

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 :

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

à Manuele.

Remerciements

Je tiens à exprimer mes remerciements et toute ma gratitude à :

Je remercie les étudiants qui m'ont fait confiance pour

Table des matières

| | |
|---|-----------|
| Introduction | 9 |
| 1 L'Hétérogénéité des Communications | 11 |
| 1 Modélisation des Performances d'un Réseau | 11 |
| 1.1 Exécution dans les systèmes distribués | 12 |
| 1.2 Modélisation des Communications | 13 |
| 2 Modélisation de l'Opération de Diffusion Broadcast | 15 |
| 2.1 Modélisation d'un Broadcast dans un réseaux homogène | 16 |
| 2.2 Modélisation du Broadcast dans une Grille | 18 |
| 2.2.1 Heuristiques Traditionnelles | 19 |
| 2.2.2 Heuristiques "grid-aware" | 21 |
| 2.3 Évaluation pratique | 23 |
| 3 Amélioration de la performance de MPI_AlltoAll sur un grid | 26 |
| 3.1 Définitions | 26 |
| 3.2 Modélisation de la congestion du réseau | 26 |
| 3.3 Modélisation de la congestion dans un cluster homogène | 27 |
| 3.4 L'algorithme LG | 28 |
| 4 Bilan et Perspectives | 30 |
| 2 L'hétérogénéité des Données | 33 |
| 1 L'Hétérogénéité des Données - la grille GRAPP&S | 33 |
| 1.1 Introduction | 33 |
| 2 L'Architecture GRAPP&S | 34 |
| 2.1 Définitions | 34 |
| 2.2 Communication et les réseaux overlay | 34 |
| 2.3 Éléments de l'Architecture GRAPP&S | 35 |
| 3 Gestion de la Communauté | 36 |
| 3.1 Gestion des Nœuds | 37 |
| 3.2 Algorithmes d'élection | 38 |
| 4 Opérations dans GRAPP&S | 39 |
| 4.1 Stockage et Indexation | 39 |
| 4.2 Recherche | 40 |
| 5 Bilan et Perspectives | 41 |
| 3 L'hétérogénéité des Ressources de Calcul | 43 |
| 1 Hétérogénéité des Tâches - application à la recherche en amarrage moléculaire | 44 |
| 1.1 Travaux proches et méthodologie de parallélisation | 44 |
| 1.2 Parallélisme Intérieur : décomposition des tâches | 45 |

| | | |
|-----------------------------------|---|-----------|
| 1.2.1 | Décomposition géométrique arbitraire | 46 |
| 1.2.2 | Décomposition géométrique avec superposition | 46 |
| 1.2.3 | Recherche de cavités | 46 |
| 1.3 | Gestion et déploiement des tâches de calcul | 47 |
| 1.3.1 | Optimisation aux clusters HPC | 48 |
| 1.4 | Évaluation pratique | 48 |
| 2 | Hétérogénéité des Ressources de Calcul - adaptation de Apache Hadoop aux grilles pervasives | 51 |
| 2.1 | Architecture et Ordonnancement dans Hadoop | 51 |
| 2.2 | État de l'Art | 53 |
| 2.3 | Ordonnancement Orienté par le Contexte | 54 |
| 2.3.1 | Le collecteur de contexte | 55 |
| 2.3.2 | Communication | 56 |
| 2.4 | Évaluation Pratique | 57 |
| 2.4.1 | Benchmarks et environnement de test | 57 |
| 2.4.2 | Résultats | 58 |
| 3 | Bilan et Perspectives | 60 |
| 4 | CloudFIT : un intergiciel pour le <i>Fog Computing</i> et l'<i>Internet des Objets</i> | 61 |
| 1 | Introduction | 61 |
| 2 | État de l'art | 62 |
| 3 | De CONFIIT à CloudFIT | 64 |
| 3.1 | Spécification des Besoins | 66 |
| 3.2 | Architecture | 66 |
| 3.3 | Communication | 69 |
| 4 | Calcul Multi-échelle et le Fog | 70 |
| 4.1 | Clustering | 71 |
| 4.2 | Optimisations pour le big data | 71 |
| 5 | Exemples d'Utilisation de CloudFIT | 74 |
| 5.1 | L'application WordCount | 74 |
| 5.1.1 | Impact de la volatilité | 75 |
| 5.1.2 | Impact de l'hétérogénéité | 75 |
| 5.2 | Détection d'Événements Sécondaires de la Couche d'Ozone | 77 |
| 5.2.1 | Identification des Événements Secondaires de l'Ozone avec CloudFIT | 78 |
| 5.2.2 | Pré-traitement des données | 79 |
| 5.2.3 | Analyse des séries temporelles et la détection des OSE | 80 |
| 5.2.4 | Résultats Préliminaires | 81 |
| 6 | Bilan et Perspectives | 82 |
| Conclusion et perspectives | | 83 |
| Publications personnelles | | 85 |
| Bibliographie | | 93 |

Introduction

La définition simple de l'hétérogénéité ("Manque d'unité, composé d'éléments de nature diverse", Dictionnaire Larousse) n'est pas suffisamment développée pour qualifier les différents défis liés à l'hétérogénéité dans les systèmes distribués. Pour cela, nous devons étendre et catégoriser les différents mécanismes liés à l'hétérogénéité.

Une première catégorie est issue directement de la définition simple ci-dessus, et 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 un système, notamment leur composition matérielle (processeurs, mémoire) et leur capacité de calcul. Ainsi, un système distribué composé d'équipements identiques (caractérisé par l'absence d'hétérogénéité matérielle) peut supposer un traitement uniforme des tâches de calcul, traitement de données et donc simplifier la gestion des applications qu'y tournent. Au contraire, un système composé par des équipements hétérogènes, même partiellement, doit être conscient de cette différence et prévoir des mécanismes pour garantir l'exécution des applications : ordonnancement sensible aux capacités des équipements, synchronisation des tâches dépendantes, équilibrage et migration de charge, etc. Même étant très réductrice, cette catégorie est souvent utilisée pour qualifier des équipements : machines parallèles (symétriques), clusters, grilles de calcul, réseaux P2P, etc.

À l'hétérogénéité matérielle s'ajoute le problème de l'hétérogénéité des communications, qui peut être causé autant par la diversité matérielle (par exemple, en utilisant différents types de réseaux) ou par la distance géographique (qui impacte les temps de communication). On retrouve l'hétérogénéité des communications surtout dans les systèmes distribués à grande échelle (grilles, réseaux P2P) où souvent le temps de communication est 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, ordonnancement basé sur la topologie du réseau, recouvrement des communications par de calculs pouvant être effectués en parallèle, etc.

La diversité matérielle et la diversité des communications (matérielle ou spatiale) constituent les gros facteurs liés à l'hétérogénéité à un moment donné. Toutefois, ce ne sont pas les seuls éléments impactant une application. En effet, nous devons assurer le fonctionnement d'un système distribué pendant toute la durée de l'exécution de l'application, et cette hétérogénéité temporelle est issu de la dynamicité d'opération d'un tel système. Plus exactement, les systèmes distribués peuvent être impactés par le départ ou l'arrivée de ressources, faisant ainsi la prise en compte de ces facteurs un besoin essentiel pour le bon fonctionnement d'un système et 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 temporelle affecte aussi l'uti-

lisation des ressources, car même sans une défaillance un système doit pouvoir être amené à gérer plusieurs tâches indépendantes, chacune avec des besoins propres.

Chapitre 1

L'Hétérogénéité des Communications

Résumé

Dans l'ensemble des éléments qui composent les systèmes informatiques, le réseau d'interconnexion est parmi les éléments qui sont les plus exposés et impactés par l'hétérogénéité. Cette hétérogénéité peut se présenter sous différentes formes : diversité d'équipements et de technologies, variations de performance et 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 applications, il s'avère souvent que les solutions se limitent à pallier les éventuels problèmes. Les travaux que j'ai mené dans le domaine de l'hétérogénéité des réseaux visent la compréhension des facteurs liés à l'hétérogénéité, leur caractérisation et aussi l'optimisation de l'usage des ressources afin de rendre les systèmes et applications plus performants et fiables.

L'une des premières étapes dans l'étude de l'hétérogénéité requiert l'observation et l'analyse des communications réseau afin de modéliser leur fonctionnement et ainsi permettre une estimation de leurs performances. Cette étude cache néanmoins une grande complexité car même sur un petit ensemble de ressources la variation de certains facteurs peut créer un nombre important de situations, chacune avec un modèle de communication différent. Afin de permettre l'application pratique de cette modélisation de performance nous devons non seulement être en mesure de découvrir la topologie du réseau, mais aussi de pouvoir quantifier les différents facteurs et émettre des hypothèses sur les modèles de communication observés.

Un mécanisme très utile dans la réduction de la complexité de la modélisation des communications réside dans la classification et catégorisation des ressources observées. Non seulement cela permet de se concentrer sur des modèles "type" qui représentent convenablement des ensembles de ressources, comme aussi on ouvre la possibilité d'utiliser ces informations dans le but d'optimiser les échanges entre les applications.

Tous ces éléments sont connectés aux travaux qui seront présentés dans les chapitres suivants.

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 la performance de ces algorithmes. 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 [54]. 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 [99], 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 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 Exécution dans les systèmes distribués

Un système distribué est classiquement 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. Les nœuds du graphe représentent les sites du réseau, les arêtes du graphe les liens, ou canaux de communication du réseau.

Par la suite, on parlera indifféremment de nœud, de site, de processus ou de processeur pour définir un nœud du réseau. De même, on parlera indifféremment de canaux ou de liens de communication pour définir les supports de la communication qu'utilise le réseau.

L'écriture d'un algorithme, ainsi que l'évaluation de ses performances, ne peut se faire que si l'on a, au préalable, défini un modèle d'exécution. La modélisation sert à imposer les contraintes permettant de se rapprocher de la réalité. Les algorithmes distribués sont écrits selon trois modèles principaux : le modèle à états, le modèle à registres et le modèle à passage de messages.

Le modèle à états a été défini par Dijkstra [41]. Les sites ne communiquent pas par l'intermédiaire d'échanges de messages, mais par leur capacité à lire l'état de la mémoire de leurs voisins. Un site accède en lecture et en écriture à l'ensemble de son environnement et aux variables qu'il détient, mais il accède uniquement en lecture à l'environnement de ses voisins.

Le modèle à registres partagés représente les liens de communication comme des registres attachés à chaque site. Un lien l_{ij} reliant les sites i et j "porte" donc un registre R_{ij} dans lequel le site i écrit les informations à transmettre au site j . De même, il existe un registre R_{ji} utilisé par le site j pour transmettre des informations au site j .

Le modèle à passage de messages définit des canaux de communication par lesquels transitent les messages. Il nécessite de poser des hypothèses sur les propriétés des canaux de communication : taille, délais de transmission, caractéristiques des échanges...

Les différents modèles permettent une approche progressive des communications dans les systèmes distribués. Lynch [86] propose toutefois des algorithmes et des méthodes permettant de transformer un algorithme écrit pour un modèle à registres partagés en un algorithme exploitant le modèle à passage de messages.

Dans notre cas spécifique, nous voulons explorer le modèle à passage de messages, étudiant notamment les modèles de performance qui peuvent apporter à la fois une représentation fidèle du comportement des communications mais aussi une utilisation simple, capable d'être appliquée sur des environnements hétérogènes.

1.2 Modélisation des Communications

Malgré le développement de processeurs et de réseaux plus efficaces, la communication efficace entre ces deux éléments a toujours été un obstacle à l'obtention de meilleures performances. En effet, l'augmentation du nombre de processeurs ne permet pas nécessairement que le temps d'une application soit réduit proportionnellement : en dehors des limitations dues au grain des tâches, le coût de communication entre les différents processeurs est l'un des plus importants obstacles.

Le modèle de Hockney [62] 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 noeuds 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 [101]. L'obtention des paramètres α et β peut se faire à travers l'utilisation de certains outils comme NWS [139].

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 processus, 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 processus pour envoyer un message (la latence d'envoi), alors que t_u représente le temps total nécessaire à la réception du message par le processus destinataire. Une des innovations du modèle Postal par rapport à ses prédecesseurs est que ce modèle permet des situations où un processus peut émettre plusieurs messages avant que le premier récepteur a reçu son message : cela est dû à une analogie avec l'envoi de lettres par la poste.

Dans une tentative de spécifier un modèle de calcul plus réaliste, Culler *et al.* [37] ont proposé le modèle de coût LogP afin de permettre la spécification du surcoût 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 grappes d'ordinateurs.

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 processus sont les suivants :

L - représente la latence de communication entre deux processus distincts,

o - le surcoût d'initialisation associée à l'envoi/réception d'un message,

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

P - le nombre de processus.

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 processus est de $\lceil L/g \rceil$. La latence maximale d'un réseau est la distance moyenne asymptotique entre les processus ; 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.* [73]. Son objectif est de mieux représenter à la fois des petits et des grands messages sous un modèle unifié.

Comme ses prédecesseurs, le modèle pLogP est défini à travers des paramètres qui représentent la latence, les surcoûts d'initialisation, le *gap* et le nombre de processus. La première différence est l'utilisation de valeurs distinctes, o_r et o_s , pour représenter le surcoût d'envoi et le surcoût de réception d'un message. Toutefois, la principale caractéristique du modèle pLogP est que les paramètres qui représentent le *gap* et les surcoûts sont paramétrés selon la taille du message 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 la latence et du *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 processus, 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 minimale entre deux transmissions ou réceptions consécutives, ce qui implique que $g(m) \geq o_s(m)$ et $g(m) \geq o_r(m)$ tient toujours. La relation entre ces différents paramètres est illustrée dans la Figure 1.1.

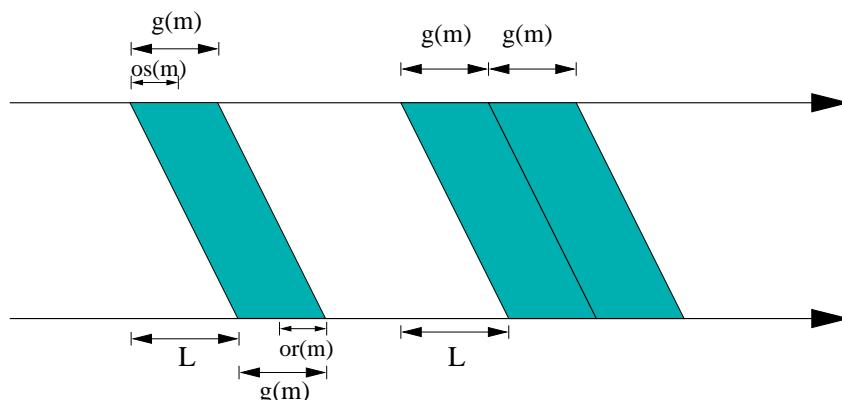


FIGURE 1.1 : Représentation d'une communication avec pLogP

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 noeuds avec des primitives de communication bloquante, 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 [127]. 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.

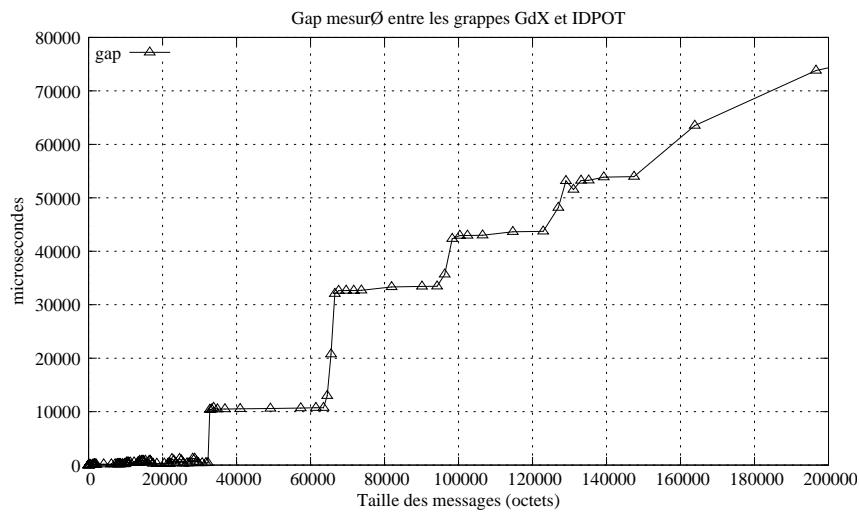


FIGURE 1.2 : Valeurs de gap mesurés entre deux machines distantes

2 Modélisation de l'Opération de Diffusion Broadcast

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 seul processus, appelé *racine*, envoie le même message de taille m à tous les autres ($P - 1$) processus. 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 données à tous les noeuds. La compréhension de ses mécanismes et la modélisation de ses coûts présente un intérêt stratégique vis-à-vis de la scalabilité et 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 hétérogènes. Afin de représenter ces deux cas, nous nous sommes intéressés à la modélisation et optimisation des communications collectives de type Broadcast dans deux types distincts de réseaux : les clusters (ou grappes d'ordinateurs) et les grids (grilles de calcul). Le premier cas souvent peut 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 ces stratégies, nous avons choisi d'utiliser comme point de départ la bibliothèque de passage de messages 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.

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 processus *racine* détient le message qui doit être diffusé ; à la fin de l'opération, une copie de ce message est déposée dans chaque processus 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 noeud 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 [14]. 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 $d, h \in [1 \dots P-1]$ tel que $\sum_{i=0}^h d^i \geq P$ soit respecté.

La performance des différentes formes fixes dépend surtout des paramètres du réseau, notamment le *gap*, la latence et le nombre de noeuds. 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 processus envoie des messages à tous les autres.

À partir des modèles de coût LogP [37] et pLogP [73] et de travaux comme ceux de Huse [65], Vadhiyar [131] 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 | $L + (P - 1) \times g(m)$ |
| Arbre Plat Rendez-vous | $3 \times L + (P - 1) \times g(m) + 2 \times g(1)$ |
| 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 [10]. Cette stratégie, possible grâce à la segmentation des messages, présente des avantages importants, comme l'indiquent [73][128][11]. 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*.

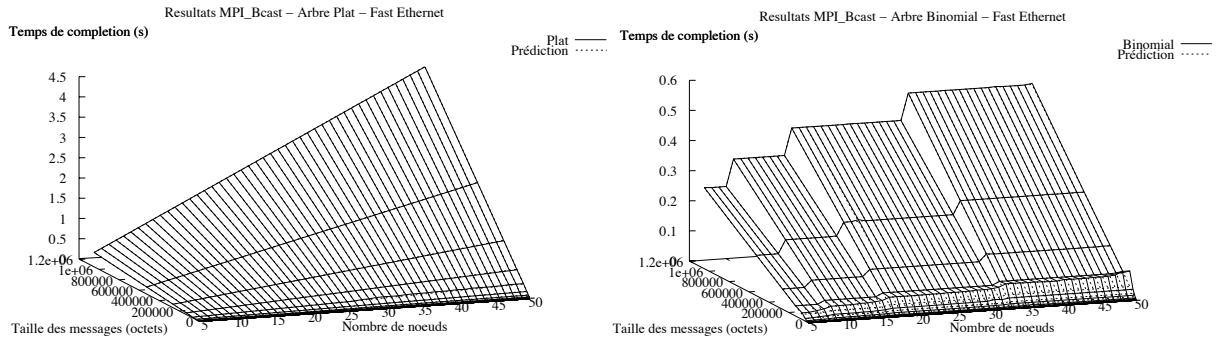
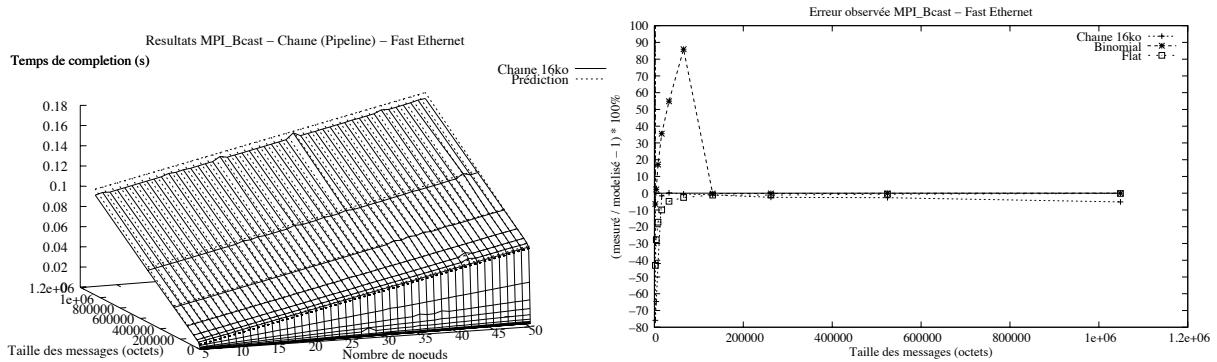
Dans ce cas, nous considérons que le segment de taille s d'un message m est un multiple de la taille du type basique de données qui est transmis, divisant alors le message initial m en k segments. Par conséquent, $g(s)$ représente le *gap* d'un segment de taille s . Toutefois, le choix de la taille des segments reste dépendant des caractéristiques du réseau. En effet, l'utilisation de segments trop petits a un surcoût 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 $i \in [0 \dots \log_2 m]$. Ensuite, on peut affiner la recherche de la taille optimale avec l'aide d'heuristiques comme le « *local hill-climbing* » [73].

| 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 [128] | $(\log_2 P + P - 1) \times L + 2 \times (\frac{p-1}{p}) \times g(m)$ |

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 [8] et [9]) 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 surcoût 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.

