Contributions à la Gestion de l'Hétérogénéité dans les Environnements Distribués et Pervasifs

Luiz Angelo Steffenel

HABILITATION
À DIRIGER DES RECHERCHES

présentée à

l'UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

École doctorale Sciences, Technologies, Santé

Luiz Angelo STEFFENEL

Contributions à la Gestion de l'Hétérogénéité dans les Environnements Distribués et Pervasifs

Soutenue publiquement le 8 décembre 2017

Composition du jury :

Président du jury :	Carine SOUVEYET	Professeur à l'Université Paris 1 Panthéon-Sorbonne
Rapporteurs :	Christophe CÉRIN	Professeur à l'Université Paris 13 - IUT Villetaneuse
	Emmanuel JEANNOT	Directeur de Recherche, INRIA Bordeaux Sud-Ouest
	Philippe ROOSE	Maître de Conférences HDR à l'Université de Bayonne
Examinateurs :	Massimo VILLARI	Professeur à l'Università degli Studi di Messina, Italie
Directeur :	Olivier FLAUZAC	Professeur à l'Université de Reims Champagne-Ardenne

Laboratoire CReSTIC (EA 3804) – Équipe CASH

à Manuele
Tu as toujours été là pour me donner des ailes

Remerciements

Ce mémoire a été rendu possible grâce aux soutiens de nombreuses personnes, et il serait impossible de tous les lister. Toutefois, je tiens spécialement à rendre hommage à certaines personnes...

Parce qu'ils ont accepté de juger ce document qui retrace plus de dix années de travaux de recherche et de m'accorder leurs conseils et leurs suggestions pour améliorer ce document, je tiens à remercier très vivement les rapporteurs de cette habilitation. Tout d'abord, j'aimerais remercier Emmanuel Jeannot, que j'estime beaucoup par ses travaux en Parallélisme et Calcul Distribué, et qui m'a accueilli lors de mon passage l'INRIA Lorraine (LORIA) en 2005. De même, je remercie Christophe Cérin, dont l'expertise en Systèmes Distribués et le Pair-à-Pair n'est plus à démontrer. Nos multiples discussions au détour des conférences et des événements scientifiques m'ont conduit sans hésiter à solliciter son avis très pointu. Finalement, un grand merci à Philippe Roose, que j'ai connu un peu plus récemment et dont l'évaluation apporte un regard frais à mon travail.

Je tiens aussi à exprimer toute ma gratitude aux examinateurs qui ont accepté de composer le jury. Merci à Carine Souveyet, qui m'a fait l'honneur de bien vouloir présider la commission d'évaluation. Ses conseils avisés et sa manière particulière de déconstruire un sujet afin de mieux le comprendre ont contribué à l'amélioration de ce document. Merci aussi à Massimo Villari, qui a accepté de participer à mon jury en tant qu'examinateur malgré la barrière de la langue (*Grazie Massimo !*). Notre intérêt commun autour de l'Internet des Objets m'a permis de tisser des contacts avec M Villari et son équipe, et j'espère voir cette collaboration s'enrichir.

Un grand merci à Olivier Flauzac, qui a accepté d'être le garant de cette habilitation et dont les conseils toujours pertinents ont contribué à la préparation de ce document. Depuis mon arrivée à Reims Olivier n'a pas cessé de m'encourager à développer de nouveaux projets de recherche, de participer à des co-encadrements de thèse et aussi à prendre des responsabilités pédagogiques et administratives.

Les travaux de recherche présentés ici sont bien évidemment les fruits d'un environnement de travail propice à la collaboration et à l'épanouissement personnel. Merci donc à l'ensemble de l'équipe CASH du Laboratoire CReSTIC (Thibault Bernard, Jean-Charles Boisson, Cyril Rabat, Pierre Delisle, Christophe Jaillet, François Alin), et plus particulièrement à Michaël Krajecki et Florent Nolot pour leur confiance et leurs nombreux conseils scientifiques et administratifs.

Je dois également remercier Thierno Diallo et Romain Vasseur pour m'avoir fait confiance pour leur thèse de doctorat, ainsi qu'aux collègues qui ont fait ces co-encadrements à mes côtés (Olivier Flauzac et Manuel Dauchez/Stéphanie Baud, respectivement).

Je remercie aussi les collègues Hervé Deleau et Arnaud Renard, pour leur amitié et les discussions animées et variées autour d'un café en arrivant au bureau le matin, mais aussi pour les travaux qu'on développe ensemble avec le Centre de Calcul ROMEO. De même, je remercie

Bénédicte Legrand de l'Université Paris 1 pour ses encouragements, lorsque je me suis lancé le défi de rédiger cette habilitation.

Merci à mon père Valduino, à ma mère Ilda, ainsi qu'à mon petit frère Felipe, qui m'ont toujours poussé vers la curiosité scientifique. De même, j'aimerais remercier ma belle-famille, qui m'a accueilli si gentiment.

Pour finir ces remerciements, je veux dire un grand `MERCI` à mon épouse Manuele. Ton soutien au quotidien a marqué indéniablement l'ensemble de mon travail, non seulement par ta touche personnelle que par ton apport professionnel. Le fait de partager le même intérêt par l'informatique (et par notre profession d'enseignant-chercheur) me rend plus fort, nos différents points de vue me donnent plus de recul, nos discussions et ta rigueur scientifique me font aller au-delà de ma zone de confort et ainsi de progresser. Merci pour tout ce que tu représentes !

Table des matières

Introduction

La définition du mot **hétérogénéité** donnée par le Dictionnaire Larousse ("*Manque d'unité, composé d'éléments de nature diverse*") n'est pas suffisamment développée pour qualifier les différents défis liés à l'hétérogénéité dans les systèmes et les applications distribuées. Afin de mieux comprendre ces défis, il est important d'identifier et de cataloguer les différents mécanismes liés à l'hétérogénéité.

Une première catégorie représente les variations des équipements composant un système informatique. Sous cette optique, l'hétérogénéité se présente comme une conséquence de la différente construction des dispositifs qui composent le système, notamment leur composition matérielle (processeurs, mémoire) et leur capacité de calcul. Même étant très réductrice, *l'hétérogénéité matérielle* est souvent utilisée pour qualifier des équipements : machines parallèles (symétriques), *clusters* (grappes d'ordinateurs), *grids* (grilles de calcul), infrastructures *cloud*, réseaux *ad-hoc* et pair-à-pair (P2P), etc.

À l'hétérogénéité matérielle s'ajoutent les problèmes de l'*hétérogénéité des tâches* et de l'*hétérogénéité des communications*. L'*hétérogénéité des tâches* résulte soit d'une différence matérielle, soit d'une distribution déséquilibrée de charge entre les différents ressources participant à un calcul. La prise en charge de l'hétérogénéité des tâches dépend beaucoup des contraintes du système ou de l'application, telles que la présence de dépendances entre les tâches ou bien des contraintes sur le temps d'exécution (Qualité de Service).

Dans le cas de l'hétérogénéité des communications, les variations peuvent être causées autant par la diversité matérielle (par exemple, en utilisant différentes technologies réseaux) que par la distance géographique (qui impacte les temps de communication). On retrouve l'hétérogénéité des communications surtout dans les systèmes distribués à grande échelle (*grids*, réseaux P2P, etc.) où le temps de communication devient un facteur non négligeable. La prise en charge de l'hétérogénéité des communications se fait notamment par l'optimisation des dépendances : limitation des communications sur les grandes distances, recouvrement des communications par de calculs, ordonnancement basé sur la topologie du réseau, etc. Toutefois, cette prise en charge ne peut pas se faire sans une connaissance des facteurs impactant la communication, d'où la nécessité de mesurer et de modéliser les communications.

Il est clair que la diversité matérielle et la diversité des communications (matérielle ou spatiale) constituent les facteurs les plus importants lors de l'exécution d'une application. Toutefois, il faut également assurer le fonctionnement d'un système distribué au travers des variations qu'il subit pendant toute la durée de son exécution. Cette forme d'hétérogénéité "*temporelle*" est issue de la dynamique de l'exécution d'un système. Plus exactement, les systèmes distribués peuvent être impactés par le départ ou l'arrivée de ressources, mais aussi par leur changement d'état ou de capacité. De ce fait, la prise en compte de ces facteurs devient essentielle pour le bon fonctionnement d'un système ou d'une application. La prise en charge de la dynamicité implique plusieurs éléments : la surveillance des ressources et la détection des pannes/états

invalides, le suivi et la récupération des tâches attribuées à des éléments disparus et aussi le rééquilibrage de charge dans le cas d'une augmentation des ressources disponibles. En effet, la dynamicité affecte l'utilisation des ressources, car même sans une défaillance un système doit pouvoir être amené à gérer plusieurs tâches indépendantes, chacune avec des besoins propres.

D'un point de vue applicatif, on peut également rencontrer des difficultés avec une tout autre catégorie d'hétérogénéité, transversale aux trois précédentes : *l'hétérogénéité des données*. En effet, le développement d'une application est souvent guidé par la manière dont on accède aux données, et le *big data* a mis en évidence le besoin de gérer non seulement des grands volumes de données mais surtout leur variété. Les données peuvent se présenter sous des abstractions bien connues, comme par exemple les objets dans la mémoire, les fichiers, les URIs ou les requêtes distantes (RPC, Web services, etc.). Toutefois, il est rare qu'une application dispose d'un accès uniforme à tout type de donnée, ce qui implicitement introduit de l'hétérogénéité au niveau de l'accès aux données et à leur traitement.

Mon travail de recherche s'inscrit donc dans la gestion de l'hétérogénéité, sous ses différentes facettes. En naviguant entre ces aspects, j'ai pu travailler à la fois sur la tolérance aux fautes, la modélisation des performances, l'adaptation au contexte et l'ordonnancement, et même la spécification et le développement d'intergiciels (*middlewares*) pour le calcul distribué. Bien souvent j'ai pu profiter des collaborations et des projets dont j'ai été membre ou responsable pour introduire ou explorer des éléments liés à la gestion de l'hétérogénéité, tout comme dans les thèses de doctorat que j'ai co-encadré.

Ce mémoire présente une partie de ces contributions, toutes ayant fait l'objet de publications dans des journaux et des conférences internationaux, mais aussi d'autres communications qui se trouvent sur la liste récapitulative à la fin de cet ouvrage. Le mémoire est organisé en cinq parties, dans lesquelles j'expose les différents aspects de mes travaux de recherche visant la gestion de d'hétérogénéité :

i l'évaluation et la modélisation des performances de communication dans les *grids* ;

ii la parallélisation et la gestion de l'hétérogénéité des tâches ;

iii l'adaptation à la dynamicité des ressources de calcul ;

iv la spécification d'une plateforme pair-à-pair visant l'accès à différentes sources de données ;

v la conception et le développement d'un intergiciel pair-à-pair pour le calcul distribué dans des environnements hétérogènes tels que le *fog computing* et l'Internet des Objets.

Dans la première partie, je présente des algorithmes originaux issus des travaux menés à la fin de ma thèse et dans les trois années subséquentes. Ces travaux visaient notamment la compréhension des facteurs impactant les opérations de communication collective dans les *grids*, et dont le but ultime était à la fois d'optimiser ces opérations mais aussi de pouvoir estimer leur performance avec une haute précision. J'exploite ainsi des méthodes de mesure de performance et de découverte de la topologie réseau afin de représenter correctement les environnements et pouvoir extraire des indicateurs aidant à choisir les algorithmes de communication collective les plus adaptés à chaque situation.

Dans la deuxième partie j'illustre les efforts pour la parallélisation et la gestion de l'exécution distribuée d'une application métier en biochimie. Ce travail a été développé dans le cadre du co-encadrement de thèse de doctorat de Romain Vasseur (thèse CIFRE en collaboration avec le Laboratoire MeDyC - Matrice Extracellulaire et Dynamique Cellulaire - UMR CNRS 7369 et la compagnie Bull-Atos), et visait le développement de stratégies HPC pour le *docking*

inversé, une technique de simulation des interactions biomoléculaires. Comme il n'était pas envisageable de paralléliser le code source de l'application, nous avons opté par le développement de stratégies visant à découper convenablement l'espace de recherche et permettre le traitement parallèle des tâches de calcul. Ceci a été accompagné par le développement d'une plateforme de déploiement capable de gérer l'exécution des tâches distribuées sur plusieurs nœuds ou bien de tirer profit des gestionnaires de tâches présents sur la plupart des *clusters* HPC.

La troisième partie de ce mémoire s'attaque à la dynamique des ressources et aux stratégies pour s'adapter à ces changements. Plus exactement, je reprends une partie des travaux effectués pendant le projet STIC-AmSud PER-MARE dans lequel nous avons apporté des améliorations à la plateforme *big data* Apache Hadoop afin de la rendre compatible avec des environnements hétérogènes et dynamiques (environnements pervasifs). Grâce à un mécanisme de collecte d'informations sur le contexte des ressources de calcul, l'ordonnanceur de Hadoop a été modifié afin d'adapter le lancement de tâches aux ressources disponibles à chaque instant. Outre la présentation du mécanisme de collecte de contexte et des stratégies d'intégration à Hadoop, cette partie inclut des *benchmarks* démontrant l'efficacité des solutions proposées.

La quatrième partie présente la spécification d'un réseau hiérarchique pour la gestion universelle de données, résultat du co-encadrement de la thèse de doctorat de Thierno Ahmadou Diallo (thèse en cotutelle avec l'Université Cheikh Anta Diop, Sénégal). La motivation pour ce travail a été celle de créer une base documentaire indépendante de la nature des données (fichiers, flux, bases de données, etc.) et qui pourrait être utilisée par les universités et les écoles en Afrique, dans lesquelles l'accès aux infrastructures de type *cloud* ne sont pas toujours évidentes à cause des vitesses d'interconnexion. Dans ce travail je présente seulement la partie dédiée à la spécification de la plateforme GRAPP&S, le cas d'usage retenu pendant la thèse (utilisation de la plateforme pour le *e-learning*) n'étant pas d'intérêt direct pour ce mémoire.

Finalement, la cinquième partie introduit la plateforme de calcul distribué CloudFIT. Développée initialement dans le cadre du projet STIC-AmSud PER-MARE, CloudFIT s'est révélé un bon outil pour le prototypage et le test de techniques pour le *fog computing* et l'Internet des Objets (*Internet of Things* - IoT). Dans un premier moment je présente l'architecture et les mécanismes de communication et de gestion des noeuds. Par la suite, j'introduis des stratégies pour l'ordonnancement adapté au contexte pour les ressources de calcul très hétérogènes, allant des dispositifs IoT aux *data centers* et infrastructures sur le *cloud*. Cette variété de ressources est un élément clé du *fog computing*, que j'aborde en proposant des stratégies pour la structuration multi-échelle du réseau ou bien de techniques pour renforcer la *data-locality*. Cette partie se termine par la présentation de deux travaux où CloudFIT a été utilisé en tant que plateforme de calcul. Dans le premier cas, il s'agit de l'exécution d'une application *big data* bien connue. Dans le deuxième cas, je présente la spécification et l'implémentation d'une application en physique de l'atmosphère destinée à la surveillance d'événements liés à la couche d'Ozone Antarctique.

Je conclurai enfin ce document en présentant les perspectives de recherche que j'espère pouvoir développer dans un avenir proche.

Chapitre 1

L'Hétérogénéité des Communications

Résumé

Dans l'ensemble des éléments qui composent les systèmes informatiques, les réseaux d'interconnexion sont parmi les éléments qui sont les plus exposés et les plus impactés par l'hétérogénéité. Cette hétérogénéité peut se présenter sous différentes formes : diversité d'équipements et de technologies, variations de performance et de disponibilité (volatilité), limitations liées aux caractéristiques physiques ou à la capacité des ressources, etc.

Si cette hétérogénéité doit être prise en compte lors du développement de systèmes et d'applications, il s'avère souvent que les solutions se limitent à pallier les éventuels problèmes. Parmi les travaux que j'ai mené dans le domaine de l'hétérogénéité des réseaux, on trouve souvent l'association entre l'observation (mesure des performances) et la modélisation (prédiction des performances). Ces deux facettes de la recherche visent la compréhension des facteurs liés à l'hétérogénéité, leur caractérisation et aussi l'optimisation des algorithmes de communication.

Dans ce sens, je me suis orienté vers l'étude et la modélisation des primitives de communication collective dans les réseaux hétérogènes de type grid. Contrairement aux communications réseaux point à point (unicast), les communications collectives visent la diffusion ou la collecte de messages auprès un ensemble de nœuds. Parmi les patrons de communication collective nous trouvons celui dit one-to-many, qui représente une diffusion (broadcast/multicast), mais aussi des patrons plus élaborés tels que many-to-many qui représente un échange total entre les nœuds : chaque nœud a un message différent à envoyer à chacun des autres nœuds du réseau.

Les travaux présentés ici concernent les développements effectués à la fin de mes travaux de thèse et poursuivis lors de mes activités en tant quATER à Nancy, entre 2005 et 2007, période dans laquelle j'ai intégré l'Équipe ALGORILLE de l'INRIA Lorraine. Dans le même registre, il faut rajouter des collaborations autour de la modélisation des performances des algorithmes parallèles de multiplication matricielle avec le professeur Wahid Nasri de l'ESSTT (Tunisie) ou des développements sur le broadcast pour les grids dans le cadre de la thèse de doctorat de Hazem Fkaier (Université Paris 13).

1.1 Modélisation des Performances d'un Réseau

Une des meilleures manières de comprendre le fonctionnement des algorithmes distribués et d'évaluer leur efficacité est de modéliser leur performance. Si d'un côté des facteurs non

déterministes influencent la performance des applications, comme par exemple la congestion des ressources, pour la plupart du temps leur impact sur le temps d'exécution est suffisamment limité [67]. Ainsi, la plus grande difficulté pour la modélisation des performances est la correcte représentation des facteurs liés à la communication.

Si les principes de la modélisation de performance peuvent être trouvés dans des travaux pionniers des années 60 et 70 [114], la modélisation de performance a connu une importante attention à partir des années 80, où des efforts importants ont été faits pour identifier des modèles de performance adaptés aux technologies et aux paradigmes qui ont surgi depuis la décennie précédente. Pour cette raison, dans ce chapitre nous voulons identifier les modèles qui ont les caractéristiques les plus adaptées à la modélisation réaliste des communications.

1.1.1 Exécution dans les Systèmes Distribués : définitions

Un système distribué peut être défini comme un ensemble d'unités de traitement ou de calcul indépendantes, reliées entre elles par des liens de communications. Ces unités de traitement contribuent au calcul d'un résultat en effectuant leurs exécutions de façon concurrente.

Un réseau d'interconnexion R, aussi appelé système distribué, est représenté par un graphe $G = (V, E)$, dans lequel V est l'ensemble des nœuds et E l'ensemble des arêtes qui représente les liens ou les canaux de communication entre les nœuds.

L'écriture d'un algorithme, ainsi que l'évaluation de ses performances, ne peut se faire que si on a défini un modèle d'exécution au préalable. Parmi les modèles existants dans la littérature, celui que nous adoptons est modèle à passage de messages. Le modèle à passage de messages définit des canaux de communication par lesquels transitent les messages. Il nécessite toutefois de poser des hypothèses sur les propriétés des canaux de communication : capacité (débit), délais de transmission (latence), caractéristiques des échanges...

Dans notre cas spécifique, nous voulons explorer le modèle à passage de messages afin d'apporter à la fois une représentation fidèle du comportement des communications mais aussi une utilisation simple, capable d'être appliquée aux environnements hétérogènes. Différentes abstractions permettent la modélisation des communications avec plus ou moins de détails, ainsi la prochaine section passera en revue certaines modèles que nous avons étudié.

1.1.2 Modélisation des Communications

Le modèle de Hockney [75] est un des plus utilisés pour décrire la communication point-à-point dans les machines parallèles à mémoire distribuée. Selon ce modèle, une communication entre deux nœuds est décrite par :

$$t = t_0 + \frac{m}{r_\infty}$$

où t_0 représente le temps nécessaire à l'envoi d'un message de taille zéro, *m* est la taille du message (en octets) et r_∞ est le débit asymptotique en Moctets/s, i.e., le débit maximal obtenu quand la taille du message s'approche de l'infini. Dans ce raisonnement, m/r_∞ représente le délai de transmission d'un message de *m* octets à travers un réseau avec un débit asymptotique de r_∞ Moctets/s.

Ce modèle est souvent transformé dans l'équation affine

$$t = \alpha + \beta m$$

où α correspond à la latence de transmission et β est le temps de transfert d'un octet sur ce réseau [116]. L'obtention des paramètres α et β peut se faire à travers l'utilisation de certains outils comme NWS [158].

Toutefois, un inconvénient de ce modèle est qu'il assume que le temps de transfert est proportionnel à la taille du message. Dans des situations réelles, certains facteurs comme la taille de la mémoire tampon et la segmentation des messages en paquets font varier le temps de transfert β selon la taille du message.

À partir de l'observation qu'un des principaux paramètres pour la modélisation des algorithmes parallèles est le coût de communication entre les nœuds, Bar-Noy et Kipnis ont proposé le modèle Postal [7]. Le modèle Postal est fondé sur un paramètre $\lambda = t_u/t_{snd}$, où t_{snd} est le temps nécessaire à un nœud pour envoyer un message (la latence d'envoi), alors que t_u représente le temps total nécessaire à la réception du message par le nœud destinataire. Une des innovations du modèle Postal par rapport à ses prédécesseurs est que ce modèle permet des situations où un nœud peut émettre plusieurs messages avant que le premier récepteur ait reçu son message : cela est dû à une analogie avec l'envoi de lettres par la poste. Ces deux modèles pêchent toutefois par le fait qu'ils considèrent un coût de transmission linéaire par rapport aux tailles des messages, ce qui n'est pas exact. Dans une tentative de spécifier un modèle de calcul plus réaliste, Culler *et al.* [43] ont proposé le modèle de coût LogP afin de permettre la spécification du surcout d'envoi, qui est normalement très significatif pour la performance d'un système. L'importance de ce paramètre est notamment observée sur des expériences en machines réelles et, surtout, dans le cas des clusters (grappes de calcul).

Bien sûr, LogP reste un modèle simplifié des architectures parallèles, dont certains aspects de la topologie du réseau, comme par exemple la congestion, ne sont pas directement considérés. Les paramètres LogP qui caractérisent la communication entre les nœuds sont les suivants :

L - représente la latence de communication entre deux nœuds distincts ;

o - le surcout d'initialisation associée à l'envoi/réception d'un message ;

g - aussi appelé "*gap*", cette valeur correspond au temps minimal nécessaire entre deux événements consécutifs d'envoi ou de réception ;

P - le nombre de nœuds.

Sous cette représentation, le modèle Postal de Bar-Noy et Kipnis devient un cas spécial du modèle LogP (avec $g = 1$ et $o = 0$).

Dans le cas du modèle LogP, le nombre maximum de messages en transit entre deux nœuds est de $\lceil L/g \rceil$. La latence maximale d'un réseau est la distance moyenne asymptotique entre les nœuds ; toutefois, dans le cas des réseaux fortement hétérogènes, la définition de ces limites est bien plus complexe.

pLogP

Le modèle pLogP (*parameterised LogP*) est une extension du modèle LogP présenté par Kielmann *et al.* [86]. Son objectif est de représenter à la fois des petits et des grands messages sous un modèle unifié. En effet, le modèle LogP ne disposait pas d'outils suffisamment précis pour représenter les différences observées lors de la transmission de petits et grands messages. Ces différences sont occasionnées par des politiques de transmission et d'acquittement dépendantes de l'implémentation des protocoles et donc impossibles à représenter avec les modèles précédents, plus théoriques.

Comme ses prédécesseurs, le modèle pLogP est inclus des paramètres qui représentent la latence, les surcouts d'initialisation, le *gap* et le nombre de nœuds. La première différence est l'utilisation de valeurs distinctes, o_r et o_s, pour représenter le surcout d'envoi et le surcout de réception d'un message. Toutefois, la principale caractéristique du modèle pLogP est que les paramètres qui représentent le *gap* et les surcouts sont paramétrés selon la taille du message *m* envoyé. Cela est spécialement important pour modéliser les communications avec des tailles de messages variables, une fois que, comme déjà observé par Alexandrov [2], la performance des communications n'est pas linéaire par rapport à la taille des messages.

D'autres aspects sont aussi différents par rapport aux modèles précédents. En effet, les notions de latence et de *gap* sont légèrement différentes de celles utilisées par le modèle LogP. Dans le cas du modèle pLogP, la latence inclut tous les facteurs qui peuvent retarder la communication entre deux nœuds, comme, par exemple, la copie de données en mémoire tampon vers les interfaces réseaux, qui s'ajoutent au temps de transfert des messages déjà considéré par LogP. Ainsi, dans le cas d'un réseau local, le paramètre *gap* est défini comme l'intervalle minimal entre deux transmissions ou réceptions consécutives, ce qui implique que $g(m) \geq os(m)$ et $g(m) \geq or(m)$ tient toujours. La relation entre ces différents paramètres est illustrée dans la Figure 1.1.

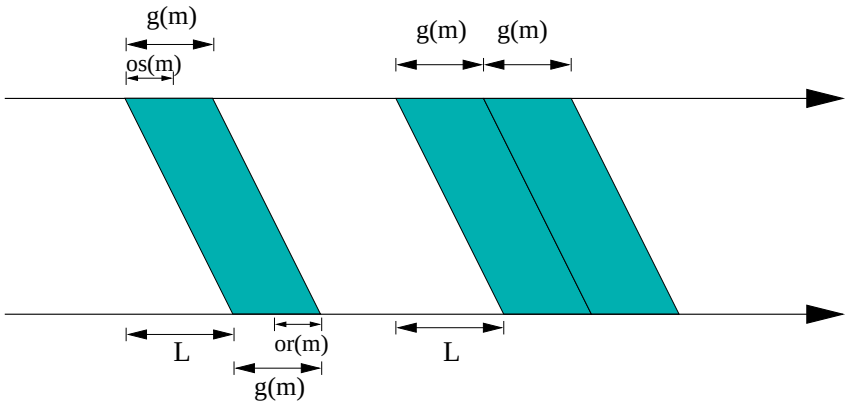

FIGURE 1.1 : Représentation d'une communication avec pLogP

En conséquence de cette nouvelle interprétation du *gap*, les paramètres o_s et o_r ont une importance moins évidente, une fois que leur coût dans un réseau local est souvent recouvert par celui du *gap*. Ainsi, pour représenter le temps nécessaire à la transmission d'un message de taille *m* entre deux nœuds avec des primitives de communication bloquantes (i.e., où l'émetteur attend la fin de la transmission), le modèle pLogP utilise l'expression $L + g(m)$, au lieu de $L + g + 2o$ comme dans le modèle LogP.

La variation des paramètres vis-à-vis de la taille des messages et des politiques d'émission est mise en évidence en Figure 1.2, où on affiche les différents temps de communication mesurés avec la bibliothèque applicative LAM-MPI [144]. On observe un changement de politique d'acquittement quand la taille des messages dépasse les 64 Ko, de manière à ce que le coût du *gap* dépend à la fois de la saturation de la fenêtre TCP et de la politique d'acquittement.

1.2 Modélisation de l'Opération de Diffusion *MPI_Bcast*

Parmi les opérations de communication collective, les diffusions de type *Broadcast* sont parmi les plus simples et les plus répandues. Une opération de *Broadcast* s'effectue quand un

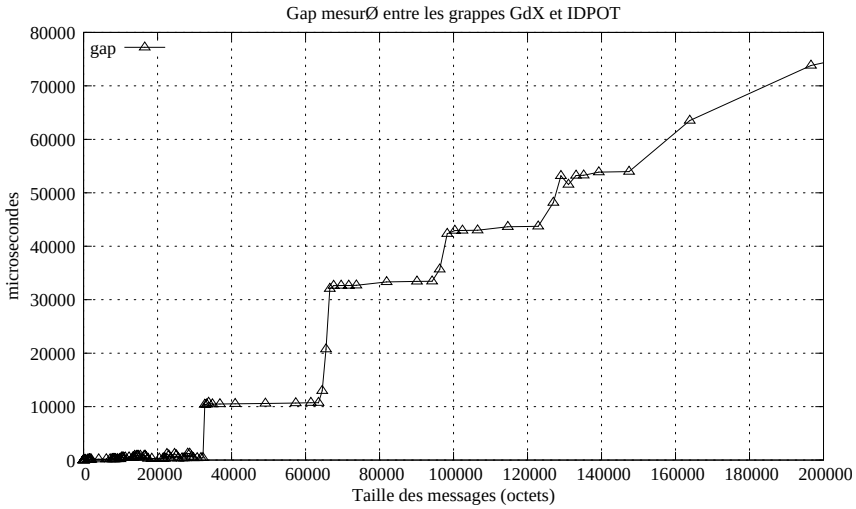

FIGURE 1.2 : Valeurs de *gap* mesurés entre deux machines distantes [8]

seul nœud, appelé *racine,* envoie le même message de taille *m* à tous les autres $(P-1)$ nœuds. Ceci représente en effet le patron de communication *one-to-all*.

Dans le cas du calcul distribué, la diffusion d'opérations est souvent nécessaire afin d'apporter des paramètres et des données à tous les nœuds. La compréhension de ces mécanismes et la modélisation de ses coûts présente un intérêt stratégique vis-à-vis de la scalabilité et de l'optimisation des algorithmes. Dans ce sens, il est aussi nécessaire prendre en compte les différences architecturales des environnements de communication : les stratégies efficaces pour un environnement homogène diffèrent de celles adaptées aux environnements où les communications sont hétérogènes. Afin de représenter ces deux cas, nous nous sommes intéressés à la modélisation et l'optimisation des communications collectives de type *Broadcast* dans deux types distincts de réseaux : les clusters (grappes de machines) et les *grids* (grilles de calcul). Le premier cas peut souvent être considéré comme homogène, alors que le deuxième apporte un degré d'hétérogénéité à cause des différences de communication *intra* et *inter-cluster*. Afin d'implémenter et valider des stratégies pour ces réseaux, nous avons choisi d'utiliser comme point de départ la bibliothèque MPI (Message Passing Interface), très utilisée par la communauté du calcul parallèle et qui dispose déjà d'une implémentation simple de l'opération *broadcast* appelée MPI_BCast.

1.2.1 Modélisation d'un *Broadcast* dans un réseaux homogène

L'opération *Broadcast* est une des plus simples opérations de communication collective : initialement, seul le nœud *racine* détient le message qui doit être diffusé ; à la fin de l'opération, une copie de ce message est déposée dans chaque nœud du groupe. L'approche classique pour implémenter l'opération *Broadcast* utilise des arbres qui sont décrits par deux paramètres, *d* et *h*, où *d* est le nombre maximum de successeurs qu'un nœud peut avoir, et *h* est la hauteur de cet arbre, le chemin le plus long qui relie la racine et les feuilles de cet arbre. Plus généralement, des arbres de diffusion avec différents degrés *d* et *h* peuvent être générés à partir d'un algorithme de type *arbre-alpha* (*alpha-tree* en anglais) suggéré par Bernaschi et Ianello [15]. À l'aide de cet algorithme et des paramètres du réseau, un arbre optimal peut être construit à partir des paramètres du réseau et avec *d, h* $\in$[1...*P*-1] tel que $\sum_{i=o}^{h} d^i \geq P$ soit respecté.

La performance des différentes formes fixes dépend surtout des paramètres du réseau, notamment le *gap*, la latence et le nombre de nœuds. Par conséquent, un réseau avec une latence

faible par rapport au *gap* favorise les algorithmes de type Arbre Binaire et Arbre Binomial, qui cherchent à minimiser le temps de communication par la multiplication des sources de transmission. Au contraire, si la latence est trop élevée par rapport au *gap*, les algorithmes de type Arbre Plat sont favorisés, où un seul nœud envoie des messages à tous les autres.

À partir des modèles de coût LogP [43] et pLogP [86] et de travaux comme ceux de Huse [78], Vadhiyar [148] et autres, nous avons déduit les formules qui représentent différentes stratégies de communication évaluées dans ce travail, comme indiqué dans le Tableau 1.1. Certaines de ces stratégies sont clairement inefficaces, comme par exemple le *broadcast* en Chaîne, qui exécute $P-1$ communications en série.

Stratégie	Modèle de Communication
Arbre Plat	$L + (P-1) \times g(m)$
Chaîne	$(P-1) \times (g(m) + L)$
Arbre Binaire	$\leq \lceil log_2 P \rceil \times (2 \times g(m) + L)$
Arbre Binomial	$\lceil log_2 P \rceil \times L + \lfloor log_2 P \rfloor \times g(m)$

TABLE 1.1 : Modèles de communication pour le *Broadcast*

Une autre possibilité de construire un *Broadcast* est la composition des chaînes de retransmission [11]. Cette stratégie, possible grâce à la segmentation des messages, présente des avantages importants, comme l'indiquent [86][145][12]. Dans un *Broadcast* Segmentée, la transmission des messages en segments permet le recouvrement de la transmission d'un segment *k* et la réception du segment *k*+1, minimisant le *gap*.

Dans ce cas, nous considérons que le segment de taille *s* d'un message *m* est un multiple de la taille du type basique de données qui est transmis, divisant alors le message initial *m* en *k* segments. Par conséquent, *g(s)* représente le *gap* d'un segment de taille *s*. Toutefois, le choix de la taille des segments reste dépendant des caractéristiques du réseau. En effet, l'utilisation de segments trop petits a un surcout non-négligeable dû à l'en-tête du message, alors que l'utilisation des segments trop grands ne permet pas l'exploitation intégrale du débit du réseau.

La recherche de la taille de segment *s* qui minimise le temps de communication se fait à l'aide des modèles de communication présentés dans le Tableau 1.2. D'abord, on cherche une taille de segment *s* qui minimise le temps de communication parmi $s = m/2^i$ pour $i \in [0 \ldots log_2 m]$. Ensuite, on peut affiner la recherche de la taille optimale avec l'aide d'heuristiques comme le "*local hill-climbing*" [86].

Stratégie	Modèle de Communication
Arbre Plat Segmenté	$L + (P-1) \times (g(s) \times k)$
Chaîne Segmentée (Pipeline)	$(P-1) \times (g(s) + L) + (g(s) \times (k-1))$
Arbre Binomial Segmenté	$\lceil log_2 P \rceil \times L + \lfloor log_2 P \rfloor \times g(s) \times k$
Pieuvre avec un degré *d*	$(d + \lceil \frac{P-(2^d+1)}{(2^d+1)} \rceil) \times (g(s) + L) + (g(s) \times (k-1))$
Scatter/Collection [145]	$(log_2 P + P - 1) \times L + 2 \times (\frac{p-1}{p}) \times g(m)$

TABLE 1.2 : Modèles de communication segmentée pour le *Broadcast*

Pour valider ces modèles de communication, nous avons choisi la comparaison entre les prédictions des modèles et les résultats réels obtenus à partir d'expérimentations sur différentes plates-formes réseaux. Pour illustrer notre approche, nous avons comparé des implémentations de MPI_Bcast selon les stratégies Arbre Plat, Arbre Binomial et Chaîne Segmentée. Ainsi,

les Figures 1.3 et 1.4, illustrent les valeurs mesurées expérimentalement des trois stratégies d'implémentation, qui suivent presque fidèlement les prédictions des modèles.

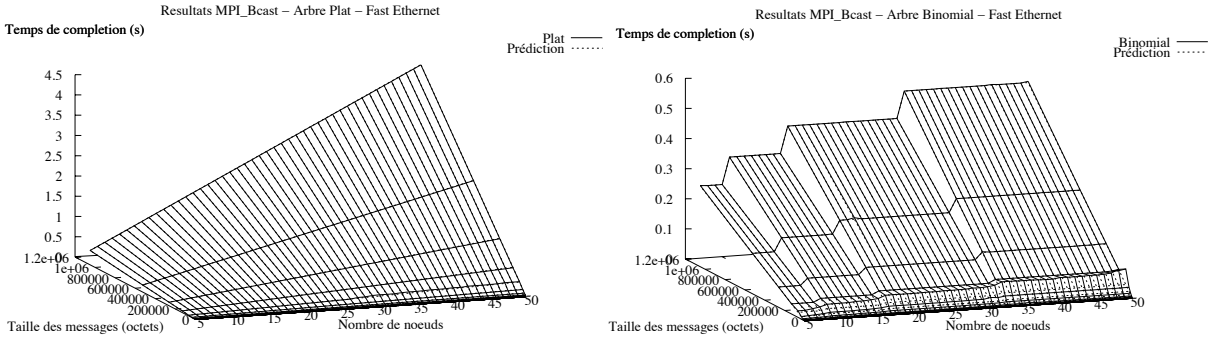

FIGURE 1.3 : Les performances réelles et prédites pour l'Arbre Plat (a) et l'Arbre Binomial (b) avec un réseau Fast Ethernet

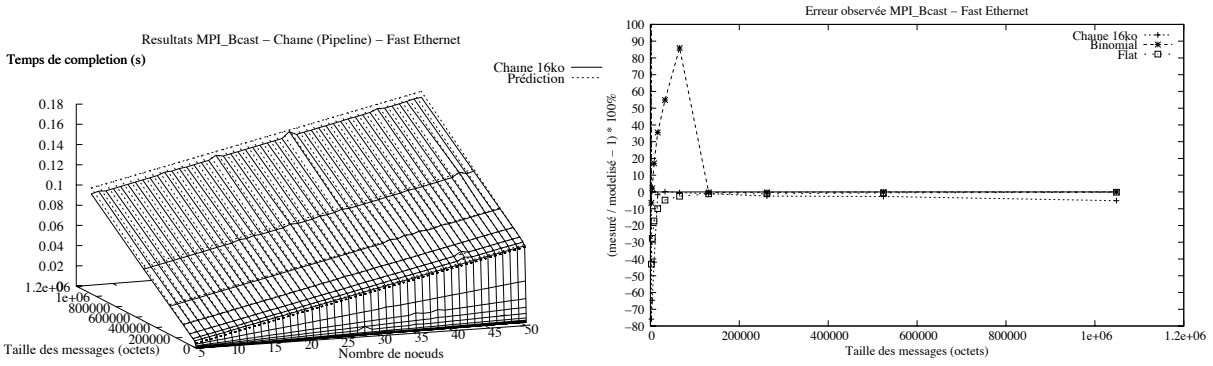

FIGURE 1.4 : Performances réelles et prédites pour la Chaîne Segmentée (a) et l'erreur des prédictions par rapport aux valeurs mesurées (b)

Plus spécifiquement, des différences entre les prédictions et les valeurs réelles sont observées surtout dans le cas de l'envoi de petits messages, qui ne suit pas un comportement linéaire par rapport à la taille des messages. Cette variation de performance a été l'objet d'analyses précédentes (cette discussion peut être retrouvée dans les articles [9] et [10]) et doit son origine à l'implémentation des politiques de transmission et d'acquittement TCP, qui parfois opte pour retarder l'envoi de petits messages.

En effet, TCP dispose d'une option *socket* TCP_NODELAY qui devrait permettre l'envoi de tout message sans attente. Seulement, nous avons observé que parfois un seul message à chaque n messages transmis n'est pas acquitté comme il le faut. Cette défaillance de l'implémentation protocole d'acquittement induit un temps supplémentaire qui se reflète sur un surcout lors de l'envoi de petits messages.

La Figure 1.4 résume aussi ces trois stratégies en affichant le taux d'erreur entre les mesures et les prédictions. Ici, nous observons que les résultats réels, normalement très proches des prédictions (à une marge de 10% maximum), s'écartent des prédictions jusqu'à 90% pour des messages autour de 128 Ko. Néanmoins, ces variations affectent des communications où la différence absolue n'est que de quelques millisecondes, ce qui n'empêche pas l'utilisation des modèles de communication pour choisir la meilleure stratégie de communication.

1.2.2 Modélisation du *Broadcast* dans un *Grid*

La détermination du meilleur arbre de diffusion pour un environnement homogène est une tâche relativement facile car dépend de paramètres de latence et débit (ou par extension, du *gap*) communs à tout le réseau.

Cependant, dans le cas d'un réseau hétérogène, ce problème devient bien plus difficile à cause des variations des paramètres de communication entre chaque pair de nœuds. En effet, l'identification du meilleur arbre de diffusion dans un réseau hétérogène est un problème NP-complet [18][13, 14][160]. La plupart des travaux dédiés à l'optimisation des communications collectives dans des environnements hétérogènes essayent donc de construire des arbres de diffusion en tenant compte du coût d'interconnexion entre chaque pair de nœud concernée par le *broadcast*. C'est le cas de Banikazemi [6], Bhat [18, 19] ou Mateescu [104].

Cependant, l'environnement de type *grid* est normalement caractérisé par un grand nombre de nœuds communicants, résultat de l'association des différents *clusters*. Dans ce cas, la complexité de la tâche d'optimisation est bien plus importante, et des simplifications s'imposent afin de permettre l'utilisation de telles méthodes dans la pratique. Une de ces simplifications est le regroupement des nœuds selon leurs performances relatives (par exemple, par rapport à la communication), de manière à ce que toute une classe de nœuds puisse être traitée comme une entité unique. De cette manière, le nombre important de nœuds dans un *grid* peut être encore facilement abordable par les méthodes d'optimisation classiques grâce à la division des communications en deux catégories, l'*inter-cluster* et l'*intra-cluster*.

Cependant, à défaut de leur apport aux algorithmes de *Broadcast*, ces techniques peuvent encore être améliorées. En effet, les travaux précédents ont été établis dans un contexte où les communications de longue distance étaient plusieurs ordres plus lents que celles à l'intérieur des réseaux locaux, et la réduction des communications *inter-clusters* permettait la minimisation de la congestion sur les liens les plus lents. Si cela est encore vrai en ce qui concerne la latence entre les nœuds, il n'est plus exact pour le débit d'un lien de longue distance. D'autre part, le faible coût du matériel informatique permet aujourd'hui que les *clusters* regroupent des centaines de nœuds. Or, plus le coût de diffusion *intra-clusters* devient important, plus son influence sur la performance sera importante au moment de définir l'ordonnancement des communications.

C'est exactement ce qui différencie les heuristiques traitant (ou ne traitant pas) la communication à l'intérieur des groupes : les heuristiques "traditionnelles" et celle appelées "heuristiques sensibles au contexte des *grids*" ("*grid-aware*" en anglais). Dans le premier cas, l'optimisation ne tient compte que des communications entre les différents coordinateurs, alors que le deuxième cas s'occupe aussi de la diffusion à l'intérieur des *clusters*.

Les prochaines sections présentent les différentes heuristiques étudiées pour l'optimisation des communications de type MPI_Bcast. Certaines de ces heuristiques sont la simple application des méthodes pour les réseaux hétérogènes dans le contexte des *clusters* hiérarchisées. Dans ce travail nous proposons trois nouvelles méthodes, qui au contraire des techniques précédentes, considèrent autant les communications entre les coordinateurs que les temps nécessaires à la diffusion des messages à l'intérieur des *clusters*.

Formalisme utilisé

Pour décrire les heuristiques présentées dans cette section, nous utilisons un formalisme de groupes similaire à celui de Bhat [19]. Dans ce formalisme, les *clusters* sont séparés en deux groupes, **A** et **B**. Le groupe **A** contient les *clusters* qui ont déjà reçu le message (la réception du message par le coordinateur du *cluster* est suffisante). Le groupe **B** contient les *clusters* qui

devront recevoir le message. De cette manière, le groupe **A** contient initialement le *cluster* du nœud *source* ou *racine*, tandis que le groupe **B** contient toutes les autres *clusters* du réseau.

À chaque étape, un émetteur appartenant au groupe **A** et un récepteur appartenant au groupe **B** sont choisis. Après la communication entre ces deux *clusters* (plus exactement, leurs coordinateurs), le *cluster* récepteur est transféré au groupe **A**.

L'implémentation de ces communications est faite de manière à rendre prioritaires les communications entre les *clusters*. En effet, les coordinateurs diffusent le message à l'intérieur de ses *clusters* seulement après la fin des communications *inter-clusters*. Cette stratégie favorise la multiplication des sources disponibles et l'application des heuristiques, ainsi que la prédiction du temps total d'exécution du *Broadcast*.

Heuristiques Traditionnelles

Diffusion en Arbre Plat (Flat)

L'heuristique en *Arbre Plat*, découpe la communication en deux niveaux, *inter-clusters* et *intra-clusters*.

Dans le premier niveau, le nœud *racine* envoie le message à tous les coordinateurs des différents *clusters*. L'ordre d'envoi suit le "rang" des différents *clusters*, prédéfini à l'initialisation. Formellement, cela veut dire qu'à chaque étape, le nœud *racine* choisi comme récepteur le premier *cluster* du groupe **B**. Dans cette "heuristique", le nœud émetteur est toujours le même (le nœud *racine*), malgré le fait que les *clusters* qui ont déjà reçu le message font désormais partie du groupe **A**. Dans le deuxième niveau de diffusion, exécuté à l'intérieur de chaque *cluster*, les coordinateurs exécutent un *broadcast* en arbre binomial.

Même si cette heuristique est très simple à implémenter, elle est toutefois très peu optimisée. En effet, la diffusion des données ne tient pas compte des performances des différents *clusters*, ni les vitesses d'interconnexion entre les *coordinateurs*. Même si l'utilisateur organise le fichier de description des *clusters* de manière à favoriser les communications émises d'un certain *cluster*, celles-ci restent soumises à une structure de diffusion *plate*.

Fastest Node First - FNF

L'heuristique *Fastest Node First* (le nœud le plus rapide d'abord) a été proposée par Banikazemi *et al.* [6]. Dans leur modèle de communication, le réseau est composé d'un certain nombre de nœuds P. À chaque nœud P_i on associe un coût d'envoi C_i. Ce coût C_i est indépendant de la destination et de la taille du message, et indique seulement la différence de vitesse entre les nœuds.

L'heuristique proposée par Banikazemi *et al.* nécessite $P - 1$ itérations, où à chaque étape l'heuristique définie un émetteur et un récepteur. Le récepteur est choisi parmi les possibles récepteurs du groupe **B** dont le coût C_i est le plus petit. L'émetteur est le nœud du groupe **A** qui peut finir la communication le plus rapidement possible. Cela dit, cette stratégie choisit l'émetteur le plus rapide et le récepteur qui pourrait retransmettre les messages le plus rapidement possible, à son tour.

L'efficacité de l'heuristique FNF dans le cadre des environnements homogènes a été démontrée par Liu [99], qui a prouvé que l'heuristique FNF produit des ordonnancements avec au plus deux fois le temps optimal.

Cependant, des environnements homogènes comme ceux considérés par Banikazemi sont assez rares dans les *grids*, ce qui rend cette heuristique très limitée par rapport à la modélisation

des communications. En effet, le modèle de coût unique C_{i} n'est pas suffisant pour représenter l'hétérogénéité d'un réseau d'interconnexion, comme indiqué par Bhat [19].

Fastest Edge First - FEF

Proposée par Bhat *et al.* [19], l'heuristique *Fastest Edge First* (l'arête la plus rapide d'abord) est un algorithme glouton qui fait partie d'une collection d'heuristiques proposées comme alternative à l'heuristique FNF.

