nous nous intéressons ici aux protocoles de gossip qui utilisent des mécanismes d'adhésion basiques. Dans ce type de mécanisme, la vue fournie à chaque nœud est composée de pairs dispersés à travers le réseau ; aucune caractéristique n'est considérée sur un nœud pour l'intégrer dans la vue d'un autre. Ce type de mécanisme peut être décomposé en deux sous-mécanismes :

- **les systèmes centralisés** qui utilisent un ensemble de serveurs centraux ayant pour principale tâche de fournir à chaque nœud une vue aléatoire. Les informations relatives à chaque pair sont ainsi dupliquées au niveau des serveurs. Ces serveurs jouent aussi le rôle de contact pour les nouveaux nœuds adhérents dont les identifiants seront communiqués aux autres nœuds.
- **les systèmes décentralisés** qui ont pour principale caractéristique l'auto-organisation. Chaque nœud est doté de manière automatique d'une vue aléatoire. Ces systèmes sont totalement indépendants et les adhésions et échecs de nœuds sont gérés localement sans serveur central.

Nous nous intéresserons ici aux protocoles qui adoptent des architectures décentralisées qui permettent d'éviter en particulier les problèmes liés à la taille de la mémoire de stockage et aux échecs des serveurs centraux (Illustration : **Figure** 1).
Relativement à la question Q1 du chapitre précédent, nous détaillons dans la suite plusieurs de ces protocoles bien connus dans la littérature en les classant dans deux groupes : (1) les protocoles utilisant une sélection aléatoire de pairs lors de l'exécution et (2) ceux utilisant une sélection déterministe.

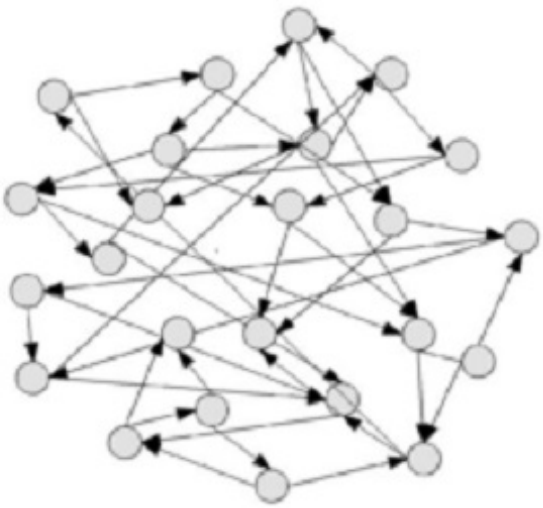


FIGURE 1: Exemple de mécanisme d'adhésion basique. Sur chaque nœud, les flèches sortants indiquent les nœuds voisins constituant sa vue locale. Les flèches entrants représentent les vues des nœuds voisins dans lesquelles le nœud est intégré.

2.1.1 Sélection aléatoire de pair

Ces protocoles de gossip ont la particularité de reposer totalement sur des choix aléatoires. Plus précisément, à l'exécution, la sélection de pairs pour la diffusion d'information est faite de manière aléatoire parmi les pairs composant la vue du nœud. Tous les nœuds composant la vue ont en principe la même probabilité d'être sollicités pour l'échange. Ceci permet de maintenir l'architecture connectée et de se rapprocher des propriétés d'un graphe aléatoire. Le défis est donc de construire et maintenir le graphe connecté en présence de défaillances de noeuds.
Le choix du paramètre fanout à appliquer sur le protocole est déterminant dans le sens où il impacte particulièrement sur la vitesse de propagation des messages. D'ou l'importance de trouver une bonne valeur du fanout.

Ganesh *et al.* [48] présentent SCAMP un service de pair-à-pair qui fonctionne de manière complètement décentralisée, où aucun nœud n'a une connaissance globale des membres. Toutefois, les informations sur les membres sont suffisamment répliquées pour maintenir les « gossip » avec une fiabilité élevée. Le système est totalement auto-organisé dans le sens où la taille des vues locales converge naturellement vers la valeur « juste » pour que les « gossip » réussissent. Cette valeur est fonction de la taille du système, mais est obtenue sans qu'aucun nœud ne connaisse la taille réelle du système.
Kermarrec *et al.* [72] avaient proposé un schéma dans lequel un ensemble de serveurs maintient la liste globale des membres et fournit à chaque nœud une vue partielle aléatoire. Ainsi, les nœuds n'ont pas tous besoin d'avoir des informations globales, mais simplement des informations de chaque membre dans leur liste locale. SCAMP élimine ce besoin de serveurs et décrit un schéma entièrement décentralisé qui atteint les mêmes objectifs : les nœuds obtiennent une vue partielle aléatoire du système et la taille de cette vue se mesure automatiquement avec la taille du système, même si aucun nœud ne connaît la taille du système. SCAMP est entièrement décentralisé dans le sens où chaque nœud ne conserve qu'une vue partielle du système. Il est

également auto-organisé : la taille des vues partielles augmente naturellement avec le nombre d'adhésion afin d'assurer les mêmes garanties de fiabilité que lorsque le groupe se développe. L'une des principales contributions est l'analyse théorique de SCAMP, qui établit des garanties probabilistes sur sa performance. Elle est confirmée par des simulations qui montrent qu'un protocole de « gossip » utilisant SCAMP comme un service d'adhésion est presque aussi résistant aux échecs qu'un protocole reposant sur la connaissance de l'appartenance globale à chaque nœud.

Jelasity *et al.* [64] introduisent un schéma de protocole générique dans lequel les implémentations connues et novatrices de « gossip » des services d'échantillonnage par les pairs peuvent être instanciées et présentent une comparaison empirique étendue de ces protocoles. Ils identifient un nouveau service abstrait, « le service d'échantillonnage par les pairs », qui est un élément fondamental sous-jacent des protocoles de « gossip ». Ce service est donc indispensable pour les implémentations de « gossip » d'un large éventail de fonctions de niveau supérieur, qui incluent la diffusion de l'information, l'agrégation, la gestion du réseau et la synchronisation. A la suite de l'identification de ce service et de l'exécution de sa séparation logique dans une classe d'applications existantes, Jelasity *et al.*. présentent un schéma générique de protocole qui généralise les protocoles de service d'échantillonnage par les pairs de « gossip » existants. Le programme permet également de mettre en œuvre de nouveaux protocoles.
L'API du service d'échantillonnage par les pairs est extrêmement simple et consiste en deux méthodes : *Init()* et *GetPeer()*. *Init()* Initialise le service sur un nœud donné si cela n'a pas été fait avant. La procédure d'initialisation proprement dite dépend de l'implémentation. *GetPeer()* renvoie une adresse de pair si le groupe contient plus d'un nœud. L'adresse renvoyée est un échantillon tiré du groupe. La spécification de cet échantillon (aléatoire, corrélation dans le temps et avec d'autres pairs) dépend de la mise en œuvre.

Jelasity *et al.* [68] présentent un framework générique qui met en œuvre un service d'échantillonnage par les pairs de manière décentralisée en construisant et en maintenant des architectures dynamiques et non structurées à partir d'informations sur les membres. Le framework généralise les approches existantes et facilite la découverte de nouvelles approches. Il permet aux concepteurs d'explorer et comparer empiriquement plusieurs implémentations du service d'échantillonnage par les pairs.
Le service d'échantillonnage par les pairs est distingué de l'application qui l'utilise et, abstraitement, le même service peut être utilisé dans différents contextes : diffusion de l'information [32][41], l'agrégation [71][82][66][65], l'équilibrage de charge [64] et la gestion des réseaux [126][128]. Le service est promu comme une abstraction de première classe d'un système distribué à grande échelle.
Le principe général de base sous-jacent au framework proposé pour concevoir le service d'échantillonnage par les pairs, est lui-même basé sur un paradigme de « gossip ». En réalité, chaque nœud (1) maintient une table d'appartenance locale relativement petite qui fournit une vue partielle sur l'ensemble complet de noeuds et (2) rafraîchit périodiquement la table en utilisant une procédure de « gossip ». Le framework est générique et peut être utilisé pour instancier des implémentations de « gossip » connues [40][63][126] et nouvelles. En fait, le framework capture de nombreuses variantes possibles de la diffusion d'information des membres basées « gossip ». Ces variantes diffèrent principalement dans la manière dont la table d'adhésion est mise à jour dans un nœud donné après l'échange de tables dans un moment de « gossip ».
Plusieurs dimensions sont considérées en identifiant les différences qualitatives entre les variantes examinées. Ces dimensions incluent le caractère aléatoire du choix d'un pair tel que perçu par un seul nœud, la précision de la vue courante du membre, la répartition de la charge sur chaque nœud, ainsi que la robustesse lors des scénarios d'échecs et de désabonnement.

Jelasity *et al.* ont ainsi montré que la communication devrait plutôt être bidirectionnelle : elle devrait suivre le modèle push-pull. Adhérer à une approche push-only ou pull-only peut facilement conduire à un partitionnement (irrévocable) de l'ensemble des nœuds. Une autre conclusion est que la robustesse contre la défaillance ou le désabonnement des nœuds peut être améliorée si les entrées de l'ancienne table sont supprimées lors de l'échange d'informations des membres.

Leitao *et al.* [77] proposent une approche permettant de mettre en œuvre des protocoles de diffusion et décrivent un protocole de membership qui permet d'utiliser cette approche avec succès. Ils présentent HyParView, un protocole de membership pour supporter le broadcast de « gossip » qui assure des niveaux élevés de fiabilité même en présence de taux élevés de défaillances de nœuds. HyParView repose sur une approche basée sur l'utilisation de deux vues partielles distinctes, qui sont maintenues avec des objectifs différents par des stratégies différentes. Le protocole repose sur les points suivants :

- Une stratégie de « gossip » basée sur l'utilisation d'un protocole de transport fiable, tel que TCP, pour les échanges entre pairs. De cette manière, les « gossip » n'ont pas besoin d'être configurés pour masquer les omissions du réseau.
- Chaque nœud maintient une petite vue symétrique active de la taille du fanout + 1. Le fanout peut être sélectionné en supposant que les liens n'omettent pas de messages ; la stratégie permet d'utiliser de plus petits fanouts que les protocoles qui utilisent le transport non fiable pour supporter les échanges de « gossip ». La diffusion est effectuée en inondant le graphe défini par les vues actives. Bien que ce graphe soit généré au hasard (en utilisant le service d'appartenance), le « gossip » est déterministe tant que le graphe reste inchangé.
- TCP est également utilisé comme détecteur de défaillances, et puisque tous les membres de la vue active sont testés à chaque étape de « gossip », la défaillance des nœuds dans la vue active est détectée rapidement.
- Chaque nœud maintient une vue passive des nœuds de sauvegarde qui peuvent être promus à la vue active, lorsqu'un des nœuds de la vue active échoue (c'est-à-dire se déconnecte, crashe ou bloque).
- Un protocole d'adhésion est chargé de maintenir la vue passive et de sélectionner les membres de la vue passive qui doivent être promus à la vue active. Deux vues sont maintenues par le protocole.

Le protocole HyParView maintient deux vues distinctes sur chaque nœud : une petite vue active, et une vue passive plus grande. Les vues actives de tous les nœuds créent une architecture qui est utilisée pour la diffusion des messages. Les liens dans l'architecture sont symétriques. Cela signifie que si le nœud q est dans la vue active du nœud p alors le nœud p est également dans la vue active du nœud q. Chaque nœud maintient également une vue passive plus grande. La vue passive n'est pas utilisée pour la diffusion des messages. Son objectif est de maintenir une liste de nœuds qui peuvent être utilisés pour remplacer les membres défaillants de la vue active. La vue passive est maintenue en utilisant une stratégie cyclique. Périodiquement, chaque nœud exécute une opération de « shuffle » avec l'un de ses voisins afin de mettre à jour sa vue passive. Lorsqu'un nœud souhaite joindre l'architecture, il doit connaître un autre nœud qui appartient déjà à la l'architecture. Ce nœud est appelé nœud de contact. Il établit une connexion TCP avec ce nœud de contact et lui envoie une requête d'adhésion.

Bortnikov *et al.* [15] présentent Brahms, un algorithme d'échantillonnage aléatoire de nœuds pour de grands systèmes dynamiques propice à des comportements malveillants. Brahms stocke de petites vues d'appartenance à chaque nœud, et surmonte aussi les attaques byzantines par une portion linéaire du système. Les attaques byzantines concernent ici, ceux visant à peupler

les vues des nœuds par des identifiants de pairs défectueux et les attaques visant à isoler un ensemble de nœuds. Brahms a deux composantes. Le composant d'échantillonnage local conserve une liste d'échantillons S, un tuple d'échantillons uniformes de l'ensemble des ids qui ont traversé le nœud. Le composant de « gossip » est un protocole distribué qui répand les identifiants localement connus à travers l'architecture et maintient une vue dynamique.
Même sans les échecs byzantins, le protocole d'adhésion garantit seulement que la représentation moyenne des nœuds dans les vues locales est uniforme [68], et non que chaque nœud obtienne un échantillon aléatoire uniforme et indépendant. Les nœuds défectueux peuvent tenter de biaiser la distribution à l'échelle du système, ainsi que la vue locale individuelle d'un nœud donné.
Brahms est un service de membership qui stocke un nombre sub-linéaire d'IDs dans chaque nœud, et fournit à chaque nœud des échantillons aléatoires de pairs qui convergent de manière uniforme dans le temps. Les principales idées sous-jacentes à Brahms sont les suivantes : 1) utiliser le protocole d'adhésion de gossip avec des mécanismes défenses supplémentaires pour le rendre viable dans un contexte d'attaques ; (2) de reconnaître qu'une telle solution est tenue de produire des vues biaisées en raison des attaques ; et (3) de corriger cette tendance sur chaque nœud. Plus précisément, chaque nœud maintient, en plus de la vue locale de « gossip », un échantillon non biaisé des IDs. Pour obtenir ce dernier, Bortnikov *et al.* introduisent le Sampler, un composant qui obtient des échantillons uniformes à partir d'un flux de données dans lequel les éléments se reproduisent avec un biais inconnu, en utilisant des permutations indépendantes de « min-wise » [17]. Brahms s'exécute en tours ("rounds") non synchronisés. Il utilise deux moyens de propagation : (1) push - envoi de l'identifiant du nœud à un autre nœud, et (2) pull - récupération de la vue à partir d'un autre nœud. Ces opérations ont deux objectifs différents : les pushs sont nécessaires pour renforcer les connaissances sur les nœuds sous-représentés dans les vues des autres nœuds, alors que les pulls sont nécessaires pour diffuser les connaissances existantes au sein du réseau. Une combinaison du push et du pull est nécessaire car la représentation des IDs propagés uniquement par pull décroît avec le temps, alors que la représentation des IDs propagés par push s'accroît.

2.1.2 Sélection déterministe de pair

Ces protocoles se différencient des protocoles à sélection aléatoire par le caractère déterministe de leurs procédures de sélection de pairs lors des échanges de données. Dans les protocoles précédents, bien que l'architecture reste effectivement connectée malgré son caractère dynamique, les liens entre pairs sont par défaut dépourvus de sens. Dans le cas d'un choix déterministe, la sélection est guidée par l'application de mécanismes basés sur des caractéristiques telles que "l'âge" du pair (temps de présence dans la vue), une distance métrique, une similarité (centres d'intérêt partagés, proximité sémantique, profil, etc.), l'ordonnancement ("Round Robin"), etc. Ce qui permet d'avoir des liens virtuels mais pertinents entre pairs dans l'architecture améliorant ainsi la qualité des vues.

Voulgaris *et al.* [126] décrivent un nouveau protocole de « gossip » de gestion des membres. Ce protocole est présenté pour construire des graphes de faibles diamètres, de faibles clusterings, de degrés très symétriques des nœuds, et qui sont très résistants aux défaillances massives des nœuds. Voulgaris *et al.* réalisent une analyse expérimentale du protocole de base pour les grands réseaux, en examinant les propriétés telles que le regroupement et la distribution des degrés des nœuds. Il en résulte que le shuffling est en effet un protocole d'échange prometteur. Plus particulièrement, ils décrivent CYCLON, un framework complet pour une gestion peu coûteuse des membres. CYCLON introduit une version améliorée du shuffling, qui se traduit par des distri-

butions de degré de nœud qui présentent de meilleures propriétés que celles trouvées dans les architectures résultantes de la conception de base, ou même dans les graphes aléatoires. De plus, il comprend une meilleure gestion, en termes d'efficacité et de qualité, des ajouts et des suppressions de nœuds, ce qui permet d'établir une gestion d'adhésion peu coûteuse qui ne perturbe pas le caractère aléatoire de l'architecture réseau. Inspiré des systèmes pair-à-pair non structuré de « gossip » CYCLON est une amélioration du protocole de shuffling basic [115].
Le shuffling amélioré suit le même modèle que le shuffling de base. La principale différence est que les nœuds ne choisissent pas au hasard avec quel voisin échanger le cache. A la place, ils sélectionnent leur voisin le plus ancien. Les échanges de voisins sont lancés de manière périodique, mais non synchronisée, à une période fixe ΔT. En plus de l'adresse réseau, les entrées des caches contiennent un champ supplémentaire nommé *âge*, qui désigne approximativement l'âge de l'entrée exprimé en intervalles ΔT depuis le moment où il a été créé par le nœud vers lequel il pointe. CYCLON est hautement évolutif, très robuste et complètement décentralisé.

Nedelec *et al.* [85] abordent les limites des approches actuelles d'échantillonnage par les pairs en introduisant Spray un protocole d'échantillonnage aléatoire de pairs, inspiré de Scamp et Cyclon. Spray est conçu pour éviter les contraintes introduites par le WebRTC, plus précisément, par sa procédure d'établissement de connexion à trois éléments, en utilisant uniquement des interactions voisin-à-voisin. L'introduction du WebRTC a ouvert une nouvelle vision pour les applications distribuées à grande échelle consistant en un grand nombre de navigateurs Web à communication directe. Dans ce contexte, les protocoles d'échantillonnage par les pairs basés « gossip » apparaissent comme un outil particulièrement prometteur grâce à leur capacité inhérente à construire des architectures réseaux qui peuvent faire face à la dynamique du réseau.
L'utilisation de services de signalisation et des connexions du WebRTC existantes permet un déploiement aisé des protocoles d'échantillonnage par les pairs dans des navigateurs pouvant fonctionner sur des téléphones portables ou des tablettes, connectés à des réseaux mobiles. Dans ce contexte, il est essentiel de maintenir le nombre de connexions aussi bas que possible afin de réduire l'utilisation de trafic et de limiter la consommation de ressources.
Spray : 1-adapte dynamiquement le voisinage de chaque pair. Ainsi, le nombre de connexions croît de manière logarithmique par rapport à la taille du réseau ; 2- utilise uniquement des interactions entre voisins pour établir des connexions. Ainsi, les connexions sont établies en temps constant ; 3-converge rapidement vers une topologie exposant des propriétés similaires à celles d'un graphe aléatoire. Ainsi, le réseau devient robuste aux défaillances massives et diffuse efficacement l'information.
Nédelec *et al.* démontrent ainsi comment l'adaptabilité de Spray améliore les performances de l'échantillonnage aléatoire par les pairs lorsque la taille du réseau est en évolution. Aussi, Spray reste robuste aux défaillances massives.
Spray et Cyclon sont assez similaires lorsque la taille du réseau est optimale pour Cyclon. Toutefois, Spray conserve les connexions lorsque Cyclon est surdimensionné et est plus robuste lorsque Cyclon est sous-dimensionné.
L'adaptabilité vient du coût des doublons dans les vues partielles. Cependant, les simulations soutenues par une analyse théorique montrent que le nombre de doublons reste très faible et devient négligeable dans les grands réseaux.

2.1.3 Synthèse

Le tableau suivant présente une synthèse des différents protocoles de gossip présentés dans cette partie. Chaque ligne représente une caractéristique nous permettant de comparer ces pro-

tocoles :

- **Push-propagation** : le protocole repose sur un flux de type "push". Dans ce type de propagation, chaque nœud envoie ses données à certains pairs choisis au hasard, mais n'attend aucune réponse de ces pairs. C'est un mécanisme très adapté à la diffusion d'informations, puisqu'il n'y a dans ce cas pas besoin pour les destinataires de répondre aux expéditeurs.
- **Pull-propagation** : le protocole repose sur un flux de type "pull". La « pull-propagation » garantit que les données ne sont transférées que lorsque cela est nécessaire. Il permet de réduire le trafic réseau lorsque la taille des données à échanger est particulièrement importante.
- **Exécution Cyclique** : le protocole est exécuté d'une manière cyclique au niveau de chaque pair.
- **Sélection de pair biaisée** : le choix d'un pair de la vue lors de l'exécution du protocole est fait de manière déterministe. Un mécanisme de sélection est appliqué pour choisir le pair le mieux adapté (le plus ancien(âge), le plus proche (distance métrique), métrique de similarité, etc.)
- **Mécanisme de défense** : le protocole intègre un ou plusieurs mécanismes de sécurité pour maintenir l'architecture connectée et/ou préserver l'anonymat de chaque pair.
- **WebRTC** : le protocole repose sur le framework WebRTC qui permet d'établir des canaux de communication navigateur-à-navigateur.

	SCAMP [2003]	CYCLON [2005]	HyParView [2007]	Brahms [2009]	SPRAY [2015]
Push-propagation	√	√	○	√	√
Pull-propagation			○	√	√
Exécution cyclique		√	○	√	√
Sélection de pair biaisée		√			√
Mécanisme de défense				√	
WebRTC		√			√

TABLE 2: Tableau de synthèse des protocoles de gossip plats.
La prise en charge totale d'une caractéristique est indiquée par le signe √, la prise en charge partielle par ○, la non prise en charge par une case vide

2.2 Protocoles de gossip à mécanismes d'adhésion à deux overlays

Une deuxième famille de protocoles est conçue au-dessus des protocoles de gestion décrits précédemment en ajoutant des mécanismes de clustering pour former des architectures à deux niveaux. Les protocoles de gossip basiques ont ici pour rôle de construire et maintenir une architecture connectée au-dessus de laquelle est appliqué ensuite un mécanisme de clustering approprié.

L'objectif commun des architectures présentées dans cette partie consiste donc à clusteriser les nœuds selon un critère de proximité géographique, sémantique, de profil ou de réseau et prendre en compte cette proximité pour fournir à chaque nœud une liste locale de ses voisins

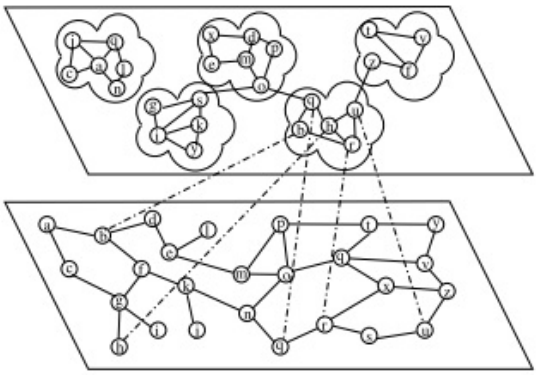


FIGURE 2: Exemple de mécanisme d'adhésion avec deux overlays. Source : [13]

composée exclusivement de nœuds appartenant au même cluster. Certains modèles sont dotés de mécanismes permettant à quelques nœuds choisis dans le cluster d'être munis d'une vue distante composée de nœuds appartenant à d'autres clusters dans le but de maintenir tout le système connecté (Illustration : **Figure** 2). Kermarrec *et al.* [72] montrent justement qu'un petit nombre de liens entre les clusters suffit pour assurer une telle connectivité. Plusieurs modèles de protocoles hiérarchiques ont été proposés. Ces modèles peuvent être classés en deux familles : les protocoles hiérarchiques sans serveur de contact et les protocoles hiérarchiques avec serveur de contact. Relativement à la question **Q1** du chapitre précédent nous regardons ici plusieurs de ces solutions.

2.2.1 Protocoles autonomes à deux overlays

Ces protocoles ont la particularité d'être complètement autonomes dans le sens ou tous les nœuds ont les mêmes rôles. La procédure d'adhésion à l'architecture pour un nœud nouvellement arrivé ne nécessite pas l'entrée en contact avec un nœud particulier (serveur de contact) ; chaque nœud peut servir de contact pour un nouveau. Les mécanismes d'adhésion diffèrent d'un protocole à un autre, le but étant de trouver un premier pair qui réponde favorablement à la requête d'adhésion. Ce dernier pair procédera au transfert des informations relatives à sa vue selon le type de flux (push et/ou pull) adopté par le protocole. Ces informations permettront ainsi au nœud à l'origine de la requête d'initialiser sa vue.

Voulgaris *et al.* [129] proposent une méthode proactive pour construire une couverture sémantique. La méthode est basée sur un protocole épidémique qui regroupe des pairs ayant des contenus similaires. Ce regroupement de pairs se fait d'une manière complètement implicite, c'est-à-dire sans exiger de l'utilisateur de spécifier ses préférences ou de caractériser le contenu des fichiers qu'il partage.
Dans ce modèle, chaque pair maintient une liste dynamique de voisins sémantiques, appelée sa vue sémantique, de petite taille fixe. Un nœud recherche un fichier en interrogeant d'abord ses voisins sémantiques. Si aucun résultat n'est renvoyé, il utilise alors le mécanisme de recherche par défaut. La probabilité qu'un voisin satisfasse une requête d'un pair est proportionnelle à la proximité sémantique entre le pair et son voisin. Le modèle suppose l'existence d'une fonction de proximité sémantique $S(F_P,F_Q)$, qui, étant donné les listes de fichiers F_P et F_Q des pairs

P et Q, respectivement, fournit une métrique numérique de la proximité sémantique entre les deux pairs. Dans le but d'exploiter le caractère transitif de S et de prendre en compte l'importance d'examiner tous les nœuds lors de la construction de la vue sémantique, le système adopte une approche à deux couches avec des protocoles de « gossip ». La couche inférieure, appelée CYCLON [126], est responsable du maintien d'une architecture connectée et de l'alimentation périodique le protocole de couche supérieure avec des nœuds uniformément sélectionnés de façon aléatoire à travers le réseau. À son tour, le protocole de couche supérieure, appelé VICINITY, est chargé de la découverte de pairs qui sont sémantiquement les plus proche, et d'ajouter ces nœuds aux vues sémantiques.
Cette approche en deux couches conduit à des architectures réseaux sémantiques de haute qualité.

Jelasity *et al.* [67] proposent un protocole de « gossip » appelé T-MAN qui peut construire une large gamme d'architectures réseaux, en se basant uniquement sur des hypothèses minimales. Le protocole est rapide, robuste et très simple. Il est également hautement configurable car la topologie souhaitée est elle-même un paramètre sous la forme d'une méthode de classement. Le papier décrit également une application pratique de T-MAN pour construire efficacement des architectures de table de hachage distribués CHORD. L'objectif principal ici est de montrer qu'un seul algorithme générique basé « gossip » peut créer de manière rapide et efficace de nombreuses architectures réseaux différentes.
T-MAN s'appuie uniquement sur un service d'échantillonnage par les pairs [68] sous-jacent qui crée une architecture réseau initial avec des liens aléatoires comme point de départ. Le protocole repose principalement sur trois paramètres : la taille m des messages, le paramètre w du service d'échantillonnage par les pairs et la méthode de classement $RANK$. Une façon de définir réellement des méthodes de classement utiles est d'utiliser une fonction de distance qui définit un espace métrique sur l'ensemble des nœuds. Il est important de noter qu'il existe des graphes cibles d'intérêt pratiques qui ne peuvent pas être définis par une fonction de distance globale. C'est la raison principale de l'utilisation de méthode de classement, par opposition à se fier exclusivement à la notion de distance ; la méthode de classement est un concept plus général que la distance.
La méthode de classement formalise l'idée suivante : face à un ensemble de nœuds, chaque nœud du réseau est capable de décider à quels nœuds de l'ensemble il ressemble le plus et lesquels il ressemble le moins. D'un point de vue formel, cette méthode est capable d'ordonner tout ensemble de nœuds ayant un nœud dit de base. L'algorithme est basé « gossip » : tous les nœuds communiquent périodiquement avec un voisin choisi au hasard et échangent des informations de voisinage afin d'améliorer la qualité de leur ensemble de voisins. Dans cette approche, chaque nœud a une adresse qui est nécessaire et suffisante pour lui envoyer un message. En outre, tous les nœuds ont un profil contenant des informations supplémentaires. Chaque nœud diffuse son descripteur à tous les autres nœuds de sorte que, finalement, chaque nœud a collecté localement les descripteurs de tous les nœuds du réseau. À ce stade, chaque nœud trie cet ensemble de descripteurs selon la méthode de classement et choisit les k premiers éléments comme étant ses voisins. La structure résultante est appelée un graphe cible.