2.2 Modélisation du Broadcast dans une Grille

La détermination du meilleur arbre de diffusion pour un environnement homogène est une tâche relativement facile car dépend de paramètres de latence et débit (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 paire de noeuds. En effet, l'identification du meilleur arbre de diffusion dans un réseau hétérogène est un problème NP-complet [15][12, 13][141]. 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 paire de processus concernée par le broadcast. C'est le cas de Banikazemi [6], Bhat [15, 16] ou Mateescu [91].

Cependant, l'environnement de type grille est normalement caractérisé par un grand nombre de processus communicants, résultat de l'association des différentes grappes de calcul. 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 processus selon leurs performances relatives (par exemple, par rapport à la communication), de manière à ce que toute une classe de processus puisse être traitée comme une entité unique. De cette manière, le nombre important de processus dans une grille de calcul peut être encore facilement abordable par les méthodes d'optimisation classiques grâce à la division des communications en deux catégories, l'*inter-grappes* et l'*intra-grappes*.

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 lentes que celles à l'intérieur des réseaux locaux, et la réduction des communications inter-grappes 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 noeuds, 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 grappes regroupent des centaines de noeuds. Or, plus le coût de diffusion « intra-grappes » 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 grilles de calcul » (« *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 grappes.

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 grappes 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 grappes.

Formalisme utilisé

Pour décrire les heuristiques présentées dans cette section, nous utilisons un formalisme de groupes similaire à celui de Bhat [16]. Dans ce formalisme, les grappes sont séparées en deux groupes, **A** et **B**. Le groupe **A** contient les grappes qui ont déjà reçu le message (la réception du message par le coordinateur de la grappe est suffisante). Le groupe **B** contient les grappes qui devront recevoir le message. De cette manière, le groupe **A** contient initialement la grappe du processus *source* ou *racine*, tandis que le groupe **B** contient toutes les autres grappes 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 grappes (plus exactement, leurs coordinateurs), la grappe réceptrice est transférée au groupe **A**.

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

2.2.1 Heuristiques Traditionnelles

Diffusion en Arbre Plat (Flat)

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

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

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

Fastest Node First - FNF

L'heuristique *Fastest Node First* (le noeud 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 noeuds P . À chaque noeud 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 noeuds.

L'heuristique proposée par Banikazemi *et al.* nécessite $P - 1$ itérations, où à chaque étape l'heuristique définit 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 processus 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 [85], 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 grilles, 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 [16].

Fastest Edge First - FEF

Proposée par Bhat *et al.* [16], 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 noeuds différents (les arêtes), correspondants à la latence de communication entre les deux processus.

Pour identifier l'ordonnancement des communications nécessaire à l'exécution de l'opération de Broadcast, l'algorithme FEF, ordonne les processus 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 processus 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 processus. 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 processus

émetteur (*Ready Time* en anglais) est alors utilisée conjointement avec le temps nécessaire à la transmission du message entre les processus (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 processus qui peuvent *effectivement* transmettre les messages aux autres processus.

Early Completion Edge First with lookahead - ECEF-LA

Pour augmenter l'efficacité de l'heuristique précédente, la dernière heuristique proposée par Bhat *et al.* [16] 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 processus dans le groupe **B**, et la paire 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 [16] propose, par exemple, le coût minimal pour que le processus j transmette à d'autres processus 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 processus 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 processus 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**.

2.2.2 Heuristiques "grid-aware"

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

Comme nous l'avons déjà exposé, le coût d'une communication hiérarchique ne dépend pas seulement des latences entre les différentes grappes, mais aussi du temps nécessaire à la diffusion des messages à l'intérieur de ces grappes. Ce coût de diffusion intra-grappes devient encore plus important avec l'augmentation du nombre de noeuds à l'intérieur des grappes, 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 noeuds 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-grappes, appelé T_k , correspond aux prédictions des modèles de communication vus précédemment. Cette notation T_k est équivalente à une notation où un noeud fictif k' est associé à chaque coordinateur k , dont :

$$L_{k,k'} + g_{k,k'} = \begin{cases} T_k & \text{si } k' \text{ est associé à } k \\ \infty & \text{pour tout autre processus } j \neq k \end{cases}$$

La présence d'un noeud 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 une grappe 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 grilles de calcul », 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-grappes est pris en compte. Cette version sert de comparaison par rapport aux heuristiques de Bhat, présentées précédemment.

ECEF-LAt

L'heuristique ECEF-LAt 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-grappes T_k .

Ainsi, l'heuristique ECEF-LAt cherche à minimiser le coût total de transmission et le temps nécessaire à la diffusion d'un message dans une grappe distante (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 grappes qui peuvent retransmettre le message le plus vite possible à d'autres grappes. 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 grappes contactées dans le futur.

ECEF-LAT

Une contrepartie de la technique précédente est que cette stratégie a tendance à favoriser les grappes rapides, ce qui peut entraîner des retards supplémentaires aux grappes plus lentes, reléguées aux dernières places. Pour éviter une telle situation, nous proposons une nouvelle fonction de *lookahead*, où le choix des grappes 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 différentes grappes, lentes ou rapides. En effet, nous cherchons dans un premier instant la grappe 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 grappes les plus distantes ou les plus lentes (dans le sens où la diffusion des messages prend plus de temps) sont contactées 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-LAT, nous choisissons comme récepteur la grappe qui prendra le moins de temps possible pour contacter la grappe la plus lente : cela garantit que si besoin est, les grappes les plus lentes seront contactées 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 grappes les plus rapides. Cependant, nous considérons que le temps d'exécution d'un Broadcast hiérarchique dépend surtout des grappes les plus lentes.

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-LAT considère que parfois il est plus intéressant d'envoyer les messages d'abord aux grappes les plus lentes, 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 grappes plus lentes 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 grappes qui pourront contacter le plus rapidement possible la grappe la plus lente 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 grappes les plus lentes soient contactées dans le plus petit temps possible, ce qui peut minimiser le retard imputé à ces grappes. Toutefois, cette technique n'offre aucune garantie sur l'efficacité future des grappes émettrices. Pour cela, il serait peut-être intéressant d'ajouter une fonction de *lookahead*, à l'exemple des heuristiques de type ECEF-LA.

2.3 Évaluation pratique

Les différentes heuristiques ont été implémentées sur une version modifiée de la bibliothèque MagPIe, que nous avons adapté pour l'acquisition et la manipulation des paramètres de communication entre les grappes. Cette procédure de découverte de topologie permet non

seulement le regroupement des noeuds 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 sites d'Orsay, Toulouse et Grenoble (IDPOT). Nous avons utilisé 60 machines de la grappe GdX d'Orsay, 20 machines de la grappe de Toulouse et 8 machines de la grappe IDPOT. 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.

TABLE 1.3 : Latence entre les différents sites (en microsecondes)

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

* cette grappe contient une seule machine.

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 grappes et du processus racine.

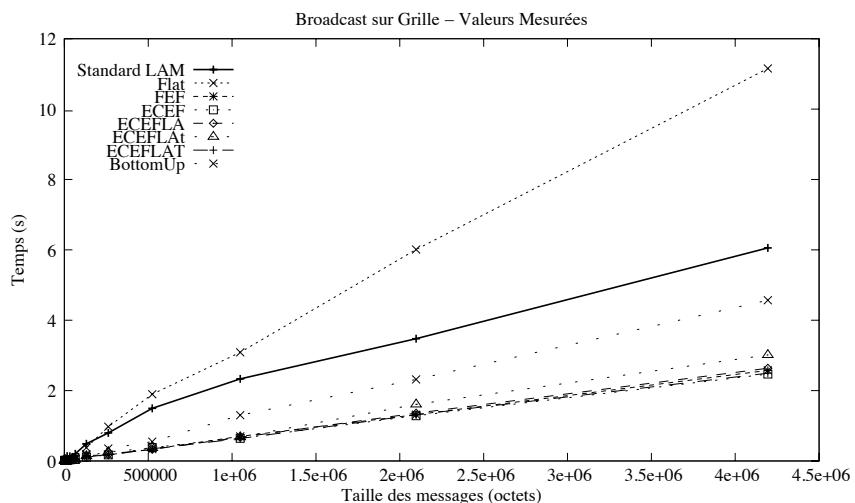


FIGURE 1.5 : Performance du Broadcast sur une grille de 88 machines

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.

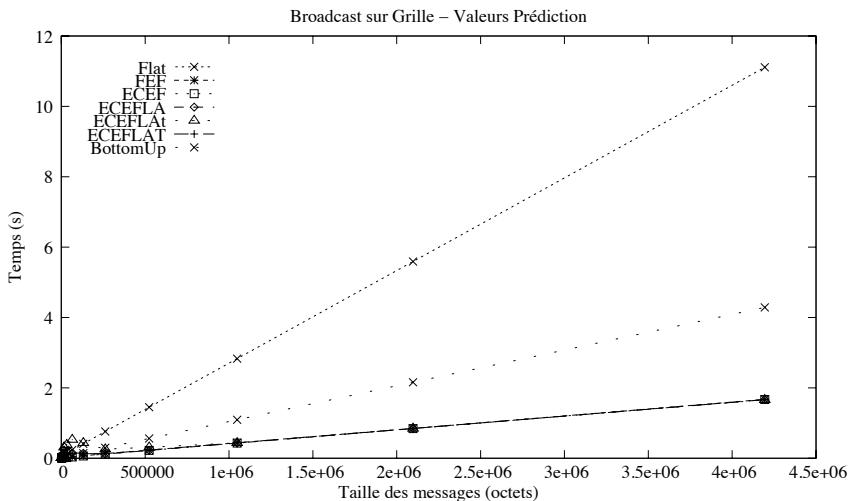


FIGURE 1.6 : Prédictions pour une grille avec 88 machines

Le faible écart observé entre les prédictions des heuristiques de type FEF et ECEF-* est justifié surtout par le nombre réduit de grappes, 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 oeuvre 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 processus racine et de la répartition des processus sur des différentes grappes sur la performance des stratégies plus simples. En effet, la performance de la stratégie Flat est fortement liée à l'ordre de représentation des grappes répartition, généralement fournie par l'utilisateur. De surcroît, la stratégie Flat utilise toujours le même ordre de diffusion, indépendamment du rang du processus racine. Au contraire, les heuristiques les plus élaborées construisent 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 processus.

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

3 Amélioration de la performance de MPI_AlltoAll sur 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 utilisées par les applications scientifiques. L'une de ces patrons, le *many-to-many*, représente un *échange total* d'informations entre les noeuds [32]. Plus exactement, chaque processus détient n items de données différentes de taille m , lesquels doivent être distribués entre n processus (le processus inclus).

Vu la complexité de l'opération, les implémentations MPI de All-to-All généralement utilisent des envois directs entre les processus, 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 de calcul doit aussi faire face à l'hétérogénéité des temps de communication entre les noeuds proches et distants.

3.1 Définitions

Soit deux clusters \mathcal{C}_1 et \mathcal{C}_2 avec respectivement n_1 noeuds et n_2 noeuds. 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 librairies (OpenMPI, MPICH2, etc.) implémentent l'opération *MPI_AlltoAll* en supposant que les noeuds se trouvent dans le même réseau et donc que les communications ont le même coût, dans notre cas certains messages sont envoyés entre noeuds d'un même cluster et d'autres entre noeuds de réseaux différents. Dans le premier cas, la latence des communications est plus favorables que dans le deuxième, et souvent le même principe s'applique pour le débit du réseau.

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 [33] 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 [32] ou [101] 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* [71]. Cela s'exprime par le modèle de performances linéaire représenté en Equation 1.1, où α correspond à la latence de communication entre les processus, $\frac{1}{\beta}$ correspond au débit du lien, m représente la taille des messages en octets et n correspond au nombre de processus :

$$T = (n - 1) \times (\alpha + \beta m) \quad (1.1)$$

Comme ce modèle linéaire n'est pas capable de prendre en compte la congestion des communications, certains auteurs tels que Bruck [20] et Clement *et al.* [36] suggèrent l'utilisation d'un facteur de ralentissement (*slowdown factor*) proportionnel au nombre de processus communicants. Labarta *et al.* [80] 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 [33], 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 ceci est directement lié au phénomène de la congestion.

3.3 Modélisation de la congestion dans un cluster homogène

Afin de mieux comprendre l'impact de la congestion sur les communications collectives de type All-to-All dans les clusters, nous avons proposé en [120] 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.* [36] avec un nouveau paramètre δ , ce dernier dépendant du nombre de processus 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 donnée, comme illustré en Figure 1.7.

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

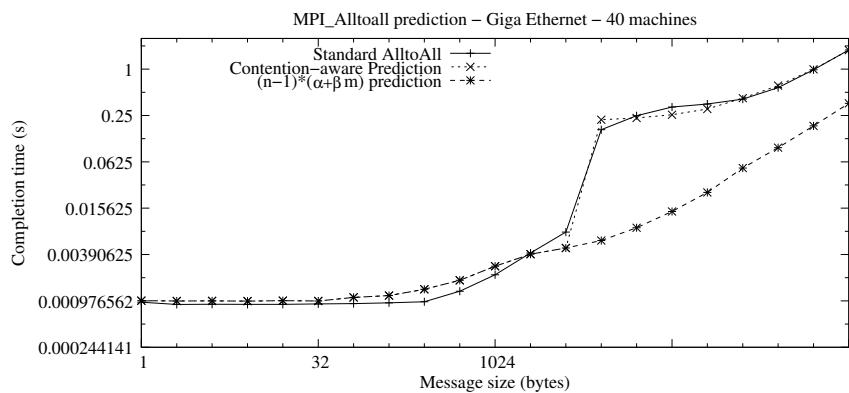


FIGURE 1.7 : Measured and predicted performance for the standard MPI_Alltoall in a Gigabit Ethernet network

Vu la précision de ce modèle pour les clusters homogènes, notre première réaction serait de l'appliquer aussi dans le cas des grilles 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.

$$T = \max(T_{C_1}, T_{C_2}) + \max(n_1, n_2) \times (\alpha_w + \beta_w \times m) \quad (1.3)$$

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* [38].

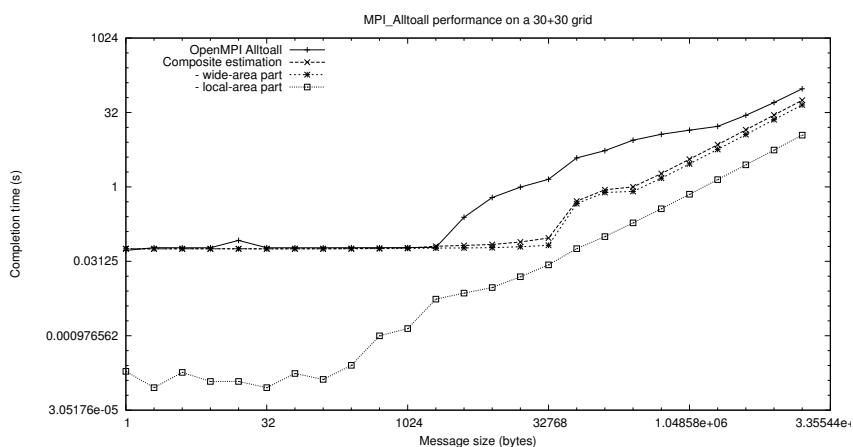


FIGURE 1.8 : Measured and predicted performance for the standard MPI_Alltoall in a grid

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 une grille.

3.4 L'algorithme LG

Lors de l'opération dans une grille de calcul, 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 grilles (PACX MPI [49], MagPIe [73]) 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 [22].

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 noeuds,

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 [69] 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 noeuds 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 noeuds 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 noeuds que le cluster \mathcal{C}_2 ($n_1 \leq n_2$). Les noeuds sont ainsi numérotés de 0 à $n_1 + n_2 - 1$, avec les noeuds allant de 0 à $n_1 - 1$ appartenant à \mathcal{C}_1 et les noeuds 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 noeud i au noeud j .

L'algorithme suit les deux étapes :

Première étape Pendant cette phase, nous effectuons les échanges locaux : Le processus i envoie $\mathcal{M}_{i,j}$ au processus 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 noeud distant j sur \mathcal{C}_2 seront d'abord stockés sur le noeud $j \bmod n_1$ de \mathcal{C}_1 . De même, les données sur un noeud i de \mathcal{C}_2 à destination de j sur \mathcal{C}_1 seront stockés sur le noeud $[i/n_1] \times n_1 + j$.

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 processus i de \mathcal{C}_1 échangera son buffer local avec le processus $j = i + n_1 \times s$ 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 noeuds, 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{LG} . Nous observons que l'algorithme \mathcal{LG} obtient un gain de performance de presque 50% par rapport à la stratégie traditionnelle.

L'ordonnancement des communications effectué par \mathcal{LG} 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 (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 :

$$T = \max(T_{C_1}, T_{C_2}) + \lceil n_2/n_1 \rceil \times (\alpha_w + \beta_w \times m \times n_1) \quad (1.4)$$

Afin d'obtenir les paramètres nécessaires aux prédictions, nous avons utilisé la procédure décrite par Kielman *et al.* [74]. Les facteurs de contention $\gamma = 2.6887$ et $\delta = 0.005039$ pour

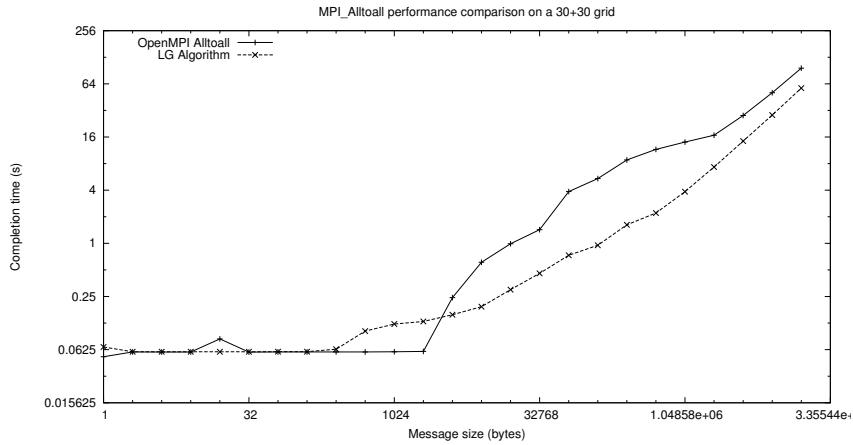


FIGURE 1.9 : Comparaison de performance entre OpenMPI et \mathcal{LG} algorithmes

$M \geq 1KB$ ont été obtenus par la méthode des moindres carrés comme décrit par [120].

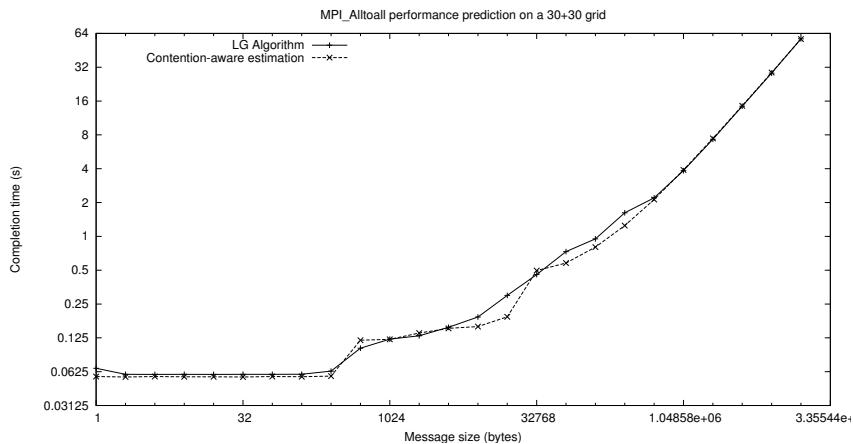


FIGURE 1.10 : Performance de \mathcal{LG} et sa prédiction

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

4 Bilan et Perspectives

Dans ce chapitre nous avons présenté une stratégie efficace pour identifier et découper les réseaux en grappes logiques homogènes. La présence des hétérogénéités réduit la précision des modèles de communication utilisés pour optimiser les communications collectives. Le *framework* proposé réussi à obtenir des paramètres de connectivité par un coût très faible, à partir du regroupement d’informations obtenues indépendamment dans chaque grappe. Notre approche, validée par des tests pratiques, démontre que la découverte de topologie peut se faire de façon rapide et précise. Associé à des modèles de prédiction de performance, le découpage des grappes s’avère essentiel à l’optimisation des primitives de communication collective, spécialement pour celles structurées en multiples couches. De plus, ce *framework* peut être étendu, de

manière à détecter aussi la présence de machines multiprocesseur, information importante pour l'optimisation des communications.

