

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 xx septembre 2016

Composition du jury :

Président du jury : Rapporteurs : Examinateurs : Directeur :

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	12
1.1 Exécution dans les systèmes distribués	12
1.2 Modélisation des Communications	13
1.3 pLogP	14
1.4 Exemple d'utilisation : modélisation d'un Broadcast : <i>MPI_Bcast</i>	15
1.5 Validation des modèles	17
2 Modélisation de MPI_Broadcast dans une Grille	18
2.1 Évaluation pratique	24
2.2 Considérations sur l'opération MPI_Bcast	26
3 Bilan et Perspectives	27
2 L'hétérogénéité des Données	29
1 L'Hétérogénéité des Données - la grille GRAPP&S	29
1.1 Introduction	29
1.2 État de l'Art	30
2 L'Architecture GRAPP&S	32
2.1 Définitions	32
2.2 Communication et les réseaux overlay	33
2.3 Éléments de l'Architecture GRAPP&S	33
2.4 Communicator (<i>c</i>)	33
2.5 Ressource manager (<i>RM</i>)	34
2.6 Data manager (<i>DM</i>)	34
3 Gestion de la Communauté	34
3.1 Gestion des Nœuds	36
3.1.1 Connexion d'un nœud	36
3.1.2 Déconnexion d'un nœud	36
3.2 Algorithmes d'élection	37
4 Opérations dans GRAPP&S	38
4.1 Stockage et Indexation	38
4.2 Recherche	38
5 Bilan et Perspectives	40
3 L'hétérogénéité des Ressources de Calcul	41
1 Hétérogénéité des Tâches - application à la recherche en amarrage moléculaire	42
1.1 Travaux proches et méthodologie de parallélisation	42

1.2	Parallélisme Intérieur : décomposition des tâches	43
1.2.1	Décomposition géométrique arbitraire	43
1.2.2	Décomposition Géométrique avec superposition	44
1.2.3	Recherche de cavités	45
1.3	Gestion et déploiement des tâches de calcul	45
1.3.1	Optimisation aux clusters HPC	46
1.4	Évaluation pratique	47
2	Hétérogénéité des Ressources de Calcul	49
2.1	About Hadoop Scheduling	51
2.1.1	Hadoop Framework Architecture	51
2.1.2	Hadoop Schedulers	51
2.2	Related Work	53
2.3	Context-Aware Scheduling	54
2.4	Experiments and Results	56
2.4.1	Execution scenarios	56
2.4.2	Application benchmarks	58
2.4.3	Environment setup and configuration for the controlled experiments	58
2.5	Discussions	65
2.6	Conclusion	67
4	CloudFIT : un intergiciel pour le <i>edge-computing</i> et les réseaux IoT	69
1	Introduction	70
2	État de l'Art	71
3	Calcul Multi-Échelle	72
3.1	Communication, coordination et clustering	73
3.2	Contexte et ordonnancement	75
3.3	Accès aux données	75
Conclusion et perspectives	81	
Publications personnelles	83	
Bibliographie	91	

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, les réseaux informatiques sont 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.

Dans certains cas, cependant, juste la connaissance des profils de performance peut s'avérer insuffisante, notamment lorsque les applications ont des besoins de fiabilité accrus. Ainsi, l'hétérogénéité dans les réseaux doit aussi s'intéresser à l'observation des ressources afin de détecter des éventuelles défaillances, et de les signaler aux applications et systèmes, qui pourront ainsi réagir et prendre des mesures nécessaires pour assurer ses engagements de fiabilité.

Dans cette première partie, la Section 1 présente les travaux qui ont été menés sur la modélisation et l'évaluation des performances de communication des réseaux. La Section ?? présente les travaux sur la découverte de la topologie et la subséquente classification des ressources. Fi-

nalement, la Section ?? présente les travaux sur la détection de défaillances. 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 il existent certains facteurs non déterministes qui 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 [38]. Ainsi, la plus grande difficulté pour la modélisation des performances est la correcte représentation des facteurs liés à la communication entre les processus répartis. Pour cette raison, la plupart des modèles de performance préfèrent décrire les systèmes de la manière la plus simple possible ; la perte de précision est compensée par la simplicité et par la portabilité des solutions à travers diverses situations et environnements.

Si les principes de la modélisation de performance peuvent être trouvés dans des travaux pionniers des années 60 et 70 [69], 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 [30]. 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 [55] 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 [41] 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 [70]. L'obtention des paramètres α et β peut se faire à travers l'utilisation de certains outils comme NWS [91].

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 [6]. 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 prédécesseurs 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.* [27] 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.

1.3 pLogP

Le modèle pLogP (*parameterised LogP*) est une extension du modèle LogP présenté par Kielmann *et al.* [51]. Son objectif est de mieux représenter à la fois des petits et des grands messages sous un modèle unifié.