Mordacchini *et al.* [83] proposent une architecture générale d'un système dédié à exploiter l'échange collaboratif d'informations entre pairs afin de construire un système capable de rassembler des utilisateurs similaires et de diffuser entre eux des recommandations utiles. Plus précisément, ils poussent plus loin l'idée d'exploiter des mécanismes de recommandation collaboratifs, basés sur le regroupement d'intérêts et obtenus grâce à des interactions entre utilisateurs. La proposition consiste à coupler les deux systèmes en exploitant des architectures réseaux pair-à-pair de type « gossip » afin de faciliter le regroupement d'utilisateurs ayant des intérêts similaires et ensuite d'utiliser les liens établis pour diffuser des recommandations à travers les pairs. L'objectif

est double. D'un côté, construire un système souple et adaptable qui permettra la création de communautés d'intérêts entre les utilisateurs de façon décentralisée et distribuée. D'autre part, exploiter de telles communautés non seulement pour partager les connaissances sur les éléments intéressants entre eux, mais aussi pour surmonter certains problèmes traditionnels des systèmes de recommandation. En particulier, l'un des problèmes les plus courants concerne la possibilité de recommander de nouveaux éléments.
Pour grouper des utilisateurs similaires, le protocole fonctionne au moyen d'un algorithme de clustering. Tout d'abord, chaque pair décide, indépendamment, qui sont les pairs auxquels il est lié. Ces relations individuelles sont choisies sur la base d'une distance basée sur les intérêts, parmi les pairs rencontrés. Chaque fois qu'il entre en contact avec un nouveau pair, il peut apprendre de l'existence de nouveaux voisins potentiels (c'est-à-dire, des utilisateurs similaires) et ensuite communiquer avec eux. Lorsque ce processus est stabilisé, un pair peut considérer son voisinage comme la représentation d'une communauté d'intérêt commun. Un point important ici est que les pairs se caractérisent par des intérêts multiples. Par conséquent, le processus décrit ci-dessus est mené séparément pour chacun des intérêts d'un pair. Les connexions sont établies et maintenues séparément pour chaque intérêt distinct. Ainsi, il existe un ensemble d'architectures virtuelles différentes, où chaque pair participe à autant de groupes nécessaires pour couvrir ses intérêts.
Ce processus se fait d'une manière auto-organisée et complètement décentralisée, en utilisant une communication de type « gossip ». Chaque pair connaît un ensemble d'autres pairs, à savoir ses voisins d'intérêt, et essaie périodiquement de choisir de nouveaux voisins qui sont plus proches de son intérêt que les précédents. Cela se fait simplement en apprenant à connaître de nouveaux pairs à partir d'un autre pair, puis en récupérant leurs profils, et finalement en choisissant les voisins les plus proches dans l'union des voisins actuels et potentiels.
Lorsqu'un pair entre dans le réseau, il est mis en contact avec un ou plusieurs pairs participant déjà au réseau de proximité d'intérêt. Ils utilisent la fonction de similarité de profil pour calculer leur similarité. Ils considèrent séparément chaque intérêt de l'ensemble des intérêts du nœud et le confrontent avec les leurs.

Bertier *et al.* [13] décrivent la construction d'un réseau de connaissances sociales anonymes à l'aide d'un protocole de "gossip" appelé Gossple, et la façon de tirer parti d'un tel réseau pour améliorer la navigation dans les applications collaboratives du Web 2.0.
Les nœuds de Gossple (utilisateurs) font périodiquement des spéculations sur leurs profils d'intérêt et calculent leurs distances (en termes d'intérêt) par rapport aux autres nœuds. Ceci est réalisé avec peu de bande passante et de stockage, avec une convergence rapide, et sans révéler quel profil est associé à quel utilisateur.
Bertier *et al.* proposent d'améliorer la navigation dans des systèmes du web 2.0 par une personnalisation implicite : chaque utilisateur est associé à un réseau de connaissances anonymes qui s'intéressent à des sujets similaires, indépendamment de la façon dont ils ont exprimé leurs intérêts (par exemple, les mots clés qu'ils ont utilisés pour étiqueter ces sujets). Ces connaissances sont alors implicitement utilisées pour guider et raffiner les opérations de recherche des utilisateurs.
Bertier *et al.* présentent Gossple, un système qui infère automatiquement des connexions personnalisées dans les systèmes à l'échelle d'Internet. Les nœuds de Gossple (utilisateurs) font des spéculations continues sur les profils marqué (de leurs utilisateurs correspondants) et calculent localement une vue personnalisée de l'architecture, qui est ensuite exploitée pour améliorer leur navigation sur le Web.
La vue couvre les intérêts multiples sans aucun soutien explicite (tels que les liens sociaux explicites ou l'ontologie) et sans violer l'anonymat : l'association entre les utilisateurs et les profils est masquée.

Fondamentalement, chaque nœud de Gossple a un proxy, choisi aléatoirement, spéculant en son nom, sur son résumé de profil. Le nœud transmet son profil à son proxy de manière cryptée par un intermédiaire qui ne pourra pas déchiffrer le profil.
Pour réduire la consommation de bande passante, la procédure d'échange des spéculations est économe : les nœuds n'échangent pas de profils mais seulement des filtres de Bloom [14] jusqu'à ce que le calcul de similarité révèle que les deux nœuds pourraient en effet bénéficier de l'échange. Pour limiter le nombre de profils maintenus par chaque nœud, tout en englobant les divers intérêts de l'utilisateur associé au nœud, les auteurs introduisent une métrique de similarité appelée ensemble de similarité du cosinus, comme une généralisation du classique métrique de similarité du cosinus [137][22], ainsi qu'une heuristique efficace pour calculer cette nouvelle métrique.
Gossple fournit à chaque utilisateur un GNet, un réseau de profils d'intérêts anonymes sémantiquement proches. La création et la maintenance du GNet d'un nœud se font d'abord en déterminant une métrique pour identifier les profils qui présentent des intérêts similaires. Un autre défi consiste à concevoir une procédure de communication efficace pour l'échange de profils tout en limitant la quantité de trafic réseau générée et en cachant l'association entre les nœuds et les profils pour préserver l'anonymat.
La métrique multi-intérêt de Gossple est essentielle pour sélectionner le meilleur ensemble de profils pour remplir le GNet d'un nœud donné. Cela serait cependant peu utile sans un mécanisme efficace pour examiner un large ensemble de profils candidats tout en assurant une convergence rapide vers le GNet idéal. Gossple réalise ceci grâce à un protocole de "gossip" pour établir et maintenir des connexions entre des nœuds. Le protocole Gossple repose sur deux sous-protocoles : un protocole d'échantillonnage des pairs (RPS) [68] et un protocole de clustering multi-intérêts (protocole GNet). Chaque nœud maintient deux structures de données correspondantes, une vue aléatoire et le GNet.

Par le biais de VICINITY, un Framework générique de gestion de couverture fondé sur l'auto-organisation, Voulgaris *et al.* [130] explorent les compromis entre la prise de décision déterministe et probabiliste pour la structuration des couvertures. Le Framework est alors utilisé pour démontrer l'importance des décisions à caractère aléatoire. La principale contribution est d'explorer systématiquement l'effet du hasard dans la construction d'architecture de « gossip ». Comme un effet secondaire de cette exploration, est présenté VICINITY, un nouvel algorithme de « gossip » qui peut être déployé pour une large gamme d'applications. Dans cette approche, l'information sur les voisins est stockée et échangée au moyen de descripteurs de nœud. Ce dernier est un tuple contenant les trois champs suivants : l'adresse du nœud (c'est-à-dire l'adresse IP et le port) ; l'âge du descripteur (un champ numérique) ; et le profil du nœud spécifique à l'application. Les nœuds participent à un service d'échantillonnage par les pairs, ce qui leur donne un flux continu de liens vers des nœuds choisis uniformément au hasard parmi tous les nœuds actifs. Voulgaris *et al.* considèrent également une fonction de sélection *SELECT (p, D, k)*, qui, étant donné le descripteur du nœud p et un ensemble D de descripteurs de nœud, renvoie l'ensemble des k descripteurs qui correspondent le mieux aux liens sortants de p dans la structure cible. Cette fonction de sélection est souvent basée sur une métrique de proximité de nœud globalement définie.
Le protocole est construit sur deux couches. La couche inférieure est le service d'échantillonnage par les pairs, responsable du maintien d'une architecture connectée et de l'alimentation périodique du protocole de couche supérieure avec des nœuds uniformément choisis de manière aléatoire dans l'ensemble du réseau. À son tour, le protocole de couche supérieure, appelé VICINITY, est chargé de découvrir les nœuds favorisés par la fonction de sélection. Chaque couche maintient sa propre vue séparée et communique avec la couche respective des autres nœuds.
Dans la version de base du protocole qui est complètement équivalent à T-Man, chaque nœud contacte périodiquement un nœud aléatoire à partir de sa vue et les deux nœuds se renvoient

les meilleurs voisins qu'ils ont dans leurs vues. Un certain nombre de choix de conception intéressants on par la suite considérablement améliorer la performance du protocole VICINITY de base. Plutôt que de choisir de manière aléatoire dans sa vue, une sélection « round robin » des partenaires de « gossip » est adoptée. Une autre façon de tirer davantage profit d'un unique échange de « gossip » consiste à augmenter la diversité des descripteurs échangés entre les nœuds. À cet égard, le thread passif d'un nœud devrait exclure tous les descripteurs reçus de l'ensemble des descripteurs potentiels à renvoyer. Aussi, dans le but de briser la boucle fermée sur l'information structurelle, le hasard est introduit comme deuxième canal d'entrée. Plutôt que d'avoir des nœuds découvrant de nouveaux voisins exclusivement à travers les liens structurels de leurs voisins actuels, il leur est offert également la possibilité d'échantillonner des nœuds de tout le réseau de manière aléatoire. À cette fin, CYCLON [126] est utilisé comme un protocole d'échantillonnage par les pairs, pour fournir des nœuds avec un flux de voisins aléatoires. Dans chaque cycle, le thread actif de chaque nœud tire les descripteurs aléatoires fournis par son instance CYCLON, les fusionne avec sa vue VICINITY normale et filtre l'union via la fonction *SELECT* pour conserver les meilleurs voisins. De cette façon, si CYCLON rencontre un bon voisin par hasard, ce voisin est repris par VICINITY pour améliorer sa vue. Une optimisation finale consiste à emprunter les liens aléatoires obtenus grâce à CYCLON non seulement pour améliorer les propres liens de structure d'un nœud, mais aussi pour améliorer la qualité des liens qu'il envoie à d'autres nœuds.
VICINITY se réfère ainsi à la version complète du protocole, y compris toutes les optimisations de conception précédemment présentée.

Frey *et al.* [47] présentent Behave, une architecture de mise en cache décentralisée reposant sur des positions comportementales et qui exploite les protocoles épidémiques pour construire des communautés superposées de pairs ayant des intérêts similaires.
Contrairement aux solutions DHT (distributed hash tables) qui cherchent à maximiser la quantité d'objets Web accessibles par le substrat pair-à-pair, Behave traite de la maximisation du nombre d'objets accessibles sans délai de recherche.
Pour ce faire, Behave construit un cache de comportement collaboratif en s'appuyant sur le filtrage collaboratif basé sur l'utilisateur [37], une technologie bien connue dans le contexte des systèmes de recommandation. Si deux utilisateurs ont des éléments communs dans leurs caches Web, alors ils vont probablement exposer des points communs par rapport aux sites Web qu'ils visiteront à l'avenir.
Les nœuds de Behave adoptent un protocole épidémique basé sur les points communs entre leurs historiques de navigation pour former une topologie basée sur les intérêts. Cela fournit à chaque nœud un ensemble de voisins superposés dont l'historique de navigation ressemble le plus à la sienne. Le cache comportemental de Behave émerge de cette topologie comme la fédération des caches locaux des voisins d'un nœud.
Frey *et al.* considèrent un cache local basé sur le LRU (less recently used) qui associe chaque URL visitée par l'utilisateur avec l'objet Web correspondant. Chaque nœud échange le contenu de son index de cache avec d'autres Nœuds similaires. Procéder à cet échange sans aucune forme de compression entraînerait une surcharge prohibitive du réseau. Behave utilise une représentation compacte des index de cache locaux sous la forme d'un bloom-filter [14]. Un bloom-filter représente un ensemble sous la forme d'un tableau de m bits en exploitant k fonctions de hachage.
Pour améliorer la précision de leurs historiques de navigation, chaque nœud conserve un profil d'intérêt séparé constitué d'une liste d'URL visitées. Tous les éléments du cache local et de l'index du cache local ont une entrée correspondante dans le profil d'intérêt. Cela permet aux nœuds de recueillir plus d'informations sur les historiques de navigation des voisins potentiels

que ceux qui seraient disponibles dans leurs index de cache local. Behave utilise également une représentation compacte du profil d'intérêt d'un nœud sous la forme d'un bloom-filter, qui est re-calculé périodiquement avec une taille appropriée.
Pour regrouper les nœuds en fonction des intérêts, Behave adopte l'approche dans [13], constituée de deux protocoles spéculatifs en couche : échantillonnage aléatoire par les pairs et clustering. Le premier fournit à chaque nœud une vue aléatoire en continu de l'architecture. Le dernier part de cette vue aléatoire et identifie progressivement les meilleurs voisins pour chaque nœud en fonction d'une métrique de similarité donnée.

2.2.2 Protocoles à deux overlays intégrant un serveur de contact

Contrairement à ceux précédemment cités, ces protocoles reposent sur l'existence d'un nœud particulier faisant office de contact pour tous les nouveaux nœuds désirant intégrer l'architecture. Le nœud de contact détient tout ou partie des informations relatives à chaque pair composant l'architecture. Il fournit à chaque nouveau pair une liste de profils de pairs pour initialiser sa vue. Pour parer à des scénarios ou le serveur fournit à un nouveau pair des informations sur des pairs indisponibles (par exemple : ayant déjà quittés le l'architecture), une procédure de mise à jour des informations détenues par le serveur est déclenchée de manière périodique ou réactive selon le protocole.

Kontominas *et al.* [73] présentent DS4 (Distributed Social and Semantic Search System), une architecture distribuée de partage de contenus qui permet aux utilisateurs de partager et de rechercher des contenus de manière entièrement décentralisée tout en maintenant le contrôle d'accès et la propriété de leurs contenus.
Dans DS4, l'organisation des nœuds est réalisée par un protocole de réassignation (rewiring protocol) périodiquement exécuté par chaque nœud. Ce protocole fonctionne en établissant des connexions entre des nœuds sémantiquement similaires (en plus des connexions sociales) et en rejetant les connexions qui sont obsolètes ou pointant vers des nœuds dissemblables. L'objectif du protocole d'assignation est de créer des clusters de nœuds avec des intérêts similaires. Chaque utilisateur conserve deux indices de routage contenant des informations pour des liens amis et des liens courts / longs portée vers d'autres nœuds de réseau. Les liens d'amis correspondent à l'aspect relation sociale du réseau, les liens à courte portée correspondent aux informations intra-cluster (les liens vers des nœuds avec des intérêts similaires), tandis que les liens à longue portée correspondent aux informations inter-cluster (ayant des intérêts différents). Ces derniers sont utilisés pour maintenir la connectivité des clusters distants dans le système. Lorsqu'un nœud utilisateur joint l'architecture DS4, ses intérêts sont automatiquement dérivés de son contenu local. Pour chaque intérêt, le nœud conserve un index sémantique (SI) contenant les détails de contact et les descriptions d'intérêt des nœuds partageant des intérêts similaires. Lors de la réassignation, le nœud calcule sa similarité moyenne à ses liens à courte portée contenus dans le SI comme mesure de cohésion de cluster. Si la similarité calculée est supérieure à un seuil alors le nœud n'a pas besoin de prendre d'autres mesures, puisqu'il est entouré par des nœuds avec des intérêts similaires. Sinon, le nœud lance un processus de raffinement de cluster en transmettant un message dans le réseau, avec un temps de vie (TTL), en utilisant les connexions sémantiques et sociales et en recueillant les intérêts des autres nœuds. Les requêtes sont émises sous forme de texte libre ou de mots-clés et sont formulées en tant que vecteurs à terme. Le nœud émettant la requête transmet un message dans le réseau avec un TTL utilisant à la fois ses connexions sociales et sémantiques. Initialement, l'initiateur du message compare la requête à ses intérêts et, si elle est similaire, la requête est transmise à tous ses liens à courte portée, c'est-à-dire que le

message est diffusé dans le voisinage du nœud. Sinon, la requête est transmise à un petit nombre fixe de nœuds qui ont la plus grande similitude avec la requête (fixed forwarding).

Boutet *et al.* [16] présentent HyRec, un système en ligne évolutif et rentable dédié à la personnalisation du filtrage collaboratif centré sur l'utilisateur. HyRec charge les tâches de recommandation sur les navigateurs Web des utilisateurs, tandis qu'un serveur orchestre le processus et gère les relations entre les profils d'utilisateur. Le système prédit les intérêts d'un utilisateur en recueillant des préférences ou des informations sur l'expérience de plusieurs autres utilisateurs (collaboration). Il adopte une stratégie dite du k-nearest-neighbor (KNN) ou k-meilleurs-voisins, qui consiste à calculer les k voisins les plus proches selon une métrique de similarité donnée (ici, métrique de similarité du cosinus) et à identifier les éléments à recommander à partir de cet ensemble de voisins.
L'architecture d'HyRec évite la nécessité de traiter les ensembles d'utilisateurs et d'articles au moyen d'une approche basée sur l'échantillonnage inspirée par l'informatique épidémique (« gossip »), et utilisée avec succès dans la littérature de la construction de graphes KNN ainsi que le traitement des requêtes Le serveur HyRec fournit au navigateur de chaque utilisateur un échantillon de profils d'autres utilisateurs (ensemble de candidats). Chaque navigateur calcule ensuite le KNN de son utilisateur et les éléments les plus pertinents basés sur cet échantillon. Le serveur délègue à la fois la sélection KNN et la recommandation d'articles aux navigateurs Web de l'utilisateur en utilisant une approche basée sur l'échantillonnage. Considérez un utilisateur, u, qui accède à une page Web avec des recommandations. Le serveur met d'abord à jour le profil de u dans sa structure de données globale. Ensuite, il identifie un ensemble personnalisé de candidat pour u, contenant les profils des candidats pour la prochaine itération KNN, et l'envoie au navigateur, qui exécute un morceau de code JavaScript intégré à la page Web. Ce code calcule les éléments recommandés, effectue les calculs de similarité entre le profil local et ceux du jeu de candidat et envoie les résultats au serveur. Le processus est exécuté sans que HyRec ne révèle l'identité d'un utilisateur à d'autres utilisateurs. L'association utilisateur/profil est masquée par un alignement anonyme : périodiquement, les identificateurs des éléments et les utilisateurs des ensembles de candidats sont anonymement remaniés.
HyRec peut aussi exploiter des clients avec de petits périphériques mobiles sans affecter les activités des utilisateurs.

Carvajal *et al.* [21] présentent WebGC, une bibliothèque basée sur le WebRTC qui prend en charge la communication basée sur « gossip » entre les navigateurs Web et leur permet de fonctionner avec les applications Node-JS. WebGC repose aussi sur le framework SimplePeer qui prend également la forme d'une bibliothèque JavaScript et fournit une couche autour du WebRTC qui simplifie l'établissement de connexions de données entre pairs.
Les implémentations actuelles des réseaux sociaux décentralisés nécessitent que les utilisateurs installent des logiciels spécifiques pour gérer les protocoles sur lesquels ils s'appuient. Le framework WebRTC offre la possibilité de se passer de cette exigence en permettant d'exécuter des applications pair-à-pair directement dans les navigateurs Web sans avoir besoin de logiciels externes ou de plugins.
WebGC est une bibliothèque JavaScript qui fournit un framework simplifié pour la construction d'applications basées « gossip ». Cela inclut la mise en œuvre de composants standard tels que l'échantillonnage aléatoire de pairs [68] et les protocoles de clustering [129]. En plus, il améliore le protocole d'initiation de connexion du WebRTC au moyen d'un mécanisme de signalisation décentralisé. Ce service remplace le serveur de signalisation centralisé utilisé dans les applications WebRTC. WebGC intègre la signalisation dans les opérations des protocoles d'échantillonnage de pairs et de clustering. Dans ces deux types de protocoles, appelés ici protocoles de couver-

tures, les nœuds maintiennent des structures de données (vues) qui contiennent des références à d'autres nœuds et échangent périodiquement des messages contenant des sous-ensembles de leurs vues. Le service de signalisation décentralisé maintient une table de routage supplémentaire qui contient une entrée pour chacune des références de nœud qui apparaissent dans l'une des vues des protocoles de couverture en cours d'exécution. Chacune de ces entrées contient la référence d'un nœud médiateur, c'est-à-dire un nœud qui a une connexion active avec le nœud dans l'entrée. Lorsqu'un nœud doit établir une connexion avec un autre nœud de sa table, il contacte simplement le médiateur du nœud et l'utilise comme serveur de signalisation. L'architecture de WebGC est composée principalement de l'objet COORDINATOR qui instancie les protocoles de « gossip » et agit en intermédiaire et distribuant des messages entrants aux différents protocoles. La bibliothèque comprend la mise en œuvre de deux protocoles d'échantillonnage par les pairs, Cyclon[126] et le protocole générique de [68], ainsi qu'un protocole de clustering [13] [129]. Tous les protocoles implémentent l'interface GOSSIPPROTOCOL.

Folz *et al.* [45] présentent CyCLaDEs, une architecture réseau basée sur les similarités LDF (Linked Data Fragment). CyCLaDEs propose une approche permettant de construire un cache comportemental décentralisé hébergé par les clients LDF et basé sur les TPF (Triple-Pattern Fragments). Plus précisément, CyCLaDEs a pour objectif de construire un cache comportemental décentralisé pour le traitement des requêtes LDF en se basant sur les similarités des profils des clients LDF. Pour chaque client, CyCLaDEs sélectionne un nombre fixe des meilleurs clients similaires appelés « voisins » et établit une connexion directe avec eux. Lors d'un traitement de requête par un client donné, chaque triplet de pattern de sous-requête est recherché en premier sur le cache local, ensuite dans le cache de ses voisins et en dernier lieu sur le serveur LDF si nécessaire. CyCLaDEs s'appuie sur une architecture d'échantillonnage aléatoire de pairs pour la gestion de la composition des membres et sur une architecture clusterisée pour gérer les k-meilleurs voisins :

- Random Peer Sampling (RPS) : architecture qui maintient l'appartenance à travers les clients connectés. Chaque client maintient une vue partielle sur le l'architecture. La vue contient un sous ensemble aléatoire de nœuds. Périodiquement, chaque client sélectionne le plus ancien nœud de sa vue et ils échangent des parties de leurs vues. Cette vue est utilisée pour amorcer et maintenir le réseau de clustering. Le réseau RPS assure ainsi que tous les clients sont connectés à un graphe aléatoire.
- Clustering Overlay Network (CON) : Construit au-dessus de RPS, il clusterise les clients par le biais de leurs profiles. Chaque client maintient une seconde vue qui contient les k-meilleures voisins en accord avec la similarité de leurs profils avec celui du client. Cette vue est mise à jour au cours de la phase de shuffling, lorsqu'un client démarre un shuffling, il sélectionne le voisin le plus ancien dans sa vue RPS et ils échangent des informations de profils, si le client distant à de meilleurs voisins dans sa vue, la vue locale est mise à jour pour conserver les k-meilleurs voisins. Pour déterminer si un profil est meilleur qu'un autre, CyCLaDEs utilise le coefficient de similarité de Jaccard.

Nedelec *et al.* [84] présentent CRATE, un éditeur collaboratif décentralisé en temps réel qui s'exécute directement sur les navigateurs web grâce au WebRTC. Les éditeurs collaboratifs en temps réel permettent aux auteurs d'écrire simultanément sur des documents partagés. Les éditeurs les plus utilisés comme Google Docs sont basés sur des serveurs centraux, ce qui soulève des problèmes de confidentialité et d'évolutivité. Aussi, la gestion de plusieurs sessions d'éditions risque de surcharger le serveur central. Les propriétés de CRATE sont basées sur deux résultats scientifiques :

- Une structure de séquence répliquée avec des bornes supérieures sub-linéaires de la com-

plexité en espace, qui épargne l'éditeur de la lourde tâche de collecte des données erronées.
- Un protocole d'échantillonnage de pairs adapté, qui évite à l'éditeur le surdimensionnement des tables de routage.

Pour assurer la disponibilité et la réactivité des documents, CRATE adopte le schéma de réplication optimiste. Chaque éditeur reproduit le document localement et y effectue directement des opérations. Ensuite l'éditeur étend ses modifications à tous les autres participants. Le système est correct si et seulement si les utilisateurs lisent le même document. Cette propriété est appelée « strong eventual consistency » [107].
CRATE est construit sur quatre couches :

- L'interface utilisateur qui présente le document à l'utilisateur
- La couche structure de séquence qui représente le document local
- La couche causalité qui assure la causalité sémantique entre les opérations
- La couche réseau qui construit le réseau de navigateurs web pour chaque session d'édition et l'utilise pour propager les mises à jour aux autres collaborateurs.

Pour la construction du réseau de navigateurs, CRATE utilise le protocole d'échantillonnage aléatoire de pairs, SPRAY qui est doté de la technologie WebRTC qui permet d'établir des canaux de communication navigateur-à-navigateur. SPRAY fournit à chaque éditeur une table de voisinage locale qui permet la communication dans un sous-ensemble d'éditeurs. CRATE utilise le protocole SPRAY pour diffuser de manière évolutive toutes les opérations de la séquence répliquée à tous les collaborateurs. Le protocole s'opère en deux étapes :

- Chaque opération effectuée sur la copie locale du document conduit à la création d'un message. Ce dernier est envoyé à tout le voisinage fourni par SPRAY.
- Chaque éditeur recevant un tel message de diffusion le transmet de nouveau à son voisinage. Ainsi, les messages atteignent transitivement tous les collaborateurs de manière très efficace.

2.2.3 Synthèse

Le tableau suivant présente une synthèse des différentes solutions présentées dans cette partie. Chaque ligne représente une caractéristique nous permettant de comparer ces différents modèles :

- **Métrique de similarité** : le modèle utilise une métrique de similarité pour estimer la distance (en termes d'intérêts, de métrique, de sémantique ou de contenu) entre pairs.
- **Clustering par gossip** : l'algorithme de clustering utilisé suit, à l'exécution, le même mécanisme qu'un protocole de gossip.
- **Compression des données échangées** : une stratégie de compression est appliquée sur les données pour maintenir une fluidité des échanges dans le système.
- **Caching** : le modèle intègre une ou plusieurs techniques de cache pour le stockage des données et assure ainsi la disponibilité et l'accès locale.
- **Clustering sémantique** : l'algorithme de clustering est basé sur des données sémantiques ; plus précisément, la métrique de similarité utilisée procède à une estimation de la distance sémantique entre pairs.
- **MMP** : Membership Management Protocol. C'est le protocole de base (protocol de gossip) utilisé dans le modèle pour construire et maintenir la connectivité de l'architecture.
- **Serveur de contact** : l'arrivée d'un nouveau pair dans l'architecture se fait à travers un serveur de contact dont le rôle principal est de stocker tout ou une partie des informations de chaque pair et de fournir à chaque nouvel arrivant les donnée nécessaires à son intégration (vue, adresse IP, identifiant, etc.)
- **Géocaching** : le modèle intègre un mécanisme de cache combiné à la géolocalisation. Les pairs situés à des distances géographiques proches peuvent formés spontanément (selon

leurs similarités) un cache collaboratif local constitué des caches locaux de chaque pair.

	Voulgaris et al. [2005]	Tman [2009]	Baraglia et al. [2010]	Gossple [2010]	DS4 [2013]	Vicinity [2013]	Behave [2014]	HyRec [2014]	Webgc [2015]	Cyclades [2016]	Crate [2016]
Métrique de similarité	√		√	√	√	√	√	√	√	√	
Clustering par gossip			√	√	√	√	√		√	√	
Double vue	√			√	√	√	√		√	√	
Compression de données				√			√				
Caching	√						√			√	
Clustering séman-tique	√	√			√				√		
MMP utilisé	Cyclon	Jelasity [68]	Cyclon	brahms		Cyclon	brahms		Cyclon	brahms	Spray
Serveur de contact								√		√	√

TABLE 3: Tableau de synthèse des protocoles de gestion à deux overlays.
La prise en charge totale d'une caractéristique est indiquée par le signe √, la non prise en charge par une case vide

3 Ontologie et Web Sémantique en pair-à-pair

Dans cette section nous regardons maintenant les architectures pair-à-pair qui utilisent ou supportent des systèmes à base de connaissances et notamment des approches reposant sur des ontologies, les formalismes du Web Sémantique ou le Web des données liées.

Les descriptions de données et de ressources détenues par des pairs dans un système pair-à-pair sont représentées de manière hétérogène selon différents aspects. Par exemple, les données peuvent être dans des fichiers XML, des tables relationnelles, des fichiers texte ou des documents RDF. Même lorsque le même type de format de représentation est utilisé pour stocker des informations, la structure et la sémantique des concepts utilisés dans la modélisation peuvent varier entre pairs. Un exemple de différences sémantiques serait l'utilisation de vocabulaires différents pour désigner le même objet physique ou conceptuel par différentes représentations d'information.

Dans un environnement ouvert et dynamique comme une architecture pair-à-pair, l'hypothèse de pairs partageant la même ontologie n'est pas réaliste. Mais si les pairs utilisent des ontologies différentes, il doit y avoir un moyen de relier les ontologies et de traduire les requêtes afin que leurs destinataires puissent les traiter. En général, cela se fait au moyen d'alignements (ensembles de correspondances entre des entités ontologiques sémantiquement liées) et trouver des alignements est le but du couplage d'ontologies. Un graphe de connaissance représente les connaissances des pairs (ou voisins) dans l'architecture. Un lien entre deux pairs reflète le fait qu'ils connaissent l'existence l'un de l'autre. En outre, il est supposé qu'il existe un alignement entre leurs ontologies respectives.