Chapitre 2

L'hétérogénéité des Données

Résumé

1 L'Hétérogénéité des Données - la grille GRAPP&S

1.1 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 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 en nuage (*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'agrégier et de présenter les données (cela 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 NAS/SAN ou cloud est devenue aussi courante que l'utilisation de disques ou clés USB, une fois que les ressources potentiellement illimités offerts par les réseaux P2P ou le 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 plate-forme 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.

2 L'Architecture GRAPP&S

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 noeuds du système et E désigne l'ensemble des liens de communications qui existent entre les noeuds. Le modèle utilisé pour notre système est étudié dans [27]. Deux noeuds 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 noeuds 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 noeud qui envoie le message, $id(v)$ est l'identifiant du noeud de réception, m' indique le contenu du message. Chaque noeud 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 noeud 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 doctets qui peut être sous différentes formes : une base de données objet ou relationnelle, un fichier (texte et hypertexte, XML), un flux (vidéo, audio, VoIP), des fichiers P2P, des requêtes de base de données ou des résultats issus d'un calcul/service.

2.2 Communication et les réseaux overlay

Le modèle de communication présenté en Section 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 connectés dirigés et qui répond donc aux besoins de l'architecture GRAPP&S.

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 [46]. 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 pré-traitement. 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 3.2).

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 issus de capteurs ou encore un service 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.

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 2.1a). 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.

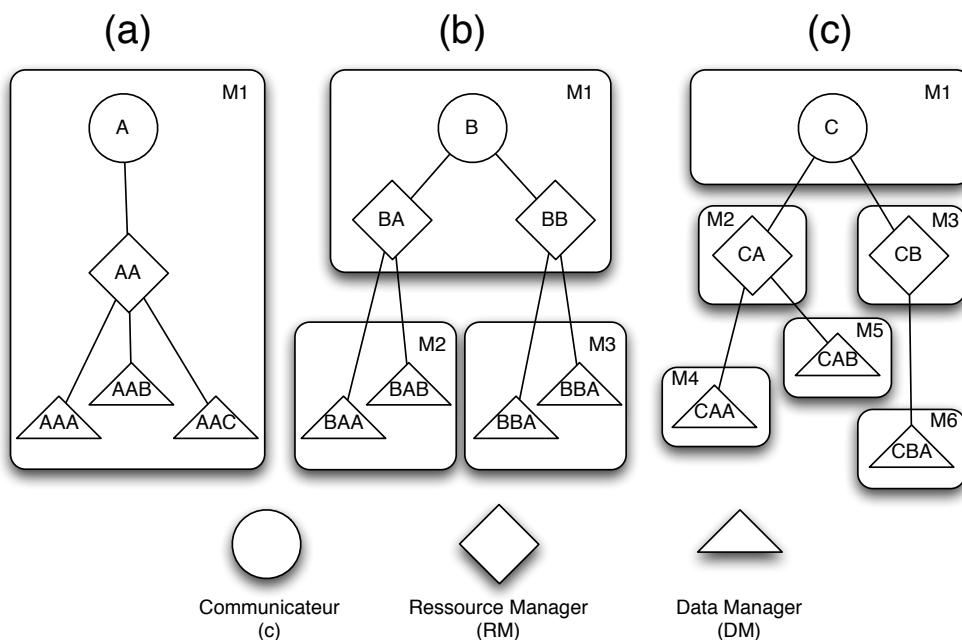


FIGURE 2.1 : Organisation des nœuds (a) dans une machine, (b) dans un cluster et (c) dans un réseau

Dans (ii) les nœuds sont organisés dans une ferme de serveurs telle qu'un cluster, ce qui est caractéristique des réseaux HPC (Figure 2.1b). Finalement, (iii) les nœuds peuvent être

regroupés s'ils partagent une même propriété de localisation ou d'administration (voir Figure 2.1c). 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 déploiement des réseaux IPv6¹. 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_communauté:uuid:chaîne-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 (FreePastry² et Phex³, par exemple) seront toujours compatibles, une fois la connexion établie entre leurs *communicator*.

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 l'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 connaît 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 connaît 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

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

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 des involontaires (pannes). Comme le cas des déconnexions volontaires est trivial, nous nous concentrerons 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*) [28] 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 temporiseurs 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, tables 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.

3.2 Algorithmes d'élection

Vu le caractère dynamique et volatile des réseaux informatiques, il est important de choisir un algorithme d'élection qui soit le plus léger et réactif possible. L'élection d'un nœud peut être nécessaire en deux situations : soit pour remplacer un nœud défaillant et garantir la continuité du service (par exemple, lors de la panne d'un nœud c), mais aussi pour simplifier la coordination entre les nœuds de même type, avec par exemple l'élection d'un RM qui agirait comme "super-node" pour l'indexation de données et services.

Il faut noter que la connaissance préalable des nœuds du niveau supérieur n'est pas obligatoire, vu que différentes techniques permettent d'obtenir les identifiants des autres nœuds. La méthode la plus simple consiste à utiliser directement 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 réintègrer 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 [96, 93, 17]. 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. 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 Opérations dans GRAPP&S

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-noeud au sein des RMs ou d'un mécanisme de DHT. Il est aussi possible de favoriser la réPLICATION des index et (voir 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 noeud ("super-node") n'est pas une obligation dans notre structure mais simplement une spécificité pouvant être présente dans une implémentation donnée.

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 2.2 illustre une partie de cette procédure de recherche :

1. Un noeud $DM_i \in C_i$ envoie la requête a son noeud RM_i $E(C_i)$
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 noeud RM_i retourne au noeud DM_i une liste de noeuds DM qui contiennent l'information recherchée
4. Sinon, le noeud 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 noeud $c_i \in C_i$ pour retransmission aux autres $RM_k \in C_i$
5. quand un noeud $RM_k \in CM_i$) trouve la bonne réponse, alors la requête sera retournée au noeud 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

En cas de réussie, le client obtient l'identifiant du noeud 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 noeud DM_x afin d'accéder à la donnée. Comme le client peut se trouver dans un autre réseau qui ne permet pas l'accès directe à DM_x , la connexion peut se faire par routage interne dans GrAPP&S. Cela se fait de la manière suivante :

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

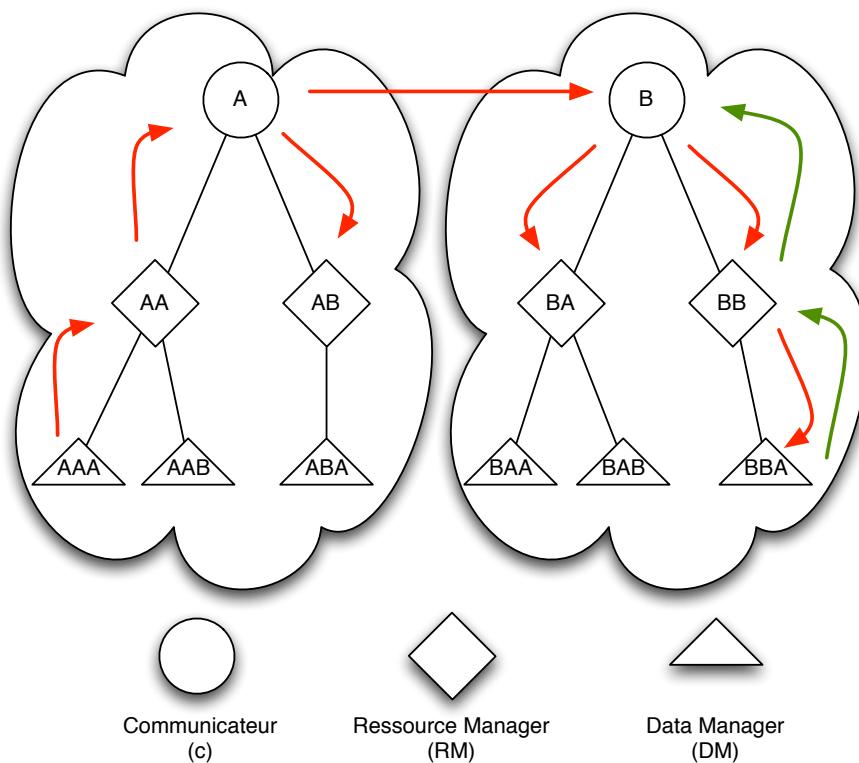


FIGURE 2.2 : Recherche d'une information dans GRAPP&S et mécanisme de routage préfixé

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* [113]) pour agréger les requêtes et réduire le nombre de messages d'une recherche.

5 Bilan et Perspectives

La spécification de GRAPP&S a permis une meilleure compréhension des différents mécanismes liés au stockage et au partage de données dans un réseau P2P structuré. Toutefois, son implémentation n'a pas eu lieu car XXX Néanmoins, certains principes liés à la clusterisation, au stockage des données dans un environnement P2P et le routage des informations a servi comme toile de fond pour des travaux plus récents, notamment la plate-forme de calcul pervasif CloudFIT décrite en Chapitre ??.

Chapitre 3

L'hétérogénéité des Ressources de Calcul

Résumé

La gestion de l'hétérogénéité des ressources peut être considérée sous plusieurs aspects. Les chapitres précédents ont donné des exemples de travaux dans lesquels l'hétérogénéité s'était concentrée sur les communications (Chapitre ??) ou sur le couple données-communication (Chapitre 2). Dans ce chapitre nous nous concentrerons sur un troisième socle, celui de l'hétérogénéité du calcul. Plus exactement, ce chapitre présente des travaux dont les contributions ont visé la prise en charge de l'hétérogénéité des ressources de calcul ou l'hétérogénéité des tâches de calcul, tous les deux résultant en une variabilité des temps d'exécution de tâches qui a dû être traitée et optimisée. Il est aussi important de remarquer que la communication et le stockage peuvent varier mais, dans ces cas, ils sont traités par des outils et mécanismes sous-jacents qui n'ont pas été la cible de nos travaux.

Ainsi, ce chapitre démarre avec la description d'une plate-forme de calcul destinée à l'exécution de problèmes d'amarrage moléculaire. Ce travail, développé dans le cadre de la codirection de thèse de Romain Vasseur, est un exemple d'application métier où l'environnement développée sert à déployer de manière distribuée une application déjà existante et dont le code source ne pourrait pas être facilement parallélisée. L'approche retenue a été composée d'une gestion distribuée d'instances de calcul sur un cluster/grappe, associée à un découpage plus fin du problème afin de multiplier les tâches de calcul et ainsi mieux utiliser les ressources disponibles.

La deuxième partie de ce chapitre décrit les efforts effectués dans le cadre du projet de collaboration international STIC-AmSud PER-MARE, dont l'un des objectifs a été de rendre la plateforme de calcul big data Hadoop sensible au contexte et donc capable d'adapter la distribution des tâches de calcul en fonction des variations des ressources.

1 Hétérogénéité des Tâches - 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, 52, 77].

Souvent cette étude se fait à travers un "criblage virtuel" (*virtual screening*), qui consiste en un déploiement à grande échelle permettant de tester un grand nombre de ligands (de centaines à plusieurs millions selon lampleur 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 [135, 136], ZINC [66] 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) [105], 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 (thèse en bioinformatique, dirigée par le prof. Manuel Dauchez et co-encadré par Stéphanie Baud et moi-même) 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 à une interaction avec le ligand, parmi un échantillon de structures de protéines plus ou moins importante. 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.

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.* [31, 30]. 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 [118] ou Auto-Dock Vina [82, 81], avec très peu d'outils dédiés exclusivement au docking inversé (INVDOCK

[31, 30] ou TarFisDock [83]). Toutefois, aucun de ces articles ne décrit ni 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 paire 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 développé.

1.2 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 3.1. 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.

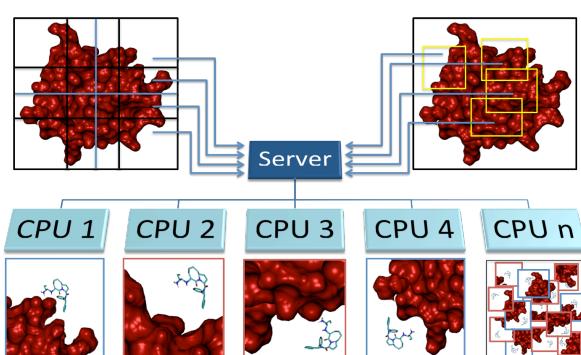


FIGURE 3.1 : Exemple d'un schéma de décomposition parallèle

1.2.1 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.

1.2.2 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 paire 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.

1.2.3 Recherche de cavités

Comme indiqué précédemment, l'amarrage des ligands est favorisé par la présence de cavités dans la surface de la protéine [51, 61], 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 [57]. 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 générerons 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.

1.3 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. Pour toutes ces raisons, nous avons opté pour la création d'une plate-forme basée sur le langage Python afin d'exploiter le parallélisme multi-coeur et multi-machine dans le but d'automatiser la génération et le déploiement des tâches de calcul liées au docking inversé.

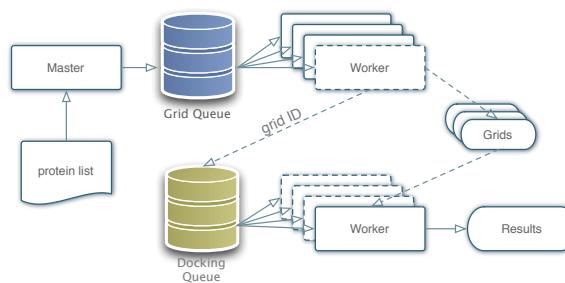


FIGURE 3.2 : Représentation du flot d'exécution dans l'architecture distribuée

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 de l'approche par recherche de cavités, des grilles 3D sont générées uniquement autour des zones identifiées par le logiciel Fpocket.

Finalement, l'exécution parallèle 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". Grâce à cette stratégie, les différents esclaves obtiennent une ou plusieurs tâches à exécuter, selon le nombre de coeurs de calcul disponibles dans leurs processeurs. 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 3.2 illustre le flot d'exécution de ces deux étapes.

1.3.1 Optimisation aux clusters HPC

L'architecture distribuée présentée ci-dessous est adaptée à une exécution sur tout type de réseau d'ordinateurs (cluster, cloud, grille pervasive, 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 [60], OAR [21] ou Slurm [144] 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 noeud maître et permettre ainsi l'exécution des tâches indépendantes selon les disponibilités du gestionnaire de tâches. En effet, la présence d'un noeud 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.

1.4 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 noeuds Westmere-EP (12 coeurs), 2 noeuds Nehalem-EX (32 coeurs), un noeud Westmere-EP (12 coeurs + 2 Fermi C2050 GPUs) et un noeud Nehalem-EP (8 coeurs + GPU Fermi M2090), et au moins 2GB de mémoire par coeur. Pour une question de régularité, les expériences ici présentées ont été lancées exclusivement sur les noeuds Westmere-EP nodes. De plus, afin de mieux évaluer la communication intra-cluster, seulement 4 coeurs de calcul ont été utilisés par noeud.

Ainsi, pour la première expérience, nous avons initialement exécuté un blind docking de la protéine 3CM2 et du ligand X23. Comme illustré en Fig. 3.3, 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).

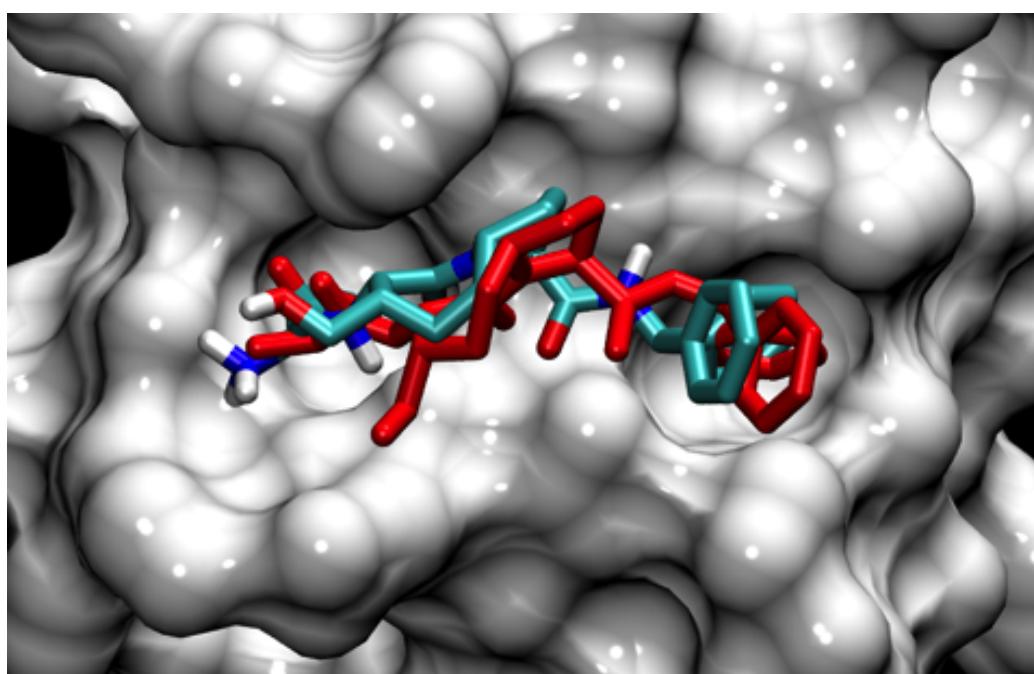


FIGURE 3.3 : Placement du ligand selon la méthode cristallographique (rouge) et par blind docking (vert)

TABLE 3.1 : 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 (\AA) |
|---|---------------------|--|--------------------------|
| $\Delta G = -10.27$ | 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 Fig. 3.4. 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 3.1). Les autres formats

de découpage ont réussi à placer le ligand dans la cavité cible, mais leurs résultats en terme 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.

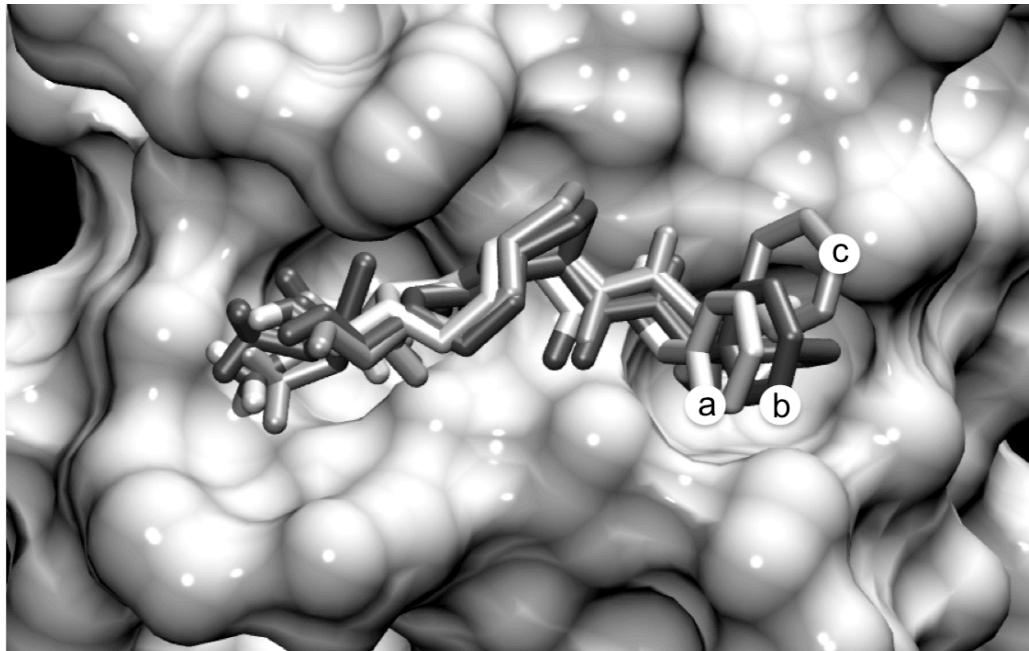


FIGURE 3.4 : Comparaison entre la pose blind docking (a) et celle d'un découpage à $n = 12$ (b) et par recherche de cavités (c).

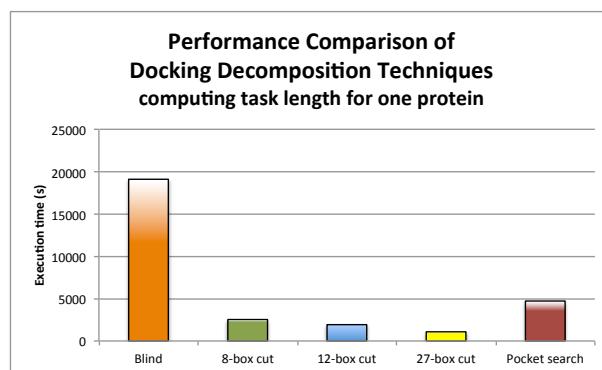


FIGURE 3.5 : Comparaison des temps d'exécution pour les tâches issues des différentes méthodes de découpage

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 Fig. 3.5 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 ré-exécution 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.

2 Hétérogénéité des Ressources de Calcul - adaptation de Apache Hadoop aux grilles pervasives

La suite logicielle Apache Hadoop¹ 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 [39]. 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 - [125]) 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 grille pervasive, c'est à dire, des environnements de calcul faiblement connexes marqués par l'hétérogénéité et par la volatilité des ressources [121]. Le projet PER-MARE était organisé autour de deux volets : d'un côté l'adaptation de la plate-forme 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 (la plate-forme CloudFIT, traitée en section XXXX).