Comme ses prédécesseurs, 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 [1], 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 $g(m) \geq os(m)$ et $g(m) \geq or(m)$ tient toujours. La relation entre ces différents paramètres est illustrée dans la Figure 1.1.

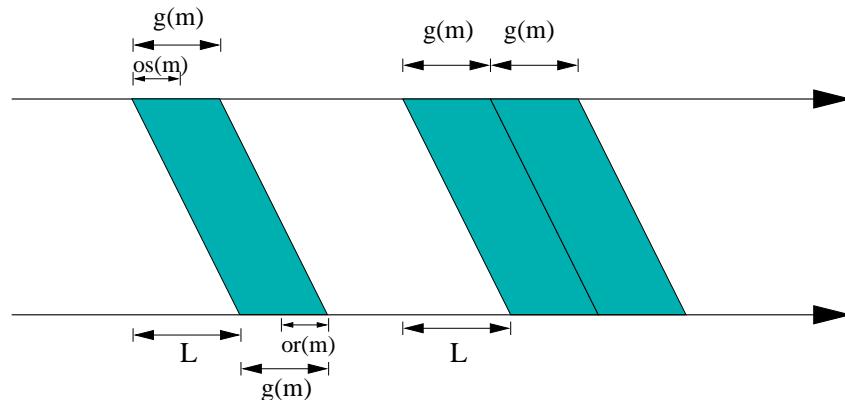


FIGURE 1.1 : Représentation d'une communication avec pLogP

En conséquence de cette nouvelle interprétation du *gap*, les paramètres o_s et o_r ont une importance moins évidente, une fois que leur coût dans un réseau local est souvent recouvert par celui du *gap*. Ainsi, pour représenter le temps nécessaire à la transmission d'un message de taille m entre deux 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 [85]. 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.

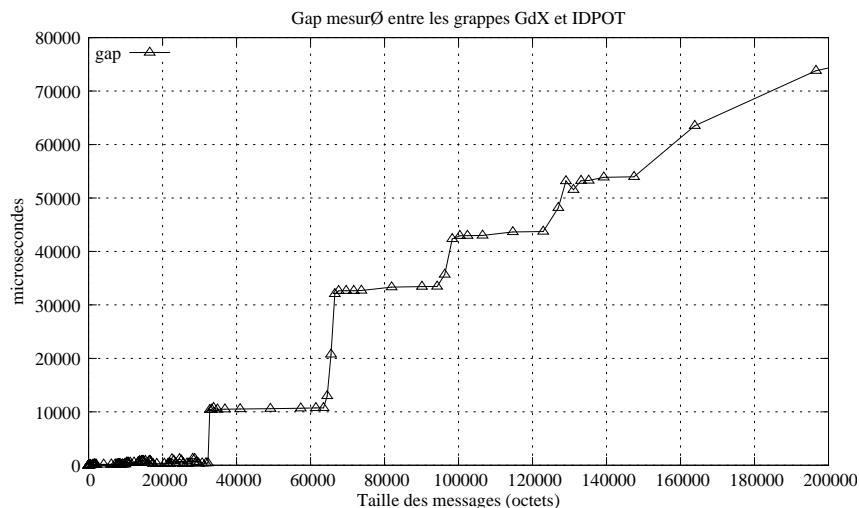


FIGURE 1.2 : Valeurs de gap mesurés entre deux grappes distantes (GdX et IDPOT)

1.4 Exemple d'utilisation : modélisation d'un Broadcast : MPI_Bcast

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.

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 [13]. À l'aide de cet algorithme et des paramètres du réseau, un arbre optimal peut être construit à partir des paramètres du réseau et avec $d, h \in [1...P-1]$ tel que $\sum_{i=0}^h d^i \geq P$ 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 la diversité de formes fixes s'ajoutent aussi différentes techniques d'implémentation. En effet, certaines techniques peuvent s'appliquer à des situations spécifiques, comme par exemple, l'envoi de messages avec des tailles plus importantes ; dans ce cas, un message de *rendez-vous* est envoyé préalablement pour préparer le récepteur, réduisant ainsi le stockage des messages dans des buffers temporaires. Autre technique souvent employée est l'utilisation des primitives de communication non bloquantes pour permettre le recouvrement des communications et du calcul. Il faut néanmoins compter que ces techniques apportent un coût supplémentaire aux opérations : alors que le *rendez-vous* ajoute une étape de communication de plus, la communication non bloquante est obtenue à partir de la copie des données vers des buffers intermédiaires. Dans le premier cas le surcoût est constant et correspond à l'envoi de deux messages de taille zéro, tandis que le deuxième cas a un coût qui dépend de la taille du message, et qui correspond à o_s .

À partir des modèles de coût LogP [27] et pLogP [51] et de travaux comme ceux de Huse [45], Vadhiyar [88] 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. D'autres stratégies, comme les variantes *rendez-vous*, ne sont utilisées qu'à partir d'une taille de message suffisamment grande, ce qui minimise l'impact des étapes de communication supplémentaires dues au *rendez-vous*.