Pour exploiter efficacement la puissance des systèmes pair-à-pair, le problème de la représentation hétérogène de l'information doit être surmonté. Dans cette optique, plusieurs stratégies ont été proposées dans la littérature.

3.1 Cas des systèmes pair-à-pair pures

Dans ce type de d'architecture, tous les pairs sont conceptuellement égaux et ont les mêmes rôles. Le système ne comporte pas de serveur central. Chaque pair est connecté à un sous-ensemble de pairs (ses voisins) et toutes ses requêtes transitent par un ou plusieurs pairs de ce sous-ensemble. Dans la littérature, plusieurs travaux ont adopté ce modèle.

Ehrig *et al.* [36] présentent SWAP (The Semantic Web and peer-to-peer). Le projet a pour objectif de fournir un système pair-à-pair qui intègre les aspects sémantiques, afin d'améliorer les services proposés. Pour cela, SWAP essaye de combiner le paradigme pair-à-pair avec les technologies du Web sémantique. SWAP suppose que tous les pairs du réseau gèrent un ensemble de connaissances propriétaires qui sont partagées par les autres pairs. Il propose un modèle de métadonnées qui fournit une sémantique pour annoter les données internes et externes. Le modèle de métadonnées est spécifié en RDF(S) et il se compose de deux classes ; Swabbi et Peer. La classe Peer permet de donner une identité pour le pair, afin d'établir les relations entre les connaissances et leur pair d'origine. La classe Swabbi gère l'annotation des données. Le système se compose d'un ensemble de pairs appelés « SWAP Nodes ». La connaissance d'un pair donné est extraite de plusieurs sources de connaissances, puis intégrée et stockée dans le répertoire local. Une interface utilisateur assure que l'utilisateur peut modifier / parcourir / interroger les connaissances. Les requêtes qui ne peuvent pas être satisfaites par les connaissances disponibles sont envoyées à

l'ensemble du système. Un composant spécialisé s'occupe de la réécriture de ces requêtes et de la sélection des pairs qui détiennent la réponse. Le routage est basé sur des métadonnées sur leurs connaissances et les chiffres de confiance. Les autres pairs répondent aux requêtes de la même manière et finalement renvoient des réponses, qui sont rassemblées et présentées à l'utilisateur. Ce dernier peut alors décider d'ajouter la réponse à sa propre représentation des connaissances. Les réponses consistent en des énoncés qui peuvent également être liés à des fichiers. Pendant qu'un pair communique avec les autres, des informations sur le réseau sont mémorisées et peuvent ensuite être utilisées pour trouver de meilleurs chemins pour les futures requêtes.

Charles *et al.* [25] proposent SwarmLinda, un système basé sur Linda[49] qui résume les concepts de ce dernier en termes de constructions d'aissaimage telles que les odeurs et la stigmergie. Ils étudient l'utilisation des techniques et observations d'essaimage dans le contexte des systèmes de tuples-space, en particulier le système de coordination de Linda. L'objectif est d'examiner le problème de l'évolutivité et d'étudier comment améliorer le scénario en utilisant des techniques adaptées à partir de modèles issus d'organismes collectifs biologiques tels que les colonies de fourmis et les moules de termites.
Le modèle de coordination Linda est basé sur le paradigme de communication de la mémoire associative. Les processus ne peuvent communiquer directement mais plutôt via des mémoires associatives partagées, appelées tuples-spaces. L'accès à ces tuples-spaces se fait via un mécanisme d'alignement. Linda fournit des processus avec des primitives leur permettant de stocker (la primitive out) et d'extraire(les primitives in et rd) des tuples à partir des tuples-spaces.
SwarmLinda est organisé sous forme d'un réseau de nœuds où chaque nœud permet des connexions à partir de plusieurs processus. Les nœuds communiquent et transfèrent des tuples entre eux. La topologie du réseau est similaire à celle d'un réseau pair-à-pair. La distribution de Tuple se rapporte principalement à la primitive out puisque c'est le moyen pour Linda de permettre à un processus de stocker des informations. La distribution de Tuple dans SwarmLinda tente de distribuer des tuples en fonction de leur type. Les tuples similaires restent proches les uns des autres. En termes essaim, cela équivaut à un trie des couvées dans les colonies de termites ou de fourmis. Pour réaliser cette abstraction, le réseau de nœuds SwarmLinda est considéré comme un terrain dans lequel se déplacent les tuples-fourmis . Ces fourmis transportent des tuples (générés par des primitives) et décident à chaque saut dans le réseau si elles doivent abandonner le tuple ou non.
L'extraction de tuple a lieu lorsque les processus produisent des requêtes en exécutant les primitives in et rd. En termes essaim, les requetes sont considérées comme des fourmis à la recherche de nourriture dans un terrain formé par le réseau de nœuds SwarmLinda. Un fourmi-modèle transmet la requête du serveur au serveur de test à chaque étape pour les tuples qui correspondent au modèle. Les fourmis-modèles recherchent les tuples dans le nœud courant et détectent également les odeurs disponibles des voisins du nœud pour prendre une décision. Si le nœud actuel a un tuple correspondant, le tuple est renvoyé au processus demandeur, et si le nœud courant ne détient pas un tel tuple, le fourmi-modèle utilise les informations sur les odeurs pour prendre une décision stochastique quant à la prochaine destination.

Adjiman *et al.* [5] décrivent le système pair-à-pair de gestion des données sémantique SomeWhere qui promeut une vision du web sémantique basée sur des ontologies simples et personnalisées (par exemple, des taxonomies de classes) qui sont distribuées à grande échelle. Dans cette vision du Web sémantique présentée dans [93], aucun utilisateur n'impose aux autres sa propre ontologie, mais des alignements logiques entre ontologies rendent possible la création d'un réseau de personnes dans lequel le marquage sémantique personnalisé des données cohabite bien avec un échange collaboratif de données.

Dans SomeWhere un nouveau pair rejoint le réseau par quelques pairs qu'il connaît (ses connaissances) en déclarant des alignements entre sa propre ontologie et les ontologies de ses connaissances. Les requêtes sont adressées à un pair donné en utilisant son ontologie locale. Les alignements sont des disjonctions, des équivalences ou des énoncés d'inclusion impliquant des classes atomiques de pairs différents. Ils expriment la correspondance sémantique qui peut exister entre les ontologies de pairs différents. Dans un réseau SomeWhere, le schéma n'est pas centralisé, mais distribué par l'union des différentes ontologies de pairs et des alignements. Chaque pair a une connaissance partielle du schéma : il connaît simplement sa propre ontologie locale et les alignements avec ses connaissances. Dans SomeWhere, chaque utilisateur interroge l'architecture pair-à-pair par un pair de son choix et utilise le vocabulaire de ce pair pour exprimer sa requête. Par conséquent, les requêtes sont des combinaisons logiques de classes d'une ontologie de pairs donnée. Les jeux de réponses correspondants sont exprimés en intention en termes de combinaisons de classes extensions qui sont des réécritures de la requête. Les classes extension de plusieurs pairs éloignés peuvent participer aux réécritures, et donc à la réponse d'une requête adressée à un pair donné.

Ives *et al.* [61] présentent le système Piazza qui aborde les défis de la médiation entre les sources de données sur le Web sémantique en procédant à l'alignement à la fois de la structure du domaine et de la structure du document. Une application Piazza se compose de plusieurs nœuds, chacun pouvant avoir un ou deux rôles à la fois : fournir des données source dans son schéma, ou fournir un schéma (et plus tard, une ontologie) qui peut être interrogé. Un nœud très simple peut ne fournir que des données (par exemple à partir d'une base de données relationnelle) ; d'autre part, un nœud peut simplement fournir un schéma auquel les schémas des autres nœuds peuvent être alignés. Le lien sémantique dans Piazza est fourni par des alignements locaux entre petits ensembles (généralement des pairs) de nœuds. Lorsqu'une requête est posée sur le schéma d'un nœud, le système utilisera les données de n'importe quel nœud qui est connecté transitivement par des alignements sémantiques, en chainant des alignements.

Adjiman *et al.* [6] décrivent les systèmes de gestion de données pairs (PDMS) de type SomeRDFS qui favorisent une vision du Web sémantique basée sur des données annotées par des ontologies RDFS. Ils décrivent le déploiement d'un PDMS en utilisant un modèle de données basé sur RDF au-dessus de l'infrastructure SomeWhere : un tel PDMS sera nommé PDMS SomeRDFS. Pour exprimer les ontologies et les alignements, Adjiman *et al.* considèrent le fragment de base du RDFS permettant de spécifier les sous-classes, les sous-propriétés, le typage du domaine et la plage des propriétés. Comme dans un PDMS SomeOWL, la topologie est induite par les alignements entre les ontologies des pairs. Adjiman *et al.* démontrent que les réponses aux requêtes dans les PDMS SomeRDFS peuvent être obtenues en utilisant une stratégie de réécriture et d'évaluation et que le problème de réécriture correspondant peut être réduit au même problème de recherche de conséquences dans les théories propositionnelles distribuées. Ils considèrent les principaux constructeurs de RDFS basés sur des relations unaires appelées classes et des relations binaires appelées propriétés. Ces constructeurs sont : l'inclusion de classe, l'inclusion de propriété et le typage domaine / plage d'une propriété. Ce langage est ainsi désigné core-RDFS. Les alignements sont la notion clé pour établir des connexions sémantiques entre des ontologies afin de partager des données. Ils sont ici définis comme des instructions core-RDFS impliquant des relations de deux vocabulaires de pairs différents.
Dans un PDMS, le schéma et les données sont distribués à travers respectivement l'union des ontologies et alignements des pairs, et l'union des descriptions de stockage des pairs. Chaque pair a une connaissance partielle du PDMS. Dans un PDMS SomeRDFS, un pair connaît simplement son ontologie, ses relations avec d'autres pairs et les relations partagées avec eux, et

ses propres données. Le schéma d'un PDMS SomeRDFS S, noté ici *schema(S)*, est l'union des ontologies et des ensembles d'alignements de tous les pairs. Les données d'un SomeRDFS PDMS S, notées *data(S)*, est l'union des descriptions de données des pairs. Une base de connaissances SomeRDFS est l'union de son schéma et de ses données. L'algorithme de réécriture de requêtes de SomeRDFS, nommé $DeCA^{rdfs}$, est conçu au-dessus de l'algorithme DeCA de SomeWhere. Sur chaque pair SomeRDFS, $DeCA^{rdfs}$ agit comme une interface entre l'utilisateur et DeCA. La stratégie de $DeCA^{rdfs}$ est de réécrire les atomes de la requête utilisateur indépendamment avec DeCA, puis de combiner leurs réécritures afin de générer quelques réécritures conjonctives de la requête utilisateur. $DeCA^{rdfs}$ garantit que toutes les réécritures conjonctives maximales de la requête utilisateur sont générées. L'approche pour répondre aux requêtes repose ainsi sur deux étapes : la réécriture de requête résulte en une union de requêtes conjonctives sur des pairs éventuellement différents ; les réponses sont alors obtenues en évaluant chacune de ces requêtes conjonctives sur les pairs appropriés.

Skaf *et al.* [111] présentent SWOOKI un wiki sémantique pair-à-pair. Un wiki sémantique pair-à-pair combine les avantages des wikis sémantiques et des Wikis pair-à-pair. L'architecture d'un wiki sémantique pair-à-pair est composé d'un ensemble de serveurs autonomes interconnectés qui peuvent joindre et quitter le système. Chaque pair accueille une réplique des données partagées. La réplication totale améliore la disponibilité et la performance des données. Le système est une combinaison de l'approche des wikis d'ontologies tels que Semantic MediaWiki et d'un wiki pair-à-pair basé sur la réplication totale et la cohérence CCI (convergence, causality preservation and intention preservation) tels que Wooki [131]. L'originalité de Wooki est son algorithme d'intégration WOOTO. Cependant, WOOTO ne peut pas être appliqué directement à un wiki sémantique pair-à-pair. Il est conçu pour synchroniser des structures linéaires, il ne peut pas synchroniser un mélange de texte et de graphes RDF. Skaf *et al.* proposent d'étendre WOOTO pour gérer l'écriture collaborative sur le modèle de données RDF répliqué.
Comme dans tout système wiki, l'élément de base est une page wiki et chaque page wiki est affectée d'un identificateur unique PageID, qui est le nom de la page. Dans les wikis sémantiques P2P, chaque pair possède un référentiel RDF local qui contient un ensemble d'instructions RDF extraites de ses pages wikis.
Le modèle de données est une extension du modèle de données WOOKI [131] pour prendre en compte les données sémantiques. Chaque pair wiki sémantique reçoit un identificateur unique global nommé NodeID.
Les wikis sémantiques pair-à-pair s'assurent que lorsque le même ensemble d'opérations est exécuté sur tous les sites : toutes les répliques de pages wikis sont identiques, toutes les répliques de référentiels RDF sont identiques.
Un serveur SWOOKI est composé des éléments suivants :

- L'interface utilisateur : ce composant est fondamentalement un éditeur de Wiki régulier. Il est ainsi capable de générer des patchs d'opérations SWOOKI.
- Le gestionnaire SWOOKI : il implémente l'algorithme SWOOKI.
- Le moteur sésame : c'est un store RDF. Il est contrôlé par le gestionnaire SWOOKI pour le stockage et la récupération des triples RDF. Il est responsable de l'analyse, la recherche, le stockage et l'extraction des données sémantiques.
- Le gestionnaire de diffusion : il est chargé de maintenir l'appartenance à l'architecture non structuré et de mettre en œuvre une diffusion fiable.

Cerqueus *et al.* [23] présentent le protocole CORDIS qui a pour objectif de réduire l'hétérogénéité sémantique de systèmes pair-à-pair non-structurés liée aux disparités entre pairs. L'hétérogénéité sémantique est une notion complexe qui provient du fait que tous les participants d'un système pair-à-pair ne s'accordent pas sur l'utilisation d'une ontologie unique. L'idée

du protocole CORDIS est de disséminer des correspondances dans le système afin de partager les correspondances connues de certains mais ignorées par d'autres. Plus les pairs connaissent de correspondances, plus ils sont en mesure de comprendre et traduire les requêtes émises par les autres pairs du système. Du point de vue de l'hétérogénéité, CORDIS permet de réduire les disparités entre les pairs. Les expérimentations montrent qu'en allouant peu d'espace pour le stockage des correspondances, et en augmentant raisonnablement le trafic réseau, l'hétérogénéité sémantique liée à la disparité entre pairs peut être réduite de manière significative.

Atencia *et al.* [10] présentent un mécanisme de confiance qui permet aux pair de sélectionner d'autres pairs de l'architecture qui sont mieux adaptés pour répondre à leurs requêtes. La confiance d'un pair par rapport à un autre dépend d'une requête spécifique et représente la probabilité que ce dernier donne une réponse satisfaisante. Afin de calculer la confiance, Atencia *et al.* exploitent à la fois les alignements et l'expérience directe des pairs, et procède à une inférence bayésienne. Ils proposent un modèle probabiliste pour gérer la confiance dans un environnement pair-à-pair sémantique. La confiance d'un pair P envers un autre pair P' par rapport à une classe C (appartenant à l'ontologie de P) est définie comme la probabilité qu'une instance retournée par P' comme une réponse à la requête demandant des instances de C, soit satisfaisante pour P. Pour calculer la confiance, Atencia *et al.* exploitent les informations fournies par les ontologies et les alignements des pairs, ainsi que les informations qui proviennent de l'expérience directe. Les valeurs de confiance sont affinées au fur et à mesure que de nouvelles requêtes sont envoyées et des réponses reçues. Les traductions de requêtes sont déterminées par des correspondances des alignements de l'architecture. Un pair ne peut prévoir si la réponse qu'un autre pair fournit à l'une de ses requêtes contient des éléments satisfaisants ou non, mais cette incertitude peut être estimée à l'aide d'un mécanisme de confiance. Si le pair P reçoit un ensemble B comme réponse à la requête, la proportion d'instances satisfaisantes est donnée par une probabilité conditionnelle. L'approche de la confiance vise ainsi à trouver des approximations de ces probabilités conditionnelles par inférence bayésienne. S'il n'y a pas d'expérience directe, les alignements sont pris pour construire des opinions antérieures. Les réponses sont ensuite utilisées pour réviser ces opinions. Comme ils peuvent être d'une taille qui ne peut pas être traitée manuellement, il est ici proposé d'effectuer un échantillonnage avec remplacement afin d'estimer le nombre d'instances satisfaisantes.

3.2 Cas des systèmes pair-à-pair hybrides

Les systèmes pair-à-pair hybrides combinent des spécificités des systèmes centralisés et des systèmes pair-à-pair purs. Ce type d'architecture introduit une hiérarchie entre les pairs. Ces derniers ayant des capacités différentes (puissance de calcul, mémoire de stockage, etc.), certains joueront le rôle de "super-pair". Précisément, les super-pairs agissent comme des serveurs centraux au sein d'un sous-ensemble fini de pairs. Ils fonctionnent en mode pair-à-pair pure entre eux pour maintenir le caractère distribué de l'architecture, alors qu'au sein d'un sous-ensemble, le système adopte le mode pair-à-pair centralisé où le super-pair joue le rôle de serveur central. Cette solution permet de résoudre le problème d'extensibilité de l'approche purement distribuée tout en gardant l'efficacité de la solution centralisée. Ce type d'architecture est bien répandu dans la littérature.

Nejdl *et al.* [86](Gnutella/JXTA) analysent les architectures pair-à-pair basées sur des schémas et décrit l'utilisation de topologies de super-pair pour ces réseaux. Ces types d'architectures s'appuient sur des pairs qui utilisent des schémas explicites décrivant leur contenu et où les métadonnées des pairs peuvent être basées sur des schémas hétérogènes. Les super-pairs ont pour

rôles de gérer les indices de routage et déterminer quelle requête est transmise à quel pair ou à quel super-pair. Chaque pair est connecté à un super-pair. Les super-pairs sont connectés entre eux et constituent ainsi la base de l'architecture super-pair.
L'introduction de super-pairs en combinaison avec des indices de routage réduit considérablement la charge de travail des pairs en distribuant les requêtes uniquement à un sous-ensemble approprié de pairs. Les premiers types d'indices nécessaires sont les indices de routage super-pair/pair (SP/P-RI). Avec ces indices chaque super-pair stocke des informations sur l'utilisation des métadonnées sur chaque pair. Cela inclut des informations de schéma telles que des schémas ou des attributs utilisés, ainsi que des indices éventuellement conventionnels sur les valeurs d'attribut. Pour chaque super pair, les éléments utilisés dans une requête sont comparés aux SP/P-RI afin de déterminer les pairs locaux pouvant répondre à la requête. Les seconds types d'indices sont les indices de routage super-pair/super-pair permettant de router vers les super-pairs. Ces indices SP/SP sont essentiellement des extraits et des résumés (éventuellement aussi des approximations) de tous les indices SP/P locaux. Ils contiennent le même type d'information que les indices SP/P, mais se rapportent aux voisins directs d'un super pair. Les requêtes sont transmises aux super-pairs voisins sur la base des indices SP/SP et envoyées aux pairs connectés en fonction des indices SP/P.
Certaines requêtes pourraient être résolues en combinant les résultats de nombreux pairs utilisant des schémas hétérogènes. Pour transformer des schémas entre différents pairs (c'est-à-dire la médiation), intégrant différents schémas de requêtes entre eux, Nejdl *et al.* considèrent des translations entre différents schémas locaux. Ils utilisent les correspondances basées sur le MBIS (mediator-based information systems) comme des règles pour décrire de telles translations et utilisent les noms de propriétés comme arguments dans les littéraux de requête pour une notation concise. Chaque super pair stocke les relations entre les correspondances et les pairs dans ses indices.

Shen *et al.* [109] proposent un framework général et extensible pour la recherche de contenu sémantique dans une architecture pair-à-pair hiérarchique (avec des super-pairs). Le framework introduit le concept de "Hierarchical Summary Indexing Structure" (Summary and Indexing) étroitement lié à l'architecture pair-à-pair hiérarchique utilisée.
Il repose sur trois niveaux de synthèse. Le niveau le plus bas, nommé "unit level", une unité d'information, comme un document ou une image, est synthétisée. Dans le deuxième niveau, "peer level", toutes les informations détenues par un pair sont synthétisées. Enfin, dans le troisième niveau,"super level", toutes les informations contenues dans un groupe de pairs sont synthétisées. Pour chaque niveau, le processus de synthèse se compose de deux étapes exécutées par des techniques de Vector Space Model (VSM) et Latent Semantic Indexing (LSI), respectivement. LSI a été proposée pour surmonter les problèmes de synonymie, de polysémie et de bruit dans la recherche d'information. LSI découvre la corrélation sémantique sous-jacente entre les documents en construisant un espace de concept. Grâce à ce dernier la recherche est basée sur des concepts plutôt que sur des termes individuels.
Chaque super pair conserve deux types de synthèses : les synthèses "super level" de son groupe et ses groupes voisins, et les synthèses "peer level" de son groupe. En examinant les synthèses "super level", un super-pair peut déterminer quel groupe de pairs est pertinent. De même, en examinant les synthèses au niveau des pairs, un super pair peut déterminer lequel de ses pairs a les réponses.
Puisque le nombre de synthèses peut être important, pour améliorer encore l'efficacité du système, trois index sont maintenus sur ces informations. Ces trois index pour les synthèses des "unit level", "peer level" et "super level" sont nommés respectivement index local, index de groupe et index global.

Lorsqu'un pair émet une requête Q, Q est d'abord transmis à son super-pair, suivi de la recherche d'indexation hiérarchique dans l'ordre suivant : index global, index de groupe et index de pair, qui est l'ordre inverse de la construction des synthèses. Enfin, chaque pair renvoie les K documents les plus similaires au client qui émet la requête. À chaque niveau d'index, la valeur K peut être définie en fonction des besoins de l'utilisateur.

3.3 Synthèse

Le tableau suivant présente une synthèse des différentes solutions présentées dans cette partie. Chaque ligne représente une caractéristique nous permettant de comparer ces différents modèles :

- **Mécanismes d'alignements** : la solution implémente un mécanisme d'alignement d'ontologies entre les pairs dans le but de gérer l'hétérogénéité sémantique des données détenues par les pairs.
- **Index** : la solution implémente un mécanisme d'indexation qui permet d'accélérer le processus de recherche de donnée dans l'architecture.
- **Control d'accès** : mécanisme de sécurité permettant aux pairs de ne partager que les données souhaitées. La nature privée de certaines données est conservée.
- **Mécanisme de confiance** : utilisation de valeur de confiance pour estimer la fiabilité d'un pair lors du processus de traitement de requête. Cette valeur permet aussi une sélection plus efficace de pair pour le traitement des requêtes à venir.
- **Type de donnée** : type des données échangées.
- **Architecture P2P** : architecture pair-à-pair sous-jacent de la solution (P : Pure et H : Hybride).

	SWAP [2003]	Nejdl et al. [2004]	H. Shen et al. [2004]	Somewhere [2005]	Piazza [2006]	SomeRDFS [2006]	Swooki [2008]	Cordis [2011]	Attentia et al. [2011]
Mécanismes d'alignements	√	√		√	√	√		√	√
Index		√	√						
Control d'accès	√								
Mécanisme de confiance	√							√	√
Type de donnée	RDF	RDF	Fichier	RDF	XML	RDF	Texte	RDF	RDF
Architecture P2P	P	H	H	P	P	P	P	P	P

TABLE 4: Tableau de synthèse Ontologie et Web Sémantique en pair-à-pair.
La prise en charge d'une caractéristique est indiquée par le signe √ , la non prise en charge par une case vide.

4 Cache distribué en pair-à-pair

Dans cette partie nous nous intéressons aux architectures pair-à-pair permettant de gérer des caches de données distribués. Nous regardons les solutions relatives à la question **Q6** du chapitre I (Comment le cache de données peut-il améliorer le processus d'accès et d'interrogation des sources de données ?). La mise en cache pair-à-pair est semblable en principe à la mise en cache de contenu longtemps utilisée par les fournisseurs de services Internet (FAI) pour accélérer l'accès aux contenus Web (HTTP). Elle permet de stocker temporairement du contenu populaire diffusé dans le réseau d'un FAI. Si le contenu demandé par un utilisateur est disponible à partir d'un cache, le cache satisfait la demande à partir de son stockage temporaire, éliminant le transfert de données via des liaisons de transport coûteuses et réduisant la congestion du réseau.

Les systèmes distribués de cache pair-à-pair adoptent des stratégies propres aux architectures pair-à-pair (les nœuds ont les mêmes rôles : client et serveur) d'une part et aux systèmes coopératifs de cache décentralisé (chaque nœud maintient un cache local accessible aux autres nœuds) d'autre part. Plus précisément, le système repose sur une architecture pair-à-pair décentralisée dans laquelle l'ensemble des caches locaux de chaque nœud forment un cache commun accessible à tous les pairs du système. Une mise en cache partielle/totale des données transitant dans le système est effectuée par les nœuds dans le but de traiter une grande partie des requêtes à partir du cache local ou des caches des pairs voisins. Plusieurs travaux ont été mené dans cette optique.

4.1 Travaux connexes

Lee *et al.* [75] proposent un mécanisme de mise en cache sémantique qui permet de stocker des données en tant que collection de blocs éventuellement liés, dont chacun est le résultat d'une requête préalablement évaluée. Ils traitent des contraintes d'accès à une base de données relationnelle dans un environnement mobile et résout le problème en réutilisant des résultats partiels d'accès antérieures aux données, exploitant la sémantique des données mises en cache.
Le système est basé sur un paradigme d'expédition de requêtes. Le serveur et le client possèdent leurs propres systèmes de bases de données relationnelles, leur fournissant ainsi un stockage de données local. Chaque système de base de données est doté de logiciels permettant de gérer la mémoire cache.
Un client est doté d'une interface utilisateur pour accepter des requêtes, un gestionnaire de requêtes pour évaluer des requêtes et un gestionnaire de cache pour gérer le cache de stockage, c'est-à-dire la base de données client. Lorsqu'un canal de communication est disponible, le gestionnaire de requêtes coopère avec le processeur de requêtes du serveur pour évaluer les requêtes. Le gestionnaire analyse d'abord la requête en déterminant si elle peut être traitée localement. Lorsque tous les éléments de données requis sont disponibles dans le cache local, la requête est transformée en une requête locale. Si seule une partie de la requête peut être évaluée localement, elle sera transformée en une requête de sonde et une requête supplémentaire. Une requête de sonde est une requête locale pour extraire des éléments de données du cache client tandis qu'une requête supplémentaire est soumise au serveur pour évaluation. Lorsqu'il reçoit le résultat d'une requête supplémentaire du serveur, le gestionnaire de cache du client l'enregistre dans le cache de stockage local. Le résultat de la requête finale est construit en intégrant les résultats de la requête de sonde et de la requête supplémentaire. Lors d'une déconnexion, le gestionnaire de requêtes traite la requête uniquement en utilisant les résultats mis en cache localement.

Clarke *et al.* [26] développent Freenet, un système de stockage et d'extraction d'informations distribuées conçu pour répondre à des contraintes de confidentialité et de disponibilité. Le sys-

tème fonctionne sur de nombreux ordinateurs individuels, comme un système de fichiers distribué indépendant de l'emplacement , ce qui permet d'insérer des fichiers stockés et de les interroger de manière anonyme. Le système est conçu comme un réseau pair à pair de nœuds qui s'interrogent mutuellement pour stocker et extraire des fichiers de données nommés par des clés indépendantes de l'emplacement. Chaque nœud maintient son propre stockage local de données qu'il met à la disposition du réseau pour les scénarios de lecture/écriture ainsi qu'une table de routage dynamique contenant les adresses des autres nœuds et les clés qu'ils sont censés contenir.
Pour extraire des données, un utilisateur hache une courte chaîne descriptive pour obtenir une clé de fichier. L'utilisateur envoie ensuite un message de requête à son propre nœud en spécifiant cette clé et une valeur "hops-to-live". Lorsqu'un nœud reçoit la requête, il vérifie d'abord son propre stockage pour les données et les renvoie s'ils sont trouvés avec une note indiquant qu'il s'agissait de la source des données. Si les données ne sont pas trouvés, le noeud recherche la clé la plus proche dans sa table de routage et transmet la requête au nœud correspondant. Si cette requête aboutit et est renvoyée avec les données, le nœud transmet les données au nœud origine de la requête , met en cache le fichier dans son propre stockage de données et crée une nouvelle entrée dans sa table de routage associant la source de données réelle à la clé demandée. Une requête ultérieure pour la même clé sera immédiatement satisfaite à partir du cache local, une demande de clé similaire (déterminée par la distance lexicographique) sera transmise à la source de données précédente.