Assez simple, cette heuristique est très similaire à l'heuristique FNF. Seulement, au lieu d'un coût de communication unique C_{i}, l'heuristique évalue le poids de chaque lien de communication $L_{i,j}$ entre deux nœuds différents (les arêtes), correspondants à la latence de communication entre les deux nœuds.

Pour identifier l'ordonnancement des communications nécessaire à l'exécution de l'opération de *Broadcast*, l'algorithme FEF, ordonne les nœuds du groupe A selon leurs arêtes les plus rapides. Cela permet le choix du lien le plus rapide parmi toutes les possibilités, et en même temps, sert à définir l'émetteur et le récepteur, déterminés implicitement par l'arête choisie. Une fois que le récepteur est désigné, celui-ci est transféré du groupe **B** vers le groupe **A**. À cet instant, les arêtes minimales doivent être recalculées.

Le raisonnement de cette heuristique est que le choix des liens les plus rapides permet d'augmenter rapidement le nombre d'émetteurs. À leur tour, ces émetteurs pourront disséminer le message vers les nœuds les plus éloignés, tout en choisissant le lien le moins coûteux.

Early Completion Edge First - ECEF

Selon les heuristiques précédentes, une fois que le récepteur était assigné, celui-ci était immédiatement transféré vers le groupe des émetteurs, le groupe **A**. Toutefois, à cause des délais de communication, il est possible que ce récepteur n'ait pas encore reçu le message et qu'il soit choisi pour le retransmettre à un deuxième nœud. La communication subira un retard supplémentaire, alors qu'une autre arête, moins rapide, pourrait finir la transmission plus vite si son émetteur a déjà le message.

Pour tenir compte des retards dus à la transmission des données, l'heuristique *Early Completion Edge First* (arête qui finit le plus tôt) considère aussi dans son évaluation l'instant où les émetteurs ont les données disponibles pour l'envoi. Ainsi, la *disponibilité* RT_i du nœud émetteur (*Ready Time* en anglais) est alors utilisée conjointement avec le temps nécessaire à la transmission du message entre les nœuds (le *gap* plus la latence) de manière à choisir le couple émetteur-récepteur qui minimise le temps :

$$RT_i + g_{i,j}(m) + L_{i,j}$$

Ainsi, l'objectif de cette heuristique est d'augmenter le nombre de nœuds qui peuvent *effectivement* transmettre les messages aux autres nœuds.

Early Completion Edge First with lookahead - ECEF-LA

Pour augmenter l'efficacité de l'heuristique précédente, la dernière heuristique proposée par Bhat *et al.* [19] propose une recherche plus approfondie sur les possibles choix. En effet, si l'objectif des heuristiques précédentes était la multiplication des sources disponibles, cela suppose que ces sources pourront, à leur tour, retransmettre les messages de manière efficace.

C'est ainsi que Bhat a proposé l'utilisation d'une fonction de *lookahead* (recherche en avant) pour évaluer si le choix d'un récepteur est réellement bon. De cette manière, l'algorithme calcule préalablement la fonction de *lookahead* F_j pour tous les nœuds dans le groupe **B**, et le pair émetteur-récepteur est celle qui minimise la somme :

$$RT_i + g_{i,j}(m) + L_{i,j} + F_j$$

Cette fonction de *lookahead* peut être définie de plusieurs façons. Bhat [19] propose, par exemple, le coût minimal pour que le nœud *j* transmette à d'autres nœuds encore dans le groupe **B**. Cette fonction est alors la suivante :

$$F_j = \min_{P_k \in B} (g_{j,k}(m) + L_{j,k})$$

Intuitivement, cette fonction indique l'utilité du nœud P_j si à son tour il est transféré vers le groupe **A**. D'ailleurs, Bhat a proposé d'autres fonctions de *lookahead*, dont par exemple la moyenne de la latence entre P_j et les autres nœuds en **B**, ou alors la latence moyenne entre les émetteurs et les récepteurs, si on considère que P_j est transféré vers le groupe **A**.

Heuristiques "grid-aware"

Les heuristiques présentées précédemment hiérarchisent les communications en deux niveaux - *inter-clusters* et *intra-clusters*, mais les fonctions d'évaluation ne tiennent compte que des coûts de transmission entre les coordinateurs des différentes *clusters* du réseau.

Comme nous l'avons déjà exposé, le coût d'une communication hiérarchique ne dépend pas seulement des latences entre les différents *clusters*, mais aussi du temps nécessaire à la diffusion des messages à l'intérieur de ces *clusters*. Ce coût de diffusion *intra-clusters* devient encore plus important avec l'augmentation du nombre de nœuds à l'intérieur des *clusters*, qui aujourd'hui dépasse facilement la centaine de machines. Par exemple, l'envoi d'un message de 1Mo entre deux *clusters* (l'un à Grenoble, l'autre à Paris) requiert 349 millisecondes, alors que le *broadcast* de ce même message entre 50 nœuds du *cluster* de Grenoble peut nécessiter jusqu'à 3 secondes selon l'algorithme utilisé. Si ce temps n'est pas pris en compte lors de la modélisation des communications, l'ordonnancement des communications risque d'être sous-optimal.

Plus exactement, le temps de diffusion *intra-clusters*, appelé T_k, correspond aux prédictions des modèles de communication vus précédemment. Cette notation T_k est équivalente à une notation où un nœud fictif k' est associé à chaque coordinateur k, dont :

$$L_{k,k'} + g_{k,k'} = \begin{cases} T_k & si\ k'\ est\ associe\ a\ k \\ \infty & pour\ tout\ autre\ noeud\ j \neq k \end{cases}$$

La présence d'un nœud fictif permet l'utilisation des heuristiques précédentes sans la nécessité d'une modification des algorithmes, notamment le ECEF. L'utilisation de T_k permet une implémentation plus simple des algorithmes, qui n'ont pas besoin de garder les deux identités *k* et *k'* associées à un *cluster k*. Toutefois, dans ce travail nous avons gardé la description séparée *L*, *g* et *T*, afin de permettre une identification plus facile des facteurs évalués.

Dans ce sens, nous présentons deux nouvelles stratégies d'évaluation dites "sensibles au contexte des *grids*", où le temps de diffusion *intra-cluster* est aussi considéré lors de la construction des arbres de diffusion. Pour mieux analyser l'efficacité de ces stratégies, nous avons aussi

développé une version de l'heuristique ECEF-LA où le temps *intra-clusters* est pris en compte. Cette version sert de comparaison par rapport aux heuristiques de Bhat, présentées précédemment.

ECEF-LA*t*

L'heuristique ECEF-LA*t* est l'évolution naturelle de l'heuristique ECEF-LA où nous utilisons une fonction de *lookahead* adaptée à la représentation du coût de communication *intra-clusters* T_k.

Ainsi, l'heuristique ECEF-LA*t* cherche à minimiser le coût total de transmission et le temps nécessaire à la diffusion d'un message dans un *cluster* distant (en effet, le "petit *t*" du nom de cette heuristique indique qu'on cherche le minimum des temps). Pour cela, elle utilise une fonction d'évaluation :

$$F_j = \min_{P_k \in B} (g_{j,k}(m) + L_{j,k} + T_k)$$

À l'instar de l'heuristique ECEF-LA, le but de cette stratégie est que le récepteur soit choisi parmi les *clusters* qui peuvent retransmettre le message le plus vite possible à d'autres *clusters*. L'adjonction du temps T_k dans la fonction d'évaluation implique aussi que le choix d'un interlocuteur minimisera le temps de complétion des *clusters* contactés dans le futur.

ECEF-LA*T*

Une contrepartie de la technique précédente est que cette stratégie a tendance à favoriser les *clusters* rapides, ce qui peut entraîner des retards supplémentaires aux *clusters* plus lents, reléguées aux dernières places. Pour éviter une telle situation, nous proposons une nouvelle fonction de *lookahead*, où le choix des *clusters* considère le maximum du temps nécessaire à la transmission et à la diffusion d'un message :

$$F_j = \max_{P_k \in B} (g_{j,k}(m) + L_{j,k} + T_k)$$

Malgré sa similarité avec l'heuristique précédente, cette nouvelle fonction d'évaluation cherche à équilibrer le temps de communication vers les différents *clusters*, lentes ou rapides. En effet, nous cherchons dans un premier instant le *cluster* la plus lente qu'il reste à contacter (la fonction de *lookahead*), et parmi les choix d'émetteurs disponibles, nous choisissons celui qui peut la contacter le plus rapidement possible (fonction d'évaluation *min* de l'heuristique ECEF).

Le raisonnement de cette heuristique est que si les *clusters* les plus distants ou les plus lents (dans le sens où la diffusion des messages prend plus de temps) sont contactés en dernière place, leur diffusion prendra encore plus de retard, ce qui augmentera le temps d'exécution du *Broadcast*. Avec la fonction de *lookahead* de ECEF-LA*T*, nous choisissons comme récepteur le *cluster* qui prendra le moins de temps possible pour contacter le *cluster* le plus lent : cela garantit que si besoin est, les *clusters* les plus lents seront contactés dans le minimum de temps possible.

BottomUp

La troisième heuristique proposée dans ce travail utilise une logique d'optimisation différente de celle utilisée par Bhat. En effet, l'approche de Bhat vise toujours la minimisation des facteurs liés à la transmission des messages et à sa diffusion, ce qui généralement finit par donner priorité aux *clusters* les plus rapides. Cependant, nous considérons que le temps d'exécution d'un *Broadcast* hiérarchique dépend surtout des *clusters* les plus lents.

À partir des heuristiques précédentes, nous observons que, malgré l'utilisation de différentes fonctions de *lookahead*, les heuristiques de type ECEF-LA suivent toujours l'approche *min-max* ou *min-min*. Or, l'heuristique ECEF-LA*T* considère que parfois il est plus intéressant d'envoyer les messages d'abord aux *clusters* les plus lents, pour ne pas retarder encore plus leur diffusion. D'autre part, il est aussi vrai qu'un grand nombre d'émetteurs favorise la conclusion rapide du *Broadcast*, et que l'envoi à des *clusters* plus lents n'aide guère à augmenter le nombre d'émetteurs. Si ces deux raisonnements sont a priori opposés, ils ne sont pas incompatibles. En effet, les deux approches peuvent être combinées si des règles précises sont déterminées.

C'est ainsi que dans l'heuristique *BottomUp* nous définissons initialement une approche de type *max-min*, où l'émetteur est choisi parmi les *clusters* qui pourront contacter le plus rapidement possible le *cluster* le plus lent du réseau :

$$\max_{P_j \in B} (\min_{P_i \in A} (g_{i,j}(m) + L_{i,j} + T_j))$$

En effet, cette approche permet que les *clusters* les plus lents soient contactés dans le plus petit temps possible, ce qui peut minimiser le retard imputé à ces *clusters*. Toutefois, cette technique n'offre aucune garantie sur l'efficacité future des *clusters* émetteurs. Pour cela, il serait peut-être intéressant d'ajouter une fonction de *lookahead*, à l'exemple des heuristiques de type ECEF-LA.

1.2.3 Évaluation pratique

Les différentes heuristiques ont été implémentées sur une version modifiée de la bibliothèque MagPIe[88], que nous avons adapté pour l'acquisition et la manipulation des paramètres de communication entre les *clusters*. Cette procédure de découverte de topologie permet non seulement le regroupement des nœuds en *clusters* logiques homogènes (plus adaptées à la modélisation de performance), mais aussi fait automatiquement l'acquisition des paramètres pLogP correspondant à chaque sous-réseau homogène. Ces paramètres pLogP, une fois chargés en mémoire, sont associés à la structure hiérarchique du réseau. Ils sont utilisés pour prédire la performance des communications et pour choisir les meilleures stratégies selon les caractéristiques des réseaux.

Pour cette validation nous avons utilisé 88 machines réparties entre les *clusters* d'Orsay, Toulouse et Grenoble. Nous avons utilisé 60 machines du *cluster* d'Orsay, 20 machines du *cluster* de Toulouse et 8 machines du *cluster* de Grenoble. Après la découverte de la topologie du réseau, ces machines ont été regroupées en 6 *clusters* homogènes différentes comme indiqué par le Tableau 1.3.

La Figure 1.5 présente ainsi les temps de communication de chacune de ces stratégies. Le premier résultat à noter est la faible performance de la stratégie *Flat*, même par rapport à l'implémentation par défaut de MPI. Cela ne veut pas dire que la stratégie *Flat* est toujours moins performante que les autres stratégies, mais indique que cette stratégie est trop dépendante de la configuration du réseau, de l'ordre de représentation des *clusters* et du nœud racine.

TABLE 1.3 : Latence entre les différents *clusters* (en microsecondes)

	Cluster 0	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
	31 x Orsay	29 x Orsay	6 x IDPOT	1 x IDPOT	1 x IDPOT	20 x Toulouse
Cluster 0	47.56	62.10	12181.52	12187.24	12197.49	5210.99
Cluster 1	62.10	47.92	12181.52	12198.03	12195.22	5211.47
Cluster 2	12181.52	12181.52	35.52	60.08	60.08	5388.49
Cluster 3	12187.24	12198.03	60.08	0*	242.47	5393.98
Cluster 4	12197.49	12195.22	60.08	242.47	0*	5394.10
Cluster 5	5210.99	5211.47	5388.49	5393.98	5394.10	27.53

* ce *cluster* contient une seule machine.

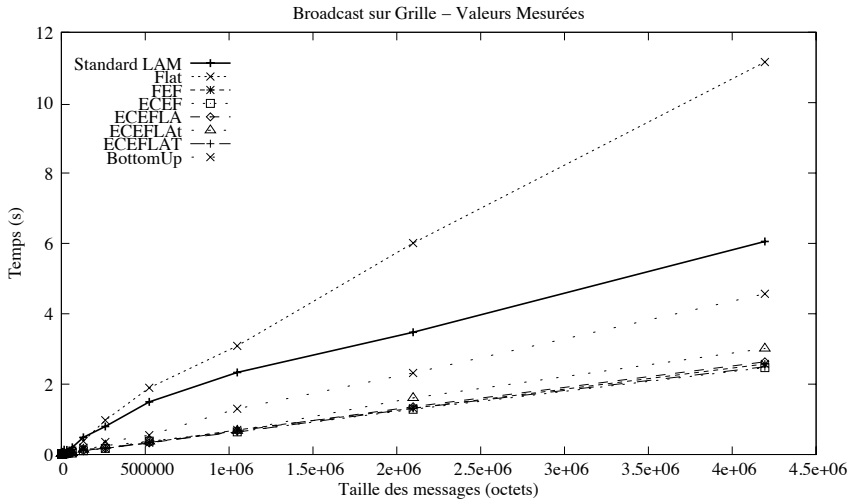

FIGURE 1.5 : Performance du *Broadcast* sur un *grid* de 88 machines

Dans le cas des autres heuristiques, on observe des gains de performance déjà très importants. L'heuristique *BottomUp* n'est pas aussi efficace que les autres heuristiques, qui de leur côté, se comportent de manière très similaire.

Le faible écart observé entre les prédictions des heuristiques de type FEF et ECEF-* est justifié surtout par le nombre réduit de *clusters*, qui réduit le nombre de combinaisons possibles et fait converger les résultats des différentes heuristiques.

Pour mieux valider les résultats des expériences, la Figure 1.6 présente les temps prévus des différentes heuristiques. Ces temps, calculés automatiquement par les heuristiques d'ordonnancement des communications, donnent une meilleure indication de la fiabilité des modèles par rapport aux résultats pratiques. Dans ce cas, nous observons que les heuristiques de type FEF et ECEF-* ont des résultats très rapprochés, certainement parce qu'elles ont obtenu le même ordonnancement des communications. D'un autre côté, l'écart entre ces prédictions et les résultats réels sont bien plus importants pour les heuristiques FEF et ECEF-* que pour le *BottomUp* ou le *Flat*. Cela indique que le coût du calcul de l'ordonnancement et le coût de la mise en œuvre de ces communications sont les facteurs les plus importants, et reflètent l'augmentation de complexité d'une communication à couches multiples.

Cette étude met en évidence l'importance du nœud racine et de la répartition des nœuds sur des différents *clusters* sur la performance des stratégies plus simples. En effet, la performance de la stratégie *Flat* est fortement liée à l'ordre des *clusters*, généralement fournie par l'utilisateur. De surcroît, la stratégie *Flat* utilise toujours le même ordre de diffusion, indépendamment du rang du nœud racine. Au contraire, les heuristiques les plus élaborées construisent

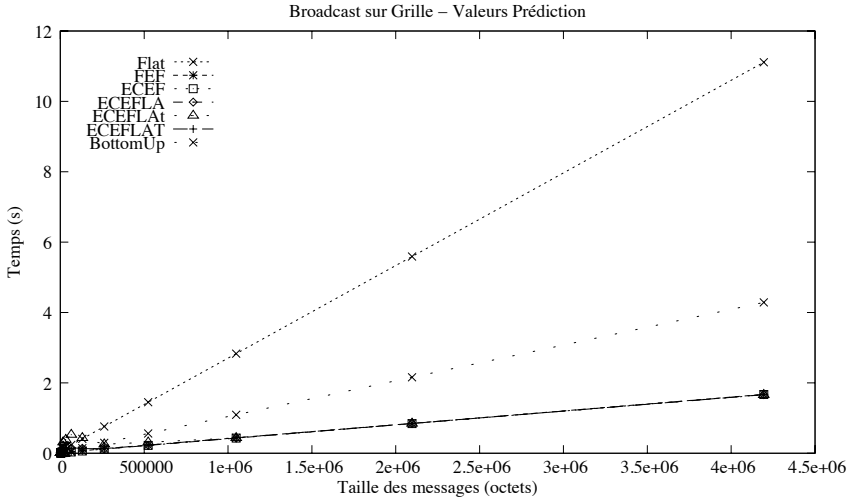

FIGURE 1.6 : Prédictions pour un *grid* avec 88 machines

des arbres de diffusion adaptés à chaque situation, ce qui rend possible un gain de performance plus important, surtout quand le rôle de *racine* est alterné entre plusieurs nœuds.

D'ailleurs, les simulations indiquent que la performance des stratégies simples, comme le *Flat*, supportent très mal l'augmentation du nombre de *clusters* interconnectés. Ainsi, nous croyons que, même si le *grid* compte un nombre réduit de *clusters*, l'utilisation d'une heuristique un peu plus élaborée, comme par exemple l'heuristique ECEFLA-T, offre le meilleur rapport coût-bénéfice-robustesse.

1.3 Amélioration de la Performance de MPI_AlltoAll dans un *Grid*

Si les stratégies présentées dans les sections précédentes permettent l'optimisation des communications dans le cadre d'une diffusion MPI_Bcast, d'autres patrons de communication encore plus couteux sont utilisés par les applications scientifiques. L'une de ces patrons, le *many-to-many*, représente un *échange total* d'informations entre les nœuds [38]. Plus exactement, chaque nœud détient n items de données différentes de taille m, lesquels doivent être distribués entre n nœuds (le nœud inclus).

Vu la complexité de l'opération, les implémentations de *many-to-many* dans la bibliothèque MPI (appelées "*All-to-All*" par celle-ci) généralement utilisent des envois directs entre les nœuds, ce qui peut occasionner la surcharge des communications et des problèmes liés à la congestion du réseau. De surcroît, l'utilisation de cette approche dans un environnement de type grille doit aussi faire face à l'hétérogénéité des temps de communication entre les nœuds proches et distants.

1.3.1 Définitions

Soit deux *clusters* $\mathcal{C}_1$ et $\mathcal{C}_2$ avec respectivement n_1 nœuds et n_2 nœuds. Un réseau, appelé *backbone*, interconnecte les deux *clusters*. Nous assumons aussi que l'interface réseau utilisée pour la communication avec un réseau distant est la même utilisée pour la communication avec

le réseau local. De ce fait et de la topologie du réseau, les communications *inter-cluster* ne seront jamais plus rapides que celles effectuées à l'intérieur d'un *cluster*.

Maintenant supposons qu'une application doit échanger des données entre les machines dans $\mathcal{C}_1$ et $\mathcal{C}_2$ et que pour chaque source, les données à destination des autres machines ne sont pas identiques (mais ont toutes une taille m). Comme résultat, nous devons transmettre l'équivalent à $(n_1+n_2)^2$ messages différents. Alors que plusieurs bibliothèques MPI (OpenMPI, MPICH2, etc.) implémentent l'opération *MPI_AllltoAll* en supposant que les nœuds se trouvent dans le même réseau et donc que les communications ont le même coût, dans notre cas certains messages sont envoyés entre nœuds d'un même *cluster* et d'autres entre nœuds de réseaux différents. Dans le premier cas, la latence des communications est plus favorable que dans le deuxième, et souvent le même principe s'applique pour le débit du réseau.

1.3.2 Modélisation de la Congestion du Réseau

La communication intensive engendrée par les opérations de type *alltoall* peut facilement saturer le réseau ou les destinataires, dégradant ainsi la performance globale de l'opération. Chun [39] a démontré que le coût des communications collectives est souvent dominé par le coût de la congestion du réseau et des subséquentes pertes de paquets.

Malheureusement, plusieurs modèles de communication tels que [38] ou [116] ne prennent pas en compte les impacts potentiels de la congestion. En effet, ces travaux considèrent que l'opération All-to-All n'est qu'une exécution parallèle de plusieurs opérations de type *personalized one-to-many* [84]. Cela s'exprime par le modèle de performances linéaire représenté en Équation 1.1, où α correspond à la latence de communication entre les nœuds, $\frac{1}{\beta}$ correspond au débit du lien, m représente la taille des messages en octets et n correspond au nombre de nœuds :

$$T = (n-1) \times (\alpha + \beta m) \tag{1.1}$$

Comme ce modèle linéaire n'est pas capable de prendre en compte la congestion des communications, certains auteurs tels que Bruck [23] et Clement *et al.* [42] suggèrent l'utilisation d'un facteur de ralentissement (*slowdown factor*) proportionnel au nombre de nœuds communicants. Labarta et *al.* [94] aussi suggère l'utilisation d'un facteur de ralentissement basé sur le nombre de messages en transit car, selon eux, si *m* messages sont prêts à être transmis mais seulement *b* canaux sont disponibles, alors les transmissions sont sérialisées en $\lceil \frac{m}{b} \rceil$ vagues de communication.

Une approche légèrement différente a été proposée par Chun [39], qui associe la congestion à la latence et donc utilise différentes valeurs de latence selon la taille des messages. Cependant, ce modèle ignore le nombre de messages circulant sur un réseau ou un lien, alors et ceci est directement lié au phénomène de la congestion.

Afin de mieux représenter l'impact de la congestion sur les communications collectives de type All-to-All dans les *clusters*, nous avons proposé en [137] un modèle où la congestion du réseau est dépendant des caractéristiques de l'environnement physique (interfaces réseaux, liens, switches) mais aussi de la taille des messages. En effet, nous avons observé expérimentalement que les temps de communication varient selon la taille des messages, et que le facteur de contention γ doit aussi prendre cela en compte.

Ainsi, notre modèle associe le facteur de contention γ de Clement et *al.* [42] avec un nouveau paramètre δ, ce dernier dépendant du nombre de nœuds et de la taille des messages, comme indiqué ci-dessous. Comme résultat, nous associons deux équations (l'une linéaire, l'autre affine) afin de mieux représenter la performance de communication de MPI_Alltoall dans un

réseau donné, comme illustré en Figure 1.7.

$$T = \begin{cases} (n-1) \times (\alpha + m\beta) \times \gamma & if\, m < M \\ (n-1) \times ((\alpha + m\beta) \times \gamma + \delta) & if\, m \geq M \end{cases} \tag{1.2}$$

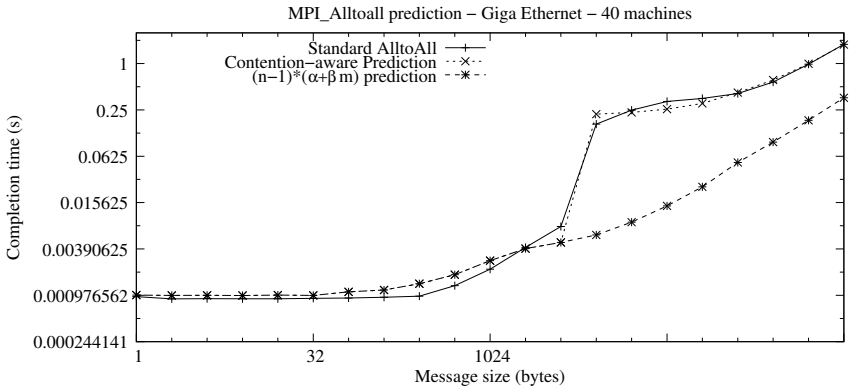

FIGURE 1.7 : Performance mesurée et modélisée pour le MPI_Alltoall dans un réseau Gigabit Ethernet

Vu la précision de ce modèle pour les *clusters* homogènes, notre première réaction serait de l'appliquer aussi dans le cas des *grids* de calcul en faisant la somme des performances des réseaux locaux et distants (Équation 1.3). Malheureusement cette stratégie n'aboutit pas car les performances observées sont largement inférieures à celles prévues par ce modèle, comme indique la Figure 1.8.

$$T = max(T_{\mathcal{C}_1}, T_{\mathcal{C}_2}) + max(n_1, n_2) \times (\alpha_w + \beta_w \times m) \tag{1.3}$$

En effet, la figure 1.8 représente des mesures effectuées entre deux *clusters*, l'un situé à Nancy et l'autre à Rennes. Les deux *clusters* étaient constitués de machines similaires (dual Opteron 246, 2 GHz) interconnectées localement par un réseau Gigabit Ethernet, alors que le lien *inter-cluster* était un réseau privé de 10 Gbps. Les mesures ont été obtenues selon la méthode *broadcast-barrier* [44].

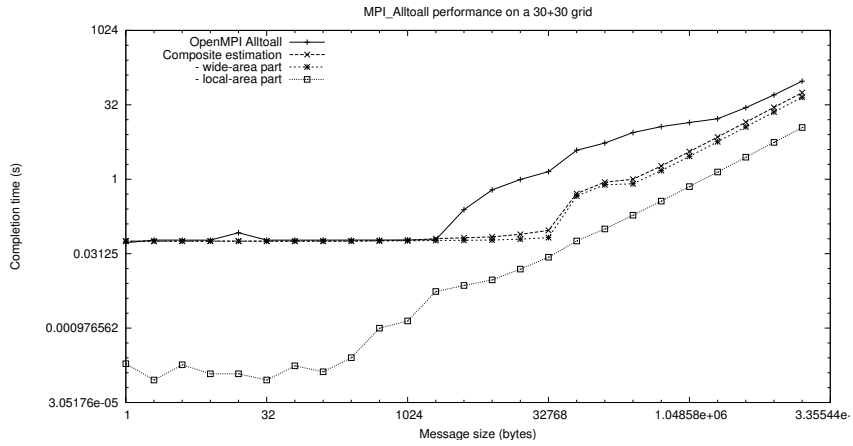

FIGURE 1.8 : Performance mesurée et modélisée pour le MPI_Alltoall dans un *grid*

Bien qu'on pourrait rajouter des paramètres supplémentaires pour s'approcher des performances observées, nous avons décidé d'attaquer le problème directement à sa source en optimisant la façon dont les communications sont effectuées sur un *grid*.

1.3.3 All-to-All pour les *Grids* : l'algorithme LG

Lors de l'opération dans un *grid*, l'un des facteurs les plus importants à prendre en compte est le temps nécessaire à ce que les messages soient livrés car ceux-ci sont affectés par la distance géographique mais aussi par l'hétérogénéité des protocoles, le routage des messages et les interférences des autres flux rencontrés sur le backbone.

La plupart des algorithmes pour les communications collectives sur les *grids* (PACX MPI [62], MagPIe [86]) visent la minimisation des communications à grande distance en choisissant un coordinateur dans chaque *cluster* qui sera responsable par les échanges *intra-cluster*. Bien que ce mécanisme présente des avantages pour d'autres patrons de communication, il n'est pas adapté aux opérations de type MPI_Alltoall. En premier lieu, ce mécanisme induit des étapes de communication supplémentaires du fait de forcer le passage par les coordinateurs des *clusters*, qui de plus deviennent des goulots d'étranglement. En deuxième lieu, cette approche n'est pas optimale par rapport à l'utilisation des liens *inter-cluster*, souvent capables de supporter des flux multiples [25].

Afin de mieux traiter ce problème, nous essayons de minimiser les échanges distants d'une autre manière. En effet, la grande complexité des échanges du All-to-All réside dans les différentes performances des liens locaux et distants, or les implémentations traditionnelles de MPI_Alltoall sont incapables de faire la différence. Si nous pouvons identifier la disposition des nœuds, on peut utiliser toutes les machines d'un *cluster* pour collecter les données à un niveau local avant de les envoyer simultanément au réseau distant, en une unique étape de communication.

Ainsi, le mécanisme que nous avons proposé en [82] est une solution "*grid aware*" qui s'exécute en deux étapes. Dans un premier moment, seulement les échanges locaux sont effectués. Cette phase inclut les échanges attendus entre les nœuds *intra-cluster* mais aussi l'échange des données destinés aux autres *clusters*, stockés dans des buffers supplémentaires pour la deuxième phase. L'avantage de cet échange est que son coût est très faible par rapport au coût d'une communication distante (voir Figure 1.8). Finalement, lors de la deuxième phase, les buffers sont transmis directement aux nœuds destinataires, complétant ainsi l'échange total.

Plus exactement, l'algorithme peut être décrit comme suit. Sans perte de généralité, considérons un *cluster* $\mathcal{C}_1$ qui contient moins de nœuds que le *cluster* $\mathcal{C}_2$ ($n_1 \leq n_2$). Les nœuds sont ainsi numérotés de 0 à $n_1 + n_2 - 1$, avec les nœuds allant de 0 à $n_1 - 1$ appartenant à $\mathcal{C}_1$ et les nœuds allant de n_1 à $n_1 + n_2 - 1$ appartenant au *cluster* $\mathcal{C}_2$. Nous appelons ainsi $\mathcal{M}_{i,j}$ le message (donnée) qui sera envoyé du nœud i au nœud j.

L'algorithme suit les deux étapes :

Première étape Pendant cette phase, nous effectuons les échanges locaux : Le nœud i envoie $\mathcal{M}_{i,j}$ au nœud j, seulement si i et j font partie du même *cluster*. Ensuite, ils préparent les buffers pour les communications distantes de manière à ce que les données à destination d'un nœud distant j on $\mathcal{C}_2$ seront d'abord stockés sur le nœud $j \ mod \ n_1$ de $\mathcal{C}_1$. De même, les données sur un nœud i de $\mathcal{C}_2$ à destination de j sur $\mathcal{C}_1$ seront stockés sur le nœud $\lfloor i/n_1 \rfloor \times n_1 + j$.

Deuxième étape Pendant cette deuxième étape, seulement n_2 communications *inter-cluster* auront lieu. Cette phase est décomposée en $\lceil n_2/n_1 \rceil$ vagues avec n_1 communications chacune. Ainsi, pendant la vague s, le nœud i de $\mathcal{C}_1$ échangera son buffer local avec le nœud $j = i + n_1 \times s$ de $\mathcal{C}_2$ (si $j < n_1 + n_2$). Plus exactement, i envoie $\mathcal{M}_{k,j}$ à j où $k \in [0, n_1]$ et j envoie $\mathcal{M}_{k,i}$ à i où $k \in [n_1 \times s, n_1 \times s + n_1 - 1]$.

Comme cet algorithme minimise le nombre de communications *inter-cluster* et les organise par vagues, nous n'avons besoin que de $2 \times \max(n_1, n_2)$ messages dans les deux directions (par rapport à $2 \times n_1 \times n_2$ messages dans l'algorithme traditionnel). Si les deux *clusters* ont le même nombre de nœuds, une seule vague d'échanges sera nécessaire. Notre algorithme est aussi optimisé sur la longue distance car il regroupe plusieurs messages ensemble, réduisant l'impact de la latence et des interférences sur le backbone. La Figure 1.9 présente une comparaison entre la performance de l'algorithme traditionnel implémenté par OpenMPI et celle de l'implémentation de l'algorithme $\mathcal{L}G$. Nous observons que l'algorithme $\mathcal{L}G$ obtient un gain de performance de presque 50% par rapport à la stratégie traditionnelle.

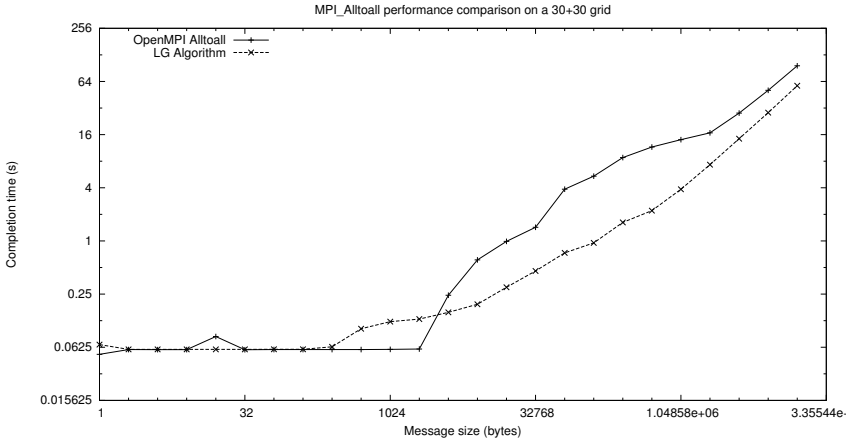

FIGURE 1.9 : Comparaison de performance entre OpenMPI et l'algorithme $\mathcal{L}G$

L'ordonnancement des communications effectué par $\mathcal{L}G$ a aussi une conséquence sur la modélisation des performances. Tout d'abord, il minimise les communications de longue distance, réduisant les risques de congestion qui sont difficiles de modéliser. De plus, la transmission groupée des messages réduit l'impact de la latence et des fluctuations de performance du réseau. C'est ainsi que nous avons pu établir un modèle composé des performances locales ($\mathcal{T}_{C_n}$, obtenues à partir de l'équation 1.2) et des prédictions pour les communications de longue distance obtenues par les méthodes traditionnelles :

$$T = max(T_{\mathcal{C}_1}, T_{\mathcal{C}_2}) + \lceil n_2/n_1 \rceil \times (\alpha_w + \beta_w \times m \times n_1) \tag{1.4}$$

Afin d'obtenir les paramètres nécessaires aux prédictions, nous avons utilisé la procédure décrite par Kielman *et al.* [87]. Les facteurs de contention $\gamma = 2.6887$ et $\delta = 0.005039$ pour $M >= 1KB$ ont été obtenus par la méthode des moindres carrés comme décrit par [137].

La Figure 1.10 compare ainsi les prédictions obtenues avec l'Équation 1.4 et les performances mesurées pour l'algorithme $\mathcal{L}G$. Nous observons une bonne adéquation des prédictions, ce qui était impossible avec l'implémentation traditionnelle de MPI_Alltoall.

1.4 Bilan et Perspectives

Les travaux présentés dans ce chapitre prouvent l'intérêt de la modélisation des communications dans les réseaux hétérogènes, autant pour la prédiction des temps de communication que pour l'optimisation des algorithmes existants. Ceci est démontré notamment dans le cas de l'algorithme LG, qui a vu le jour uniquement parce que l'observation et la modélisation des performances indiquait des situations non-optimales dans les échanges entre deux *clusters*.

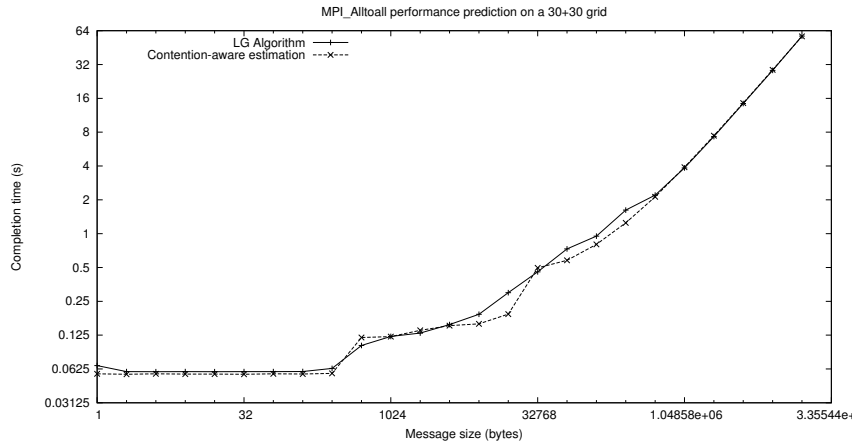

FIGURE 1.10 : Performance de $\mathcal{L}G$ et sa prédiction

Au fil des années mes intérêts se sont diversifiés et la modélisation des performances telles que présentées dans ce chapitre ne sont plus si fréquentes. Dans la plupart des cas mes travaux visent la comparaison des performances entre différentes approches logicielle et environnements d'exécution, ou bien l'amélioration des performances des applications grâce à la parallélisation de certaines routines. Ceci est le cas de deux sujets de recherche et expérimentation que j'ai démarré récemment :

Utilisation de GPUs pour Augmenter la Performance d'une Application de *Data Mining*

Dans ces travaux effectués en collaboration avec Andrea Charão et Tiago Engel (Universidade Federal de Santa Maria, Brésil), nous nous sommes intéressés à l'optimisation des performances d'une application de *data mining* et *machine learning* très connue, Weka [1].

Bien que les termes *data mining* et *machine learning* soient fréquemment associés au *big data*, on retrouve encore un nombre assez important d'applications et outils qui ne sont pas parallélisés ni s'exécutent sur un *cluster* ou une infrastructure hébergée sur *cloud*. Le plus souvent ceci est dû au fait que les masses de données ne sont pas tellement importantes pour avoir recours à des infrastructures plus puissantes, ou tout simplement car des instances moins importantes des données sont utilisées pour explorer et étudier un problème avant son déploiement.

Weka est un environnement très connu des chercheurs en *data mining*, souvent utilisé pour l'apprentissage des techniques de classification, régression, clusterisation, etc. Weka est aussi une bibliothèque qui peut être intégrée aux application.

Dans le cadre de ce travail nous avons constaté que la plupart des opérations sur Weka ne sont pas parallélisés (ni sur les multiples coeurs de la CPU, ni sur un accélérateur GPU). En choisissant les bonnes routines, nous croyons pouvoir améliorer sensiblement la performance de cette plateforme.

Grâce à l'étude des profils d'exécutions lors de l'exécution d'une application métier (étude d'images de mammographie afin d'identifier des possibles sites tumoraux), nous avons pu déterminer un petit nombre d'opérations qui consommaient la plupart du temps de calcul. Ainsi, nous avons étudié le remplacement de ces opérations par des versions multi-coeur et GPU, ayant par résultat une réduction de plus de 50% du temps d'exécution de l'application [52, 53].

1. `https://www.cs.waikato.ac.nz/ml/weka/`

En ce moment nous poursuivons ces collaborations avec l'étude des performances d'accès aux services cloud [34] ou en associant des étudiants, comme par exemple les deux articles publiés récemment au Brésil ([109, 107]) et conduits dans le cadre du projet de collaboration international CAPES-Cofecub MESO.

Évaluation des Performances de la Virtualisation sur les Dispositifs SoC (*System on a Chip*)

L'analyse de performance ne se limite pas à l'exécution ou aux communications d'une application. Souvent, on doit évaluer les capacités d'un dispositif afin de pouvoir établir ses possibilités et ses limitations. Un sujet auquel je me suis consacré récemment est celui de l'analyse de performance des nano-ordinateurs de type SoC (System on a Chip), comme par exemple les Raspberry Pi, Banana Pi, etc. Ces dispositifs sont fréquemment utilisés en tant que *gateway* (passerelle) entre les dispositifs IoT installés dans une maison ou bâtiment et le *cloud*, responsable par le stockage et le traitement des données. Malgré leur faible puissance de calcul, il est tout à fait envisageable d'utiliser les nano-ordinateurs pour l'exécution de certaines tâches.

Dans ce cas précis, je me suis intéressé par l'analyse de performance des machines virtuelles de type conteneur. Des conteneurs LXC ou Docker rencontrent un grand succès et plusieurs travaux pointent sur leur usage dans le cadre du déploiement de micro-services.

Ainsi, des campagnes de mesure et d'analyse des performances ont été effectuées l'année dernière, donnant lieu à deux publications dans des conférences internationales [17, 16]. Ces travaux, effectués dans le cadre d'une collaboration avec David Beserra, doctorant à l'Université Paris 1, pourront à terme être intégrés au développement d'une plateforme de calcul distribué que je présenterai plus en détails en Chapitre 5.

Chapitre 2

L'Hétérogénéité des Tâches de Calcul

Résumé

La gestion de l'hétérogénéité peut (et doit) être considérée sous plusieurs aspects. Alors que le chapitre précédent a donné des exemples de travaux dans lesquels l'étude sur l'hétérogénéité s'était concentrée sur les aspects liés à la communication, dans ce chapitre nous nous concentrons sur l'hétérogénéité de calcul. Ce type d'hétérogénéité est souvent le plus simple à traiter car le plus fréquent : il est presque impossible de développer une application parallèle dans laquelle les tâches de calcul sont parfaitement régulières. Bien, sûr, la gestion des tâches régulières ou irrégulières dépend de leur interdépendance. Plus elles sont découplées, moins on trouve de contraintes pour leur exécution, ce que simplifie leur gestion.

Dans ce chapitre nous considérons que les tâches de calcul sont hétérogènes si les différentes tâches d'une application présentent des variabilités dans leurs temps d'exécution à cause des facteurs propres à chaque tâche : variations dans le volume ou la nature des données à traiter, variations de la complexité des opérations à effectuer, etc. Contrairement au chapitre précédent, nous n'essayons pas de modéliser les performances ni de les prédire, mais nous essayons simplement d'appliquer des mécanismes de gestion des tâches dans le cas de la parallélisation d'une application métier et de son déploiement sur un cluster de calcul.

Ainsi, ce chapitre démarre avec la description d'une application destinée à l'exécution de problèmes d'amarrage moléculaire. Nous devons la rendre parallèle pour mieux la déployer à plus grande échelle, sans toutefois modifier son code source. Les prochaines sections détaillent donc les spécificités de l'application métier et les approches retenues pour découper les instances de calcul en unités pouvant être traitées en parallèle. Ensuite, on présente deux stratégies de gestion des tâches qui ont été implémentées, destinées non seulement à garantir le bon déroulement du calcul et le regroupement des résultats, mais également destinées à mieux utiliser les ressources disponibles dans les environnements de calcul.

Le travail décrit dans ce chapitre a été développé dans le cadre de la codirection de thèse de Romain Vasseur (thèse en bio-informatique, dirigée par le prof. Manuel Dauchez et co-encadré par Stéphanie Baud et moi-même). Cette thèse, effectuée entre 2012 et 2015, était une thèse CIFRE portée par le Laboratoire MeDyC - Matrice Extracellulaire et Dynamique Cellulaire (UMR CNRS 7369), le laboratoire CReSTIC (EA - 3804) et la compagnie Bull-Atos. Ces travaux ont fait l'objet de publication de 2 articles en journaux internationaux, 2 conférences internationales (dont un "Best Paper Award") et 3 communications courtes (posters et résumés).

2.1 Application à la Recherche en Amarrage Moléculaire

L'utilisation d'approches informatiques pour identifier les interactions biomoléculaires est devenu l'un des principaux piliers de la recherche de nouvelles drogues et principes actifs. En effet, la simulation *in silico* permet de faire une première prospection sur un grand nombre de candidats potentiels, tout avec un gain de temps important et un coût nettement moins onéreux que l'expérimentation *in vitro*.

L'amarrage moléculaire (aussi appelé *docking moléculaire*) est donc une technique qui vise à étudier les interactions au niveau moléculaire entre certaines structures du vivant, comme par exemple les interactions protéine-ADN/ARN, protéine-protéine, peptides-protéine, protéine-ligand ou glucide-protéine. L'industrie pharmaceutique s'intéresse particulièrement à l'étude des interactions protéine-ligand, notamment la recherche de principes actifs de médicaments (ligands) qui puissent se connecter à certaines protéines cible. La prédiction des modes d'amarrage d'un ligand à une protéine, la structure du complexe résultant et l'estimation de l'affinité de cet amarrage sont essentiels pour le développement de nouveaux composés thérapeutiques, et les méthodes numériques ont été le choix principal de plusieurs travaux dans la littérature [1, 65, 91].

Souvent cette étude se fait à travers un "criblage virtuel" (*virtual screening*), qui consiste en un déploiement a grande échelle permettant de tester un grand nombre de ligands (de centaines a plusieurs millions selon l'ampleur de la campagne) sur un nombre très restreint de cibles. En effet, nous trouvons des millions de composants catalogués dans des bases de données telles que la Cambridge Structural Database [3], PDBbind [153, 154], ZINC [79] et tant d'autres collections privées des groupes pharmaceutiques. De même, un riche catalogue de protéines peut être obtenu à partir du Research Collaboratory for Structural Biology (RCSB) Protein Data Bank (PDB) [120], une base de données ouverte qui contient plus de 120 000 protéines cataloguées et qui est enrichie de plus de 7000 nouvelles protéines par an. Ainsi, un chercheur ou un industriel qui souhaite activer ou désactiver une protéine afin de combattre une maladie peut donc effectuer ce criblage virtuel entre la protéine cible et les milliers de ligands catalogués (enzymes, peptides, etc.).

Dans le cadre des travaux de thèse de Romain Vasseur nous nous sommes penchés sur le développement et l'exécution parallèle d'une application pour le criblage moléculaire inversé. Plus exactement, un criblage inversé a pour objectif de discriminer les cibles protéiques les plus favorables a une interaction avec le ligand, parmi un échantillon de structures de protéines plus ou moins important. De cette manière, il est possible d'identifier des cibles secondaires pour un ligand développé, ou bien faire une étude préalable des risques d'interactions indésirables.

2.1.1 Travaux Proches et Méthodologie de Parallélisation

Le terme *inverse docking* a fait son apparition dans la littérature en 2001 avec les deux articles de Chen *et al*. [37, 36]. Une quinzaine d'articles ont été recensés depuis cette date, avec des méthodes de traitement à plus ou moins grande échelle. La plupart de ces travaux font l'usage d'applications pour le docking traditionnel telles que AutoDock [134] ou AutoDock Vina [96, 95], avec très peu d'outils dédiés exclusivement au docking inversé (INVDOCK [37, 36] ou TarFisDock [97]). Toutefois, aucun de ces articles ne décrit ni ne mentionne le développement d'une méthode de déploiement sur des architectures de type HPC.

Ainsi, nous avons développé nos propres stratégies dans le but de pouvoir traiter des centaines en parallèle de protéines grâce aux architectures HPC. Ces stratégies concernent la parallélisation du calcul d'un couple ligand-protéine mais aussi le déploiement large échelle de ces calculs.