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 plate-forme.

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 développé initialement par Facebook). D'autres outils faisaient partie du projet dès son départ mais ont gagné un statut "projet" propre, comme par exemple le service Zookeeper, responsable de la coordination distribuée fiable entre les noeuds et souvent au cœur des efforts de tolérance aux fautes de Hadoop.

La plate-forme Hadoop elle aussi a subit 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 plate-forme. À partir de 2012 la version 2.x Hadoop devient une plate-forme 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.

Au coeur 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

1. <http://hadoop.apache.org/>

(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 Fig. 3.6 : les processus *NameNode* et *ResourceManager* correspondent aux rôles de maître dans HDFS et YARN, respectivement, et les processus *DataNode* et *NodeManager* correspondent aux parties esclaves.

Nous pouvons observer aussi dans la Figure 3.6 la présence de deux autres composants appelés *ApplicationMaster* et *Containers* (conteneurs). L'*ApplicationMaster* est un processus désigné pour effectuer l'ordonnancement des tâches de calcul d'une seule application, qui seront exécutées par des éléments *Container* associés.

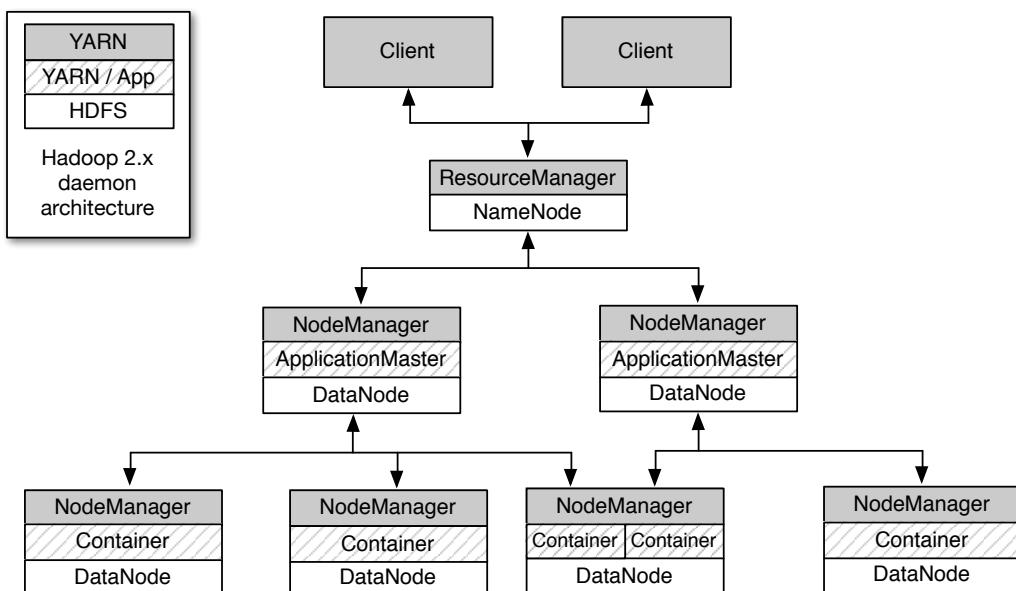


FIGURE 3.6 : Architecture de base de Apache Hadoop 2.x

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 *ResourceManager*, la seule entité qui a une vue globale des ressources du système grâce aux informations envoyées par les *NodeManager*. Grâce à ces informations, le *ResourceManager* 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 *ApplicationMaster* 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 *ResourceManager* a été projeté de manière à pouvoir être optimisé selon contraintes et paramètres des 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 *ApplicationMaster*, qui a une vue plus proche de l'application mais qui est aussi limité aux ressources que lui sont attribuées au départ.

Malheureusement, le fonctionnement de l'*ApplicationMaster* 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 [24]). Ce que ressort est un simple mécanisme de remplissage des noeuds visant la proximité des tâches : on remplit un noeud avec autant de conteneurs qu'il peut supporter, pour ensuite commencer le remplissage du prochain noeud.

Ceci nous a permis aussi d'observer que l'*ApplicationMaster* se limite à répartir les conteneurs sans une véritable adéquation au contexte d'exécution. Toute connaissance sur la capacité des noeuds provient du *ResourceManager*, 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 *ResourceManager*.

2.2 État de l'Art

La littérature propose différentes approches pour rendre Hadoop plus compatible avec les environnements hétérogènes. Des travaux comme [79], [129] ou [104] assument que les applications Map-Reduce 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, [67] 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, [145] et [29] essayent d'améliorer la distribution des tâches afin de réduire le temps de réponse dans des clusters de grande taille. Pour cela, [145] utilise des heuristiques pour estimer la progression des tâches et ainsi décider si il faut lancer des tâches spéculatives. Les tâches spéculatives sont des doublons (ré-soumissions) qui sont lancées lorsqu'il y a la suspicion qu'une tâche originale est retardée à cause d'un noeud défaillant ou trop lent. Dans une ligne similaire, [29] propose l'utilisation des traces historiques d'exécution afin d'aider cette décision.

Une autre manière d'augmenter la performance passe par une meilleur placement des données et par l'utilisation de cette information pour le déploiement des jobs [142]. En faisant un placement optimisé des données, on réduit les transferts de données occasionnés par le lancement de tâches spéculatives sur d'autres noeuds. Une approche similaire est présentée par [26],

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 noeuds les plus rapides à travailler plus et les moins performants à exécuter moins de tâches. Une manière de rompre cette logique est utilisée par [111], qui permet d'influencer l'ordonnancement grâce à des profils d'exécution suggérés par l'utilisateur (par exemple, privilégier les noeuds lents si le job n'est pas prioritaire).

Il faut observer cependant que la difficulté à adapter l'exécution de Map-Reduce sur des environnements hétérogènes (et dynamiques) est en grande partie due à la conception même de la plate-forme Apache Hadoop, qui est très hiérarchique (voir Figure 3.6). Certains travaux essaient de s'affranchir des ces barrières en développant d'autres plates-formes compatibles avec Map-Reduce mais plus adaptées à la dynamité des ressources. L'utilisation de overlays P2P est ainsi un choix naturel, comme le montrent [90] et [123]. Dans le système proposé par [90], les noeuds 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'exploré pas les possibilités d'optimisation de l'ordonnancement des jobs et des tâches.

L'approche adoptée par la plate-forme CloudFIT [123] est différente car même si elle repose aussi sur un overlay P2P, on n'essaie pas d'imiter le fonctionnement de Hadoop. CloudFIT est une plate-forme générique de calcul distribué, où des tâches Map et Reduce sont distribuées aux noeuds de façon opportuniste, selon un mécanisme "bag of tasks". Cette distribution est aussi guidée un ordonnancement guidé par le contexte des ressources et par des profils d'exécution fournis par les applications. Nous détaillerons le fonctionnement de CloudFIT dans le chapitre suivant.

Il faut aussi noter que, à l'exception de CloudFIT, les travaux cités précédemment ne tiennent pas compte de 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, 87, 103, 94], Hadoop reste essentiellement une plate-forme statique. 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.

2.3 Ordonnancement Orienté par le Contexte

Comme indiqué dans la section 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 *NodeManager* déclare ses ressources au *ResourceManager* 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 *NodeManager* de communiquer périodiquement ses ressources au *ResourceManager*.

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.

2.3.1 Le collecteur de contexte

Par défaut, Hadoop obtient des informations sur les ressources des noeuds à partir de fichiers de configuration au format XML. Ces fichiers contiennent plusieurs paramètres, dont le nombre d'unités d'exécution (coeurs de calcul) et la capacité de la mémoire des noeuds. Une fois lues, ces informations ne sont pas mises à jour, sauf en cas de redémarrage du noeud. 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 *ResourceManager*.

Ce collecteur de contexte a été développé dans le cadre du projet PER-MARE[125] et est structuré selon le diagramme de classes présenté en Figure 3.7 [24]. La capture des différents éléments de contexte se font grâce à l'API standard Java Monitoring API [95], 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 (coeurs de calcul), la mémoire du système, ou la charge de ma 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 la collecte des données. De plus, en raison de sa conception, il est simple d'intégrer des nouveaux collecteurs et ainsi diversifier les informations de contexte.

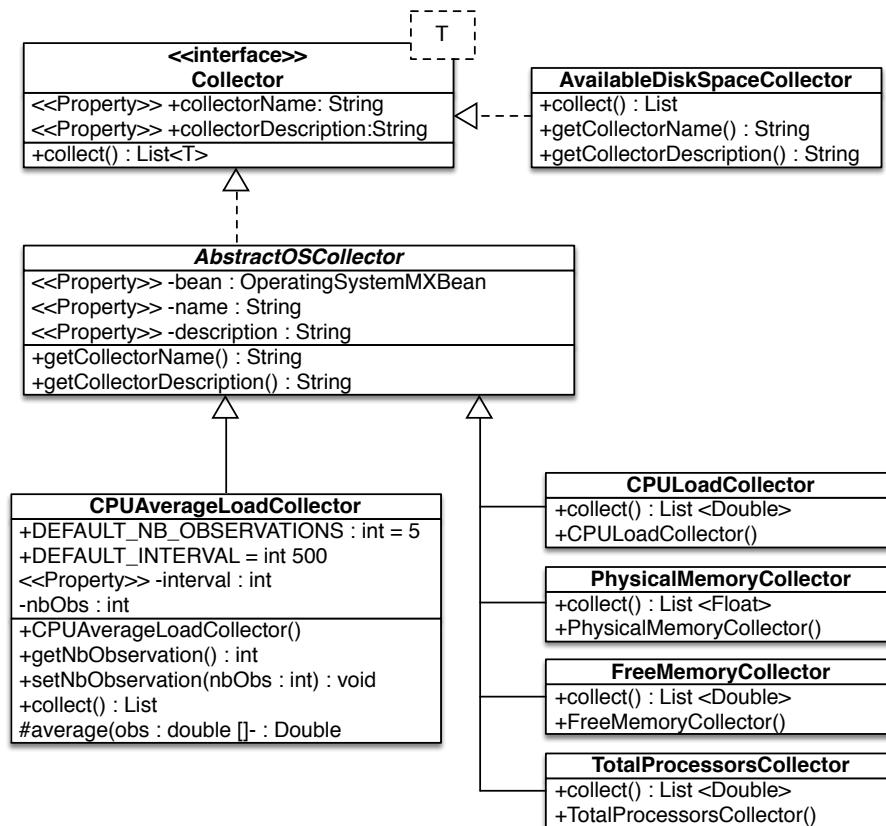


FIGURE 3.7 : Structure du collecteur de contexte

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 *ResourceManager*, il faut que le collecteur de contexte de chaque noeud puisse communiquer son état au *ResourceManager*, et cela à n'importe quel moment de l'exécution. Afin de rendre ceci possible, nous avons étendu les possibilités de communication entre le *ResourceManager* et les

NodeManager, comme expliqué dans la section suivante.

2.3.2 Communication

Dans l'architecture Hadoop, les informations de contexte collectées par les noeuds esclaves (*NodeManager*) doivent être transmises au noeud maître (*ResourceManager*), 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 [64], 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 disponibles auprès le *ResourceManager*.

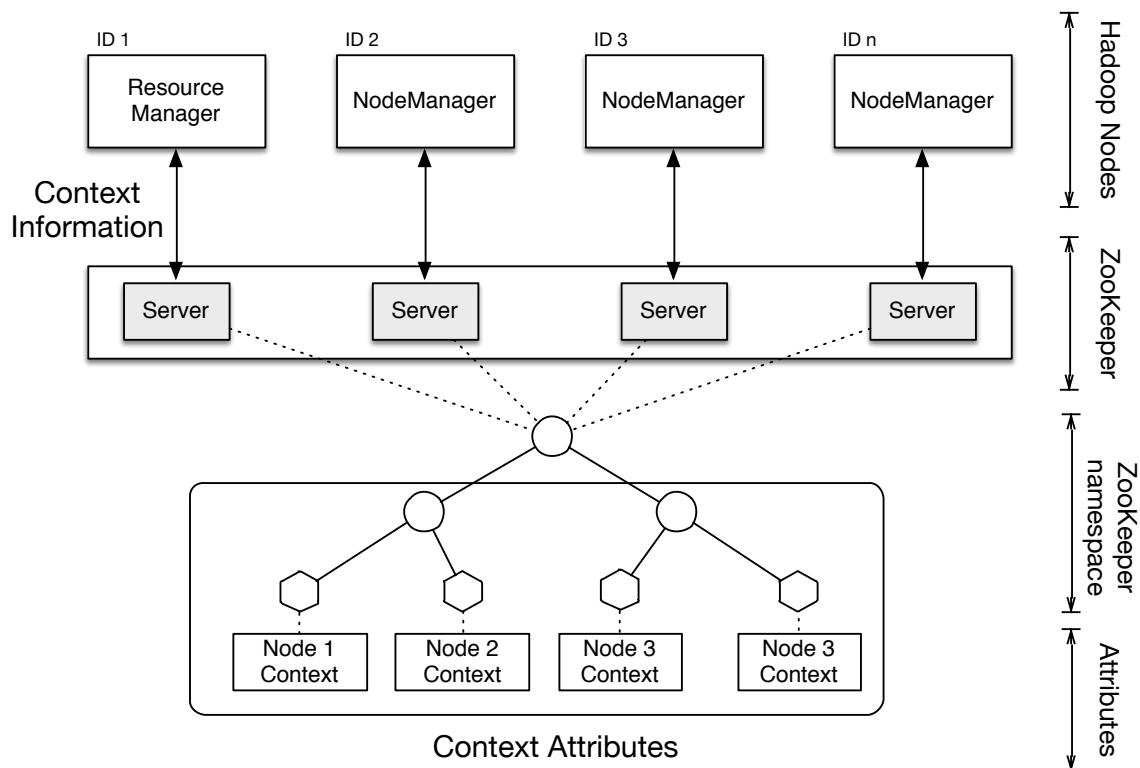


FIGURE 3.8 : Utilisation de ZooKeeper pour distribuer l'information de contexte

Comme illustré en Figure 3.8, tous les esclaves (*NodeManager*) 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/plafond, le tableau dans ZooKeeper sera mis à jour. Ce seuil/plafond 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 noeud. Ce mécanisme contribue aussi pour réduire la quantité d'informations échangées et évite trop d'événements qui pourrait impacter la performance de l'algorithme d'ordonnancement.

De manière similaire, le maître (*ResourceManager*) 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'ordonnateur. Les modifications apportées au code source du *ResourceManager* et des *NodeManager* 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.

2.4 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 coeurs de calcul (*v-cores*). Ces paramètres sont toujours renseignés au *ResourceManager* et font partie des principaux attributs utilisés par l'algorithme Capacity Scheduler. En effet, la mémoire totale disponible et le nombre de coeurs permettent la définition du nombre de tâches simultanées (conteneurs) qui peuvent être exécutées par un noeud. 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 coeurs de calcul renseigné correspond au nombre de coeurs 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 coeurs 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.

2.4.1 Benchmarks et environnement de test

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'en-

trée/sortie (I/O) sont aussi importantes. Pour cela, les deux applications choisies ont des profils différentes par rapport à leurs besoins en mémoire, CPU et I/O [84], comme indiqué ci-dessous :

- TeraSort : L'application TeraSort [58] 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 plate-forme de tests HiBench [63]. 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 plate-forme Grid'5000 [53]. 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 coeurs par noeud) et 8 GB de mémoire RAM. Tous les noeuds 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 noeud 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 noeud séparé afin de ne pas surcharger les noeuds esclaves avec des activités de gestion de Hadoop.

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

2.4.2 Résultats

Les exécutions des benchmarks dans les différents scénarios sont représentées par les diagrammes de Gantt des Figures 3.9 et 3.10, respectivement pour TeraSort et TestDFSIO. De même, les Tableaux 3.2 et 3.3 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.

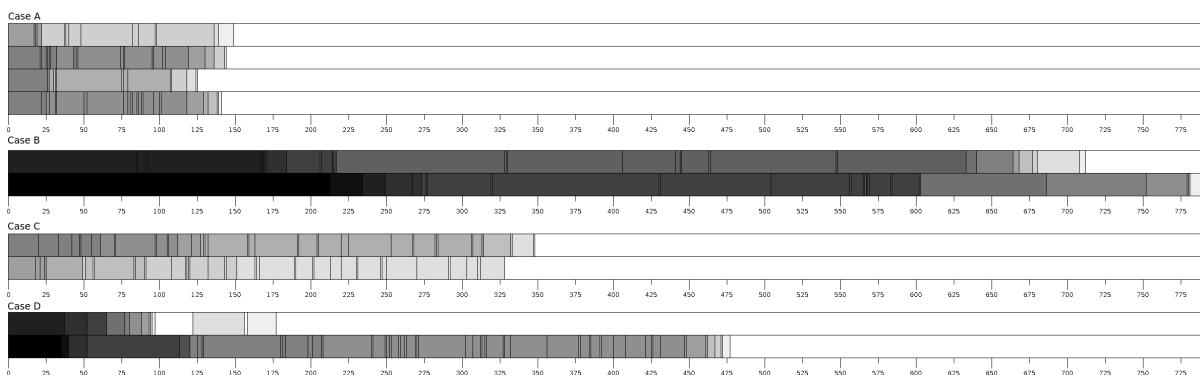
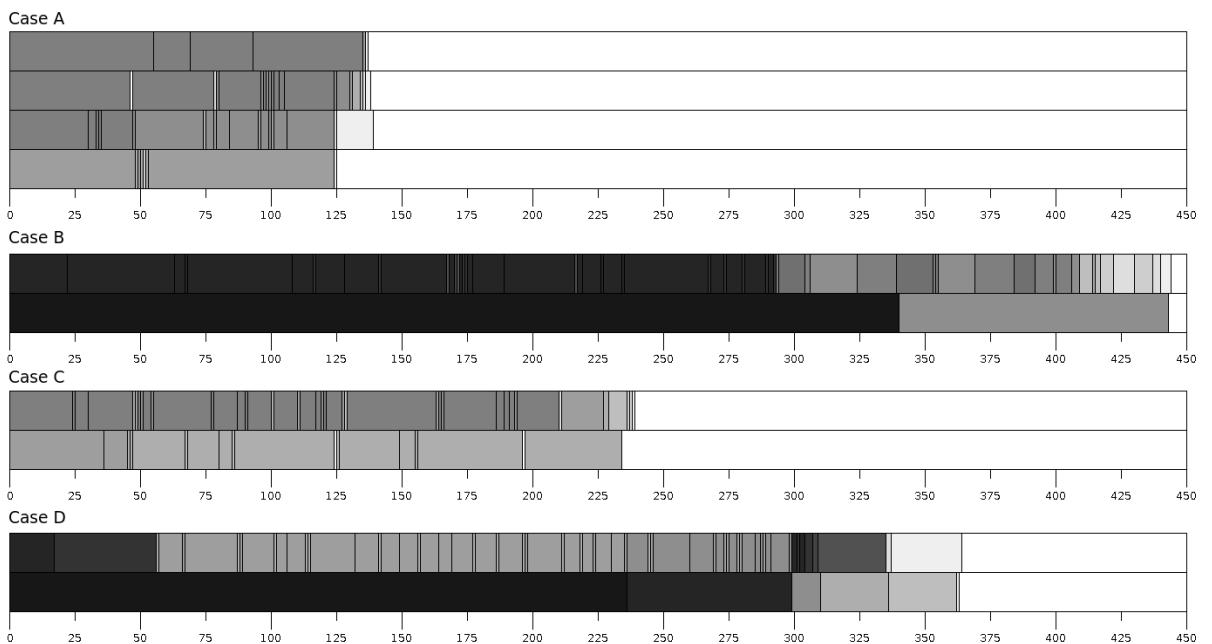


FIGURE 3.9 : Diagramme de Gantt pour l'exécution de TeraSort

**FIGURE 3.10 :** Diagramme de Gantt pour l'exécution de TestDFSIO**TABLE 3.2 :** Tableau récapitulatif de l'exécution de TeraSort

| Scénario | A | B | C | D |
|----------------------------|-------|--------|-------|-------|
| Temps total <i>map</i> (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 <i>map</i> | 76 | 76 | 76 | 76 |
| # tâches spéculatives | 2 | 1 | 3 | 1 |

Pour les diagrammes de Gantt, chaque scénario est composé de 2 ou 4 lignes correspondant au nombre de noeuds utilisés. Comme indiqué précédemment, les scénarios B, C et D n'utilisent que la moitié des noeuds du scénario A afin de simuler la réduction des ressources. L'échelle de couleurs présent dans chaque ligne indique la surcharge des noeuds : 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 noeud). 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'exécution : le scénario A

TABLE 3.3 : Tableau récapitulatif de l'exécution de TeraSort

| Scénario | A | B | C | D |
|----------------------------|-------|-------|-------|-------|
| Temps total <i>map</i> (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 <i>map</i> | 90 | 90 | 90 | 90 |
| # tâches spéculatives | 0 | 9 | 0 | 1 |

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 moyen et les plus petites variations de performance, indépendamment de l'application. Ceci s'explique par le fait que dans ces deux scénarios les noeuds 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 spéculatives : une tâche spéculative n'est lancée que seulement 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 noeuds. 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 noeud 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.