Iyer *et al.* [62] présentent Squirrel un cache web pair-à-pair décentralisé. Squirrel permet de faciliter le partage mutuel des objets de données Web entre les pairs clients et permet aux pairs d'exporter leurs caches locaux vers d'autres pairs dans le réseau, créant ainsi un grand cache Web virtuel partagé. Le cache Web coopératif est construit au-dessus du substrat de routage pair-à-pair Pastry [99]. Chaque pair effectue à la fois la navigation Web et la mise en cache Web, sans besoin de matériel coûteux et dédié comme pour la mise en cache web centralisée. Squirrel fait face à un nouveau défi par lequel les pairs dans un cache décentralisé encourent des frais supplémentaires d'avoir à traiter les requêtes des autres pairs, et cette charge supplémentaire doit être maintenue à un niveau bas. Chaque nœud participant exécute une instance de Squirrel avec la même politique d'expiration et configure le navigateur Web sur ce nœud pour utiliser cette instance Squirrel comme cache proxy. Le navigateur et l'instance de Squirrel partage un même cache géré par Squirrel. Ceci est réalisé en désactivant le cache du navigateur. Le navigateur Web lance sa requête au proxy Squirrel qui s'exécute sur le même nœud. Si le proxy reconnaît l'objet comme « non-cachable », la requête est renvoyée directement au serveur Web d'origine, sinon, il vérifie le cache local. Si une nouvelle copie de l'objet n'est pas trouvée dans le cache, Squirrel tente de localiser une copie sur un autre nœud.

Freedman *et al.* [46] présentent CoralCDN, une architecture réseau décentralisé et auto-organisé de distribution de contenu web. CoralCDN est un réseau de distribution de contenu pair-à-pair qui permet à un utilisateur d'exploiter un site Web offrant des performances élevées et répondant à une demande énorme, le tout pour le prix d'une connexion Internet haut débit bon marché. Les sites bénévoles qui fonctionnent sous CoralCDN répliquent automatiquement leurs contenus comme un effet secondaire des accès utilisateurs. Pour utiliser CoralCDN, un éditeur de contenu (ou un utilisateur affichant un lien vers un portail à fort trafic) ajoute simplement ".nyud.net : 8090" au nom d'hôte dans l'URL. Grâce à la redirection DNS, les clients dotés de navigateurs web non modifiés sont redirigés de manière transparente vers des caches web Coral. Ces caches coopèrent pour transférer des données à partir des pairs voisins, minimisant dans la mesure du possible, à la fois la charge sur le serveur Web d'origine et la latence de bout en bout subie par les navigateurs.

Ranger *et al.* [94] présentent le projet Mind-Alliance qui a pour but de développer une plateforme de partage d'information avec une architecture entièrement décentralisée qui prend en charge les activités d'échange d'information, axées sur la communauté. La plateforme utilise une infrastructure pair-à-pair pour créer des « réseaux adressables de connaissances » basés sur RDF. Dans l'environnement cible de cette plateforme, chaque pair dans un groupe connecté partage une partie de son contenu avec le groupe. Les pairs sont supposés avoir les mêmes rôles et caractéristiques. Le contenu peut être dupliqué entre pairs, à la discrétion des utilisateurs. Chaque pair peut accéder efficacement à son propre contenu en tant que graphe Web sémantique. Un algorithme a été développé pour supporter : de potentielles très grandes architectures pair-à-pair, des bases de connaissances RDF distribuées, des recherches structurées (de bases de connaissances RDF réparties sur des architectures pair-à-pair à grande échelle). Chaque pair est une source potentielle d'information / connaissance. La connaissance doit être extraite (ou combinée à partir) de l'information répartie sur l'architecture pair-à-pair. Les requêtes complexes sont traitées de manière fiable (s'il y a une réponse, il sera trouvé avec une probabilité presque certainement) et efficace (le temps de traitement est acceptable et n'augmente que légèrement avec le nombre de pairs).

Han *et al.* [53] présentent SocialCDN, un système de distribution de contenu social efficace basé sur un mécanisme de sélection de cache social distribué qui n'exige pas une connaissance globale du graphe social sous-jacent. Un réseau social distribués en ligne (Distributed online social networks : DOSN) est une infrastructure pair-à-pair qui prend en charge les fonctionnalités des réseaux sociaux en ligne (online social networks : OSN) de manière distribuée. Les DOSNs permettent aux utilisateurs d'héberger et d'organiser leurs profils personnels et leurs connexions sociales tout en conservant un contrôle total sur leurs propres données. Cependant, l'un des principaux problèmes des DOSNs est de pouvoir diffuser efficacement des mises à jour sociales à travers les pairs. SocialCDN propose des caches sociaux pour réduire le nombre total de connexions pair-à-pair nécessaires à la diffusion des mises à jour sociales dans un DOSN. Les nœuds envoient leurs mises à jour sociales et récupèrent celles de leurs amis au niveau des caches sociaux auxquels ils sont connectés. Si le nombre de caches sélectionnés est nettement inférieur au nombre d'amis, l'utilisation de caches sociaux réduira significativement le nombre total de connexions par rapport à une approche pair-à-pair push/pull d'où l'objectif de minimiser le nombre de caches sociaux en utilisant des algorithmes entièrement distribués.

Zhang *et al.* [138] présentent Maygh, un système qui construit un réseau de distribution de contenu à partir des navigateurs Web du client, sans avoir besoin de plug-ins supplémentaires ou de logiciels côté client. Avec Maygh, les utilisateurs visitant le site Web de l'opérateur aident automatiquement à diffuser du contenu statique à d'autres pendant la durée de leur visite ; Maygh utilise des techniques de communication navigateur-à-navigateur disponibles pour permettre l'échange direct de messages avec d'autres navigateurs. Maygh utilise des coordinateurs centralisés pour suivre le contenu statique stocké dans les navigateurs, servant de répertoire pour trouver du contenu.
Le code client Mayghi communique avec le coordinateur via RTMFP ou WebRTC. Une fois la page Web chargée, le client lance une connexion avec le coordinateur. Après la prise de contact RTMFP / WebRTC, le client informe le coordinateur du contenu qu'il stocke localement en envoyant un message de mise à jour Maygh. Ce message contient une liste de hash de contenu. Le client et le coordinateur maintiennent alors la connexion ouverte. Lorsqu'un client souhaite extraire du contenu, il envoie un message de recherche au coordinateur contenant le hachage de contenu de la donnée recherchée et les identifiants de tous les autres clients auxquels le client est

actuellement connecté. Le coordonnateur répond par un message lookup-response contenant un identifiant d'un pair en ligne qui stocke ce contenu. Dans le cas ou d'autres clients stockent le contenu demandé, le coordonnateur tente de sélectionner d'autres pairs avec lesquels le client est déjà connectés ou qui sont proches de lui (par exemple, en utilisant une base de données géo-IP).

Frey *et al.* [47] présentent Behave, une approche dédiée aux applications orientées cache pair-à-pair telles que les réseaux de distribution de contenu (Content Delivery Networks : CDN). Behave exploite des protocoles épidémiques pour créer des communautés superposées de pairs ayant des intérêts similaires. Les pairs de la même communauté fédèrent leurs index de cache dans un cache comportemental. La localité comportementale de Behave étend les spécificités des CDNs en exploitant les similitudes entre les comportements de navigation des utilisateurs. Contrairement aux solutions DHT (Distributed Hash Tables), qui visent à maximiser la quantité d'objets Web accessibles par le substrat pair-à-pair, Behave se concentre sur la maximisation du nombre d'objets accessibles sans délai de recherche. Le cache de comportement collaboratif est construit en se basant sur le filtrage collaboratif basé sur l'utilisateur reposant sur l'hypothèse suivante : les utilisateurs ayant manifesté des goûts semblables dans le passé présenteront probablement des goûts similaires à l'avenir. Plus précisément, si deux utilisateurs ont des éléments communs dans leurs caches Web, alors ils présenteront probablement des points communs dans les sites Web qu'ils visiteront à l'avenir. Chaque nœud identifie l'ensemble des nœuds qui lui sont le plus semblables en termes d'historique de navigation : ses voisins. L'agrégation des index de caches des voisins d'un nœud constitue un index de cache comportemental.

Folz *et al.* [45] présentent CyCLaDEs, une approche qui permet de construire un cache comportemental décentralisé hébergé par des clients LDF. CyclaDEs est une architecture fondée sur la similarité des fragments LDF. Pour chaque client LDF, CyCLaDEs construit un voisinage de clients LDF hébergeant des fragments liés dans leur cache. Lors du traitement des requêtes, le cache de voisinage est vérifié avant d'interroger le serveur LDF. Pour construire un cache de comportement décentralisé, chaque client LDF doit avoir un nombre limité de voisins avec un accès de latence zéro. Pendant le traitement des requêtes, pour chaque sous-requête de motifs de triplet, CyCLaDEs vérifie si le motif de triplet peut être satisfait à partir du cache local, sinon dans le cache des voisins. Une requête est envoyée au serveur LDF uniquement si le motif de triplet ne peut être satisfait ni avec le cache local ni avec le cache des nœuds voisins. Le système adopte l'hypothèse que des clients ayant traité les mêmes requêtes dans le passé traiteront probablement des requêtes similaires à l'avenir. Une métrique de similarité est utilisée en comptant le nombre de prédicats dans les motifs de triplets.

4.2 Synthèse

Le tableau suivant présente une synthèse des différentes solutions présentées dans cette partie. Chaque ligne représente une caractéristique nous permettant de comparer ces différents modèles :

- **Caching distribué** : la solution implémente un mécanisme de cache distribué. Chaque pair détient dans son cache une partie de données pertinente pouvant être utile aux autres pairs. L'architecture pair-à-pair sous-jacente assure l'accessibilité des données.
- **Cache client** : le client détient un cache local lui permettant de traiter une partie de ces requêtes de manière autonome sans avoir à accéder au cache du serveur ou des autres clients. Par contre, ces derniers sont tout de même disponibles et accessibles pour les cas où le cache client serait insuffisant pour le traitement de la requête.
- **Cache de navigateur** : le client utilise le cache du navigateur. Le partage de données est

dans ce cas facilité par la capacité des navigateurs à communiquer directement. Le client peut alors profiter des données disponibles sur les caches de ses voisins pour un traitement plus efficace de ses requêtes.

- **Architecture P2P** : Architecture pair-à-pair sous-jacente (P : pure et H : Hybride).
- **Type de donnée** : types des données échangées.
- **Technologie de base** : DHT ou Peer Sampling

	Ken C. Lee [1999]	Squirel [2002]	CoralCDN [2004]	Mind-Alliance [2005]	SocialCDN [2012]	Maygh [2013]	Behave [2014]	Cyclades [2016]
Technologie de base		DHT	DHT	DHT		DHT	RPS	RPS
Caching distribué	√	√	√	√	√	√	√	√
Cache client	√	√		√		√	√	√
Cache de navigateur						√	√	√
Architecture P2P	C	P	H	P	H	P	P	P
Type de donnée	Tuples BDR	Objet Web	Objet Web	RDF		Objet Web	Objet Web	RDF

TABLE 5: Tableau de synthèse des solutions traitant du Cache distribué en pair-à-pair. La prise en charge d'une caractéristique est indiquée par le signe √ , la non prise en charge par une case vide.

5 Géolocalisation et pair-à-pair

Dans cette partie dédiée à la question **Q4** du chapitre I (Comment la géolocalisation peut-être prise en compte pour les échanges de données sémantiques ?), nous regardons les systèmes pair-à-pair qui prennent en considération la localisation des pairs pour gérer les vues, les échanges, etc. La recherche dans les environnement virtuels a démontré l'importance des informations relatives à la position des partenaires pour soutenir des processus collaboratifs tels que la répartition de tâche au sein du groupe [88]. A l'aide des coordonnées géographiques, les nœuds d'une architecture pair-à-pair peuvent former ou améliorer la qualité de leurs vues. Les relations entre les pairs constituant l'architecture ne sont plus uniquement basées sur des liens sociaux ou sur des intérêts similaires, mais elles reposent en plus sur des situations de proximité géographique. Chaque nœud dispose de ses propres coordonnées et, par un mécanisme sous-jacent (message diffusé, serveur central, requête, etc.), accède à celles des autres pairs du système. Dans les systèmes pair-à-pair utilisant des protocoles de "gossip" par exemple, ces coordonnées seront utilisées pour le calcul de distance géographique dans le but de déterminer lesquels des pairs sont les plus proches (en terme de distance métrique) du nœud exécutant le protocole pour ensuite les intégrer à sa vue locale ou pour initier des échanges. Plus précisément, la distance euclidienne peut être considérée dans ce cas à deux niveaux : soit lors de l'échantillonnage pour l'obtention de pairs constituant la vue ; soit lors de la sélection de pairs au niveau de la vue pour déclencher un processus d'échange de profils dans le but d'améliorer la qualité de la vue ou pour simplement diffuser un message dans le système. Les informations géographiques s'avèrent aussi pertinentes pour l'optimisation des systèmes de routages (utilisant parfois des protocoles de gossip) dans le cadre des réseaux de capteurs où les nœuds du réseau sont en principes limités en termes de ressource énergétiques.

5.1 Travaux connexes

Dimakis *et al.* [33] proposent et analysent un schéma alternatif de "gossip" qui exploite l'information géographique. En utilisant une méthode de ré-échantillonnage simple, les auteurs démontrent des gains substantiels par rapport aux protocoles de "gossip" précédemment proposés. Ils tirent parti du fait que les nœuds de capteurs connaissent généralement leurs emplacements et peuvent donc utiliser ces connaissances pour effectuer le routage géographique. Dimakis *et al.* proposent un algorithme qui, comme un protocole de gossip standard, est entièrement aléatoire, distribué et robuste, mais nécessite beaucoup moins de communication en exploitant l'information géographique. L'idée est que, au lieu d'échanger des informations avec des voisins à saut unique (one-hop), le routage géographique peut être utilisé pour échanger avec des nœuds aléatoires qui sont éloignés dans l'architecture. Le coût supplémentaire du routage multi-sauts est compensé par la diffusion rapide de l'information.

Picone *et al.* [92] présentent une approche totalement décentralisée pour la localisation en plein air, utilisant des appareils mobiles connectés dans un réseau pair-à-pair, sans dégradation de performance par rapport à l'approche centralisée. GeoKad définit et implémente une architecture pair-à-pair où chaque nœud (programmes logiciels exécutés sur un périphérique mobile) est conscient des nœuds qui sont « géographiquement » à proximité. Dans un système de ce type, les requêtes relatives à la position d'un pair spécifique ne sont pas traitées en interrogeant un nœud central, mais en les acheminant à travers l'architecture pair-à-pair dans le but d'éviter l'utilisation d'un serveur central et en même temps augmenter la disponibilité d'informations de mises à jour sur le réseau. Chaque pair GeoKad connaît sa position globale (GP) récupérée avec le système GPS ou avec d'autres technologies de localisation, et connaît un ensemble de voisins réels organisés en une structure basée sur la distance que ces nœuds ont par rapport à la

position du nœud initial. Chaque nœud de la liste des voisins se décrit par trois caractéristiques : la position GPS, l'adresse IP, le port UDP. Lorsqu'un nœud joint l'architecture, il reçoit d'un nœud bootstrap (le super-pair) une petite liste aléatoire de pairs disponibles dans le système qui sont actifs et proches (liste des voisins). Lorsqu'il reçoit le premier groupe de voisins, il entre en contact avec eux pour recevoir leurs positions mises à jour. A partir de la liste initiale mise à jour, le nœud démarre un processus itératif où les nœuds communiquent entre eux, en interrogeant des pairs connus et actifs, proches de leurs positions. Chaque nœud contacté effectue la même opération sur les nœuds qu'il connaît et qui sont proches. Cette procédure de "recherche" permet aux nœuds de récupérer les informations concernant les "voisins géographiques" à travers l'architecture pair-à-pair sans avoir à interagir avec une entité centralisée.

Fiorese *et al.* [44] proposent une approche traitant du problème de sélection du meilleur pair dans les "service overlay networks" (SON) pair-à-pair. Le service de sélection du meilleur pair (BPSS) proposé est une composante d'une architecture de gestion des services (OMAN) précédemment proposée par les mêmes auteurs [43]. Ce service a pour objectif la recherche et la sélection du meilleur pair avec lequel un pair demandeur de service devrait interagir dans le contexte d'un SON pair-à-pair. Le choix d'un pair approprié améliore la performance du service prévue, qui est liée non seulement aux caractéristiques mesurables du réseau, telles que la bande passante, le délai, la perte, etc., mais aussi à l'emplacement des pairs. Afin de tester et évaluer le comportement du BPSS, une métrique de distance spécifique a été utilisée. A. Fiorese *et al.* adoptent la stratégie de [69] qui consiste en un modèle prédictif de l'espace des délais d'Internet qui prend en compte l'emplacement géographique des nœuds et le délai entre eux. Ils effectuent un alignement des nœuds end-to-end mesurés en un modèle spatial euclidien à 5 dimensions d'Internet en combinant les informations globales de positionnement du réseau avec les données mesurées relatives au round trip time (RTT) et à l'instabilité du lien. En utilisant les coordonnées de chaque pair dans ce modèle à 5 dimensions, il est alors possible de calculer la distance euclidienne entre pairs, ce qui permet de tenir compte non seulement des conditions du réseau, mais aussi de l'emplacement des pairs.

Voulgaris *et al.* [130] proposent l'algorithme de "gossip" VICINITY conçu pour explorer les compromis entre la prise de décision déterministe et probabiliste pour la structuration des architectures pair-à-pair. Le protocole s'appuie sur une fonction de sélection *SELECT (p, D, k)*, qui, étant donné le descripteur du nœud p et un ensemble D de descripteurs de nœud, renvoie l'ensemble des k descripteurs qui correspondent le mieux aux liens sortants de p dans la structure cible. Pour les besoins des tests du protocole, les auteurs choisissent d'utiliser une heuristique de proximité euclidienne. Ils considèrent un espace bidimensionnel, et assignent à chaque nœud ses coordonnées (x, y) de tels sorte qu'ils soient virtuellement alignés dans une organisation régulière de grille carrée. Les coordonnées d'un nœud constituent son profil. L'objectif de chaque nœud est d'établir des liens vers ses quatre voisins les plus proches, au nord, au sud, à l'est et à l'ouest (entourant les bords de la grille). Le nœud initie ainsi sa vue avec ses quatre voisins.

Soares *et al.* [112] proposent GeoSpray, un protocole de routage basé sur l'emplacement géographique conçu pour les VDTNs (Vehicle Delay-Tolerant Network). Le VDTN est une architecture réseau basée sur le concept de Delay Tolerant Networks (DTN). C'est une architecture qui gère les applications non-temps réel à faible coût, dans des conditions peu fiables, permettant la connectivité dans divers scénarios, en utilisant des véhicules pour transporter des données entre les nœuds terminaux. GeoSpray s'inspire des directives générales du protocole de routage géographique GeOpps [78]. Il utilise les informations de position géographique et d'autres paramètres de mobilité, ainsi que les adresses de destination du paquet, en s'assurant que les paquets soient

redirigés vers la destination. GeoSpray utilise le concept "Spray phase" du protocole "Spray and Wait" [114], où un petit nombre fixe de copies de paquets est distribué à des nœuds distincts du réseau. Cependant, à la place d'une réplication aveugle (telle que proposée dans Spray and Wait), GeoSpray garantit que les copies de paquets ne sont distribuées qu'à des nœuds du réseau qui se rapprochent (et/ou arriveront plus tôt) à la destination prévue du paquet. De plus, au lieu d'attendre que l'un de ces nœuds rencontre la destination et délivre sa copie de paquet (comme proposé dans la phase "wait phase" de Spray and Wait), GeoSpray permet à chaque nœud de rediriger sa copie de paquet vers un autre nœud qui peut rapprocher d'avantage les données de la destination.

5.2 Synthèse

Le tableau suivant présente une synthèse des différentes solutions présentées dans cette partie. Chaque ligne représente une caractéristique nous permettant de comparer ces différents modèles :

- **Auto-organisation** : les pairs sont organisés de manière autonome, chaque pair détient des données sur sa position géographique. Un mécanisme parallèle est mis en place pour obtenir les positions géographiques des pairs voisins. Aucun serveur central n'est prévu pour stocker et fournir les coordonnées géographiques des pairs.
- **Domaine** : domaine d'application de la solution.
- **Initialisation de vue** : la localisation géographique des pairs est prise en considération lors de l'initialisation de la vue. Chaque pair sera doté d'une vue composée de donnés sur des pairs choisis par rapport à leurs distances géographiques. D'autres paramètres peuvent aussi être considérés en combinaison avec la localisation pour améliorer les choix.
- **Mécanisme de Gossip** : les échanges entre les pairs sont supportés par un mécanisme de gossip sous-jacent. Rappelons que ces mécanismes s'avèrent très efficaces pour les environnements dynamiques principalement du fait de leurs résistances par rapport aux échecs de liens et d'exécution au niveau des nœuds.
- **Routage Géographique** : le routage des données au niveau de l'architecture est appuyé par des choix relatifs à la localisation des pairs. L'objectif principale est de réduire à chaque étape du routage la distance géographique aux nœuds destinataires.

	Dimakis et al. [2006]	Picone et al. [2010]	Fiorese et al. [2012]	VICINITY [2013]	GeoSpray [2014]
Auto-organisation	√	√	√	√	√
Domaine	Calcul d'agrégat	Protocole de localisation	BPSS	Gestion d'architecture P2P	VTDN
Initialisation de vue	√	√		√	
Mécanisme de gossip	√	√		√	
Routage Géographique	√				√

TABLE 6: Tableau de synthèse des solutions adoptant des mécanismes de géolocalisation dans des architectures pair-à-pair.
La prise en charge d'une caractéristique est indiquée par le signe √ , la non prise en charge par une case vide.

6 Accès mobile à des sources de données

Dans cette partie nous regardons la problématique de conception d'interaction et d'ergonomie dans la consultation et la contribution à des sources de données depuis un périphérique mobile. Nous traitons précisément de la question **Q5** (Quelles sont les solutions existantes pour un accès local à des données ?) du chapitre I.
Du fait de la croissance de plus en plus remarquable de la puissance des dispositifs mobiles comme les smartphones et les tablettes, les applications mobiles sont aujourd'hui au cœur des interactions au niveau du Web. Les appareils mobiles disposent présentement de résolution d'écran très performantes, permettant de supporter une large gamme d'applications. Grâce aussi à leurs capacités de transmission de données via des équipements tels que la carte wifi, le Bluetooth, etc., ces appareils ont des capacités d'accès à internet et d'échanges de données les rendant connectés de manière quasiment permanente. Les utilisateurs peuvent ainsi être équipée d'outils leurs permettant d'accéder et contribuer à des sources de données sans réelles contraintes au niveau ressource. Parallèlement, par une évolution très rapide ces dernières décennies, le Web Sémantique est devenu le choix privilégié pour représenter l'information, avec le Framework de Description de Ressource RDF pour représenter les données sémantiques et les langages d'interrogation tels que SPARQL. Ainsi grâce aux notions de schémas, vocabulaires, ontologies et dictionnaires introduit par le Web Sémantique, les ordinateurs (et les machines en général) disposent alors de moyens fiables leur permettant de comprendre et explorer le Web.
Par ailleurs, un facteur clé dans la réalisation du Web sémantique est la mise au point d'un accès mobile efficace aux données et leurs descriptions. La plupart des applications mobiles dédiées au Web sémantique accèdent à des ensembles de données RDF distants via leurs endpoints de requêtes SPARQL exposés, ce qui permet un accès efficace aux ensembles de données en ligne sans contraindre l'appareil mobile.
Cependant, l'accès et le traitement de ces données sous-entend un besoin important de ressources, de connexion internet fiable et permanent, ce qui qui n'est pas toujours évident. Mais grâce aux capacité des appareils mobiles récents, les données sémantiques peuvent être traitées au niveau local (sur les appareils mobiles). Plusieurs travaux ont été menés dans ce sens permettant ainsi de décentraliser le traitement des données sémantiques vers les appareils mobiles. Aussi les données générées par ces derniers peuvent être accessibles directement à distances par d'autres appareils sans besoins d'internet.

6.1 Travaux connexes

Schandl *et al.* [105] proposent un framework permettant la réplication et la synchronisation adaptative de graphes RDF sur les appareils mobiles. Le framework interpose des composantes qui analysent diverses sources de données d'applications sémantiques (y compris les ontologies, les requêtes et les utilisateurs) et les utilisent pour sélectionner des parties de bases de données RDF, qui sont ensuite rendues disponibles en mode offline à l'aide d'un nœud SPARQL. Ainsi, il fournit l'accès aux données du Web sémantique sans la nécessité d'une connectivité réseau permanente.
Les applications RDF se composent généralement de deux composants principaux : un endpoint SPARQL, qui encapsule un ensemble de données RDF et cache ses détails d'implémentation au client ; une application qui accède aux données RDF en émettant des requêtes SPARQL vers le endpoint et interprète les résultats.
Pour introduire une couche de réplication et de synchronisation dans une telle application sémantique, B. Schandl et al présentent deux composantes qui servent de couche médiateur entre

l'application cliente et le nœud final SPARQL : le moteur de réplication côté client et le gestionnaire de réplication côté serveur.
Le moteur de réplication est instancié sur l'ordinateur client et agit comme un proxy transparent pour les applications. La seule modification apportée aux applications est une modification de configuration : les applications doivent être reconfigurées pour interroger le endpoint SPARQL local au lieu du endpoint distant d'origine.
La tâche du gestionnaire de réplication consiste à calculer un classement pour la réplication sélective ; c'est-à-dire qu'il détermine quel sous-ensemble de données doit être répliqué sur le client. Pour ce faire, il a besoin d'accéder à l'ensemble des données RDF, qui peut généralement être accessible à travers le endpoint SPARQL.
Le gestionnaire de réplication et le moteur de réplication échangent des informations sur l'état actuel du endpoint d'origine et du cache du client via un protocole de contrôle de réplication qui est également utilisé pour coordonner l'exécution des tâches de réplication de données. Le protocole de contrôle de réplication doit garantir un maximum de disponibilité des données en mode offline dans le cache du moteur de réplication client à tout moment.

Le *et al.* [74] présentent RDF On the Go, un outil complet de stockage de données RDF et de traitement de requêtes SPARQL pour les appareils mobiles. Ils adaptent les outils Jena pour la conception et ARQ comme moteur de requêtes largement utilisés pour les environnements limités des périphériques mobiles. Les données RDF sont stockées dans les B-Trees fournies par la version légère de Berkeley DB pour les appareils mobiles.
Alors que la plupart des applications mobiles basées sur la sémantique doivent envoyer leurs requêtes vers des Endpoint SPARQL distants, certaines applications, telles que micro-Jena, Androjena et i-MoCo, stockent et interrogent les données RDF localement.
En stockant et interrogeant localement les données RDF sur l'appareil mobile de l'utilisateur, RDF On the Go contribue à améliorer l'évolutivité, à réduire les coûts de transmission et à contrôler l'accès aux informations personnelles de l'utilisateur.
Des index sont créés pour accélérer l'accès aux données, où les R-arbres sont utilisés pour l'indexation des données spatiales, et l'outil de traitement de requêtes prend en charge les requêtes SPARQL standards et spatiales.

Van *et al.* [124] présentent un service de requête orienté client qui peut être utilisé par des applications mobiles pour interroger de manière transparente et efficace de grandes quantités de petites sources de données RDF en ligne. Ce service est construit au-dessus de moteur de requête mobile existant (tel que Androjena ou RDF On the Go) pour interroger localement les données RDF (S) / OWL. Le service de requête repose sur deux composants clés :

1. Un schéma léger d'indexation, pour identifier les sources pertinentes pour une requête particulière basée sur des métadonnées de sources sémantiques (c'est-à-dire des prédicats, des objets et des types d'objet).
2. Un mécanisme de mise en cache qui stocke localement des données fréquemment utilisées.

Pour identifier les sources des ensembles de données qui contiennent des informations pertinentes pour une requête posée, avec un haut degré de sélectivité, Van *et al.* construisent et maintiennent un index contenant des métadonnées sur les sources de données. Une fois obtenue, les sources de données doivent être mises en cache localement pour un accès efficace et une réutilisation ultérieure. Chaque source de données qui peut contenir des informations pertinentes pour la requête doit être répliquée dans son intégralité. Naturellement, la quantité de données mises en cache sera limitée en raison des restrictions d'espace sur les appareils mobiles.