Stratégie	Modèle de Communication
Arbre Plat	$L + (P - 1) \times g(m)$
[h]	Arbre Plat Rendez-vous
	$3 \times L + (P - 1) \times g(m) + 2 \times g(1)$
	Chaîne
	$(P - 1) \times (g(m) + L)$
	Chaîne Rendez-vous
	$(P - 1) \times (g(m) + 2 \times g(1) + 3 \times L)$
	Arbre Binaire
	$\leq \lceil \log_2 P \rceil \times (2 \times g(m) + L)$
	Arbre Binomial
	$\lceil \log_2 P \rceil \times L + \lfloor \log_2 P \rfloor \times g(m)$
	Arbre Binomial Rendez-vous
	$\lceil \log_2 P \rceil \times (2 \times g(1) + 3 \times L) + \lfloor \log_2 P \rfloor \times g(m)$

TABLE 1.1 : Modèles de communication pour le *Broadcast*

Une autre possibilité de construire un *Broadcast* est la composition des chaînes de retransmission [9]. Cette stratégie, possible grâce à la segmentation des messages, présente des avantages importants, comme l'indiquent [51][86][10]. 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 seg-

ments. 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* » [51].

	Stratégie	Modèle de Communication
[h]	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	$\lceil \log_2 P \rceil \times L + \lceil \log_2 P \rceil \times g(s) \times k$
	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 [86]	$(\log_2 P + P - 1) \times L + 2 \times (\frac{p-1}{p}) \times g(m)$

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

1.5 Validation des modèles

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.

Réseau Fast Ethernet

Comme nous pouvons observer dans les Figures 1.3, 1.4 et 1.5, les prédictions des trois stratégies d'implémentation représentent presque fidèlement les résultats expérimentaux obtenus sur un cluster.

Plus spécifiquement, des différences entre les prédictions et les valeurs réelles sont observées surtout dans le cas de la stratégie en Arbre Binomial, où le temps mesuré pour l'envoi de petits messages est plus élevé que celui prévu, et ne suit pas un comportement linéaire par rapport à la taille des messages. Cette variation de performance, observée aussi dans des expériences avec d'autres réseaux, a été l'objet d'analyses précédentes (cette discussion peut être retrouvée dans les articles [7] et [8]) et dont l'origine des retards est due à l'implémentation des politiques d'acquittement TCP sur Linux, qui retarde des messages même si l'option *socket TCP_NODELAY* est activée.

En effet, parfois un seul message à chaque n messages transmis n'est pas acquitté comme il le faut. Cette défaillance du protocole d'acquittement induit un temps supplémentaire nécessaire à la retransmission du message et à son acquittement.

Les variations de performance observées dans le cas de la stratégie en Arbre Binomial sont plus visibles dans la Figure 1.6. 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

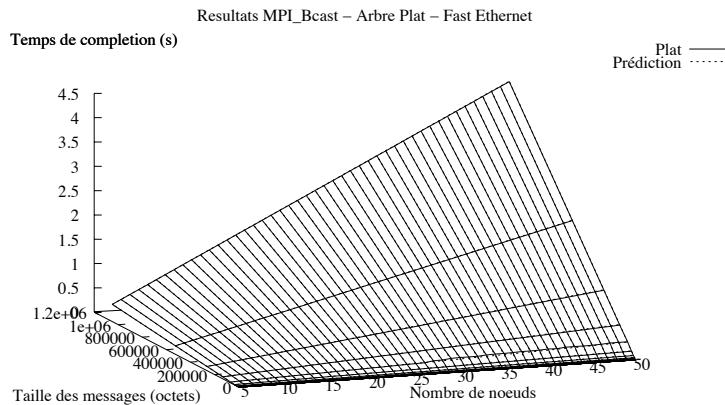


FIGURE 1.3 : Les performances réelles et prédictes pour l’Arbre Plat avec un réseau Fast Ethernet

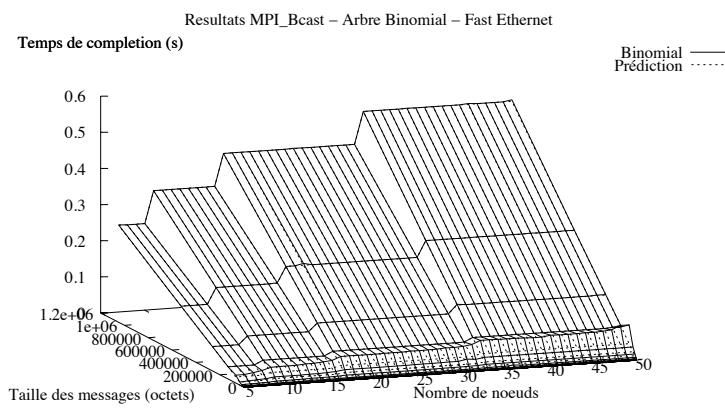


FIGURE 1.4 : Les performances réelles et prédictes pour l’Arbre Binomial avec un réseau Fast Ethernet

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 Modélisation de MPI_Broadcast dans une Grille

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. La détermination du meilleur arbre de diffusion pour un environnement homogène est une tâche relativement facile : le choix de l'arbre dépend surtout des paramètres d'interconnexion.