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 plate-forme 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 modification au niveau du code source de Hadoop, rendant la solution suffisamment générique et intégrable aux différentes versions de ce framework.

3 Bilan et Perspectives

Chapitre 4

CloudFIT : un intergiciel pour le *Fog Computing* et l'Internet des Objets

Résumé

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 de manière adaptée à ces 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 niveau d'humidité du sol pour nos plantes.

De nos jours, les principales limitations d'utilisation de ces dispositifs ne sont pas liées à leur capacité calcul ou de communication (WiFi, Bluetooth, etc.), mais surtout à la difficulté d'une application exploiter et 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 qui 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 l'interaction et la collecte et le traitement d'informations à grande échelle. De plus, l'augmentation de la bande passante et de la puissance de calcul des appareils, couplés avec un coût décroissant de capteurs [72], nous permettent d'envisager des applications et stratégies de traitement de données bien différentes de celles développées jusqu'à aujourd'hui.

En effet, la plupart des applications reposent sur un modèle client-serveur. Dans le cas de l'IoT, l'agrégation et l'analyse des données sont effectuées principalement sur des infrastructures déportées de type *cloud computing*. Plusieurs travaux [92, 56, 45] comptent sur ces infrastructures car elles offrent de la puissance de calcul et de la flexibilité pour l'exécution de services et applications [115]. Malgré ses avantages, les plates-formes de type *cloud* ont aussi quelques inconvénients importants. En effet, le transfert de données peut prendre du temps 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 défective ou trop lente.

Par conséquent, nous devons repenser la façon de transmettre, de stocker et d’analyser les données dans ces environnements. Les architectures réseau 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 elles 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 grilles pervasives [97], le *mobile edge computing* [40, 43, 112], le *fog computing* [18] ou bien le *edge-centered computing* [50]. 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 plate-forme 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 [47], Apache Hadoop [4, 119], etc.) nous fait prendre conscience de la complexité de ces systèmes et des limitations à leur extension ou modification.

Dans les sections suivantes nous allons présenter

2 État de l’art

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 traitement des données se fait sur un ou plusieurs serveurs distants (serveurs, clusters, data-centers, cloud). C’est ainsi que des approches telles que les grilles pervasives [97], le *mobile edge computing* [40, 43, 112], le *edge-centered computing* [50] ou bien le *fog computing* [18] ont été proposées dans le but de placer les applications et les services plus près de l’utilisateur final.

Les travaux sur l’*edge computing* et le *fog computing* partagent souvent les mêmes définitions [133]. En effet, le *fog computing* a été défini par CISCO [35] comme "un paradigme qui étend le *cloud computing* et des 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" [133]. Des exemples de *edge/fog* comprennent des services *fog* [18] et les *cloudlets* [112], tous les deux proposant le déploiement des serveurs de proximité offrant des services avec une latence réduite. À quelques exceptions près, comme [40], 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. [50] 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 prox-

mité, 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 grille pervasive [97], qui vise l'intégration des dispositifs de détection/d'actionnement ainsi que des systèmes de haute performance classiques. Ces grilles reposent sur l'utilisation des ressources habituellement sous-utilisées, composant ainsi une plate-forme de calcul dynamique [124]. En effet, les grilles pervasives offrent la possibilité d'intégrer les différents ressources disponibles allant des petits appareils de type Raspberry Pi jusqu'aux machines virtuelles déployées sur les infrastructures d'un data-center. Pour l'IoT, les grilles pervasives 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.

Un autre avantage des grilles pervasives est son indépendance par rapport à des architectures et services opérateur. 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 télécom et cloud tels que Cisco, Intel ou Microsoft, réunis par exemple au sein de l'Open Fog Consortium¹ 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 au déploiement de plates-formes légères, ce qui est notre objectif principal.

Finalement, afin de répondre aux différents besoins des architectures et applications IoT, les grilles pervasives peuvent être enrichies avec 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 [106, 107]. 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. Nous croyons que ce concept offre plusieurs niveaux de granularité et d'interconnexion nécessaires à l'autonomie des dispositifs IoT et permet des services plus réactifs et de meilleure qualité. 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 clé.

La diversité de travaux et variations autour du fog computing ne sont pas nécessairement suivies par un offre en outils et plate-formes pour sa mise en oeuvre. En effet, la plupart des auteurs cherchent encore la meilleure manière de déployer et coordonner les noeuds dans des tels environnements. Bien que souvent cités, des approches basées sur la virtualisation [112], micro-clouds[42], micro-services [134] ou des workflows [59] ne sont que des possibilités pour la mise en place du fog. Comme remarqué par *Yi et al.* [143], les challenges sont multiples et incluent aussi le réseau (overlays P2P ou SDN), le déploiement, l'orchestration, la migration de tâches/services, etc. Parmi les rares plates-formes opérationnelles dédiées au fog on peut citer IOx de Cisco [34] et Paradrop [138]. La plate-forme de Cisco repose sur l'hébergement de machines virtuelles sur des routeurs et switches compatibles, et les utilisateurs disposent de

1. <http://www.openfogconsortium.org/>

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 poussé. 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), lesquelles doivent se connecter à des serveur ParaDrop, ce qui empêche la décentralisation du fog. L'absence de plates-formes vraiment dédiées au fog computing peut s'expliquer par le manque de standardisation. Ceci pourra changer dans le futur avec la publication des spécifications du Open Fog Consortium (initialement prévue pour le début 2017 mais toujours pas publiées), mais dans pour le moment les initiatives sont rares et limitées.

À une moindre mesure, nous pouvons essayer d'utiliser des plates-formes existantes comme point de départ pour le développement d'un réseau fog computing. Une plate-forme prometteuse serait celle du projet Apache Storm[126], basée sur un réseau P2P et permettant la soumission de services (topologies) dédiées au traitement de streams. Toutefois, la plate-forme Apache Storm a été développée pour un déploiement sur un cluster et donc a l'inconvénient de ne pas prendre en compte les caractéristiques des machines ni leur contexte d'exécution. De plus, c'est une plate-forme assez complexe car riche en fonctionnalités tiers (authentification, RPC, APIs indépendantes du langage de programmation) et en mutation (elle vient de subir une mise à jour majeure), un aspect qu'on souhaite éviter à cause des problèmes rencontrés précédemment avec Apache Hadoop lors du projet STIC-AmSud PER-MARE.

3 De CONFIIT à CloudFIT

CloudFIT est une plate-forme de calcul distribué que reprend et étend le paradigme FIIT (Finite number if Independent and Irregular Tasks) défini par Krajecki [78]. 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 modèle correspond au modèle de programmation BSP (Bulk-Synchronous Parallel) [132], 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 assumption 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 4.1 illustre l'exécution d'un superstep BSP dans lequel les noeuds 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 simple-

ment 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.

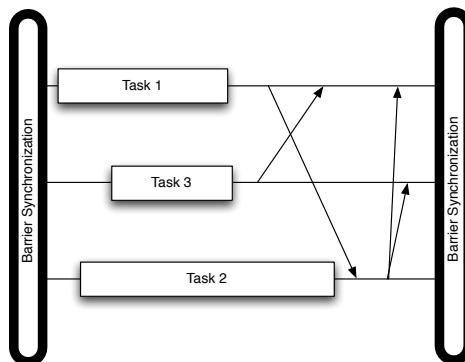


FIGURE 4.1 : Un superstep dans le modèle BSP

Une première implémentation du paradigme FIIT a vu le jour avec la plate-forme CONFIIT (Computation Over Network for FIIT) [48, 47]. CONFIIT a été développé en tant que middleware pour le calcul distribué, en s'appuyant sur un anneau logique (overlay) et des échanges XML-RPC entre les noeuds. À la suite de sa version initiale, CONFIIT a été fortement étendu 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 plate-forme de calcul pour plusieurs travaux, notamment lors de la résolution parallèle des instances L(2,23) et L(2,24) du problème de Langford [68].

Lors du démarrage du projet STIC-AmSud PER-MARE, nous avons voulu utiliser CONFIIT comme plate-forme pour l'exécution de tâches *big data* Map-Reduce, 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 tâches et les données associées. 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 noeuds, causant ainsi leur déconnexion.

Après plusieurs tentatives, nous avons pris la décision de développer une nouvelle plate-forme, 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.

3.1 Spécification des Besoins

La décision d’implémenter une nouvelle plate-forme 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’objectifs visant la généricité et la maintenance de la plate-forme :

- 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 exigences des applications ;
- R2** CloudFIT doit être modulable 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 ou à des propriétés lors du lancement des jobs ;
- R3** CloudFIT doit supporter le déploiement d’applications à la volée. Un utilisateur doit pouvoir soumettre ses classes applicatives à CloudFIT, qui les intégrera à la file d’exécution et les déployera aux différents noeuds du réseau.

L’objectif [R1] vient directement de l’expérience avec CONFIIT, où une dépendance trop forte par rapport au middleware P2P pourrait gêner l’évolution de la plate-forme. Comme les réseaux pervasifs présentent des grandes variations de performance et 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 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 le overlay P2P initialement retenu (FreePastry²) s’est plus tard révélé peu performant et on a pu facilement migrer vers un nouveau overlay, TomP2P³.

La règle [R2] est autant une prérogative visant l’évolution et la maintenance de la plate-forme 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 module.

Finalement, l’objectif [R3] garantit la scalabilité de la solution. En effet, il est inconcevable d’avoir à distribuer l’application aux noeuds de calcul avant de lancer une application, comme c’était le cas avec CONFIIT. C’est le rôle de la plate-forme d’effectuer ce déploiement et d’assurer le lancement des applications selon des appels bien définis dans l’interface applicative.

3.2 Architecture

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

Malgré son nom, la couche **Network** est responsable pour toute interaction avec les systèmes tiers sur lesquels CloudFIT s’exécute (overlay P2P, système d’exploitation, systèmes de stockage). Le Network Adapter exécute ainsi les opérations élémentaires d’encapsulation et décapapsulation des messages, grâce à des primitives conçues selon les capacités des overlays P2P

2. <http://www.freepastry.org/>

3. <http://tomp2p.net>

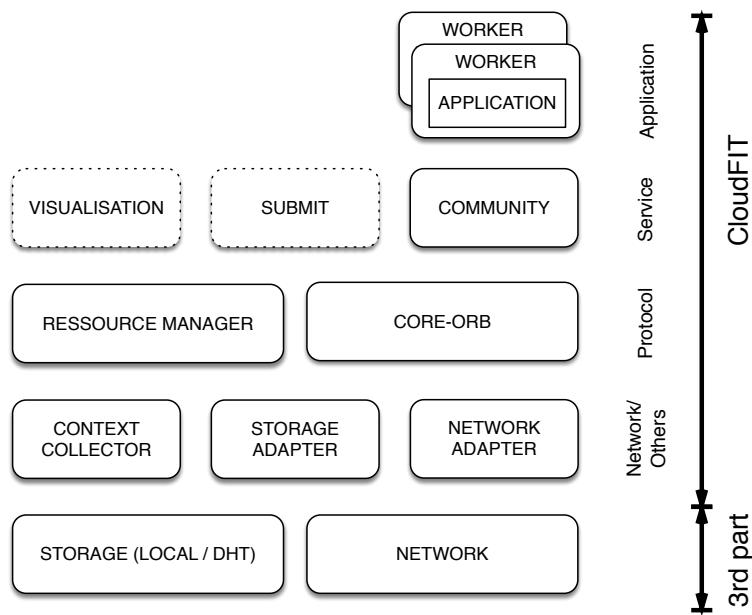


FIGURE 4.2 : Représentation simplifiée de la pile logicielle CloudFIT

subjacents (*send*, *sendAll*, *receive*, etc.). Le même principe s'applique au 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 collecteur de contexte, déjà présenté dans le chapitre précédent en Section 2.3.1 et qui a été intégré aussi à CloudFIT.

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

Le Ressource Manager, de son côté, puise dans les informations extraites par le collecteur de contexte pour vérifier si les ressources présents dans la machine sont compatibles avec les besoins des applications. Ces besoins sont des propriétés fonctionnelles (mémoire disponible, espace disque, etc.) renseignées par l'application lors de sa soumission. Le Ressource 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 et responsable par le déploiement des applications et la gestion des événements liés aux noeuds (entrée, sortie, retransmission de messages, etc.). Chaque communauté est associée à un JobScheduler, qui gère la file de jobs (applications soumises) et choisi lesquels peuvent être lancés sur la machine, en croisant les contraintes des applications et les informations du ResourceManager. Il faut remarquer que plusieurs communautés peuvent coexister sur un noeud et dans le réseau, permettant ainsi la création de sous-ensembles de noeuds 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 des micro-contrôleurs Arduino) : dans ce cas, il suffit d’offrir un accès à un noeud CloudFIT grâce à une interface REST ou JSON, comme suggère la Figure 4.3. Ces interfaces peuvent être utilisées autant pour le simple stockage de données que pour le déclenchement d’applications.

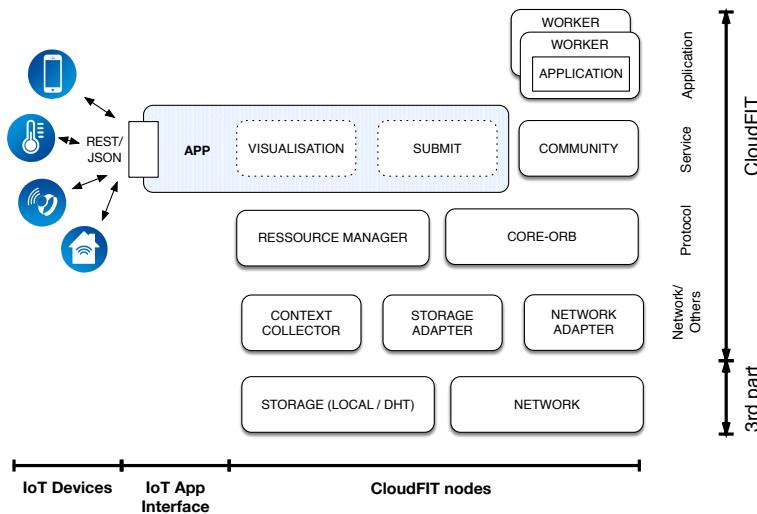


FIGURE 4.3 : Exemple d’interface IoT pour l’interaction avec CloudFIT

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 TaskScheduler ;

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 TaskScheduler pour gérer l’ordre d’exécution afin de respecter les dépendances ;

finalizeApplication() méthode optionnelle qui est exécutée par le TaskScheduler 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 Map-Reduce.

On y trouve aussi la classe TaskScheduler, un ordonnanceur associé à chaque job, qui décide de l’ordre d’exécution des tâches et interagit avec le RessourceManager afin d’obtenir des Workers pour exécuter les tâches. L’interaction entre le TaskScheduler, l’application et les autres éléments de la couche Service est résumée en Figure 4.4.

Il faut noter que la classe TaskScheduler est extensible et personnalisable. Par défaut CloudFIT fournit un ordonnanceur simple, mais celui-ci peut être remplacé par des ordonnanceurs

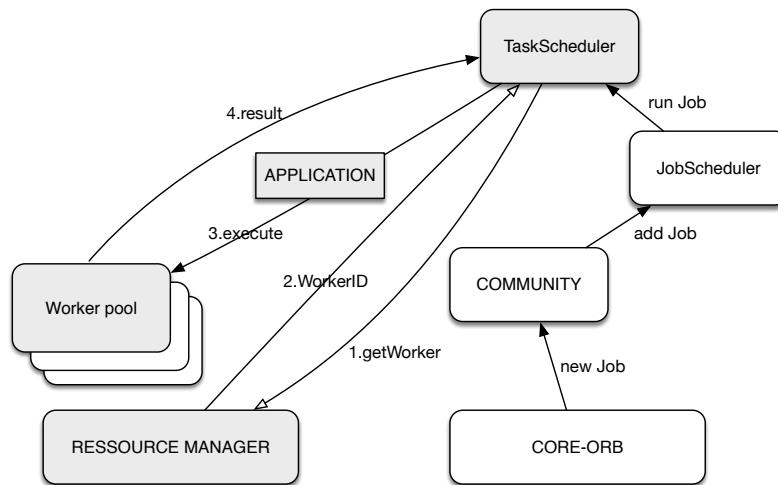


FIGURE 4.4 : Diagramme simplifié de l'interaction entre les éléments lors du lancement d'un job et de ses tâches

plus élaborés. L'ordonnanceur par défaut fait juste une redistribution aléatoire des tâches, une technique simple qui réduit le risque de travail en doublet entre les noeuds. 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 noeud pour minimiser le temps d'accès aux données.

3.3 Communication

La section précédente illustre la structuration et l'interaction des modules dans une machine. Cependant, une plate-forme de calcul distribué se doit de garantir les échanges entre les différents noeuds 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 plate-forme, 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 noeud 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. Ce message est envoyé à travers le réseau, grâce aux mécanismes de diffusion de l'overlay P2P ou, le cas échéant, grâce à une diffusion de type *best-effort*.

Comme spécifie l'objectif [R3], nous devons garantir qu'une application sera déployée par la plate-forme elle-même car il serait très contraignant pour l'utilisateur de placer l'application sur chaque noeud. Afin de répondre à ce besoin, nous avons choisi d'utiliser le stockage DHT habituellement associé aux overlays P2P. En effet, le DHT offre à tous les noeuds de l'overlay un accès réseau à des objets et fichiers, du moment où ces noeuds connaissent la clé de ressources associée à ces fichiers. 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 JobScheduler récupère ce fichier et extrait ses classes, qui seront chargées grâce à un *classloader*.

Le stockage DHT peut aussi être utilisé pour la mise à disposition de données d'entrée pour les applications, comme par exemple dans le cas d'une application big data. 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 TaskScheduler 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*.

Lorsqu’une tâche est lancée, un message est envoyé aux noeuds de la Community indiquant le ID du job et de la tâche. Ceci permet aux autres TaskScheduler de savoir qu’une autre machine est en train de calculer cette tâche et ainsi réduire le travail en doublon : ces tâches sont marquées comme *STARTED_DISTANT* et passent à la fin de la file d’exécution. Une tâche marquée ainsi ne sera exécutée que lorsque toutes les tâches *NEW* ont été épuisées et, bien sûr, si aucun message n’est venu indiquer que la tâche a été complétée. En effet, le TaskScheduler 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, si stockés dans la DHT ou dans une ressource externe).

La mise en place du mécanisme de calcul des tâches en cours de résolution lorsqu’il ne reste plus de tâches non calculées permet d'accélérer la terminaison du calcul dans certains cas : si les tâches qui restent en cours de calcul sont placées sur des nuds dont la capacité de calcul est faible, elles seront recalculées sur des nuds plus puissants qui pourront en fournir le résultat plus rapidement.

En plus de mettre à jour les autres noeuds, ces échanges de messages ont aussi le rôle d’alerter un nouveau noeud qui rejoint le réseau. En voyant passer des messages de type "task completed", les noeuds peuvent demander à un voisin de les transmettre le message avec la description du job. Son TaskManager va donc marquer les tâches comme *DISTANT* et procéder à leur récupération grâce à des requêtes spécifiques. Ce mécanisme permet ainsi de garantir l’intégration des noeuds dans un environnement volatile et d’assurer la pérennité des résultats. En effet, il suffit qu’une machine résiste 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 toutefois à la disposition de toute application ou noeud qui souhaite accéder à ses résultats.

4 Calcul Multi-échelle et le Fog

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 sur la frontière la plus proche 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 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 doit être enrichi par l’utilisation du concept du calcul multi-échelle [106, 107] associé à des techniques de *clustering*. En effet, le regroupement des ressources sous la forme de *clusters* est une manière efficace pour organiser les couches de calcul multi-échelle et ainsi fournir une base de coordination pour le déploiement efficace des services.

4.1 Clustering

Plusieurs approches de clustering sont proposées dans la littérature [70] 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, choisissant un *cluster-head* en fonction de leurs identités uniques (ID). Malheureusement, ces métriques sont insuffisantes pour assurer le clustering dans un scénario hétérogène, vu qu'elles ne permettent pas d'exprimer les besoins du calcul multi-échelle. Au contraire, nous devons permettre le regroupement des nœuds selon différentes stratégies de manière à co-localiser les données et les ressources nécessaires pour les services de proximité mais aussi faire des ponts pour relier les dispositifs proches et ceux plus éloignés, même jusqu'au cloud. Ainsi, afin de s'adapter à l'hétérogénéité des environnements pervasifs et l'IoT, les métriques de clustering doivent être inclure des informations de contexte telles que la proximité, les capacités informatiques et de stockage des appareils, la fiabilité et même le niveau d'autorisation/confiance des nœuds collaborateurs.

Dans le cas de CloudFIT, ce clustering peut être mis en œuvre grâce aux *communautés*, en créant des sous-ensembles de nœuds qui peuvent être adressés séparément et donc utilisés pour répartir/cloisonner les opérations. Bien entendu, un nœud peut appartenir à plusieurs communautés, permettant à des données et des tâches de circuler entre les différentes couches multi-échelle, comme illustrée en Figure 4.5. 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.