Dans ce travail nous sommes parti de la méthode appelée *blind docking* qui consiste à détecter des points d'amarrage possibles en faisant un balayage sur l'ensemble de la surface de la protéine. Pour cela, des applications telles que Autodock doivent générer une grille d'affinités basée sur les énergies de liaison de chaque atome qui compose la protéine. Cette grille d'affinité se présente comme une boîte 3D qui contient toute la surface de la protéine, plus une marge afin de permettre le placement d'un ligand à l'intérieur de la boîte. Par la suite, un algorithme génétique ira explorer le volume autour de la protéine afin d'évaluer l'énergie de liaison à différents endroits.

L'approche *blind docking* est naïve et difficilement parallélisable car chaque pair protéine-ligand représente une boîte et donc une tâche de calcul Autodock. De plus, selon les caractéristiques de la protéine, un nombre important d'itérations (générations dans l'algorithme génétique) sont nécessaires pour couvrir systématiquement la surface de la protéine et permettre l'obtention de résultats satisfaisants. Comme résultat, l'exécution d'une seule instance du *blind docking* avec l'application AutoDock peut prendre plusieurs heures. Malgré ces contraintes, cette approche est assez répandue et a donc été utilisée comme référence de comparaison pour les approches parallèles que nous avons développées.

2.2 Parallélisme Intérieur : décomposition des tâches

Afin d'optimiser l'utilisation des ressources de calcul lors d'une campagne de docking inversé, il est impératif d'intégrer le parallélisme au cœur du traitement des tâches de calcul, comme illustré en figure 2.1. La décomposition d'une instance de docking permet une meilleure utilisation des ressources de calcul en distribuant les tâches sur plusieurs machines, mais aussi grâce à un traitement en pipeline qui permet l'enchaînement des opérations lorsque certaines tâches finissent plus tôt. De plus, cela rend l'exécution plus tolérante aux fautes, une tâche qui a été interrompue peut être redéployée sur une autre machine sans obliger le redémarrage de toute l'exécution. Dans le cadre de ce travail, nous avons étudié les techniques de décomposition présentées dans les sections suivantes.

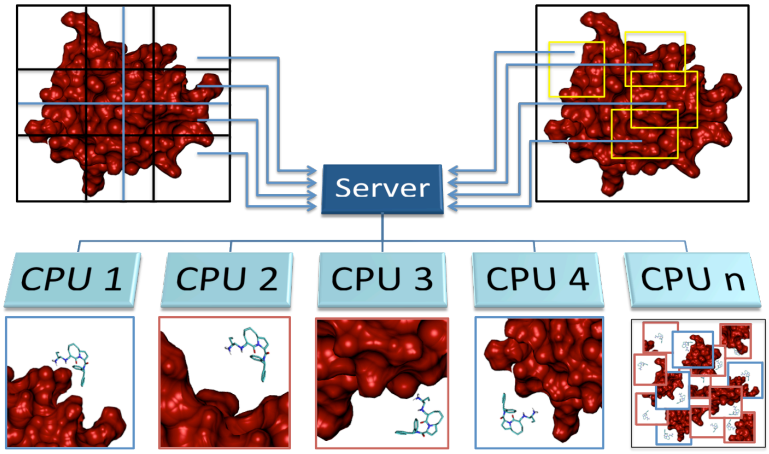

FIGURE 2.1 : Exemple d'un schéma de décomposition parallèle

2.2.1 Décomposition Géométrique Arbitraire

Vu la nature des données utilisées en entrée pour le docking, nous avons initialement étudié une stratégie de décomposition géométrique qui consiste à découper la grille d'affinité 3D en plusieurs boîtes plus petites, chacune couvrant un secteur de la protéine. Cette stratégie considère une décomposition géographique régulière de telle manière que le nombre de tâches (boîtes) ont une taille similaire, permettant ainsi la génération de n^3 sous-grilles : 8 (2x2x2), 27 (3x3x3), 64 (4x4x4), etc. Le choix du bon nombre de découpages dépend à la fois du gain en parallélisme mais aussi de la liberté de mouvement du ligand à l'intérieur d'une grille. En effet, on peut espérer un gain de performance du fait de pouvoir déployer en parallèle les différentes sous-grilles comme des tâches de calcul indépendantes. Toutefois, un nombre trop important de découpages aura pour effet la génération de sous-grilles "inutiles" car elles couvrent que des zones inaptes à la recherche de points d'amarrage (espace non connecté à la surface de la protéine, "intérieur" de la protéine, etc.). De plus, une boîte 3D trop petite peut empêcher le positionnement du ligand et donc rendre l'évaluation de l'amarrage impossible.

Un autre inconvénient de cette technique est que le découpage se fait de manière arbitraire, sans prendre en compte les spécificités de la surface de la protéine. Par exemple, les "cavités" présentes dans la surface de la protéine sont souvent des bons sites pour l'amarrage, mais un découpage arbitraire qui l'ignore peut simplement scinder cette cavité en deux et la rendre bien moins intéressante vis-à-vis de l'algorithme de docking. De même, l'évaluation de docking se fait en considérant que le ligand se trouve totalement à l'intérieur de la sous-grille : n'importe quelle conformation où des atomes ligand dépassent la grille serait invalide et donc ignorée.

2.2.2 Décomposition Géométrique avec Superposition

Les inconvénients de la décomposition géométrique arbitraire cités dans la section précédente nous ont conduit à développer une technique alternative de découpage qui préserve la liberté de placement des ligands et permet une couverture intégrale de la surface de la protéine. Cette technique consiste à effectuer un découpage avec superposition entre les sous-grilles voisines, de manière à pouvoir évaluer le placement du ligand même sur les zones proches des bords des sous-grilles. Bien sûr, cette superposition est dépendante de la taille des ligands, permettant ainsi une configuration qui optimise l'utilisation des ressources pour chaque pair protéine-ligand.

Ainsi, dans le cadre du travail effectué, nous avons considéré deux valeurs de référence pour la superposition. La superposition entre deux boîtes serait d'un tiers de la longueur de la boîte si la longueur du ligand est inférieure à cela. Dans le cas contraire, la superposition correspond à la longueur du ligand. Grâce à cette configuration, le ligand a une liberté complète de placement (rotation, translation, etc.) et on peut effectuer une recherche exhaustive sur l'espace d'amarrage. Dans l'exemple illustré dans les prochaines sections nous utilisons aussi un schéma de décomposition en douze parties, 3x2x2 (où 3 correspond à l'axe principal de la protéine) et avec une superposition d'1/3 sur chaque sous-grille.

2.2.3 Recherche de Cavités

Comme indiqué précédemment, l'amarrage des ligands est favorisé par la présence de cavités dans la surface de la protéine [64, 74], or les méthodes de découpage par décomposition ne prennent pas ces facteurs en compte. En effet, même avec la décomposition avec superposition,

l'algorithme génétique utilisé pour la recherche de points d'amarrage ne fait que parcourir la surface sans un objectif précis.

Nous pouvons améliorer la précision de notre docking inversé en effectuant la détection des zones avec cavités, par exemple à l'aide d'un programme dédié à ce fin, l'application Fpocket [70]. La prise en compte des zones avec un plus grand potentiel peut augmenter la performance du docking inversé car cela nous permet de concentrer la recherche sur une zone plus spécifique. L'inconvénient est que son application nécessite un réglage fin des paramètres afin d'inclure les spécificités des protéines et de ne pas exclure des zones avec un potentiel moindre mais réel.

Ainsi, au lieu de se reposer uniquement sur la recherche de cavités, notre travail a misé sur la complémentarité entre celle-ci et la décomposition géométrique avec superposition. En plus de générer des tâches de calcul pour les différentes sous-grilles issues de la décomposition géométrique, nous générons aussi des recherches ciblées sur les cavités identifiées par FPocket. Ces paramètres permettent une meilleure couverture des zones avec un plus grand potentiel (car couvertes par les deux techniques), tout en limitant le nombre de tâches de calcul supplémentaires.

2.3 Gestion et Déploiement des Tâches de Calcul

Les techniques de découpage présentées dans la section précédente permettent la parallélisation du traitement d'un couple protéine-ligand. Dans le cas du docking inversé, ce parallélisme dit "interne" doit être associé à la génération et au traitement des multiples tâches de calcul issues de chaque combinaison entre un ligand cible et la base de données de protéines recherchée. Enfin, les tâches de préparation des données (génération des sous-grilles, définition des paramètres, regroupement des résultats, etc.) doivent aussi être prises en compte.

Il faut noter que la présence de tâches de différentes natures (préparation des grilles, criblage virtuel, détection de cavités) rendent la gestion des tâches un peu moins évident. Même les tâches de nature similaire peuvent présenter des variations selon la nature des données à traiter. Si nous prenons par exemple les tâches de criblage virtuel, le temps d'exécution dépend du degré de liberté des peptides à l'intérieur des boîtes 3D : une boîte découpée "à l'intérieur" de la protéine n'aura aucune surface exploitable et sera rapidement écartée.

Pour toutes ces raisons, nous avons créé deux implémentations visant la gestion et le déploiement des tâches, toutes les deux intégrées à l'outil AMIDE [150]. Dans le premier cas, une plateforme générique qui gère tout seule l'ensemble des tâches a été conçue. Dans le deuxième cas, une partie des responsabilités est déléguée aux gestionnaires de tâches des *clusters*, simplifiant ainsi son exécution dans les environnements déjà pourvus de tels outils.

2.3.1 Plateforme Générique

Vu les besoins de parallélisme interne et externe, nous avons dans un premier moment spécifié et développé une plateforme générique basée sur le langage Python et capable d'exploiter le parallélisme multi-cœur et multi-machine pour le docking inversé.

Ainsi, un ensemble de scripts Python a été créé afin d'automatiser toutes les étapes liées à la préparation et à l'exécution du docking inversé. Parmi ces étapes nous pouvons citer :

(i) l'acquisition des fichiers PDB qui décrivent les protéines et ligands,

(ii) la préparation des fichiers PDB afin de sélectionner les structures cible,

(iii) l'extraction des coordonnées pour la création des grilles d'affinité,

(iv) la décomposition des grilles et
(v) le déploiement des tâches de calcul.

Les étapes (i) et (ii) concernent majoritairement la manipulation de fichiers et le *parsing* des informations, alors que les étapes (iii) et (iv) sont liées à l'exécution de Autogrid, un outil qui fait partie de la suite Autodock et qui permet la création des grilles pour le docking. Selon la stratégie de décomposition retenue, l'étape (iv) peut créer une ou plusieurs grilles correspondant au découpage 3D. Dans le cas où l'on rajoute l'approche par recherche de cavités, il faut rajouter des grilles 3D générées autour des zones identifiées par le logiciel Fpocket.

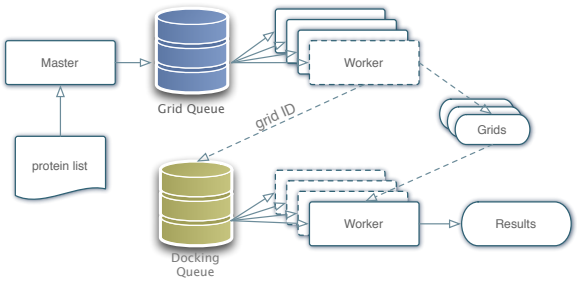

FIGURE 2.2 : Représentation du flot d'exécution dans l'architecture distribuée

Pour cela, la plateforme utilise une architecture distribuée maître-esclave avec gestion d'une file d'exécution contenant les identifiants des tâches (task ID) et accessible en mode "sac de tâches" (*bag of tasks* en anglais). Grâce à cette stratégie, les différents esclaves obtiennent une ou plusieurs tâches à exécuter, selon le nombre de cœurs de calcul disponibles. Pour être plus exacte, dans cette architecture nous trouvons deux files d'exécution, l'une dédiée à la préparation des sous-grilles et l'autre dédiée à l'exécution des tâches de docking. La première file est alimentée par le maître qui, à partir des paramètres d'entrée, indique aux esclaves les différentes protéines à analyser et aussi les stratégies de découpage à mettre en place, ainsi que la génération des sous-grilles avec l'outil Autogrill de la suite Autodock. Les esclaves doivent d'abord finir toutes les tâches dans cette file avant de passer à la file suivante.

Pour plus d'efficacité, la deuxième file d'exécution n'est plus alimentée par le maître mais directement par les esclaves. Lorsque ceux-ci finissent la préparation d'une tâche de la première file, il suffit de déposer le TaskID correspondant dans la deuxième file d'exécution. La Figure 2.2 illustre le flot d'exécution de ces deux étapes.

2.3.2 Optimisation aux Clusters HPC

L'architecture distribuée présentée ci-dessous est adaptée à une exécution sur tout type de réseau d'ordinateurs (*cluster*, infrastructures *cloud*, *grid* pervasif, etc.). Toutefois, il est possible d'optimiser son fonctionnement sur les *clusters* HPC si on prend en compte l'existence d'un gestionnaire de tâches propre à ces systèmes. En effet, la plupart des *clusters* HPC fait l'usage d'un gestionnaire de tâches afin de garantir la réservation et l'équité d'usage des ressources. Des systèmes tels que PBS [73], OAR [24] ou Slurm [163] sont capables de gérer plusieurs files d'exécution ainsi que de déployer des tâches en mode *best effort* de manière à occuper les ressources de calcul lorsque les réservations ne suffisent pas.

Dans le cas de l'optimisation pour les *clusters* HPC notre stratégie a été d'éliminer la dépendance vis-à-vis d'un nœud maître et permettre ainsi l'exécution des tâches indépendantes selon les disponibilités du gestionnaire de tâches. En effet, la présence d'un nœud maître était

nécessaire afin de gérer les files d'exécution mais aussi de vérifier la terminaison des tâches (garantissant la tolérance aux fautes). Cela oblige donc que le processus maître reste actif pendant toute l'exécution, ce qui impose des problèmes lors de la réservation des ressources destinées au maître (combien de temps faut-il le garder actif ?). Cette limitation est encore plus importante dans le cas d'un déploiement *best-effort*, où la terminaison des tâches risque d'être fortement étalée au gré de l'occupation des ressources. Comme les gestionnaires de tâches des *clusters* HPC sont capables de détecter une tâche interrompue et la relancer, il suffit de soumettre dans une même tâche les paramètres nécessaires à la génération des sous-grilles et à leur docking.

Grâce à cette optimisation, l'outil AMIDE issu de ces travaux est capable d'effectuer le docking inversé autant dans un *cluster* que dans un agglomérat de machines indépendantes.

2.4 Évaluation Pratique

Lors de la mise en place de l'outil AMIDE nous avons exécuté plusieurs expériences dans le but de valider notre approche de travail. L'un de ces tests consistait à évaluer la précision des stratégies de décomposition en étudiant un complexe ligand(X23)-protéine(3CM2) bien connu dans la littérature. La comparaison s'est fait par rapport à la technique classique du *blind docking* mais aussi par rapport à des données expérimentales obtenues par cristallographie. Dans un deuxième moment, nous avons conduit des tests de performance dans lesquels le docking inverse était déployé sur un large ensemble de protéines issues de la bibliothèque PDB.

Toutes les expériences ont été conduites sur le *cluster* Clovis du centre de calcul ROMEO à Reims. Clovis était un *cluster* hybride avec 36 nœuds Westmere-EP (12 cœurs), 2 nœuds Nehalem-EX (32 cœurs), un nœud Westmere-EP (12 cœurs + 2 Fermi C2050 GPUs) et un nœud Nehalem-EP (8 cœurs + GPU Fermi M2090), et au moins 2GB de mémoire par cœur. Pour une question de régularité, les expériences ici présentées ont été lancées exclusivement sur les nœuds Westmere-EP. De plus, afin de mieux évaluer la communication *intra-cluster*, seulement 4 cœurs de calcul ont été utilisés par nœud.

Ainsi, pour la première expérience, nous avons initialement exécuté un blind docking de la protéine 3CM2 et du ligand X23. Comme illustré en Figure 2.3, le blind docking a permis la détection de la cavité et d'une conformation presque identique à celle obtenue par cristallographie (différence RMSD de 1.60 Angstroms seulement).

TABLE 2.1 : Tableau comparatif des précisions du docking de 3CM2 selon les stratégies de décomposition et le nombre d'itérations. La distance RMSD est comparée à la pose cristallographique et les énergies de liaison à celles obtenues par la méthode blind docking.

ΔG blind docking ($kcal.mol^{-1}$)	Nombre d'itérations	énergie de liaison ΔG ($kcal.mol^{-1}$)	RMSD (Å)
$\Delta G = -10.27$	20	$\Delta G_{12} = -10.09$	1.79
	50	$\Delta G_{pocket} = -10.11$	1.86
	70	$\Delta G_{pocket} = -10.39$	1.62

Le découpage avec le meilleur score d'énergie de liaison est celui en $n = 12$, comme illustré en Figure 2.4. Avec seulement 20 itérations, ce type de décomposition a permis l'obtention d'un placement similaire à celui de la méthode blind docking (voir Table 2.1). Les autres formats de découpage ont réussi à placer le ligand dans la cavité cible, mais leurs résultats en termes

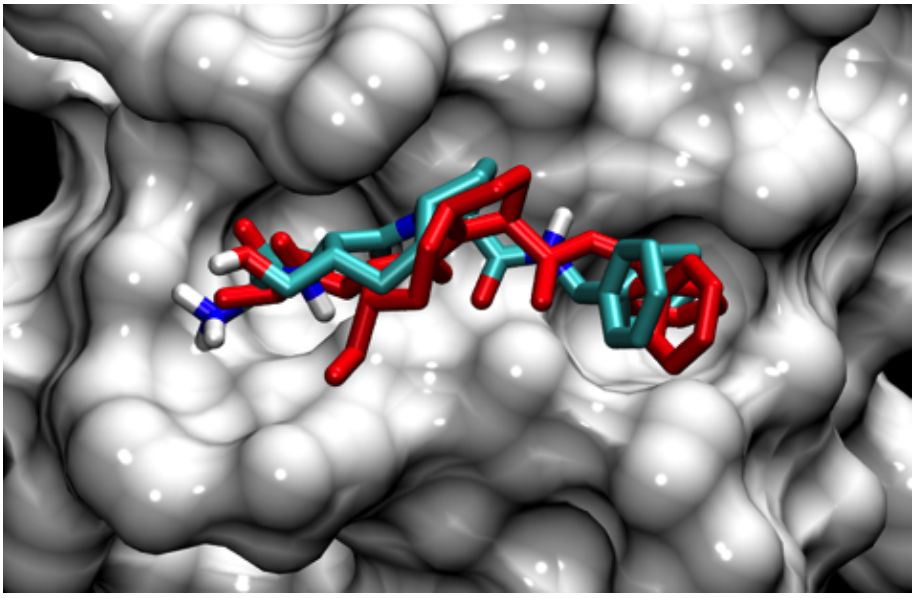

FIGURE 2.3 : Placement du ligand selon la méthode cristallographique (rouge) et par blind docking (vert)

d'énergie de liaison et de distance RMSD ne sont pas suffisants. La méthode de recherche de cavités a aussi permis le placement du ligand avec une bonne précision, au bout de 50 itérations.

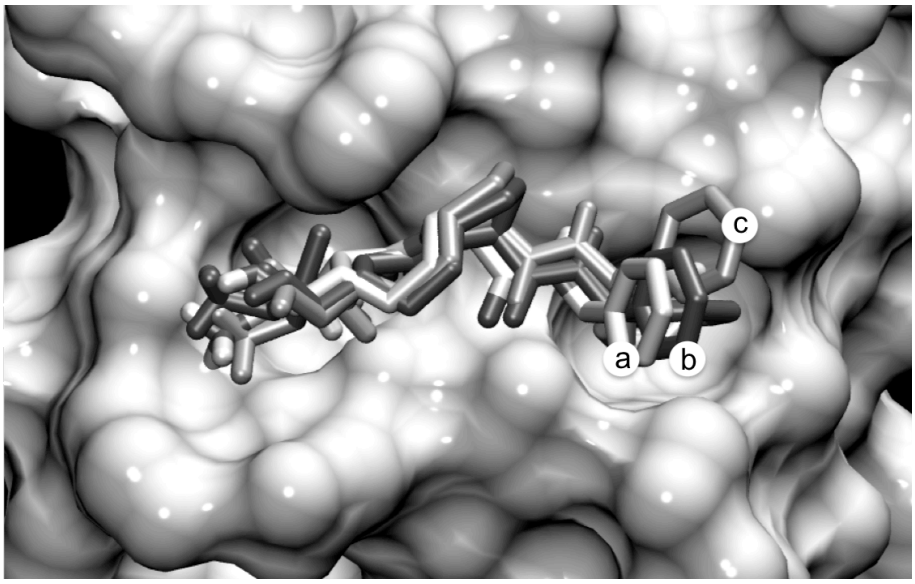

FIGURE 2.4 : Comparaison entre la pose blind docking (a) et celle d'un découpage à $n = 12$ (b) et par recherche de cavités (c).

Dans un deuxième moment nous avons effectué le docking inversé du ligand X23 sur un ensemble de 100 protéines issues de la base PDB. Ici, l'utilisation du découpage a permis une meilleure utilisation des ressources grâce à l'équilibrage de charge mais aussi meilleure prise en charge de la tolérance aux fautes. En effet, la Figure 2.5 affiche la durée moyenne d'exécution d'une tâche selon les différentes méthodes considérées dans ce travail. Ainsi, une exécution de type blind docking nécessitait plus de 5h30, et toute interruption oblige la réexécution complète de la tâche. À l'opposé, l'interruption d'une tâche issue d'une décomposition en 12 parties ne demande qu'une demi-heure de ré-exécution, en cas de défaillance.

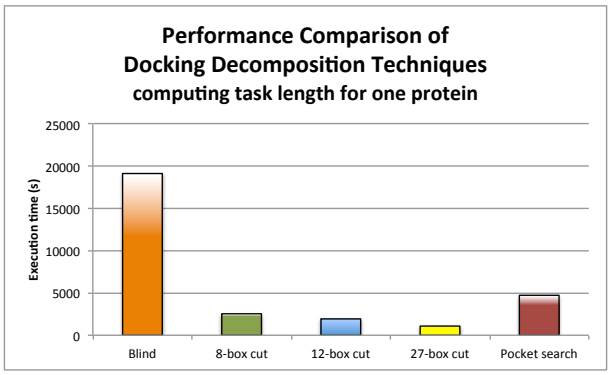

FIGURE 2.5 : Comparaison des temps d'exécution pour les tâches issues des différentes méthodes de découpage

2.5 Bilan et Perspectives

Les travaux présentés dans ce chapitre démontrent l'intérêt des mécanismes de gestion de l'exécution distribuée lorsque les tâches de calcul sont hétérogènes. En effet, c'est l'un des types de hétérogénéité les plus simples à gérer et dont le besoin se fait sentir de plus en plus à cause de la multiplication de ressources parallèles à notre disposition (cœurs de calcul, GPUs, etc.).

L'expérience avec le développement de la plateforme AMIDE a aussi exposé une facette moins connue de la parallélisation des applications : l'étude des stratégies pour la parallélisation lorsqu'on ne peut pas modifier le code source de l'application. En effet, la communauté HPC a souvent tendance à négliger ce type de contrainte alors qu'on est de plus en plus confrontés à des situations où la réécriture de l'application métier est trop complexe.

D'un point de vue performance et volume de calcul, plusieurs autres campagnes de simulation ont été conduites ces dernières années, portant sur d'autres peptides d'intérêt pharmaceutique tels que les ligands BAT et NGH, inhibiteurs du MMP-3, une protéine associée à certaines maladies telles que l'arthrite et la métastase des tumeurs. L'intégration avec le gestionnaire de tâches Slurm du Centre de Calcul ROMEO rend plus simple le déploiement de ces campagnes.

Bien évidemment, ça a été aussi une expérience humaine très enrichissante, du fait de pouvoir travailler avec des chercheurs et un doctorant issus d'autres domaines que l'Informatique. Ceci a affecté non seulement les méthodes et outils de travail (par exemple, le choix des langages de programmation) comme a permis la confrontation de différentes perceptions des mécanismes de publication et de divulgation scientifique. Cette proximité avec des chercheurs d'autres domaines est une spécificité de la recherche HPC à Reims, où des chercheurs en chimie, biologie ou physique côtoient les informaticiens au sein des plateaux techniques de la Maison de Simulation de Champagne-Ardenne (Centre de Calcul ROMEO, Centre Image et P3M - Plateau de Modélisation Moléculaire Multi-échelles). Cette expérience m'a encouragé à poursuivre l'ouverture à d'autres domaines, comme, par exemple, le projet CAPES-Cofecub MESO mené en partenariat avec des chercheurs en physique de l'atmosphère.

Parmi les perspectives, on peut citer la poursuite de la collaboration autour de AMIDE qui prend la forme d'un proposition de projet ANR en 2017. Ce projet porte sur la possibilité d'intégrer les N-glycosylation aux simulations effectuées sur les protéines de la matrice extra cellulaire. Les N-glycosylation sont des importants marqueurs dans les études sur le diabète et les récepteurs de l'insuline et, pour certaines protéines, les sites sont nombreux et les données expérimentales ne permettent pas toujours de connaître le nombre et les types de N-glycosylation portés par la molécule. Ma participation à ce projet relève de l'adaptation et de l'évolution de la plateforme AMIDE afin de déployer ces simulations à plus grande échelle.

Chapitre 3

L'Adaptation à la Dynamicité des Ressources

Résumé

La dynamique des systèmes distribués est un facteur d'hétérogénéité toujours présent mais souvent ignoré par les applications et les intergiciels, pour lesquels la gestion de la dynamique se limitait pendant longtemps à gérer les cas de déconnexion des nœuds. Même aujourd'hui, des plates-formes conçues pour un déploiement à grande échelle peinent à introduire une gestion plus fine de cette dynamique, même en sachant que la prise en compte du contexte est devenue une pièce clé pour d'autres domaines tels que l'informatique mobile, les systèmes d'information pervasifs ou bien les systèmes autonomes.

L'adaptation aux changements de contexte des ressources est un besoin qui se justifie non seulement par le besoin de garantir la continuité des opérations (surveillance des ressources et la détection des pannes/états invalides, le suivi et la récupération des tâches attribuées à des éléments disparus, etc.) mais aussi pour une question de performance et d'efficacité car une configuration inadaptée peut mener à la surcharge et à une mauvaise utilisation des ressources.

Cette troisième partie du mémoire illustre une partie de mes travaux visant la gestion de la dynamique des ressources et les différentes stratégies pour s'adapter à ces changements. J'ai choisi comme exemple le cas de l'adaptation de la plateforme Apache Hadoop aux environnements pervasifs, l'une des contributions du projet STIC-AmSud PER-MARE dont j'ai été le coordinateur international. Nous avons développé un mécanisme de collecte d'informations sur le contexte des ressources de calcul qui a été intégré à l'ordonnanceur de Hadoop. Grâce à ces informations, Hadoop est devenu capable d'adapter le lancement de tâches selon les ressources disponibles à chaque instant. On démontre l'efficacité de cette solution par le biais de comparaisons entre la configuration standard de Hadoop et notre contribution.

La majorité des travaux présents dans cette partie ont été réalisés en collaboration avec Guilherme Cassales et Andrea Charão de l'Universidade Federal de Santa Maria (Brésil), ainsi que Manuele Kirsch Pinheiro à l'Université Paris 1 Panthéon-Sorbonne, toujours dans le cadre du projet STIC-AmSud PER-MARE. Ces travaux ont conduit à la publication d'un article dans un journal international ([29]) et deux articles dans des conférences internationales ([28, 27]).

3.1 Hétérogénéité et Dynamicité des Ressources

La suite logicielle Apache Hadoop [1] est très populaire dans le domaine du *big data* et du calcul distribué. En effet, c'est l'un des outils pionniers dans le traitement de grandes masses de données grâce au support du paradigme de programmation *MapReduce* [45]. Bien que Apache Hadoop puisse être déployé sur des *clusters* composés de milliers de machines, ces ressources sont supposées être homogènes, sauf dans le cas d'une configuration spécifique de la part de l'administrateur. En effet, l'exécution des application *MapReduce* dépend d'une bonne corrélation entre l'ordonnancement des tâches et les ressources allouées, or la présence de ressources hétérogènes ou dynamiques n'est pas suffisamment prise en charge par Hadoop.

C'est pour cette raison que nous avons lancé le projet STIC-AmSud PER-MARE (Adaptive Deployment of MapReduce-based Applications over Pervasive and Desktop Grid Infrastructures [143]) dont j'ai été l'idéalisateur et le coordinateur international. Le but de ce projet de collaboration international entre la France, le Brésil et l'Uruguay était de permettre le support aux applications *big data* de type *MapReduce* dans des environnements de type *grid* pervasif, c'est-à-dire, des environnements de calcul faiblement connexes marqués par l'hétérogénéité et par la volatilité des ressources [138]. Le projet PER-MARE était organisé autour de deux volets : d'un côté l'adaptation de la plateforme Hadoop aux environnements pervasifs et, de l'autre, le développement d'une solution de calcul distribué totalement répartie capable d'exécuter des applications de type *MapReduce*.

Dans la suite de cette section on présentera donc l'architecture du framework Apache Hadoop et ses limitations concernant l'hétérogénéité des ressources. Par la suite on verra les efforts effectués afin d'introduire des éléments liés au contexte dans l'ordonnancement des tâches, améliorant ainsi la performance et l'adaptabilité de la plateforme.

3.2 Le Paradigme MapReduce et le *Framework* Hadoop

MapReduce [45] est un paradigme de programmation parallèle très utilisé pour le traitement massif de données. La force de ce paradigme est son modèle de calcul très simple et facile à déployer à grande échelle. En effet, le calcul (ou traitement des données) se fait en deux étapes, la phase *map* et la phase *reduce*, tous les deux utilisant des tuples *clé-valeur* (k, V) en tant qu'entrée ou sortie [156]. Plus exactement, un algorithme *MapReduce* peut être décrit selon la procédure suivante :

1. **map** : à partir d'une ou plusieurs tuples *clé-valeur* en entrée, la fonction *map* génère un ensemble de tuples intermédiaires $(k_1; V_1) \rightarrow \{(k_2; V_2)\}$;
2. **reduce** : à partir de l'ensemble de tuples intermédiaires, la fonction *reduce* associe les valeurs de toutes les tuples avec la même clé, produisant une nouvelle tuple en sortie $(k_2; \{V_2\}) \rightarrow \{(k_3; V_3)\}$.

L'étape *map* est donc facilement parallélisable car la génération des clés intermédiaires ne dépend que des tuples en entrée. Lors de son implémentation dans un système distribué, les tuples intermédiaires pour une clé k_2 peuvent être éparpillées sur plusieurs noeuds. L'implémentation doit ainsi regrouper toutes les tuples avec la clé k_2 de manière à ce que la fonction *reduce* puisse les associer et produire un résultat final.

1. `http://hadoop.apache.org/`

L'exemple de la Figure 3.1 illustre une implémentation *MapReduce* destinée compter le nombre de clients de chaque rue, à partir des entrées dans un annuaire téléphonique. Les données à l'entrée ont la forme $(ligne, adresse)$ et correspondent aux multiples adresses figurant dans l'annuaire. Ces données seront réparties entre les noeuds qui, pendant la phase *map*, associeront une valeur *1* à chaque rue, créant ainsi des tuples intermédiaires $(rue, 1)$. Les multiples tuples intermédiaires seront triées par leurs clés, de manière à ce que les nœuds exécutant la phase *reduce* puissent faire la somme des valeurs et sortir le nombre de clients par rue.

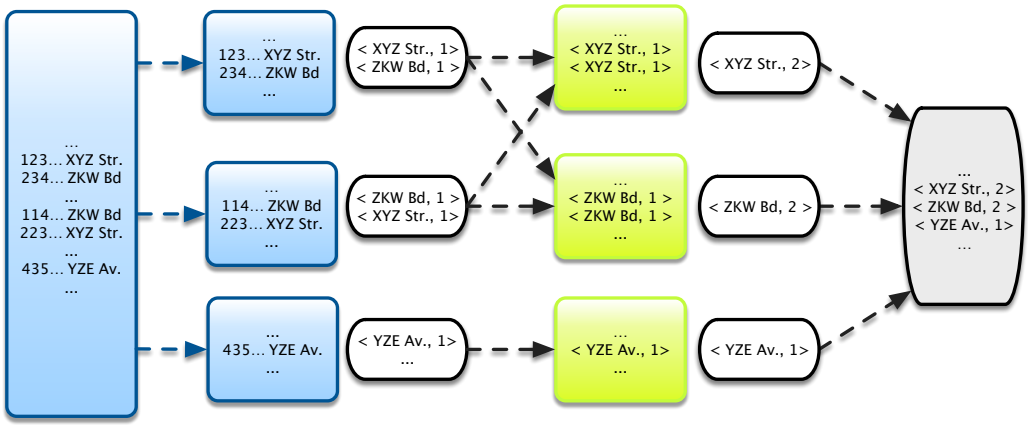

FIGURE 3.1 : Schéma *MapReduce* pour de compter le nombre de clients dans chaque rue d'une ville

Parmi les différentes implémentations de *MapReduce*, la plus populaire est celle du projet Apache Hadoop [4]. Pour la soumission d'un programme *MapReduce*, le programmeur ne doit fournir que les instructions pour l'exécution des tâches *map* et *reduce* [156]. Le framework Hadoop se charge de tout le reste, autant en ce que concerne la répartition des tâches entre les machines que la distribution des données nécessaires à chaque tâche et à chaque phase.

3.2.1 Architecture et Ordonnancement dans Hadoop

Le framework Apache Hadoop est en réalité un écosystème assez important composé de presque une dizaine d'outils et services, allant de la gestion "bas niveau" des données et tâches de calcul à l'intégration avec des sources extérieures et le requêtage haut-niveau (parfois en imitant le langage SQL). Certains de ces outils ont été rajoutés au fil du temps grâce à des efforts de différents contributeurs (par exemple, le système de base de données HBASE [2] développé initialement par Facebook). D'autres outils faisaient partie du projet dès son départ mais ont gagné un statut de "projet" propre, comme par exemple le service ZooKeeper [3], responsable de la coordination distribuée fiable entre les nœuds et souvent au cœur des efforts de tolérance aux fautes de Hadoop.

La plateforme Hadoop elle aussi a subi des modifications au fil du temps. La version initiale (version 1.x) était extrêmement ancrée sur le paradigme *MapReduce*, qui était le seul moyen d'utiliser la plateforme. À partir de 2012 la version 2.x Hadoop devient une plateforme plus générique, où l'on peut toujours exécuter des applications *MapReduce* à côté d'autres applications. En effet, Hadoop devient surtout un gestionnaire de ressources qui aide à déployer et exécuter les tâches qui lui sont assignées.

2. `https://hbase.apache.org/`
3. `https://zookeeper.apache.org/`

Au cœur de la version 2.0 de Hadoop nous trouvons deux services principaux organisés chacun selon une architecture maître-esclave : le système de stockage distribué nommé HDFS (*Hadoop Distributed File System*) et le système de gestion des ressources nommé YARN (*Yet Another Resource Negotiator*). Les deux services présentent des composants jouant les rôles de maître ou esclave comme présenté en Figure 3.2 : les processus `Name Node` et *Resource Manager* correspondent aux rôles de maître dans HDFS et YARN, respectivement, et les processus `Data Node` et `Node Manager` correspondent aux parties esclaves.

Nous pouvons observer aussi dans la Figure 3.2 la présence de deux autres composants appelés `Application Master` et `Containers` (conteneurs). L'`Application Master` est un processus désigné pour effectuer l'ordonnancement des tâches de calcul d'une seule application, qui seront exécutées par des éléments `Container` associés.

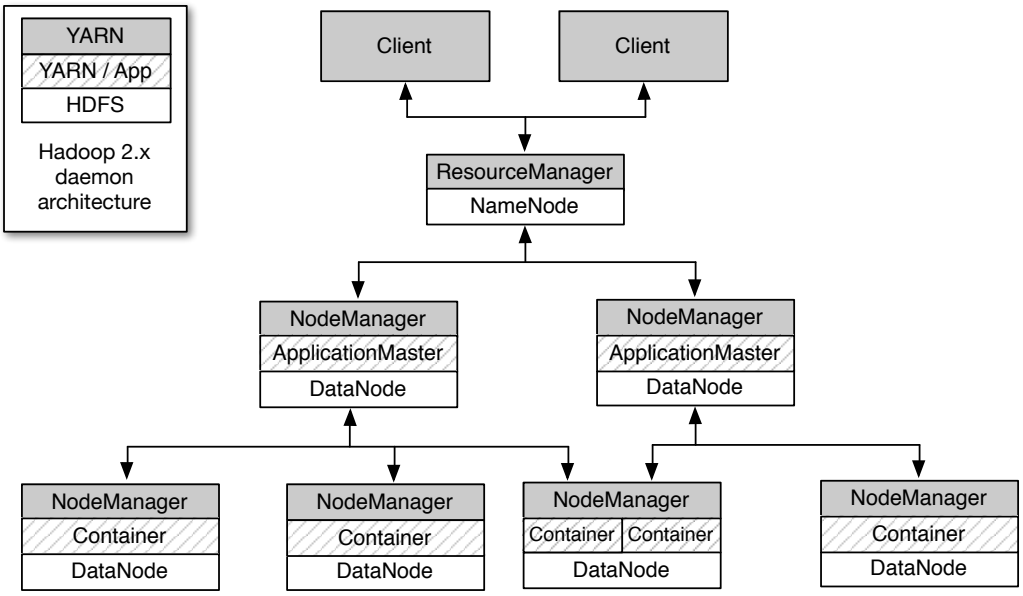

FIGURE 3.2 : Architecture de base de Apache Hadoop 2.x

On observe donc que le framework Hadoop utilise deux niveaux d'ordonnancement. Les "*jobs*" représentent des instances avec une granularité plus grande, alors que les tâches représentent des instances de plus fin grain présentes au sein d'un job.

L'ordonnancement au niveau des *jobs* est effectué par le `Resource Manager`, la seule entité qui a une vue globale des ressources du système grâce aux informations envoyées par les `Node Manager`. Grâce à ces informations, le `Resource Manager` peut arbitrer la répartition des ressources entre les applications, en se basant sur différentes métriques telles que l'utilisation des ressources, l'équité, les contrats SLA, etc. Un `Application Master` est ainsi détaché pour chaque application et devient responsable par l'ordonnancement et l'exécution des tâches de cette application par le biais des *conteneurs*, des unités de calcul isolées et disposant d'un accès limité aux ressources (mémoire, CPU).

Vu cette complexité, le `Resource Manager` a été projeté de manière à pouvoir être optimisé selon des contraintes et des paramètres propres aux utilisateurs, grâce à un mécanisme d'extensions. Toutefois, la plupart des utilisations répertoriées dans la littérature n'utilisent que les ordonnanceurs livrés avec Hadoop. Le plus simple de ces algorithmes d'ordonnancement est le `Internal Scheduler`, une simple liste d'exécution où les jobs sont servis selon leur ordre d'arrivée (FIFO). Évidemment, cet algorithme n'est indiqué que pour les *clusters* où la compétition pour des ressources n'est pas un problème.

Deux autres algorithmes sont souvent cités : l'algorithme `Fair Scheduler` et l'algorithme `Capacity Scheduler`. `Fair Scheduler` utilise un mécanisme d'ordonnancement à deux niveaux pour effectuer un partage équitable entre des jobs de petite taille [4]. `Capacity Scheduler`, de son côté, a été créé pour l'utilisation de Hadoop dans un environnement où plusieurs partenaires contribuent afin de composer un grand *cluster*. En effet, le `Capacity Scheduler` offre des garanties minimales d'accès aux ressources pour chaque partenaire, tout en permettant l'utilisation élargie du *cluster* lorsque des ressources se trouvent libres [4].

Les trois algorithmes cités ci-dessous illustrent des approches différentes pour la gestion des jobs, mais cela se fait uniquement par rapport à des facteurs tels que la disponibilité de ressources ou les politiques d'équité, sans jamais prendre en compte la dynamicité et l'hétérogénéité de l'environnement d'exécution. En effet, Hadoop considère que la gestion à grain fin de l'exécution incombe au `Application Master`, qui a une vue plus proche de l'application mais qui est aussi limité aux ressources que lui sont attribués au départ.

Malheureusement, le fonctionnement de l'`Application Master` est peu documenté. Afin de combler ce manque d'information, nous avons analysé son code source et conduit des expériences pour comprendre ses politiques d'allocation des tâches (résultats publiés dans [30]). Ce que ressort est un simple mécanisme de remplissage des nœuds visant la proximité des tâches : on remplit un nœud avec autant de conteneurs qu'il peut supporter, pour ensuite commencer le remplissage du prochain nœud.

Ceci nous a permis aussi d'observer que l'`Application Master` se limite à répartir les conteneurs sans une véritable adéquation au contexte d'exécution. Toute connaissance sur la capacité des nœuds provient du `Resource Manager`, la seule entité qui est alimentée avec ces informations. Ainsi, la modification des algorithmes d'ordonnancement dans le but d'inclure des informations de contexte doit se faire en étroite relation avec le `Resource Manager`.

3.2.2 Dynamicité des Ressources dans Hadoop

La littérature propose différentes approches pour rendre Hadoop plus compatible avec les environnements hétérogènes. Des travaux comme [93], [146] ou [119] assument que les applications *MapReduce* sont exécutées régulièrement dans un environnement de "production", et que chacune des applications a des besoins spécifiques en CPU, mémoire, réseaux ou en stockage. Cette hypothèse considère donc la possibilité d'optimiser l'exécution des applications en faisant la correspondance entre les besoins et les caractéristiques des ressources. De même, [80] propose un algorithme d'ordonnancement où une fonction de coût basée sur un graphe "capacité-demande" permet l'ordonnancement des jobs.

Les travaux cités ci-dessus considèrent des ressources hétérogènes mais statiques et, une fois lancés, ces jobs ne sont plus "suivis" car l'environnement est supposé immuable. Une manière de rendre cet ordonnancement plus dynamique est d'incorporer des informations sur le déroulement des tâches. Par exemple, [164] et [35] essayent d'améliorer la distribution des tâches afin de réduire le temps de réponse dans des *clusters* de grande taille. Pour cela, [164] utilise des heuristiques pour estimer la progression des tâches et ainsi décider s'il faut lancer des tâches spéculatives. Les tâches spéculatives sont des doublons (ré-soumissions) qui sont lancées lorsqu'il y a le soupçon qu'une tâche originale est retardée à cause d'un nœud défaillant ou trop lent. Dans une ligne similaire, [35] propose l'utilisation des traces historiques d'exécution afin d'aider cette décision.

Une autre manière d'augmenter la performance passe par un meilleur placement des données et par l'utilisation de cette information pour le déploiement des jobs [161]. En faisant un placement optimisé des données, on réduit les transferts de données occasionnés par le lancement de tâches spéculatives sur d'autres nœuds. Une approche similaire est présentée par [31], qui étudie les problèmes d'ordonnancement et répartition des données dans les *clusters* géographiquement distribués. Ainsi, ces auteurs présentent un mécanisme d'ordonnancement basé sur les ressources de calcul mais aussi sur le débit du réseau.

Sans aucun paramètre supplémentaire, les mécanismes cités jusqu'à présent ont comme résultat un équilibrage de charge, obligeant les nœuds les plus rapides à travailler plus et les moins performants à exécuter moins de tâches. Une manière de rompre cette logique est utilisée par [126], qui permet d'influencer l'ordonnancement grâce à des profils d'exécution suggérés par l'utilisateur (par exemple, privilégier les nœuds lents si le job n'est pas prioritaire).

Il faut observer cependant que la difficulté à adapter l'exécution de *MapReduce* sur des environnements hétérogènes (et dynamiques) est en grand partie due à la conception même de la plateforme Apache Hadoop, qui est très hiérarchique (voir Figure 3.2). Certains travaux essayent de s'affranchir de ces barrières en développant d'autres plates-formes compatibles avec *MapReduce* mais plus adaptées à l'a dynamicité des ressources. L'utilisation de overlays P2P est ainsi un choix naturel, comme le montrent [103] et [140]. Dans le système proposé par [103], les nœuds incarnent les différentes fonctions de l'architecture Apache Hadoop (NameNode, etc.) selon les besoins de l'application. Cependant, ce travail vise la tolérance aux fautes et n'explore pas les possibilités d'optimisation de l'ordonnancement des jobs et des tâches.

Il faut aussi noter que les travaux cités précédemment ne tiennent pas en compte l'évolution des ressources au fil de l'exécution : les ressources sont décrites mais pas observées. Malgré la diversité de travaux sur l'importance de la prise en compte du contexte d'exécution [5, 100, 118, 108], Hadoop reste essentiellement une plateforme statique, particulièrement adaptée aux clusters homogènes. Pour toutes ces raisons, une partie de notre travail au sein du projet STIC-AmSud PER-MARE a été d'intégrer les informations de contexte à l'exécution de Hadoop.

3.3 Ordonnancement Orienté par le Contexte

Comme indiqué dans la section 3.2.1, l'élément central de l'ordonnancement est le `Resource Manager`. En effet, c'est grâce aux informations fournies par cet élément que les ordonnanceurs de Hadoop tels que le `Capacity Scheduler` décident du démarrage et du placement des tâches.

L'implémentation par défaut de Hadoop considère qu'un `Node Manager` déclare ses ressources au `Resource Manager` lors de sa connexion au réseau Hadoop, or la description de ces ressources est usuellement obtenue à partir de fichiers de configuration statiques. Afin de rendre cette information de contexte dynamique, nous devons mettre en place un mécanisme de capture de contexte et aussi permettre au `Node Manager` de communiquer périodiquement ses ressources au `Resource Manager`.

Afin de modifier le moins possible le code de Hadoop, nous avons développé un module de capture de contexte qui peut être greffé à Hadoop et ainsi mettre à jour les informations sur les ressources disponibles. Les sous-sections suivantes détaillent le fonctionnement de ce module et aussi le mécanisme retenu pour son intégration à Hadoop.

3.3.1 Le Collecteur de Contexte

Par défaut, Hadoop obtient des informations sur les ressources des nœuds à partir de fichiers de configuration au format XML. Ces fichiers contiennent plusieurs paramètres, dont le nombre d'unités d'exécution (cœurs de calcul) et la capacité de la mémoire des nœuds. Une fois lues, ces informations ne sont pas mises à jour, sauf en cas de redémarrage du nœud. Afin de rendre possible l'exécution de Hadoop dans un environnement pervasif, nous avons mis en place un mécanisme de collecte d'informations de contexte qui peut être utilisé pour ajourner la base de connaissances du `Resource Manager`.

Ce collecteur de contexte a été développé dans le cadre du projet PER-MARE[143] et est structuré selon le diagramme de classes présenté en Figure 3.3 [30]. La capture des différents éléments de contexte se font grâce à l'API standard Java Monitoring API [110], qui permet l'accès aux caractéristiques de la machine virtuelle Java et de la machine hôte. En effet, cela nous permet d'obtenir des informations de contexte telles que le nombre de processus (cœurs de calcul), la mémoire du système, ou la charge de la machine. Le collecteur de contexte a été structuré avec un ensemble d'interfaces et de classes abstraites, ce qui permet de généraliser le processus de collecte des données. De plus, en raison de sa conception, il est simple d'intégrer des nouveaux collecteurs et ainsi de diversifier les informations de contexte observées.