Ibanez *et al.* [58] présentent SU-Set (SPARQL-Update Set), un CRDT (Commutative Replicated Data Type) pour les graphes RDF mise à jour avec SPARQL Update 1.1. Un CRDT est un type de données dont les opérations, lorsqu'elles sont simultanées, donnent le même résultat indépendamment de l'ordre d'exécution. Les auteurs conçoivent un CRDT, un formalisme émergent pour la réplication optimiste pour les graphes RDF mis à jour avec les opérations SPARQL Update 1.1, garantissant ainsi la cohérence éventuelle d'un réseau social Live Linked Data (LLD). Ce dernier est considéré ici comme un système d'édition coopératif avec des besoins de faible latence et sans contraintes d'édition. Le modèle de cohérence CCI (Convergence, Causality, Intention prévention) de [119] est utilisé.
Un CRDT est constitué d'une charge utile (payload), structure interne qui détient l'état de l'objet, d'une fonction de recherche (lookup) qui interroge la charge utile et renvoie l'élément de donnée et une fonction de mise à jour (update) pour les opérations de traitement (ajout et suppression). SU-Set étend les opérations d'insertion et de suppression d'OR-Set (Observed-Removed Set) de [108] à l'union et à la différence. Dans OR-Set, chaque élément inséré est étiqueté avec un identifiant globalement unique. Ceci permet de garantir que les éléments stockés dans la charge utile sont toujours uniques, et donc, ajoutés et supprimés une seule fois. Les opérations SPARQL sur les triplets sont exécutées par l'utilisateur, puis réécrites en des opérations SU-Set avec des paires (triple, id) de manière transparente pour l'utilisateur et envoyées en aval vers les autres nœuds, où elles sont ré-exécutées à la réception.
L'approche de réplication optimiste [101] pour les données liées est instanciée comme protocole de synchronisation entre les participants autonomes. Dans ce type de protocole, les répliques peuvent mettre à jour les données et les conflits sont résolus s'ils apparaissent. Chaque site envoi ses opérations locales de façon asynchrone, et les autres nœuds les reçoivent et les exécutent, éventuellement dans des ordres différents. Le système est cohérent s'il conserve le CCI.
L'utilisation d'identifiants globaux uniques produit une surcharge sur la taille des données. Cependant, cette surcharge est considérablement réduite en considérant qu'en ajoutant un ensemble de triplets, la même étiquette peut être partagée entre eux [57], étant donné que l'unicité des paires insérées dans la charge utile est conservée. Ainsi, un seul id est envoyé en aval pour un ensemble de triplets ajoutés, délocalisant la construction des paires vers les nœuds distants, au prix d'une suppression plus coûteuse, puisque l'envoie de l'identifiant uniquement devient insuffisant pour supprimer un triple.

Ibanez *et al.* [59] proposent Col-Graph, une approche permettant de résoudre les problèmes de disponibilité, d'évolutivité et d'écriture des données liées. Ibanez *et al.* conçoivent une extension du Linked Data avec des données répliquées par des éditeurs du Linked Data. Les éditeurs peuvent effectuer des requêtes intensives et améliorer la qualité des données sur leurs répliques locales. Les sous-ensembles de données répliqués sont ici appelés "fragments". Dans Col-Graph, les éditeurs créent des fragments, des copies partielles d'autres ensembles de données, basés sur des requêtes fédérées simples de types CONSTRUCT, leur permettant d'effectuer localement des requêtes intensives sur l'union des fragments et de faire des mises à jour pour améliorer la qualité des données. Ils publient ensuite l'ensemble de données mis à jour, permettant à d'autres de copier des fragments. Ils peuvent également contacter leurs sources de données pour que ces dernières intègrent leurs mises à jour. Pour suivre les mises à jour effectuées par un participant, le triplet RDF est considéré comme le plus petite élément de connaissance directement gérable et l'insertion et la suppression d'un triplet RDF comme les deux types de base de mises à jour. Chaque mise à jour est identifiable de manière unique et transpose le graphe RDF dans un nouvel état.
Ibanez *et al.* proposent également un critère de cohérence et définissent un protocole sans coordination permettant de conserver les fragments de façon incrémentielle sans réévaluer la requête

sur la source de données. Ils proposent un comptage du nombre d'insertions et de suppressions d'un triple pour la synchronisation des données, et une annotation de chaque triplet RDF avec un élément d'un monoïde commutatif. Dans le cas où deux triples sont sémantiquement incompatibles, le critère principal pour choisir lequel d'entre eux supprimer est la confiance par rapport à la provenance des triplets. Le monoïde est utilisé pour encoder des informations de provenance supplémentaires permettant d'identifier lequel des participants est l'auteur de tel triplet.

Van *et al.* [123] proposent un service de requête mobile qui prend en charge le requêtage à la volée et intégré des données sémantiques en provenance d'une partie largement inutilisée du Web sémantique, comprenant des fichiers RDF en lignes et des informations sémantiques intégrés dans des pages Web annotées. Le service effectue l'identification dynamique, la récupération et la mise en cache des données sémantiques pertinentes par rapport à la requête. Le service de requêtes met en œuvre deux solutions, à savoir l'identification de sources pertinentes en ligne et le cache local des données, via deux composants clés. Le composant d'identification de source, appelé Source Index Model (SIM), indexe les métadonnées des sources en ligne à partir de sources sémantiques en ligne, dans le but de permettre une identification pointilleuse des sources. Il en résulte un index compact rapide à mettre à jour et à maintenir. Etant données des métadonnées analogues extraites des requêtes, la SIM utilise les métadonnées indexées pour identifier les sources pertinentes pour la requête.
Le composant de mise en cache met en cache localement les données source téléchargées et comporte deux implémentations variantes, appelées Source Cache et Meta Cache. Chaque variante présente une organisation de cache différente : le cache source organise les données en cache autour de la source d'origine, tandis que le méta-cache arrange les données en fonction des métadonnées partagées des sources. Pour gérer la mémoire occupée et l'espace de stockage, des stratégies de remplacement (ou de suppression) identifient les données à déplacer vers un stockage persistant ou à supprimées entièrement. Enfin, une stratégie de validation du cache est appliquée pour assurer la mise à jour du cache.

6.2 Synthèse

Le tableau suivant présente une synthèse des différentes solutions présentées dans cette partie. Chaque ligne représente une caractéristique nous permettant de comparer ces différents modèles :

- **Accès local** : les données sont disponibles sur la mémoire locale des pairs. Une réplication totale ou partielle des sources de données est effectuée au niveau des pairs. Les sources les plus pertinentes par rapport aux requêtes sont analysées et répliquées partiellement en locale pour le traitement des prochaines requêtes.
- **Mise en cache** : un mécanisme de mise en cache est implémenté pour permettre aux pairs d'accéder rapidement aux données les plus demandées. Ceci s'avère aussi très pertinent dans des situations de connexion en intermittence ou les requêtes sont analysées avec les données caches avant les données en mémoire locale.
- **Indexe de source** : des mécanismes d'indexation de sources sont mise en place. Elles permettent aux pairs d'accéder rapidement aux sources de données les plus interrogées.
- **Mise à jour automatique** : une mise à jour automatique des données locales est effectuée. Ceci permet de maintenir les données locales et les données en lignes au même niveau (à jour) sans besoin d'intervention manuelle, sachant que les sources de données en ligne peuvent être assujetties à de modifications fréquentes.
- **CCI** : Convergence, Causality Preservation and Intention preservation. Le système adopte un modèle de cohérence qui maintient les trois propriétés précédentes [119]. Convergence :

lorsque le même ensemble d'opérations est exécuté sur tous les sites, ils auront tous le même état ; Causality preservation : si une opération O est exécutée avant une autre opération O', le même ordre d'exécution est respecté sur tous les sites ; Intention préservation : pour tout opération O, les effets de l'exécution de O sur tous les sites sont les mêmes que les intentions de O, et l'effet de l'exécution de O ne change pas les effets des opérations indépendantes.

- **Données de provenance** : des informations de provenance des données sont considérées lors des mises à jour. Cela permet de gérer des conflits entre des données provenant de sources différentes en se basant sur la confiance par rapport à la source.
- **OWA** : Semantic Web's Open World Assumption. La solution est implémentée en prenant en considération l'hypothèse de monde ouvert du web sémantique ce qui permet un accès totalement intégré au web sémantique. Les requêtes peuvent alors référencer plusieurs sources à la fois.
- **Type de données** : types des données échangées

	Schandel et al. [2009]	Le Phuoc [2011]	Woensel et al. [2011]	Ibanez et al. [2013]	Ibanez et al. [2014]	Woensel et al. [2016]
Accès local	√	√	√	√	√	√
Mise en cache	√		√			√
Index de source		√	√			√
Mise à jour automatique			√	√	√	√
CCI				√		
donnée de provenance					√	
OWA						√
Type de donnée	RDF	RDF	RDF	RDF	RDF	RDF

TABLE 7: Tableau de synthèse des solutions traitant de l'accès mobile à des données. La prise en charge d'une caractéristique est indiquée par le signe √ , la non prise en charge par une case vide.

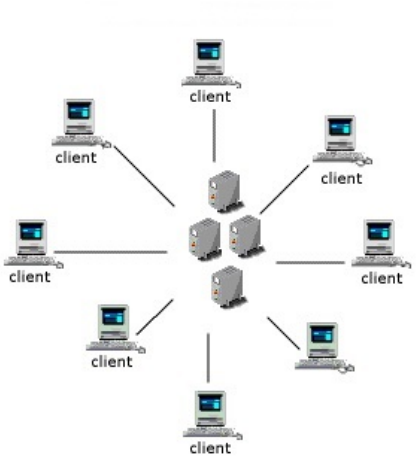


FIGURE 3: Exemple d'architecture client-serveur

7 Partage et modification collaborative de graphe RDF

Dans cette partie, nous nous intéressons à classifier les approches existantes pour la construction d'un système de partage de données en se référant aux modèles de données RDF adoptés.
De nombreux travaux ont été menés pour la construction d'infrastructures constituées de nœuds centralisés dédiés au partage de données et capables d'assurer la disponibilité et la cohérence de ces données. Ces infrastructures clients-serveurs du web de donnnées (Illustration : **Figure** 3) permettent la mise en place de clients légers pour les utilisateurs, transférant ainsi les charges de traitements et de calculs au niveau des serveurs. Elle sont aussi trés adaptées (par rapport aux infrastructures décentralisées) en termes de rapidité lors des scénarios de recherche d'informations sur de vastes quantités de données. Les nœuds centraux peuvent coopérer pour partager des connaissances. Mais le problème majeur lié à cette approche est que les résultats des requêtes peuvent changer avec le temps, en fonction de la disponibilité des nœuds. Si un nœud devient indisponible, il n'est relayé par aucun autre nœud du réseau. Ainsi, tout nœud peut constituer un point d'échec pour le système.
Pour pallier ces contraintes centralisées, plusieurs approches dans la littérature considèrent des scénarios distribués, basés sur des réseaux pair-à-pair. Le réseau assure une forte résistance aux défaillances techniques des nœuds. Ces systèmes pair-à-pair (P2P) offrent de nombreux avantages des systèmes distribués décentralisés mais souffrent de problèmes liés à la disponibilité et à la fiabilité des sources et des données. Pour surmonter ces contraintes, les systèmes pair-à-pair décentralisés s'appuient sur des mécanismes permettant d'atténuer l'impact des scénarios de déconnexion sur la disponibilité des données, mais aussi d'équilibrer les charges de traitement du système sur l'ensemble des pairs.
Dans le cas des systèmes décentralisés de partage de données RDF, ces mécanismes permettent pour un graphe, de pouvoir être en partie ou totalement répliqué, partagé et modifié collaborativement à travers les pairs participant à l'architecture. Dans cette optique, plusieurs solutions ont été proposées dans la littérature adoptant chacune des principes conceptuellement différents :

1. Le graphe est stocké sur tous les pairs du réseau qui peuvent modifier et/ou supprimer des portions de manière autonome. Ces opérations d'ajout ou de suppression seront ensuite

propagées à travers l'architecture suivant des techniques variables qui visent à assurer la cohérence des différentes copies.

2. Le graphe est distribué à travers l'ensemble des pairs sur un environnement pair à pair structuré (une DHT). Dans ce cas, les requêtes sont routées directement vers les pairs détenant les données souhaitées.
3. Le graphe est distribué sur l'ensemble des participants en adoptant une architecture pair-à-pair sémantique (semantic peer-to-peer network). Là aussi, les requêtes sont routées vers les nœuds voisins et les nœuds appartenant à d'autres clusters si nécessaire.
4. Le graphe est hébergé sur le cloud. Un cache collaboratif sur les navigateurs est alors mis en place sur l'architecture pour pallier aux scénarios de déconnexion.

7.1 Réplication de graphe

Afin d'accroître la disponibilité et la fiabilité, les techniques de réplication des données sont considérées comme monnaie courante dans les systèmes informatiques pair-à-pair.
La réplication de données représente aujourd'hui un principe de conception pertinent pour assurer la fiabilité, l'équilibrage de charge, mais surtout la haute disponibilité des données et services dans les systèmes distribués. Elle permet d'améliorer les performances et la disponibilité du partage d'informations dans un réseau à grande échelle.
La réplication de données consiste précisément à conserver plusieurs copies d'objets de données appelées répliques sur des sites distincts. Le graphe est stocké sur la totalité (ou une partie) des pairs du réseau qui peuvent modifier et/ou supprimer des portions de manière autonome. Ces opérations d'ajout ou de suppression seront ensuite propagées à travers l'architecture pour assurer la cohérence des différentes répliques.
Ceci permet d'assurer une haute disponibilité des données localement pertinentes notamment dans les systèmes pair-à-pair où le stockage et le traitement des données sont répartis entre les pairs qui peuvent avoir des comportements dynamiques (joindre et quitter le réseau à tout instant).

7.1.1 Modèles de réplication

Les mécanismes de contrôle des répliques peuvent être classés selon trois critères [51] : où se déroulent les mises à jour (single-master ou multi-master), quand les mises à jour sont propagées à toutes les répliques (synchrone vs asynchrone) et comment les répliques sont réparties sur le réseau (réplication complète ou partielle).
Dans l'approche Single-master, il n'y a qu'une seule copie primaire pour chaque objet répliqué. Le Single-master (maître-unique) permet à un seul site d'avoir un contrôle total sur la réplique (droits de lecture et d'écriture) alors que les autres sites ne peuvent avoir que le droit de lecture. Ce modèle est également connu comme l'approche Master-slave (maître-esclave) en raison de l'interaction du nœud maître avec les autres nœuds (esclaves) hébergeant la réplique. L'avantage de ce modèle est la centralisation des mises à jour sur une copie unique, simplifiant le contrôle des accès concurrents. Cependant, cette centralisation introduit un goulot d'étranglement potentiel et un point d'échec. Une panne sur un site maître bloque ainsi les opérations de mise à jour et limite ainsi la disponibilité des données.
Dans l'approche multi-master, plusieurs sites contiennent une copie primaire du même objet. Toutes ces copies peuvent être mises à jour simultanément. Chaque site peut modifier ses répliques. Cette approche est plus flexible que le Single-master, car dans le cas d'une défaillance d'un master, d'autres masters peuvent gérer les répliques.

Il existe deux approches de base pour le placement des répliques : la réplication complète et la réplication partielle.
La réplication complète consiste à stocker une copie de chaque objet partagé sur tous les sites participants. Cette approche offre un équilibrage de charge simple puisque tous les sites ont les mêmes capacités et la disponibilité maximale que n'importe quel site peut remplacer tout autre site en cas d'échec.
Avec une réplication partielle, chaque site détient une copie d'un sous-ensemble d'objets partagés, de sorte que les objets répliqués sur un site peuvent être différents des objets répliqués sur un autre site. Cette approche nécessite moins d'espace de stockage car les mises à jour ne se propagent que vers les sites concernés. Ainsi, ces mises à jour produisent une charge réduite pour le réseau et les sites.

La réplication peut être effectuée d'une manière synchrone ou asynchrone [51].
L'approche de propagation synchrone des mises à jour applique les modifications apportées à toutes les répliques dans le contexte de l'opération qui génére les mises à jour. Par conséquent, lorsque l'opération est réalisée, toutes les répliques ont le même état. Avec la réplication synchrone, le nœud origine de la transaction (ensemble d'opérations de mise à jour) propage les opérations de mise à jour dans le contexte de la transaction à toutes les autres répliques avant d'exécuter cette transaction. Il existe plusieurs algorithmes et protocoles permettant de réaliser ce process [70, 52]. Ainsi, la propagation synchrone impose une cohérence éventuelle entre les répliques. Bernstein *et al.* [12] définissent ces critères de cohérence comme une one-copy-serializability (sérialisation d'une copie), c'est-à-dire que malgré l'existence de plusieurs copies, un objet apparaît comme une copie logique (équivalent d'une copie) et un ensemble d'accès à l'objet sur plusieurs sites est équivalent à exécuter en série ces accès sur un seul site.
L'approche asynchrone ne modifie pas toutes les répliques dans le contexte de la transaction englobant les mises à jour. La transaction est d'abord exécutée sur le site local et ensuite, les mises à jour se propagent sur les sites distants. Un avantage de la propagation asynchrone est que l'indisponibilité des répliques ne bloque pas le processus de mise à jour, ce qui améliore la disponibilité des données.

Les solutions de réplication asynchrone peuvent être classées comme optimistes ou non-optimistes en fonction de leurs hypothèses par rapport aux mises à jour conflictuelles [101, 89].
La réplication asynchrone non optimiste suppose que des conflits de mise à jour sont susceptibles de se produire et implémente des mécanismes de propagation permettant de les éviter. Elle utilise la synchronisation pendant la propagation des répliques et bloque les autres utilisateurs au cours d'une mise à jour. De manière générale, chaque élément de données répliqué est assigné à un site de copie principale (primaire) et des sites de copie secondaires multiples et seule la copie primaire peut être modifiée. La réplication asynchrone non optimiste n'est pas aussi flexible que l'approche optimiste, mais elle fournit des valeurs à jour pour les accès locaux avec une forte probabilité.
En général, la réplication asynchrone optimiste repose sur l'hypothèse optimiste selon laquelle les mises à jour conflictuelles ne se produisent que rarement, voire jamais. Les mises à jour sont donc propagées en arrière-plan. Les mises à jour conflictuelles sont traitées plus tard, ce qui signifie que l'application doit tolérer un certain niveau de divergence entre les répliques. Cela est acceptable pour une large gamme d'applications (par exemple : le service de nommage Internet DNS, les systèmes de bases de données mobiles, le développement de logiciels collaboratifs, etc.).
Les algorithmes optimistes passent à l'échelle sur un grand nombre de répliques puisse qu'il y a peu de synchronisation entre les sites. Ces approches améliorent la disponibilité ; les applications

ne bloquent pas lorsque le site local ou distant est indisponible. Ce type de réplication permet également une configuration dynamique du réseau, les pairs peuvent joindre ou quitter le réseau sans affecter la propagation de la mise à jour. Il existe des techniques qui permettent une réplication épidémique qui propage des opérations de manière fiable à toutes les répliques.

Le dénominateur commun de tous les systèmes de réplication existants est la nécessité de maintenir la cohérence des répliques. La plupart de ces systèmes de réplication optimistes assurent une cohérence éventuelle [101, 106] entre les répliques. La cohérence éventuelle peut être formellement définie en fonction de la notion d'équivalence de planning. Un planning représente ici une liste d'opérations ordonnées. Deux plannings sont équivalents lorsque, à partir du même état initial, ils produisent le même état final. Notez qu'un état final n'inclut pas les opérations provisoires (c'est-à-dire les opérations non encore exécutées), mais seulement celles exécutées.
Si un planning contient des opérations commutatives, l'échange de leur commande conserve l'équivalence. Par conséquent, un objet répliqué est finalement cohérent lorsqu'il répond aux conditions suivantes, en supposant que toutes les répliques partent du même état initial :

- À tout moment, pour chaque réplique, il existe un préfixe du planning qui équivaut à un préfixe du planning de chaque autre réplique.
- Le préfixe exécuté de chaque réplique augmente au fil du temps d'une manière monotone.
- Pour chaque opération soumise α, α ou $\neg\alpha$ seront finalement inclus dans le préfixe exécuté, où $\neg\alpha$ désigne une opération annulée.
- Toutes les opérations non annulées dans le préfixe exécuté peuvent être exécutées avec succès.

Le degré de réplication (complet ou partiel), ainsi que la source et le mode de propagation des mises à jour dans le système représentent des caractères fondamentaux pour les systèmes de réplication de données.

7.1.2 Travaux connexes

BAYOU [121] est un un système mobile de recherche de base de données qui permet à un utilisateur de répliquer une base de données sur un ordinateur mobile, de le modifier tout en étant déconnecté et de se synchroniser avec toute autre réplique de la base de données. Dans Bayou, chaque opération est associée à une vérification de dépendance et une procédure de fusion. La vérification de dépendance est exécutée pour vérifier si l'opération est en conflit avec d'autres alors que la procédure de fusion est exécutée pour réparer l'état de la réplique en cas de conflit. Dans Bayou, un seul site principal décide quelles opérations doivent être exécutées ou interrompues et notifie les autres sites sur la séquence d'éxécution des opérations. Bayou reste différent des systèmes single-master car il permet à tout site de soumettre des opérations et de les propager, ce qui permet aux utilisateurs de voir rapidement les effets des opérations.

RDFGrowth [122] se focalise sur le partage de données sémantiques où un seul pair peut modifier les connaissances partagées alors que les autres ne peuvent qu'accéder en lecture seul. RDFGrowth vise un scénario particulier où les pairs participent à des groupes d'intérêt afin de développer leurs connaissances internes sur un ou plusieurs thémes. Dans un environnement peu fiable, les pairs répondent uniquement aux requêtes nécessitant une charge de calcul minimale et ne fournissent aucune garantie de réponse. Les pairs effectuent la navigation et l'interrogation d'instructions Web sémantiques sur une base de données locale sans générer directement de trafic réseau ou d'exécution de requêtes distantes. La base de données s'agrandi en apprenant d'autres pairs du groupe P2P en utilisant uniquement une quantité minimale de requêtes directes qui sont exécutables avec un coût de calcul faible et prévisible. Bien que les graphes RDF complets

puissent être traités, la conception permet à un pair d'apprendre uniquement par des ressources jugées intéressantes par une communauté spécifique et permet de marquer les informations reçues selon des règles de confiance individuelles.

Le concept de OT (Operational Transformation) [118, 119, 38] a été développée pour les éditeurs collaboratifs. OT suppose qu'un utilisateur applique des opérations directement sur le site local, puis propage ces commandes vers d'autres sites. Par conséquent, tous les sites effectuent le même ensemble d'opérations, mais peut-être dans des ordres différentes. L'objectif d'OT est de préserver l'intention des opérations et d'assurer la convergence des répliques. Ceci est obtenu en définissant pour chaque paire d'opérations simultanées une règle de réécriture. Dans [90], il est prouvé l'exactitude d'OT pour une feuille de calcul partagée. Molli *et al.* [80] étendent l'approche OT pour prendre en charge un système de fichiers répliqué. Ferrie *et al.* [42] traite des opérations d'annulation dans le contexte de l'OT en fournissant un algorithme d'annulation général basé sur la définition d'une transformation générique.

Skaf *et al.* [111] présentent SWOOKI le premier wiki sémantique pair-à-pair. Le réseau wiki sémantique pair-à-pair est composé d'un ensemble de nœuds wiki sémantiques autonomes interconnectés qui peuvent joindre et quitter le réseau dynamiquement. C'est un système P2P non structuré et décentralisé qui ne requiert aucune coordination ou connaissance centrale. Il s'appuie sur un modèle de communication symétrique où chaque pair peut agir à la fois comme serveur et client. Chaque pair héberge une réplique de données partagées.
La gestion des données est basée sur une réplication optimale des données où chaque pair héberge une copie des pages wiki et les données sémantiques associées. Chaque pair peut offrir de manière autonome tous les services d'un serveur wiki sémantique, l'accès, la recherche et les requêtes sont exécutées localement sans aucun routage des requêtes sur le réseau (réplication total du graphe). Le modèle de données est une extension du modèle de données WOOKI [131] pour tenir compte des données sémantiques. Chaque wiki sémantique se voit attribuer un identificateur unique global nommé NodeID. Comme dans n'importe quel système wiki, l'élément de base est une page wiki et chaque page wiki reçoit un identificateur unique PageID, qui est le nom de la page.

Weiss *et al.* [132] présentent l'algorithme Logoot-Undo CRDT qui intègre la fonctionnalité "undo anywhere, anytime" ("annuler n'importe où et n'importe quand"). Pour des raisons d'efficacité et de tolérance de panne, le contenu du réseau est répliqué. Cette réplication peut être soit totale - c'est-à-dire que chaque pair possède une réplique - ou partielle. L'approche Logoot-Undo appartient au framework CRDT (Conflict-Free Replicated Data Type) dont l'idée principale est de fournir une commutabilité réelle entre les opérations simultanées. Lorsque le type de données est une liste ordonnée, comme un document texte, la commutabilité entre les insertions dans la liste peut être obtenue avec un identifiant unique et totalement ordonné pour chaque ligne (ou chaque caractère). L'idée principale est d'associer un identifiant à chaque ligne. Cet identificateur de ligne doit être unique, dense et totalement ordonné. Un utilisateur peut modifier le document en insérant ou en supprimant des lignes. Chaque opération effectuée sur le document implique une modification de la table d'identificateurs associée. Ensuite, la suppression d'une ligne dans le document nécessite de supprimer l'identifiant associé à cette ligne. De même, l'insertion d'une ligne dans le document nécessite d'insérer un identifiant pour cette ligne. Une opération est créée pour chaque modification sur le document. Une opération est soit une insertion, soit une suppression. Les opérations sont regroupées en patchs et envoyées à toutes les autres répliques pour ensuite être intégrer. Chaque patch est livré une et une seule fois pour chaque réplique.

Dans [113], Spaho *et al.* considèrent les systèmes pair-à-pair avec une architecture super-pair

(SP). Un groupe de pairs comporte plusieurs pairs qui peuvent être géographiquement éloignés les uns des autres mais qui ont un objectif commun. Le travail se compose de plusieurs tâches qui doivent être complétées par les pairs du groupe. L'une des tâches les plus importantes est la réplication des documents sur les pairs du groupe. Chaque pair peut communiquer directement avec n'importe quel autre pair du groupe. Les groupes de pairs sont composés du même nombre de pairs. Le groupe de pairs possède un gestionnaire central qui est le super-pair. Ce dernier attribue des tâches aux pairs du groupe et conserve la trace de l'accomplissement du travail. Il facilite la communication avec d'autres pairs dans d'autres groupes de pairs. Le super-pair assure la connexion entre les pairs du groupe et les autres pairs et super-pairs du réseau. Si une partie ou la totalité du document dans un pair change, d'autres pairs qui ont la réplique de ce document appliquent ces modifications. Pendant la réplication, il est important de trouver un bon facteur de réplication. Le facteur de réplication est considéré comme le nombre total de documents répliqués sur le nombre total de documents, c'est-à-dire la somme des répliques et des documents originaux dans tous les pairs. Une planification minutieuse devrait être faite lors de la prise de décision sur les documents à reproduire et sur le choix des pairs.

CRATE [84] est un éditeur collaboratif décentralisé en temps réel qui s'exécute directement dans les navigateurs Web grâce au WebRTC. Pour fournir la disponibilité et la réactivité des documents, Crate suit le schéma de réplication optimiste. Chaque éditeur reproduit le document localement et exécute directement ses opérations. Ensuite, l'éditeur diffuse ses modifications à tous les autres participants. Le système est correct si les éditeurs intégrant un même ensemble de modifications ont des répliques convergentes vers un état équivalent, c'est-à-dire que les utilisateurs lisent le même document. Cette propriété est la forte consistance éventuelle [107]. Crate utilise un type de données répliqué sans conflit (Conflict-Free Replicated Data Type : CRDT) pour les séquences pour assurer cette propriété. Les CRDT garantissent une cohérence au prix d'un identifiant unique associé à chaque élément de la séquence.

7.2 Graphe réparti et système pair-à-pair structuré(cas des DHTs)

Des progrès significatifs ont été réalisés dans le cadre de l'effort de recherche consistant à construire des systèmes de gestion de données RDF à haute performance qui fonctionnent sur une seule machine.Ces systèmes ont démontré de grandes performances sur une seule machine pour les ensembles de données contenant des millions et, dans certains cas, des milliards de triples. Malheureusement, comme la quantité de données RDF continue d'évoluer, il n'est plus possible de stocker des ensembles de données complets sur une seule machine et d'être en mesure d'accéder aux données avec des performances raisonnables.
Les approches basées sur la partition permettent de partitionner un graphe RDF en plusieurs fragments et de les placer sur des sites différents dans un système parallèle/distribué. Chaque site héberge un store RDF centralisé. L'adaptation du Web sémantique pour la gestion de l'information personnelle et le désir croissant de mobilité s'accompagnent souvent de situations où aucune connectivité réseau n'est disponible et que l'accès aux données à distance est limité. De telles situations pourraient être évitées lorsque les appareils mobiles peuvent fonctionner sur des répliques de données hors ligne et synchroniser les modifications lorsque la connectivité est rétablie. Toutefois, en raison du stockage et de la puissance de calcul encore limités des appareils mobiles, une sélection soigneuse des informations à reproduire devient primordiale ; idéalement d'une manière automatique, transparente et adaptative.