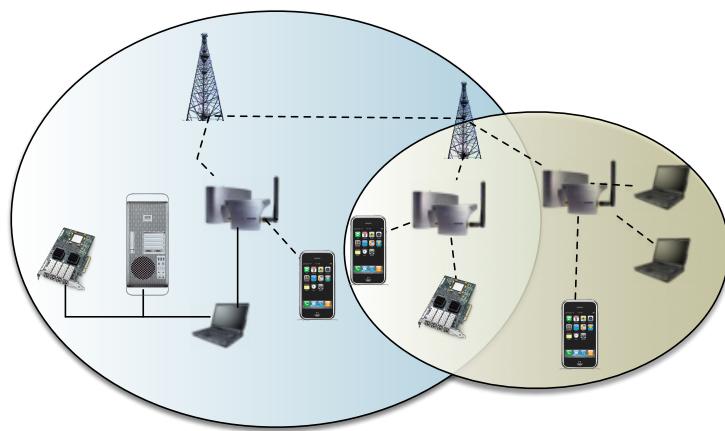


FIGURE 4.5 : Exemple de communautés interconnectées dans une plate-forme multi-échelle

4.2 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, et les plates-formes telles que Apache Hadoop se sont illustrées par leur capacité d'optimiser l'accès aux données en plaçant les tâches préférentiellement là où les données sont présentes (grâce au concept de la *data locality*).

Dans le passé [124] nous avons déjà mené des expérimentations avec CloudFIT démontrant que des performances similaires ou supérieures à celles de Apache Hadoop pourraient être

atteintes avec un système P2P, comme illustré en Fig. 4.6. Bien que ces résultats ont été positifs, nous avons observé deux points qui pourraient être encore améliorés : la surcharge de gestion des données sur un noeud et la prise en charge de la *data locality*.

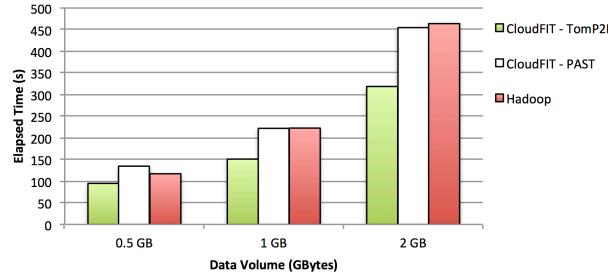


FIGURE 4.6 : Comparaison des temps d’exécution de WordCount avec CloudFIT et Hadoop

Dans le premier cas, nous avons constaté un fort écart de performances d'accès aux données lorsque nous déployons CloudFIT dans un environnement hétérogène. Ces écarts de performance sont en effet une combinaison de la vitesse d'accès aux données et de la surcharge de gestion des systèmes de stockage (voir aussi les limitations physiques de stockage), et affectent notamment les dispositifs de faible capacité. 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 coeurs de calcul). Afin de contourner cette limitation, nous avons modifié la couche de stockage afin que les noeuds puissent choisir d'agir seulement en tant que clients distants. Ces noeuds peuvent donc interroger le service de stockage P2P via le réseau mais ils ne sont plus obligés à gérer le stockage, réduisant leur surcharge et aussi leur utilisation du disque.

Por ce qui est de la prise en charge de la *data-locality*, c'est un problème plus général qui affecte la plupart des architectures de stockage P2P. En effet, les API de stockage P2P sont souvent basés sur les tables de hachage distribuées (DHT). Ces DHTs 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 [140], rendant difficile l'optimisation des transferts réseau.

Une première approche que peut facilement être mise à l'oeuvre consiste à instruire l'ordonnanceur de tâches (*TaskScheduler*) 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 locaux, on peut espérer augmenter la performance global 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 noeuds responsables par le stockage et indexation des ressources sont définis par la clé de hachage de la ressource. La Figure 4.7 illustre le cas du routage d'un message dans l'overlay Pastry [109, 25], mais le même exemple peut être utilisé pour la localisation des ressources dans la DHT PAST [108] car dans ce système les clés des ressources et les clés des noeuds se superposent. Finalement, même avec de la réplication, il y a le risque que dans un grand réseau les ressources ne se trouvent sur aucun des noeuds de calcul. Ceci nous amène à l'élaboration d'une stratégie pour renforcer la proximité des données, grâce à un calcul personnalisé de la clé de localisation des ressources.

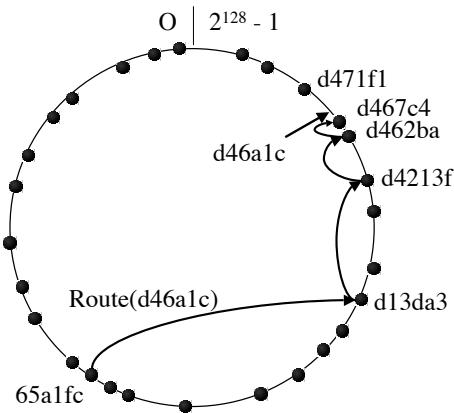


FIGURE 4.7 : Exemple de routage d'un message dans l'overlay Pastry [25]

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 unique, TomP2P identifie les ressources par quatre clés différentes $\{k_l, k_d, k_c, k_v\}$, selon l'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 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 les mêmes, mais peuvent être différentes par exemple en cas de collision (deux ressources avec la même clé de localisation).

La clé de domaine est liée à un mécanisme d'authentification simple de TomP2P, son but étant de renforcer le cloisonnement des données des différents clients (cette authentification doit 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 un 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 Fig. 4.8 montre l'exemple de cette cartographie en calculant la clé de localisation d'une ressource r_3 par rapport à une communauté $Comm_1$.

Cette cartographie augmente la probabilité que la copie primaire d'une donnée se trouve parmi les membres de la communauté, sans pour autant empêcher la réPLICATION des données

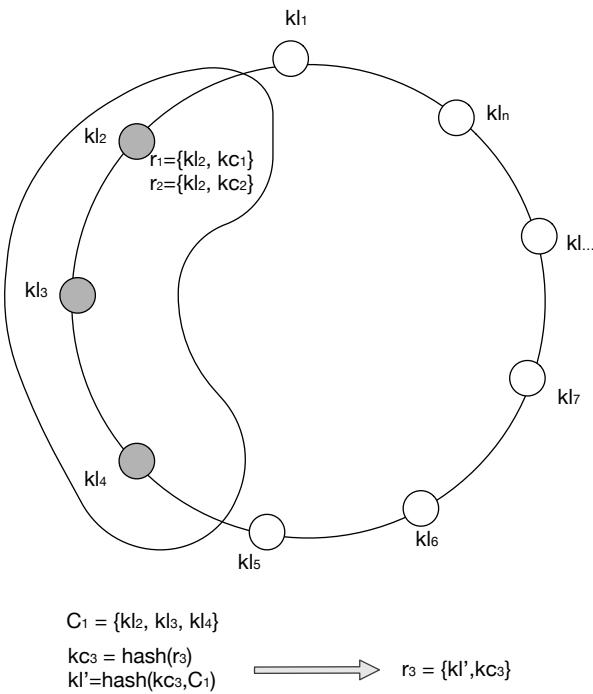


FIGURE 4.8 : Cartographie des ressources renforçant la data-locality

sur d’autres nœuds. 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.

5 Exemples d’Utilisation de CloudFIT

En tant que plate-forme expérimentale pour le calcul distribué et le fog computing, CloudFIT est en constante évolution. Cela n’empêche pas son utilisation comme plate-forme 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 plate-forme MapReduce pour les environnements pervasifs. Le deuxième exemple démontre une utilisation "production", où CloudFIT est utilisé pour exécuter un workflow destiné aux sciences de l’atmosphère (et qui a servi de base de travail pour la proposition du projet CAPES-Cofecub MESO).

5.1 L’application WordCount

Le projet STIC-AmSud PER-MARE avait pour but le développement de stratégies pour le déploiement d’applications Map-Reduce sur des environnements pervasifs. Si l’un des volets du projet a été celui d’adapter Apache Hadoop, l’autre volet consistait à utiliser CloudFIT en tant que plate-forme de calcul distribuée. Si dans l’article présenté à CLIoT 2015 [122] nous nous sommes concentrés sur la performance de CloudFIT (voir aussi la Figure 4.6), le travail présenté à CN4IoT [124] analysait l’exécution de CloudFIT par rapport à la volatilité et l’hétérogénéité des ressources.

5.1.1 Impact de la volatilité

Cette première expérience illustre le déploiement d'une application Map-Reduce simple (le WordCount) sur un corpus de textes faisant de 1GB de données et réparti en blocs uniformes de 64 MB. Cette répartition vise à reproduire le comportement de Hadoop, qui lui aussi stocke les données par blocs de 64MB. Aussi afin de rendre la lecture plus simple, les noeuds 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 4.9 présente le diagramme de Gantt pour une exécution sans incidents. Nous pouvons observer ainsi le déploiement des tâches *map* (lesquelles ont des temps d'exécution variable selon le nombre de mots dans les documents), plus une tâche *reduce* qui s'exécute à la fin.

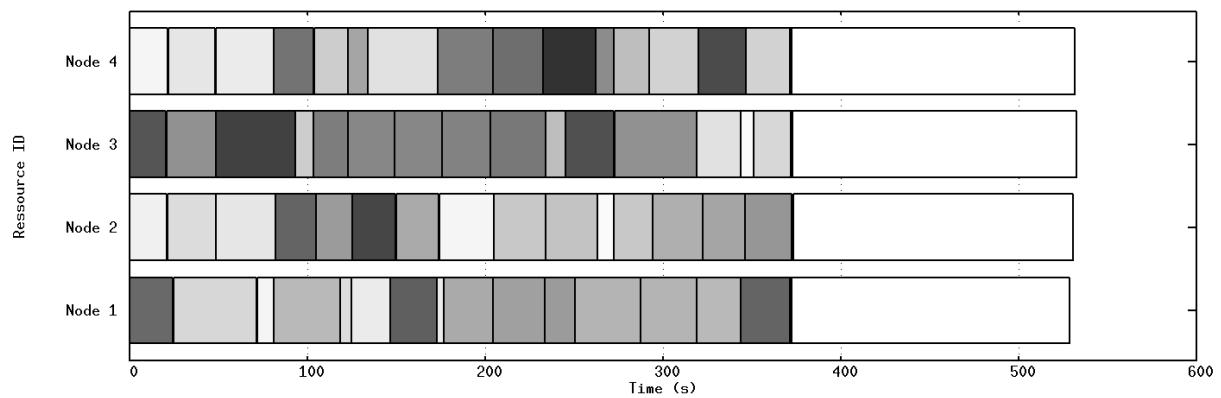


FIGURE 4.9 : Exécution de WordCount sur un cluster uniforme (1 GB de données en blocs de 64MB)

Nous pouvons aussi avoir un aperçu du mécanisme d'ordonnancement distribué par défaut de CloudFIT, déjà discuté en Section 3.3. En effet, lorsque la liste de tâches est reçue par le TaskScheduler, celle-ci est réordonnée de manière aléatoire. L'ordonnanceur choisit ainsi la première tâche disponible (marquée "NEW") et avertit les autres noeuds 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 noeud peut lancer des tâches spéculatives parmi celles marquées "STARTED_DISTANT".

Grâce à ces échanges, il est aussi possible de compléter les tâches initiées par les noeuds en défaillance ou bien lancer des tâches spéculatives afin d'aider les noeuds trop lents. Pour la même raison, un noeud qui vient de rejoindre une communauté CloudFIT non seulement est mis au courant des tâches complétées et en cours comme peut se lancer au calcul des tâches encore disponibles. La Figure 4.10 représente ainsi une situation où un noeud tombe en panne, avec la subséquente reprise des tâches par les autres noeuds. La Figure 4.11 va au delà de cette situation en rajoutant un nouveau noeud, qui récupère l'état actuel des tâches et peut ainsi contribuer avec l'effort de calcul.

5.1.2 Impact de l'hétérogénéité

Cette deuxième expérience vise l'observation de CloudFIT dans un environnement hétérogène. Pour cela, nous avons interconnecté quatre noeuds avec des spécifications assez différentes (cf. le Tableau 4.1). Comme dans l'expérience précédente, nous limitons le nombre de coeurs (Workers) sur les machines pour rendre la visualisation plus simple.

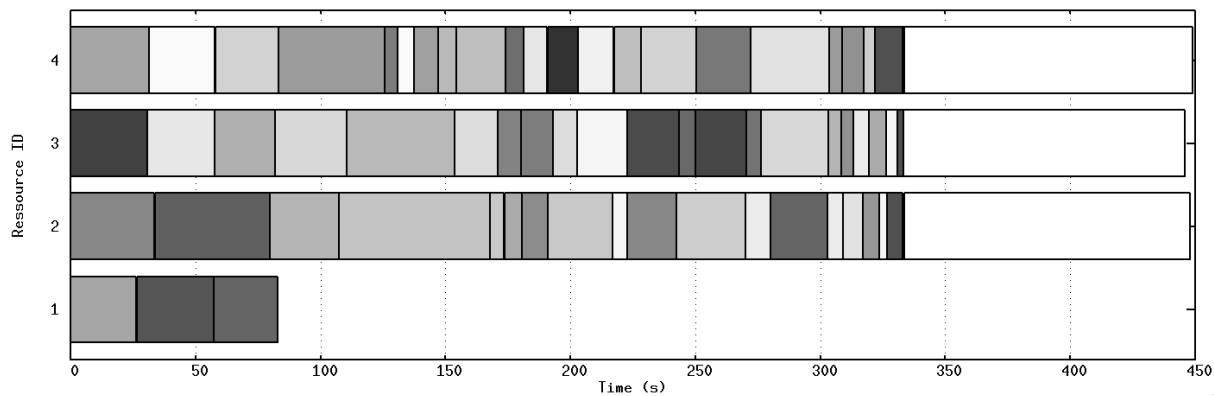


FIGURE 4.10 : Exécution de WordCount execution lorsqu'un noeud disparaît (1 GB de données en blocs de 64MB)

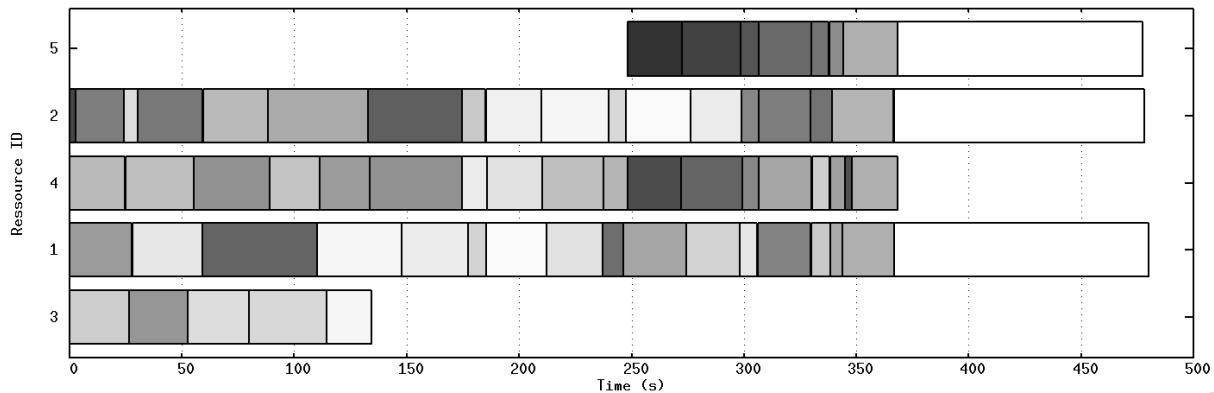


FIGURE 4.11 : Exécution de Wordcount lorsqu'un noeud rejoint la communauté après la défaillance d'un autre noeud (1 GB de données en blocs de 64MB)

| Type de Noeud | 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 4.1 : Spécification des noeuds du cluster pervasif

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. Cette multiplication de tâches avec un coût individuel plus réduit rend aussi possible la participation des noeuds avec moins de puissance de calcul. La Figure 4.12 affiche 1 diagramme de Gantt pour une exécution de ce scénario.

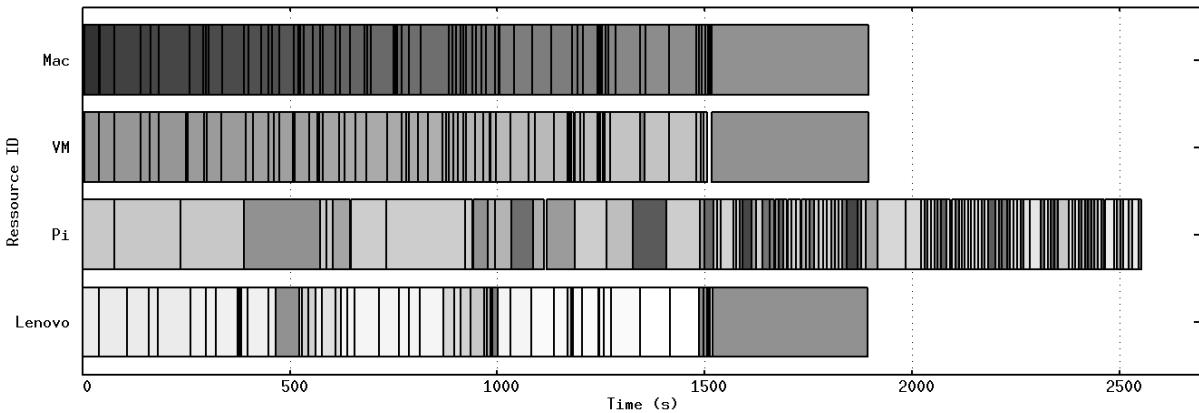


FIGURE 4.12 : Exécution de Wordcount dans un cluster hétérogène (512MB en blocs de 2MB)

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 noeuds (voir la longueur des tâches dans la première partie de l'exécution). De plus, ce noeud est tellement surchargé qu'il perd plusieurs messages de mise à jour et ne détecte pas la fin de la phase map. Comme résultat, il essaye vainement d'exécuter toutes les tâches juste pour se rendre compte que leur résultat est déjà dans la DHT, la raison pour la petite durée des tâches vers la fin (cette vérification préalable était prévue pour éviter le travail en doublon et les risques de corruption des données).

Au lieu de freiner notre intérêt pour 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 aussi 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 des stratégies de collecte de contexte pour un meilleur ordonnancement et aussi les méthodes d'optimisation du stockage, lesquels nous avons traité dans la section précédente.

5.2 Détection d'Événements Sécondaires de la Couche d'Ozone

La découverte du trou d'Ozone de l'Antarctique [44] 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 [117][110]. 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

[75][89] ou bien à cause du transit de masses d’air détachées du vortex polaire et donc avec une faible concentration d’Ozone. Ce dernier cas est appelé "les 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, causant ainsi une réduction temporaire de la colonne totale d’Ozone (*Total Column Ozone - TCO*) sur des zones qui sont souvent habitées [102][137][88]. Comme résultat, des niveaux élevés de radiation ultraviolette nocive atteignent la surface [23], 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 ultraviolette mesurée sur le sud du Brésil [55]. Ces événements secondaires de l’Ozone sont régulièrement observés sur des zones peuplées en moyenne latitude, comme par exemple en Amérique du Sud [76][100], en Afrique du Sud [114][116], à la Nouvelle Zélande [19] mais aussi sur l’Île de la Réunion [130].

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 [89]. En effet, la plupart des modèles climatiques se limitent aux couches inférieures de l’atmosphère, notamment celles liées à aux prévisions météorologiques, et n’exploront pas les interactions avec les couches supérieures comme celle où se trouve la couche d’Ozone. Plus récemment, un modèle obtenu par Vaz Peres [98] a permis une certaine compréhension de ces phénomènes. En se concentrant sur les données d’épisodes de 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 extraire des modèles plus précis, notamment dans le but d’effectuer des prévisions de l’occurrence de ces événements secondaires de l’Ozone.

L’utilisation de techniques du big data est essentiel car les données sur la couche d’Ozone s’accumulent d’année en année. Par exemple, chaque année d’observations de l’équipement TOMS/OMI placé dans les satellites de la NASA satellite représente plus d’1GB de données brutes qui doivent être préparés et analysés. Seulement les observations des satellites TOMS/OMI remontent à 1978, et on peut rajouter d’autres sources de données satellites telles que les satellites de l’ESA et des observations effectuées au sol.

5.2.1 Identification des Événements Secondaires de l’Ozone avec CloudFIT

Si dans un premier temps l’usage d’une plate-forme de pourrait être envisagée, notre attention s’est portée sur CloudFIT car celui-ci a l’avantage d’exploiter les ressources 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 mais aussi sur le type d’analyse effectuée : la détection d’événements passés, utilisée pour améliorer les modèles, est assez simple, alors qu’une analyse plus poussée prenant en compte les corrélations entre les événements et les courants atmosphériques ou une analyse prédictive peuvent s’avérer bien plus demandeurs de ressources. L’utilisation de CloudFIT permet la création d’une plate-forme de calcul élastique et le développement d’une base logicielle capable de s’exécuter sur différents types de machines.