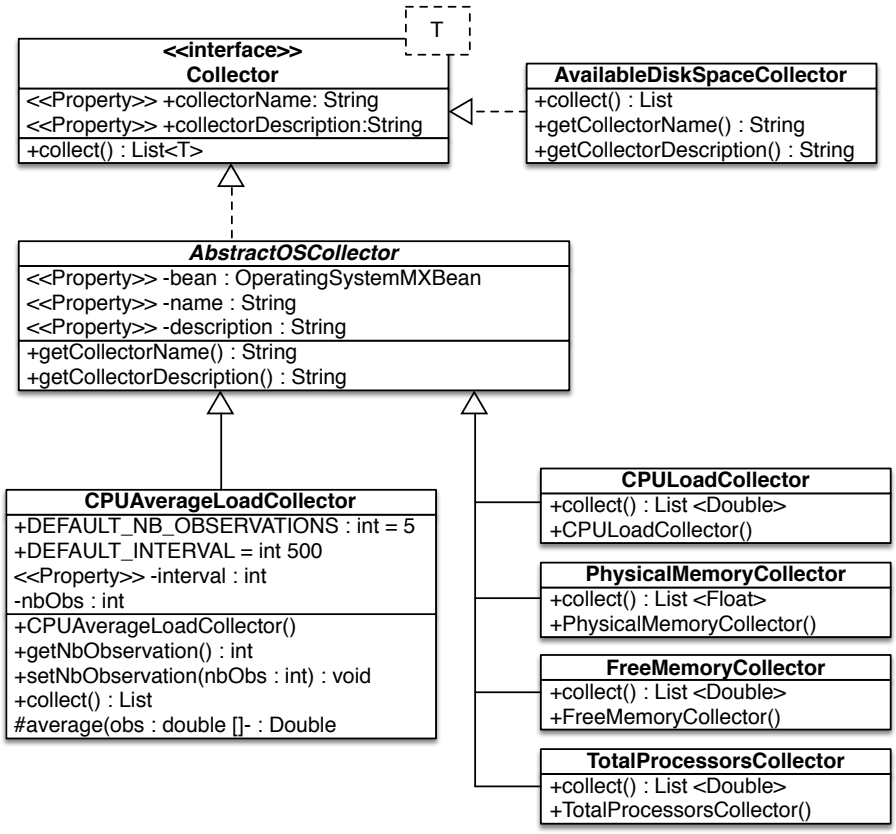

FIGURE 3.3 : Structure du collecteur de contexte

Cependant, il ne suffit pas de remplacer les fichiers de configuration XML par les informations du collecteur car ces informations resteraient statiques. Afin d'ajourner le `Resource Manager`, il faut que le collecteur de contexte de chaque nœud puisse communiquer son état au `Resource Manager`, et cela à n'importe quel moment de l'exécution. Afin de rendre ceci

possible, nous avons étendu les possibilités de communication entre le `Resource Manager` et les `Node Manager`, comme expliqué dans la section suivante.

3.3.2 Communication

Dans l'architecture Hadoop, les informations de contexte collectées par les nœuds esclaves (`Node Manager`) doivent être transmises au nœud maître (`Resource Manager`), qui sera en charge de l'ordonnancement. Au lieu de créer un mécanisme séparé, nous avons choisi d'intégrer cette communication au sein de l'API ZooKeeper [77], qui fait partie de l'écosystème Hadoop. Dans notre cas, les services de ZooKeeper seront utilisés pour récupérer les informations de contexte et les rendre disponible auprès le `Resource Manager`.

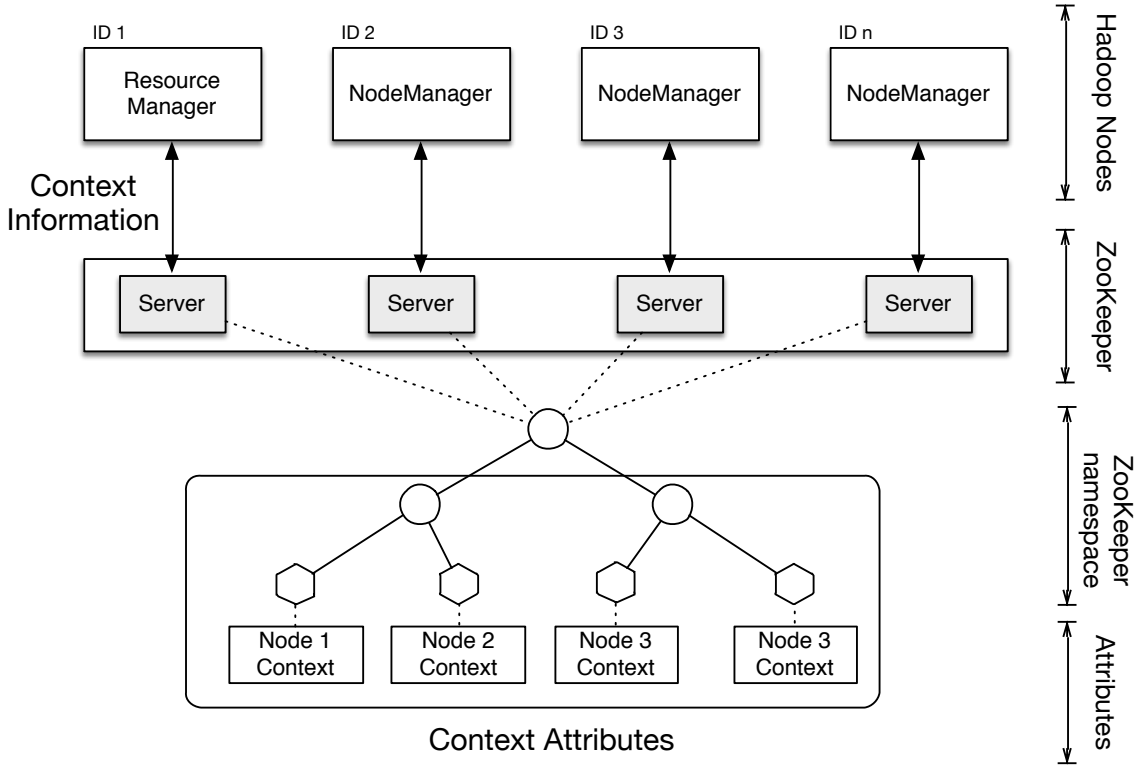

FIGURE 3.4 : Utilisation de ZooKeeper pour distribuer l'information de contexte

Comme illustré en Figure 3.4, tous les esclaves (`Node Manager`) exécutent une instance du service *NodeStatusUpdater*, lequel collecte régulièrement les données sur la disponibilité des ressources (par exemple, à chaque 30 secondes). Si les ressources varient plus qu'un certain seuil, le tableau dans ZooKeeper sera mis à jour. Ce seuil est nécessaire car le système d'exploitation peut subir des légères variations des ressources (par exemple, la quantité de mémoire disponible), alors que ces variations n'ont pas un impact sur la capacité d'un nœud. Ce mécanisme contribue aussi à réduire la quantité d'informations échangées et évite trop d'événements qui pourraient impacter la performance de l'algorithme d'ordonnancement.

De manière similaire, le maître (*Resource Manager*) crée aussi un service pour surveiller les informations sur de ZooKeeper. Lorsque ZooKeeper détecte une modification des données, le maître sera notifié et pourra mettre à jour les informations utilisées par l'ordonnanceur. Les

modifications apportées au code source du *Resource Manager* et des *Node Manager* est assez limitée, permettant son application sur différentes versions de Hadoop.

Dans la section suivante nous allons montrer les résultats de quelques expériences faites pour valider ce mécanisme.

3.4 Évaluation Pratique

Afin d'évaluer l'impact de la prise en compte du contexte dans le cadre de l'ordonnancement des tâches, nous avons conduit une série d'expériences sur un petit *cluster* dédié. Cet environnement nous permet de contrôler les ressources disponibles et aussi ceux "observés" par le framework Hadoop, de manière à pouvoir mesurer l'impact d'une mauvaise détection et les avantages de l'adaptation au contexte. Dans les tests effectués, nous avons observé le comportement de Hadoop selon deux métriques, la ressource "mémoire disponible" et le nombre de cœurs de calcul (*v-cores*). Ces paramètres sont toujours renseignés au *Resource Manager* et font partie des principaux attributs utilisés par l'algorithme Capacity Scheduler. En effet, la mémoire totale disponible et le nombre de cœurs permettent la définition du nombre de tâches simultanées (conteneurs) qui peuvent être exécutées par un nœud. Une mauvaise information peut donc créer une surcharge de la machine, affectant la performance.

Pour la définition des scénarios d'exécution, nous avons travaillé avec l'hypothèse que la performance est dégradée si la mémoire disponible annoncée au gestionnaire de ressources est supérieure à celle réellement disponible. La situation contraire (plus de mémoire disponible que celle annoncée) n'impacte pas l'exécution d'une tâche. Ainsi, nous avons défini 4 situations d'exécution :

Scénario A : dans ce scénario "de contrôle" la mémoire disponible annoncée au gestionnaire de ressources correspond à la mémoire disponible. De même, le nombre de cœurs de calcul renseigné correspond au nombre de cœurs disponibles. Les ressources ne varient pas pendant l'exécution, ce qui peut être considéré comme le "*best case*".

Scénario B : dans ce cas, la mémoire disponible et le nombre de cœurs sont inférieurs à ceux annoncés. Cependant, elle ne sera pas mise à jour au niveau du gestionnaire de ressources, reproduisant ainsi le comportement par défaut de Hadoop. Comme l'ordonnanceur ne s'adapte pas, ceci peut être considéré comme un scénario "*worst case*".

Scénario C : dans ce troisième cas, le collecteur de contexte est actif dès le départ et renseigne les ressources effectivement disponibles à chaque 30 secondes. Ainsi, quand l'application est lancée, l'ordonnanceur est au courant du contexte d'exécution et peut lancer les tâches conformément à ces ressources, sans surcharger les machines.

Scénario D : finalement, ce scénario représente une extension du Scénario C dans lequel l'exécution de l'application *MapReduce* démarre avant la mise à jour du collecteur de contexte. De cette manière l'ordonnanceur est initialisé avec des informations incorrectes et doit s'adapter pendant l'exécution. Cette adaptation n'est pas immédiate car elle ne concerne que l'ordonnancement des tâches en attente, pas celle des tâches déjà en exécution.

3.4.1 Benchmarks et Environnement de Tests

Deux types différents d'application ont été utilisés comme benchmarks afin de vérifier l'impact de l'adaptation au contexte. Même si les applications *big data* sont fortement dépendantes

de l'accès mémoire, d'autres facteurs comme l'utilisation de la CPU ou les opérations d'entrée/sortie (I/O) sont aussi importantes. Pour cela, les deux applications choisies ont des profils différents par rapport à leurs besoins en mémoire, CPU et I/O [98], comme indiqué ci-dessous :

— TeraSort : L'application TeraSort [71] est une application destinée à effectuer le tri d'un grand ensemble de données. C'est un benchmark très populaire car les algorithmes de tri stressent la mémoire et la CPU au même temps qu'ils sollicitent l'I/O à cause des masses des données à trier ;

— TestDFSIO : Le benchmark TestDFSIO a été conçu spécifiquement pour étudier l'interaction de Hadoop avec HDFS, permettant la découverte de goulots d'étranglement au niveau du réseau d'interconnexion, du système d'exploitation et de la configuration Hadoop. Dans cette application, la mémoire et la CPU sont moins sollicitées.

Les deux benchmarks font partie de la plateforme de tests HiBench [76]. Le tri TeraSort a été exécuté sur un ensemble de données de 15 GB, alors que TestDFSIO a été exécuté avec 90 fichiers de 250 MB chacun. Les différents scénarios ont été exécutés sur la plateforme Grid'5000 [66]. Nous avons configuré un réseau dédié avec 5 machines (dont une "maître" et quatre "esclaves"), chacune avec la configuration suivante : 2 Intel Xeon CPU E5420 @ 2.50 GHz (8 cœurs par nœud) et 8 GB de mémoire RAM. Tous les nœuds exécutent Ubuntu-x64-12.04, avec JDK 1.7 et la distribution Apache Hadoop 2.5.1.

L'analyse des performances se fait grâce à l'étude des fichiers de log de chaque tâche (conteneur), qui contiennent des informations sur le nœud d'allocation, le moment de démarrage et le temps nécessaire pour l'exécution de chaque tâche. Nous avons choisi d'exécuter les tâches "maître" sur un nœud séparé afin de ne pas surcharger les nœuds esclaves avec des activités de gestion de Hadoop.

Finalement, afin d'émuler la réduction des ressources en mémoire et cœurs de calcul nécessaires aux scénarios B, C et D, nous avons choisi de réduire le nombre effectif de nœuds utilisés, une méthode drastique mais plus fiable que la limitation logicielle des ressources disponibles.

3.4.2 Résultats

Les exécutions des benchmarks dans les différents scénarios sont représentées par les diagrammes de Gantt des Figures 3.5 et 3.6, respectivement pour TeraSort et TestDFSIO. De même, les Tableaux 3.1 et 3.2 résument les données clés de ces expériences, avec le temps total d'exécution des tâches *map*, le temps moyen d'exécution, l'écart-type, le nombre de tâches *map* et aussi le nombre de tâches spéculatives démarrées.

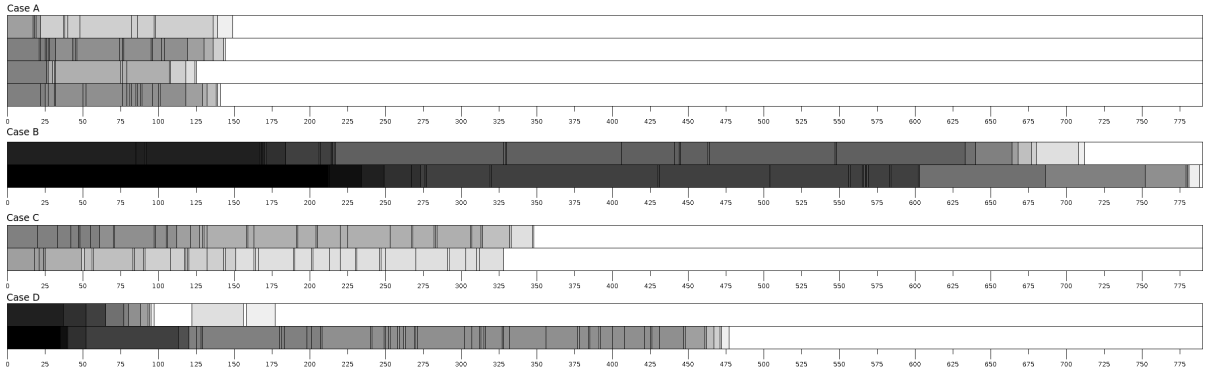

FIGURE 3.5 : Diagramme de Gantt pour l'exécution de TeraSort

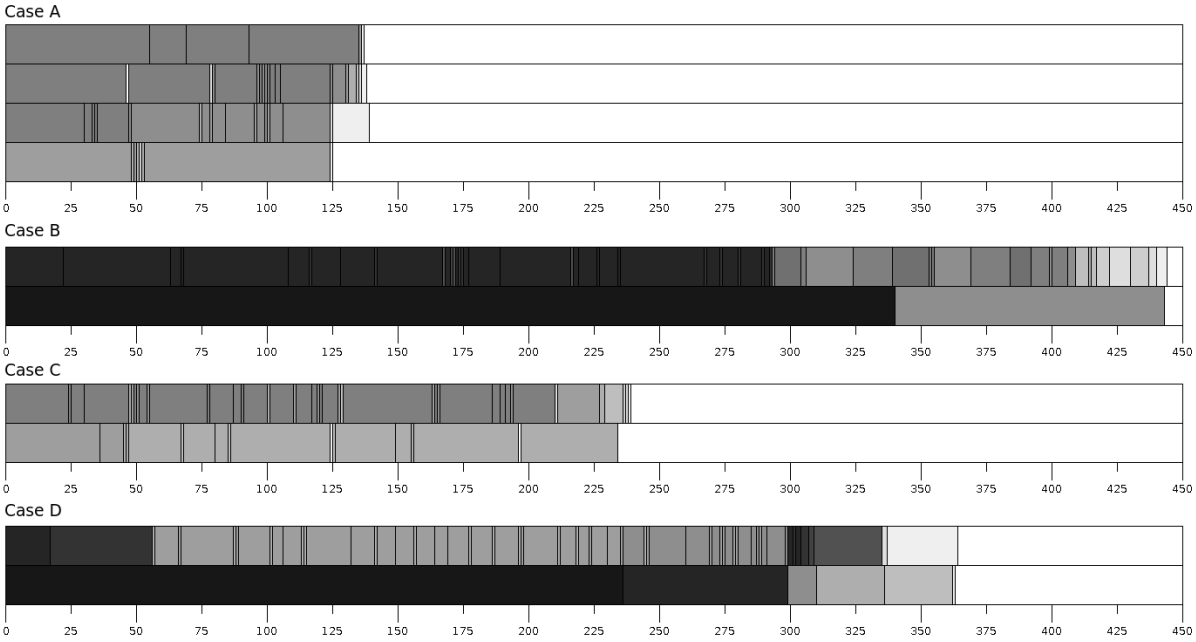

FIGURE 3.6 : Diagramme de Gantt pour l'exécution de TestDFSIO

TABLE 3.1 : Tableau récapitulatif de l'exécution de TeraSort

Scénario	**A**	**B**	**C**	**D**
Temps total *map* (*s*)	149	788	348	477
Temps moyen (*s*)	39.47	222.97	38.38	68.42
Écart-type	15.73	59.86	18.09	29.91
# tâches *map*	76	76	76	76
# tâches spéculatives	2	1	3	1

TABLE 3.2 : Tableau récapitulatif de l'exécution de TeraSort

Scénario	**A**	**B**	**C**	**D**
Temps total *map* (*s*)	139	444	239	364
Temps moyen (*s*)	38.95	85.01	32.20	81.62
Écart-type	17.20	69.08	8.30	73.60
# tâches *map*	90	90	90	90
# tâches spéculatives	0	9	0	1

Pour les diagrammes de Gantt, chaque scénario est composé de 2 ou 4 lignes correspondant au nombre de nœuds utilisés. Comme indiqué précédemment, les scénarios B, C et D n'utilisent que la moitié des nœuds du scénario A afin de simuler la réduction des ressources. L'échelle de gris présent dans chaque ligne indique la surcharge des nœuds : plus sombre est le créneau, plus de conteneurs s'exécutent simultanément. De cette manière, un créneau "blanc" n'a aucun conteneur en exécution, alors qu'un créneau "noir" en contient 16 conteneurs (le double de la capacité d'un nœud). Les séparations des créneaux indiquent soit le démarrage d'une tâche, soit une fin d'exécution, mais ne permettent pas de suivre le temps d'exécution d'une tâche précise.

L'analyse des tableaux permet d'identifier certaines tendances. En effet, toutes les exécutions présentent un motif similaire quand on observe le temps total d'exécutionă: le scénario A

est toujours le plus rapide, suivi de des cas C et D puis finalement B. Nous observons aussi que les scénarios A et C ont les plus petits temps moyens et les plus petites variations de performance, indépendamment de l'application. Ceci s'explique par le fait que dans ces deux scénarios les nœuds ne sont jamais surchargés car l'ordonnanceur a des informations précises au moment du démarrage de l'application. Ceci s'observe aussi par la tonalité des créneaux, indiquant un nombre moins important de tâches en simultané. Le temps total d'exécution du scénario C est aussi deux fois plus important que celui du scénario A, une conséquence attendue à cause de la réduction des ressources.

L'analyse du nombre de tâches spéculatives apporte aussi quelques renseignements. Dans le cas de TeraSort, tous les scénarios se comportent de manière similaire. Par contre, dans le cas de TestDFSIO le déploiement de tâches spéculatives ne se fait que lorsque le système est surchargé (notamment dans le scénario B). La raison pour cette différence vient des facteurs qui sont liés au lancement de tâches spéculativesă: une tâche spéculative n'est lancée qu'après le lancement de toute autre tâche "originale", et déclenchée seulement lorsque ces tâches sont en exécution depuis un certain temps (au moins une minute) et n'ont pas progressé autant que la moyenne des autres tâches du job. Dans le cas de TeraSort, les tâches dépendent autant de la mémoire que de la CPU et de l'I/O, et le recouvrement de ces besoins compense d'une certaine manière le manque d'une ressource. TestDFSIO, à l'opposé, s'appuie sur des ressources plus spécifiques et est donc plus enclin à la surcharge des nœuds. Et même dans les scénarios surchargés, l'utilisation d'un mécanisme de détection du contexte sur le scénario D permet à l'ordonnanceur de lisser la charge lors de la mise à jour des informations sur les ressources.

Il faut aussi pointer un détail concernant les diagrammes de Gantt. Dans tous les benchmarks il y a un nœud qui semble moins chargé que les autres. Ceci n'est pas la faute à une mauvaise répartition de la charge mais plutôt à la présence de tâches *reduce*, qui ne sont pas affichées dans les diagrammes. En effet, Hadoop permet le démarrage de tâches *reduce* aussitôt un certain nombre de tâches *map* a été complété, ce qui est le cas pour ces applications.

3.5 Bilan et Perspectives

Dans ce chapitre j'ai essayé de mettre en évidence l'importance de la prise en compte des variations dynamiques des ressources. L'adaptation au contexte est une discipline essentielle à d'autres domaines tels que les applications mobiles et les systèmes d'information pervasifs, mais cela n'est pas encore ancré comme une pratique courante dans le calcul haute performance, ni même dans les outils *big data* les plus populaires.

Le résultat de ces expériences démontre que l'utilisation d'un mécanisme de collecte et de mise à jour des informations de contexte permet l'adaptation de l'ordonnanceur Hadoop aux aléas d'une plateforme d'exécution dynamique. La solution que nous avons proposé dans ce travail permet non seulement un gain de performance dans les scénarios avec risque de surcharge mais aussi impose très peu de modifications au niveau du code source de Hadoop, rendant la solution suffisamment générique et intégrable aux différentes versions de ce *framework*.

Il faut aussi citer un deuxième travail effectué en parallèle à celui présenté ici. Aussi dans le cadre du projet PER-MARE, ce travail s'est porté sur un autre élément critique de la version 1.x de Hadoop, le *JobTracker*. Contrairement au *RessourceManager* de Hadoop 2.x, la version 1 de Hadoop a un élément centralisé qui n'est pas tolérant aux pannes (c'est un *single point of failure*). Si le processus exécutant le *JobTracker* disparaît, aucune nouvelle soumission ne sera distribuée aux nœuds esclave. Le travail que j'ai conduit avec en collaboration avec l'équipe

de l'Universidad de la República (Uruguay) consistait à mettre en place une sauvegarde des paramètres du *JobTracker* sur ZooKeeper, et de faire surveiller son état par une machine "*backup*" choisie en raison de sa stabilité. Nous avons pu développer et tester cette fonctionnalité et publier ses résultats [122, 121]. Malheureusement, l'arrivée de la version 2.x de Hadoop a introduit des modifications qui ont rendu ce travail obsolète.

Parmi les perspectives de recherche visant la prise en charge du contexte des ressources je peux citer les suivantes :

Ordonnancement sensible au contexte pour le *fog computing*

Cette activité de recherche entre dans le cadre du développement de la plateforme Cloud-FIT, qui sera présentée en Chapitre 5. Dans ce cas précis, nous utilisons les informations sur les capacités des ressources pour orienter l'ordonnancement des tâches. De surcroît, ces informations peuvent être utilisées pour l'établissement d'une organisation multi-échelle du réseau, ce qui permet une meilleure gestion des ressources dans des environnements hétérogènes tels que le *fog computing*. On étudie également les possibilités pour la gestion du déploiement et la migration de micro-services, dans le cadre d'une collaboration avec un étudiant de doctorat à l'Université Paris 1.

Expérimentation avec des capteurs IoT pour la *smart agriculture* et les sciences de l'atmosphère

Je développe depuis un certain temps des activités expérimentales autour des réseaux de capteurs. Ces activités incluent par exemple le développement de solutions basées sur des micro-contrôleurs Arduino pour la surveillance des ressources hydriques des cultures, le suivi de paramètres tels que la température, de l'ensoleillement, etc. Cette recherche vise la *smart-agriculture*, l'un des axes prioritaires de recherche de l'Université de Reims.

Au delà de la *smart agriculture*, je peux également citer l'utilisation de capteurs pour les sciences de l'atmosphère, dans le cadre du projet CAPES-Cofecub MESO. Nous étudions les possibilités d'utilisation et de déploiement de capteurs UV de la gamme ML8511 ou similaires pour l'établissement d'un réseau de surveillance "*crowdsourcing*" des variations de la couche d'Ozone, en complément des mesures effectuées par des instruments spécifiques et bien plus chers (spectre-photomètres Dobson et Brewer). Après calibration, nous espérons que ces capteurs puissent être disséminées sur une large zone géographique et ainsi fournir des indicateurs plus détaillés que ceux obtenus actuellement avec les équipements existants ou les mesures satellite. L'autre avantage de ce type de déploiement par rapport aux données issues des instruments à bord des satellites est que nous pouvons effectuer plusieurs mesures par jour, permettant un meilleur suivi de la dynamique de l'atmosphère.

Gestion de la consommation énergétique

Un autre niche très prometteur pour l'utilisation de l'adaptation au contexte est celui de la lutte contre le gaspillage énergétique. Dans ce sens, je participe à deux efforts de recherche où la surveillance des ressources et l'analyse des profils de consommation peuvent conduire à une meilleure utilisation des ressources disponibles.

Le projet STIC-AmSud CC-SEM actuellement en cours (2017-2018) vise le développement d'une plateforme intégrée pour la surveillance et le contrôle de la consommation électrique dans les milieux urbains. Le projet a pour le moment développé des modules de surveillance de la

consommation basés sur le micro-contrôleur STPM01/10 et intégrés à des boîtiers Raspberry Pi, qui seront déployés prochainement sur un ensemble résidentiel à Buenos Aires. Grâce aux informations obtenues sur la consommation des ménages et à d'autres éléments de contexte (température ambiante et extérieur, humidité, etc.), il sera possible d'orienter les utilisateurs à mieux répartir leur consommation électrique ou même d'agir en contrôlant l'activation de certains dispositifs très gourmands (le chauffe-eau, par exemple), afin de répartir la charge et éviter des coupures de courant. Ce travail est fait en collaboration avec l'Université de Buenos Aires (Argentina) et l'Universidad de la Republica (Uruguay), et compte également avec le support de la compagnie d'énergie électrique de l'Uruguay.

D'autres domaines que les *smart cities* peuvent aussi bénéficier d'une meilleure prise en compte du contexte des ressources. Par exemple, les applications mobiles sont souvent gourmandes en énergie, occasionnant une surconsommation des batteries des *smartphones* souvent décriée par les utilisateurs. Afin de mieux gérer les services et les applications mobiles en vue de leur consommation électrique, il faut utiliser les informations de contexte dans le but d'intégrer des stratégies de placement et la migration de services entre les dispositifs mobiles et les infrastructures. La migration de composants et dapplications à des fins déconomie dénergie sont au coeur dune proposition de projet ANR autour de la *Green IT* qui sera soumise cette année, alors que la prise en compte du contexte de consommation énergique dune application pour la gestion de ressources fait partie de mes perspectives de collaboration avec lUniversité Paris 1, notamment à travers la thèse de David Beserra.

Chapitre 4

L'Hétérogénéité des Données

Résumé

Le big data a remis en évidence plusieurs défis pour les systèmes distribués, notamment les fameux "big V's" : volume, variété, vitesse. Alors que le volume de données est plus ou moins simple d'être géré et que la vitesse dépend à la fois des ressources et des algorithmes, la variété des données est devenue l'un des points clé dans l'intégration des systèmes d'information [1]*. S'attaquer à cette hétérogénéité des données devient de plus en plus un défi pour le développement d'applications.*

Sans avoir l'ambition de proposer des solutions nouvelles pour la gestion de la variété de données, ce chapitre présente nos efforts visant à spécifier une plateforme documentaire garantissant un accès transparent à différentes sources d'information, indépendamment de la nature des données (fichiers, flux, bases de données, etc.). Cette spécification repose sur un ensemble d'éléments dédiés à l'indexation, à la recherche et à l'accès aux données. Également, la spécification s'occupe des mécanismes d'interconnexion et de coordination entre les éléments de la plateforme. Grâce à une organisation hiérarchique, cette plateforme a l'avantage de pouvoir être déployée sur une grande variété d'infrastructures.

La majorité des travaux présentés dans cette partie résultent du travail de thèse de Thierno Ahmadou Diallo, effectuée sous la co-direction de Olivier Flauzac (Université de Reims Champagne Ardenne), de Samba Ndiaye (Université Cheikh Anta Diop, Sénégal) et moi-même. Ces travaux on fait l'objet de publication de deux journaux ([48, 61]) et quatre conférences ([47, 50, 49, 60]).

4.1 Introduction

La gestion de données à grande échelle est un problème récurrent autant dans les domaines scientifiques que dans le monde de l'entreprise. Malgré les constantes avancées en matière de capacité des mémoires et disques, l'utilisation d'un seul dispositif de stockage n'est plus une option vu que l'accès concurrent, la fiabilité, la consommation énergétique et le coût sont des obstacles au développement des systèmes. C'est pour cette raison que les chercheurs et

1. http://sloanreview.mit.edu/article/variety-not-volume-is-driving-big-data-initiatives/

développeurs se sont tournés depuis longtemps vers le développement de solutions de stockage distribué, afin de contourner ces limitations.

Dans les cas où les données peuvent être représentés sous la forme de fichiers, les solutions de type NAS/SAN, réseaux P2P et aussi le stockage dans des infrastructures de type (*clouds*) représentent des choix technologiques capables d'offrir un stockage à grande échelle pour un coût raisonnable. Ces choix concernent aussi les bases de données de type relationnelle ou NoSQL, mais dans ce cas l'accès aux données requiert toujours une entité (pseudo)centralisée capable d'agréger et de présenter les données (ce qui n'exclut pas le traitement parallèle des requêtes).

À travers différentes stratégies, ces solutions distribuées proposent des solutions transparentes à l'augmentation des besoins de stockage, tout en offrant suffisamment de garanties pour assurer la consistance et la pérennité des données. Aujourd'hui, l'utilisation de solutions de stockage de fichiers sur un NAS/SAN ou sur le *cloud* est devenue aussi courante que l'utilisation de disques ou clés USB, une fois que les ressources potentiellement illimitées offerts par les réseaux P2P ou les infrastructures de type *cloud* présentent plusieurs avantages en ce qui concerne le coût, la disponibilité et l'utilisation des ressources physiques. Cependant, ces solutions peuvent aussi présenter des inconvénients liés à la vitesse d'accès et à la sécurité des données ; la solution à ces inconvénients est encore loin d'être garantie et dépend majoritairement des solutions propriétaires proposées par les fournisseurs des services de stockage. Un autre aspect à considérer est la compatibilité entre les systèmes : si certaines APIs rendent la manipulation des fichiers relativement simple, l'intégration d'autres représentations de données est moins évidente. Nous considérons qu'un système de gestion de données doit être capable aussi d'accéder à des requêtes sur une base de données, des flux de données ou encore faire appel à des Web services, le tout d'une manière uniforme et transparente pour l'utilisateur.

C'est dans le but de proposer une architecture unifiée pour les données et les services que nous avons présenté la spécification de la plateforme GRAPP&S (*GRid Applications and Services*), une architecture générique pour l'agrégation de données et services. Ce framework a été conçu de manière à intégrer de manière transparente les données de type fichier mais aussi les bases de données, les flux (audio, vidéo, données de capteurs et de l'Internet des Objets), les services Web et le calcul distribué. À travers une structuration hiérarchique basée autour du concept de "communautés", GRAPP&S rend possible l'intégration de sources de d'information disposant de protocoles d'accès hétérogènes et des règles de sécurité variés.

4.2 L'Architecture GRAPP&S

4.2.1 Définitions

Pour la définition de l'architecture GRAPP&S nous considérons un modèle de communication représenté par un graphe non orienté et connexe $G = (V, E)$, où V désigne l'ensemble des nœuds du système et E désigne l'ensemble des liens de communications qui existent entre les nœuds. Le modèle utilisé pour notre système est étudié dans [32]. Deux nœuds u et v sont dits adjacents ou voisins si et seulement si u, v est un lien de communication de G. $u_i, v_j \in E$ est un canal bidirectionnel connecté au port i pour u et au port j pour v. Donc les nœuds u et v peuvent mutuellement envoyer ou recevoir des messages en mode asynchrone.

Un message m en transit est noté $m(id(u), m', id(v))$ où $id(u)$ est l'identifiant du nœud qui envoie le message, $id(v)$ est l'identifiant du nœud de réception, m' indique le contenu du

message. Chaque nœud u du système a un identifiant unique id et dispose de deux primitives : `send(message)` et `receive(message)`. Par soucis de clarté, nous introduisons quelques définitions.

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

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

4.2.2 Communication et les Réseaux *Overlay*

Le modèle de communication présenté en Section 4.2.1 est suffisamment générique pour ne pas inférer sur la manière dont les messages sont effectivement livrés, se limitant uniquement à la définition des propriétés de communication bidirectionnelles entre deux vertex. Pour cette raison, l'architecture de GRAPP&S peut s'appuyer sur n'importe quel un réseau de communication *overlay* qui garantit une communication bidirectionnelle fiable et qui permet d'explorer différents chemins de communication pour chaque arête dirigée (réseau routé). Ceci donne une plus grande liberté d'implémentation et d'adaptation à l'environnement d'exécution, vu que les opérations send/receive peuvent être implémentées de différentes façons, selon les capacités de communication des nœuds. Dans ce cas, trois scénarios principaux peuvent être considérés :

— `Push`, où l'émetteur est capable d'envoyer un message directement au destinataire ;
— `Pull`, où le récepteur cherche régulièrement des messages en attente (ce modèle est fréquemment utilisé dans le cas des réseaux derrière un NAT/pare-feu) ; et
— `Proxy`, où les voisins doivent passer par un nœud intermédiaire afin d'échanger des messages (par exemple, grâce à un *middleware publisher/subscriber*).

Dans ces trois scénarios, il est toujours possible d'établir un voisinage direct ou partiel entre les processus, ce qui est compatible avec le modèle par graphes connectes dirigés et qui répond donc aux besoins de l'architecture GRAPP&S.

4.2.3 Éléments de l'Architecture GRAPP&S

Afin de présenter notre architecture, nous introduisons dans un premier temps quelques notations. Une communauté (C_i) est une entité autonome, qui regroupe des nœuds qui peuvent se communiquer et qui partagent une propriété définie : même localisation, même autorité d'administration (des serveurs distants appartenant à la même entreprise, par exemple) ou même domaine d'application (base de données métier, par exemple). Une communauté contient un seul processus *Communicator* (c) et au moins un processus *Ressource Manager* (RM) et un *Data Manager* (DM) et ces processus sont organisés de façon hiérarchique dans une communauté. L'interconnexion entre différentes communautés C se fait grâce à des liens de voisinage point-à-point entre les processus Communicator.

Communicator (*c*)

Le nœud *Communicator* (c) joue un rôle essentiellement lié au transport d'informations et à l'interconnexion entre différentes communautés, comme par exemple lors du passage de

messages à travers des pare-feu. C'est le point d'entrée de la communauté, et il assure sa sécurité vis-à-vis de l'extérieur, grâce à l'établissement de *Service-Level Agreements* (SLAs) avec les autres communautés. De même, le communicateur coordonne la sécurité intérieure de la communauté, et peut modifier ses politiques d'accès grâce à des décisions prises au sein de la communauté [59]. Un nœud *c* dispose d'un identifiant unique (ID) à partir duquel on construit les identités des autres nœuds de la communauté. Ce nœud ne stocke pas de données et ne fait pas d'indexation.

Ressource manager (*RM*)

Les processus *Ressource Manager (RM)* assure l'indexation et l'organisation des données et services dans la communauté. Ils reçoivent les requêtes des utilisateurs et assurent leur prétraitement. Les nœuds *RM* participent à la recherche de données dans la communauté. Pour des fins de tolérance aux fautes et performance, les informations indexées par les *RM* peuvent être redondantes et/ou partiellement distribuées (DHTs, par exemple). Afin de rendre plus performante la coordination des RMs, nous préconisons l'élection d'un RM désigné (voir section 4.3).

Data manager (*DM*)

Les processus *Data Manager (DM)* interagissent avec les sources de données, qui peuvent être dans différents supports tels que les bases de données (objet ou relationnelle), les documents (texte/XML/multimédia), des flux (vidéo, audio, VoIP), des données issues de capteurs ou encore une service hébergé dans le *cloud*. Un nœud DM est un service qui dispose des composants suivants :

(i) une interface (proxy) adaptée aux différentes sources de données (disque dur, serveur WebDAV, FTP, base de données, stockage sur *cloud* type Dropbox, etc.) et reliée à ceux-ci par un protocole de connexion spécifique au type de donnée, par exemple JDBC, ODBC, FTP, etc ;

(ii) un gestionnaire de requêtes qui permet d'exprimer des requêtes locales ou globales ; et

(iii) un gestionnaire de communication qui permet au nœud *DM* de communiquer avec le nœud *RM* auquel il est connecté.

4.3 Gestion de la Communauté

GRAPP&S peut être déployé dans plusieurs types d'architecture selon le placement des nœuds. Dans le modèle de placement (*i*), les nœuds peuvent être regroupés dans une seule machine physique (voir Figure 4.1a). C'est l'exemple typique d'une machine d'un particulier, qui souhaite héberger une communauté de l'architecture. Le placement des nœuds sous cette forme peut être justifié par sa simplicité à mettre en œuvre lors de sa phase d'implémentation, en utilisant les concepts d'héritage et de polymorphisme. Les nœuds sont interconnectés par des sockets, des solutions RPC pour qu'ils puissent communiquer par message dans les deux sens entre deux nœuds.

Dans (*ii*) les nœuds sont organisés dans une ferme de serveurs telle qu'un *cluster*, ce qui est caractéristique des réseaux HPC (Figure 4.1b). Finalement, (*iii*) les nœuds peuvent être regroupés s'ils partagent une même propriété de localisation ou d'administration (voir Figure

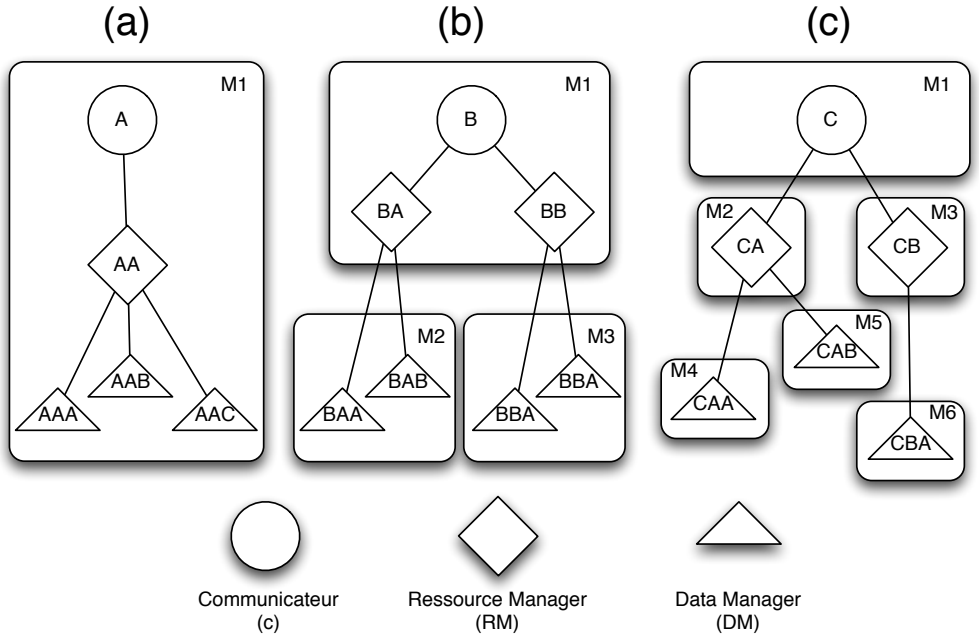

FIGURE 4.1 : Organisation des nœuds (a) dans une machine, (b) dans un *cluster* et (c) dans un réseau

4.1c). Ceci est l'exemple d'un réseau formé par les nœuds d'une entreprise ou d'un laboratoire de recherche.

Chaque nœud de GRAPP&S a un identifiant (ID) unique. Les adresses IP ou MAC ne sont pas des identifiants suffisamment précis car ils ne permettent pas d'identifier de manière unique les différents nœuds qui peuvent résider sur une même machine (par exemple, un RM et plusieurs DM). De plus, l'utilisation des adresses IP ou MAC ne garantit pas une identité unique, vu que les adresses IP privés peuvent être réutilisées tout autant que les adresses MAC. En effet, l'utilisation massive de la virtualisation commence à poser des problèmes vis-à-vis de la réutilisation des adresses MAC, posant aussi des problèmes au niveau du le déploiement des réseaux IPv6 [2]. Ainsi, nous préconisons l'utilisation d'un mécanisme d'adressage inspiré par JXTA [8]. Dans le cas de GRAPP&S, chaque nœud dispose d'une chaîne unique ID_{local} de 128bits, sous la forme "`urn:nom_communaute:uuid:chaine-de-bit`". L'expression de l'adressage hiérarchique se fait par la concaténation des IDs sous forme de préfixe, c'est-à-dire., l'ID du nœud c_i est équivalent à son ID_{local}, l'ID du nœud RM_i est formé par ID_{c_i}/ID_{RM_i}, et l'ID du nœud DM_i présente la forme $ID_{c_i}/ID_{RM_i}/ID_{DM_i}$.

Un avantage de l'utilisation d'un modèle d'adressage propre à GRAPP&S est que cela le rend indépendant du modèle d'adressage du réseau *overlay* sur lequel GRAPP&S est implémenté. Ainsi, deux communautés GRAPP&S implémentées sur des *middlewares* différents (FreePastry [3] et Phex [4], par exemple) seront toujours compatibles, une fois la connexion établie entre leurs *communicator*.

2. `http://tools.ietf.org/html/draft-gont-v6ops-slaac-issues-with-duplicate-macs-00`
3. `http://www.freepastry.org`
4. `http://www.phex.org`

4.3.1 Gestion des Nœuds

La topologie du réseau change fréquemment à cause de la mobilité des nœuds. Nous travaillons dans l'hypothèse où les tout nœud qui arrive dans le réseau est initialement un nœud DM. Selon les conditions de l'environnement où ce nœud se trouve, il peut se voir attribuer des rôles supplémentaires et "monter" dans la hiérarchie.

Connexion d'un nœud

Quand un nœud DM arrive dans le réseau, il dispose de deux moyens pour trouver un nœud RM sur lequel il peut se connecter :

- Si le nœud DM_i connait un ou plusieurs nœuds RM, il envoie un message de diffusion `Hello()` et collecte toutes les identités des nœuds RM, qu'il garde dans un tableau ordonné par l'identifiant. Il peut ainsi se connecter au nœud ***RM*** qui a l'identifiant le plus grand. Si ce dernier se déconnecte, alors le DM_i le supprime du tableau et se connecte au nœud RM suivant ;
- Si par contre le nœud DM ne connait aucun nœud RM, il doit effectuer une découverte sur le réseau local (par exemple, grâce à un multicast) ou contacter un service d'annuaire qui peut indiquer l'identifiant d'un nœud RM_i. Comme la manière de trouver le nœud RM_i dépend de l'implémentation, elle n'est pas précisée dans notre architecture.

Finalement, si aucune tentative de connexion à un nœud RM (et par extension, un nœud c) ne réussit, le nœud DM a la possibilité de former sa propre communauté. Il assume ainsi les trois rôles c, RM et DM, jusqu'à ce que d'autres nœuds le rejoignent. À ce moment, une élection pourra avoir lieu afin de redistribuer les rôles entre les nœuds.

Déconnexion d'un nœud

Les nœuds peuvent subir des déconnexions volontaires ou involontaires (pannes). Comme le cas des déconnexions volontaires est trivial, nous nous concentrons ici sur les déconnexions involontaires.

Entre deux niveaux hiérarchiques, les pannes peuvent être détectées soit par des messages périodiques de type *Pull* (aussi connu comme *heartbeat*), à la demande par des messages *Push* (*ping-pong*) [33] ou encore en s'appuyant sur un mécanisme propre au *middleware overlay*. Pour les nœuds appartenant à un même niveau hiérarchique, la surveillance peut aussi se faire grâce à un mécanisme de passage de jetons "de service". Cela permet non seulement l'allégement du mécanisme de détection (il suffit de surveiller son prédécesseur et son successeur) comme permet la diffusion rapide des informations à l'ensemble des nœuds.

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

Lorsque aucun message du nœud N_k n'est reçu jusqu'à l'expiration du temps ta_k, une suspicion de défaillance est levée et doit être vérifiée auprès d'un deuxième nœud qui est aussi en contact direct avec le nœud suspect. Ainsi, si la suspicion concerne le nœud c, un nœud RM_i interroge son voisin direct RM_{i+1} avec un message jeton initialisé à `faux`. Si RM_{i+1} a reçu un message du nœud c avant l'expiration de son temps d'attente ta_c, RM_{i+1} modifie la valeur du jeton à `vrai` et retourne le message jeton à son émetteur RM_i. Ceci signifie (indirectement)

que le nœud c n'est pas déconnecté et le nœud RM_i peut envoyer à nouveau un message au nœud c. Si par contre RM_{i+1} n'a pas été contacté récemment par c, il fera suivre un jeton la valeur `faux` qui, grâce au passage du jeton, alertera tous les nœuds RM $\{RM_1, , RM_n\}$ de la défaillance de c.

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

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

4.3.2 Coordination entre les Nœuds

Vu le caractère dynamique et volatile des réseaux informatiques, il est important de garantir la coordination entre les nœuds, notamment dans le cas des RM. Une manière simple et performante de faire ceci est de définir un nœud avec des responsabilités étendues au sein de son groupe, ce choix étant fait par exemple grâce à une élection.

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

Il faut noter que la connaissance préalable des nœuds du niveau supérieur n'est pas obligatoire, vu que différentes techniques permettent d'obtenir les identifiants des autres nœuds. La méthode la plus simple consiste à utiliser directement le mécanisme d'adressage GRAPP&S : étant indépendant du *middleware* de communications, ce système d'adressage permet facilement de remonter la hiérarchie GRAPP&S et de contacter d'autres nœuds (grâce au routage du réseau *overlay*). Il suffit donc de remonter les niveaux de son propre identifiant ou de contacter d'autres nœuds dont on récolté les identifiants (ceux dont on a reçu des requêtes récemment, par exemple). Cette technique permet aussi de contacter d'autres *communicators* c_j et de réintégrer un réseau de communautés auprès la déconnexion involontaire de son *communicator* c_i. En dernier recours, GRAPP&S peut s'appuyer sur les éventuels mécanismes de découverte de topologie (broadcast/multicast) offerts par le propre *overlay*.