7.2.1 Table de Hachage Distribuée (DHT)

Les systèmes de gestion de bases RDF centralisés et les systèmes de recherche de gestion des triplets tels que RDFStore [95], Jena [20] ont été conçus pour prendre en charge le stockage et l'extraction de données RDF. Bien que ces systèmes soient de conception simple, ils souffrent des limites traditionnelles des approches centralisées. En alternative à ces systèmes centralisés, des approches pair-à-pair ont été proposées pour surmonter certaines de ces limitations en créant des systèmes (entièrement) décentralisés de stockage et d'extraction des données.
Les réseaux pair-à-pair structurés utilisant des tables de hachages distribuées (DHT) maintiennent un réseau de superposition structuré parmi les pairs et utilisent le routage des messages à la place du flooding. La fonctionnalité de base qu'ils offrent est la recherche (par clé), qui renvoie l'identité du nœud stockant l'objet avec cette clé. Une table de hachage distribuée (ou DHT pour Distributed Hash Table), est une technologie permettant la mise en place d'une table de hachage dans un système réparti. La DHT fournit une interface générique, ce qui facilite son adoption par une large variété d'applications comme substrat de stockage : stockage et axtraction des données par le biais de clés. Les DHTs stockent des blocs de données sur des centaines ou des milliers de machines connectées à Internet, répliquent les données pour plus de fiabilité et les localisent rapidement malgré l'exécution sur des liaisons étendues à grande latence.
Pour assurer la disponibilité des objets malgré la saturation, les DHT s'appuient sur la réplication : plusieurs copies de chaque objet sont stockées sur différents nœuds. Un protocole de réplication est chargé d'assurer qu'à tout moment, chaque objet est répliqué sur un nombre suffisamment grand de répliques (appelé ensemble de répliques). Le protocole de réplication est également chargé de décider où les répliques devraient être localisées. Plusieurs stratégies de réplication ont été proposées ces dernières années.
Réplication de voisinage : dans tous les DHT, les nœuds stockent et maintiennent une liste des noeuds distants dans leur table de routage selon l'algorithme de sélection DHT. Dans Chord [116], par exemple, comme la DHT stocke et maintient naturellement une liste de successeurs, la réplication de voisinage signifierait une réplication de successseurs. Dans Pastry [99], une liste des nœuds feuilles est maintenue et la réplication des feuilles peut être employée à la place. Aucun état supplémentaire n'est nécessaire car la sélection et la maintenance du successeur ou des feuilles sont prises en charge par le protocole d'origine. Cela rend certains programmes de réplication plus appropriés à certains DHT que d'autres. Cependant, il est toujours possible d'appliquer toutes les techniques de réplication à toutes les DHT mais au coût d'un état supplémentaire, requis pour la gestion des répliques.
Réplication de chemin : la réplication de chemin réplique un objet de données le long du chemin de recherche traversé par le message de recherche, qui forme la source du nœud racine. Dans le routage DHT, au fur et à mesure que les messages sont acheminés par des nœuds numériquement plus proches de la clé, les recherches générées par différentes sources convergent probablement sur un nœud avant la racine réelle. La réplication du chemin tend à répliquer les objets vers des nœuds qui se situent sur le même chemin et, par conséquent, raccourcit les chemins et accélère les recherches. Tapestry [139] utilise la technique de réplication de chemin d'accès en insérant des données entre des nœuds entre la source et le nœud racine de l'objet.
Réplication à clé de publication multiple : pour améliorer la disponibilité des données, une clé est associée à n points dans l'espace des coordonnées et, en conséquence, répliquée à n nœuds distincts du système. Une paire de clés n'est alors disponible que lorsque toutes les n répliques et les n routes correspondantes sont simultanément indisponibles. Il s'agit d'un schéma de réplication différent, car les nœuds doivent connaître la méthode de réplication pour insérer les données correctement et pour pouvoir les rechercher.

7.2.2 Travaux connexes

Freenet [26] est une application de partage de fichiers (texte, sons, données,etc) qui protège également l'anonymat des auteurs et des lecteurs. Les nœuds Freenet contiennent 3 types d'informations : les clés (qui sont analogues aux URLs Web), les adresses d'autres nœuds Freenet qui sont également susceptibles de connaître des clés similaires et éventuellement les données correspondant à ces clés. Un nœud qui reçoit une requête pour une clé pour laquelle elle ne connaît pas l'emplacement exact transmet la requête à un nœud Freenet connu et dont les clés sont plus proches de la clé demandée. Si un nœud ne parvient pas à localiser le contenu souhaité, il renvoie un message d'échec à son nœud prédécesseur qui essaiera alors le nœud successeur alternatif qui est le meilleur choix suivant. Les auteurs émettent l'hypothèse que la qualité du routage devrait s'améliorer avec le temps, pour deux raisons. D'abord, les nœuds devraient se spécialiser dans la localisation d'ensembles de clés similaires, car un nœud répertorié dans les tables de routage sous une clé particulière aura tendance à recevoir principalement des demandes de clés similaires. De plus, il sera mieux informé sur ses tables de routage sur les autres nœuds qui portent ces clés. Deuxièmement, les nœuds devraient devenir spécialisés dans le stockage de clusters de fichiers ayant des clés similaires. La transmission d'une requête avec succès entraînera que le nœud gagne une copie du fichier demandé et la plupart des demandes seront pour des clés similaires et, par conséquent, le nœud va acquérir des fichiers avec des clés similaires.

PAST [100] est un utilitaire de stockage persistant pair à pair à grande échelle utilisant Pastry[99]. Dans PAST les répliques d'un fichier sont stockées sur les n nœuds les plus proches de l'ID du fichier. Le message d'insertion est routé à l'aide de l'identifiant du fichier comme destination. Lorsque le message atteint le premier parmi les n nœuds les plus proches de l'ID du fichier, le nœud accepte la responsabilité d'une réplique du fichier et transmet la demande d'insertion aux n-1 autres nœuds avec l'ID du nœud le plus proche de l'ID du fichier.

Le système de fichiers coopératifs (CFS) [29] est un système pair-à-pair qui fournit une garantie de l'efficacité, de la robustesse et de l'équilibre des charges du stockage et de la récupération des fichiers (de type mp3 par exemple). CFS fournit un stockage distribué de fichiers en lecture seule. Il est structuré comme une collection de serveurs qui fournissent un stockage de niveau bloc (block-level).
Le noyau du logiciel CFS se compose de deux couches, DHash[29] et Chord[116]. La couche DHash (hash distribué) effectue des extractions de blocs pour le client, distribue les blocs entre les serveurs et maintien des copies mises en cache et répliquées. DHash utilise le système de recherche distribué Chord pour localiser les serveurs responsables d'un bloc. Chord implémente une opération de type hash qui mappe des identifiants de bloc aux serveurs.
DHash fournit un équilibre de charge pour les grands fichiers populaires en répartissant les blocs de chaque fichier sur de nombreux serveurs. Pour équilibrer la charge imposée par les petits fichiers populaires, DHash met en cache chaque bloc sur les serveurs susceptibles d'être consultés par de futures recherches Chord pour ce bloc.
Chaque client CFS détient trois couches logicielles : un système de fichiers client, une couche de stockage DHash et une couche de recherche Chord. Le système de fichiers client utilise la couche DHash pour récupérer les blocs. La couche DHash client utilise la couche Chord client pour localiser les serveurs qui contiennent les blocs recherchés.

Nejdl *et al.* [86] proposent une architecture P2P basée sur RDF appelée Edutella qui utilise des super-pairs. Dans ce type de topologie, un ensemble de nœuds est sélectionné pour former le réseau super-pair, en construisant le "backbone" du réseau P2P, tandis que les autres pairs se

connectent dans une topologie en étoile aux super-pairs. Les super-pairs forment une topologie appelée HyperCuP et sont responsables du traitement des requêtes. Dans cette organisation, chaque super-pair peut être considéré comme la racine d'un arbre d'extension qui est utilisé pour le routage des requêtes, la diffusion et la mise à jour des indices. Chaque pair envoie des indices de ses données à son super-pair.
Edutella implémente un système P2P basé sur un schéma où le système prend en compte des informations de schéma et considère aussi l'optimisation et le routage des requêtes. Les indices de données peuvent être stockés dans différentes granularités, comme le schéma, la propriété, la valeur de la propriété ou la plage de valeur de la propriété.
Le routage des requêtes dans Edutella s'améliore en utilisant une stratégie de regroupement basée sur la similarité au niveau super-pair pour éviter autant que possible la diffusion pendant la phase de routage des requêtes. Cette approche tente d'intégrer de nouveaux pairs avec des pairs existants qui ont des caractéristiques similaires, par exemple, sur une ontologie thématique partagée.

RDFPeers [18] est l'une des premières solutions de stores RDF à système pair-à-pair structuré. L'idée principale est d'utiliser une superposition MAAN (Multi-Attribute Addressable Network) [19] pour indexer trois fois un triplet : une fois basée sur le sujet, une autre sur le prédicat et une derniere sur l'objet. Il y a quelques inconvénients avec cette conception. D'abord, il existe un facteur de réplication de trois pour tous les triplets. En plus, les sémantiques du schéma RDF sont totalement ignorées lors du routage des requêtes. En raison de sa couche de stockage sous-jacente, les RDFPeers peuvent dans le pire des cas, par exemple, pour les requêtes à faible sélectivité, nécessiter un nombre linéaire de sauts par rapport à la taille de la superposition. RDFPeers est aussi susceptible de provoquer des déséquilibres de charge, puisque les pairs responsables des mots-clés populaires et réservés deviennent rapidement submergés.

GridVine [4] est une infrastructure de gestion de données distribuée basée sur P-Grid DHT [3] sur la couche de superposition et qui maintient en plus une couche de médiation sémantique. GridVine permet la recherche distribuée, le stockage persistant et l'intégration sémantique des données RDF. La couche supérieure profite de l'architecture de superposition pour gérer efficacement les données hétérogènes, y compris les schémas et les alignements de schéma. Comme dans RDFPeers [18], GridVine indexe les triples trois fois sur la couche P-Grid en fonction de leurs sujets, objets et prédicats. Afin d'intégrer toutes les données hétérogènes sémantiquement liées mais syntaxiquement partagées par des pairs au niveau P-Grid, GridVine prend en charge la définition des alignements sémantiques par paires. L'alignement permet de reformuler une requête contre un schéma donné dans une nouvelle requête posée contre un schéma sémantiquement similaire. L'alignement de schéma utilise les instructions de OWL (Web Ontology Language) pour relier deux schémas similaires. Les opérations de recherche sont effectuées en hachant la (les) partie (s) constante (s) du motif du triplet. Une fois que l'espace clé est atteint, la requête sera transmise aux pairs responsables de cet espace clé.

Le projet BRICKS [98] offre une disponibilité de données élevée dans les DHT par réplication. Pour la réplication d'une donnée, BRICKS stocke les données dans la DHT à l'aide de plusieurs clés, qui sont corrélées à la clé de données. Pour pouvoir récupérer une réplique à jour, BRICKS utilise le versioning. Chaque réplique a un numéro de version qui s'incrémente après chaque mise à jour. Cependant, en raison de mises à jour simultanées, il se peut que deux répliques différentes aient le même numéro de version, ce qui rend impossible la détection de la dernière réplique.

PAGE (Put and Get Everywhere) [31] est un référentiel distribué RDF basé sur Bamboo DHT

[96]. Le modèle de données RDF étend le modèle de données RDF standard en introduisant la notion de contexte. Ce concept dépend de l'application. Par exemple, dans le cas d'utilisation d'intégration d'informations, le contexte est l'URI du fichier ou le référentiel à partir duquel un triplet RDF est originaire. Dans d'autres scénarios, les triples RDF partageant la même sémantique peuvent avoir le même contexte. Par conséquent, dans PAGE, chaque triplet RDF, désigné par t = (s, p, o), est associé à un contexte, étiqueté c. Le triplet RDF combiné avec le contexte forme un quad. Chaque quad est indexé six fois et identifié par un identifiant construit en concaténant les valeurs de hash de ses éléments (s, p, o, c).

7.3 Graphe réparti et superposition sémantique

Le problème fondamental qui rend difficile la recherche dans les systèmes pair-à-pair existants est qu'en ce qui concerne la sémantique, les documents sont répartis d'une manière aléatoire. Considérant une requête, soit le système doit rechercher sur un grand nombre de nœuds, soit l'utilisateur coure le risque de manquer des documents pertinents. Pour résoudre ce problème, la notion de superposition sémantique est introduite : un réseau logique où les contenus sont organisés autour de leur sémantique, de sorte que la distance entre deux documents du réseau est proportionnelle à leur dissemblance en terme sémantique. Les systèmes sémantiques pair-à-pair (SP2P) combinent deux technologies complémentaires : réseautage pair-à-pair et ontologies.
Outre les réseaux pair-à-pair non-structurés et structurés et basés sur DHT, le réseau pair-à-pair sémantique conserve les caractères sémantiques des rôles des nœuds dans les communautés. Les nœuds sont identifiés comme des noms de domaine classés par la signification sémantique des rôles dans les organisations. Les nœuds construisent le réseau pair-à-pair sémantique selon leurs domaines classés sous forme d'arbre hiérarchique.
Au cours des dernières années, plusieurs infrastructures de gestion de données pair-à-pair permettant le partage de données sémantiques ont émergé. Ces systèmes portent sur une approche d'intégration décentralisée : chaque pair représente un système d'information autonome et l'intégration de données sémantiques est réalisée en établissant des alignement directement entre les différents pairs. Même si ces systèmes partagent l'intégration de schéma en adoptant différents formalismes, tous se résument à l'infrastructure sous-jacente et considèrent le système comme un graphe de sources de données interconnectées(**Figure** 4).
L'utilisation des techniques du Web sémantique dans les systèmes pair-à-pair remonte au projet SWAP [36], coordonné par l'Université de Karlsruhe.

7.3.1 Réseaux de superposition sémantique

Crespo *et al.* [28] ont proposé le concept de réseau de superposition sémantique (Semantic Overlay Networks : SON) dans lequel les pairs sont regroupés par les relations sémantiques des documents qu'ils stockent. Les réseaux de superposition sémantique ont été proposés comme moyen d'organiser les contenus dans les réseaux pair-à-pair. Chaque pair stocke des informations supplémentaires sur la classification des contenus et les requêtes d'itinéraire vers les SONs appropriés, ce qui augmente les chances que les objets correspondants soient trouvés rapidement et réduit la charge relative à la recherche. Il existe de nombreux défis lors de la création de SON, qui concernent notamment la façon dont les nœuds sont affectés aux SONs et à quels SONs une requête doit être envoyée. Les pairs devraient être uniformément répartis entre les SONs, afin de pouvoir répondre rapidement aux requêtes, le moins de pairs possible doit être interrogé, et chaque pair devrait appartenir à un petit nombre de SONs, de sorte que chaque pair ne doit gérer que quelques connexions. Cependant, dans le monde réel, la répartition des pairs sur les classes sémantiques est susceptible d'être peu fiable et dynamique, car de nombreux pairs appartiendront à des classes à sujets très populaires qui changent constamment, alors que certaines

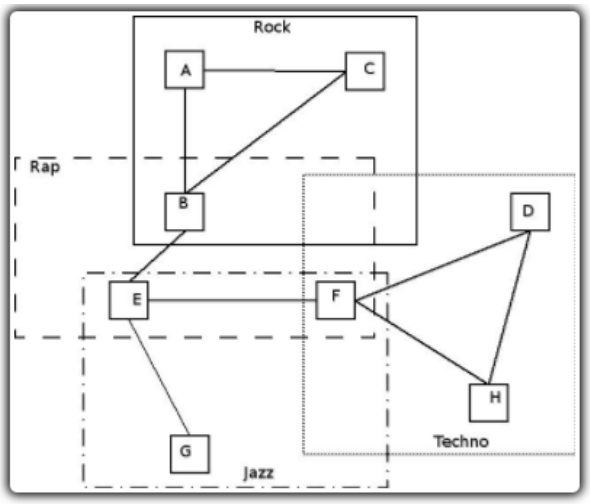


FIGURE 4: Exemple de couverture sémantique P2P avec 8 nœuds classés par genre de musique. Un nœud est relié à un autre par un lien logique. Les liens ayant le même genre de musique logique forment une superposition sémantique. Nous avons les superpositions suivantes : Rock, avec les nœuds A, B et C ; Rap, avec B, E, et F ; Jazz, avec E, F et G ; Techno, avec F, D et H . Source :[2]

classes peu communes seront moins peuplées. En outre, dans la plupart des premieres approches, les noeuds décident de quels SONs rejoindre en fonction de la classification de leurs documents. Cela conduit à une configuration fixe des réseaux de superposition sémantique, de sorte que la performance est très dépendante d'un bon choix de l'algorithme de classification et requiert un classifieur identique pour tous les pairs.
La classification des documents et des requêtes peut se faire automatiquement, manuellement ou par un processus hybride. Les exemples de classifieurs automatiques comprennent la correspondance de texte (text matching) [11], les réseaux bayésiens [97] et les algorithmes de clustering [134]. Ces techniques automatiques ont été largement étudiées. La classification manuelle peut être obtenue en obligeant les utilisateurs à marquer chaque requête avec le style ou le sous-type des résultats escomptés. Si l'utilisateur ne connaît pas le sous-type ou le style des résultats potentiels, il peut toujours sélectionner la racine de la hiérarchie afin que tous les nœuds soient interrogés. Enfin, les classifieurs hybrides renforcent la classification manuelle avec les bases de données.

7.3.2 Travaux connexes

Piazza [60] comprend une infrastructure et un langage d'alignement pour l'alignement sémantique et la gestion des données dans un environnement pair-à-pair. Le système prend en compte la structure du domaine et du document. Dans Piazza, les requêtes destinées à un pair peuvent être reformulées et adressées à des pairs sémantiquement liés. La fermeture transitive des translations entre pairs est utilisée pour répondre aux requêtes. La contribution de Piazza à la gestion des données est un développement important vers le passage à la gestion des données sémantiques distribuées par opposition à la pratique de l'intégration et de la méditation des données. Cependant, les pairs du système Piazza sont des entités statiques. Ils ne devraient pas quitter le réseau. Ainsi, Piazza ne dispose pas de la propriété ad hoc des réseaux ouverts P2P.

Il est également difficile pour les nouveaux pairs de rejoindre un réseau Piazza en raison de sa structure rigide.

Loser *et al.* [79] introduisent le concept de clusters de superposition sémantiques dans des réseaux à base de super-pairs. Les SOC (Semantic Overlay Cluster) sont conçus pour de très grands réseaux fortement distribués améliorant la recherche et l'interopérabilité sémantique. En particulier, la topologie super-pair, constituée d'un backbone super-pair avec des ordinateurs puissants et des clients plus petits qui sont liés à ces super-pairs, est très approprié pour cette approche. En outre, les auteurs ont montré quatre extensions sur un réseau super-pair existant, permettant un clustering dynamique de pairs fournisseurs d'informations à des clusters basés sur des super-pairs :

1. Des modèles basés sur RDF pour les pairs fournisseurs d'informations formulés en utilisant les connaissances issues des approches existantes de la communauté de base de données
2. Les politiques de clustering exprimant la demande sur les fournisseurs d'informations en fonction des langages de requête RDF existantes
3. Une distribution de modèles de concepts basés sur l'algorithme HyperCuP
4. Des approches de correspondances

Dans [127], Voulgaris *et al.* présentent une superposition sémantique reliant des nœuds sémantiquement proches. Cette superposition sémantique fournit le mécanisme de recherche principal, tandis que le système pair-à-pair initial fournit le mécanisme de recherche par failover (basculement). Les auteurs se focalisent sur les approches implicites pour découvrir la proximité sémantique.
Ils étudient des techniques implicites permettant de regrouper des nœuds ayant des intérêts similaires afin d'améliorer la recherche de contenu dans les systèmes de partage de fichiers. Chaque nœud, sur la base de l'historique des ses requêtes/réponses, crée une liste de voisins sémantiques vers lesquels ses requêtes sont envoyées avant d'utiliser comme failover l'algorithme de recherche standard.
Ils considèrent la même architecture que dans [120]. Chaque nœud maintient une liste de voisins sémantiques auxquels les requêtes sont soumises en premier, avant de basculer vers un mécanisme de recherche par défaut si aucun voisin sémantique ne peut répondre à la requête. Afin d'étudier l'efficacité de différentes stratégies pour le maintien de ces listes de voisins sémantiques, ils développent un modèle synthétique de structure sémantique reliant des nœuds et des documents dans le système.

L'idée de p2pDating [91]est de créer des réseaux de superposition sémantiques dans un environnement P2P, où un pair décide de maniére autonome quels SONs il souhaite rejoindre. L'approche fonctionne en obligeant les pairs à entrer en contact avec d'autres pairs dont ils ignorent encore l'existence. Si un pair "s'intéresse" à un autre pair, c'est-à-dire que si cet autre pair a des informations intéressantes pour le premier, il pourrait vouloir se souvenir de ce pair, et l'insérer dans la liste des amis. D'autre part, si le pair décide que l'autre interlocuteur n'est pas intéressant, il est probable qu'il pourrait ne pas vouloir se souvenir de ce pair, donc aucun lien n'est créé ou s'il existe déjà un lien entre eux, il pourrait être abandonné . Les auteurs considèrent que la mise en cache de pairs de haute qualité est la manière naturelle de créer des réseaux de superposition sémantiques. Il peut être considéré comme le critère pour créer des liens réseau entre pairs, ou être utilisé dans des systèmes peer-to-peer existants puisse qu'il n'est pas nécessaire de modifier l'infrastructure existante. Le processus commence par un réseau connecté de manière aléatoire et s'exécute indéfiniment puisque les pairs rejoignent et quittent continuellement le réseau. Les

réseaux de superposition sémantique évolueront dynamiquement à partir de ce processus, puisse que les liens sémantiques sont de plus en plus affinés. Dans un réseau P2P dynamique, les SONs changent continuellement pour s'adapter aux changements dans le réseau, au comportement des pairs, au contenu des pairs, etc. Les liens sémantiques sont représentés par des entrées dans la liste des amis. Lorsqu'un pair rejoint le réseau, sa liste d'amis est vide et sera rempli au fil du temps.

Himali *et al.* [54] proposent un schéma sémantique de clustering et de routage qui vise à améliorer la qualité et l'efficacité de la recherche dans les systèmes P2P. Les bases des clusters sémantiques sont des concepts d'une ontologie sémantique partagée déjà convenue. Cette ontologie sémantique est utilisée dans le but d'améliorer la recherche d'informations et de faciliter l'interopérabilité. Une nouvelle technique de construction de clusters dynamiques est proposée. Les pairs découvrent des voisins sémantiques en tenant compte des relations structurelles des concepts dans l'ontologie. Himali *et al.* proposent également un nouveau mécanisme de routage de requêtes qui exploite la hiérarchie conceptuelle pour acheminer rapidement une requête vers le cluster cible. La recherche de trafic et le gosssip sont utilisés par les pairs pour acquérir des informations globales par rapport à leurs intérêts.
Le schéma organise la structure de l'overlay basée sur des concepts sémantiques et leurs liens taxonomiques dans l'ontologie partagée. En plus de ces liens taxonomiques, un pair établit également d'autres types de liens pour assurer une localisation plus rapide des clusters cibles et une diffusion adéquate d'une requête à des pairs pertinents dans un cluster cible lors de l'interrogation. Le système utilise une combinaison de messages de recherche et de gossip pour découvrir des voisins sémantiquement pertinents.

7.4 Graphe réparti : cloud et fog computing

La gestion de gros volumes de données RDF est difficile, en raison de la taille, de l'hétérogénéité et de la complexité accrue induit par le raisonnement RDF. Pour s'attaquer aux défis liés à la taille, des architectures de stockage distribuées sont nécessaires. Le Cloud computing est un paradigme émergent adopté massivement dans de nombreuses applications pour les fonctionnalités d'évolutivité, de tolérance aux panne et d'élasticité qu'il fournit, permettant un déploiement facile d'architectures distribuées et parallèles.
Les chercheurs de la communauté Web sémantique se concentrent sur la résolution des problèmes d'évolutivité et de performance des outils Web sémantiques traditionnels en exploitant les technologies de cloud computing. L'avènement du Cloud Computing a ouvert la voie à un écosystème distribué de bases de triplets RDF qui peut potentiellement permettre un stockage à l'échelle de la planète ainsi que des capacités de traitement de requêtes réparties.
Les technologies du Cloud computing reçoivent une attention globale de l'industrie informatique et du milieu universitaire. Le paradigme MapReduce [30], qui est construit sur le système de fichiers de Google [50], est le paradigme de programmation parallèle et distribué dominant dans la communauté du cloud computing en raison de sa haute performance et sa capacité de tolérance aux pannes.
MapReduce est un modèle de programmation fonctionnel adapté au traitement de grande quantité de données en parallèle. C'est une technologie évolutive qui a été bien accueillie par la communauté. Google l'utilise pour l'indexation Web, le stockage de données, les réseaux sociaux. Apache implémente également MapReduce dans le framework open-source Hadoop [110], qui est appliqué avec succès pour résoudre des problèmes intensifs en matière de données dans différents domaines [136, 8]. Hadoop est un système de fichiers distribué où les fichiers peuvent être sauvegardés avec réplication. Il offre une haute tolérance aux pannes et une fiabilité élevée.

7.4.1 Cloud pour données distribuées (RDF)

Au cours des dernières années, le Cloud computing a fourni de nombreuses opportunités pour les entreprises en offrant à leurs clients une gamme de services informatiques. Le modèle «pay-as-you-go» de cloud computing actuel devient une alternative efficace à la possession et à la gestion de centres de données privés pour les clients confrontés aux applications Web et au traitement par lots [9]. Le Cloud computing libère les entreprises et leurs utilisateurs finaux de la spécification de nombreux détails, tels que les ressources de stockage, la limitation de calcul et le coût de la communication réseau.
Le cloud computing devient d'une part l'approche Internet générale pour le stockage, l'extraction et la gestion de l'information. Parallèlement les périphériques mobiles se présentent comme les principales applications de service. L'intégration réussie du cloud computing et des périphériques mobiles représente donc la tâche clé du réseau de prochaine génération. Cela pose cependant plusieurs défis fondamentaux : l'agilité des services, la réponse en temps réel, la connexion à long terme.
Pour relever ces défis entre les applications cloud et mobiles, le fog computing, également appelée edge computing, a récemment émergé comme une solution plus pratique pour permettre une convergence fluide entre le cloud et le mobile pour la diffusion de contenu et le traitement de données en temps réel [117].
Le Fog computing peut répondre à ces problèmes en fournissant des ressources et des services élastiques aux utilisateurs finaux au bord du réseau, tandis que le cloud computing vise plus à fournir des ressources réparties sur le réseau principal.

7.4.2 Fog pour données distribuées (RDF)

Le fog computing est un paradigme informatique distribué qui agit comme une couche intermédiaire entre les datacenters du Cloud et les périphériques/capteurs. Il offre des installations de calcul, de mise en réseau et de stockage afin que les services basés sur le Cloud soient plus proches des périphériques/capteurs. Le concept fog computing a d'abord été introduit par Cisco en 2012 pour relever les défis des applications IoT dans le cloud computing conventionnel.
Typiquement, un environnement de Fog computing est constitué de composants de réseau traditionnels (**Figure** 5). Exemple : les routeurs, les commutateurs, les décodeurs, les serveurs proxy, les stations de base (BS), etc., et peuvent être placés à proximité des périphériques/capteurs. Ces composants sont dotés de diverses fonctionnalités de calcul, de stockage, de réseautage, etc., et peuvent supporter l'exécution des applications de service. Par conséquent, les composants fonctionnels permettent au fog computing de créer de grandes distributions géographiques de services basés sur le cloud. De plus, le Fog computing facilite la prise en compte de l'emplacement, le support de la mobilité, les interactions en temps réel, l'évolutivité et l'interopérabilité. Ainsi, il peut fonctionner efficacement en termes de latence de service, consommation d'énergie, trafic réseau, dépenses de capital et d'exploitation, distribution de contenu, etc. En ce sens, le Fog computing répond mieux aux exigences relatives aux applications IoT par rapport à l'utilisation unique du cloud computing [102].
Il existe des concepts similaires tels que le mobile cloud computing (MCC) et le mobile-edge computing (MEC) qui se chevauchent avec le fog computing. MCC se réfère à une infrastructure dans laquelle le stockage et le traitement des données se produisent en dehors des appareils mobiles. Les applications mobiles basées sur le cloud déplacent la puissance de calcul et de stockage de données des téléphones mobiles vers le cloud, fournissant des applications et des traitements mobiles non seulement aux utilisateurs de smartphones, mais aussi à une gamme beaucoup plus large d'abonnés mobiles [34]. MEC peut être considéré comme un serveur cloud fonctionnant au bord d'un réseau mobile et effectuant des tâches spécifiques qui ne peuvent être réalisées avec

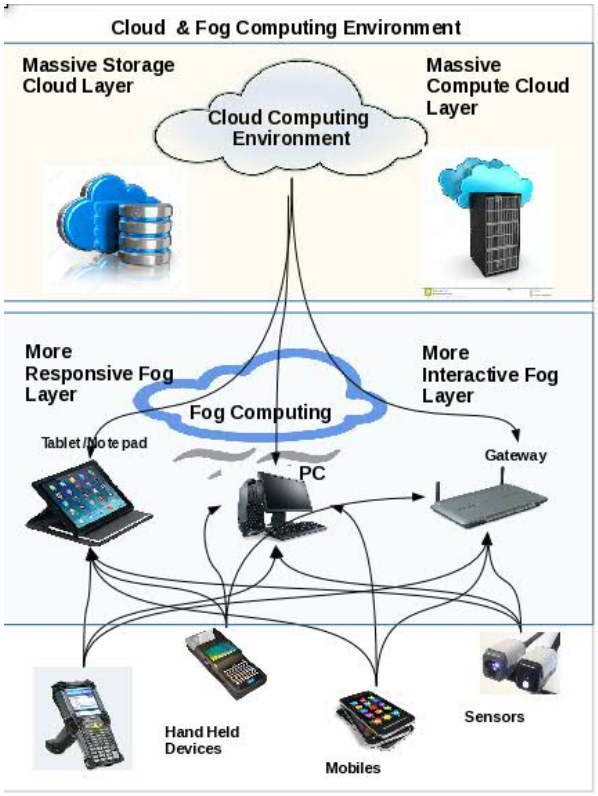


FIGURE 5: Exemple d'architecture Fog [1]

l'infrastructure réseau traditionnelle [39]. Le fog computing semble être la combinaison de MCC et de MEC, alors qu'il se distingue comme un paradigme informatique plus prometteur et bien généralisé dans le contexte de l'internet des objets (IoT).