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. **Pré-traitement des données** - transformation des données brutes OMI

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 pré-traitement des données est nécessaire car les données brutes fournis par l'équipement TOMS/OMI sont dans un format difficile à utiliser. 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 degré, alors que d'autres sources de données ont parfois des grilles plus détaillées (par exemple, avec 0.25 degré d'arc entre chaque mesure).

La détection d'un événement secondaire de l'Ozone est liée à l'observation d'une chute anormale de la concentration de l'Ozone. Pour cela, il faut calculer la moyenne et la variation (écart-type) des dernières mesures, pour chaque coordonnée analysée. Cette procédure est détaillée dans la section suivante. 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 une alerte peut être envoyée, indiquant aux autorités sanitaires qu'une risque lié à la radiation UV menace 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 décisions. Nous avons donc établi un workflow qui peut être transcrit comme un enchaînement de jobs CloudFIT, comme illustré en Figure 4.13. Par conséquent, les quatre premiers jobs (Pre-process, Filtering, Time-series et Detection) peuvent même être exécutés sur des communautés CloudFIT différentes : des équipements entrée de gamme regroupés en Community C1 peuvent être utilisés pour pré-traiter et stocker les données des satellites mais aussi ceux des spectre-photomètres Brewer ou Dobson installés au sol. Ces 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. Cette étape inclut le filtrage et l'analyse des séries temporelles, demandant les ressources de calcul plus importants offerts par la communauté C2. La Figure 4.13 inclut aussi d'autres communautés (C3 et C4) qui pourraient être déployées séparément afin d'effectuer d'autres activités de détection et prévision (ces étapes n'ont pas été implémentées). Cette organisation répond aussi aux principes du calcul multi-échelle et du fog computing.

5.2.2 Pré-traitement 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 obtenus par les satellites. Ces données sont présentées selon le format illustré en Figure 4.14(a), ce qui n'est pas vraiment adapté à l'utilisation direct pour nos calculs. Chaque fichier contient une entête avec des informations sur le fichier (date, les coordonnées du grillage, le pas), suivie des mesures pour chaque latitude (indiquée à

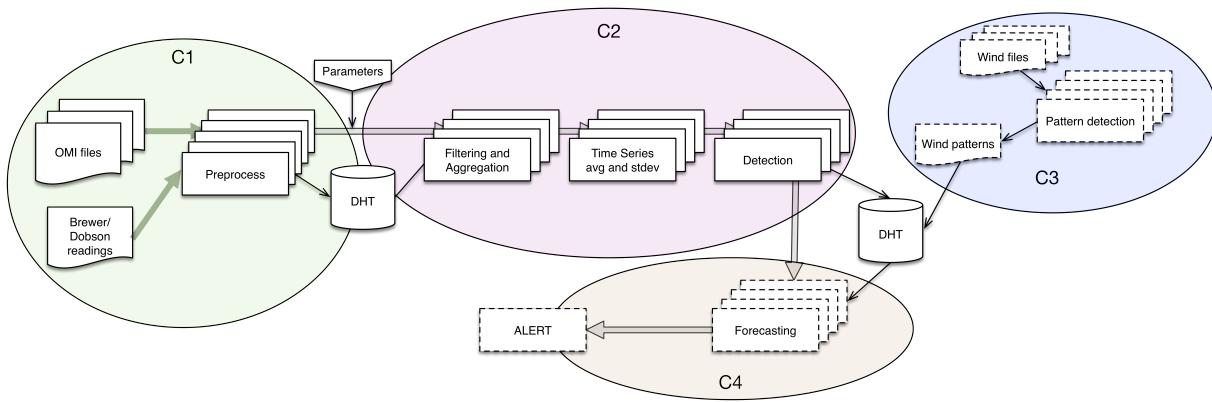


FIGURE 4.13 : Organisation des activités dans un réseau CloudFIT

la fin de la ligne), et cela pour toutes les longitudes couvertes. Chaque mesure est exprimé 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 4.14(a) ont la valeur 280, les coordonnées (-89.5,-178.5) ont aussi la valeur 280, et ainsi de suite.

FIGURE 4.14 : Fichier brute OMI Ozone (a) et sa représentation JSON (b)

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 pré-traiter ces fichiers et les stocker sur la DHT en tant que objets JSON, selon le template présenté en Figure 4.14(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 pré-traitement 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 mais aussi l'ajout de nouvelles entrées.

5.2.3 Analyse des séries temporelles et la détection des OSE

Comme indiqué précédemment, les OSE peuvent être détectés par des réductions anomalies de la colonne totale d'Ozone alors que cela n'est pas directement lié à l'expansion du trou d'Ozone Antarctique. Cette détection est faite par la comparaison entre 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 moyenne et la perception des événements. En effet, on ne peut même pas utiliser la moyenne annuelle car la concentration d'Ozone varie saisonnièrement (dans l'hémisphère sud elle est plus basse à l'Automne et plus haute au Printemps).

L'approche utilisé par Perez et al.[98] considérait la moyenne historique mensuelle, i.e., la moyenne historique de chaque mois des années enregistrées. De cette forme, une mesure effectuée le 2 Octobre et l'autre le 30 Octobre seraient comparées à la moyenne historique du mois d'Octobre. Si cela permet déjà la détection de certains OSE, il n'est pas suffisamment précis 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 par 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 est plus proche de la réalité et prend en compte la variation naturelle pour la période étudiée.

Comme pour le pré-traitement 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 4.15) avec un grillage avec un pas d'1 degré, 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 4.1. 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.

$$Detection(v) = \begin{cases} \text{True} & \text{if } v < (\text{average} - 1.5 \times \text{stdev}) \\ \text{False} & \text{otherwise} \end{cases} \quad (4.1)$$

5.2.4 Résultats Préliminaires

Afin de valider l'implémentation, nous avons comparé les données de Peres et al.[98] avec les résultats obtenus à partir du workflow CloudFIT. L'Comme attendu, notre implémentation permet d'observer la progression du front OSE entre le 18 et le 22 Octobre 2013 (Figure 4.15). 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).

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 4.16, 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. Ces résultats encouragent la poursuite de l'étude de la corrélation entre les OSE et la vorticité

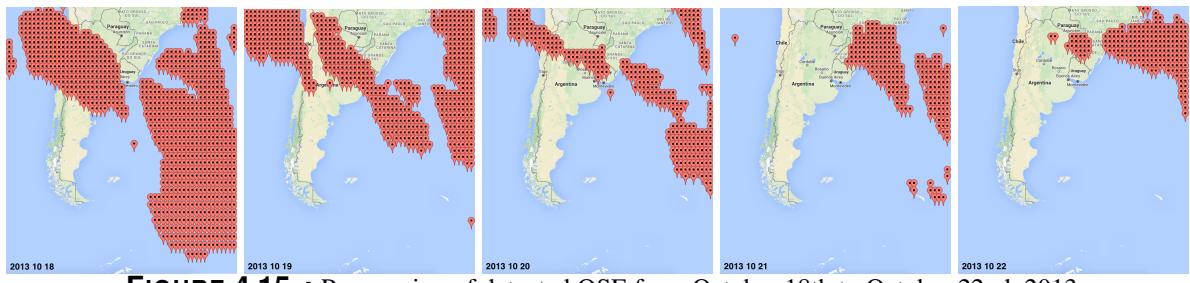


FIGURE 4.15 : Progression of detected OSE from October 18th to October 22nd, 2013

potentielle.

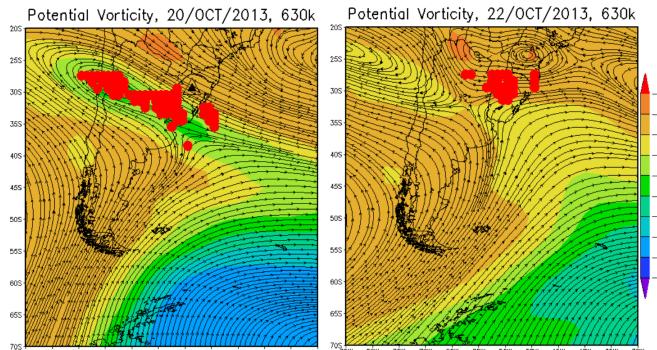


FIGURE 4.16 : Superposition des OSE identifiés sur Santa Maria, Brésil (29.68 S, 53.81 W) et les cartes de la vorticité potentielle

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 coeurs, 8GB RAM - et un Dell Precision T5610 - 2x Intel Xeon E5-2620, 12 coeurs, 32GB RAM). En utilisant l’ensemble des coeurs de calcul de chaque machine, l’analyse a duré 570 secondes. Ceci peut être optimisé en augmentant le nombre de coordonnées traitées par chaque tâche (cela réduit le surcout du démarrage d’une nouvelle tâche) mais aussi on peut rajouter d’autres noeuds au cluster pervasif. Pour comparaison, l’évaluation d’une seule journée avec un Raspberry Pi 2 a nécessité 40 minutes. En dépit de sa faible performance, des dispositifs bas de gamme comme les Raspberry Pi restent une alternative économique et suffisante pour certains types de tâches, comme le pré-traitement et la détection à partir des données journaliers. Si par contre nous avons besoin d’explorer un grand ensemble de données historiques, des machines plus puissantes peuvent être allouées à des communautés CloudFIT dédiées à ces tâches plus lourdes.

6 Bilan et Perspectives

Conclusion et perspectives

Publications personnelles

Articles dans des revues avec comité de lecture

Internationales

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, (accepted, to be published).

Cassales, G. W., Charao, A., Kirsch-Pinheiro, M., Souveyet, C., Steffenel, L.A. "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.

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.

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.

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 Bioinformatics, Elsevier, Vol 42,, pp 48-59. February 2015.

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.

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.

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.

Nasri, W., Steffenel, L.A., Trystram, D. "Adaptive Approaches for Efficient Parallel Algorithms on Cluster-based Systems". International Journal in Grid and Utility Computing, Inderscience, vol 1 n. 2, 2009, pp 99-108.

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.

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

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.

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

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.

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.

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.

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.

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.

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.

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.

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.

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.

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

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.

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

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

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.

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.

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

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.

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.

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.

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, Australia, 19-21 October 2009. pp. 67-72

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

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.

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.

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.

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.

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

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.

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.

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

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.

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.

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.

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.

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.

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.

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

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, Ijuí, Brazil.

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, Ijuí, Brazil.

Steffenel, L.A., Kirsch-Pinheiro, M., "Stratégies Multi-Échelle pour les Environnements Pervasifs et l'Internet des Objets". Proceedings of 11èmes Journées Francophones Mobilité et Ubiquité (Ubimob 2016), July 5, Lorient, France.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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

Ouvrage Scientifique / Livres

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

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

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

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

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 interac-

tions using HPC simulations : getting new therapeutic targets", 9th Ter@tec Forum, Palaiseau, France, July 1-2, 2014.

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.

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

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

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.

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.

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

Conférences invitées

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

CLIoT/CloudWay workshops joint panel "Migrating to Cloud and IoT Solutions : Challenges and Perspectives" , 15 septembre 2015 Moderator : Nabor Mendonça, University of Fortaleza, Brazil Panelists : Pooyan Jamshidi, Imperial College London, U.K., & IC4, Ireland Maria Fazio, University of Messina, Italy Orazio Tormachio, University of Catania, Italy Luiz Angelo Steffenel, Université de Reims Champagne-Ardenne, France

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

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

IUT Villetaneuse (Université Paris 13), mai 2011 "Structure et gestion du Centre de Calcul ROMEO"

Journée ROMEO, 18 mai 2011. "Deploying large scale application with CONFIIT : study on the Quantum Molecular Docking problem".

Universidade Federal de Santa Maria (Brésil), juillet 2010 "RemoteLabz - Environnement virtuel pour l'apprentissage des technologies réseau"

École Supérieure de Sciences et Techniques de Tunis, mai 2009. "CONFIIT : un système pair-à-pair pour le calcul". "GPGPU : du calcul haute performance dans votre machine".

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

Réunion GridExplorer (GdX). Lyon, 21 juin 2005. "Modélisation des Communications pour les Grilles de Calcul".

Séminaires Laboratoire ID-IMAG. Grenoble, 14 novembre 2003. "Tolérance aux Pannes et le Protocole iRBP".

Thèses et dissertations

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

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.

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.

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 Cryptography, using Java and CORBA). Bsc. Dissertation, UFSM, Santa Maria, Brazil, March 1999.

Rapports de recherche

Steffenel, L. A. "D2.1 - First Steps on the Development of a P2P Middleware for Map-Reduce". Deliverable for project STIC-AmSud PER-MARE (13STIC-07), July 2013.

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

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

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.

F. Cappello, P. Owezarski, R. Namyst, O. Richard, P. Vicat-Blanc-Primet, E. Jeannot, L.A. Steffenel, D. Caromel, P. Sens, P. Fraigniaud, C. Cérim, S. Petitot, 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.

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

Bibliographie

- [1] R. Abagyan and M. Totrov. High-throughput docking for lead generation. *Curr. Opin. Chem. Biol.*, 5(4) :375–382, 2001.
- [2] Albert Alexandrov, Mihai Ionescu, Klaus Schauser, and Chris Scheiman. 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] F. H. Allen. Acta crystallogr. b. *The Cambridge Structural Database : a quarter of a million crystal structures and rising*, 58(3) :380–388, 2002.
- [4] Apache. Apache hadoop, 2014. <http://hadoop.apache.org/docs/r2.6.0/index.html>. Last access : November 2014.
- [5] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *Int. J. Ad Hoc Ubiquitous Comput.*, 2(4) :263–277, June 2007.
- [6] Mohammad Banikazemi, Vijay Moorthy, and Dhabaleswar K. Panda. Efficient collective communication on heterogeneous networks of workstations. In *Proceedings of the International Conference on Parallel Processing (ICPP'98)*, pages 460–467, 1998.
- [7] Amotz Bar-Noy and Shlomo Kipnis. Designing broadcasting algorithms in the postal model for message-passing systems. *Math. Systems Theory*, 27(5) :431–452, 1994.
- [8] Luiz Angelo Barchet-Steffenel and Gregory Mounie. Fast tuning of intra-cluster collective communications. In *Proceedings of the Euro PVM/MPI 2004*, LNCS Vol. 3241, pages 28–35, Budapest, Hungary, 2004.
- [9] Luiz Angelo Barchet-Steffenel and Gregory Mounie. Performance characterisation of intra-cluster collective communications. In *Proceedings of the 16th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2004)*, pages 254–261, Foz do Iguacu, Brazil, Sep 2004.
- [10] Mike Barnett, David Payne, Robert van de Geijn, and Jerrel Watts. Broadcasting on meshes with wormhole routing. *Journal of Parallel and Distributed Computing*, 35(2) :111–122, 1996.
- [11] Olivier Beaumont and Loris Marchal. Pipelining broadcasts on heterogeneous platforms under the one-port model. Technical report, LIP, ENS Lyon, France, 2004.
- [12] Olivier Beaumont, Loris Marchal, and Yves Robert. Broadcast trees for heterogeneous platforms. Technical report, LIP, ENS Lyon, France, 2004.
- [13] Olivier Beaumont, Loris Marchal, and Yves Robert. Broadcasts trees for heterogeneous platforms. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2005)*, 2005.

- [14] Massimo Bernaschi and Giulio Iannello. Collective communication operations : Experimental results vs. theory. *Concurrency : Practice and Experience*, 10(5) :359–386, April 1998.
- [15] P. B. Bhat, V.K. Prasanna, and C.S. Raghavendra. Adaptive communication algorithms for distributed heterogeneous systems. *Journal of Parallel and Distributed Computing*, 1999(59) :252–279, 1999.
- [16] P. B. Bhat, C.S. Raghavendra, and V.K. Prasanna. Efficient collective communication in distributed heterogeneous systems. *Journal of Parallel and Distributed Computing*, 2003(63) :251–279, 2003.
- [17] Manav Bhatia, Vishwas Manral, and Yasuhiro Ohara. IS-IS and OSPF Difference Discussion. IETF Internet Draft, January 2006.
- [18] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the 1st MCC Workshop on Mobile Cloud Computing*, MCC ’12, pages 13–16, New York, NY, USA, 2012. ACM.
- [19] E. J. Brinksma, Y. J. Meijer, B. J. Connor, G. L. Manney, J. B. Bergwerff, G. E. Bodeker, I. S. Boyd, J. B. Liley, W. Hogervorst, J. W. Hovenier, N. J. Livesey, and D. P. J. Swart. Analysis of record-low ozone values during the 1997 winter over lauder, new zealand. *Geophysical Research Letters*, 25(15) :2785–2788, 1998.
- [20] Jehoshua Bruck, Ching-Tien Ho, Shlomo Kipnis, Eli Upfal, and Derrick Weathersby. Efficient algorithms for all-to-all communications in multiport message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, 8(11) :1143–1156, November 1997.
- [21] N. Capit, G. da Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounie, P. Neyron, and O. Richard. A batch scheduler with high level components. In *May*, volume 2, pages 776–783, 2005.
- [22] Henri Casanova. Network modeling issues for grid application scheduling. *International Journal of Foundations of Computer Science*, 16(2) :145–162, 2005.
- [23] Claudio Casiccia, Felix Zamorano, and A. Hernandez. Erythemal irradiance at the magellan’s region and antarctic ozone hole 1999-2005. *Atmosfera*, 21(1) :2–12, 2008.
- [24] Guilherme W. Cassales, Andrea S. Charao, Manuele Kirsch Pinheiro, Carine Souveyet, and Luiz A. Steffenel. Bringing context to apache hadoop. In *8th International Conference on Mobile Ubiquitous Computing*, Rome, Italy, 2014.
- [25] M. Castro, P. Druschel, A-M. Kermarrec, and A. Rowstron. One ring to rule them all : Service discovery and binding in structured peer-to-peer overlay networks. In *SIGOPS European Workshop*, France, Sep 2002.
- [26] Marco Cavallo, Lorenzo Cusma, Giuseppe Di Modica, Carmelo Polito, and Orazio Tomarchio. 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.
- [27] Jérémie Chalopin, Emmanuel Godard, Yves Métivier, and Rodrigue Ossamy. Mobile agent algorithms versus message passing algorithms. In *Principle of distributed systems*, volume 4305 of *LNCS*, pages 185–199, France, 2006. Springer.
- [28] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2) :225–267, 1996.

- [29] Quan Chen, Daqiang Zhang, Minyi Guo, Qianni Deng, and Song Guo. Samr : A self-adaptive mapreduce scheduling algorithm in heterogeneous environment. In *Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology*, CIT '10, pages 2736–2743, Washington, DC, USA, 2010. IEEE Computer Society.
- [30] Y. Z. Chen and C. Y. Ung. 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) :199–218, 2001.
- [31] Y. Z. Chen and D. G. Zhi. 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) :217–226, 2001.
- [32] Christina Christara, Xiaoliang Ding, and Ken Jackson. An efficient transposition algorithm for distributed memory computers. In *Proceedings of the High Performance Computing Systems and Applications*, pages 349–368, 1999.
- [33] Anthony Tam Tat Chun. *Performance Studies of High-Speed Communication on Commodity Cluster*. PhD thesis, University of Hong Kong, 2001.
- [34] Cisco. Cisco iox.
- [35] 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.
- [36] Mark Clement, Michael Steed, and Phyllis Crandall. Network performance modelling for PM clusters. In *Proceedings of Supercomputing*, 1996.
- [37] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schausser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. LogP - a practical model of parallel computing. *Communication of the ACM*, 39(11) :78–85, 1996.
- [38] Bronis R. de Supinski and Nicholas T. Karonis. Accurately measuring MPI broadcasts in a computational grid. In *Proceedings of the IEEE International Symposium on High Performance Distributed Computing (HPDC'99)*, August 1999.
- [39] Jeffrey Dean and Sanjay Ghemawat. Mapreduce : Simplified data processing on large clusters. *Commun. ACM*, 51(1) :107–113, January 2008.
- [40] Swarnava Dey, Arijit Mukherjee, Himadri Sekhar Paul, and Arpan Pal. Challenges of using edge devices in iot computation grids. In *Int. Conf. on Parallel and Distributed Systems*, pages 564–569, 2013.
- [41] EW. Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the ACM*, 17 :643–644, 1974.
- [42] Yehia Elkhatib, Barry Porter, Heverson B. Ribeiro, Mohamed Faten Zhani, Junaid Qadir, and Etienne Rivière. On using micro-clouds to deliver the fog. *IEEE Internet Computing*, 21(2) :8–15, 2017.
- [43] 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%202018-09-14.pdf.
- [44] J.C. Farman, G.G. Gardiner, and J.D. Shanklin. Large losses of total ozone in antarctica reveal seasonal clox/nox interaction. *Nature*, 315 :207–210, 1985.