Vu que le problème de la reconnexion au reste de la communauté peut être traité de manière plus ou moins simple au sein de la propre architecture GRAPP&S, il est intéressant de se pencher sur les algorithmes d'élection eux-mêmes. Dans GRAPP&S, nous préconisons un algorithme d'élection distribué inspiré des protocoles de routage OSPF et IS-IS [111, 106, 20]. En effet, les nœuds GRAPP&S disposent d'un identifiant unique qui peut être utilisé de manière systématique par ces algorithmes d'élection.

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

fort taux de volatilité peut occasionner des élections à répétition, soit lors de la déconnexion du leader, soit lors de la connexion d'un nœud avec un identifiant prioritaire.

Dans les cas où la volatilité risque d'impacter la performance du réseau, il est possible d'utiliser un mécanisme non-déterministe comme celui d'OSPF [20]. Dans ce type d'algorithme, plus conservateur, le choix d'un leader (*DR - Designated Router*) n'est nécessaire que si le leader actuellement en place disparaît. Ainsi, l'entrée de nouveaux nœuds dans la communauté a un impact moins important sur le fonctionnement du réseau.

4.4 Opérations dans GRAPP&S

4.4.1 Stockage et Indexation

Le stockage de donnée dans le réseau GrAPP&S fait intervenir les nœuds Data Manager DM, alors que les nœuds Ressource Manager RM permettent d'indexer les données et les services. À la fin, chaque donnée est identifiée de manière unique grâce à l'identifiant du nœud DM, auquel s'ajoute une extension contenant des informations et le type MIME des données. Ceci permet de franchir la barrière du simple "nom de fichier", et peut donc faire cohabiter des données statiques (fichiers), des données dynamiques (requêtes sur une base de données, résultats d'un calcul) et des données à caractère temporaire (flux voix ou vidéo, état d'un capteur, etc.).

L'ajout d'une nouvelle donnée dans le réseau se fait ainsi : quand un nœud DM_i arrive dans le réseau, il se connecte à un nœud RM et publie les caractéristiques de ses données pour être indexées. Toute modification des données sur un DM sont propagées au RM auquel il est connecté, qui par la suite peut mettre à jour ces informations et les partager avec les autres RM.

Cette propagation des informations peut prendre différentes formes selon les politiques utilisées lors de l'implémentation du réseau des RM. Une implémentation qui veut garder la simplicité pourra simplement garder un index local sur chaque RM, qui sera consulté lors d'une recherche. Au contraire, une implémentation souhaitant minimiser les échanges lors d'une recherche de données penchera sur l'utilisation d'un super-nœud au sein des RMs ou d'un mécanisme de DHT. Il est aussi possible de favoriser la réplication des index et des données, ce qui exige une coordination entre les RM afin de garder la cohérence des copies. Dans tous les cas, la surcharge des fonctionnalités d'un nœud ("super-node") n'est pas une obligation dans notre structure mais simplement une spécificité pouvant être présente dans une implémentation donnée.

4.4.2 Recherche

Quand un client cherche une donnée sur GRAPP&S, il entre en contact avec un proxy DM_i, qui envoie une requête Y contenant des informations qui identifient la donnée ou le service. Cette recherche dans une communauté de GrAPP&S se fait par paliers, de manière à respecter l'organisation hiérarchique du réseau. La Figure 4.2 illustre une partie de cette procédure de recherche :

1. Un nœud $DM_i \in C_i$ envoie la requête a son nœud $RM_i \in C_i$;
2. RM_i vérifie dans son indexe s'il y a parmi ses voisins un DM qui contient la donnée recherchée ;

3. Si oui, alors le nœud RM_i retourne au nœud DM_i une liste de nœuds DM qui contiennent l'information recherchée ;
4. Sinon, le nœud RM_i fait suivre la requête, soit directement aux voisins $RM_k \in C_i$ (si le mécanisme de communication le permet), soit à son nœud $c_i \in C_i$ pour retransmission aux autres $RM_k \in C_i$;
5. Quand un nœud $RM_k \in CM_i$ trouve la bonne réponse, alors la requête sera retournée au nœud DM_i émetteur en suivant le chemin inverse ;
6. Si la donnée recherchée n'est pas dans la communauté CM_i, alors le c_i fait suivre la requête vers d'autres communautés CM_j.

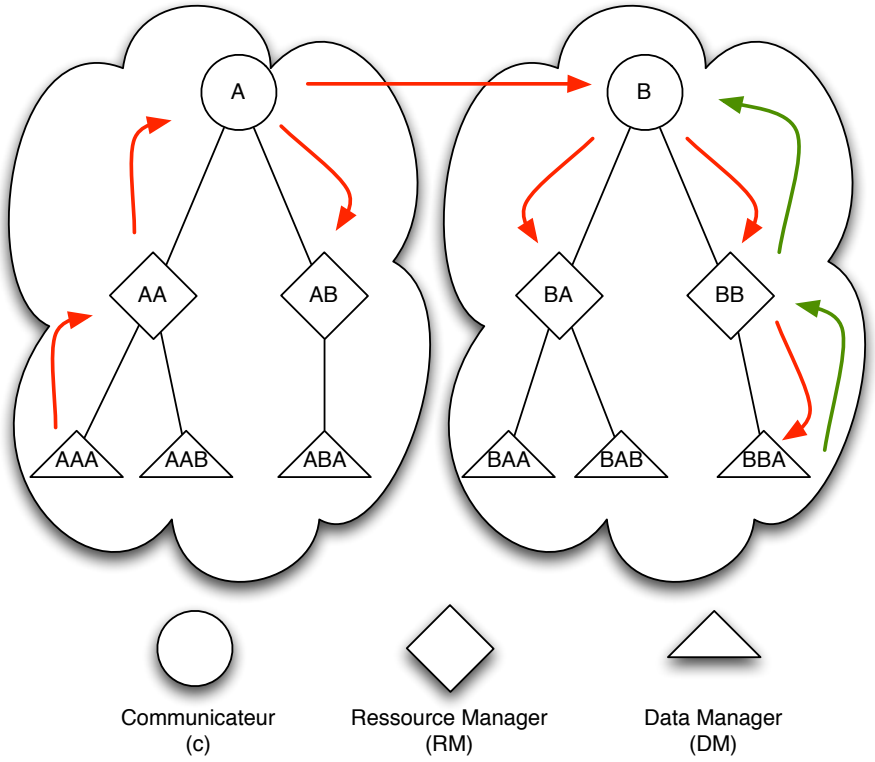

FIGURE 4.2 : Recherche d'une information dans GRAPP&S et mécanisme de routage préfixé

En cas de réussite, le client obtient l'identifiant du nœud DM_x responsable par la donnée. Dans ce cas, le client a deux possibilités pour accéder à la donnée, soit par connexion directe, soit par une connexion routée. Dans le cas de la connexion directe, le client fait une requête directe au nœud DM_x afin d'accéder à la donnée. Comme le client peut se trouver dans un autre réseau qui ne permet pas l'accès direct à DM_x, la connexion peut se faire par routage interne dans GrAPP&S. Cela se fait de la manière suivante :

— Le client, par intermède d'un nœud DM_i, envoie une requête `Req`$(id(DM_x, Y, id(c_i))$ au nœud communicateur c_i ;
— le nœud c_i, par routage préfixé, envoie la requête `Req` vers le nœud RM qui a indexé la donnée ;
— Ce dernier fait suivre la requête vers le nœud DM_x responsable de la donnée ;
— Une fois la donnée trouvée, le nœud DM_x retourne la bonne réponse au client en suivant le chemin inverse.

Ce mécanisme de recherche hiérarchique empêche l'inondation des liens du réseau. La hiérarchisation permet de définir des chemins par lesquels transitent les requêtes et comme la connectivité logique de notre architecture est par définition de $(n-1)$, il suffit d'appliquer un algorithme de type PIF (*Propagation of Information with Feedback* [128]) pour agréger les requêtes et réduire le nombre de messages d'une recherche.

4.5 Bilan et Perspectives

Ce chapitre présente une spécification pour une plateforme distribuée basée sur les principes des réseaux hiérarchiques et visant l'établissement d'une base documentaire générique et extensible. Contrairement aux autres travaux que j'ai pu développer autour des systèmes distribués, l'objectif ici n'était pas celui de la distribution de tâches de calcul et leur gestion mais plutôt celui de l'indexation et de la recherche de documents et des sources d'information, le tout afin de minimiser l'hétérogénéité dans l'accès aux données.

Bien que présente dans la spécification proposée, la coordination des nœuds fut d'abord conditionnée à la création et à la maintenance de réseaux de proximité pour cette gestion documentaire. La spécification de GRAPP&S a aussi une qualité moins évidente, celle de l'indépendance vis-à-vis des technologies d'implémentation. De par sa propre organisation en "communautés", différentes instances de GRAPP&S peuvent s'interconnecter et échanger des données, selon des politiques SLA définies par chaque partie.

Également, la spécification de GRAPP&S a permis une meilleure compréhension des différents mécanismes liés à la coordination et gestion des nœuds dans les réseaux hiérarchiques. En effet, la majorité de mes travaux repose sur une organisation uniforme du réseau où tous les nœuds possèdent les mêmes prérogatives ou au moins disposent de mécanismes de communication directe entre-eux. Dans le cas d'un réseau hiérarchique, les prérogatives dépendent du rôle joué par les nœuds, ce qui apporte des défis différents de ceux que je traite habituellement, soit dans le cas des réseaux P2P, soit dans le cas de l'algorithmique distribuée [142, 136].

Après avoir conclu la spécification de GRAPP&S, les travaux de thèse de M Diallo se sont tournés vers la recherche et la description de scénarios d'applications, comme par exemple dans le cas de l'*e-learning*. Malheureusement, cela s'est fait au détriment de l'implémentation d'un prototype. Certains concepts développés restent néanmoins présents dans les autres travaux que je développe, comme par exemple le concept de communautés. D'une autre part, je développe des activités d'enseignement autour du *big data* qui me permettent d'être en contact direct avec les nouvelles technologies dans ce domaine.

Chapitre 5

CloudFIT, un Intergiciel pour le *Fog Computing* et l'Internet des Objets

Résumé

CloudFIT est un intergiciel pour le calcul distribué particulièrement conçu pour une exécution en environnements hétérogènes. Cette hétérogénéité peut assumer plusieurs formes parmi celles citées dans les chapitres précédents : ***l'hétérogénéité matérielle****,* ***l'hétérogénéité des communications****,* ***l'hétérogénéité des tâches de calcul*** *et la* ***dynamicité des ressources****. En effet, CloudFIT peut être déployé sur un environnement pervasif composé de nœuds présentant des caractéristiques très diverses (allant des petits nano-ordinateurs jusqu'aux serveurs HPC). De plus, il supporte des tâches de calcul irrégulières selon le paradigme FIIT ainsi que des variantes où les tâches présentent un certain niveau de dépendance (DAGs, Map-Reduce, etc.).*

Pour cela, CloudFIT repose sur un overlay pair-à-pair (P2P) qui gère les aspects liés à l'interconnexion des nœuds, alors que les services CloudFIT s'occupent de la gestion des tâches de calcul. D'un point de vue structurelle, la spécification de CloudFIT est modulaire, offrant au concepteur la possibilité d'intervenir sur la totalité de la pile logicielle. Cette modularité non seulement garantit une indépendance vis-à-vis des bibliothèques existantes comme fait de CloudFIT une plateforme idéale pour le prototypage de nouvelles technologies dédiées à l'Internet des Objets (IoT), au big data, aux environnements pervasifs et au fog computing.

CloudFIT a été initialement développé dans le cadre du projet STIC-AmSud PER-MARE, où il a été utilisé pour l'étude sur le déploiement d'une application big data dans des environnements pervasifs. Au fil des années, son développement a pu bénéficier de multiples collaborations (avec l'Université Paris 1, par exemple) et aujourd'hui il est appliqué dans le cadre du projet international CAPES-Cofecub MESO, portant sur la physique de l'atmosphère.

Dans ce chapitre nous allons présenter les éléments principaux de l'architecture de CloudFIT et les mécanismes liés à la gestion des noeuds et des tâches, ainsi que des résultats obtenus jusqu'à présent. On discutera aussi des futures directions de recherche, notamment la mis en place d'un réseau multi-échelle pour le fog computing ou des stratégies pour accélérer l'exécution d'applications big data.

5.1 Introduction

Avec l'augmentation exponentielle du nombre de dispositifs informatiques de proximité (*smartphones*, tablettes, nano-ordinateurs, etc.), il est important de comprendre comment organiser ces ressources et comment gérer l'information dans ces environnements hétérogènes et dynamiques (que nous appellerons "environnements pervasifs"). En effet, notre vie quotidienne présente de plus en plus d'appareils connectés pour suivre notre santé et nos mouvements, la sécurité de nos maisons ou le stress hydrique de nos plantes et de nos cultures.

De nos jours, les limitations majeures pour une utilisation orchestrée de ces dispositifs ne se trouve pas en leur capacité de calcul ou de communication (WiFi, *Bluetooth*, etc.), mais surtout à la difficulté d'exploiter et de coordonner ces appareils. Heureusement, cette frontière est en train de tomber avec l'avènement de l'Internet des Objets (*Internet of Things* - IoT). L'IoT représente une nouvelle tendance de l'industrie informatique où l'environnement physique est peuplé d'objets interconnectés et communicants, lesquels interagissent les uns avec les autres et avec l'environnement lui-même. La force de ce concept réside dans l'intégration transparente des capteurs, des actionneurs et d'autres dispositifs, ce qui permet la collecte et le traitement d'informations à grande échelle mais aussi la prise de décisions au plus proche des utilisateurs. À vrai dire, l'augmentation de la bande passante et de la puissance de calcul des appareils, couplés avec un coût décroissant de capteurs [85], nous permettent d'envisager des applications et des stratégies de traitement de données bien différentes de celles développées jusqu'à aujourd'hui.

En effet, la plupart des applications courantes repose sur un modèle client-serveur. Dans le cas des premières solutions IoT sur le marché, l'agrégation et l'analyse des données sont effectuées principalement sur des infrastructures déportées de type *cloud computing*. Plusieurs travaux [105, 69, 56] comptent sur ces infrastructures car elles offrent autant la puissance de calcul que la flexibilité pour l'exécution de services et des applications [130]. Malgré ces avantages, les infrastructures de type *cloud* ont aussi quelques inconvénients importants. En effet, comme nous avons vu en Chapitre 1, le transfert de données à longue distance peuvent induire des délais considérables et ralentir le traitement et la prise des décisions. En outre, les applications qui dépendent entièrement des services distants peuvent échouer si la connexion est défectueuse ou trop lente.

Par conséquent, nous devons repenser la façon de transmettre, de stocker et d'analyser les données dans ces environnements. Les architectures réseaux traditionnelles ne sont pas préparées pour l'hétérogénéité qui caractérise l'IoT (à la fois sur les capacités de calcul, de mémoire, d'autonomie et de communication), ni sont préparées pour la nature spontanée de leurs interactions. Cette préoccupation a conduit les chercheurs à développer une série de solutions alternatives au *cloud computing*, telles que les *grids* pervasifs [112], le *mobile edge computing* [46, 54, 127], le *fog computing* [21] ou bien le *edge-centered computing* [63]. Toutes ces alternatives partagent un même objectif : utiliser la puissance de calcul des dispositifs environnants pour effectuer des tâches habituellement déléguées à une installation distante.

La plateforme CloudFIT a été développé afin de fournir un support logiciel à ce type de calcul distribué mais surtout dans le but d'offrir un cadre expérimental où nous pouvons intervenir sur toute la pile logicielle et tester différents concepts issus de nos recherches. En effet, l'expérience passée avec des plates-formes de calcul distribué tiers (CONFIIT [58], Apache Hadoop [4, 135], etc.) nous fait prendre conscience de la complexité de ces systèmes et des limitations à leur extension ou modification. Le fait de pouvoir spécifier, contrôler et modifier l'ensemble des composants de la plateforme offre des innombrables possibilités pour l'investigation de

nouveaux concepts ou tous simplement pour l'adéquation à des nouveaux besoins.

Dans les sections suivantes nous allons présenter les solutions architecturales et pratiques utilisées pour le développement de CloudFIT, des cas d'usage et aussi des directions de recherche pour l'avenir de cette plateforme.

5.2 État de l'Art et Définitions

La diffusion des dispositifs de proximité avec des capacités de calcul non-négligeables (*smartphones*, tablettes, ordinateurs portables et nano-ordinateurs tels que le Raspberry Pi) encourage l'intégration de ces dispositifs dans le traitement des données, à l'opposé d'une approche purement "client-serveur" où tout le stockage et le traitement des données se fait sur un ou plusieurs serveurs distants (serveurs, *clusters*, *data centers*, infrastructures *cloud*). C'est ainsi que des approches telles que les *grids* pervasifs [112], le *mobile edge computing* [46, 54, 127], le *edge-centered computing* [63] ou bien le *fog computing* [21] ont été proposées dans le but de placer certaines applications et services au plus près de l'utilisateur final.

Les travaux sur l'*edge computing* et le *fog computing* partagent souvent les mêmes définitions [151]. En effet, le *fog computing* a été défini par CISCO [41] comme "un paradigme qui étend le *cloud computing* et ses services à la périphérie du réseau", tandis que le *(mobile) edge computing* vise à transformer les stations de base proches en "centres de services intelligents capables de fournir des services hautement personnalisés" [151]. Plus globalement, le *fog computing* fait référence à une infrastructure décentralisée dans laquelle les ressources de calcul sont distribuées de manière intelligente sur des emplacements répondant au mieux aux besoins des applications, comme par exemple la latence des communications, la bande passante ou les besoins de confidentialité. En effet, le terme *fog* (un brouillard) exprime l'idée que les services doivent se rapprocher et entourer les utilisateurs et les sources de données, au lieu de rester distant comme dans le cas des *clouds*. La Figure 5.1 représente une tentative de représentation conceptuelle de la relation entre l'IoT, le *fog* et le *cloud*.

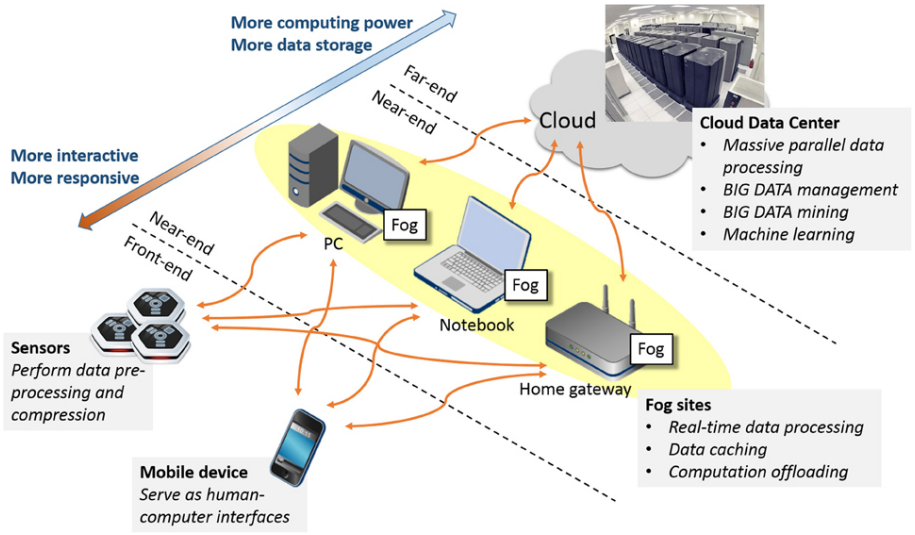

FIGURE 5.1 : Architecture conceptuelle d'une infrastructure *fog/cloud* [165]

On retrouve plusieurs tentatives de spécifier l'organisation ce cette infrastructure. Des exemples de *edge/fog* comprennent les services *fog* [21] et les *cloudlets* [127], tous les deux proposant le déploiement des serveurs de proximité capables d'offrir des services avec une latence

réduite. A quelques exceptions près, comme [46], ces travaux considèrent que les dispositifs IoT ne contribuent pas à l'effort de calcul, restant dépendants d'un service tiers (à proximité ou à distance).

Garcia Lopez et al. [63] explorent une autre facette de l'*edge computing* en se concentrant sur le rôle de l'homme dans la boucle de contrôle. En effet, ces auteurs affirment la nécessité de recentrer le contrôle sur les équipements situés au bord du réseau, au lieu de simplement les considérer comme une première couche de calcul reliée à un réseau plus grand et plus puissant. Malheureusement, dans cette définition, les dispositifs IoT ne contribuent pas non plus aux efforts de calcul, étant considérés comme des capteurs/actionneurs pilotés par les interactions entre l'homme et l'*edge*.

Bien que ces travaux préconisent la nécessité d'un environnement informatique de proximité, ils oublient souvent de détailler l'interconnexion ou les exigences de coordination entre les processus. Cela est particulièrement nécessaire dans l'optique de l'IoT, qui impose des défis importants pour l'évolutivité, la dynamicité et l'hétérogénéité des ressources.

Dans la littérature nous trouvons aussi la notion de *grid* pervasif [112], qui vise l'intégration des dispositifs de détection/d'actionnement ainsi que des systèmes de haute performance classiques. Ces *grids* reposent sur l'utilisation des ressources habituellement sous-utilisés, composant ainsi une plateforme de calcul dynamique [141]. En effet, les *grids* pervasifs offrent la possibilité d'intégrer les différentes ressources disponibles allant des petits appareils de type Raspberry Pi jusqu'aux machines virtuelles déployées sur les infrastructures d'un data-center. Pour l'Internet des Objets, les *grids* pervasifs représentent une opportunité de déployer des tâches informatiques sur des ressources situées à proximité des dispositifs IoT, minimisant ainsi le transfert de données vers un réseau distant. De plus, selon les besoins, ces tâches peuvent être allouées aux ressources avec la capacité de calcul adéquate à chaque service, sans avoir à externaliser les données et les services.

Une caractéristique souvent avancée par les idéalisateurs des *grids* pervasifs est leur indépendance par rapport à des architectures et des services opérateurs. Malgré l'appel à la décentralisation et à l'affranchissement du "tout *cloud*" prôné par les premiers travaux sur l'*edge computing* et le *fog computing*, nous observons une appropriation de ces concepts par les grands opérateurs du marché tels que Cisco, Intel ou Microsoft. En effet, ceux-ci se sont réunis au sein de l'*Open Fog Consortium* [1] afin de créer une architecture de référence pour le *fog computing*. Bien que de telles initiatives sont nécessaires pour la maturation d'une technologie, elles sont souvent source de contraintes. En effet, l'une de nos préoccupations lors du développement de CloudFIT est de le maintenir le plus léger possible, avec une dépendance logicielle réduite et une empreinte mémoire minimale afin de pouvoir le déployer sur le plus grand nombre de dispositifs, inclusive des nano-ordinateurs.

Finalement, afin de répondre aux différents besoins des architectures et applications IoT, nous pouvons explorer le concept des systèmes multi-échelle. Les *systèmes multi-échelles* sont des systèmes distribués où les services sont organisés en couches à travers une ou plusieurs dimensions (dispositifs, réseau, localisation géographique, etc.), chaque couche fournissant un niveau de service supplémentaire qui peut être consulté en fonction du contexte de l'appareil [123, 124]. En vertu de cette approche, des actions primaires peuvent être décidées/interprétées à proximité, tandis qu'une analyse plus poussée de l'information peut être effectuée par des serveurs externes. Cette analyse stratifiée peut également être utilisée pour renforcer les aspects liés à la vie privée comme, par exemple, l'anonymisation des données qui seront externalisés.

1. `http://www.openfogconsortium.org/`

Nous croyons que ce concept offre la granularité et les modalités d'interconnexion nécessaires à l'autonomie des dispositifs IoT et permet des services plus réactifs et de meilleure qualité car exécutés au plus près des dispositifs et des utilisateurs. Ceci est notamment utile dans des domaines tels que la domotique, où l'adaptation au contexte et le respect de la vie privée sont de facteurs clés.

La diversité de travaux autour du *fog computing* n'est pas nécessairement suivie par une offre en outils ou en plateformes pour sa mise en œuvre. En effet, la plupart des auteurs cherchent encore la meilleure manière de déployer et de coordonner les nœuds dans des tels environnements. Bien que souvent cités, des approches basées sur la virtualisation [127], les *micro-clouds* [51], les micro-services [152] ou les *workflows* [72] ne sont que des possibilités pour la mise en place du *fog*. Comme remarqué par *Yi et al.* [162], les challenges sont multiples et incluent aussi la gestion du réseau (via les *Software Defined Network* - SDN- ou les *overlays* P2P), le déploiement, l'orchestration, la migration de tâches/services, etc. L'absence de plates-formes vraiment dédiées au *fog computing* peut s'expliquer par le manque de standardisation. Ceci pourra changer avec la publication des spécifications de l'*Open Fog Consortium* (initialement prévue pour le début 2017 mais toujours pas publiées), mais, pour le moment, ces initiatives sont rares et limitées. Parmi les rares plates-formes opérationnelles dédiées au *fog computing*, on peut citer IOx de Cisco [40] et Paradrop [157]. La plateforme de Cisco repose sur l'hébergement de machines virtuelles sur des routeurs et switches compatibles, et pour cela les utilisateurs disposent de APIs et scripts pour créer et déployer leurs propres images et applications. Malheureusement le code source de IOx est fermé, empêchant toute extension ou étude plus poussée. ParaDrop, de son côté, se base sur un réseau de passerelles (installés, par exemple, sur les points d'accès WiFi ou sur les box Internet à la maison), mais celles-ci doivent se connecter à des serveurs ParaDrop, ce qui empêche la décentralisation du *fog*.

Dans un souci de simplicité, par la suite de ce chapitre on utilisera le terme *fog computing* indistinctement pour représenter les défis de l'*edge computing*, des *cloudlets*, des *grids pervasifs* aussi que pour tout autre réseau faiblement couplés caractérisé par l'hétérogénéité des ressources.

5.3 De CONFIIT à CloudFIT

CloudFIT est une plateforme de calcul distribué que reprend et élargit le paradigme FIIT (*Finite number of Independent and Irregular Tasks*) défini par Krajecki [92]. Par définition, un problème FIIT peut être décomposé en un ensemble de tâches qui respectent les trois conditions suivantes :

1. une tâche ne peut faire aucune hypothèse sur la résolution d'une autre ;
2. le temps d'exécution d'une tâche n'est pas prévisible ;
3. un algorithme unique est utilisé pour résoudre les tâches, seules les données en entrée changent.

Ce paradigme de computation permet la représentation de la plupart des problèmes de calcul parallèle qui ne requièrent pas une dépendance forte entre les tâches. Il faut noter que cette restriction peut être dépassée et donc supporter des applications plus complexes, grâce à l'utilisation d'une synchronisation, à petit ou gros grain :

Synchronisation à gros grain Avec une synchronisation à gros grain, deux *jobs* sont exécutés en séquence, ce qui permet la synchronisation à la fin de chaque exécution. Ce mo-

dèle correspond au modèle de programmation BSP (Bulk-Synchronous Parallel) [149], qui repose sur la succession de *supersteps*. Un *superstep* est défini comme une séquence d'opérations locales, suivies par une barrière globale de synchronisation, exactement ce que se produit à la fin de l'exécution d'un *job* FIIT. Comme dans BSP, aucune assomption n'est faite sur l'ordre d'exécution des tâches, la seule contrainte est que les données nécessaires au prochain *superstep* soient disponibles au moment de la barrière. La Figure 5.2 illustre l'exécution d'un *superstep* BSP dans lequel les nœuds exécutent leurs tâches et préparent les données pour le prochain *superstep* avant la barrière.

Synchronisation à grain fin La synchronisation à grain fin permet d'obtenir la dépendance entre les tâches, à l'instar des *Directed Acyclic Graph* (DAG). Pour cela, il suffit de modifier l'ordonnanceur de tâches afin de prendre en compte l'état des tâches et une liste de dépendances : une tâche ne sera lancée que si les tâches dont elle dépend sont déjà terminées. Bien que ceci viole la première propriété du modèle FIIT (l'indépendance entre les tâches), son implémentation est simple et permet le déploiement d'autres types d'application.

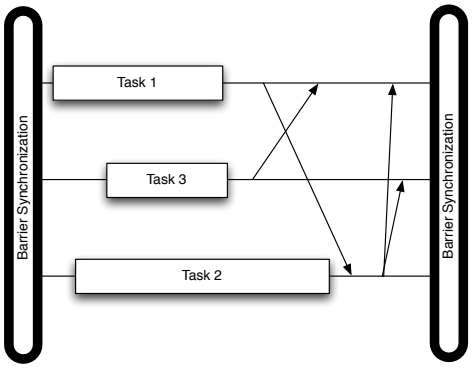

FIGURE 5.2 : Un superstep dans le modèle BSP

Une première implémentation du paradigme FIIT a vu le jour avec la plateforme CONFIIT (*Computation Over Network for FIIT*) [57, 58]. CONFIIT a été développé en tant que *middleware* pour le calcul distribué, en s'appuyant sur un anneau logique (*overlay*) géré par le *middleware* et sur des échanges XML-RPC entre les nœuds. À la suite de sa version initiale, CONFIIT a été fortement modifié entre 2004 et 2006, avec l'addition de différents modes de calcul (distribué, centralisé), isolation (*sandboxing*), observateurs extérieurs, etc. Ce développement a été fait notamment dans le cadre d'un projet supporté par l'agence ANVAR dans le but de créer une *startup* dans le domaine du calcul distribué. CONFIIT a été utilisé comme plateforme de calcul pour plusieurs travaux, notamment lors de la résolution parallèle des instances L(2,23) et L(2,24) du problème de Langford [81].

Lors du démarrage du projet STIC-AmSud PER-MARE, nous avons voulu utiliser CONFIIT comme plateforme pour l'exécution d'applications *MapReduce*, mais nous avons rencontré plusieurs difficultés qui n'ont pas permis l'utilisation de CONFIIT pour la suite du projet. En effet, nous avons trouvé une faille dans la conception de l'anneau logique qui empêchait l'utilisation de CONFIIT pour les applications de type *big data* : l'anneau était utilisé autant pour le passage des messages de service que pour la diffusion des données associées aux tâches. La transmission de masses de données plus importantes que quelques kilo-octets (ce qui était le cas avec

Langford) occasionnait la congestion de l'anneau logique et empêchait la synchronisation des nœuds, causant ainsi leur déconnexion.

Après plusieurs tentatives, nous avons pris la décision de développer une nouvelle plateforme, plus à jour et disposant de ressources capables de supporter aussi les applications de type *big data*. Cette nouvelle implémentation, appelée CloudFIT, sera décrite dans les sections suivantes.

5.3.1 Spécification des besoins

La décision d'implémenter une nouvelle plateforme capable de supporter le paradigme FIIT (et ses variantes) dans un univers d'applications allant du calcul combinatoire au *big data* a été suivie d'une liste d'exigences visant la généricité et la maintenance de la plateforme :

R1 CloudFIT doit être indépendant de l'*overlay* P2P. Ce choix rend possible le test de différents *overlays* P2P, afin de mieux s'adapter aux environnements et besoins des applications ;

R2 CloudFIT doit être modulaire afin de supporter la composition et l'ajout de nouveaux modules, grâce à des interfaces et services bien définis. De plus, ceci doit permettre à l'utilisateur de composer "sa" pile logicielle sans avoir à modifier le code source, par exemple, grâce à un fichier de configuration externe ;

R3 CloudFIT doit supporter le déploiement d'applications à la volée. Un utilisateur doit pouvoir soumettre ses propres classes applicatives à CloudFIT, qui les intégrera à la file d'exécution et les déploiera aux différents nœuds du réseau.

L'exigence [R1] vient directement de l'expérience précédente avec CONFIIT, où une dépendance trop forte par rapport au *middleware* P2P a rendu la correction des *bugs* trop difficile. Comme les environnements pervasifs présentent des grandes variations autant en termes de performance que de capacité, les couches les plus proches du réseau doivent pouvoir s'adapter à ces contraintes, comme par exemple l'impossibilité d'effectuer des diffusions, la présence de proxies et de passerelles NAT, voir même le support à des communications par mémoire partagée si l'environnement le supporte. Le respect à cet objectif a été très utile car l'*overlay* P2P initialement retenu (FreePastry [2]) s'est plus tard révélé peu performant et on a pu facilement migrer vers un nouvel *overlay*, TomP2P [3].

L'exigence [R2] est autant une prérogative visant l'évolution et la maintenance de la plateforme qu'une manière de rendre plus simple l'expérimentation avec CloudFIT. De plus, le fait de reposer sur des interfaces permet une moindre dépendance entre les modules, demandant peu ou aucune modification de code en cas de remplacement d'un composant.

Finalement, l'exigence [R3] garantit la scalabilité de la solution. En effet, il serait inconcevable de devoir déployer lapplication sur les nœuds de calcul avant de la lancer à laide de la plateforme, comme cétait le cas avec CONFIIT. En effet, CONFIIT exigeait que les binaires de lapplication soient déjà présents sur les nœuds avant même lexécution de la plateforme, ce qui n'est pas toujours évident à faire dans un réseau dynamique où des nouveaux nœuds peuvent rejoindre la plateforme à tout moment. À notre avis, c'est bien le rôle de la plateforme d'effectuer ce déploiement et d'assurer le lancement des applications selon des appels bien définis dans l'interface applicative.

2. `http://www.freepastry.org/`
3. `http://tomp2p.net`

5.3.2 Architecture

L'architecture de CloudFIT a été conçue selon les objectifs cités précédemment. Afin de renforcer la modularité de la plateforme, nous l'avons conçue sous la forme d'une pile logicielle, inspirée des modèles réseau TCP/IP et OSI. Ainsi, nous avons défini quatre couches représentant les différentes fonctionnalités de la plateforme : **Network**, **Protocol**, **Service** et **Application**.

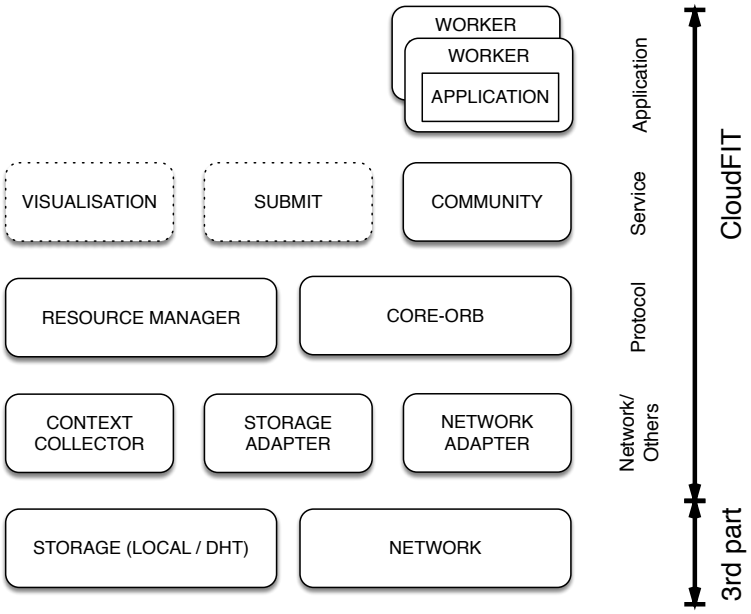

FIGURE 5.3 : Représentation simplifiée de la pile logicielle CloudFIT

Malgré son nom, la couche **Network** est responsable pour toute interaction avec les systèmes tiers sur lesquels CloudFIT s'appuie (*overlay* P2P, système d'exploitation, systèmes de stockage). Par exemple, la classe `Network Adapter` exécute les opérations élémentaires d'encapsulation et décapsulation des messages, grâce à des primitives conçues selon les capacités des *overlays* P2P subjacents (*send*, *sendAll*, *receive*, etc.). Le même principe s'applique à la classe `Storage Adapter`, où des primitives comme *read*, *write*, *delete*, *lookup* font l'interface avec les différentes solutions de stockage possibles (fichiers locaux, DHTs, bases de données, stockage sur le cloud). On y trouve finalement le `Context Collector`, déjà présenté dans le chapitre précédent en Section 3.3.1 et qui a été intégré à CloudFIT aussi.

La couche **Protocol** est responsable notamment par la gestion des messages et les ressources de calcul. Ainsi, le module `Core-ORB` (nommé ainsi car son rôle est comparable à celui d'un *Object Request Broker*) stocke les messages reçus de la couche Network et délivre ces messages aux services adéquats de la couche supérieure. Grâce à un mécanisme *publish-subscriber*, différents services peuvent s'enregistrer auprès le `Core-ORB`, obtenant ainsi un identifiant unique utilisé pour la réception des messages mais aussi pour des éventuelles communications entre les services dans le même nœud.

Le `Resource Manager`, de son côté, puise dans les informations extraites par le collecteur de contexte pour vérifier si les ressources présentes dans la machine sont compatibles avec les besoins des applications. Ces besoins sont exprimés en tant que propriétés (mémoire nécessaire, espace disque, etc.) renseignées par l'application lors de sa soumission. Le `Resource`

`Manager` est aussi responsable pour la gestion du pool de `Workers`, qui sont alloués aux applications selon les demandes faites par les ordonnanceurs de tâches.

La couche **Service** contient les services nécessaires à l'exécution des applications distribuées. À cette couche appartient notamment la classe `Community`, une abstraction d'un groupe de machines qui gère le déploiement des applications et la gestion des événements liés aux nœuds (entrée, sortie, retransmission de messages, etc.). CloudFIT définit une communauté par défaut regroupant l'ensemble des nœuds sur lé réseau, mais des instances supplémentaires peuvent être lancés afin de créer des sous-réseaux répondant à des besoins spécifiques (cloisonnement, localisation, etc.). Chaque communauté est associée à un `Job Scheduler`, qui gère la file d'applications soumises (*jobs*) et choisit lesquelles peuvent être lancées sur la machine, en croisant les contraintes des applications et les informations du `Resource Manager`. Il faut remarquer que plusieurs communautés peuvent coexister sur un nœud et dans le réseau, permettant ainsi la création de sous-ensembles de nœuds et le déploiement d'applications selon différents critères.

D'autres services de cette couche incluent des interfaces de soumission ou de visualisation des résultats. Ces services sont notamment utiles dans l'interaction avec des dispositifs IoT qui n'ont pas la possibilité d'exécuter une instance de CloudFIT, comme par exemple les microcontrôleurs Arduino. Dans ce cas, il suffit d'offrir un accès à un nœud CloudFIT grâce à une interface REST ou JSON, comme suggère la Figure 5.4. Ces interfaces peuvent être utilisées autant pour le simple stockage de données que pour le déclenchement d'applications.

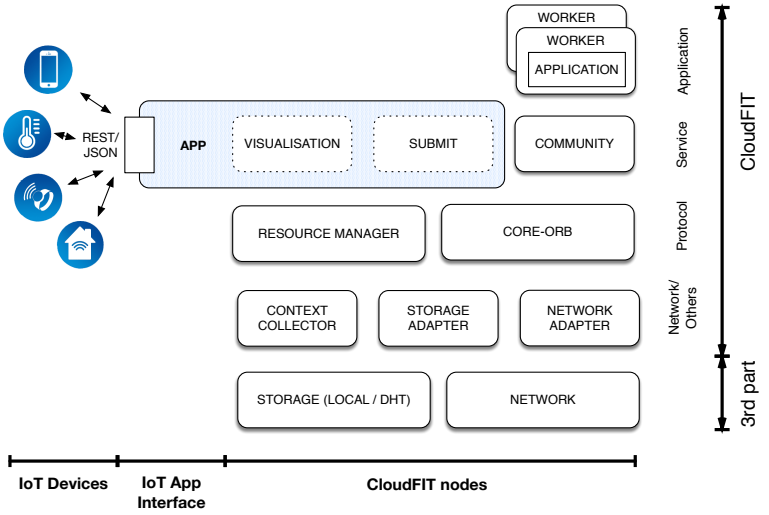

FIGURE 5.4 : Exemple d'interface IoT pour l'interaction avec CloudFIT

Finalement, la couche **Application** contient les éléments nécessaires à l'exécution de l'application fournie par l'utilisateur. Cette couche spécifie l'interface applicative qui doit être implémentée par l'application utilisateur afin d'être exécutée par CloudFIT. L'interface applicative est assez simple et intuitive, suivant les principes du paradigme FIIT. Ainsi, le développeur n'a besoin que d'écrire les méthodes suivantes :

numberOfBlocks() méthode qui retourne le nombre de tâches à lancer. Cette méthode est appelée pendant la configuration du `Task Scheduler`;

executeBlock(taskID, required[]) méthode qui démarre l'exécution proprement dite de la tâche, c'est le point d'accroche pour les `Workers`. Le *taskID* permet à la tâche de

personnaliser son exécution, et l'élément *required[taskID]* indique les éventuelles dépendances de cette tâche. Ce paramètre est aussi utilisé par le `Task Scheduler` pour gérer l'ordre d'exécution afin de respecter les dépendances ;

finalizeApplication() méthode optionnelle qui est exécutée par le `Task Scheduler` une fois que l'ensemble de tâches est terminé. Cette méthode permet l'agrégation des résultats, à l'instar d'une phase *Reduce* dans le paradigme *MapReduce*.

On y trouve aussi la classe `Task Scheduler`, un ordonnanceur associé à chaque *job* et responsable par la gestion des tâches et l'interaction avec le `Resource Manager` afin d'obtenir les `Workers` nécessaires pour l'exécution de ces tâches. La Figure 5.5 présente de manière simplifiée l'interaction entre le `Task Scheduler`, l'application et les autres éléments de la couche Service : à la réception d'une nouvelle soumission (*job*), celle-ci est confiée au `Job Scheduler` et puis au `Task Scheduler`, qui négocie l'obtention des `Workers`.

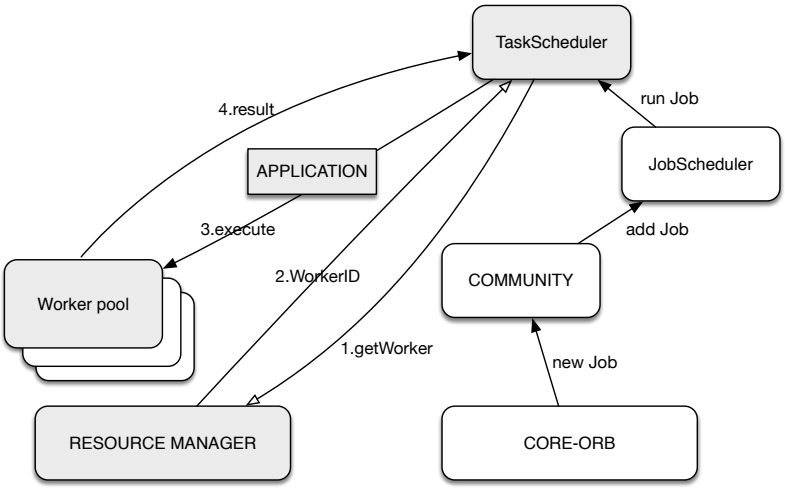

FIGURE 5.5 : Diagramme simplifié de l'interaction entre les éléments lors du lancement d'un *job* et de ses tâches

Il faut noter que la classe `Task Scheduler` est extensible et personnalisable. Par défaut CloudFIT fournit un ordonnanceur simple, cependant celui-ci peut être remplacé par des ordonnanceurs plus élaborés ou particulièrement adaptés aux besoins des applications. L'ordonnanceur par défaut fait juste une redistribution aléatoire des tâches, une technique simple qui réduit le risque de travail en double entre les nœuds. Parmi les exemples d'ordonnanceurs plus élaborés, on peut citer ceux qui prennent en charge les dépendances entre les tâches (dans le cas d'une application DAG) ou qui utilisent des éléments de contexte, comme par exemple la localisation d'un nœud, pour minimiser le temps d'accès aux données.

5.3.3 Communication

La section précédente illustrait la structuration et l'interaction des modules à l'intérieur d'une instance de CloudFIT. Cependant, une plateforme de calcul distribué se doit de garantir les échanges entre les différents nœuds sur le réseau. Dans le cas de CloudFIT, le choix d'utiliser un *overlay* P2P tiers simplifie les opérations de découverte de pairs, la gestion du réseau (entrées, sorties), routage des messages, etc. Nous pouvons ainsi nous concentrer sur la communication intrinsèque à la plateforme, comme par exemple le déploiement des applications, le suivi de la progression de l'exécution et la distribution/récupération des résultats.

Tout démarre par la soumission d'un *job*, effectué directement par un nœud déjà connecté au réseau ou grâce à une interface de soumission. Cette soumission contient le code applicatif, le nom de la `Community` cible, ainsi qu'une liste de propriétés nécessaires à la bonne exécution du *job*. Un message contenant les paramètres et propriétés de la soumission est diffusé à travers le réseau, grâce aux mécanismes de communication de l'*overlay* P2P.

Comme spécifié par l'exigence [R3], nous devons garantir qu'une application sera déployée par la plateforme elle-même, car il serait très contraignant pour l'utilisateur de devoir placer lui-même l'application sur chaque nœud. Afin de répondre à ce besoin, nous avons choisi d'utiliser le stockage DHT, un service habituellement intégré aux différents *overlays* P2P. En effet, le DHT offre à tous les nœuds de l'*overlay* un accès réseau à des objets et des fichiers, du moment où ces nœuds connaissent la clé de ces ressources. Ainsi, la soumission d'un *job* comprend l'enregistrement d'un fichier *jar* contenant le code applicatif et la clé DHT de cette ressource. Au moment du lancement du *job*, le `Job Scheduler` récupère ce fichier et extrait ses classes, qui seront chargées grâce à un *classloader*.

Le stockage DHT peut également être utilisé pour la mise à disposition de données d'entrée pour les applications *big data*, dans lesquelles un important jeu de données doit être mis à disposition des applications. Ceci n'est pas obligatoire, vu que les applications ont aussi la possibilité d'obtenir les données via des ressources extérieurs (URLs, stockage cloud, bases de données).

Lorsqu'un *job* démarre sur une machine, son statut passe de *NEW* à *STARTED*. À ce moment, le `Task Scheduler` associé à ce *job* est démarré, et les tâches peuvent être lancées. Celles-ci ont 5 états possibles : *NEW*, *STARTED*, *STARTED_DISTANT*, *COMPLETED* et *DISTANT*, comme illustré par le cycle de vie présenté en Figure 5.6.