Cependant, dans le cas d'un réseau hétérogène, ce problème devient bien plus difficile : en effet, l'identification du meilleur arbre de diffusion est un problème NP-complet [14][11, 12][93].

La plupart des travaux dédiés à l'optimisation des communications collectives dans des environnements hétérogènes font référence aux processus communicants. C'est-à-dire, les optimisations sont faites en tenant compte du coût d'interconnexion entre chaque paire de processus concernée par le broadcast. C'est le cas de Banikazemi [5], Bhat [14, 15] et Mateescu [58], qui utilisent différentes stratégies pour construire des arbres de diffusion optimisés.

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

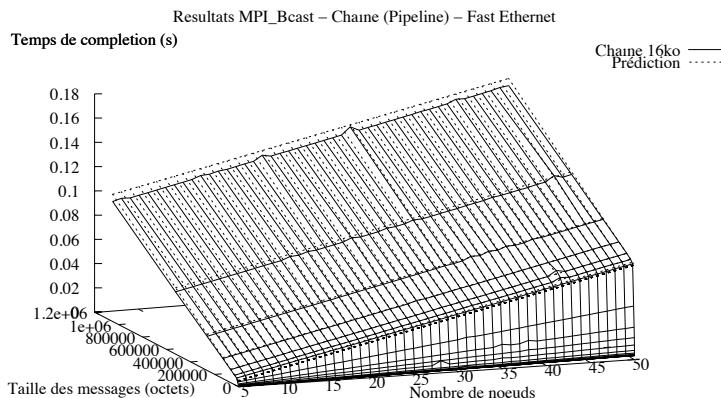


FIGURE 1.5 : Les performances réelles et prédictes pour la Chaîne Segmentée avec un réseau Fast Ethernet

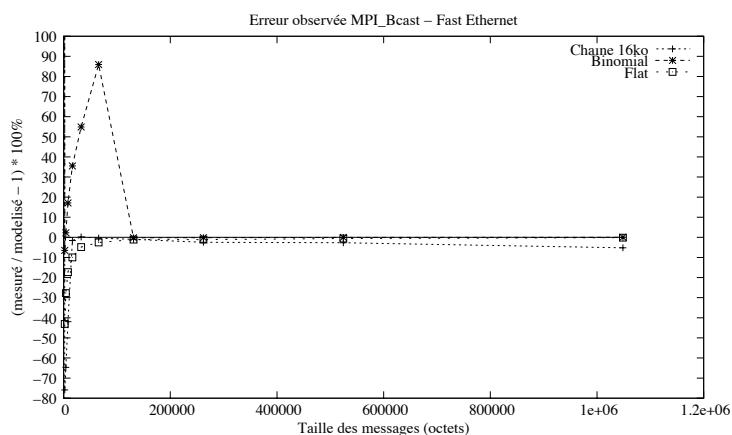


FIGURE 1.6 : L'erreur des prédictions par rapport aux valeurs mesurées, dans un réseau Fast Ethernet

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, l'augmentation massive du nombre de processus dans une grille de calcul peut être encore facilement abordable par les méthodes d'optimisation classiques, i.e., la division des communications en deux couches, l'*inter-grappes* et l'*intra-grappes*.

Cependant, à défaut de leur apport aux algorithmes de Broadcast, ces techniques peuvent 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 [15]. 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 prédition du temps total d'exécution du Broadcast.

Diffusion en Arbre Plat (Flat)

L'heuristique en *Arbre Plat* (approche utilisée par la bibliothèque MagPIe), 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, prédéfini à l'initialisation de MagPIe (normalement, à travers le fichier *magpie_clusters*). 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 planter, 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.* [5]. 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 [54], 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 [15].

Fastest Edge First - FEF

Proposée par Bhat *et al.* [15], 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.* [15] 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 [15] 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**.

Heuristiques "sensibles au contexte des grilles de calcul"

Si parmi les heuristiques présentées précédemment nous avons déjà hiérarchisé les communications en deux niveaux - *inter-grappes* et *intra-grappes* -, jusqu'à présent les fonctions d'évaluation ne tiennent compte que des coûts de transmission entre les coordinateurs des différentes grappes du réseau.

Cependant, 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 les grappes *Grid eXplorer* et *icluster-2* requiert 349 millisecondes, alors que le broadcast de ce même message entre 50 noeuds de la grappe *icluster-2* peut nécessiter jusqu'à 3 secondes selon le réseau et 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. Ce temps est obtenu grâce aux modèles présentés en chapitre ???. Cette notation T_k est équivalente à une notation où un noeud fictif k' est associé à chaque coordinateur k , dont :

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

Alors que les deux notations sont équivalentes, chacune a des avantages : l'adjonction 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 . Si ces deux notations sont facilement transformées, nous avons gardé toutefois 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-grappe* 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.