7.4.3 Travaux connexes

Cloudlet [104] est considéré comme une implémentation exemplaire de nœuds Fog riches en ressources. Son architecture est constituée de trois couches. La couche inférieure est constituée du noyau Linux et du cache de données provenant du cloud, la couche intermédiaire est la virtualisation avec un ensemble de logiciels cloud tels que OpenStack [27], et la couche supérieure est constituée d'applications isolées par des instances différentes de machine virtuelle (VM). Les Cloudlets ont été spécifiquement conçus pour fournir des services aux utilisateurs mobiles avec des ressources locales limitées qui peuvent agir comme un client léger et accéder aux ressources cloudlet qui sont à un saut via un réseau sans fil. Satyanarayanan *et al.* affirment que les périphériques mobiles resteront toujours limités en ressources par rapport aux appareils stationnaires tels que les ordinateurs de bureau, les ordinateurs portables et les serveurs en raison de leurs autres exigences, telles ques la taille réduite, le poids faible et la durée de vie plus longue de la batterie. Mais au contraire, les applications et les paradigmes émergents tels que les médias interactifs, la réalité augmentée, le traitement du langage naturel, la reconnaissance de la parole, etc. nécessitent une grande quantité de ressources pour le traitement avec des latences minimales. Par conséquent, la localisation des ressources de grande capacité le plus près possible de l'uti-

lisateur final est la seule solution. Cloudlet se positionne ainsi efficacement pour gérer ce type d'exigences et de demandes.

Zhu *et al.* [140] appliquent les méthodes existantes pour l'optimisation Web de manière innovante. Dans le contexte du Fog computing, ces méthodes peuvent être combinées avec des connaissances uniques qui ne sont disponibles que sur les appareils Fog. Une adaptation plus dynamique aux conditions de l'utilisateur peut également être réalisée avec des connaissances spécifiques au bord du réseau. En conséquence, les performances de rendu de la page Web d'un utilisateur sont améliorées au-delà de celles obtenues en appliquant simplement ces méthodes sur le serveur Web.

Lee *et al.* [76] présentent une approche de partitionnement de hash sémantique qui combine un partitionnement de graphe RDF optimisé par la localisation avec un partitionnement de requête à coût réduit pour étendre les requêtes sur de grands graphes RDF. La méthode de partitionnement de hash sémantique utilise la localisation des accès pour partitionner des graphes RDF de grandes tailles sur plusieurs nœuds de calcul en maximisant la capacité de traitement intra-partition et en minimisant le coût de communication entre les partitions. Lee *et al.* génèrent ensuite des plans d'exécutions de requêtes optimisés par la localisation qui sont plus efficace que les systèmes populaires de gestion de données RDF multi-nœuds, en minimisant efficacement le coût de communication inter-machine pour le traitement des requêtes. Ils fournissent également un ensemble de techniques d'optimisation tenant compte de la localisation pour réduire davantage la taille de la partition et le coût de communication inter-machine lors du traitement des requêtes.

Hong *et al.* [55] présentent un modèle de programmation PaaS (Platform as a Service), appelé Mobile Fog, qui fournit une abstraction de programmation simplifiée et prend en charge les applications dynamiquement mise à l'échelle à l'exécution. Dans ce modèle, une application se compose de processus distribués de Fog mobiles qui sont allignés sur des instances de calcul réparties sur le fog et le cloud, ainsi que divers dispositifs aux bords du réseaux. À l'éxécution chaque processus effectue des tâches spécifiques à l'application telles que la détection et l'agrégation par rapport à son emplacement et son niveau dans la hiérarchie du réseau. Chaque processus Mobile Fog gère la charge de travail d'une certaine région géospatiale. Mobile Fog est constitué d'un ensemble de gestionnaires d'événements que l'application doit implémenté et un ensemble de fonctions que l'application peut appeler. Les gestionnaires d'événements sont appelés par le système d'exécution de Mobile Fog sur certains événements.

Nishio *et al.* [87] proposent un framework pour le partage hétérogène des ressources avec le fog computing en procédant à un alignement des ressources hétérogènes telles que les CPU, la bande passante de communication et les ressources à stockage continu. Les nœuds mobiles voisins sont connectés à d'autres nœuds et forment un réseau local de partage de ressources en utilisant des connexions sans fil à courte portée telles que le WiFi en mode ad-hoc et/ou le Bluetooth. Les messages de partage de ressources tels que les demandes de ressources, les instructions de tâche et les résultats des tâches sont transmises via le réseau local. Certains des nœuds ont un accès Internet sans fil en utilisant éventuellement la 3G ou le WiFi. Les nœuds voisins dans un réseau local forment un groupe appelé un cloud local. Ils partagent leurs ressources avec d'autres nœuds du même cloud local. Un coordinateur de ressources local (LRC) est choisi parmi les nœuds de chaque cloud local. Le coordinateur gère les demandes de ressources et alloue des tâches aux nœuds du cloud local ou du data-center distant (sur Internet) si nécessaire.

Willis *et al.* [133] introduisent un framework spécifique de edge computing, appelé ParaDrop, qui permet aux développeurs de tirer parti de l'un des derniers réserves de ressources persistantes

proches du client final : la passerelle (exemple :le point d'accès WiFi ou le décodeur domestique). ParaDrop est implémenté sur une passerelle, ce qui est un choix de noeud de fog idéal en raison de sa proximité avec l'utilisateur final. Avec ce framework, les développeurs peuvent concevoir des conteneurs de calcul virtuellement isolés pour assurer une présence computationnelle persistante à proximité des utilisateurs. Cependant, il est conçu pour les scénarios d'utilisation domestique et n'est pas entièrement décentralisé, tous les serveurs d'applications ont besoin d'utiliser un serveur ParaDrop comme point d'entrée aux services fournis par les passerelles. ParaDrop est considéré comme une implémentation complémentaire du Fog computing pour les scénarios de tâches légères.

DiploCloud [135] est un système de gestion de données RDF distribué efficace et évolutif pour le cloud. Contrairement aux approches antérieures, DiploCloud effectue une analyse physiologique des informations d'instance et de schéma avant de partitionner les données. Il utilise un format de stockage résolument non relationnel, où les modèles de données sémantiquement liés sont extraits à la fois au niveau de l'instance et au niveau du schéma de données et sont co-localisés pour minimiser les opérations inter-nœuds.
Le système est construit sur trois structures principales : les clusters de molécules RDF (qui peuvent être considérées comme des structures hybrides empruntant à la fois des tables de propriétés et des sous-graphes RDF), des listes de modèles (stockage de littéraux dans des listes compactes comme dans un système de base de données orienté colonne) et un index de clé efficace indexant les URIs et les littéraux basés sur les clusters auxquels ils appartiennent.
La conception du système suit l'architecture de nombreux systèmes distribués modernes basés sur le cloud (par exemple, BigTable [24] de Google), où un nœud principal est responsable d'interagir avec les clients et d'orchestrer des opérations effectuées par les autres nœuds (Worker).

Faruqueet *et al.* [7] introduisent le Fog computing comme une plateforme novatrice pour la gestion de l'énergie. L'évolutivité, l'adaptabilité et les logiciels/matériels open source présentés dans la plateforme proposée permettent à l'utilisateur de mettre en œuvre la gestion de l'énergie avec le contrôle-de-services personnalisé, tout en minimisant le coût de mise en œuvre et le délai de commercialisation.
La plateforme proposée utilise l'interopérabilité, l'évolutivité, l'adaptabilité et la connectivité entre les plateformes intelligentes sur la plateforme de fog computing qui consiste en un dispositif à faible consommation et à faible coût pour le calcul, le stockage et la communication. L'architecture logicielle et l'infrastructure matérielle Open Source intégrées à une plateforme fog computing permettent d'étendre la plateforme (évolutivité). Le logiciel de gestion ou contrôle de l'énergie est mis en œuvre en tant que service en utilisant les profils des périphériques pour les services Web (Devices Profile for Web Services : DPWS) qui est également utilisé pour la découverte pour fournir la fonction de plug-n-play. Cette architecture orientée service résume également l'hétérogénéité des communications et du matériel.

Compte tenu d'un nombre énorme de véhicules dans les zones urbaines, mettre les ressources de véhicules sous-utilisées en service offre une excellente opportunité et de la valeur. Hou *et al.* [56] conçoivent ainsi l'idée d'utiliser les véhicules comme les infrastructures de communication et de calcul, nommés le fog computing de véhicule (vehicular fog computing VFC), qui est une architecture qui utilise une multitude de clients/utilisateurs finaux collaborative ou des dispositif de bord(réseau) proche de l'utilisateur pour effectuer des communications et des calculs, en fonction d'une meilleure utilisation des ressources individuelles de communication et de calcul de chaque véhicule. En agrégeant les ressources abondantes des véhicules individuels, la qualité des services et des applications peut être améliorée considérablement.

Les auteurs présentent un aperçu des capacités potentielles et des problèmes ouverts pour le nouveau paradigme VFC. Spécifiquement, ils considèrent quatre scénarios, à savoir l'utilisation de véhicules en mouvement et en stationnement comme infrastructure de communication et de calcul, respectivement. D'une part, les véhicules en mouvement et en stationnement utilisés comme infrastructures de communication peuvent supporter une plus grande capacité de flux en raison de l'amélioration du retard de livraison des paquets et des connexions améliorées entre les véhicules. Des modèles de déplacement prévisibles peuvent être utilisés à travers les véhicules pour obtenir une meilleure communication. D'autre part, les véhicules en mouvement et en stationnement utilisés comme infrastructures de calcul (de traitement) peuvent permettre une utilisation optimale des ressources informatiques de chaque véhicule.

7.5 Synthèse

Le tableau suivant présente une synthèse des différentes solutions présentées dans cette partie. Chaque ligne représente une caractéristique nous permettant de comparer ces différents modèles :

- **Type de donnée** : Types des données échangées
- **Réplique partielle** : Le système adopte un mécanisme de réplication partielle sur le graphe.
- **Réplique totale** : Le système adopte un mécanisme de réplication totale sur le graphe.
- **DHT** : L'architecture repose sur du DHT
- **Superposition Sémantique** : Le système construit une superposition sémantique sur les nœuds.
- **Couche Fog** : L'architecture comporte une couche Fog
- **Caching** : Le système offre un service de mise en cache.

	freenet [2000]	edutella [2002]	piazza [2003]	SOC [2003]	rdfpeers [2004]	Voulgaris [2004]	p2p-dating [2007]	swooki [2008]	cloudlet [2009]	logoot-undo [2010]	crate [2016]	diplo-cloud [2016]
Type de donnée	fichier	rdf	rdf	rdf	rdf	rdf	rdf	rdf	fichier	fichier	fichier	rdf
DHT	√	√			√		√			√		
Réplique partielle	√				√		√			√		√
Semantic overlay			√	√		√	√					
Réplique totale								√		√	√	
Caching	√					√	√					
Couche Fog									√			√

TABLE 8: Tableau de synthèse des solutions traitant du partage et de la modification collaborative de graphe RDF.
La prise en charge d'une caractéristique est indiquée par le signe √ , la non prise en charge par une case vide.

8 Conclusion et Perspectives

8.1 Synthèse

Dans le tableau suivant, nous proposons une synthèse des différentes approches importantes retenues tout au long de cet état de l'art. Pour un système adoptant ces approches, nous indiquons s'il prend en compte ou non les différents éléments qui suivent.

- **Données RDF** : l'approche est applicable à des données RDF
- **Mobilité** : l'approche permet de prendre en compte la mobilité des nœuds
- **Accès local** : l'approche permet un accès local aux données
- **Résistance aux échecs des nœuds** : l'approche assure à l'architecture une forte résistance aux échecs des nœuds (tolérance aux pannes)
- **Haute Disponibilité** : l'approche assure une haute disponibilité des données
- **Scalabilité** : l'approche permet à l'architecture de passer efficacement à l'échelle
- **Besoin en ressource matérielle** : le niveau d'exigence matérielle (faible, moyenne ou forte)
- **Mise à Jour (MAJ)** : le mécanisme de mise à jour adopté par l'approche (simple ou complexe)
- **Latence faible** : le temps de latence lors de scénarios de recherche de données est considéré comme faible (assurée ou non par l'approche)

	Géolocalisation	Cache coopératif	Protocol de gossip	Répartition de graphe	Réplication de graphe
Données RDF	√	√	√	√	√
Mobilité	√	√	√	√	√
Accès local		√		√	√
Résistance aux échecs		√	√	√	√
Scalablilité			√	√	√
Haute Disponibilité		√			√
Latence faible	Assurée	Assurée	Non Assurée	Assurée	Assurée
MAJ		Complexe	Simple	Complexe	Complexe
Internet	obligatoire	conditionnel	facultatif	conditionnel	facultatif

TABLE 9: Tableau de synthèse des différentes approches vues dans cet état de l'art. La prise en charge totale d'une caractéristique est indiquée par le signe √, la non prise en charge par une case vide

8.2 Méthodes d'évaluation

L'interrogation **Q7** citée dans le chapitre I concerne les méthodes d'évaluation utilisées dans le domaine. Nous nous sommes intéressés précisément aux techniques d'évaluation des protocoles de gossip et des mécanismes de réplication locale et intelligente des données. Les protocoles de gossip sont généralement évalués en fonction des paramètres comme la taille de la vue, le fanout, les degrés entrant et sortant des nœuds. Ces derniers permettent d'analyser les propriétés des protocoles et le comportement de l'architecture lors des tests. Des plateformes de simulation telles que PeerSim [81] sont utilisées pour comparer les différents protocoles. Nous avons aussi vu les méthodes dédiées aux mécanismes de réplications. Ces évaluations sont généralement effectuées

en analysant et comparant les performances des solutions proposées en fonction de métriques telles que la complexité en temps et en espace, la scalabilité, l'effet des flux sur l'architecture réseaux.
Nous listons ici plusieurs des métriques les plus fréquents pour ces évaluations.

- **La résilience aux échecs de nœud (tolérance aux pannes)** : cette dimension fait référence à l'impact des pannes massives sur la fiabilité des protocoles de gossip. Elle constitue l'une des propriétés les plus intéressantes de ces protocoles.
- **L'impact sur le réseau en termes de stress de lien** : le principal avantage attendu du protocole hiérarchique est une diminution de la charge sur les liens de routage de base. Pour évaluer cette diminution, les charges sur chaque lien physique sont considérées.
- **Les propriétés des graphes** : les superpositions produites par les protocoles d'adhésion doivent présenter de bonnes propriétés telles que le faible coefficient de clustering, la moyenne des chemins les plus courts et la répartition équilibrée. Les protocoles de gossip peuvent être évalué par rapport à la variation de ces paramètres.
- **La charge de requête (Query load)** : elle est relative au nombre de requêtes reçues par chaque nœud de l'architecture. Elle est utilisée pour évaluer les applications dédiées aux systèmes pair-à-pair en comparant les distributions des charges à travers les nœuds.
- **Le nombre de cycles** : cette dimension se réfère au nombre de cycles de communication nécessaire pour que tous les nœuds participants à une stratégie de CRDT convergent vers l'état final. Un cycle de communication est une séquence de production-envoi-réception-exécution d'opérations entre les nœuds.
- **Les coûts généraux** : cette dimension se réfère aux frais généraux en temps d'exécution et en espace disque utilisé pour des stratégies de réplication de types de données.
- **Exhaustivité, Concision, cohérence (métrique de qualité des données)** : ces trois métriques de qualité de données sont utilisées pour évaluer l'utilité de l'approche de synchronisation. L'exhaustivité se réfère au degrés auquel toutes les informations requises sont présentes dans un ensemble de données. La cohérence assure que les valeurs ne doivent pas être contradictoires. La concision mesure le degré auquel l'ensemble de données ne contient pas d'informations redondantes.

8.3 Futurs travaux

L'interrogation **Q8** est dédiée aux futures directions de recherche les plus fréquemment annoncées par les documents que nous avons traités. Nous nous sommes focalisés sur les travaux relatifs à des architectures reposant sur des protocoles de gossip mais aussi aux techniques visant à permettre un accès local à des données notamment la réplication intelligente de sources. Dans la suite, nous présentons quelques un de ces futures directions de recherche.

- **Le paramètre géographique** : prendre en compte la localité géographique dans la génération des vues partielles.
- **Le fanout adaptatif** : ceci concerne la prise en compte de l'hétérogénéité des nœuds pour maximiser l'utilisation des ressources disponibles telles que la bande passante.
- **Mise en œuvre en javascript** : une implémentation dans un navigateur permet de réaliser des émulations, et des applications pair-à-pair distribuées et décentralisées.
- **L'exploitation des avantages d'un réseau social d'amis explicites** : prendre en compte ce type de lien comme une base de connaissance pour former le réseau personnalisé d'un utilisateur et ajouter automatiquement de nouvelles connaissances sémantiques implicites.
- **Se passer de la causalité** : étudier si un CRDT de store sémantique peut être construit sans qu'il soit nécessaire que l'architecture fournisse une causalité. Ce qui allégerait les

contraintes liées à la mise en place de CRDT.
- **Synchronisation** : l'ajout de mécanismes de synchronisation pour prendre en charge les opérations non fréquentes et non critiques d'un nœud, telles que la validation d'un état ou l'application d'une ré-initialisation globale, lors des processus de réplications.

8.4 Discussion

Dans cette partie nous revenons sur les différentes questions relatives au domaine que nous nous étions posées dans le chapitre I.

Q1 vise à identifier les travaux pertinents dédiés à la conception et la mise en place d'architectures reposant sur des protocoles de gossip. Nous avons traité dans le chapitre II plusieurs de ces architectures proposées à travers des papiers de références. Nous avons distingué deux types d'architectures. Le premier appelé ici protocole de base fait référence à des systèmes basiques dédiés à la conception et la maintenance de l'architecture sous-jacente sur laquelle repose les différents échanges entre les pairs membres. Le deuxième type d'architecture est dédié à la formation automatique de cluster d'intérêt ou de proximité visant à regrouper les pairs membres de l'architecture qui ont un ou plusieurs centres d'intérêts en commun ou qui sont proches selon une métrique de similarité donnée. Ces systèmes hiérarchiques sont conçus au-dessus des protocoles de base.

Q2 fait référence aux enjeux soulevés par l'utilisation des protocoles de gossip dans les systèmes de communication en pair-à-pair. Ce type de protocole est de plus en plus utilisé dans les systèmes à grande échelle particulièrement du fait de son évolutivité, sa fiabilité et surtout sa résistance aux défaillances des réseaux et des nœuds. Plus précisément, grâce à un niveau de redondance élevé des données dans le système, les défaillances des réseaux et des nœuds sont masqués. En plus, la charge est distribuée à travers tous les nœuds de l'architecture, ce qui permet d'atteindre un niveau d'équilibrage des charges acceptable sur l'architecture. Ce type de systèmes est aussi répandu du fait de sa facilité de mise en œuvre et de déploiement. Il a ainsi été appliqué avec succès dans les systèmes à grande échelle tels que le calcul d'agrégat, et la gestion des réseaux.

Q3 s'interroge sur les mécanismes d'échange de données sémantiques reposant sur des protocoles de gossip. Ce sont des mécanismes qui adoptent en générale des architectures hiérarchiques qui sont en principe conçus au-dessus des protocoles de gossip de base. La hiérarchie est formée par des clusters construits de manière automatique par le système. Chaque cluster est constitué de pairs similaires selon une métrique de proximité sémantique. Ainsi, au niveau de chaque cluster, les pairs membres peuvent échanger des données en interrogeant leurs voisins. Les données sémantiques peuvent aussi transiter d'un cluster à un autre selon les requêtes des pairs.

Q4 traite de la prise en compte de la localisation dans les échanges de données sémantiques. Certaines des solutions proposées dans la littérature permettent de prendre en compte la localisation des pairs pour la construction des clusters. Ces derniers seront alors formés de pairs sémantiquement proches mais aussi qui ont une certaine proximité géographique, c'est-à-dire qu'une limite maximale en terme de distance géographique est fixée entre les pairs membres d'un cluster. Un autre moyen d'utiliser la localité géographique est d'appliquer une sélection déterministe de pair lors de l'initialisation d'un échange en se basant sur la proximité géographique. Certains systèmes se réfèrent aussi à la localisation pour réduire la distance géographique entre les pairs lors des processus d'échange de données. La localisation géographique est alors utilisée

lors du choix des noeuds de routage intermédiaires.

Q5 regarde les solutions proposées dans la littérature pour fournir un accès local à des données. Nous nous sommes focalisé dans les chapitre VI et VII sur les mécanismes d'accès à des sources de données dans des environnements ou l'accès à internet se fait d'une manière intermittente. Les solutions existantes reposent en générale sur des systèmes de réplication intelligente des sources de données en ligne. Ces derniers sont boostés par des mécanismes de mise en cache local, d'indexation des sources et aussi de synchronisation. Ce dernier élément soulève la problématique de la pertinence des données durant l'exécution du système, notamment de la divergence des états entre les sources distantes et locales lorsque toute les opérations de synchronisation sont exécutées. Pour pallier cette contrainte, certaines solutions adoptent le modèle CCI (Convergence, Causality Preservation and Intention preservation) pour assurer la cohérence du système. Certaines solutions reposent aussi sur des mécanismes de mise en cache de données pertinentes selon les métas donnés des requêtes des pairs. Des stratégies de remplacement (ou de suppression) identifient les données à déplacer vers un stockage persistant ou à supprimer entièrement pour procéder à une bonne gestion de la mémoire cache.

Q6 fait référence aux systèmes de caching en pair-à-pair dédiés à améliorer les processus d'accès et d'interrogation des sources de données. Nous nous sommes intéressés particulièrement aux systèmes dédiés à des architectures pair-à-pair hiérarchiques distribuées. Ces dernières étant formées de clusters constitués de pairs relativement proches selon une métrique, des caches coopératifs sont construits au niveau de chaque cluster. Chaque cache coopératif est composé de l'ensemble des caches locaux des différents pairs membres du cluster. Les requêtes sont d'abord traitées au niveau du cache local de chaque pair. En cas d'échec, le cache coopératif est interrogé en premier avant de se référer éventuellement aux sources de données en lignes. Ce procédé permet d'accéder localement aux données fréquemment interrogées par les pairs pour pouvoir pallier dans certains cas l'indisponibilité de l'accès à internet. Il permet aussi d'accélérer l'accès aux données puisque le processus de lecture est faite au niveau des mémoires caches des dispositifs. Grâce aussi à la puissance de WebRTC, les mémoires caches des navigateurs sont partagées entre les pairs lors des processus d'échanges au niveau des architectures hiérarchiques reposant sur des protocoles de gossip.

Q7 se penche sur les techniques adoptées pour l'évaluation des protocoles de gossip et des solutions dédiées à l'accès local à des sources de données notamment les mécanismes de réplication locale. Se basant sur des paramètres tels que le fanout, la taille de la vue, le degré de nœuds, les protocoles de gossip sont évalués généralement en comparant les différents comportements de l'architecture lors de l'exécution. Les propriétés du protocole telles que la résistance aux défaillances, le temps de convergence, la distribution des charges, les propriétés du graphe (coefficient de clustering, moyenne des chemin les plus courts, la répartition équilibrée), sont analysées et interprétées après des séances de tests sur des plateformes de simulations tels que PeerSim [81]. Nous avons aussi vu les méthodes dédiées aux mécanismes de réplications. Grâce à des métriques telles que la complexité en temps et en espace, l'effet des flux sur l'architecture réseaux, les solutions proposées sont comparées en analysant leurs performances par rapport à des propriétés qui leurs sont propres. Ces derniers concernent notamment les métriques de qualité des données (exhaustivité, concision, cohérence), le nombre de cycle, la charge de requête (query load), la scalabilité.

Q8 traite des directions les plus pertinentes pour les travaux futurs par rapport aux différentes solutions que nous avons vues. Dans la section précédente nous nous sommes intéressés

aux directions de recherche les plus pertinentes par rapport aux travaux dédiés aux systèmes d'échange de données reposant sur des protocoles de gossip et aux mécanismes de réplication, vus dans les chapitres 2 et 6. Concernant les protocoles de gossip, les directions que nous avons identifiées sont les plus fréquemment annoncées dans les documents que nous avons traités. Nous avons entre autres : la prise en compte du paramètre géographique, l'intégration du fanout adaptatif, l'implémentation des solutions en JavaScript et l'exploitation des avantages des réseaux sociaux d'amis explicites. Ces différents points ont été définis dans la section précédente. Par rapport aux mécanismes de réplications, nous avons aussi distingué des points intéressants pour les travaux futurs. Ils se résument principalement en deux directives. La première est relative à la problématique de la synchronisation entre les sources locales et distantes. Elle projette la mise en place de mécanismes pertinents pouvant améliorer la cohérence et la fiabilité des données échangées. La deuxième direction vise à alléger les contraintes liées à la mise en œuvre de CRDT en se passant de l'exigence que le réseau sous-jacent assure la propriété de causalité.

8.5 Perspectives

Dans cette partie nous concluons en identifiant dans l'état de l'art que nous venons de donner les directions intéressantes pour proposer un accès mobile et restreint à des données liées partagées en pair-à-pair.

Nous avons vu dans les chapitres précédents plusieurs solutions qui traitent de problèmes que nous pourrons considérer pour construire notre système. Bien que ces solutions soient dans certains cas spécifiques à des scénarios et environnements différents de la nôtre, nous pouvons nous en inspirer pour proposer des alternatives pertinentes et efficaces. Particulièrement, pour notre scénario qui consiste, pour faire simple, à « faire du web sémantique sans Internet », plusieurs de ces solutions reposant sur des architectures pair-à-pair apportent des points importants pour contourner le besoin permanent de connexion internet. Nous proposons dans la suite un ensemble de mécanismes pouvant nous conduire à un accès mobile et restreint à des données liées et partagées en pair-à-pair.

Nous nous intéressons d'abord à l'architecture pair-à-pair devant supporter notre solution. Une architecture pair-à-pair hybride se présente comme une bonne option dans le sens où la scalabilité du système est un aspect important dans notre cas. En effet, nous considérons dans notre scénario, l'existence de nœuds particuliers dotés de caractéristiques plus importantes par rapport aux autres : mémoire de stockage, processeurs, RAM, accès internet. Ces nœuds, disposés par exemple sur des édifices publics comme la mairie, l'université ou le stade, pourront jouer le rôle de super-pairs dans notre architecture. Ainsi chaque nouveau pair désirant joindre l'architecture, sera intégré dans l'un des sous-ensembles connectés à un des super-pairs. Les super-pairs pouvant faire l'objet de déconnexion ou de crash, nous supposons qu'ils peuvent joindre et quitter l'architecture à tout moment. Cependant ceci ne devrait pas conduire à un arrêt total des processus d'accès en lecture et écriture au niveau local, c'est-à-dire au niveau des nœuds connectés à un super-pair en échec (momentanément déconnecté de l'architecture).
Pour un nœud désirant joindre l'architecture, nous supposerons que le processus d'intégration se fera selon sa situation géographique. Plus précisément, nous considérerons que le nouveau nœud sera intégré à l'ensemble de pairs connectés au super-pair le plus proche par rapport à sa situation géographique. Ceci requiert notamment que chaque nœud intègre ses coordonnées géographiques dans son profil.

Nous construirons notre architecture sur un protocole de gossip à mécanisme d'adhésion à deux overlays. Comme nous l'avons vu dans le deuxième chapitre de ce document, ces protocoles ont la particularité d'être auto-organisés, résistants aux échecs et surtout adaptés aux environnements dynamiques où les nœuds joignent et quittent l'architecture sans aucune condition. Nous nous intéresserons particulièrement à un protocole hiérarchique sans serveur de contact. Bien que les super-pairs de l'architecture peuvent aussi jouer le rôle d'un serveur de contact, leur caractère dynamique peut conduire à leur indisponibilité et donc à un blocage du processus d'intégration d'un nouveau nœud. De ce fait, nous considérerons que toute requête d'intégration d'un nœud désirant intégrer l'architecture pourra être traitée par n'importe quel autre nœud.
Le caractère hiérarchique du protocole de gossip sous-jacent nous permettra d'appliquer un mécanisme de cluster sur les nœuds créant alors des liens virtuels entre les pairs selon des critères de proximité que nous définirons. Nous voulons par ce mécanisme, aboutir à une architecture où à chaque super-pair, seront connectés des pairs organisés en de petits clusters représentant des groupes d'intérêts communs. Nous supposerons également l'existence de liens inter-cluster et intra-cluster pour assurer une architecture localement connectée. Une fois qu'un nœud aura initialisé sa vue avec les données reçues d'un nœud de contact aléatoire, l'exécution cyclique du protocole de gossip lui permettra d'améliorer sa vue qui sera au fur et à mesure composée d'informations sur des nœuds avec des points d'intérêts similaires formant ensemble un ou plusieurs clusters.