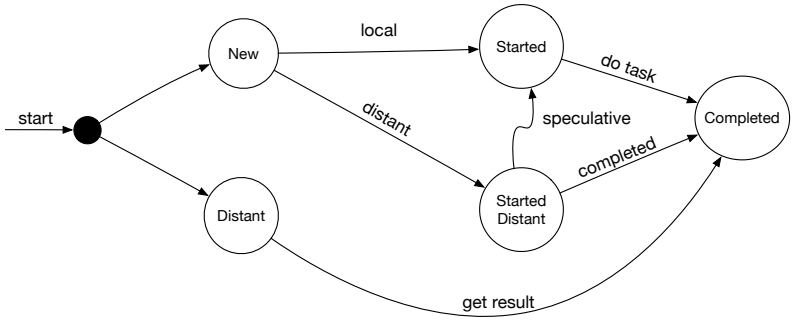

FIGURE 5.6 : Cycle de vie des tâches dans CloudFIT

Lorsqu'une tâche est lancée, un message est envoyé aux nœuds de la `Community` indiquant le ID du *job* et de la tâche. Ceci permet aux `Task Scheduler` des autres nœuds de savoir si cette tâche est traitée par une autre machine et ainsi de réduire le travail en double : ces tâches "distantes" sont marquées comme *STARTED_DISTANT* et sont placées à la fin de la file d'exécution. Une tâche marquée ainsi ne sera exécutée que lorsque toutes les tâches *NEW* auront été épuisées et, bien sûr, si aucun autre message n'est venu indiquer que la tâche aurait été complétée. En effet, le `Task Scheduler` envoie un deuxième message à la fin de l'exécution d'une tâche, indiquant le changement de son statut à *COMPLETED* et aussi indiquant le résultat de son calcul (ou bien les coordonnées pour retrouver ce résultat, qu'il soit stocké dans la DHT ou dans une ressource externe).

Si un nœud finit l'exécution de toutes ses tâches *NEW*, il peut démarrer l'exécution de tâches spéculatives parmi celles marquées *STARTED_DISTANT*. Ce mécanisme garantit la terminaison

de toutes les tâches (si le nœud original est défaillant, par exemple) et permet même d'accélérer la terminaison du calcul si le nœud original est trop lent.

En plus de mettre à jour les autres nœuds, ces échanges de messages ont aussi le rôle de mettre au courant un nouveau nœud qui rejoint le réseau. En voyant passer des messages de type "*task completed*", les nœuds peuvent demander à un voisin de les transmettre le message avec la description du *job*. Son `TaskManager` va donc procéder à la récupération des tâches *DISTANT* grâce à des requêtes spécifiques. Ce mécanisme permet ainsi de garantir l'intégration des nœuds dans un environnement volatile et d'assurer la pérennité des résultats. En effet, il suffit qu'une machine subsiste dans le réseau pour que les résultats restent accessibles.

À la fin de l'exécution de toutes les tâches, les `TaskManager` récupèrent l'ensemble des résultats locaux ou distants et le statut du *job* devient *COMPLETED*. Ce *job* reste à la disposition de toute application ou nœud qui souhaite récupérer ses résultats.

5.4 Calcul Multi-échelle et le *Fog Computing*

Comme indiqué précédemment, la plupart des travaux sur le *edge/fog computing* ont la tendance à faire une distinction entre l'utilisateur final (ou les périphériques finaux) et les dispositifs qui se trouvent à la frontière de l'Internet/*cloud*. Dans de telles approches, les appareils IoT sont des simples clients des services déployés dans un voisinage proche, ce qui est d'une certaine manière contraire aux principes du *fog computing*, où tous les dispositifs peuvent contribuer au calcul des tâches selon leurs propres capacités et les ressources disponibles.

Cependant, les réseaux P2P les plus connus organisent les nœuds indistinctement de leur emplacement réel, ce qui empêche l'établissement de services de proximité à faible latence. Pour contourner ces inconvénients, nous considérons que le réseau P2P de CloudFIT doit être enrichi par l'utilisation du concept de calcul multi-échelle [123, 124] associé à des techniques de *clustering*. En effet, le regroupement des ressources sous la forme de *clusters* est une manière efficace d'organiser les couches de calcul multi-échelle et ainsi de fournir une base de coordination pour le déploiement efficace des services.

Plusieurs approches de *clustering* sont proposées dans la littérature [83] et utilisées, par exemple, pour le routage des informations dans les réseaux de capteur sans fil. La plupart des algorithmes de *clustering* utilisent des paramètres simples comme la densité du réseau environnant ou la distance relative entre les nœuds. Malheureusement, ces métriques ne sont pas suffisamment riches pour exprimer les besoins du calcul multi-échelle dans un réseau hétérogène.

En effet, l'un de nos défis est de permettre le regroupement des nœuds selon différentes stratégies guidées par les besoins des applications. Par exemple, certaines applications ont besoin de co-localiser les données et les ressources de calcul à des endroits précis afin de rendre des services de proximité. Dans d'autres cas, des réseaux classés selon les capacités de calcul des noeuds pourront être mis à disposition des applications nécessitant des ressources plus homogènes. Ainsi, afin de s'adapter à l'hétérogénéité des environnements pervasifs et l'IoT, les métriques de clustering doivent permettre l'utilisation d'informations de contexte telles que la localisation ou la distance relative entre les nœuds, les vitesses CPU, la quantité de mémoire et de stockage des appareils, la fiabilité et même le niveau d'autorisation/confiance des nœuds collaborateurs. Bien entendu, un nœud doit pouvoir appartenir à plusieurs clusters, permettant à des données et des tâches de circuler entre les différentes couches multi-échelle, comme illustrée en Figure 5.7. Cette organisation à plusieurs niveaux en fonction du contexte des ressources

disponibles permettrait de mieux coordonner la communication entre ces ressources et d'ainsi mieux gérer la variabilité d'échelle de ces environnements.

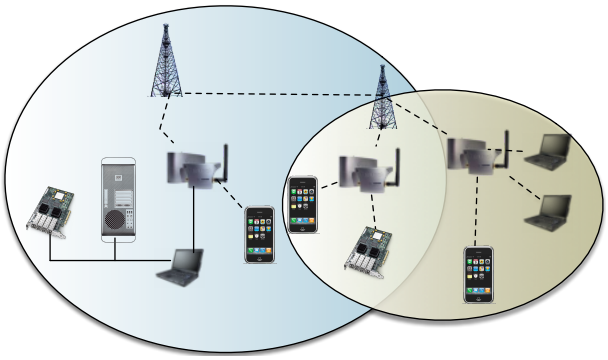

FIGURE 5.7 : Exemple de communautés interconnectées dans une plateforme multi-échelle

Dans le cas de CloudFIT, ce *clustering* pourrait être mis en œuvre grâce aux *communautés*. En effet, dans la version actuelle de CloudFIT on peut déjà lancer différentes instances de la classe `Community`, chacune dotée d'un identifiant propre. Les soumissions transmises à une communauté X ne seront transmises qu'aux nœuds ayant des instances avec le même identifiant X, et donc il serait possible de créer des sous-ensembles de nœuds adressés séparément et donc utilisés pour répartir/cloisonner les opérations. Si actuellement la création des instances des communautés se fait de manière statique lors du lancement d'un nœud, il est tout à fait envisageable de permettre la création à la volée de communautés en utilisant des messages spécifiques, transmises par le biais de la communauté par défaut qui relie tous les nœuds.

Un autre problème ouvert dans le développement de CloudFIT est celui du partage de ressources. L'implémentation actuelle ne fait pas distinction entre les communautés, avec les demandes d'accès aux `Workers` étant traitées par ordre d'arrivée. Dans le cas où multiples communautés seront créés pour répondre à des besoins spécifiques, il serait peut-être nécessaire de revoir le partage de ressources afin de mieux respecter les spécifications de chaque communauté.

5.5 Optimisations pour le *big data*

Dans le cadre du traitement de données issus des dispositifs de l'IoT, un autre facteur à prendre en compte est celui de la performance liée à l'accès et à la gestion des données. En effet, la plupart des opérations impliquent la collecte, la transformation et l'analyse des données. En effet, les plates-formes telles que Apache Hadoop se sont illustrées par leur capacité d'optimiser l'accès aux données grâce au concept de la *data locality* : les tâches sont lancées en priorité sur les nœuds qui détiennent une copie des données à traiter. Cette allocation est généralement faite de manière transparente par le gestionnaire de ressources de Hadoop (le `Resource Manager`), qui est très couplé au gestionnaire des données de HDFS (le *NameNode*). En effet, cette allocation peut expliquer le remplissage graduel des ressources observés lors de nos expériences en Chapitre 3 : si les données sont distribués entre les nœuds dans un ordre précis, l'allocation des tâches aura tendance à suivre cet ordre.

Dans le cas de CloudFIT, l'accès aux données devient une préoccupation essentielle car nous ne disposons pas d'un service de stockage spécialement intégré aux mécanismes d'ordonnancement des tâches. Si nous avons déjà eu des résultats encourageants par le passé [141] laissant envisager des performances similaires ou supérieures à celles dApache Hadoop (voir la Figure

5.8), il est claire que cela peut encore être amélioré. À ce fin, nous avons identifié deux points clés à traiter, (*i*) la surcharge due à la gestion des données sur un nœud et (*ii*) la prise en charge de la *data locality*.

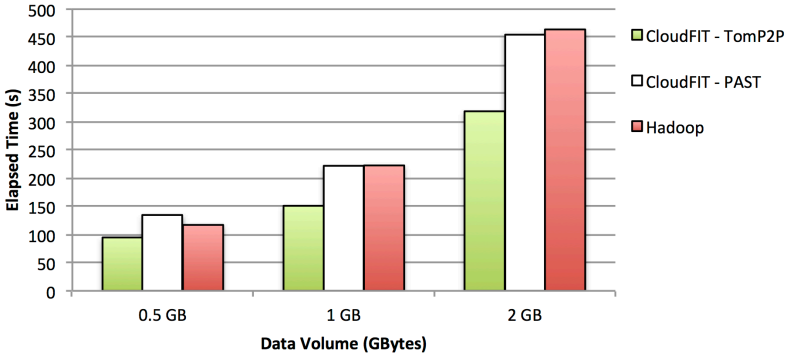

FIGURE 5.8 : Comparaison des temps d'exécution de WordCount avec CloudFIT et Hadoop

Le premier cas concerne notamment les situations où des dispositifs de faible capacité sont associés à des équipements plus puissantes lors d'un déploiement d'un environnement hétérogène (l'un de ces scénarios sera présenté plus bas, en Section 5.6.1). En effet, nous avons observé des écarts de performance très préoccupants sur ces dispositifs, dus à une combinaison de la faible vitesse d'accès aux données et de la surcharge due à la gestion des systèmes de stockage DHT. Ainsi, par exemple, un Raspberry Pi est fortement pénalisé par la vitesse et la capacité de stockage de sa carte SD, malgré une capacité de calcul suffisante (notamment en utilisant tous ses cœurs de calcul).

En tenant compte que d'autres dispositifs de faible capacité tendent à devenir populaires grâce à l'IoT, nous avons rajouté une option au démarrage de CloudFIT visant à limiter la surcharge due à la gestion du stockage DHT. Cette option, qui consiste à renseigner un espace de stockage zéro au moment du démarrage de la DHT, permet aux nœuds d'agir seulement en tant que clients distants de la DHT. Ces nœuds peuvent donc interroger le service de stockage DHT via le réseau mais ils ne sont plus obligés à gérer le stockage, réduisant la surcharge liée à la gestion de la DHT ainsi que leur besoin d'espace en disque (ce que peut être un atout quand la capacité de la carte SD n'est que de quelques gigaoctets).

Une toute autre approche est nécessaire pour répondre aux besoins du deuxième cas, la prise en charge de la localisation des données (*data locality*). Tout d'abord, ceci est un problème plus général, qui affecte la plupart des architectures de stockage P2P. En effet, les API de stockage de type DHT sont conçues de manière à répartir les données sur le réseau et les répliquer lorsque cela est possible, notamment afin d'éviter la perte de données en cas de désabonnement (*churn*). Un inconvénient de cette procédure est qu'on observe une perte d'information concernant la localisation des données [159], rendant difficile l'optimisation des transferts réseau. Nous avons donc développé deux mécanismes distincts visant à contourner cette limitation.

La première approche que nous avons développée peut facilement être mise à l'œuvre si on un accès à la bibliothèque DHT. Cette stratégie consiste à instruire l'ordonnanceur de tâches (*Task Scheduler*) à vérifier préalablement quelles tâches seraient favorisées par la présence des données dans son cache DHT local. Même si c'est une opération de bas niveau, la plupart des DHT P2P offrent la possibilité d'effectuer un *lookup* pour savoir si la ressource requise se trouve déjà dans le cache local ou s'il faut la chercher sur le réseau. En donnant la priorité aux tâches qui peuvent travailler avec des données locales, on peut espérer augmenter la performance globale de l'exécution.

Toutefois, il n'est pas toujours possible d'avoir des données en local dans un *overlay* P2P. En effet, dans une DHT les nœuds responsables par le stockage et l'indexation des ressources sont définis par la clé de hachage de la ressource. Si la fonction de hachage n'est pas spécifiquement conçue pour la *data locality*, elle aura tendance à répartir équitablement les clés, provocant l'éparpillement des données. Ceci nous amène à réfléchir sur une stratégie pour renforcer la proximité des données, grâce à un calcul personnalisé de la clé de localisation des ressources.

Cette technique a été élaborée sur la base des spécificités de la DHT de TomP2P, et donc ne peut pas être facilement généralisée. Contrairement à la plupart des systèmes de P2P qui ont seulement une clé de hachage, TomP2P identifie les ressources par quatre clés différentes $\{k_l, k_d, k_c, k_v\}$, selon la hiérarchie suivante :

— k_l - clé de localisation, utilisée pour la localisation d'une ressource dans la DHT ;
— k_d - clé de domaine, fonctionne comme une clé d'authentification, permet la séparation des données ;
— k_c - clé de contenu, permet d'identifier une ressource. Par défaut celle-ci est identique à la clé de localisation ;
— k_v - clé de version, permet la gestion de versions multiples d'une ressource.

La clé de localisation est celle qui s'approche le plus des clés DHT traditionnelles, ayant par fonction l'association d'une ressource (copie primaire ou index) au nœud avec l'ID le plus proche. Sans aucune instruction supplémentaire, la clé de localisation et la clé de contenu sont identiques, mais peuvent être différenciées, par exemple, afin résoudre les cas de collision de clés (deux ressources générant la même clé de localisation).

La clé de domaine est liée à un mécanisme d'authentification simple de TomP2P. Son but est de renforcer le cloisonnement de données des différents clients (cette authentification peut être renforcée par l'utilisation de la cryptographie afin de garantir une véritable confidentialité). Dans le cas de CloudFIT, la clé de domaine est utilisée comme un *namespace* pour la séparation des données de différents communautés ou *jobs* de calcul.

Finalement, la clé de version permet la coexistence de différentes versions d'une ressource, ce qui permet une meilleure gestion des données "mutables" avec, par exemple, l'accès à l'historique des modifications ou l'écriture en parallèle d'une ressource par plusieurs nœuds. Cette clé de version est utilisée dans les nouvelles versions de l'application *MapReduce* développée sur CloudFIT.

Ainsi, afin de renforcer la *data locality*, nous avons travaillé sur le découplage entre la clé de localisation et la clé de contenu grâce à une double fonction de hachage. Dans un premier moment, la clé de contenu est obtenue avec une méthode de hachage classique. Ensuite, la clé de localisation est calculée en faisant une association limitée aux ID des nœuds d'une communauté. La Figure 5.9 montre l'exemple de cette cartographie en calculant la clé de localisation d'une ressource r_3 par rapport à une communauté $Comm_1$.

Comme la clé de localisation se trouvera parmi les nœuds de la communauté, on augmente la probabilité de trouver la copie primaire dans les nœuds concernés. De plus, cette stratégie n'empêche pas la réplication des données sur d'autres nœuds, garantissant la persistance des données en cas de défaillance. Cette approche est aussi tolérante aux variations du nombre de membres de la communauté : en cas de disparition d'un nœud, c'est une réplique qui prend le relais ; en cas d'un nouveau membre, celui-ci sera intégré à la fonction de hachage normalement.

Tout comme la création à la volée des communautés dans CloudFIT, la stratégie présentée ici n'a pas encore été implémentée. Elle fait donc partie des développements que je compte poursuivre dans les prochains mois, afin de mieux équiper CloudFIT et poursuivre mes recherches autour du *fog computing*.

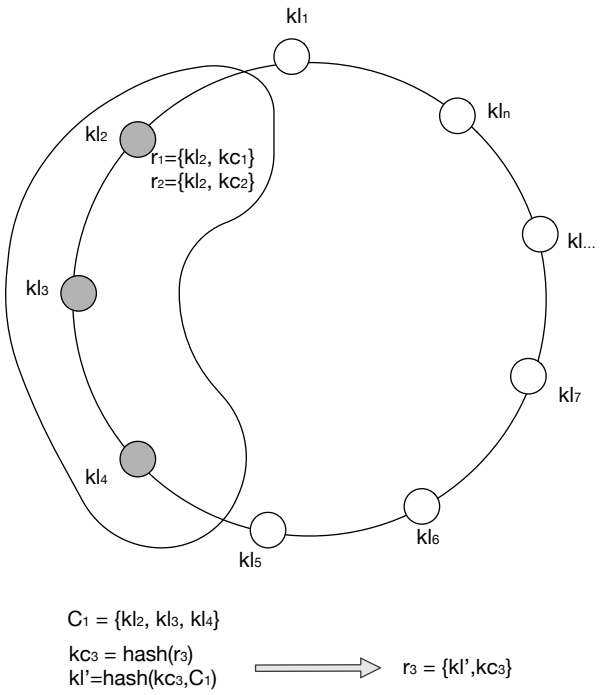

FIGURE 5.9 : Cartographie des ressources renforçant la *data-locality*

5.6 Exemples d'Utilisation de CloudFIT

En tant que plateforme expérimentale pour le calcul distribué et le *fog computing*, CloudFIT est en constante évolution. Cela n'empêche pas son utilisation comme plateforme de calcul dans certains de nos projets, notamment ceux dont l'objectif est d'utiliser des réseaux avec des éléments volatiles ou avec des ressources hétérogènes. Le premier exemple ci-dessous illustre une utilisation "recherche" pour le projet STIC-AmSud PER-MARE, dans le but d'évaluer le comportement de CloudFIT en tant que plateforme *MapReduce* pour les environnements pervasifs. Le deuxième exemple démontre une utilisation "production", où CloudFIT a été utilisé pour exécuter un workflow destiné aux sciences de l'atmosphère. Ce dernier travail a servi de base pour la proposition du projet de collaboration CAPES-Cofecub MESO.

5.6.1 L'application WordCount

Le projet STIC-AmSud PER-MARE (*Adaptive Deployment of MapReduce-based Applications over Pervasive and Desktop Grid Infrastructures*) avait pour but le développement de stratégies pour le déploiement d'applications *MapReduce* sur des environnements pervasifs, comme nous l'avons expliqué en Chapitre 3. Si l'un des volets du projet a été celui d'adapter Apache Hadoop, l'autre volet consistait à utiliser CloudFIT en tant que plateforme de calcul distribuée. Si dans l'article présenté à CLIoT 2015 [139] nous nous sommes concentrés sur la performance de CloudFIT (voir aussi la Figure 5.8), le travail présenté à CN4IoT [141] analysait l'exécution de CloudFIT par rapport à la volatilité et l'hétérogénéité des ressources.

Impact de la volatilité

Afin d'évaluer l'impact de la volatilité des nœuds dans CloudFIT, nous avons effectué le déploiement d'une application *MapReduce* simple (`WordCount`) sur un corpus de textes faisant

1 GB de données et réparti en blocs uniformes de 64 MB. Cette répartition vise à reproduire le comportement de Hadoop, qui lui aussi traite les données par blocs de 64 MB. Aussi afin de rendre la visualisation des expériences plus simple, les nœuds sont identiques et ont été explicitement limités à une seule exécution simultanée (un seul `Worker`).

Dans un premier moment et afin d'avoir un barème de comparaison, la Figure 5.10 présente le diagramme de Gantt pour une exécution sans incidents. Nous pouvons apercevoir les effets du mécanisme dordonnancement distribué utilisé par défaut sur CloudFIT (cf. 5.3.3). En effet, lorsque la liste de tâches est reçue par le Task Scheduler, celle-ci est réordonnée de manière aléatoire, ce qui s'observe grâce aux tonalités des tâches (ces tonalités ont été attribuées avant le réordonnancement, en utilisant une échelle partant du blanc pour la tâche 0 puis en la noircissant jusqu'à la tâche *n*). L'ordonnanceur choisit ainsi la première tâche disponible marquée "*NEW*" dans sa liste et avertit les autres nœuds que cette tâche est en exécution. De manière similaire, à la fin de son exécution son statut est diffusé pour annoncer la fin de la tâche. Si toutes les tâches "*NEW*" ont déjà été prises, un nœud peut lancer des tâches spéculatives parmi celles marquées "*STARTED_DISTANT*".

Plus spécifiquement, nous pouvons observer le déploiement de plusieurs tâches *map* (lesquelles ont des temps d'exécution variable selon le nombre de mots dans les documents), puis d'une grande tâche *reduce* qui s'exécute à la fin. Bien que l'exemple `WordCount` ne contienne qu'une tâche *reduce*, celle-ci se trouve exécutée par tous les nœuds car si le premier nœud a marqué la tâche comme "*STARTED*" et prévient les autres, ceux-ci n'ont rien de plus dans leur file d'exécution et lancent le *reduce* en tant que tâche spéculative. Mis à part l'utilisation des ressources de calcul, ceci n'a aucun impact sur le résultat final de l'application car les nœuds vérifient la présence d'un fichier de sortie avant de l'écrire sur la DHT, empêchant tout écrasement ou corruption.

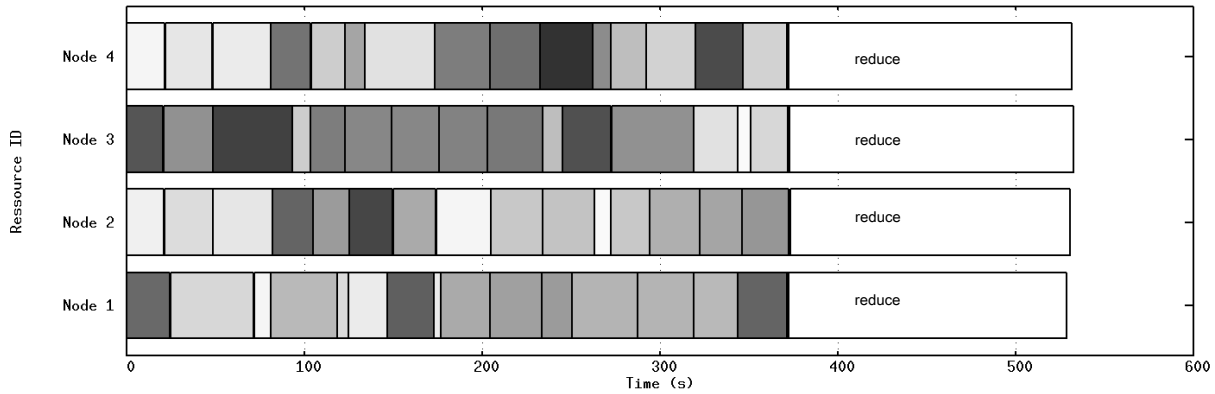

FIGURE 5.10 : Exécution de WordCount sur un *cluster* uniforme (1 GB de données en blocs de 64MB)

Grâce à ces échanges, il est aussi possible de compléter les tâches initiées par les nœuds en défaillance ou mettre au courant un nœud qui vient de rejoindre une communauté CloudFIT. La Figure 5.11 représente ainsi une situation où le nœud numéro 1 tombe en panne, avec la subséquente reprise des tâches par les autres nœuds. La Figure 5.12 va au-delà de cette situation en rajoutant un nouveau nœud (numéro 5), qui récupère l'état actuel des tâches et peut ainsi contribuer avec l'effort de calcul.

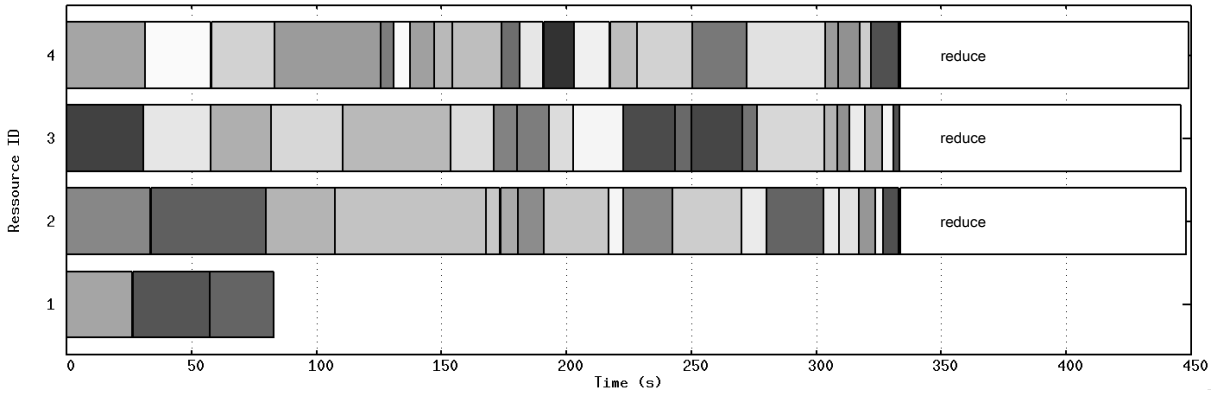

FIGURE 5.11 : Exécution de WordCount lorsqu'un nœud disparaît (1 GB de données en blocs de 64MB)

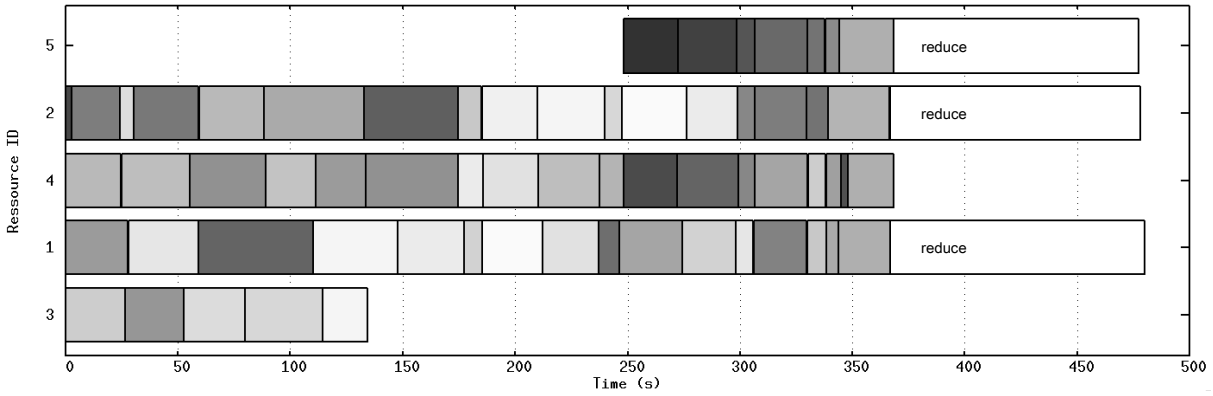

FIGURE 5.12 : Exécution de Wordcount lorsqu'un nœud rejoint la communauté après la défaillance d'un autre nœud (1 GB de données en blocs de 64MB)

Impact de l'hétérogénéité

Les expériences de la section précédente ont donné un aperçu de la distribution des tâches de CloudFIT en cas de volatilité des nœuds. Maintenant, nous souhaitons observer ce comportement dans un environnement hétérogène, où des facteurs tels que la puissance de calcul peuvent déséquilibrer la répartition des tâches entre les nœuds. Pour cela, nous avons interconnecté quatre nœuds avec des spécifications assez différentes (cf. le Tableau 5.1). Comme dans l'expérience précédente, nous limitons le nombre de cœurs (`Workers`) sur les machines pour rendre la visualisation plus simple.

Type de Nœud	Processeur	GHz	Mémoire	OS
MacBook Air	Intel Core i7-4650U	1.7	8 GB	MacOS 10.10.5
Lenovo U110	Intel Core2 Duo L7500	1.6	4 GB	Ubuntu Linux 15.4
Raspberry Pi 2	ARM Cortex-A7	0.9	1 GB	Raspbian Linux Wheezy
VM Virtualbox	Intel Core i7*	2.2*	1 GB	Debian Linux 8.2

* ces valeurs sont celles vues par la machine virtuelle

TABLE 5.1 : Spécification des nœuds du *cluster* hétérogène

Nous avons aussi modifié les paramètres de l'expérience afin d'exécuter le WordCount sur 512MB de données divisées en petits blocs de 2MB seulement ; nous pensons que cette configuration est plus proche de celle rencontrée lors de la transmission de données par les dispositifs IoT. La multiplication de tâches avec un coût individuel plus réduit rend aussi possible la participation des nœuds avec moins de puissance de calcul. La Figure 5.13 affiche le diagramme de Gantt pour une exécution de ce scénario, dans lequel nous pouvons trouver côte à côte deux ordinateurs portables, une machine virtuelle exécutant sur un ordinateur de bureau et un Raspberry Pi 2.

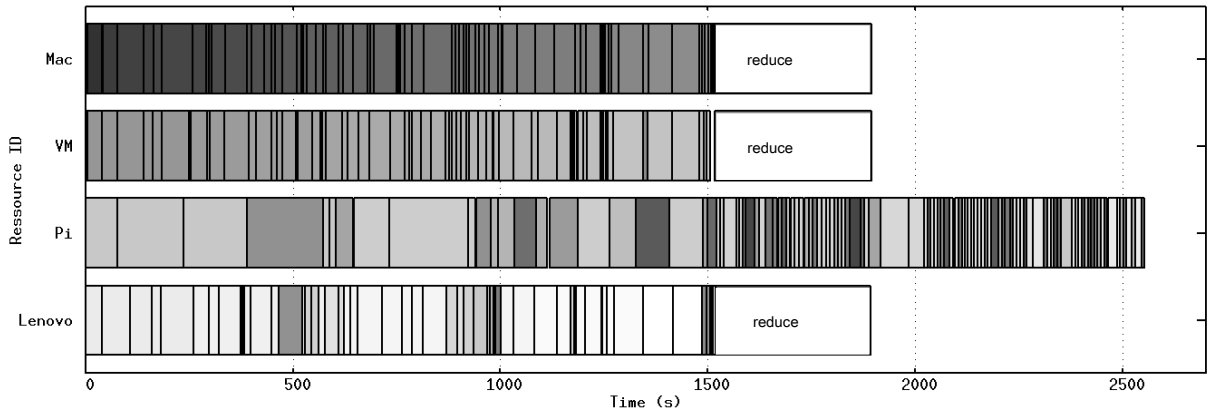

FIGURE 5.13 : Exécution de Wordcount dans un *cluster* hétérogène (512MB en blocs de 2MB)

Alors que la répartition des tâches entre les notebooks et la machine virtuelle ne présentent pas une différence significative, le Raspberry Pi, sans surprise, n'arrive pas à exécuter les tâches aussi vite que les autres nœuds (voir la longueur des tâches dans la première partie de l'exécution). De plus, ce nœud est tellement surchargé qu'il perd plusieurs messages de mise à jour et ne détecte pas la fin de la phase *map* et vers l'instant 1500s il essaye vainement d'exécuter

toutes les tâches qu'il estime incomplètes, juste pour se rendre compte que leurs résultats sont déjà dans la DHT et donc avorter leur exécution.

Au lieu de freiner notre intérêt par les dispositifs de faible puissance, ces résultats nous incitent à vouloir comprendre les raisons de ces problèmes. En effet, les dispositifs de faible puissance tels que les Raspberry Pi n'ont pas seulement des processeurs moins rapides mais ont également des limitations sur la taille et la vitesse d'accès à la mémoire et au stockage (quelques centaines de MB de RAM, des mémoires SD à la place des disques durs, etc.). Dans ces dispositifs, les tâches de gestion de l'*overlay* P2P et de la DHT (par exemple, la réplication des données) peuvent occuper une partie importante de leurs ressources et finir par interférer avec le traitement des messages échangé via l'*overlay*. Ces résultats ont motivé la mise en place d'une stratégie de collecte de contexte pour un meilleur ordonnancement et aussi de méthodes d'optimisation du stockage que nous avons détaillé dans la section précédente.

5.6.2 Détection d'Événements Secondaires de la Couche d'Ozone

La découverte du trou d'Ozone de l'Antarctique [55] a galvanisé l'intérêt de la communauté scientifique et, depuis ce moment, plusieurs études ont été menées dans le but de surveiller la variation de la densité de la couche d'Ozone sur les régions polaires [133][125]. La réduction de la couche d'Ozone peut aussi déclencher plusieurs événements sur des zones situées à des latitudes moyennes, soit à cause du mouvement de la bordure du vortex polaire sur ces régions [89][102] ou bien à cause du transit de masses d'air pauvres en Ozone détachées du vortex polaire. Ce dernier cas est appelé "événements dus à l'influence du trou d'ozone Antarctique", ou plus simplement des Événements Secondaires de l'Ozone" (*Ozone Secondary Events* - OSE).

Causés par la circulation de l'atmosphère, ces masses d'air continuent à se déplacer pendant 7 à 20 jours après leur séparation du vortex polaire et peuvent atteindre des latitudes plus élevées, occasionnant une réduction temporaire de la colonne totale d'Ozone (*Total Column Ozone* - TCO) sur des zones qui sont souvent habitées [117][155][101]. Comme résultat, des niveaux élevés de radiation ultraviolet nocive (UVB et UVC) atteignent la surface [26], à un tel point qu'une réduction de 1% de la colonne totale d'Ozone peut occasionner une augmentation de 1.2% de la radiation UV mesurée sur le sud du Brésil [68]. Ces événements secondaires de l'Ozone sont régulièrement observés sur des zones peuplés en moyenne latitude, comme par exemple en Amérique du Sud [90][115], en Afrique du Sud [129][131], à la Nouvelle Zélande [22] et aussi sur l'Île de la Réunion [147].

Malgré une forte liaison avec la dynamique de la stratosphère, le nombre d'études visant la modélisation de la circulation dynamique de la couche d'Ozone sont encore très rares [102]. En effet, la plupart des modèles climatiques se limitent aux couches inférieures de l'atmosphère, notamment celles liées à aux prévisions météorologiques, et n'explorent pas les interactions avec les couches supérieures comme celle où se trouve la couche d'Ozone. Plus récemment, un modèle obtenu par Vaz Peres [113] a permis une certaine compréhension de ces phénomènes. En se concentrant sur les données d'épisodes OSE déjà identifiés dans le passé, le modèle de Vaz Peres a permis la reproduction des événements observés. En partant de cette étude, notre but était d'utiliser des techniques du *big data* et du *data mining* afin de ramasser plus de données et possiblement d'en extraire des modèles plus précis, permettant la prévision de l'occurrence de ces événements.

L'utilisation de techniques du *big data* est essentiel car les données concernant la couche d'Ozone s'accumulent d'année en année. Par exemple, l'équipement TOMS/OMI placé dans les satellites de la NASA produit plus d'1GB de données brutes par an. Juste les observations

des satellites TOMS/OMI remontent à 1978 (ce qui fait presque 40 GB en ce moment), et à cela on peut rajouter d'autres sources de données satellites telles que les satellites de l'ESA mais aussi des observations effectuées au sol par des appareils de type Dobson ou Brewer.

Identification des événements secondaires de l'Ozone avec CloudFIT

Si dans un premier temps l'usage d'une plateforme type *cluster* ou *cloud* pourrait être envisagée, notre attention s'est portée rapidement sur CloudFIT car celui-ci a l'avantage d'exploiter les ressources de calcul disponibles, sans obliger l'installation ou la maintenance d'un parc informatique dédié. En autre, le traitement des données et la détection des OSE varie selon la zone géographique couverte et selon le type d'analyse effectuée : la détection d'événements passés utilisée pour améliorer les modèles est assez simple, alors qu'une analyse plus poussée visant l'étude des corrélations entre les événements et les courants atmosphériques peut s'avérer bien plus demandeuse de ressources. L'utilisation de CloudFIT permettrait ainsi la création d'une plateforme de calcul élastique.

Dans le cas précis de la détection des OSE, nous avons identifié quatre activités principales qui peuvent être transposées sur CloudFIT. Ces activités sont les suivantes :

1. **Prétraitement des données** - transformation des données brutes OMI au format JSON, plus adapté à nos besoins ;
2. **Filtrage et agrégation** - sélection des données concernant une zone géographique et une période donnée, puis des opérations d'agrégation si nécessaire ;
3. **Extraction des paramètres** - extraction des moyennes et écarts types pour une région et une période donnée ;
4. **Détection des événements** - identification des valeurs anormales d'Ozone, génération d'alertes.

Le prétraitement des données est nécessaire car les données brutes fournis par l'équipement TOMS/OMI sont dans un format spécifique qui requiert un *parser* propre (la NASA fournit un code FORTRAN pour cela). Dans notre cas, ces données sont transposés sur une structure JSON qui permet la recherche des entrées à partir de leurs coordonnées. Le filtrage permet de limiter la recherche sur une zone géographique et/ou sur une période d'étude, alors que l'agrégation permet l'obtention de données à une granularité différente de celle d'origine (utile notamment pour la corrélation avec d'autres types de données). En effet, la plupart des données TOMS/OMI ont une résolution d'1 dégrée, alors que d'autres sources de données ont parfois des cartographies plus détaillées (par exemple, avec 0.25 dégrée d'arc entre chaque mesure). Cette procédure sera détaillée dans la section suivante.

L'extraction des paramètres est la prochaine étape car la détection d'un OSE est liée à l'observation d'une chute anormale de la concentration de l'Ozone. Pour cela, il faut extraire des paramètres historiques tels que la moyenne et la variation (écart-type), et cela pour chaque coordonnée analysée. Cette procédure est détaillée dans les sections suivantes. Finalement, la détection se fait en comparant la mesure d'un instant précis avec la série temporelle des journées précédentes. Si les valeurs sont inférieures à la variation normale de la période, alors on peut déclencher une procédure d'alerte, signalant aux autorités sanitaires un risque lié à la radiation UV qui menacerait la population.

Les trois premières activités sont des exemples d'opérations ETL (*Extract, Transform, Load*) typiques du *big data*. La dernière activité peut être considérée comme un algorithme de prise de

décision. Nous avons donc établi un workflow qui peut être transcrit comme un enchaînement de *jobs* CloudFIT, comme illustré en Figure 5.14.

Afin de mieux répartir la charge entre les nœuds, nous avons aussi utilisé le concept de calcul multi-échelle pour répartir les quatre premiers *jobs* (*Preprocess*, *Filtering*, *Time-series* et *Detection*). Comme expliqué en section 5.4, nous pouvons organiser les nœuds en démarrant des instances différentes des communautés CloudFIT. Ainsi, une communauté C1 composée d'équipements entrée de gamme sera dédiée au prétraitement et au stockage des données des satellites et des spectre-photomètres Brewer ou Dobson installés au sol.

Par la suite, les données peuvent donc être traitées pour la détection ou bien être utilisées pour des analyses plus poussées telles que la recherche de motifs récurrents ou la prévision d'événements futurs. Les activités telles que le filtrage et l'analyse des séries temporelles demandent des ressources de calcul plus importants, fournis par la communauté C2. La Figure 5.14 inclut aussi d'autres communautés (C3 et C4) qui pourraient être déployées séparément afin d'effectuer les autres activités liées à la détection et à la prévision des OSE (ces étapes n'ont pas encore été implémentées). Il faut noter que cette organisation répond aussi aux principes du calcul multi-échelle et du *fog computing*, l'un des objectifs de CloudFIT.

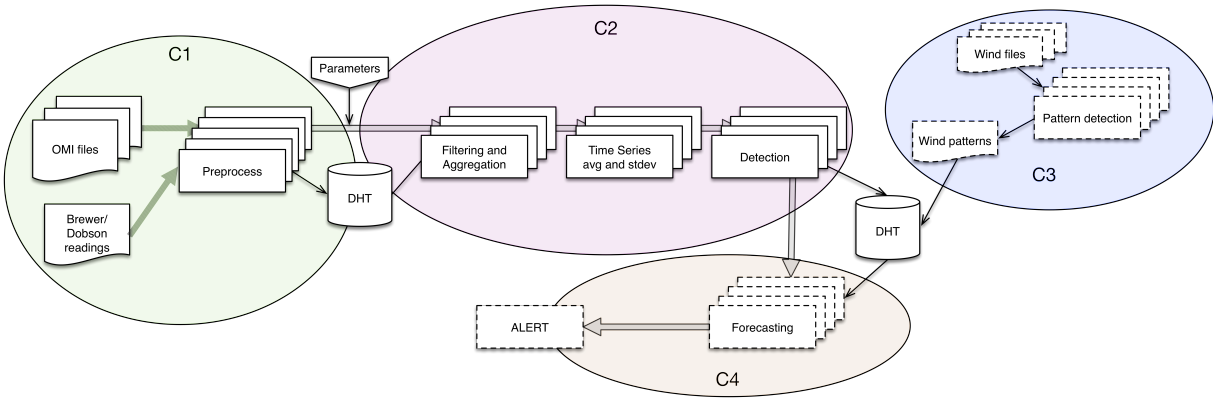

FIGURE 5.14 : Organisation des activités dans un réseau CloudFIT

Prétraitement des données

Les mesures de la colonne totale d'ozone peuvent être obtenues par des équipements au sol mais aussi par le biais des satellites, qui ont l'avantage d'offrir une couverture globale. L'un de ces équipements, l'instrument TOMS/OMI, rend publique les données consolidées de la couverture du globe, une fois par jour. Dans le cas de la détection des événements secondaires de l'Ozone, nous avons besoin des données brutes obtenues par les satellites. Ces données sont présentées selon le format illustré en Figure 5.15(a), ce qui n'est pas vraiment adapté à l'utilisation directe pour nos calculs. Chaque fichier contient un entête avec des informations sur le fichier (date, les coordonnées du grillage, le pas), suivi des mesures pour chaque latitude (indiquée à la fin de la ligne), et cela pour toutes les longitudes couvertes. Chaque mesure est exprimée en unités Dobson (UD), représentées par un entier à 3 chiffres qui doit être séparé des mesures des autres longitudes. Par exemple, les coordonnées (-89.5,-179.5) de la Figure 5.15(a) ont la valeur 280, les coordonnées (-89.5,-178.5) ont aussi la valeur 280, et ainsi de suite.

Comme ce format est difficile à comprendre et à traiter (il faut parcourir l'ensemble des entrées d'une latitude pour obtenir une mesure à une longitude donnée), nous avons décidé de

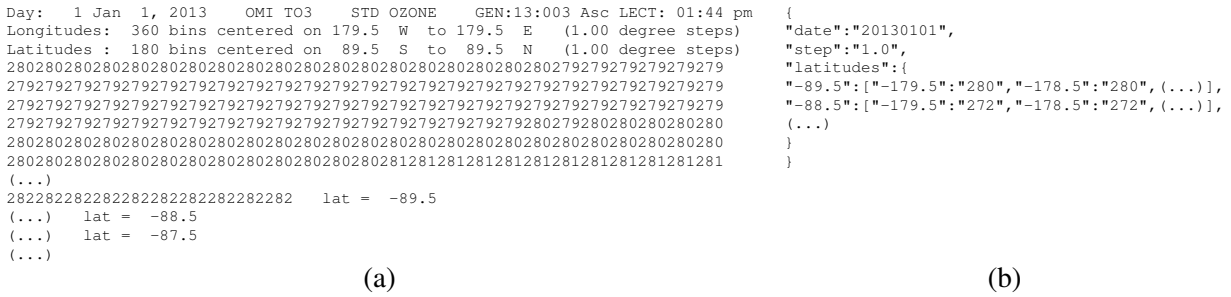

FIGURE 5.15 : Fichier brute OMI Ozone (a) et sa représentation JSON (b)

prétraiter ces fichiers et les stocker sur la DHT en tant quobjets JSON, selon le *template* présenté en Figure 5.15(b). JSON est un format structuré de données bien connu, qui peut être facilement requêté, importé sur des bases de données ou bien stocké directement dans des bases NoSQL orientées documents. Le prétraitement est facilement parallélisable car chaque journée peut être traitée indépendamment des autres. De plus, le stockage sur une DHT permet la réutilisation de données déjà traitées et l'ajout de nouvelles entrées à chaque jour.

Analyse des séries temporelles et la détection des OSE

Comme indiqué précédemment, les OSE sont caractérisés par la réduction anormale de la colonne totale d'Ozone, alors que cela n'est pas directement lié à l'expansion du vortex polaire. Le détection des OSE est faite en comparant la mesure d'une journée et les moyennes historiques pour cette région. Cependant, le choix de ce qu'on considère la "période historique" a un impact important sur la perception des événements. Pour commencer, on ne peut pas utiliser la moyenne annuelle car la concentration d'Ozone varie de façon saisonnière (dans l'hémisphère sud elle est plus basse à l'Automne et plus haute au Printemps).

L'approche utilisé par Perez *et al.* [113] considérait la moyenne historique mensuelle, i.e., la moyenne historique de chaque mois des années enregistrées. Si cela permet déjà la détection de certains OSE, cette méthode n'est pas suffisamment précise car la concentration naturelle varie d'année en année et aussi parce que la variation entre le début et la fin d'un mois est très importante dans les mois de transition comme juillet ou novembre. L'utilisation d'une moyenne standardisée aurait comme conséquence un nombre important de faux-positifs et faux-négatifs. Vu que nous disposons de ressources de calcul, nous avons décidé d'utiliser une approche par fenêtres glissantes, où par exemple chaque mesure est comparée à la moyenne des 15 jours précédents. Cette solution (l'utilisation de *time series*) est plus proche de la réalité et prend en compte la variation naturelle pour la période étudiée.

Comme pour le prétraitement des fichiers d'entrée, cette activité peut être exécuté en parallèle car chaque coordonnée (X, Y) a son propre ensemble de données. De même, le calcul de la moyenne et de l'écart type est indépendant pour chaque jour choisi, vu que la fenêtre glissante couvre des dates différentes. Ainsi, dans l'implémentation CloudFIT, les tâches de calcul sont définies en fonction du nombre de coordonnées. Chaque tâche lit les valeurs pour les 15 jours précédents et calcule la moyenne et l'écart type. Par exemple, si nous considérons la zone couverte par les coordonnées {(-70.5, -84.5), (-20.5, -29.5)} (les mêmes utilisées en Figure 5.16) avec un pas d'1 dégrée pour le grillage, nous avons 50x55 points à analyser (2750 tâches).