2.1 Évaluation pratique

Les différentes heuristiques ont été implantées 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. Pour cela, nous utilisons la procédure de découverte de topologie, présentée dans le chapitre ???. Cette procédure de découverte de topologie permet non seulement

le regroupement des noeuds en grappes 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 décrite par la bibliothèque MagPIe. 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. À l'aide de notre outil de découverte de topologie *subnets*, ces machines ont été regroupées en 6 grappes 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.

Le premier résultat à noter est la faible performance de la stratégie Flat, même par rapport à l'implémentation par défaut de LAM-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.

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.

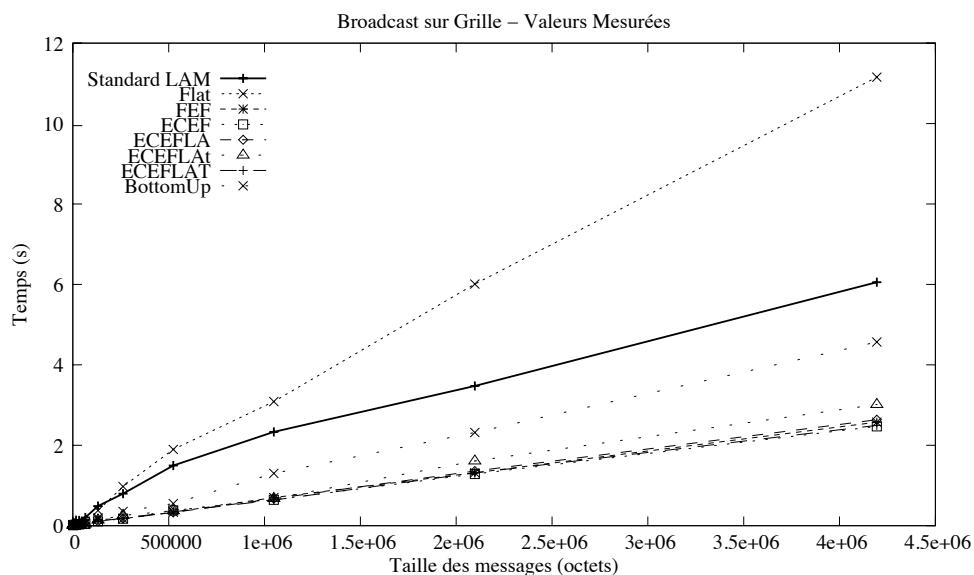


FIGURE 1.7 : Performance du Broadcast sur une grille de 88 machines

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 1.8 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.

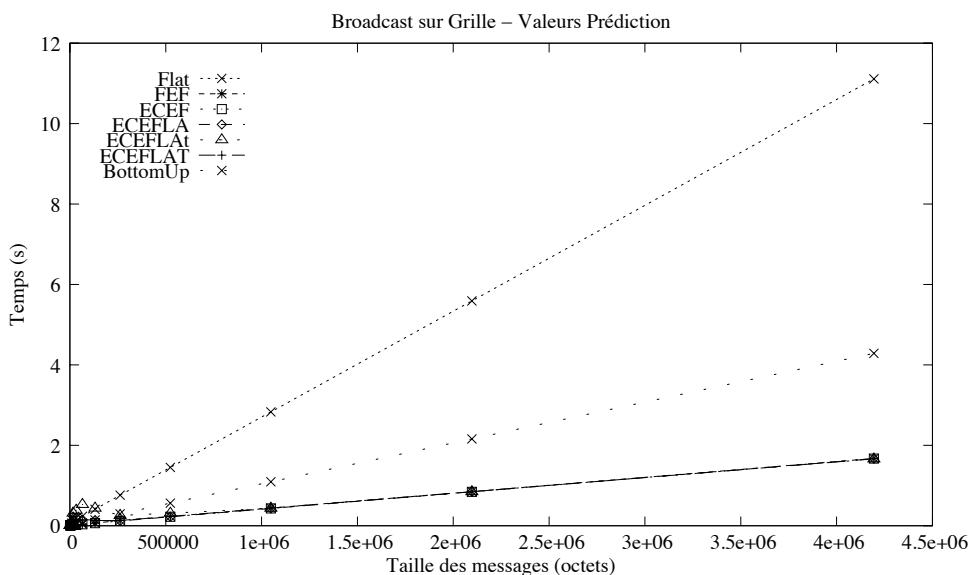


FIGURE 1.8 : Prédictions pour une grille avec 88 machines

2.2 Considérations sur l'opération MPI_Bcast

Si dans un premier temps l'analyse des trois cas d'étude permet la validation des certaines prédictions faites à travers la simulation de différents environnements réseau, son apport le plus important est l'observation du comportement des implémentations des heuristiques.