- [45] M. Fazio, A. Celesti, A. Puliafito, and M. Villari. Big data storage in the cloud for smart environment monitoring. *Procedia Computer Science*, 52 :500 – 506, 2015. The 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015).
- [46] O. Flauzac, F. Nolot, C. Rabat, and L.A. Steffenel. Grid of security : a decentralized enforcement of the network security. In IGI Global, editor, *Threats, Countermeasures and Advances in Applied Information Security*, pages 426–443. Manish Gupta, John Walp, and Raj Sharman (Eds.), April 2012.
- [47] Oliver Flauzac, Michael Krajecki, and Luiz-Angelo Steffenel. Confit : a middleware for peer-to-peer computing. *The Journal of Supercomputing*, 53 :86–102, 2010.
- [48] Olivier Flauzac, Michaël Krajecki, and Jean Fugère. CONFIIT : a middleware for peer to peer computing. In M.L. Gravilova, C.J.K. Tan, and P. L’Ecuyer, editors, *The 2003 International Conference on Computational Science and its Applications (ICCSA 2003)*, volume 2669 (III) of *Lecture Notes in Computer Science*, pages 69–78. Springer-Verlag, Montréal, Québec, June 2003.
- [49] E. Gabriel, M. Resch, T. Beisel, and R. Keller. Distributed computing in a heterogeneous computing environment. In *Proc. of the Euro PVM/MPI 1998*, LNCS 1497, pages 180–187. Springer, 1998.
- [50] Pedro Garcia Lopez, Alberto Montresor, Dick Epema, Anwitaman Datta, Teruo Higashino, Adriana Iamnitchi, Marinho Barcellos, Pascal Felber, and Etienne Riviere. Edge-centric computing : Vision and challenges. *SIGCOMM Comput. Commun. Rev.*, 45(5) :37–42, September 2015.
- [51] D. Ghersi and R. Sanchez. Improving accuracy and efficiency of blind protein-ligand docking by focusing on predicted binding sites. *Proteins : Structure, Function and Bioinformatics*, 74(2) :417–424, 2009.
- [52] D. Giganti, H. Guillemain, J.-L. Spadoni, M. Nilges, J.-F. Zagury, and M. Montes. Comparative evaluation of 3d virtual ligand screening methods : Impact of the molecular alignment on enrichment. *J. Chem. Inf. Model.*, 50(6) :992–1004, 2010.
- [53] Grid'5000. Grid 5000, 2013. <https://www.grid5000.fr/>, Last access : July 2014.
- [54] Duncan Grove. *Performance Modelling of Message-Passing Parallel Programs*. PhD thesis, University of Adelaide, 2003.
- [55] R.A Guarnieri, L.F Padilha, F.L Guarnieri, E Echer, K Makita, D.K Pinheiro, A.M.P Schuch, L.S Boeira, and N.J Schuch. 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) :764 – 76, 2004.
- [56] Jayavaradhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot) : A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7) :1645 – 1660, 2013.
- [57] V. Le Guilloux, P. Schmidtke, and P. Tuffery. Fpocket : An open source platform for ligand pocket detection. *BMC Bioinformatics*, 10(1) :168, 2009.
- [58] James Hamilton. Hadoop wins terasort, Jul 2008. <http://perspectives.mvdirona.com/2008/07/hadoop-wins-terasort/>, Last access : Sep 2015.
- [59] Zijiang Hao, Ed Novak, Shanhe Yi, and Qun Li. Challenges and software architecture for fog computing. *IEEE Internet Computing*, 21(2) :44–53, 2017.

- [60] Robert L Henderson. Job scheduling under the portable batch scheduler. In *Proceedings of the IPPS'95 workshop*, LNCS Vol. 949, pages 279–294, 1995.
- [61] C. Hetenyi and D. van der Spoel. Toward prediction of functional protein pockets using blind docking and pocket search algorithms. *Protein Science*, (20) :880–893, 2011.
- [62] R.W. Hockney. The communication challenge for MPP : Intel Paragon and Meiko CS-2. *Parallel Computing*, 20 :389–398, 1994.
- [63] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. The hibench benchmark suite : Characterization of the mapreduce-based data analysis. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, pages 41–51, March 2010.
- [64] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. Zookeeper : Wait-free coordination for internet-scale systems. In *Proceedings of the USENIX Annual Technical Conference*, pages 11–11, Boston, MA, 2010. USENIX Association.
- [65] Lars Paul Huse. Collective communication on dedicated clusters of workstations. In *Proceedings of the European PVM/MPI Users' Group Meeting*, pages 469–476, 1999.
- [66] J. J. Irwin and B. K. Shoichet. Zinc-a free database of commercially available compounds for virtual screening. *J. Chem. Inf. Model.*, 45(1) :177–182, 2005.
- [67] Michael Isard, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar, and Andrew Goldberg. Quincy : fair scheduling for distributed computing clusters. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, pages 261–276, New York, NY, USA, 2009. ACM.
- [68] C. Jaillet and M. Krajecki. Solving the langford problem in parallel. In *Proceedings of ISPDC'04, International Symposium on Distributed and Parallel Computing*, Cork (Ireland), 2004.
- [69] Emmanuel Jeannot and Luiz Angelo Steffenel. Fast and efficient total exchange on two clusters. In *EuroPar'07 - 13th International Euro-Par Conference European Conference on Parallel and Distributed Computing*, LNCS 4631, pages 832–841, Rennes, France, September 2007. Springer.
- [70] Colette Johnen and Fouzi Mekhaldi. 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*, Euro-Par'11, pages 117–129. Springer, 2011.
- [71] S. Lenart Johnsson and Ching-Tien Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Transactions on Computers*, 38(9) :1249–1268, Sep 1989.
- [72] Michelle Jones. Internet of things : Shifting from proprietary to standard. <http://www.valuewalk.com/2014/07/internet-of-things-iot/>.
- [73] Thilo Kielmann, Henri Bal, Sergey Gorlatch, Kees Verstoep, and Rutger Hofman. Network performance-aware collective communication for clustered wide area systems. *Parallel Computing*, 27(11) :1431–1456, 2001.
- [74] Thilo Kielmann, Henri Bal, and Kees Verstoep. Fast measurement of LogP parameters for message passing platforms. In *Proceedings of the 4th Workshop on Runtime Systems for Parallel Programming*, LNCS Vol. 1800, pages 1176–1183, 2000.

- [75] V. W. J. H. Kirchhoff, Y. Sahai, C. A. R. S. Casiccia, B. F. Zamorano, and V. V. Valderrama. Observations of the 1995 ozone hole over punta arenas, chile. *Journal of Geophysical Research : Atmospheres*, 102(D13) :16109–16120, 1997.
- [76] V. W. J. H. Kirchhoff, N.J. Schuch, D.K. Pinheiro, and J.M. Harris. Evidence for an ozone hole perturbation at 30° south. *Athmospheric Environment*, 33(9) :1481–1488, 1996.
- [77] G. Klebe. Virtual ligand screening : strategies, perspectives and limitations. *Drug Discov. Today*, 11(13-14) :580–594, 2006.
- [78] Michaël Krajecki. An object oriented environment to manage the parallelism of the FIIT applications. In V. Malyshkin, editor, *Parallel Computing Technologies, 5th International Conference, PaCT-99*, volume 1662 of *Lecture Notes in Computer Science*, pages 229–234. Springer-Verlag, St. Petersburg, Russia, September 1999.
- [79] K. Arun Kumar, Vamshi Krishna Konishetty, Kaladhar Voruganti, and G. V. Prabhakara Rao. Cash : context aware scheduler for hadoop. In *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, ICACCI ’12, pages 52–61, New York, NY, USA, 2012.
- [80] Jess Labarta, Sergi Girona, Vincent Pillet, Toni Cortes, and Luis Gregoris. DiP : A parallel program development environment. In *Proceedings of the 2nd Euro-Par Conference*, volume 2, pages 665–674, 1996.
- [81] G. Lauro, M. Masullo, S. Piacente, R. Riccio, and G. Bifulco. Inverse virtual screening allows the discovery of the biological activity of natural compounds. *Bioorganic and Medicinal Chemistry*, 20(11) :3596–3602, 2012.
- [82] G. Lauro, A. Romano, R. Riccio, and G. Bifulco. Inverse virtual screening of antitumor targets : Pilot study on a small database of natural bioactive compounds. *Journal of Natural Products*, 74(6) :1401–1407, 2011.
- [83] H. Li, Z. Gao, L. Kang, H. Zhang, K. Yang, K. Yu, X. Luo, W. Zhu, K. Chen, J. Shen, X. Wang, and H. Jiang. Tarfisdock : a web server for identifying drug targets with docking approach. *Nucleic Acids Research*, 34(web) :219–224, 2006.
- [84] J. Li, Qingyang Wang, D. Jayasinghe, Junhee Park, Tao Zhu, and C. Pu. Performance overhead among three hypervisors : An experimental study using hadoop benchmarks. In *Big Data (BigData Congress), 2013 IEEE International Congress on*, pages 9–16, June 2013.
- [85] Pangfeng Liu and Tzu-Hao Sheng. Broadcast scheduling optimization for heterogeneous cluster systems. In *Proceedings of 12th SPAA*, pages 129–136, 2000.
- [86] N. Lynch. *Distributed algorithms*. Morgan Kaufman, 1996.
- [87] Zakaria Maamar, Djamel Benslimane, and Nanjangud C. Narendra. What can context do for web services ? *Commun. ACM*, 49(12) :98–103, December 2006.
- [88] G. L. Manney, R. W. Zurek, A. O’Neill, and R. Swinbank. On the motion of air through the stratospheric polar vortex. *Jounal of the Atmospheric Sciences*, 51 :2973–2994, Oct 1994.
- [89] M. Marchand, S. Bekki, A. Pazmino, F. Lefèvre, S. Godin-Beekmann, and A. Hauchecorne. Model simulations of the impact of the 2002 antarctic ozone hole on the midlatitudes. *Jounal of the Atmospheric Sciences*, 62 :871–884, Mar 2005.

- [90] Fabrizio Marozzo, Domenico Talia, and Paolo Trunfio. P2p-mapreduce : Parallel data processing in dynamic cloud environments. *J. Comput. Syst. Sci.*, 78(5) :1382–1402, September 2012.
- [91] Gabriel Mateescu. A method for MPI broadcast in computational grids. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2005)*, 2005.
- [92] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. Internet of things : Vision, applications and research challenges. *Ad Hoc Networks*, 10(7) :1497 – 1516, 2012.
- [93] J. Moy. OSPF Version 2. RFC 2328 (INTERNET STANDARD), April 1998. Updated by RFCs 5709, 6549, 6845, 6860.
- [94] Salma Najar, Manuele Kirsch Pinheiro, and Carine Souveyet. Service discovery and prediction on pervasive information system. *Journal of Ambient Intelligence and Humanized Computing*, 6(4) :407–423, 2015.
- [95] 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.
- [96] D. Oran. OSI IS-IS Intra-domain Routing Protocol. RFC 1142 (Informational), February 1990.
- [97] M. Parashar and J.-M. Pierson. Pervasive grids : Challenges and opportunities. In K. Li, C. Hsu, L. Yang, J. Dongarra, and H. Zima, editors, *Handbook of Research on Scalable Computing Technologies*, pages 14–30. IGI Global, 2010.
- [98] Lucas Vaz Peres. *Efeito Secundário do Buraco de Ozônio Antártico Sobre o Sul do Brasil*. Msc in Meteorology. Universidade Federal de Santa Maria, Brazil, 2013.
- [99] James Lyle Peterson. *Petri Net Theory and Modeling of Systems*. Prentice-Hall, Englewood-Cliffs, New Jersey, 1981.
- [100] D.K. Pinheiro, N.P. Leme, L.V. Peres, and E. Kall. *Influence of the Antarctic ozone hole over South of Brazil in 2008 and 2009*, volume 1, pages 33–37. National Institute of Science and Technology, 2011.
- [101] Jelena Pjesivac-Grbovic, Thara Angskun, George Bosilca, Graham E. Fagg, Edgar Gabriel, and Jack J. Dongarra. 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.
- [102] Michael Prather and Andrew H. Jaffe. Global impact of the antarctic ozone hole : Chemical propagation. *Journal of Geophysical Research : Atmospheres*, 95(D4) :3473–3492, 1990.
- [103] Arun Ramakrishnan, Davy Preuveneers, and Yolande Berbers. Enabling self-learning in dynamic and open IoT environments. In Elhadi Shakshuki and Ansar Yasar, editors, *The 5th International Conference on Ambient Systems, Networks and Technologies (ANT-2014), the 4th International Conference on Sustainable Energy Information Technology (SEIT-2014)*, volume 32, pages 207–214, 2014.
- [104] Aysan Rasooli and Douglas G. Down. 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, SCC '12*, pages 1284–1291, Washington, DC, USA, 2012. IEEE Computer Society.

- [105] RCBS. Rcsb protein data bank.
- [106] Sam Rottenberg, Sébastien Leriche, Claire Lecocq, and Chantal Taconet. Vers une définition d'un système réparti multi-échelle. In *UBIMOB'12 - 8èmes Journées Francophones Mobilité et Ubiquité*, pages 178–183, 2012.
- [107] Sam Rottenberg, Sébastien Leriche, Chantal Taconet, Claire Lecocq, and Thierry Desprats. Musca : A multiscale characterization framework for complex distributed systems. In *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems*, pages 1657–1665, 2014.
- [108] A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *ACM Symposium on Operating Systems Principles (SOSP'01)*, Banff, Canada, October 2001.
- [109] Antony Rowstron and Peter Druschel. Pastry : Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.
- [110] Murry L. Salby, Evgenia A. Titova, and Lilia Deschamps. Changes of the antarctic ozone hole : Controlling mechanisms, seasonal predictability, and evolution. *Journal of Geophysical Research : Atmospheres*, 117(D10) :n/a–n/a, 2012. D10111.
- [111] Thomas Sandholm and Kevin Lai. Dynamic proportional share scheduling in hadoop. In *Proceedings of the 15th International Conference on Job Scheduling Strategies for Parallel Processing*, JSSPP'10, pages 110–131, Berlin, Heidelberg, 2010.
- [112] Mahadev Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4) :14–23, December 2009.
- [113] A. Segall. Distributed network protocols. *Transaction on Information Theory*, (29) :23–35, 1983.
- [114] N Semane, H. Bencherif, B. Morel, A. Hauchecorne, and R.D. Diab. An unusual stratospheric ozone decrease in southern hemisphere subtropics linked to isentropic air-mass transport as observed over irene (25.5° s, 28.1° e) in mid-may 2002. *Atmospheric Chemistry and Physics*, 6 :1927–1936, 2006.
- [115] Martin Serrano, Danh Le-Phuoc, Maciej Zaremba, Alex Galis, Sami Bhiri, and Manfred Hauswirth. Resource optimisation in iot cloud systems by using matchmaking and self-management principles. In *The Future Internet*, volume 7858 of *LNCS*, pages 127–140. Springer, 2013.
- [116] V. Sivakumar, T. Portafaix, H. Bencherif, S. Godin-Beekmann, and S. Baldy. Stratospheric ozone climatology and variability over a southern subtropical site : Reunion island (21°s; 55°e). *Annales Geophysicae*, 25 :2321–2334, 2007.
- [117] Susan Solomon. Stratospheric ozone depletion : A review of concepts and history. *Reviews of Geophysics*, 37(3) :275–316, 1999.
- [118] A. Steffen, C. Thiele, S. Tietze, C. Strassnig, A. Kämper, T. Lengauer, G. Wenz, and J. Apostolakis. Improved cyclodextrin-based receptors for camptothecin by inverse virtual screening. *Chemistry - A European Journal*, 13(24) :6801–6809, 2007.
- [119] L.A. Steffenel, O. Flauzac, A. Schwertner Charao, P. Pitthan Barcelos, B. Stein, S. Nesmachnow, M. Kirsch Pinheiro, and D. Diaz. 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.

- [120] Luiz Angelo Steffenel. Modeling network contention effects on alltoall operations. In *Proceedings of the IEEE Conference on Cluster Computing (CLUSTER 2006)*, Barcelona, Spain, September 2006. IEEE Computer Society.
- [121] Luiz Angelo Steffenel, Olivier Flauzac, Andrea Schwertner Charão, Patricia Pitthan Barcelos, Benhur Stein, Sergio Nesmachnow, Manuele Kirsch Pinheiro, and Daniel Diaz. 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*, 3PGCIC '13, pages 17–24, Washington, DC, USA, 2013. IEEE Computer Society.
- [122] Luiz Angelo Steffenel and Manuele Kirsch-Pinheiro. CloudFIT, a PaaS platform for IoT applications over pervasive networks. In *3rd International Workshop on Cloud for IoT (CLIoT 2015)*, Taormina, Italy, September 2015.
- [123] Luiz Angelo Steffenel and Manuele Kirsch Pinheiro. Leveraging data intensive applications on a pervasive computing platform : The case of mapreduce. *Procedia Computer Science*, 52 :1034 – 1039, 2015. The 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015).
- [124] Luiz Angelo Steffenel and Manuele Kirsch-Pinheiro. 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, October 2015.
- [125] STIC-AmSud. PER-MARE project, 2014. <http://cosy.univ-reims.fr/PER-MARE>, Last access : July 2014.
- [126] Apache Storm.
- [127] LAM-MPI Team. Lam/mpi version 7. <http://www.lam-mpi.org>, 2004.
- [128] Rajeev Thakur and William Gropp. Improving the performance of collective operations in MPICH. In *Proceedings of the Euro PVM/MPI 2003*, LNCS Vol. 2840, pages 257–267. Springer-Verlag, 2003.
- [129] Chao Tian, Haojie Zhou, Yongqiang He, and Li Zha. A dynamic mapreduce scheduler for heterogeneous workloads. In *Proceedings of the 2009 Eighth International Conference on Grid and Cooperative Computing*, GCC '09, pages 218–224, Washington, DC, USA, 2009. IEEE Computer Society.
- [130] Abdoulwahab Mohamed Tohir, Hassan Bencherif, V Sivakumar, Laaziz El Amraoui, Thierry Portafaix, and N Mbatha. 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*, September 2015.
- [131] Sathish Vadhiyar, Graham Fagg, and Jack Dongarra. Automatically tuned collective communications. In *Proceedings of the Supercomputing 2000*. IEEE Computer Society, 2000.
- [132] Leslie G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8) :103–111, 1990.

- [133] Ovidiu Vermesan, Peter Friess, Patrick Guillemin, Raffaele Giaffreda, Hanne Grindvoll, Markus Eisenhauer, Martin Serrano, Klaus Moessner, Maurizio Spirito, Lars-Cyril Blystad, and Elias Z. Tragos. Internet of things beyond the hype : Research, innovation and deployment. In *Internet of Things - From Research and Innovation to Market Deployment*. River Publishers, 2014.
- [134] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan. Osmotic computing : A new paradigm for edge/cloud integration. *IEEE Cloud Computing*, 3(6) :76–83, Nov 2016.
- [135] R. Wang, X. Fang, Y. Lu, and S. Wang. The pdbsbind database : Collection of binding affinities for proteinligand complexes with known three-dimensional structures. *J. Med. Chem.*, 47(12) :2977–2980, 2004.
- [136] R. Wang, X. Fang, Y. Lu, C.-Y. Yang, and S. Wang. The pdbsbind database : Methodologies and updates. *J. Med. Chem.*, 48(12) :4111–4119, 2005.
- [137] D. W. Waugh, R. A. Plumb, R. J. Atkinson, M. R. Schoeberl, L. R. Lait, P. A. Newman, M. Loewenstein, D. W. Toohey, L. M. Avallone, C. R. Webster, and R. D. May. Transport out of the lower stratospheric arctic vortex by rossby wave breaking. *Journal of Geophysical Research : Atmospheres*, 99(D1) :1071–1088, 1994.
- [138] D. F. Willis, A. Dasgupt, and S. Banerjee. Paradrop : a multi-tenant platform for dynamically installed third party services on home gate- ways. In *ACM SIGCOMM workshop on Distributed cloud computing*, 2014.
- [139] Rich Wolski, Neil Spring, and Chris Peterson. Implementing a performance forecasting system for metacomputing : The network weather service. In *Proceedings of the Supercomputing*, 1997.
- [140] Di Wu, Ye Tian, and Kam-Wing Ng. Aurelia : Building locality-preserving overlay network over heterogeneous p2p environments. In *Proc. of the International Conference on Parallel and Distributed Processing and Applications*, ISPA’05, pages 1–8. Springer, 2005.
- [141] Jan-Jan Wu, Shih-Hsien Yeh, and Pangfeng Liu. Efficient multiple multicast on heterogeneous network of workstations. *Journal of Supercomputing*, 29(1) :59–88, 2004.
- [142] Jiong Xie, Xiaojun Ruan, Zhiyang Ding, Yun Tian, James Majors, Adam Manzanares, Shu Yin, and Xiao Qin. Improving mapreduce performance through data placement in heterogeneous hadoop clusters. In *Parallel and Distributed Processing, Workshops and Phd Forum (IPDPSW)*, 2010.
- [143] Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li. Fog computing : Platform and applications. In IEEE, editor, *Third IEEE Workshop on Hot Topics in Web Systems and Technologies*, pages 73–78, 2015.
- [144] A. B. Yoo, M. A. Jette, and M. Grondona. Slurm : Simple linux utility for resource management. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 44–60, 2003.
- [145] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, OSDI’08, pages 29–42, Berkeley, CA, USA, 2008. USENIX Association.