Une fois obtenus la moyenne et l'écart type pour chaque coordonnée et pour chaque jour cible, la détection des OSE peut être effectuée. Pour cela, nous utilisons la formule simple

présentée en Équation 5.1. Cette formule considère qu'un OSE existe si la valeur mesurée est inférieure à un seuil déterminé en fonction de la moyenne des 15 derniers jours et de la latitude (dans le cas du sud du Brésil on considère ce seuil à $1.5\times$ l'écart type). Des paramètres additionnels tels que la vorticité potentielle pourraient être rajoutés afin d'augmenter la précision des détections.

$$Detection(v) = \begin{cases} \textbf{True} & if\ v\ < (average - 1.5 \times stdev) \\ \textbf{False} & otherwise \end{cases} \tag{5.1}$$

Résultats préliminaires

Afin de valider l'implémentation, nous avons comparé les données de Peres *et al.* [113] avec les résultats obtenus à partir du workflow CloudFIT. Comme attendu, notre implémentation permet d'observer la progression du front OSE entre le 18 et le 22 octobre 2013 (Figure 5.16). On observe que le mécanisme de détection mis en place permet de se concentrer uniquement sur les zones ayant subi une variation importante de la colonne d'Ozone et pas sur celles qui habituellement ont une concentration réduite (comme par exemple le pôle ou les régions australes de l'Argentine et du Chili).

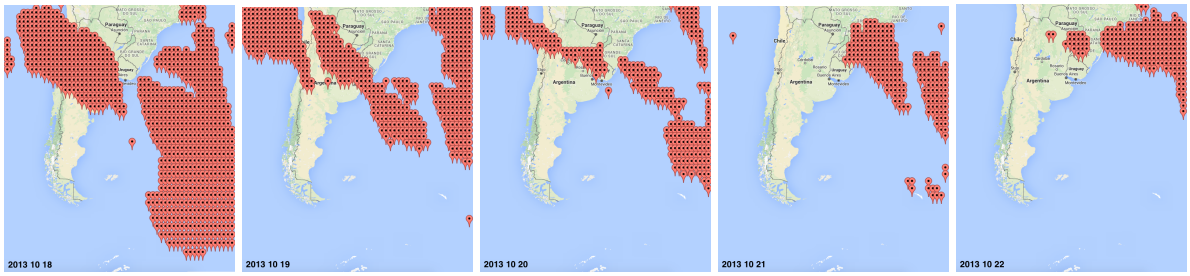

FIGURE 5.16 : Progression de l'OSE observé entre le 18 et le 22 Octobre 2013

Nous avons aussi comparé les coordonnées des OSE par rapport aux données de vorticité potentielle, l'un des facteurs étudiés par Peres. Afin de ne pas surcharger l'image, nous avons tracé seulement les points situés à proximité de l'observatoire de Santa Maria (Brésil). Comme montrent les cartes dans la Figure 5.17, on observe une forte corrélation entre la vorticité potentielle et l'approximation des masses d'air pauvres en Ozone, spécialement sur la carte du 20 octobre. On peut aussi observer que ces OSE prennent du temps à se dissiper : même si la vorticité potentielle s'est déplacée, une poche pauvre en Ozone persiste sur une zone habitée deux jours plus tard. Ces résultats encouragent la poursuite de l'étude de la corrélation entre les OSE et la vorticité potentielle.

La scalabilité de l'application a aussi été étudiée car cet algorithme peut être utilisé autant pour des détections journalières que pour des analyses plus poussées sur des périodes ou zones plus importantes. Ainsi, par exemple, l'analyse de la période entre le 15 et le 31 octobre 2013 a été fait dans une *cluster* pervasif composé de seulement deux machines (un Macbook Air - Intel i7-4650U, 2 cœurs, 8GB RAM - et un Dell Precision T5610 - 2x Intel Xeon E5-2620, 12 cœurs, 32GB RAM).

En utilisant l'ensemble des cœurs de calcul de chaque machine, l'analyse a duré 570 secondes. En comparaison, l'évaluation d'une seule journée avec un Raspberry Pi 2 a nécessité 40 minutes. Ceci peut être optimisé en augmentant le nombre de coordonnées traitées par chaque tâche (réduisant le surcout du démarrage d'une nouvelle tâche) mais aussi on en rajoutant d'autres nœuds au *cluster* pervasif.

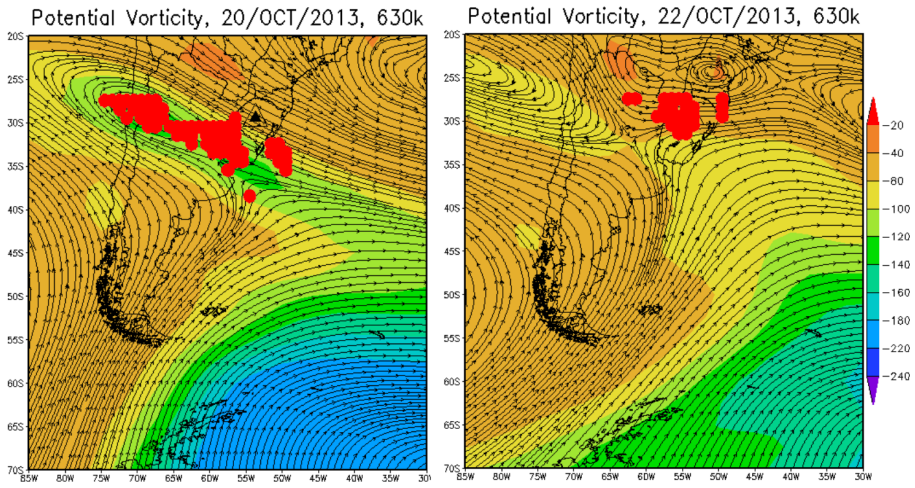

FIGURE 5.17 : Superposition des cartes de la vorticité potentielle et le OSE identifié sur l'observatoire de Santa Maria, Brésil (29.68 S, 53.81 W)

5.7 Travaux en Cours et Perspectives

Le développement de CloudFIT occupe une place important dans ma recherche du fait de pouvoir l'enrichir avec des nouvelles techniques issues de mes travaux de recherche. Ainsi, au delà de la consolidation du réseau multi-échelle et de la mise en place de la *data-locality* discutés en sections 5.4 et 5.5, je compte développer CloudFIT dans le cadre des collaborations suivantes :

Plateforme pour la détection et la prédiction des OSE

Comme indiqué en Section 5.6.2, CloudFIT a été utilisé lors de la préparation du du projet international CAPES-Cofecub MESO. Les travaux autour de la détection et la prédiction des OSE doivent se poursuivre dans les années à venir, et dans un premier moment je vais accueillir deux chercheurs brésiliens à Reims, à partir de l'hiver 2017 (une doctorante en co-tutelle et un post-doctorant). Ces chercheurs, spécialisés dans le domaine de la météorologie, auront notamment pour mission la création des modèles atmosphériques spécifiques pour les conditions de la stratosphère, utilisant par exemple le modèle WRF [132].

Un autre objectif de ces missions est celui de développer un environnement de calcul à faible coût pour l'exécution de ces modèles atmosphériques. L'approche privilégiée est celle des *clusters* de nano-ordinateurs, pour des raisons telles que le faible coût d'achat et de maintenance. Pour cela, on envisage d'associer des machines virtuelles de type conteneur avec CloudFIT, ce dernier jouant le rôle de gestionnaire de tâches et des ressources.

De même, nous étudions les possibilités d'utilisation et de déploiement de capteurs UV de la gamme ML8511 ou similaire pour l'établissement d'un réseau de surveillance "bas coût" des variations de la radiation UV. L'intégration des données issues de ces dispositifs IoT à la plateforme de calcul permettra une meilleure couverture des événements liés à la couche d'Ozone Antarctique, notamment dans les zones plus proches des tropiques qui ne sont pas totalement couvertes par l'orbite des satellites.

Ordonnancement sensible au contexte, déploiement et migration de micro-services dans un environnement *fog computing*

Le *fog computing* est un concept qui englobe plusieurs définitions à la fois. L'un des principaux défis est celui de garder toute la flexibilité et la disponibilité du *cloud* au même temps où l'on cherche à placer les services au plus près des utilisateurs, en utilisant l'ensemble de ressources qui l'entourent. De ce fait, il est difficile de concevoir un système répondant à tous ces critères, mais on peut essayer de répondre à quelques uns.

Les sections précédentes ont montré certaines fonctionnalités de CloudFIT qui peuvent être mis au service de la recherche sur les réseaux hétérogènes dont le *fog*. Plus qu'une démarche d'ingénieur, l'utilisation de CloudFIT donne des outils pour la recherche, qui autrement ne serait que conceptuelle (comme c'est encore le cas de plusieurs travaux sur le *fog*) ou dépendante de technologies tiers.

Ainsi, je souhaite continuer à faire évoluer la plateforme dans le but de fournir les outils pour cette recherche. Parmi les priorités, il y a bien sûr la poursuite des travaux sur l'ordonnancement sensible au contexte. Les outils de base sont déjà en place (collecteur de contexte, ordonnanceur simple) mais les cas étudiés jusqu'à présent n'ont pas encore été suffisamment poussés pour permettre des grandes avancées dans ce domaine.

De même, l'organisation multi-échelle des ressources me semble essentielle pour le passage à l'échelle nécessaires à ces environnements. Si les promesses du *fog computing* se réalisent, celui-ci sera amené à intégrer des ressources allant de l'IoT au *cloud*, ce qui serait impossible à gérer sans une organisation modulaire des ressources.

Finalement, je compte investir dans le support aux machines virtuelles (et les micro-services) car, contrairement aux applications actuellement supportées, celles-ci permettent une migration plus facile des tâches. Sans le support à la migration, les stratégies d'ordonnancement sensibles au contexte développées sur CloudFIT resteraient incapables d'attaquer des problèmes essentiels comme l'équilibrage de charge, la minimisation des temps de réponse, etc.

Conclusion et Perspectives

Tout au long de ce document j'ai essayé de mettre en évidence les différentes façons dont l'hétérogénéité peut affecter l'opération d'un système ou d'une application, ainsi que mes contributions pour leur prise en charge.

L'hétérogénéité peut se présenter sous différentes formes, demandant des actions spécifiques selon l'application, l'environnement et le contexte d'utilisation. Une première catégorie représente les *variations matérielles* des composants d'un système informatique. À l'hétérogénéité matérielle s'ajoutent les problèmes de l'*hétérogénéité des communications*, dont la prise en charge se fait notamment par l'optimisation des échanges des messages (Chapitre 1), et *l'hétérogénéité des tâches*, résultat d'une distribution déséquilibrée de charge entre les différents ressources participant à un calcul (Chapitre 2).

Il faut également être capable de supporter les variations des ressources tout au long de l'exécution, vu que *l'hétérogénéité issue de la dynamique* (Chapitre 3) d'un système peut prendre différentes formes telles que le départ ou l'arrivée de ressources ou bien par leur changement d'état ou de capacité. Finalement, *l'hétérogénéité des données* (Chapitre 4) peut aussi impacter le développement d'une application à cause de leur variété et des spécificités d'accès selon les sources : des objets en mémoire, des fichiers, des URIs ou des requêtes distantes (RPC, Web services, etc.).

Les travaux que j'ai conduit au fil des années touchent une ou plusieurs de ces manifestations de l'hétérogénéité, et dans ce document j'ai voulu montrer des cas représentatifs de ces travaux. Ainsi, au cours de la première partie de ce mémoire, j'ai présenté des algorithmes issus de mes travaux dont l'objectif était la compréhension des facteurs qui impactent les opérations de communication collective dans les *grids*. J'ai pu exploiter ainsi des méthodes de mesure de performance et de découverte de la topologie réseau, et cela dans le but d'optimiser ces opérations mais aussi afin de pouvoir estimer leur performance avec une haute précision. Avec les années, mes travaux de recherche dans ce domaine se sont déportés vers l'analyse de performance des applications et systèmes, mais ayant toujours comme objectif l'optimisation des performances.

Dans la deuxième partie, j'ai utilisé comme exemple les efforts faits pour gérer l'hétérogénéité des tâches lors de la parallélisation et la gestion de l'exécution distribuée d'une application en biochimie. À partir d'une exécution monolithique, j'ai participé au développement de stratégies visant à découper l'espace de données et d'ainsi permettre la création de tâches de calcul pouvant être exécutées en parallèle. Ceci a été accompagné par le développement d'une plateforme de déploiement capable de gérer l'exécution des tâches distribuées sur un *cluster* HPC ou sur un ensemble de nœuds volontaires. Bien que relativement simple d'un point de vue technique, les activités développées pendant le co-encadrement de cette thèse m'ont apporté du recul vis-à-vis des processus nécessaires à la parallélisation et au déploiement d'une application tiers.

La troisième partie de ce mémoire se consacre à la dynamique des ressources et aux stratégies pour sa prise en charge. Ceci est illustré par une expérience visant à améliorer le comportement de la plateforme *big data* Apache Hadoop dans les environnements hétérogènes et dynamiques (qu'on nomme ici environnements pervasifs). En s'aidant d'un mécanisme de collecte d'informations sur le contexte des ressources, il a été possible de modifier ce *framework* et ainsi d'adapter la gestion des tâches aux ressources disponibles à chaque instant. Les concepts développés ici ont fortement influencé l'ensemble de mes activités de recherche, qui depuis se sont orientées vers le support à la dynamique des ressources dans les milieux hétérogènes. Ce n'est pas par hasard si mes recherches en ce moment se concentrent sur le *fog computing*.

La quatrième partie présente la spécification d'une base documentaire permettant l'accès transparent aux sources de données, peu importe leur nature (fichiers, flux, bases de données, etc.). Pour cela, j'ai présenté une spécification pour la construction d'un réseau hiérarchique pouvant héberger cette base documentaire. Les contributions de cette partie sont peut-être moins présentes dans ma recherche du fait d'avoir une préférence sur les réseaux P2P au détriment des réseaux hiérarchiques, toutefois la problématique liée à la diversité des sources de données reste un défi majeur que je fais ressortir notamment dans mes activités d'enseignement.

Finalement, dans la dernière partie de cette habilitation, j'ai présenté en détails la plateforme expérimentale de calcul distribué CloudFIT. Dans un premier moment j'ai montré l'architecture et les mécanismes de communication et de gestion des noeuds, spécifications que jusqu'à présent n'avaient pas été publiées. Par la suite, j'ai introduit les stratégies pour le support à l'ordonnancement adapté au contexte qui font déjà partie de CloudFIT ou qui seront implémentées dans un avenir proche. Cette partie se termine par la présentation de deux exemples d'utilisation de CloudFIT, tous les deux issus de projets internationaux dont j'ai participé. Plus qu'un récapitulatif, ce chapitre a présenté l'évolution de CloudFIT au fil des années, et son rôle en tant que plateforme expérimentale pour mes recherches dans les domaines de l'IoT et du *fog computing* et l'Internet des Objets (*Internet of Things* - IoT). .

L'ensemble de mes travaux a été à l'origine d'un nombre significatif de publications internationales (11 revues, 37 conférences, 3 posters, 3 chapitres de livre) et nationales (2 revues, 15 conférences, 3 posters). J'ai aussi participé au co-encadrements de deux thèses de doctorat et d'un post-doctorat : la thèse de Romain Vasseur (thèse CIFRE, dirigée par Manuel Dauchez et aussi co-encadrée par Stéphanie Baud), celle de Thierno Ahmadou Diallo (thèse en co-tutelle avec l'Université Cheikh Anta Diop - Sénégal, dirigée par Olivier Flauzac et Samba Ndiaye), et le stage post-doctoral de Iyad Alshabani (dans le cadre du projet ANR USS-SIMGRID).

J'ai également été en charge de l'élaboration de trois projets de collaboration internationale entre l'Université de Reims et l'Amérique du Sud : le projet STIC-AmSud PER-MARE (2013-2014, avec l'Universidad de la República - Uruguay et l'Universidade Federal de Santa Maria - Brésil), le projet STIC-AmSud CC-SEM (2017-2018, avec L'Universidad de Buenos Aires - Argentine et l'Universidad de la República - Uruguay), puis le projet CAPES-Cofecub MESO (2017-2020, avec l'Université de la Réunion et l'Universidade Federal de Santa Maria - Brésil). Dans tous ces projets j'ai exercé le rôle de coordinateur pour l'équipe de Reims et, dans le cas du projet PER-MARE, j'ai été aussi le coordinateur international.

Les perspectives associées à mes travaux sont nombreuses, d'une part à cause des projets en cours et d'autre part par les opportunités d'innovation dans les domaines de mes activités. Néanmoins, je souhaite à l'avenir privilégier trois thèmes.

Le *fog computing* et les environnements pervasifs

Les environnements pervasifs et le *fog computing* constituent pour moi des sujets de recherche prioritaires. Tout d'abord, ils représentent l'opportunité de poursuivre les sujets de recherche auxquels je me suis dédié ces dernières années : la prise en compte de l'hétérogénéité des ressources, de l'hétérogénéité des tâches, de la volatilité, de l'ordonnancement sensible au contexte et du calcul multi-échelle.

Le grand attractif des environnements pervasifs et *fog computing* reste toutefois la diversité de problèmes ouverts. Le manque de standardisation n'a pas encore permis un consensus entre les chercheurs, résultant en multiples visions qui ne sont pas toujours compatibles. Même si une spécification voit le jour demain, elle sera forcément réductrice et donnera naissance à de nouveaux verrous scientifiques qu'il faudra explorer : une spécification trop orientée "services dans le *edge*" laissera forcément du vide en ce qui concerne la migration de tâches et de micro-services ; des solutions réseaux contrôlées par les opérateurs auront par conséquence la multiplication des recherches autour des réseaux éphémères contrôlés par les applications.

À mon avis le principal défi à répondre est celui de la perméabilité des frontières entre l'Internet des objets, le *fog computing* et le *cloud*. Une différentiation trop marquée serait rapidement dépassée du fait de la multiplication des dispositifs IoT et de leur montée en performance. Une perméabilité trop importante induirait plus de défis liés à l'hétérogénéité que nécessaire. En fondant mes recherches dans ce thème, j'espère pouvoir apporter ma contribution.

Évidemment, les objectifs à court terme incluent ceux décrits dans les chapitres précédents : continuer le développement de la plateforme CloudFIT, poursuivre la recherche sur le support à la virtualisation et aux micro-services dans les nano-ordinateurs, etc. L'arrivée d'une doctorante en cotutelle et d'un post-doctorant à la fin 2017, dans le cadre du projet CAPES-Cofecub MESO, sera aussi l'occasion de tester l'exécution distribuée de modèles atmosphériques spécifiques pour la couche d'Ozone. Ces projets forment donc le socle de base pour mes recherches à plus long terme sur le *fog computing* et les environnements pervasifs.

L'Internet of Things et les *smart cities*

L'IoT est sans aucun doute un sujet prioritaire de ma recherche, autant parce que l'IoT représente un domaine où l'hétérogénéité se présente sous toutes ses différentes facettes, mais aussi parce que l'IoT est une cible prioritaire pour la compréhension du *fog computing*. En effet, dans ma vision, le progrès de l'électronique embarquée fera qu'au moins une partie des dispositifs IoT évoluera jusqu'à devenir des véritables nœuds de calcul. À partir de ce moment, ces nœuds seront capables d'effectuer une partie des tâches qui aujourd'hui sont transmises au *cloud* ou aux dispositifs à la périphérie des réseaux. Apprendre à contrôler et à orchestrer le calcul sur ces dispositifs marqués par une forte hétérogénéité serait une étape de plus dans la consolidation d'un véritable *fog computing*.

Les recherches en IoT peuvent aussi s'associer à celles sur les *smart cities*. Cette association nous ouvre un volet applicatif très intéressant, non seulement par la possibilité de mettre en pratique des algorithmes mais aussi à cause de leur impact sur notre vie quotidienne. Ce n'est pas par hasard que plusieurs projets que j'intègre sont orientés dans ce sens. Dans le cas du projet STIC-AmSud CC-SEM, l'objectif principal est le développement dune plateforme intégrée pour la surveillance et le contrôle de la consommation électrique dans les milieux urbains. Ceci implique donc le développement des dispositifs électroniques pour la mesure de la consommation mais aussi la mise en place du réseau de collecte et d'analyse des données.

On retrouve aussi la thématique des *smart cities* sous le nom *smart agriculture*, l'un des axes prioritaires de l'Université de Reims Champagne-Ardenne. Dans ce cas, je suis engagé dans un projet avec les chercheurs de l'Unité de Recherche Vignes et Vin de Champagne afin de mettre en place un réseau de capteurs déployé au pied des vignes. Ce réseau IoT permettra la surveillance de paramètres tels que l'humidité du sol, la luminosité et la température autour des plantes, données qui seront ensuite analysées, notamment afin de corréler ces données aux apparitions de certaines maladies.

Une autre opportunité de recherche à court et moyen terme viendra avec le projet ANR autour de la *Green IT* qui sera soumis cette année. Dans ce projet, nous nous intéressons au développement de stratégies d'ordonnancement basés sur le contexte et sur des objectifs de consommation préétablis afin d'aider à la réduction de l'empreinte énergétique des applications. Ces stratégies peuvent donc être appliquées au développement de services et d'applications mobiles, en utilisant par exemple la migration de composants et d'applications à des fins d'économie d'énergie.

Les recherches sur l'*Internet of Things* et sur les *smart cities* représentent donc des sujets de recherche à forte valorisation à le court terme, avec l'avantage d'être toujours alignés avec mes objectifs principaux à long terme.

Le calcul distribué et le big data

Le dernier thème prioritaire de ma recherche concerne le calcul distribué et le *big data*. Pour être plus précis, je considère que ces deux sujets sont de moins en moins dissociés, les avancées dans le domaine du *big data* étant calquées essentiellement sur des plateformes et techniques issues du calcul distribué. De même, la recherche sur le calcul distribué ne peut plus ignorer les couts liés à la gestion et à la transmission des grandes masses de données, un paramètre qui tend à être encore plus décisif dans le cas des environnements hétérogènes tels que le *fog computing*.

Les outils pour le calcul distribué (dont le *big data*) sont toutefois encore trop dépendants d'infrastructures stables et homogènes. Comme le prouve notre travail sur le framework Hadoop, il est nécessaire de repenser ou, tout au moins, d'adapter les plateformes et frameworks de manière à mieux gérer l'hétérogénéité et la dynamicité des ressources. En effet, je surveille de près d'autres plateformes telles que Apache Storm, vu que ces plateformes peuvent dans certains cas répondre plus efficacement à certaines exigences d'un déploiement *fog computing* ou, au contraire, bénéficier des techniques développées pour CloudFIT.

Cette thématique présente aussi un volet pratique qui peut venir à supporter mes recherches futures. En effet, la 2ème phase du projet STIC-AmSud CC-SEM, dédié à la consommation électrique, doit s'élargir afin d'inclure des éléments de traitement *big data* et d'apprentissage automatique (*deep learning*). Le projet CAPES-Cofecub MESO, de son côté, dépend de l'analyse de données afin de comprendre les phénomènes atmosphériques et créer des modèles de prédiction fiables. Afin de remplir ces tâches, je compte autant sur l'expertise des membres de mon équipe sur le *deep learning* que sur l'expérience obtenue pendant mes interventions dans le module "Outils Big Data", lequel j'enseigne aux étudiants en Master 2 Informatique et en Master 2 Statistique pour l'Évaluation et Prospective.

En perfectionnant mes connaissances sur les outils et les algorithmes de traitement des données, j'espère bien élargir les possibilités de mes recherches futures, tout en contribuant au développement des thématiques telles que le *fog computing* qui est au centre de mes intérêts.

Publications Personnelles

Articles dans des revues avec comité de lecture

Internationales

[1] **Steffenel, L.A.**, Kirsch-Pinheiro, M., Vaz Peres, L., Kirsch Pinheiro, D., Strategies to implement Edge Computing in a P2P Pervasive Grid, *International Journal of Information Technologies and Systems Approach (IJITSA)*, IGI Global, 11(1), pp 1-15, 2018.

[2] Cassales, G. W., Charao, A., Kirsch-Pinheiro, M., Souveyet, C., **Steffenel, L.A.**, Kirsch-Pinheiro, M., Vaz Peres, L., Kirsch Pinheiro, D., Improving the Performance of Apache Hadoop on Pervasive Environments through Context-Aware Scheduling, *Journal of Ambient Intelligence and Humanized Computing*, Springer, 7(3), pp. 333-345, 2016. doi :10.1007/s12652-016-0361-8.

[3] Engel, T.A., Charao, A., Kirsch-Pinheiro, M., **Steffenel, L.A.** Performance Improvement of Data Mining in Weka through Multi-core and GPU Acceleration : opportunities and pitfalls, *Journal of Ambient Intelligence and Humanized Computing*, Springer, June 2015.

[4] Diallo, T. A., Flauzac, O., **Steffenel, L.A.**, Ndiaye, S., and Dieng, Y. GRAPP&S, a Peer-to-Peer Middleware for Interlinking and Sharing Educational Resources. *Transactions on Industrial Networks and Intelligent Systems*, EAI, 2(3) :e1, May 2015.

[5] Vasseur, R., Baud, S., **Steffenel, L.A.**, Vigouroux, X., Martiny, L., Krajecki, M., Dauchez, M. Inverse Docking Method for New Proteins Targets Identification : A Parallel Approach. *Journal of Parallel Computing - special issue on Parallelism in Bionformatics*, Elsevier, Vol 42,, pp 48-59. February 2015.

[6] **Steffenel, L.A.**, Flauzac, O., Charao, A. S., P. Barcelos, P., Stein, B., Cassales, G., Nesmachnow, S., Rey, J., Cogorno, M., Kirsch-Pinheiro, M. and Souveyet, C., Mapreduce challenges on pervasive grids, *Journal of Computer Science*, vol. 10 n. 11, pp. 2194-2210, July 2014.

[7] Vasseur, R., Baud, S., **Steffenel, L.A.**, Vigouroux, X., Martiny, L., Krajecki, M., Dauchez, M. AMIDE Automatic Molecular Inverse Docking Engine for Large-Scale Protein Targets Identification, *International Journal On Advances in Life Sciences*, 6 :(3&4), IARIA, 2014.

8] Flauzac, O., Krajecki, M., **Steffenel, L.A.** CONFIIT : a middleware for peer-to-peer computing. *Journal of Supercomputing*, Springer, vol 53 n. 1, July 2010, pp. 86-102.

[9] Nasri, W., **Steffenel, L.A.**, Trystram, D. Adaptive Approaches for Efficient Parallel Algo-

rithms on Cluster-based Systems. *International Journal in Grid and Utility Computing*, Inderscience, vol 1 n. 2, 2009, pp 99-108.

[10] **Steffenel, L.A.**, Martinasso, M., Trystram, D. Assessing Contention Effects of All-to-All Communications on Clusters and Grids. *International Journal of Pervasive Computing and Communications - Special Issue on Towards merging Grid and Pervasive Computing*, Vol. 4 n. 4, 2008, pp. 440-459.

[11] **Steffenel, L.A.**, Mounié, G. A Framework for Adaptive Collective Communications for Heterogeneous Hierarchical Computing Systems. *Elsevier Journal of Computer and Systems Sciences - Special Issue on Performance Analysis and Evaluation of Parallel, Cluster, and Grid Computing Systems*, vol 74 n. 6, 2008, pp. 1082-1093.

Nationales

[12] Vaz Peres, L., Kirsch Pinheiro, D., **Steffenel, L.A.**, Mendes, D., Valentin Bageston, J., Dornelles Bittencourt, G., Passáglia Schuch, A., Anabor, V., Paes Leme, N.M., Schuch, N.J. Monitoramento de Longo Prazo e Climatologia de Campos Estratosféricos quando da Ocorrência dos Eventos de Influência do Buraco de Ozônio Antártico sobre o Sul do Brasil, *Revista Brasileira de Meteorologia*, SBMET/Thomson-Reuters, 2017.

[13] Flauzac, O., **Steffenel, L.A.**, Diallo, T.H., Niang, I., Ndiaye, S. GRAPP&S Data Grid : Une approche de type grille et système pair-à-pair pour le stockage de données. *Revue URED (Université-Recherche-Développement)*, Presses Universitaires de l'Université Gaston Berger de Saint-Louis du Sénégal, 2012.

Communications avec actes et comité de sélection

Internationales

[14] Beserra, D., Kirsch-Pinheiro, M., **Steffenel, L.A.**, Moreno, E.D., Comparing the Performance of OS-level Virtualization Tools in SoC-based Systems : The Case of I/O-bound Applications, *The 22nd IEEE Symposium on Computers and Communications (ISCC 2017)*, Heraklion, Greece, July 3-6, 2017.

[15] Charao, A., Hoffmann, G., **Steffenel, L.A.**, Kirsch-Pinheiro, M., Stein, B., Performance Evaluation of Cloud-based RDBMS through a Cloud Scripting Language, *19th International Conference on Enterprise Information Systems (ICEIS)*, Porto, Portugal, April 26-29, 2017.

[16] Beserra, D., Kirsch-Pinheiro, M., **Steffenel, L.A.**, Souveyet, C., Moreno, E.D., Performance Evaluation of OS-level Virtualization Solutions for HPC Purposes on SoC-based Systems, *IEEE International Conference on Advanced Information Networking and Applications (AINA)*, Taipei, Taiwan, March 27-29, 2017.

[17] **Steffenel, L.A.**, Kirsch-Pinheiro, M., Kirsch-Pinheiro, D., Vaz Peres, L., Using a Pervasive Computing Environment to Identify Secondary Effects of the Antarctic Ozone Hole, *2nd*

Workshop on Big Data and Data Mining Challenges on IoT and Pervasive (Big2DM), Madrid, Spain, May 23 - 26, 2016. Procedia Computer Science, v 83, pp. 1007-1012, Elsevier.

[18] **Steffenel, L.A.**, Kirsch-Pinheiro, M., When the Cloud goes Pervasive : approaches for IoT PaaS on a mobiquitous world, *EAI International Conference on Cloud, Networking for IoT systems (CN4IoT 2015)*, Rome, Italy, October 126-27, 2015.

[19] **Steffenel, L.A.**, Kirsch-Pinheiro, M., CloudFIT, a PaaS platform for IoT applications over Pervasive Networks, *3rd Workshop on CLoud for IoT (CLIoT 2015)*, Taormina, Italy, September 15, 2015. Springer CCIS 567, pp 20-32.

[20] **Steffenel, L.A.**, Kirsch-Pinheiro, M., Leveraging Data Intensive Applications on a Pervasive Computing Platform : the case of MapReduce, *1st Workshop on Big Data and Data Mining Challenges on IoT and Pervasive (Big2DM)*, London, UK, June 2 - 5, 2015. Procedia Computer Science, vol. 52, Jun 2015, Elsevier, pp. 10341039.

[21] Cassales, G.W., Charao, A.,Kirsch-Pinheiro, M., Souveyet, C., **Steffenel, L.A.**, Context-Aware Scheduling for Apache Hadoop over Pervasive Environments, *The 6th International Conference on Ambient Systems, Networks and Technologies (ANT 2015)*, London, UK, June 2 - 5, 2015. Procedia Computer Science, vol. 52, Jun 2015, Elsevier, pp. 202209.

[22] Rey, J., Cogorno, M., Nesmachnow, S. and **Steffenel, L.A.** Efficient Prototyping of Fault-Tolerant Map-Reduce Applications with Docker-Hadoop. *WoC : First International Workshop on Container Technologies and Container Clouds*, Tempe, AZ, USA, March 9-13, 2015.

[23] Cassales, G.W., Charao, A.,Kirsch-Pinheiro, M., Souveyet, C., **Steffenel, L.A.**, Bringing Context to Apache Hadoop, *8th International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2014)*, Rome, Italy, August 24 - 28, 2014. ISBN : 978-1-61208-353-7, IARIA, pp. 252-258

[24] Engel, T.A., Charao, A., Kirsch-Pinheiro, M., **Steffenel, L.A.** Performance Improvement of Data Mining in Weka through GPU Acceleration, *5th International Conference on Ambient Systems, Networks and Technologies (ANT 2014)*, Hasselt, Belgium, June 2 - 5, 2014. Procedia Computer Science, vol. 32, 2014, Elsevier, pp. 93100.

[25] Rey, J., Cogorno, M., Nesmachnow, S. and **Steffenel, L.A.** Fast Prototyping of Map-Reduce Applications with Docker-Hadoop. *Conférence dinformatique en Parallélisme, Architecture et Système (ComPAS'14)*, April 22-25, 2014, Neuchatel, Switzerland.

[26] Vasseur, R., Baud, S., **Steffenel, L.A.**, Vigouroux, X., Martiny, L., Krajecki, M., Dauchez, M. A Framework for Inverse Virtual Screening. *6th International Conference on Bioinformatics, Biocomputational Systems and Biotechnologies (BIOTECHNO 2014)*, Chamonix, France, 20-24 April 2014 BEST PAPER Award

[27] Diallo, T. A., Flauzac, O., **Steffenel, L.A.**, Ndiaye, S., and Dieng, Y. GrAPP&S : A Distributed Framework for E-learning Resources Sharing. *Proceedings of the 5th International IEEE EAI Conference on eInfrastructure and eServices for Developing Countries (AFRICOMM 2013)*, Blantyre, Malawi, November 25-27 2013. Springer LNICST v. 135, pp. 219-228.

[28] **Steffenel, L.A.**, Flauzac, O., Schwertner Charao, A., Pitthan Barcelos, P., Stein, B., Nesmachnow, S., Kirsch Pinheiro, M., Diaz, D. PER-MARE : Adaptive Deployment of MapReduce over Pervasive Grids. *Proceedings of the 8th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC'13)*, Compiegne, France, October 28-30 2013.

[29] Vasseur, R., Baud, S., **Steffenel, L.A.**, Vigouroux, X., Martiny, L., Krajecki, M., Dauchez, M. Parallel Strategies for an Inverse Docking Method. *PBio 2013 : International Workshop on Parallelism in Bioinformatics (part of EuroMPI 2013)*, Madrid, Spain, 15-18 September 2013. pp. 253-258

[30] Najar, S. Kirsch-Pinheiro, M., Souveyet, C., **Steffenel, L.A.** Service Discovery Mechanisms for an Intentional Pervasive Information System. *Proceedings of 19th IEEE International Conference on Web Services (ICWS 2012)*, Honolulu, Hawaii, 24-29 June 2012.

[31] **Steffenel, L.A.**, Jaillet, C., Flauzac, O., Krajecki, M. Impact of nodes distribution on the performance of a P2P computing middleware based on virtual rings. *Proceedings of the Conferencia Latino Americana de Computación de Alto Rendimiento (CLCAR 2010)*, Gramado, Brazil, 25-28 August 2010.

[32] Fkaier, H., Cérin, C., **Steffenel, L.A.**, Jemni, M. A new heuristic for broadcasting in clusters of clusters. *Proceedings of the 5th International Conference on Grid and Pervasive Computing (GPC 2010)*, Hualien, Taiwan, 10-14 May 2010.

[33] Flauzac, O., Nolot, F., Rabat, C., **Steffenel, L.A.** Grid of security : a new approach of the network security. *Proceedings of the 3rd International Conference on Network & System Security (NSS 2009)*, Gold Coast, Autralia, 19-21 October 2009. pp. 67-72

[34] **Steffenel, L.A.**, Kirsch-Pinheiro, M. Strong Consistency for Shared Objects in Pervasive Grids. *Proceedings of the 5th IEEE International Conference on Wireless and Mobile Computing, Networking and Communication (WiMob'2009)*, Marrakesh, Morroco, 12-14 October 2009. pp. 73-78

[35] Fathallah, K., Nasri, W., **Steffenel, L.A.** On the Evaluation of OpenMP Memory Access in Multi-core Architectures. *4th International Workshop on Automatic Performance Tuning (iWAPT 2009)*, Tokio, Japan, 1-2 October 2009.

[36] Achour, S., Nasri, W., **Steffenel, L.A.** On the use of performance models for adaptive algorithm selection on heterogeneous clusters. *Proceedings of the 17th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2009)*, Weimar, Germany, 18-20 February 2009.

[37] **Steffenel, L.A.**, Kirsch-Pinheiro, M., Bebers, Y. Total Order Broadcast on Pervasive Systems. *Proceedings of the 23rd Annual ACM Symposium on Applied Computing (SAC 2008)*, Fortaleza, Brazil, 16-20 March 2008. pp. 2202-2206.

[38] Jeannot, E., **Steffenel, L.A.** Fast and Efficient Total Exchange on Two Clusters. *Proceedings of the 13th International Conference on Parallel Computing (EURO-PAR 2007)*, Rennes, France, 28-31 August 2007. Springer. LNCS 4641, pp. 848-857.

[39] **Steffenel, L.A.**, Jeannot, E. Total Exchange Performance Prediction on Grid Environments : modeling and algorithmic issues. *Towards Next Generation Grid - Proceedings of the CoreGRID Symposium*, Rennes, France, 27-28 August 2007. Springer, pp. 131-140.

[40] **Steffenel, L.A.**, Martinasso, M. and Trystram, D. Assessing contention effects on MPI_Alltoall communications. *GPC 07 - International Conference on Grid and Pervasive Computing*, Paris, France, May 2007. LNCS 4459, Springer Verlag, pp 424-435.

[41] Nasri, W., **Steffenel, L.A.** and Trystram, D. Adaptive performance modeling on hierarchical grid computing environments. *7th IEEE International Symposium on Cluster Computing and the Grid - CCGrid 2007*, Rio de Janeiro, Brazil, pp 505-512, May 2007. IEEE Society.

[42] **Steffenel, L.A.** Modeling Network Contention Effects on AlltoAll Operations. *IEEE Conference on Cluster Computing (CLUSTER 2006)*. Barcelona, Spain, 25-28 September 2006.

[43] **Barchet-Steffenel, L. A.**, Mounié, G. Scheduling Heuristics for Efficient Broadcast Operations on Grid Environments. *International Workshop on Performance Modeling, Evaluation, and Optimisation of Parallel and Distributed Systems (PMEO-PDS'06)*, in conjunction with IPDPS'06. Rhodes Island, Greece, 25-29 Avril 2006.

[44] **Barchet-Steffenel, L. A.**, Mounié, G. Total Exchange Performance Modelling under Network Contention. *Proceedings of the 6th International Conference on Parallel Processing and Applied Mathematics*, LNCS vol. 3911, Springer-Verlag, pp 100-107. Poznan, Pologne. 2005.

[45] **Barchet-Steffenel, L. A.**, Mounié, G. Performance Characterisation of Intra-Cluster Collective Communications. *Proceedings of the SBAC-PAD 2004 16th Symposium on Computer Architecture and High Performance Computing*, Foz-do-Iguacu, Brazil, IEEE Press, pp. 254-261, 2004.

[46] **Barchet-Steffenel, L. A.**, Mounié, G. Identifying Logical Homogeneous Clusters for Efficient Wide-area Communications. *Proceedings of the EuroPVM/MPI 2004 11th European PVM/MPI Users' Group Meeting (2004)*, Budapest, Hungary, September 2004. LNCS vol. 3241, Springer-Verlag, pp. 319-326, 2004.

[47] **Barchet-Steffenel, L. A.**, Mounié, G. Fast Tuning of Intra-Cluster Collective Communications. *Proceedings of the EuroPVM/MPI 2004 11th European PVM/MPI Users' Group Meeting (2004)*, Budapest, Hungary, September 2004. LNCS vol. 3241, Springer-Verlag, pp. 28-35, 2004.

[48] **Barchet-Steffenel, L. A.** iRBP, A Fault Tolerant Total Order Broadcast for Large Scale Systems. *Proceedings of the 9th International Conference on Parallel Computing (EURO-PAR 2003)*, University Klagenfurt, Klagenfurt, Austria, August 2003. LNCS vol. 2790, Springer-Verlag, pp. 632-639, 2003.

[49] **Barchet-Steffenel, L. A.**; Jansch-Pôrto, I. On the Evaluation of heartbeat-like Detectors. *Proceedings of the IEEE 2nd Latin-American Test Workshop*, Cancun, México, February 2001, pp 142-147.

Nationales

[50] Nesi, L., Koslovski, G., Charão, A., Pinheiro, D., **Steffenel, L.A.**, Paralelização de Cálculos Estatísticos sobre Dados de Monitoramento da Camada de Ozônio : um Estudo com GPU. *Fórum de iniciação científica Escola Regional de Alto Desempenho (ERAD/RS)*, April 5-7, 2017, Ijui, Brazil.

[51] Muenchen, B., Siqueira, T., Nesi, L., Koslovski, G., Charão, A., Pinheiro, D., **Steffenel, L.A.**, Análise de Dados Observacionais sobre a Camada de Ozônio : uma Abordagem Usando MPI e OpenMP. *Fórum de iniciação científica Escola Regional de Alto Desempenho (ERAD/RS)*, April 5-7, 2017, Ijui, Brazil.

[52] **Steffenel, L.A.**, Kirsch-Pinheiro, M., Stratégies Multi-Échelle pour les Environnements Pervasifs et lInternet des Objets. *Proceedings of 11èmes Journées Francophones Mobilité et Ubiquité (Ubimob 2016)*, July 5, Lorient, France.

[53] Diallo, T.H., Flauzac, O., **Steffenel, L.A.**, Ndiaye, S., Routage Préfixé dans GRAPP&S. *Proceedings of Colloque Africain sur la Recherche en Informatique et en Mathématiques Appliquées (CARI'2014)*, October 20-23, 2014, Saint Louis, Sénégal.

[54] Diallo, T.H., Ndiaye, S., Flauzac, O., **Steffenel, L.A.**, GRAPP&S, une Architecture Multi-Échelle pour les Données et le Services. *Proceedings of 9èmes Journées Francophones Mobilité et Ubiquité (Ubimob 2013)*, June 5-6, 2013, Nancy, France.

[55] Najar, S. Kirsch-Pinheiro, M., **Steffenel, L. A.**, Souveyet, C. Analyse des mécanismes de découverte de services avec prise en charge du contexte et de l'intention. *Proceedings of 8èmes Journées Francophones Mobilité et Ubiquité (Ubimob 2012)*, June 4-6, 2012, Anglet, France.

[56] Flauzac, O., **Steffenel, L.A.**, Diallo, T.H., Niang, I., Ndiaye, S. GRAPP&S Data Grid : Une approche de type grille et système pair-à-pair pour le stockage de données. *Colloque National sur la Recherche en Informatique et ses Applications (CNRIA'2012)*, Thiès/Bambey, Senegal. April 25-27, 2012.

[57] **Steffenel, L.A.**, Boisson, J-C., Barberot, C., Gérard, S., Hénon, E., Jaillet, C., Flauzac, O., Krajecki, M. Deploying a fault-tolerant computing middleware over Grid5000 : performance analysis of CONFIIT and its integration with a quantum molecular docking application. *4th Grid'5000 Spring School*, Reims, France, April 18-21, 2011.

[58] **Barchet-Steffenel, L. A.**, Mounié, G. Prédiction de Performances pour les Communications Collectives. *Proceedings of the 16ème Rencontre Francophone du Parallélisme (RenPar'16)*, Le Croisic, France, pp. 101-112, April 2005.

[59] **Barchet-Steffenel, L. A.**, Jansch-Pôrto, I. On the Evaluation of Failure Detectors Performance. *Proceedings of the IX Simpósio de Computação Tolerante a Falhas (IX SCTF)*, Florianópolis, Brazil, March 2001, pp 73-84.

[60] **Barchet-Steffenel, L. A.** Avaliação Prática do Desempenho dos Detectores de Defeitos (Practical Evaluation of Failure Detectors Performance). *Workshop de Teses e Dissertações do IX SCTF*, Florianópolis, Brazil, March 2001.

[61] **Barchet-Steffenel, L. A.**, Jansch-Pôrto, I. Comunicação Não Confiável em Detectores de Defeitos com Falhas por Crash (Unreliable Communication in Failure Detectors with Crash Faults). *Proceedings of the II Workshop de Testes e Tolerância a Falhas*, Curitiba, Brazil, July 2000.

[62] **Barchet-Steffenel, L. A.**, Jansch-Pôrto, I. Avaliação Prática de um Detector de Defeitos : teoria versus implementação (Practical Evaluation of a Failure Detector : theory versus implementation). *Proceedings of the II Workshop de Testes e Tolerância a Falhas*, Curitiba, Brazil, July 2000.

[63] **Barchet-Steffenel, L. A.** Avaliação Prática dos Detectores de Defeitos e sua Influência no Desempenho das Operações de Consenso (Practical Evaluation of Failure Detectors and their Influence on the Consensus Operations Performance). *Proceedings of the V Semana Acadêmica do PPGC-UFRGS*, Porto Alegre, Brazil, July 2000.

[64] **Barchet-Steffenel, L. A.** Estudo sobre Comunicação de Grupos para Tolerância a Falhas (A Survey on Group Communication for Fault Tolerance). *Proceedings of the IV Simpósio Nacional de Informática*, Centro Universitário Franciscano, Santa Maria, Brazil, 1999.

[65] **Barchet-Steffenel, L. A.** Csockets - classes para comunicação de rede com autenticação e criptografia (CSocketS - Communication Classes with Authentication and Cryptography). *Proceedings of the V Jornada Integrada de Pesquisa da UFSM*, Santa Maria, Brazil, 1998.

Ouvrage Scientifique / Livres

[66] **Steffenel, L.A.**, Communications Collectives pour les Grilles de Calcul , *Éditions Universitaires Européennes*, 2010. 188 pages. ISBN 978-613-1-53126-2

Chapitres de Livres

[67] Najar, S., Vanrompay, Y., Kirsch-Pinheiro, M., **Steffenel, L.A.**, Souveyey, C., Intention Prediction Mechanism in an Intentional Pervasive Information System in K Kolomvatsos, C Anagnostopoulos, and C Hadjiefthymiades (Eds.), *Intelligent Technologies and Techniques for Pervasive Computing*, IGI Global, pp. 251-275. Mai 2013. pp. 251-275. ISBN 978-1-4666-4038-2

[68] Flauzac, O., Nolot, F., Rabat, C., **Steffenel, L.A.**, Grid of Security : a decentralized enforcement of the network security in Manish Gupta, John Walp, and Raj Sharman (Eds.), *Threats, Countermeasures and Advances in Applied Information Security*, IGI Global, april 2012. pp. 426-443. ISBN 9781466609785

[69] Cérin, C., **Steffenel, L.A.**, Fkaier, H., Broadcasting for Grids in Frédéric Magoules (Eds.), *Fundamentals in Grid Computing : Theory, Algorithms and Technologies*, Chapman & Hall/-CRC Numerical Analysis and Scientific Computing Series, chapter 8, pp 209-236, december 2009. ISBN 9781439803677

Abstracts/Posters avec actes et comité de sélection

Internationales

[70] Vasseur, R., Haschka, T., Verzeaux, L., Albin, J., **Steffenel, L.A.**, Khartabil, H., Belloy, N., Baud, S., Henon, E., Krajecki, M., Martiny, L., Dauchez, M., Deciphering molecular interactions using HPC simulations : getting new therapeutic targets, *9th Ter@tec Forum*, Palaiseau, France, July 1-2, 2014.

[71] Deleau, H., Jaillet, C., Krajecki, M. and **Steffenel, L.A.**, Towards the Parallel Resolution of the Langford Problem on a Cluster of GPU Devices, *Sixth SIAM Workshop on Combinatorial Scientific Computing (CSC14)*, Lyon, France, July 21-23, 2014.