Les trois situations étudiées ont notamment mis 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. Ceci met en

cause l'efficacité de ces stratégies dans un futur proche. 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 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.

Toutefois, la découverte de topologie a aussi des applications avec d'autres domaines que le calcul parallèle. En effet, nous avons travaillé au début de cette thèse avec l'application de la topologie sur les algorithmes repartis comme la Diffusion Totalement Ordonnée. Ainsi, l'Annexe ?? montre comme la connaissance de la topologie du réseau peut être utilisée pour augmenter la performance des algorithmes repartis, réputés peu performants.

Chapitre 2

L'hétérogénéité des Données

Résumé

This article proposes to improve Apache Hadoop scheduling through a context-aware approach. Apache Hadoop is the most popular implementation of the MapReduce paradigm for distributed computing, but its design does not adapt automatically to computing nodes' context and capabilities. By introducing context-awareness into Hadoop, we intent to dynamically adapt its scheduling to the execution environment. This is a necessary feature in the context of pervasive grids, which are heterogeneous, dynamic and shared environments. The solution has been incorporated into Hadoop and assessed through controlled experiments. The experiments demonstrate that context-awareness provides comparative performance gains, especially when some of the resources disappear during execution.

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 représentées 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 solutions P2P ou 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, il est moins évident l’intégration d’autres représentations de données telles que les requêtes sur une base de données, des flux de données ou encore l’exécution de services.

C’est dans le but de proposer une architecture unifiée pour les données et les services que nous présentons GRAPP&S (GRid Applications and Services), une architecture multi-échelle pour l’agrégation de données et services. Ce framework 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), les services Web et le calcul distribué. À travers une structuration hiérarchique basée autour du concept de "communautés", GRAPP&S permet l’intégration de sources de d’information disposant de protocoles d’accès hétérogènes et des règles de sécurité variés.

D’autres contributions de GRAPP&S sont la mise en place d’une spécification détachée du middleware de communication (P2P ou autre), et la définition d’une solution de stockage générique. Dans le premier cas, ceci est possible car la spécification de GRAPP&S est basée sur des propriétés telles que la gestion de la connexité de la communauté et la gestion du routage entre les nœuds. Dans le deuxième cas, l’utilisation de nœuds "proxy data" permet d’unifier l’accès à des ressources variées telles que des fichiers, des flux, des services (FTP, mail) ou des données créées à la volée par des tâches de calcul ou des capteurs.

Enfin, l’accès aux données n’est pas dépendant d’une interconnexion directe entre les nœuds, comme c’est le cas de la plupart des réseaux P2P. Dans GRAPP&S, il est toujours possible de router le transfert des données par le chemin utilisé lors de la recherche, au cas où une connexion directe entre les nœuds n’est pas possible.

La suite de cet article est organisée comme suit : la section 1.2 introduit brièvement l’état de l’art concernant les systèmes multi-échelle et les réseaux P2P. La section 2 présente les principaux éléments qui constituent l’architecture GRAPP&S et comment ces éléments s’interconnectent, alors que la section 4 détaille les principales opérations liées à la gestion et à la recherche d’informations dans GRAPP&S. La section ?? détaille l’état actuel du développement de GRAPP&S et présente le module GAIA destiné à l’optimisation du stockage multi-échelle. Finalement, la section ?? propose nos conclusions.

1.2 État de l’Art

Systèmes Multi-Échelle

D’une manière générale, les systèmes multi-échelle sont donc structurés autour de services déployés sur plusieurs niveaux et qui se complètement afin de répondre aux besoins plus ou moins immédiats des clients. Rottenberg *et al* [74] formalisent cette définition sur la forme :

Un système multi-échelle est un système réparti sur plusieurs niveaux de tailles différentes dans une ou plusieurs dimensions (équipement, réseau, géographie, etc.)

Satyanarayanan présente un exemple de système multi-échelle dans ses travaux autour des "cloudlets" [77, 78]. Dans ces travaux, Satyanarayanan se penche sur les limitations des équipements mobiles et sur l'inadéquation des solutions actuellement mises en place pour l'externalisation des services mobiles (notamment le transfert de ces services sur le *cloud computing*). La solution proposée est la mise en place de serveurs de proximité sans état et connectés à Internet (*cloudlets*) auxquels les équipements mobiles peuvent se connecter directement via un réseau Wi-Fi. Ces équipements sont déployés comme des hotspots Wi-Fi dans des cafés, magasins etc. Ils possèdent une forte puissance de calcul et permettent aux équipements mobiles d'externaliser les calculs trop complexes sur une machine proche, ce qui résout les problèmes de latence du cloud computing.

De tels systèmes ont forcément des besoins de coordination très spécifiques. Ainsi, l'impact de l'hétérogénéité des technologies impliquées dans un système multi-échelle est étudié par Blair et Grace [17] alors que Kessis et al. [50] analyse le caractère malléable de ce système, capables de changer d'architecture et de granularité au cours de l'exécution. En effet, les systèmes multi-échelle requièrent un système de gestion flexible et adaptable capable de contrer la complexité et la volatilité de ces systèmes.