Vu que la connexion internet est peu fiable dans notre environnement, un mécanisme d'accès local aux données doit aussi être mis en place. Nous considérons précisément que l'ensemble des nœuds super-pair de l'architecture procède à une réplication locale et intelligente des données pertinentes depuis les sources en lignes lorsqu'ils sont connectés à internet. Ils formeront alors une source de données locale distribuée interrogée avant les sources en lignes lors des traitements de requêtes. Les données pourront être modifiées en locale et nous mettrons en place un processus de mise à jour en temps réel pour garder une bonne cohérence entre sources locales et distantes mais aussi entre sources locales de différents super-pairs. Lorsqu'aucun processus de mise à jour entre les super-pairs n'est en cour, les différentes sources locales seront supposées identiques. De ce fait, aucun scénario de routage de requêtes vers des super-pairs voisins ne peut se produire.
Nous supposerons aussi que dans chaque cluster, les données relatives au point d'intérêt considéré, sont répliquées au niveau du super-pair.

Pour pallier à des scénarios de crash des super-pairs et en même temps réduire leurs charges de traitement de requêtes, nous mettrons en place un mécanisme de cache intelligent des données les plus interrogées par les pairs composant l'architecture. Un cache coopératif local constitué des caches locaux des différents pairs est construit de manière automatique au niveau de chaque cluster. Grâce notamment à l'existence de liens inter-clusters, les requêtes pourront être routées vers les caches locaux des différents clusters. L'ensemble de ces caches locaux formera ainsi un cache coopératif commun qui pourra être interrogé par les pairs avant que les requêtes ne soient transmises au super-pair.
Nous nous focaliserons particulièrement sur les caches des navigateurs (ou le Web Strorage) pour le stockage des données au niveau des noeuds.

Références

[1] Arunachalam Jayaraman. Fog Computing ! Is it real ? `https://www.linkedin.com/pulse/fog-computing-real-hoax-arunachalam-jayaraman/`. 2015-12-28.

[2] Fabrice Schuler. Étude et utilisation des technologies des P2P. `http://schuler.developpez.com/articles/p2p/images/son.jpg`. 2005-02-28.

[3] Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt. P-grid : a self-organizing structured p2p system. *ACM SIGMOD Record*, 32(3) :29–33, 2003.

[4] Karl Aberer, Philippe Cudré-Mauroux, Manfred Hauswirth, and Tim Van Pelt. Gridvine : Building internet-scale semantic overlay networks. In *International semantic web conference*, volume 3298, pages 107–121. Springer, 2004.

[5] Philippe Adjiman, Philippe Chatalic, François Goasdoué, M-C Rousset, and Laurent Simon. Somewhere in the semantic web. In *International workshop on principles and practice of semantic web reasoning*, pages 1–16. Springer, 2005.

[6] Philippe Adjiman, François Goasdoué, and Marie-Christine Rousset. Somerdfs in the semantic web. In *Journal on data semantics VIII*, pages 158–181. Springer, 2007.

[7] Mohammad Abdullah Al Faruque and Korosh Vatanparvar. Energy management-as-a-service over fog computing platform. *IEEE internet of things journal*, 3(2) :161–169, 2016.

[8] Nasullah Khalid Alham, Maozhen Li, Yang Liu, and Suhel Hammoud. A mapreduce-based distributed svm algorithm for automatic image annotation. *Computers & Mathematics with Applications*, 62(7) :2801–2811, 2011.

[9] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4) :50–58, 2010.

[10] Manuel Atencia, Jérôme Euzenat, Giuseppe Pirro, and Marie-Christine Rousset. Alignment-based trust for resource finding in semantic p2p networks. In *International Semantic Web Conference*, pages 51–66. Springer, 2011.

[11] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[12] Philip A Bernstein, Vassos Hadzilacos, and Nathan Goodman. *RRENCY CONTROL AND RECOVERY IN DATABASE SYSTEMS*. Addison- Wesley, 1987.

[13] Marin Bertier, Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, and Vincent Leroy. The gossple anonymous social network. In *Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware*, pages 191–211. Springer-Verlag, 2010.

[14] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7) :422–426, 1970.

[15] Edward Bortnikov, Maxim Gurevich, Idit Keidar, Gabriel Kliot, and Alexander Shraer. Brahms : Byzantine resilient random membership sampling. *Computer Networks*, 53(13) :2340–2359, 2009.

[16] Antoine Boutet, Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, and Rhicheek Patra. Hyrec : Leveraging browsers for scalable recommenders. In *Proceedings of the 15th International Middleware Conference*, pages 85–96. ACM, 2014.

[17] Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3) :630–659, 2000.

[18] Min Cai and Martin Frank. Rdfpeers : a scalable distributed rdf repository based on a structured peer-to-peer network. In *Proceedings of the 13th international conference on World Wide Web*, pages 650–657. ACM, 2004.

[19] Min Cai, Martin Frank, Jinbo Chen, and Pedro Szekely. Maan : A multi-attribute addressable network for grid information services. *Journal of Grid Computing*, 2(1) :3–14, 2004.

[20] Jeremy J Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson. Jena : implementing the semantic web recommendations. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 74–83. ACM, 2004.

[21] Raziel Carvajal-Gómez, Davide Frey, Matthieu Simonin, and Anne-Marie Kermarrec. Webgc gossiping on browsers without a server. In *International Conference on Web Information Systems Engineering*, pages 332–336. Springer, 2015.

[22] Ciro Cattuto, Dominik Benz, Andreas Hotho, and Gerd Stumme. Semantic analysis of tag similarity measures in collaborative tagging systems. *arXiv preprint arXiv :0805.2045*, 2008.

[23] Thomas Cerqueus, Sylvie Cazalens, and Philippe Lamarre. Gossiping correspondences to reduce semantic heterogeneity of unstructured p2p systems. In *International Conference on Data Management in Grid and P2P Systems*, pages 37–48. Springer, 2011.

[24] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable : A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2) :4, 2008.

[25] Ahmed Charles, Ronaldo Menezes, and Robert Tolksdorf. On the implementation of swarmlinda. In *Proceedings of the 42nd annual Southeast regional conference*, pages 297–298. ACM, 2004.

[26] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W Hong. Freenet : A distributed anonymous information storage and retrieval system. In *Designing privacy enhancing technologies*, pages 46–66. Springer, 2001.

[27] Rackspace Cloud Computing. Openstack open source cloud computing software, 2012.

[28] Arturo Crespo and Hector Garcia-Molina. Semantic overlay networks for p2p systems. In *AP2PC*, volume 3601, pages 1–13. Springer, 2004.

[29] Frank Dabek, M Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with cfs. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 202–215. ACM, 2001.

[30] Jeffrey Dean and Sanjay Ghemawat. Mapreduce : simplified data processing on large clusters. *Communications of the ACM*, 51(1) :107–113, 2008.

[31] Emanuele Della Valle, Andrea Turati, and Alessandro Ghioni. Page : A distributed infrastructure for fostering rdf-based interoperability. In *DAIS*, volume 6, pages 347–353. Springer, 2006.

[32] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12. ACM, 1987.

[33] Alexandros G Dimakis, Anand D Sarwate, and Martin J Wainwright. Geographic gossip : Efficient aggregation for sensor networks. In *Proceedings of the 5th international conference on Information processing in sensor networks*, pages 69–76. ACM, 2006.

[34] Hoang T Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing : architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18) :1587–1611, 2013.

[35] Zhenhai Duan, Kartik Gopalan, and Yingfei Dong. Push vs. pull : Implications of protocol design on controlling unwanted traffic. *SRUTI*, 5 :25–30, 2005.

[36] Marc Ehrig, Peter Haase, Ronny Siebes, Steffen Staab, Heiner Stuckenschmidt, Rudi Studer, and Christoph Tempich. The swap data and metadata model for semantics-based peer-to-peer systems. In *German Conference on Multiagent System Technologies*, pages 144–155. Springer, 2003.

[37] Michael D Ekstrand, John T Riedl, Joseph A Konstan, et al. Collaborative filtering recommender systems. *Foundations and Trends® in Human–Computer Interaction*, 4(2) :81–173, 2011.

[38] Clarence A Ellis and Simon J Gibbs. Concurrency control in groupware systems. In *Acm Sigmod Record*, volume 18, pages 399–407. ACM, 1989.

[39] MEC ETSI. Mobile-edge computing. *Introductory Technical White Paper*, 2014.

[40] P Th Eugster, Rachid Guerraoui, Sidath B Handurukande, Petr Kouznetsov, and A-M Kermarrec. Lightweight probabilistic broadcast. *ACM Transactions on Computer Systems (TOCS)*, 21(4) :341–374, 2003.

[41] Patrick T Eugster, Rachid Guerraoui, A-M Kermarrec, and Laurent Massoulié. Epidemic information dissemination in distributed systems. *Computer*, 37(5) :60–67, 2004.

[42] Jean Ferrié, Nicolas Vidot, and Michelle Cart. Concurrent undo operations in collaborative environments using operational transformation. *On the Move to Meaningful Internet Systems 2004 : CoopIS, DOA, and ODBASE*, pages 155–173, 2004.

[43] Adriano Fiorese, Paulo Simoes, and Fernando Boavida. Oman–a management architecture for p2p service overlay networks. In *IFIP International Conference on Autonomous Infrastructure, Management and Security*, pages 14–25. Springer, 2010.

[44] Adriano Fiorese, Paulo Simões, and Fernando Boavida. Peer selection in p2p service overlays using geographical location criteria. In *International Conference on Computational Science and Its Applications*, pages 234–248. Springer, 2012.

[45] Pauline Folz, Hala Skaf-Molli, and Pascal Molli. Cyclades : A decentralized cache for linked data fragments. In *ESWC : Extended Semantic Web Conference*, 2016.

[46] Michael J Freedman, Eric Freudenthal, and David Mazieres. Democratizing content publication with coral. In *NSDI*, volume 4, pages 18–18, 2004.

[47] Davide Frey, Mathieu Goessens, and Anne-Marie Kermarrec. Behave : Behavioral cache for web content. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 89–103. Springer, 2014.

[48] Ayalvadi J Ganesh, A-M Kermarrec, and Laurent Massoulié. Peer-to-peer membership management for gossip-based protocols. *IEEE transactions on computers*, 52(2) :139–149, 2003.

[49] David Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(1) :80–112, 1985.

[50] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *ACM SIGOPS operating systems review*, volume 37–5, pages 29–43. ACM, 2003.

[51] Jim Gray, Pat Helland, Patrick O'Neil, and Dennis Shasha. The dangers of replication and a solution. *ACM SIGMOD Record*, 25(2) :173–182, 1996.

[52] V Hadzilacos and N Goodman. Concurrency control and recovery in database systems. 1987.

[53] Lu Han, Magdalena Punceva, Badri Nath, S Muthukrishnan, and Liviu Iftode. Socialcdn : Caching techniques for distributed social networks. In *Peer-to-Peer Computing (P2P), 2012 IEEE 12th International Conference on*, pages 191–202. IEEE, 2012.

[54] DM Rasanjalee Himali, Shamkant B Navathe, and Sushil K Prasad. Sas : Semantics aware search in p2p networks. In *Distributed Computing Systems Workshops (ICDCSW), 2013 IEEE 33rd International Conference on*, pages 178–183. IEEE, 2013.

[55] Kirak Hong, David Lillethun, Umakishore Ramachandran, Beate Ottenwälder, and Boris Koldehofe. Mobile fog : A programming model for large-scale applications on the internet of things. In *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*, pages 15–20. ACM, 2013.

[56] Xueshi Hou, Yong Li, Min Chen, Di Wu, Depeng Jin, and Sheng Chen. Vehicular fog computing : A viewpoint of vehicles as the infrastructures. *IEEE Transactions on Vehicular Technology*, 65(6) :3860–3873, 2016.

[57] Luis Daniel Ibáñez, Hala Skaf-Molli, Pascal Molli, and Olivier Corby. Synchronizing semantic stores with commutative replicated data types. In *Proceedings of the 21st International Conference on World Wide Web*, pages 1091–1096. ACM, 2012.

[58] Luis-Daniel Ibáñez, Hala Skaf-Molli, Pascal Molli, and Olivier Corby. Live linked data : synchronising semantic stores with commutative replicated data types. *International Journal of Metadata, Semantics and Ontologies 4, 8*, 8(2) :119–133, 2013.

[59] Luis-Daniel Ibáñez, Hala Skaf-Molli, Pascal Molli, and Olivier Corby. Col-graph : Towards writable and scalable linked open data. In *International Semantic Web Conference*, pages 325–340. Springer, 2014.

[60] Zachary G Ives, Alon Y Halevy, Peter Mork, and Igor Tatarinov. Piazza : mediation and integration infrastructure for semantic web data. *Web Semantics : Science, Services and Agents on the World Wide Web*, 1(2) :155–175, 2004.

[61] Zachary G Ives, Alon Y Halevy, Peter Mork, and Igor Tatarinov. Scalable, peer-based mediation across xml schemas and ontologies., 2006.

[62] Sitaram Iyer, Antony Rowstron, and Peter Druschel. Squirrel : A decentralized peer-to-peer web cache. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 213–222. ACM, 2002.

[63] M Jelasity, W Kowalczyk, and M Van Steen. Newscast computing. technical report ir-cs-006. *Vrije Universiteit Amsterdam, Dept. of Computer Science*, 2003.

[64] Márk Jelasity, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten Van Steen. The peer sampling service : Experimental evaluation of unstructured gossip-based implementations. In *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 79–98. Springer-Verlag New York, Inc., 2004.

[65] Márk Jelasity, Wojtek Kowalczyk, and Maarten Van Steen. An approach to massively distributed aggregate computing on peer-to-peer networks. In *Parallel, Distributed and Network-Based Processing, 2004. Proceedings. 12th Euromicro Conference on*, pages 200–207. IEEE, 2004.

[66] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems (TOCS)*, 23(3) :219–252, 2005.

[67] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. T-man : Gossip-based fast overlay topology construction. *Computer networks*, 53(13) :2321–2339, 2009.

[68] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten Van Steen. Gossip-based peer sampling. *ACM Transactions on Computer Systems (TOCS)*, 25(3) :8, 2007.

[69] Sebastian Kaune, Konstantin Pussep, Christof Leng, Aleksandra Kovacevic, Gareth Tyson, and Ralf Steinmetz. Modelling the internet delay space based on geographical locations. In *Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on*, pages 301–310. IEEE, 2009.

[70] Bettina Kemme and Gustavo Alonso. A new approach to developing and implementing eager database replication protocols. *ACM Transactions on Database Systems (TODS)*, 25(3) :333–379, 2000.

[71] David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, pages 482–491. IEEE, 2003.

[72] A-M Kermarrec, Laurent Massoulié, and Ayalvadi J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed systems*, 14(3) :248–258, 2003.

[73] Dionisis Kontominas, Paraskevi Raftopoulou, Christos Tryfonopoulos, and Euripides GM Petrakis. \ mathcal {DS}ˆ 4 : A distributed social and semantic search system. In *European Conference on Information Retrieval*, pages 832–836. Springer, 2013.

[74] Danh Le-Phuoc, Josiane Xavier Parreira, Vinny Reynolds, and Manfred Hauswirth. Rdf on the go : An rdf storage and query processor for mobile devices. In *Proceedings of the 2010 International Conference on Posters & Demonstrations Track-Volume 658*, pages 149–152. CEUR-WS. org, 2010.

[75] Ken CK Lee, Hong Va Leong, and Antonio Si. Semantic query caching in a mobile environment. *ACM SIGMOBILE Mobile Computing and Communications Review*, 3(2) :28–36, 1999.

[76] Kisung Lee and Ling Liu. Scaling queries over big rdf graphs with semantic hash partitioning. *Proceedings of the VLDB Endowment*, 6(14) :1894–1905, 2013.

[77] Joao Leitao, José Pereira, and Luis Rodrigues. Hyparview : A membership protocol for reliable gossip-based broadcast. In *Dependable Systems and Networks, 2007. DSN'07. 37th Annual IEEE/IFIP International Conference on*, pages 419–429. IEEE, 2007.

[78] Ilias Leontiadis and Cecilia Mascolo. Geopps : Geographical opportunistic routing for vehicular networks. In *World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007. IEEE International Symposium on a*, pages 1–6. Ieee, 2007.

[79] Alexander Löser, Felix Naumann, Wolf Siberski, Wolfgang Nejdl, and Uwe Thaden. Semantic overlay clusters within super-peer networks. In *International Workshop on Databases, Information Systems, and Peer-to-Peer Computing*, pages 33–47. Springer, 2003.

[80] Pascal Molli, Gérald Oster, Hala Skaf-Molli, and Abdessamad Imine. Using the transformational approach to build a safe and generic data synchronizer. In *Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work*, pages 212–220. ACM, 2003.

[81] Alberto Montresor and Márk Jelasity. Peersim : A scalable p2p simulator. In *Peer-to-Peer Computing, 2009. P2P'09. IEEE Ninth International Conference on*, pages 99–100. IEEE, 2009.

[82] Alberto Montresor, Márk Jelasity, and Ozalp Babaoglu. Robust aggregation protocols for large-scale overlay networks. In *Dependable Systems and Networks, 2004 International Conference on*, pages 19–28. IEEE, 2004.

[83] Matteo Mordacchini, Ranieri Baraglia, Patrizio Dazzi, and Laura Ricci. A p2p recommender system based on gossip overlays (prego). In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 83–90. IEEE, 2010.

[84] Brice Nédelec, Pascal Molli, and Achour Mostefaoui. Crate : Writing stories together with our browsers. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 231–234. International World Wide Web Conferences Steering Committee, 2016.

[85] Brice Nédelec, Julian Tanke, Davide Frey, Pascal Molli, and Achour Mostefaoui. *Spray : an Adaptive Random Peer Sampling Protocol.* PhD thesis, LINA-University of Nantes ; INRIA Rennes-Bretagne Atlantique, 2015.

[86] Wolfgang Nejdl, Martin Wolpers, Wolf Siberski, Christoph Schmitz, Mario Schlosser, Ingo Brunkhorst, and Alexander Löser. Super-peer-based routing and clustering strategies for rdf-based peer-to-peer networks. In *Proceedings of the 12th international conference on World Wide Web*, pages 536–543. ACM, 2003.

[87] Takayuki Nishio, Ryoichi Shinkuma, Tatsuro Takahashi, and Narayan B Mandayam. Service-oriented heterogeneous resource sharing for optimizing service latency in mobile cloud. In *Proceedings of the first international workshop on Mobile cloud computing & networking*, pages 19–26. ACM, 2013.

[88] Nicolas Nova, Fabien Girardin, and Pierre Dillenbourg. Etude empirique de l'utilisation de la géolocalisation en collaboration mobile. In *Proceedings of the 17th Conference on l'Interaction Homme-Machine*, pages 207–210. ACM, 2005.

[89] Esther Pacitti, Pascale Minet, and Eric Simon. *Fast algorithms for maintaining replica consistency in lazy master replicated databases.* PhD thesis, INRIA, 1999.

[90] Christopher R Palmer and Gordon V Cormack. Operation transforms for a distributed shared spreadsheet. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, pages 69–78. ACM, 1998.

[91] Josiane Xavier Parreira, Sebastian Michel, and Gerhard Weikum. p2pdating : Real life inspired semantic overlay networks for web search. *Information Processing & Management*, 43(3) :643–664, 2007.

[92] Marco Picone, Michele Amoretti, and Francesco Zanichelli. Geokad : A p2p distributed localization protocol. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, pages 800–803. IEEE, 2010.

[93] Michel Plu, Pascal Bellec, Layda Agosto, and Walter Van de Velde. The web of people : A dual view on the www. In *WWW (Alternate Paper Tracks)*, 2003.

[94] Denis Ranger and Jean-François Cloutier. Scalable peer-to-peer rdf query algorithm. In *International Conference on Web Information Systems Engineering*, pages 266–274. Springer, 2005.

[95] Alberto Reggiori, Dirk-Willem van Gulik, and Zavisa Bjelogrlic. Indexing and retrieving semantic web resources : the rdfstore model. In *SWAD-Europe Workshop on Semantic Web Storage and Retrieval*, pages 13–14, 2003.

[96] Sean Rhea, Dennis Geels, Timothy Roscoe, John Kubiatowicz, et al. Handling churn in a dht. In *Proceedings of the USENIX Annual Technical Conference*, volume 6, pages 127–140. Boston, MA, USA, 2004.

[97] Elaine Rich and Kevin Knight. Artificial intelligence. *McGraw-Hill, New*, 1991.

[98] Thomas Risse, Predrag Knezevic, Carlo Meghini, Robert Hecht, and Fiore Basile. The bricks infrastructure-an overview. In *8th International Conference EVA*. Citeseer, 2005.

[99] Antony Rowstron and Peter Druschel. Pastry : Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 329–350. Springer, 2001.

[100] Antony Rowstron and Peter Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 188–201. ACM, 2001.

[101] Yasushi Saito and Marc Shapiro. Optimistic replication. *ACM Computing Surveys (CSUR)*, 37(1) :42–81, 2005.

[102] Subhadeep Sarkar, Subarna Chatterjee, and Sudip Misra. Assessment of the suitability of fog computing in the context of internet of things. *IEEE Transactions on Cloud Computing*, 2015.

[103] Mohamed Sarwat, Jie Bao, Chi-Yin Chow, Justin Levandoski, Amr Magdy, and Mohamed F Mokbel. Context awareness in mobile systems. In *Data Management in Pervasive Systems*, pages 257–287. Springer, 2015.

[104] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4), 2009.

[105] Bernhard Schandl and Stefan Zander. A framework for adaptive rdf graph replication for mobile semantic web applications. In *Joint Workshop on Advanced Technologies and Techniques for Enterprise Information Systems (Session on Managing Data with Mobile Devices), Milan, Italy*, pages 154–163, 2009.

[106] Marc Shapiro, Karthikeyan Bhargavan, and Nishith Krishna. A constraint-based formalism for consistency in replicated systems. In *International Conference On Principles Of Distributed Systems*, pages 331–345. Springer, 2004.

[107] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. *A comprehensive study of convergent and commutative replicated data types.* PhD thesis, Inria–Centre Paris-Rocquencourt ; INRIA, 2011.

[108] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In *Symposium on Self-Stabilizing Systems*, pages 386–400. Springer, 2011.

[109] Heng Tao Shen, Yanfeng Shu, and Bei Yu. Efficient semantic-based content search in p2p network. *IEEE Transactions on Knowledge and Data Engineering*, 16(7) :813–826, 2004.

[110] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*, pages 1–10. IEEE, 2010.

[111] Hala Skaf-Molli, Charbel Rahhal, and Pascal Molli. Peer-to-peer semantic wikis. In *International Conference on Database and Expert Systems Applications*, pages 196–213. Springer, 2009.

[112] Vasco NGJ Soares, Joel JPC Rodrigues, and Farid Farahmand. Geospray : A geographic routing protocol for vehicular delay-tolerant networks. *Information Fusion*, 15 :102–113, 2014.

[113] Evjola Spaho, Admir Barolli, Fatos Xhafa, and Leonard Barolli. P2p data replication : Techniques and applications. In *Modeling and Processing for Next-Generation Big-Data Technologies*, pages 145–166. Springer, 2015.

[114] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S Raghavendra. Spray and wait : an efficient routing scheme for intermittently connected mobile networks. In *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, pages 252–259. ACM, 2005.

[115] Angelos Stavrou, Dan Rubenstein, and Sambit Sahu. A lightweight, robust p2p system to handle flash crowds. *IEEE Journal on Selected Areas in Communications*, 22(1) :6–17, 2004.

[116] Ion Stoica, Robert Morris, David Liben-Nowell, David R Karger, M Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord : a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking (TON)*, 11(1) :17–32, 2003.

[117] Ivan Stojmenovic, Sheng Wen, Xinyi Huang, and Hao Luan. An overview of fog computing and its security issues. *Concurrency and Computation : Practice and Experience*, 28(10) :2991–3005, 2016.

[118] Chengzheng Sun and Clarence Ellis. Operational transformation in real-time group editors : issues, algorithms, and achievements. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, pages 59–68. ACM, 1998.

[119] Chengzheng Sun, Xiaohua Jia, Yanchun Zhang, Yun Yang, and David Chen. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 5(1) :63–108, 1998.

[120] Chunqiang Tang, Zhichen Xu, and Sandhya Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 175–186. ACM, 2003.

[121] Douglas B Terry, Marvin M Theimer, Karin Petersen, Alan J Demers, Mike J Spreitzer, and Carl H Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *ACM SIGOPS Operating Systems Review*, volume 29, pages 172–182. ACM, 1995.

[122] Giovanni Tummarello, Christian Morbidoni, Joackin Petersson, Paolo Puliti, and Francesco Piazza. Rdfgrowth, a p2p annotation exchange algorithm for scalable semantic web applications. *P2PKM*, 108, 2004.

[123] William Van Woensel and Sven Casteleyn. A mobile query service for integrated access to large numbers of online semantic web data sources. *Web Semantics : Science, Services and Agents on the World Wide Web*, 36 :58–76, 2016.

[124] William Van Woensel, Sven Casteleyn, Elien Paret, and Olga De Troyer. Transparent mobile querying of online rdf sources using semantic indexing and caching. In *International Conference on Web Information Systems Engineering*, pages 185–198. Springer, 2011.

[125] Luis M Vaquero and Luis Rodero-Merino. Finding your way in the fog : Towards a comprehensive definition of fog computing. *ACM SIGCOMM Computer Communication Review*, 44(5) :27–32, 2014.

[126] Spyros Voulgaris, Daniela Gavidia, and Maarten Van Steen. Cyclon : Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2) :197–217, 2005.

[127] Spyros Voulgaris, A-M Kermarrec, and Laurent Massoulié. Exploiting semantic proximity in peer-to-peer content searching. In *Distributed Computing Systems, 2004. FTDCS 2004. Proceedings. 10th IEEE International Workshop on Future Trends of*, pages 238–243. IEEE, 2004.

[128] Spyros Voulgaris and Maarten Van Steen. An epidemic protocol for managing routing tables in very large peer-to-peer networks. In *International Workshop on Distributed Systems : Operations and Management*, pages 41–54. Springer, 2003.

[129] Spyros Voulgaris and Maarten Van Steen. Epidemic-style management of semantic overlays for content-based searching. In *European Conference on Parallel Processing*, pages 1143–1152. Springer, 2005.

[130] Spyros Voulgaris and Maarten Van Steen. Vicinity : A pinch of randomness brings out the structure. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 21–40. Springer, 2013.

[131] Stéphane Weiss, Pascal Urso, and Pascal Molli. Wooki : a p2p wiki-based collaborative writing tool. In *International Conference on Web Information Systems Engineering*, pages 503–512. Springer, 2007.

[132] Stephane Weiss, Pascal Urso, and Pascal Molli. Logoot-undo : Distributed collaborative editing system on p2p networks. *IEEE Transactions on Parallel and Distributed Systems*, 21(8) :1162–1174, 2010.

[133] Dale Willis, Arkodeb Dasgupta, and Suman Banerjee. Paradrop : a multi-tenant platform to dynamically install third party services on wireless gateways. In *Proceedings of the 9th ACM workshop on Mobility in the evolving internet architecture*, pages 43–48. ACM, 2014.

[134] Ian H Witten and Eibe Frank. Data mining : practical machine learning tools and techniques with java implementations, 1999.

[135] Marcin Wylot and Philippe Cudré-Mauroux. Diplocloud : Efficient and scalable management of rdf data in the cloud. *IEEE Transactions on Knowledge and Data Engineering*, 28(3) :659–674, 2016.

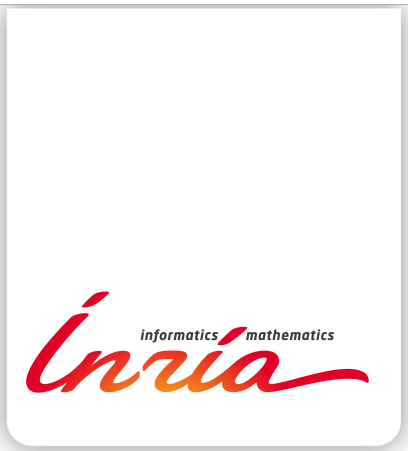
[136] Wei Xue, JuWei Shi, and Bo Yang. X-rime : cloud-based large scale social network analysis. In *Services Computing (SCC), 2010 IEEE International Conference on*, pages 506–513. IEEE, 2010.

[137] Hilmi Yildirim and Mukkai S Krishnamoorthy. A random walk method for alleviating the sparsity problem in collaborative filtering. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 131–138. ACM, 2008.

[138] Liang Zhang, Fangfei Zhou, Alan Mislove, and Ravi Sundaram. Maygh : Building a cdn from client web browsers. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 281–294. ACM, 2013.

[139] Ben Yanbin Zhao, John Kubiatowicz, Anthony D Joseph, et al. Tapestry : An infrastructure for fault-tolerant wide-area location and routing. 2001.

[140] Jiang Zhu, Douglas S Chan, Mythili Suryanarayana Prabhu, Preethi Natarajan, Hao Hu, and Flavio Bonomi. Improving web sites performance using edge servers in fog computing architecture. In *Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on*, pages 320–323. IEEE, 2013.



RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399