[72] Barberot, C., Boisson, J-C., Thiriot, E., Gérard, S., Monard, G., **Stefenel, L-A.**, Hénon, E. Study of PDE4 Inhibitors : Quantum Mechanical Molecular Docking Deployed in a Grid using CONFIIT. *9th Triennial Congress of the World Association of Theoretical and Computational Chemists (WATOC 2011)*. Santiago de Compostela, Spain, 17-22 July 2011. BEST POSTER AWARD

[73] Nasri, W., Achour, S., **Steffenel, L. A.** Integrating performance models and adaptive approaches for efficient parallel algorithms. *5th International Workshop on Parallel Matrix Algorithms and Applications (PMAA'08)*, Neuchâtel, Switzerland, pp. 18, 20-22 June 2008.

Nationales

[74] Vasseur, R., Baud, S., **Steffenel, L.A.**, Vigouroux, N., Martiny, L., Krajecki, M., Dauchez, M., 'Developments for a Novel Inverse Docking Method , *Journées de la Société Française de Chémoinformatique*, Nancy, France, 10 et 11 octobre 2013.

[75] Vasseur, R., Baud, S., **Steffenel, L.A.**, Vigouroux, N., Martiny, L., Krajecki, M., Dauchez, M., Développements HPC pour une nouvelle méthode de docking inverse , *Journée ROMEO*, Université de Reims Champagne-Ardenne, Reims, France, 26 juin 2012.

[76] **Barchet-Steffenel, L. A.**; Ceretta-Nunes, R. Implementação de uma Situação de Corrida Crítica em Java (Implementation of a Critical Run Situation in Java), *Proceedings of the II Ciclo de Palestras do Curso de Informática*, UFSM, Santa Maria, Brazil, october 1997

Thèses et dissertations

[77] **Barchet-Steffenel, L.A.** LaPIe : Communication Collectives Adaptées aux Grilles de Calcul. *PhD Thesis*, INPG, Grenoble, France, December 2005.

[78] **Barchet-Steffenel, L.A.** Analyzing RBP, a Total Order Broadcast Protocol for Unreliable Channels. *MSc. Dissertation*, Doctoral School in Communication Systems, EPFL, Lausanne, Switzerland, July 2002.

[79] **Barchet-Steffenel, L. A.** Avaliação dos Detectores de Defeitos e sua Influência nas Operações de Consenso (Evaluation of Failure Detectors and their Influence on the Consensus Operations), *Master Thesis*, PPGC :UFRGS, Porto Alegre, Brazil, March 2001.

[80] **Barchet-Steffenel, L. A.** CsocketS - Classes para Comunicação de Rede com Autenticação e Criptografia usando Java e CORBA (CSocketS - Classes for Network Communication with Authentication and Criptography, using Java and CORBA). *Bsc. Dissertation*, UFSM, Santa Maria, Brazil, March 1999.

Rapports de recherche

[81] **Steffenel, L. A.** D2.1 - First Steps on the Development of a P2P Middleware for MapReduce. Deliverable for project STIC-AmSud PER-MARE (13STIC-07), July 2013.

[82] **Steffenel, L. A.** Modeling Network Contention Effects on AlltoAll Operations. Rapport de recherche INRIA RR-6038 (extended version of the Cluster06 paper), November 2006.

[83] **Steffenel, L. A.** Fast and Scalable Total Order Broadcast for Wide-area Networks. Rapport de recherche INRIA RR-6037, November 2006.

[84] **Steffenel, L. A.** A Framework for Adaptive Collective Communications on Heterogeneous Hierarchical Networks. Rapport de recherche INRIA RR-6036 (extended version of the IPDPS06 paper), November 2006.

[85] F. Cappello, P. Owezarski, R. Namyst, O. Richard, P. Vicat-Blanc-Primet, E. Jeannot, **L.A. Steffenel,** D. Caromel, P. Sens, P. Fraigniaud, C. Cérin, S. Petiton, J. Gustedt, C. Blanchet, C. Randriamaro, S. Tixieuil. Data Grid Explorer. Rapport LAAS No05491, Projet ACI Masse de Données. Data Grid eXplorer, Septembre 2005, 48p.

[86] **Steffenel, L. A.** Detectores de Defeitos não Confiáveis (Unreliable Failure Detectors). Research Report (Trabalho Individual), PPGC-UFRGS, January 2000.

Conférences invitées

[87] Keynote *Harvesting the mist : expanding HPC with the help of surrounding resources*. Latin America High Performance Computing Conference (CARLA 2017), 21 september 2017. Buenos Aires, Argentina

[88] Panel *Migrating to Cloud and IoT Solutions : Challenges and Perspectives*, CLIoT/CloudWay workshops joint panel, 15 septembre 2015, Catania, Italy

[89] *PER-MARE : adaptation du paradigme MapReduce aux réseaux pervasifs*, Réunion RGE - Réseaux Grand Est, 13 février 2014, Reims.

[90] *O Ensino Superior em Computação na França" (L'enseignement supérieur en Informatique en France)*, Rentrée solennelle du cours en Informatique à l'UFSM, Universidade Federal de Santa Maria (UFSM, Brésil), 30 avril 2013

[91] *Structure et gestion du Centre de Calcul ROMEO*, IUT Villetaneuse (Université Paris 13), mai 2011

[92] *Deploying large scale application with CONFIIT : study on the Quantum Molecular Docking problem*, Journée ROMEO, 18 mai 2011, Reims

[93] *RemoteLabz - Environnement virtuel pour lapprentissage des technologies réseau*, Universidade Federal de Santa Maria (Brésil), juillet 2010

[94] *CONFIIT : un système pair-à-pair pour le calcul*, École Supérieure de Sciences et Techniques de Tunis, mai 2009.

[95] *GPGPU : du calcul haute performance dans votre machine*, École Supérieure de Sciences et Techniques de Tunis, mai 2009.

[96] *Modélisation des Communications de type All-to-All sujettes à la congestion du réseau*, Réunion RGE - Réseaux Grand Est, Strasbourg, 1er juin 2006.

[97] *Modélisation des Communications pour les Grilles de Calcul*, Réunion GridExplorer (GdX). Lyon, 21 juin 2005.

Bibliographie

[1] ABAGYAN, R., AND TOTROV, M. High-throughput docking for lead generation. *Curr. Opin. Chem. Biol. 5*, 4 (2001), 375–382.

[2] ALEXANDROV, A., IONESCU, M., SCHAUSER, K., AND SCHEIMAN, C. LogGP : Incorporating long messages into the LogP model - one step closer towards a realistic model for parallel computation. In *Proceedings of the 7th Annual Symposium on Parallel Algorithms and Architecture (SPAA'95)* (1995).

[3] ALLEN, F. H. Acta crystallogr. b. *The Cambridge Structural Database : a quarter of a million crystal structures and rising 58*, 3 (2002), 380–388.

[4] APACHE. Apache hadoop, 2014. `http://hadoop.apache.org/docs/r2.6.0/index.html`. Last access : November 2014.

[5] BALDAUF, M., DUSTDAR, S., AND ROSENBERG, F. A survey on context-aware systems. *Int. J. Ad Hoc Ubiquitous Comput. 2*, 4 (June 2007), 263–277.

[6] BANIKAZEMI, M., MOORTHY, V., AND PANDA, D. K. Efficient collective communication on heterogeneous networks of workstations. In *Proceedings of the International Conference on Parallel Processing (ICPP'98)* (1998), pp. 460–467.

[7] BAR-NOY, A., AND KIPNIS, S. Designing broadcasting algorithms in the postal model for message-passing systems. *Math. Systems Theory 27*, 5 (1994), 431–452.

[8] BARCHET-STEFFENEL, L. A. *LaPIe : communications collectives adaptées aux grilles de calcul*. PhD thesis, INPG, France, Dec. 2005.

[9] BARCHET-STEFFENEL, L. A., AND MOUNIE, G. Fast tuning of intra-cluster collective communications. In *Proceedings of the Euro PVM/MPI 2004* (Budapest, Hungary, 2004), LNCS Vol. 3241, pp. 28–35.

[10] BARCHET-STEFFENEL, L. A., AND MOUNIE, G. Performance characterisation of intra-cluster collective communications. In *Proceedings of the 16th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2004)* (Foz do Iguacu, Brazil, Sep 2004), pp. 254–261.

[11] BARNETT, M., PAYNE, D., VAN DE GEIJN, R., AND WATTS, J. Broadcasting on meshes with wormhole routing. *Journal of Parallel and Distributed Computing 35*, 2 (1996), 111–122.

[12] BEAUMONT, O., AND MARCHAL, L. Pipelining broadcasts on heterogeneous platforms under the one-port model. Tech. rep., LIP, ENS Lyon, France, 2004.

[13] BEAUMONT, O., MARCHAL, L., AND ROBERT, Y. Broadcast trees for heterogeneous platforms. Tech. rep., LIP, ENS Lyon, France, 2004.

[14] BEAUMONT, O., MARCHAL, L., AND ROBERT, Y. Broadcasts trees for heterogeneous platforms. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2005)* (2005).

[15] BERNASCHI, M., AND IANNELLO, G. Collective communication operations : Experimental results vs. theory. *Concurrency : Practice and Experience 10*, 5 (April 1998), 359–386.

[16] BESERRA, D., KIRSCH-PINHEIRO, M., STEFFENEL, L. A., AND MORENO, E. Comparing the performance of os-level virtualization tools in soc-based systems : The case of i/o-bound applications. In *22nd IEEE Symposium on Computers and Communications (ISCC 2017)* (Heraklion, Greece, July 2017).

[17] BESERRA, D., KIRSCH-PINHEIRO, M., STEFFENEL, L. A., SOUVEYET, C., AND MORENO, E. Performance evaluation of os-level virtualization solutions for hpc purposes on soc-based systems. In *IEEE International Conference on Advanced Information Networking and Applications (AINA)* (Taipei, Taiwan, March 2017).

[18] BHAT, P. B., PRASANNA, V., AND RAGHAVENDRA, C. Adaptive communication algorithms for distributed heterogeneous systems. *Journal of Parallel and Distributed Computing 1999*, 59 (1999), 252–279.

[19] BHAT, P. B., RAGHAVENDRA, C., AND PRASANNA, V. Efficient collective communication in distributed heterogeneous systems. *Journal of Parallel and Distributed Computing 2003*, 63 (2003), 251–279.

[20] BHATIA, M., MANRAL, V., AND OHARA, Y. IS-IS and OSPF Difference Discussion. IETF Internet Draft, Jan. 2006.

[21] BONOMI, F., MILITO, R., ZHU, J., AND ADDEPALLI, S. Fog computing and its role in the internet of things. In *Proceedings of the 1st MCC Workshop on Mobile Cloud Computing* (New York, NY, USA, 2012), MCC '12, ACM, pp. 13–16.

[22] BRINKSMA, E. J., MEIJER, Y. J., CONNOR, B. J., MANNEY, G. L., BERGWERFF, J. B., BODEKER, G. E., BOYD, I. S., LILEY, J. B., HOGERVORST, W., HOVENIER, J. W., LIVESEY, N. J., AND SWART, D. P. J. Analysis of record-low ozone values during the 1997 winter over lauder, new zealand. *Geophysical Research Letters 25*, 15 (1998), 2785–2788.

[23] BRUCK, J., HO, C.-T., KIPNIS, S., UPFAL, E., AND WEATHERSBY, D. Efficient algorithms for all-to-all communications in multiport message-passing systems. *IEEE Transactions on Parallel and Distributed Systems 8*, 11 (November 1997), 1143–1156.

[24] CAPIT, N., DA COSTA, G., GEORGIOU, Y., HUARD, G., MARTIN, C., MOUNIE, G., NEYRON, P., AND RICHARD, O. A batch scheduler with high level components. In *May* (2005), vol. 2, pp. 776–783.

[25] CASANOVA, H. Network modeling issues for grid application scheduling. *International Journal of Foundations of Computer Science 16*, 2 (2005), 145–162.

[26] CASICCIA, C., ZAMORANO, F., AND HERNANDEZ, A. Erythemal irradiance at the magellan's region and antarctic ozone hole 1999-2005. *Atmosfera 21*, 1 (2008), 2–12.

[27] CASSALES, G. W., CHARAO, A. S., KIRSCH-PINHEIRO, M., SOUVEYET, C., AND STEFFENEL, L. A. Bringing context to apache hadoop. In *8th International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2014)* (Rome, Italy, Aug 2014), IARIA, pp. 252–258.

[28] CASSALES, G. W., CHARAO, A. S., KIRSCH-PINHEIRO, M., SOUVEYET, C., AND STEFFENEL, L. A. Context-aware scheduling for apache hadoop over pervasive environments. In *The 6th International Conference on Ambient Systems, Networks and Technologies (ANT 2015)* (London, UK, Jun 2015), vol. 52 of *Procedia Computer Science*, Elsevier, pp. 202–209.

[29] CASSALES, G. W., CHARAO, A. S., KIRSCH-PINHEIRO, M., SOUVEYET, C., AND STEFFENEL, L. A. Improving the performance of apache hadoop on pervasive environments through context-aware scheduling. *Springer Journal of Ambient Intelligence and Humanized Computing 7*, 3 (2016), 333–345.

[30] CASSALES, G. W., CHARAO, A. S., PINHEIRO, M. K., SOUVEYET, C., AND STEFFENEL, L. A. Bringing context to apache hadoop. In *8th International Conference on Mobile Ubiquitous Computing* (Rome, Italy, 2014).

[31] CAVALLO, M., CUSMA, L., MODICA, G. D., POLITO, C., AND TOMARCHIO, O. A scheduling strategy to run hadoop jobs on geodistributed data. In *3rd Workshop on CLoud for IoT (CLIoT 2015), in conjunction with the European Conference on Service-Oriented and Cloud Computing (ESOCC 2015)* (2015).

[32] CHALOPIN, J., GODARD, E., MÉTIVIER, Y., AND OSSAMY, R. Mobile agent algorithms versus message passing algorithms. In *Principle of distributed systems* (France, 2006), vol. 4305 of *LNCS*, Springer, pp. 185–199.

[33] CHANDRA, T. D., AND TOUEG, S. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM 43*, 2 (1996), 225–267.

[34] CHARAO, A. S., HOFFMANN, G., STEFFENEL, L. A., KIRSCH-PINHEIRO, M., AND STEIN, B. Performance evaluation of cloud-based rdbms through a cloud scripting language. In *19th International Conference on Enterprise Information Systems (ICEIS)* (Porto, Portugal, April 2017).

[35] CHEN, Q., ZHANG, D., GUO, M., DENG, Q., AND GUO, S. Samr : A self-adaptive mapreduce scheduling algorithm in heterogeneous environment. In *Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology* (Washington, DC, USA, 2010), CIT '10, IEEE Computer Society, pp. 2736–2743.

[36] CHEN, Y. Z., AND UNG, C. Y. Prediction of potential toxicity and side effect protein targets of a small molecule by a ligand–protein inverse docking approach. *Journal of Molecular Graphics and Modelling 20*, 3 (2001), 199–218.

[37] CHEN, Y. Z., AND ZHI, D. G. Ligand–protein inverse docking and its potential use in the computer search of protein targets of a small molecule. *Proteins : Structure, Function, and Bioinformatics 43*, 2 (2001), 217–226.

[38] CHRISTARA, C., DING, X., AND JACKSON, K. An efficient transposition algorithm for distributed memory computers. In *Proceedings of the High Performance Computing Systems and Applications* (1999), pp. 349–368.

[39] CHUN, A. T. T. *Performance Studies of High-Speed Communication on Commodity Cluster*. PhD thesis, University of Hong Kong, 2001.

[40] CISCO. Cisco iox.

[41] CISCO RESEARCH CENTER REQUESTS FOR PROPOSALS (RFPS). Fog computing, ecosystem, architecture and applications. `http://www.cisco.com/web/about/ac50/ac207/crc_new/university/RFP/rfp13078.html`.

[42] CLEMENT, M., STEED, M., AND CRANDALL, P. Network performance modelling for PM clusters. In *Proceedings of Supercomputing* (1996).

[43] CULLER, D., KARP, R., PATTERSON, D., SAHAY, A., SCHAUSER, K. E., SANTOS, E., SUBRAMONIAN, R., AND VON EICKEN, T. LogP - a practical model of parallel computing. *Communication of the ACM 39*, 11 (1996), 78–85.

[44] DE SUPINSKI, B. R., AND KARONIS, N. T. Accurately measuring MPI broadcasts in a computational grid. In *Proceedings of the IEEE International Symposium on High Performance Distributed Computing (HPDC'99)* (August 1999).

[45] DEAN, J., AND GHEMAWAT, S. Mapreduce : Simplified data processing on large clusters. *Commun. ACM 51*, 1 (Jan. 2008), 107–113.

[46] DEY, S., MUKHERJEE, A., PAUL, H. S., AND PAL, A. Challenges of using edge devices in iot computation grids. In *Int. Conf. on Parallel and Distributed Systems* (2013), pp. 564–569.

[47] DIALLO, T., FLAUZAC, O., STEFFENEL, L. A., AND NDIAYE, S. Routage préfixé dans grapp&s. In *Colloque Africain sur la Recherche en Informatique et en Mathématiques Appliquées (CARI'2014)* (Saint Louis, Sénégal, October 2014).

[48] DIALLO, T., FLAUZAC, O., STEFFENEL, L. A., AND NDIAYE, S. Grapp&s, a peer-to-peer middleware for interlinking and sharing educational resources. *Transactions on Industrial Networks and Intelligent Systems 2(3)*, e1 (May 2015).

[49] DIALLO, T., FLAUZAC, O., STEFFENEL, L. A., NDIAYE, S., AND DIENG, Y. Grapp&s : A distributed framework for e-learning resources sharing. In *5th International IEEE EAI Conference on eInfrastructure and eServices for Developing Countries (AFRICOMM 2013)* (Blantyre, Malawi, Nov 2013), Springer, Ed., vol. 135 of *LNICST*, pp. 219–228.

[50] DIALLO, T., NDIAYE, S., FLAUZAC, O., AND STEFFENEL, L. A. Grapp&s, une architecture multi-échelle pour les données et le services. In *9èmes Journées Francophones Mobilité et Ubiquité (Ubimob 2013)* (Nancy, France, June 2013).

[51] ELKHATIB, Y., PORTER, B., RIBEIRO, H. B., ZHANI, M. F., QADIR, J., AND RIVIÈRE, E. On using micro-clouds to deliver the fog. *IEEE Internet Computing 21*, 2 (2017), 8–15.

[52] ENGEL, T. A., CHARAO, A. S., KIRSCH-PINHEIRO, M., AND STEFFENEL, L. A. Performance improvement of data mining in weka through gpu acceleration. In *5th International Conference on Ambient Systems, Networks and Technologies (ANT 2014)* (Hasselt, Belgium,, June 2014), vol. 32 of *Procedia Computer Science*, Elsevier, pp. 93–100.

[53] ENGEL, T. A., CHARAO, A. S., KIRSCH-PINHEIRO, M., AND STEFFENEL, L. A. Performance improvement of data mining in weka through multi-core and gpu acceleration : opportunities and pitfalls. *Journal of Ambient Intelligence and Humanized Computing 6*, 4 (June 2015), 377–390.

[54] ETSI. Mobile-edge computing - introductory technical white paper. `https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobile-edge_Computing_-_Introductory_Technical_White_Paper_V1%2018-09-14.pdf`.

[55] FARMAN, J., GARDINER, G., AND SHANKLIN, J. Large losses of total ozone in antarctica reveal seasonal clox/nox interaction. *Nature 315* (1985), 207–210.

[56] FAZIO, M., CELESTI, A., PULIAFITO, A., AND VILLARI, M. Big data storage in the cloud for smart environment monitoring. *Procedia Computer Science 52* (2015), 500 – 506. The 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015).

[57] FLAUZAC, O., KRAJECKI, M., AND FUGÈRE, J. CONFIIT : a middleware for peer to peer computing. In *The 2003 International Conference on Computational Science and its Applications (ICCSA 2003)*, M. Gravilova, C. Tan, and P. L'Ecuyer, Eds., vol. 2669 (III) of *Lecture Notes in Computer Science*. Springer-Verlag, Montréal, Québec, June 2003, pp. 69–78.

[58] FLAUZAC, O., KRAJECKI, M., AND STEFFENEL, L.-A. Confiit : a middleware for peer-to-peer computing. *The Journal of Supercomputing 53* (2010), 86–102.

[59] FLAUZAC, O., NOLOT, F., RABAT, C., AND STEFFENEL, L. Grid of security : a decentralized enforcement of the network security. In *Threats, Countermeasures and Advances in Applied Information Security*, I. Global, Ed. Manish Gupta, John Walp, and Raj Sharman (Eds.), April 2012, pp. 426–443.

[60] FLAUZAC, O., STEFFENEL, L., DIALLO, T., NIANG, I., AND NDIAYE, S. Grapp&s data grid un système pair-à-pair de stockage des données. In *Colloque National sur la Recherche en Informatique et ses Applications (CNRIA'2012)* (Thiès/Bambey, Senegal, April 2012).

[61] FLAUZAC, O., STEFFENEL, L. A., DIALLO, T., NIANG, I., AND NDIAYE, S. Grapp&s data grid : Une approche de type grille et système pair-à-pair pour le stockage de données. *Revue URED (Université-Recherche-Développement)* (2012).

[62] GABRIEL, E., RESCH, M., BEISEL, T., AND KELLER, R. Distributed computing in a heterogeneous computing environment. In *Proc. of the Euro PVM/MPI 1998* (1998), LNCS 1497, Springer, pp. 180–187.

[63] GARCIA LOPEZ, P., MONTRESOR, A., EPEMA, D., DATTA, A., HIGASHINO, T., IAMNITCHI, A., BARCELLOS, M., FELBER, P., AND RIVIERE, E. Edge-centric computing : Vision and challenges. *SIGCOMM Comput. Commun. Rev. 45*, 5 (Sept. 2015), 37–42.

[64] GHERSI, D., AND SANCHEZ, R. Improving accuracy and efficiency of blind protein-ligand docking by focusing on predicted binding sites. *Proteins : Structure, Function and Bioinformatics 74*, 2 (2009), 417–424.

[65] GIGANTI, D., GUILLEMAIN, H., SPADONI, J.-L., NILGES, M., ZAGURY, J.-F., AND MONTES, M. Comparative evaluation of 3d virtual ligand screening methods : Impact of the molecular alignment on enrichment. *J. Chem. Inf. Model. 50*, 6 (2010), 992–1004.

[66] GRID'5000. Grid 5000, 2013. `https://www.grid5000.fr/`, Last access : July 2014.

[67] GROVE, D. *Performance Modelling of Message-Passing Parallel Programs*. PhD thesis, University of Adelaide, 2003.

[68] GUARNIERI, R., PADILHA, L., GUARNIERI, F., ECHER, E., MAKITA, K., PINHEIRO, D., SCHUCH, A., BOEIRA, L., AND SCHUCH, N. A study of the anticorrelations between ozone and uv-b radiation using linear and exponential fits in southern brazil. *Advances in Space Research 34*, 4 (2004), 764 – 76.

[69] GUBBI, J., BUYYA, R., MARUSIC, S., AND PALANISWAMI, M. Internet of things (iot) : A vision, architectural elements, and future directions. *Future Generation Computer Systems 29*, 7 (2013), 1645 – 1660.

[70] GUILLOUX, V. L., SCHMIDTKE, P., AND TUFFERY, P. Fpocket : An open source platform for ligand pocket detection. *BMC Bioinformatics 10*, 1 (2009), 168.

[71] HAMILTON, J. Hadoop wins terasort, Jul 2008. `http://perspectives.mvdirona.com/2008/07/hadoop-wins-terasort/`, Last access : Sep 2015.

[72] HAO, Z., NOVAK, E., YI, S., AND LI, Q. Challenges and software architecture for fog computing. *IEEE Internet Computing 21*, 2 (2017), 44–53.

[73] HENDERSON, R. L. Job scheduling under the portable batch scheduler. In *Proceedings of the IPPS'95 workshop* (1995), LNCS Vol. 949, pp. 279–294.

[74] HETENYI, C., AND VAN DER SPOEL, D. Toward prediction of functional protein pockets using blind docking and pocket search algorithms. *Protein Science*, 20 (2011), 880–893.

[75] HOCKNEY, R. The communication challenge for MPP : Intel Paragon and Meiko CS-2. *Parallel Computing 20* (1994), 389–398.

[76] HUANG, S., HUANG, J., DAI, J., XIE, T., AND HUANG, B. The hibench benchmark suite : Characterization of the mapreduce-based data analysis. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on* (March 2010), pp. 41–51.

[77] HUNT, P., KONAR, M., JUNQUEIRA, F. P., AND REED, B. Zookeeper : Wait-free coordination for internet-scale systems. In *Proceedings of the USENIX Annual Technical Conference* (Boston, MA, 2010), USENIX Association, pp. 11–11.

[78] HUSE, L. P. Collective communication on dedicated clusters of workstations. In *Proceedings of the European PVM/MPI Users' Group Meeting* (1999), pp. 469–476.

[79] IRWIN, J. J., AND SHOICHET, B. K. Zinc-a free database of commercially available compounds for virtual screening. *J. Chem. Inf. Model. 45*, 1 (2005), 177–182.

[80] ISARD, M., PRABHAKARAN, V., CURREY, J., WIEDER, U., TALWAR, K., AND GOLDBERG, A. Quincy : fair scheduling for distributed computing clusters. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (New York, NY, USA, 2009), SOSP '09, ACM, pp. 261–276.

[81] JAILLET, C., AND KRAJECKI, M. Solving the langford problem in parallel. In *Proceedings of ISPDC'04, International Symposium on Distributed and Parallel Computing* (Cork (Ireland), 2004).

[82] JEANNOT, E., AND STEFFENEL, L. A. Fast and efficient total exchange on two clusters. In *EuroPar'07 - 13th International Euro-Par Conference European Conference on Parallel and Distributed Computing* (Rennes, France, Sept. 2007), LNCS 4631, Springer, pp. 832–841.

[83] JOHNEN, C., AND MEKHALDI, F. Self-stabilization versus robust self-stabilization for clustering in ad-hoc network. In *Proceedings of the 17th International Conference on Parallel Processing - Volume Part I* (2011), Euro-Par'11, Springer, pp. 117–129.

[84] JOHNSSONN, S. L., AND HO, C.-T. Optimum broadcasting and personalized communication in hypercubes. *IEEE Transactions on Computers 38*, 9 (Sep 1989), 1249–1268.

[85] JONES, M. Internet of things : Shifting from proprietary to standard. `http://www.valuewalk.com/2014/07/internet-of-things-iot/`.

[86] KIELMANN, T., BAL, H., GORLATCH, S., VERSTOEP, K., AND HOFMAN, R. Network performance-aware collective communication for clustered wide area systems. *Parallel Computing 27*, 11 (2001), 1431–1456.

[87] KIELMANN, T., BAL, H., AND VERSTOEP, K. Fast measurement of LogP parameters for message passing platforms. In *Proceedings of the 4th Workshop on Runtime Systems for Parallel Programming* (2000), LNCS Vol. 1800, pp. 1176–1183.

[88] KIELMANN, T., HOFMAN, R., BAL, H., PLAAT, A., AND BHOEDJANG, R. Magpie : MPI's collective communication operations for clustered wide area systems. In *Proceedings of the 7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (1999), pp. 131–140.

[89] KIRCHHOFF, V. W. J. H., SAHAI, Y., CASICCIA, C. A. R. S., ZAMORANO, B. F., AND VALDERRAMA, V. V. Observations of the 1995 ozone hole over punta arenas, chile. *Journal of Geophysical Research : Atmospheres 102*, D13 (1997), 16109–16120.

[90] KIRCHHOFF, V. W. J. H., SCHUCH, N., PINHEIRO, D., AND HARRIS, J. Evidence for an ozone hole perturbation at 30ž south. *Athmospheric Environment 33*, 9 (1996), 1481–1488.

[91] KLEBE, G. Virtual ligand screening : strategies, perspectives and limitations. *Drug Discov. Today 11*, 13-14 (2006), 580–594.

[92] KRAJECKI, M. An object oriented environment to manage the parallelism of the FIIT applications. In *Parallel Computing Technologies, 5th International Conference, PaCT-99*, V. Malyshkin, Ed., vol. 1662 of *Lecture Notes in Computer Science*. Springer-Verlag, St. Petersburg, Russia, Sept. 1999, pp. 229–234.

[93] KUMAR, K. A., KONISHETTY, V. K., VORUGANTI, K., AND RAO, G. V. P. Cash : context aware scheduler for hadoop. In *Proceedings of the International Conference on Advances in Computing, Communications and Informatics* (New York, NY, USA, 2012), ICACCI '12, pp. 52–61.

[94] LABARTA, J., GIRONA, S., PILLET, V., CORTES, T., AND GREGORIS, L. DiP : A parallel program development environment. In *Proceedings of the 2nd Euro-Par Conference* (1996), vol. 2, pp. 665–674.

[95] LAURO, G., MASULLO, M., PIACENTE, S., RICCIO, R., AND BIFULCO, G. Inverse virtual screening allows the discovery of the biological activity of natural compounds. *Bioorganic and Medicinal Chemistry 20*, 11 (2012), 3596–3602.

[96] LAURO, G., ROMANO, A., RICCIO, R., AND BIFULCO, G. Inverse virtual screening of antitumor targets : Pilot study on a small database of natural bioactive compounds. *Journal of Natural Products 74*, 6 (2011), 1401–1407.

[97] LI, H., GAO, Z., KANG, L., ZHANG, H., YANG, K., YU, K., LUO, X., ZHU, W., CHEN, K., SHEN, J., WANG, X., AND JIANG., H. Tarfisdock : a web server for identifying drug targets with docking approach. *Nucleic Acids Research 34*, web (2006), 219–224.

[98] LI, J., WANG, Q., JAYASINGHE, D., PARK, J., ZHU, T., AND PU, C. Performance overhead among three hypervisors : An experimental study using hadoop benchmarks. In *Big Data (BigData Congress), 2013 IEEE International Congress on* (June 2013), pp. 9–16.

[99] LIU, P., AND SHENG, T.-H. Broadcast scheduling optimization for heterogeneous cluster systems. In *Proceedings of 12th SPAA* (2000), pp. 129–136.

[100] MAAMAR, Z., BENSLIMANE, D., AND NARENDRA, N. C. What can context do for web services ? *Commun. ACM 49*, 12 (Dec. 2006), 98–103.

[101] MANNEY, G. L., ZUREK, R. W., O'NEILL, A., AND SWINBANK, R. On the motion of air through the stratospheric polar vortex. *Jounal of the Atmospheric Sciences 51* (Oct 1994), 2973–2994.

[102] MARCHAND, M., BEKKI, S., PAZMINO, A., LEFÈVRE, F., GODIN-BEEKMANN, S., AND HAUCHECORNE, A. Model simulations of the impact of the 2002 antarctic ozone hole on the midlatitudes. *Jounal of the Atmospheric Sciences 62* (Mar 2005), 871–884.

[103] MAROZZO, F., TALIA, D., AND TRUNFIO, P. P2p-mapreduce : Parallel data processing in dynamic cloud environments. *J. Comput. Syst. Sci. 78*, 5 (Sept. 2012), 1382–1402.

[104] MATEESCU, G. A method for MPI broadcast in computational grids. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2005)* (2005).

[105] MIORANDI, D., SICARI, S., PELLEGRINI, F. D., AND CHLAMTAC, I. Internet of things : Vision, applications and research challenges. *Ad Hoc Networks 10*, 7 (2012), 1497 – 1516.

[106] MOY, J. OSPF Version 2. RFC 2328 (INTERNET STANDARD), Apr. 1998. Updated by RFCs 5709, 6549, 6845, 6860.

[107] MUENCHEN, B., SIQUEIRA, T., NESI, L., KOSLOVSKI, G., CHARAO, A. S., KIRSCH-PINHEIRO, D., AND STEFFENEL, L. A. Análise de dados observacionais sobre a camada de ozônio : uma abordagem usando mpi e openmp. In *Fórum de iniciação científica Escola Regional de Alto Desempenho (ERAD/RS)* (Ijui, Brazil, April 2017).

[108] NAJAR, S., KIRSCH PINHEIRO, M., AND SOUVEYET, C. Service discovery and prediction on pervasive information system. *Journal of Ambient Intelligence and Humanized Computing 6*, 4 (2015), 407–423.

[109] NESI, L., KOSLOVSKI, G., CHARAO, A. S., KIRSCH-PINHEIRO, D., AND STEFFENEL, L. A. Paralelização de cálculos estatísticos sobre dados de monitoramento da camada de ozônio : um estudo com gpu. In *Fórum de iniciação científica Escola Regional de Alto Desempenho (ERAD/RS)* (Ijui, Brazil, April 2017).

[110] ORACLE. Overview of java se monitoring and management, 2014. `http://docs.oracle.com/javase/7/docs/technotes/guides/management/overview.html`, Last access : July 2014.

[111] ORAN, D. OSI IS-IS Intra-domain Routing Protocol. RFC 1142 (Informational), Feb. 1990.

[112] PARASHAR, M., AND PIERSON, J.-M. Pervasive grids : Challenges and opportunities. In *Handbook of Research on Scalable Computing Technologies*, K. Li, C. Hsu, L. Yang, J. Dongarra, and H. Zima, Eds. IGI Global, 2010, pp. 14–30.

[113] PERES, L. V. *Efeito Secundário do Buraco de Ozônio Antártico Sobre o Sul do Brasil*. Msc in Meteorology. Universidade Federal de Santa Maria, Brazil, 2013.

[114] PETERSON, J. L. *Petri Net Theory and Modeling of Systems*. Prentice-Hall, Englewood-Cliffs, New Jersey, 1981.

[115] PINHEIRO, D., LEME, N., PERES, L., AND KALL, E. *Influence of the Antarctic ozone hole over South of Brazil in 2008 and 2009*, vol. 1. National Institute of Science and Technology, 2011, pp. 33–37.

[116] PJESIVAC-GRBOVIC, J., ANGSKUN, T., BOSILCA, G., FAGG, G. E., GABRIEL, E., AND DONGARRA, J. J. Performance analysis of MPI collective operations. In *Proceedings of the Wokshop on Performance Modeling, Evaluation and Optimisation for Parallel and Distributed Systems (PMEO), in IPDPS 2005* (2005).

[117] PRATHER, M., AND JAFFE, A. H. Global impact of the antarctic ozone hole : Chemical propagation. *Journal of Geophysical Research : Atmospheres 95*, D4 (1990), 3473–3492.

[118] RAMAKRISHNAN, A., PREUVENEERS, D., AND BERBERS, Y. Enabling self-learning in dynamic and open IoT environments. In *The 5th International Conference on Ambient Systems, Networks and Technologies (ANT-2014), the 4th International Conference on Sustainable Energy Information Technology (SEIT-2014)* (2014), E. Shakshuki and A. Yasar, Eds., vol. 32, pp. 207–214.

[119] RASOOLI, A., AND DOWN, D. G. Coshh : A classification and optimization based scheduler for heterogeneous hadoop systems. In *Proceedings of the 2012 SC Companion : High Performance Computing, Networking Storage and Analysis* (Washington, DC, USA, 2012), SCC '12, IEEE Computer Society, pp. 1284–1291.

[120] RCBS. Rcsb protein data bank.

[121] REY, J., COGORNO, M., NESMACHNOW, S., AND STEFFENEL, L. A. Fast prototyping of map-reduce applications with docker-hadoop. In *Conférence d'informatique en Parallélisme, Architecture et Système (ComPAS'14)* (Neuchatel, Switzerland, April 2014).

[122] REY, J., COGORNO, M., NESMACHNOW, S., AND STEFFENEL, L. A. Efficient prototyping of fault-tolerant map-reduce applications with docker-hadoop. In *WoC : First International Workshop on Container Technologies and Container Clouds* (Tempe, AZ, USA, Mar 2015).

[123] ROTTENBERG, S., LERICHE, S., LECOCQ, C., AND TACONET, C. Vers une définition d'un système réparti multi-échelle. In *UBIMOB'12 - 8èmes Journées Francophones Mobilité et Ubiquité* (2012), pp. 178–183.

[124] ROTTENBERG, S., LERICHE, S., TACONET, C., LECOCQ, C., AND DESPRATS, T. Musca : A multiscale characterization framework for complex distributed systems. In *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems* (2014), pp. 1657–1665.

[125] SALBY, M. L., TITOVA, E. A., AND DESCHAMPS, L. Changes of the antarctic ozone hole : Controlling mechanisms, seasonal predictability, and evolution. *Journal of Geophysical Research : Atmospheres 117*, D10 (2012), n/a–n/a. D10111.

[126] SANDHOLM, T., AND LAI, K. Dynamic proportional share scheduling in hadoop. In *Proceedings of the 15th International Conference on Job Scheduling Strategies for Parallel Processing* (Berlin, Heidelberg, 2010), JSSPP'10, pp. 110–131.

[127] SATYANARAYANAN, M., BAHL, P., CACERES, R., AND DAVIES, N. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing 8*, 4 (Dec. 2009), 14–23.

[128] SEGALL, A. Distributed network protocols. *Transaction on Information Theory*, 29 (1983), 23–35.

[129] SEMANE, N., BENCHERIF, H., MOREL, B., HAUCHECORNE, A., AND DIAB, R. An unusual stratospheric ozone decrease in southern hemisphere subtropics linked to isentropic air-mass transport as observed over irene (25.5ž s, 28.1ž e) in mid-may 2002. *Atmospheric Chemistry and Physics 6* (2006), 1927–1936.

[130] SERRANO, M., LE-PHUOC, D., ZAREMBA, M., GALIS, A., BHIRI, S., AND HAUSWIRTH, M. Resource optimisation in iot cloud systems by using matchmaking and self-management principles. In *The Future Internet*, vol. 7858 of *LNCS*. Springer, 2013, pp. 127–140.

[131] SIVAKUMAR, V., PORTAFAIX, T., BENCHERIF, H., GODIN-BEEKMANN, S., AND BALDY, S. Stratospheric ozone climatology and variability over a southern subtropical site : Reunion island (21 °s ; 55 °e). *Annales Geophysicae 25* (2007), 2321–2334.

[132] SKAMAROCK, W. C., KLEMP, J. B., DUDHIA, J., GILL, D. O., BARKER, D. M., DUDA, M. G., HUANG, X.-Y., WANG, W., AND POWERS, J. G. A description of the advanced research wrf version 3. Tech. Note NCAR/TN-475+STR, NCAR, 2008.

[133] SOLOMON, S. Stratospheric ozone depletion : A review of concepts and history. *Reviews of Geophysics 37*, 3 (1999), 275–316.

[134] STEFFEN, A., THIELE, C., TIETZE, S., STRASSNIG, C., KÄMPER, A., LENGAUER, T., WENZ, G., AND APOSTOLAKIS, J. mproved cyclodextrin-based receptors for camptothecin by inverse virtual screening. *Chemistry - A European Journal 13*, 24 (2007), 6801–6809.

[135] STEFFENEL, L., FLAUZAC, O., CHARAO, A. S., BARCELOS, P. P., STEIN, B., NESMACHNOW, S., PINHEIRO, M. K., AND DIAZ, D. Per-mare : Adaptive deployment of mapreduce over pervasive grids. In *Proceedings of the 8th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC'13)* (Compiegne, France, Oct 2013).

[136] STEFFENEL, L., AND KIRSCH-PINHEIRO, M. Strong consistency for shared objects in pervasive grids. In *Proceedings of the 5th IEEE International Conference on Wireless and Mobile Computing, Networking and Communication (WiMob'2009)* (Marrakesh, Morroco, October 2009), pp. 73–78.

[137] STEFFENEL, L. A. Modeling network contention effects on alltoall operations. In *Proceedings of the IEEE Conference on Cluster Computing (CLUSTER 2006)* (Barcelona, Spain, Sept. 2006), IEEE Computer Society.

[138] STEFFENEL, L. A., FLAUZAC, O., CHARÃO, A. S., BARCELOS, P. P., STEIN, B., NESMACHNOW, S., KIRSCH PINHEIRO, M., AND DIAZ, D. Per-mare : Adaptive deployment of mapreduce over pervasive grids. In *Proceedings of the 2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing* (Washington, DC, USA, 2013), 3PGCIC '13, IEEE Computer Society, pp. 17–24.

[139] STEFFENEL, L. A., AND KIRSCH-PINHEIRO, M. CloudFIT, a PaaS platform for IoT applications over pervasive networks. In *3rd International Workshop on Cloud for IoT (CLIoT 2015)* (Taormina, Italy, Sept. 2015).

[140] STEFFENEL, L. A., AND KIRSCH PINHEIRO, M. Leveraging data intensive applications on a pervasive computing platform : The case of mapreduce. *Procedia Computer Science 52* (2015), 1034 – 1039. The 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015).

[141] STEFFENEL, L. A., AND KIRSCH-PINHEIRO, M. When the cloud goes pervasive : approaches for IoT PaaS on a mobiquitous world. In *EAI International Conference on Cloud, Networking for IoT systems (CN4IoT 2015)* (Rome, Italy, Oct. 2015).

[142] STEFFENEL, L. A., KIRSCH-PINHEIRO, M., AND BERBERS, Y. Total order broadcast on pervasive systems. In *Proceedings of the 23rd Annual ACM Symposium on Applied Computing (SAC 2008)* (Fortaleza, Brazil, Mar 2008), pp. 2202–2206.

[143] STIC-AMSUD. PER-MARE project, 2014. http://cosy.univ-reims.fr/PER-MARE, Last access : July 2014.

[144] TEAM, L.-M. Lam/mpi version 7. http ://www.lam-mpi.org, 2004.

[145] THAKUR, R., AND GROPP, W. Improving the performance of collective operations in MPICH. In *Proceedings of the Euro PVM/MPI 2003* (2003), LNCS Vol. 2840, Springer-Verlag, pp. 257–267.

[146] TIAN, C., ZHOU, H., HE, Y., AND ZHA, L. A dynamic mapreduce scheduler for heterogeneous workloads. In *Proceedings of the 2009 Eighth International Conference on Grid and Cooperative Computing* (Washington, DC, USA, 2009), GCC '09, IEEE Computer Society, pp. 218–224.

[147] TOIHIR, A. M., BENCHERIF, H., SIVAKUMAR, V., EL AMRAOUI, L., PORTAFAIX, T., AND MBATHA, N. Comparison of total column ozone obtained by the IASI-MetOp satellite with ground-based and OMI satellite observations in the southern tropics and subtropics. *Annales Geophysicae* (Sept. 2015).

[148] VADHIYAR, S., FAGG, G., AND DONGARRA, J. Automatically tuned collective communications. In *Proceedings of the Supercomputing 2000* (2000), IEEE Computer Society.

[149] VALIANT, L. G. A bridging model for parallel computation. *Communications of the ACM 33*, 8 (1990), 103–111.

[150] VASSEUR, R. *Développements HPC pour une nouvelle méthode de docking inverse : applications aux protéines matricielles*. PhD thesis, Université de Reims Champagne-Ardenne, Dec 2015.

[151] VERMESAN, O., FRIESS, P., GUILLEMIN, P., GIAFFREDA, R., GRINDVOLL, H., EISENHAUER, M., SERRANO, M., MOESSNER, K., SPIRITO, M., BLYSTAD, L.-C., AND TRAGOS, E. Z. Internet of things beyond the hype : Research, innovation and deployment. In *Internet of Things - From Research and Innovation to Market Deployment*. River Publishers, 2014.

[152] VILLARI, M., FAZIO, M., DUSTDAR, S., RANA, O., AND RANJAN, R. Osmotic computing : A new paradigm for edge/cloud integration. *IEEE Cloud Computing 3*, 6 (Nov 2016), 76–83.

[153] WANG, R., FANG, X., LU, Y., AND WANG, S. The pdbbind database : Collection of binding affinities for proteinligand complexes with known three-dimensional structures. *J. Med. Chem 47*, 12 (2004), 2977–2980.

[154] WANG, R., FANG, X., LU, Y., YANG, C.-Y., AND WANG, S. The pdbbind database : Methodologies and updates. *J. Med. Chem. 48*, 12 (2005), 4111–4119.

[155] WAUGH, D. W., PLUMB, R. A., ATKINSON, R. J., SCHOEBERL, M. R., LAIT, L. R., NEWMAN, P. A., LOEWENSTEIN, M., TOOHEY, D. W., AVALLONE, L. M., WEBSTER, C. R., AND MAY, R. D. Transport out of the lower stratospheric arctic vortex by rossby wave breaking. *Journal of Geophysical Research : Atmospheres 99*, D1 (1994), 1071–1088.

[156] WHITE, T. *Hadoop : The definitive guide*, 2nd edition ed. Yahoo Press, O'Reilly, 2010.

[157] WILLIS, D. F., DASGUPT, A., AND BANERJEE, S. Paradrop : a multi-tenant platform for dynamically installed third party services on home gate- ways. In *ACM SIGCOMM workshop on Distributed cloud computing* (2014).

[158] WOLSKI, R., SPRING, N., AND PETERSON, C. Implementing a performance forecasting system for metacomputing : The network weather service. In *Proceedings of the Supercomputing* (1997).

[159] WU, D., TIAN, Y., AND NG, K.-W. Aurelia : Building locality-preserving overlay network over heterogeneous p2p environments. In *Proc. of the International Conference on Parallel and Distributed Processing and Applications* (2005), ISPA'05, Springer, pp. 1–8.

[160] WU, J.-J., YEH, S.-H., AND LIU, P. Efficient multiple multicast on heterogeneous network of workstations. *Journal of Supercomputing 29*, 1 (2004), 59–88.

[161] XIE, J., RUAN, X., DING, Z., TIAN, Y., MAJORS, J., MANZANARES, A., YIN, S., AND QIN, X. Improving mapreduce performance through data placement in heterogeneous hadoop clusters. In *Parallel and Distributed Processing, Workshops and Phd Forum (IPDPSW)* (2010).

[162] YI, S., HAO, Z., QIN, Z., AND LI, Q. Fog computing : Platform and applications. In *Third IEEE Workshop on Hot Topics in Web Systems and Technologies* (2015), IEEE, Ed., pp. 73–78.

[163] YOO, A. B., JETTE, M. A., AND GRONDONA, M. Slurm : Simple linux utility for resource management. In *Workshop on Job Scheduling Strategies for Parallel Processing* (2003), pp. 44–60.

[164] ZAHARIA, M., KONWINSKI, A., JOSEPH, A. D., KATZ, R., AND STOICA, I. Improving mapreduce performance in heterogeneous environments. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation* (Berkeley, CA, USA, 2008), OSDI'08, USENIX Association, pp. 29–42.

[165] ZAO, J. K., GAN, T.-T., YOU, C.-K., CHUNG, C.-E., WANG, Y.-T., RODRÍGUEZ MÉNDEZ, S. J., MULLEN, T., YU, C., KOTHE, C., HSIAO, C.-T., CHU, S.-L., SHIEH, C.-K., AND JUNG, T.-P. Pervasive brain monitoring and data sharing based on multi-tier distributed computing and linked data technology. *Frontiers in Human Neuroscience 8* (2014), 370.