De même, Rottenberg *et al.* [74] cite la gestion de la confidentialité comme l'une des caractéristiques importantes des systèmes multi-échelle. À gestion fine de la confidentialité à chaque niveau de l'échelle vient s'ajouter au besoin d'une couche intergicielle dédiée à la sécurité de l'ensemble d'un réseau [34], tout en masquant l'hétérogénéité technique des systèmes multi-échelle.

Réseaux P2P

Differentes architectures hiérarchique P2P ont été proposée dans [68, 47, 35, 73]. Dans [47], l'auteur propose une architecture hiérarchique à deux niveaux. Au niveau inférieur il regroupe les pairs d'une même région, organisés sur un anneau de Chord et coordonnés par un super nœud. Au niveau supérieur se trouvent les super-nœuds de chaque région. [47] propose un algorithme de recherche régionale basé sur les super-nœuds, qui gardent une table de routage bidirectionnelle dans le but de réduire efficacement la redondance de la table de routage originale Chord.

Toutefois, même si l'architecture passe à l'échelle, elle ne résiste pas dans un environnement dynamique. Dans [35] les auteurs ont proposé un modèle hiérarchique de DHT (HDHT), où les pairs sont organisés en groupes. L'objectif des HDHTs est d'améliorer l'architecture plane DHT conventionnelle, par une exploitation des ressources hétérogènes des pairs, la mise en cache des infrastructures, la transparence et l'autonomie de différentes parties du système, afin de rendre la recherche de clés plus efficace tout en produisant moins de trafic.

Les auteurs de [73] proposent l'architecture hiérarchique CBT pour améliorer le protocole de téléchargement de fichier de BitTorrent dans un réseau de grande échelle. Il utilise trois types de pairs : les sources, les téléchargeurs et les super pair. Tous les supers pairs sont connectés au torrent tracker (gérant de l'index des ressources) pour former un réseau dédié. La disponibilité des informations est fortement liée au torrent tracker, car si ce dernier tombe en panne tout le service disparaît.

Les travaux effectués dans [47, 35, 73] reposent sur une architecture hiérarchique à deux niveau. Leur objectif est d'améliorer efficacement les performances tels que la latence lors d'une recherche et générer moins de trafic réseau. Cependant, ces architectures montrent leur limite dans les environnements dynamiques à cause de l'instabilité des nœuds. Une alternative est

proposée dans [68], où les auteurs proposent HP2P, une architecture hiérarchique hybride à deux niveaux, combinant DHT et systèmes P2P non structuré. HP2P utilise dans son premier niveau Chord et au deuxième niveau Kazaa, et procède par inondation lors d'une recherche. Ceci fait de HP2P un système robuste car elle combine les avantages des DHT et des systèmes non structurés, mais reste aussi limité par leurs inconvénients, comme la dépendance aux ressources de même type (fichiers uniquement) et le nombre de messages exponentiel générés lors d'une recherche par inondation. Face à ces travaux, nous sommes motivés à proposer la spécification d'une architecture hiérarchique plus générique, qui permet le stockage de tous les formats de données tout en conservant les avantages vis-à-vis de la performance réseau.

Kessis et al. [50] proposent un intericiel de gestion de ressources dans un contexte multi-échelle. Le but de ce système est de proposer une gestion flexible d'un ensemble de ressources réparties de façon multi-échelle. Dans cet article, le terme multi-échelle est employé pour décrire la répartition des ressources à travers des réseaux de tailles différentes : PAN (Personal Area Network), LAN (Local Area Network), WAN (Wide Area Network). La solution proposée consiste à regrouper les ressources en domaines et fédérations de domaines afin d'implémenter différents modèles de gestion. Il est également possible de passer d'un modèle de gestion à un autre au cours de l'exécution. Une représentation architecturale est extraite des ressources gérées afin de créer une couche d'abstraction homogène qui représente des ressources hétérogènes. Cette couche est composée d'éléments basiques qui peuvent être assemblés pour former des éléments composites. Les éléments basiques et composites sont tous considérés comme des éléments gérés de façon homogène du point de vue de l'application.

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 nuds du système et E désigne l'ensemble des liens de communications qui existent entre les nuds. Le modèle utilisé pour notre système est étudié dans [23]. Deux nuds 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 nuds u et v peuvent mutuellement envoyer ou recevoir des messages en mode asynchrone.

Un message m en transite est noté $m(id(u), m', id(v))$ où $id(u)$ est l'identifiant du nud qui envoie le message, $id(v)$ est l'identifiant du nud de réception, m' indique le contenu du message. Chaque nud 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 nud 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 entre deux vertex, e 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 noeuds. 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 noeud 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 noeuds 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 Communicateur.

2.4 Communicator (c)

Le noeud *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 [34]. Un noeud c dispose d'un identifiant unique (ID) à partir duquel on construit les identités des autres noeuds de la communauté. Ce noeud ne stocke pas de données et ne fait pas d'indexation.

2.5 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 designé (voir section 3.2).

2.6 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 une 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.

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, certains fabricants peu scrupuleux réutilisent les adresses MAC qui les sont attribués, et cela occasionne de nombreux problèmes dans les réseaux locaux, tout comme sur le déploiement des

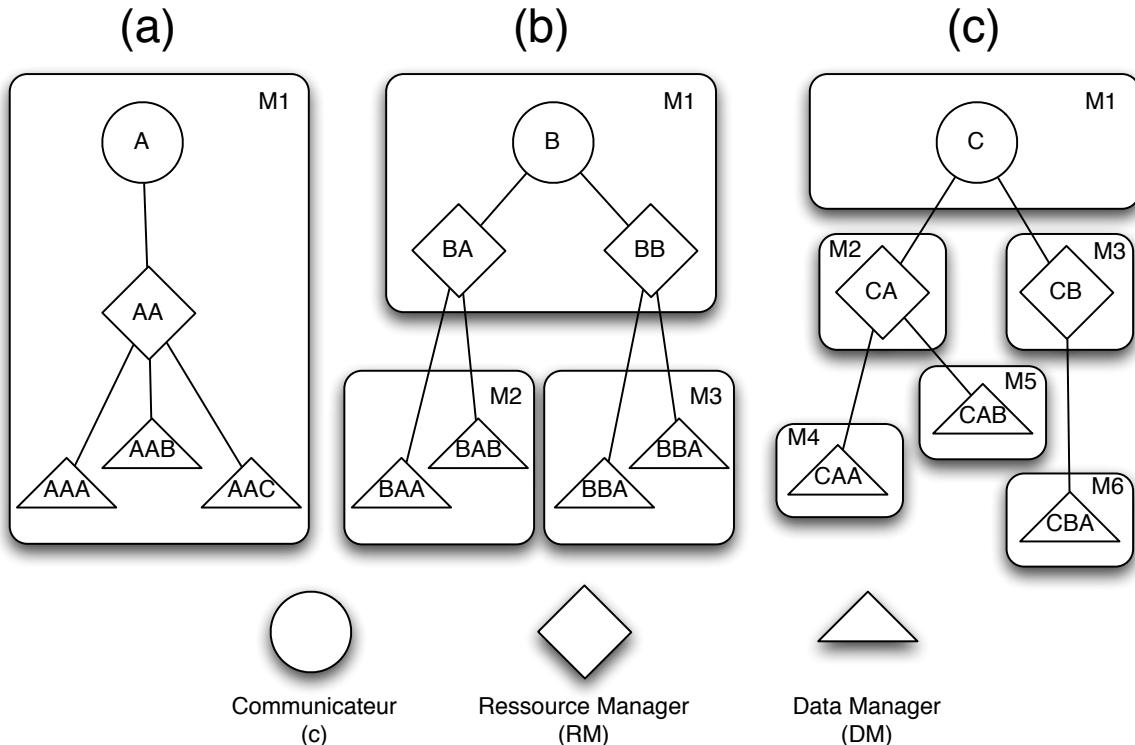


FIGURE 2.1 : Organisation des nœuds (a) dans une machine, (b) dans un cluster et (c) dans un réseau

réseaux IPv6¹. Ainsi, la solution la plus intéressante est celle proposée par JXTA [8] et qui utilise une chaîne de 128 bits. Chaque nœud dispose ainsi d'une chaîne unique ID_{local} , 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, i.e., 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 implanté. 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*.

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>

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.

3.1.1 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 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.

3.1.2 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*) [24] 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 défaiances, nous préconisons une procédure en deux étapes. Tout d'abord, chaque nœud dispose d'une liste de voisins $\{N_1, \dots, N_n\}$ 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 $\{ta_1, \dots, ta_n\}$.

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 a récolté les identifiants (ceux dont on a reçu des requêtes récemment, par exemple). Cette technique permet aussi de contacter d'autres *communicators* c_j et de réintégrer un réseau de communautés auprès la déconnexion involontaire de son *communicator* c_i . En dernier recours, GRAPP&S peut s'appuyer sur les éventuels mécanismes de découverte de topologie (broadcast/multicast) offerts par le propre *overlay*.

Vu que le problème de la reconnexion au reste de la communauté peut être traité de manière plus ou moins simple au sein de la propre architecture GRAPP&S, il est intéressant de se pencher sur les algorithmes d'élection eux-mêmes. Dans GRAPP&S, nous préconisons un algorithme d'élection distribué inspiré des protocoles de routage OSPF et IS-IS [65, 61, 16]. 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 managers 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 nœud ("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 nœud $DM_i \in C_i$ envoie la requête à son nœud 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 nœud RM_i retourne au nœud DM_i une liste de nœuds DM qui contiennent l'information recherchée

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

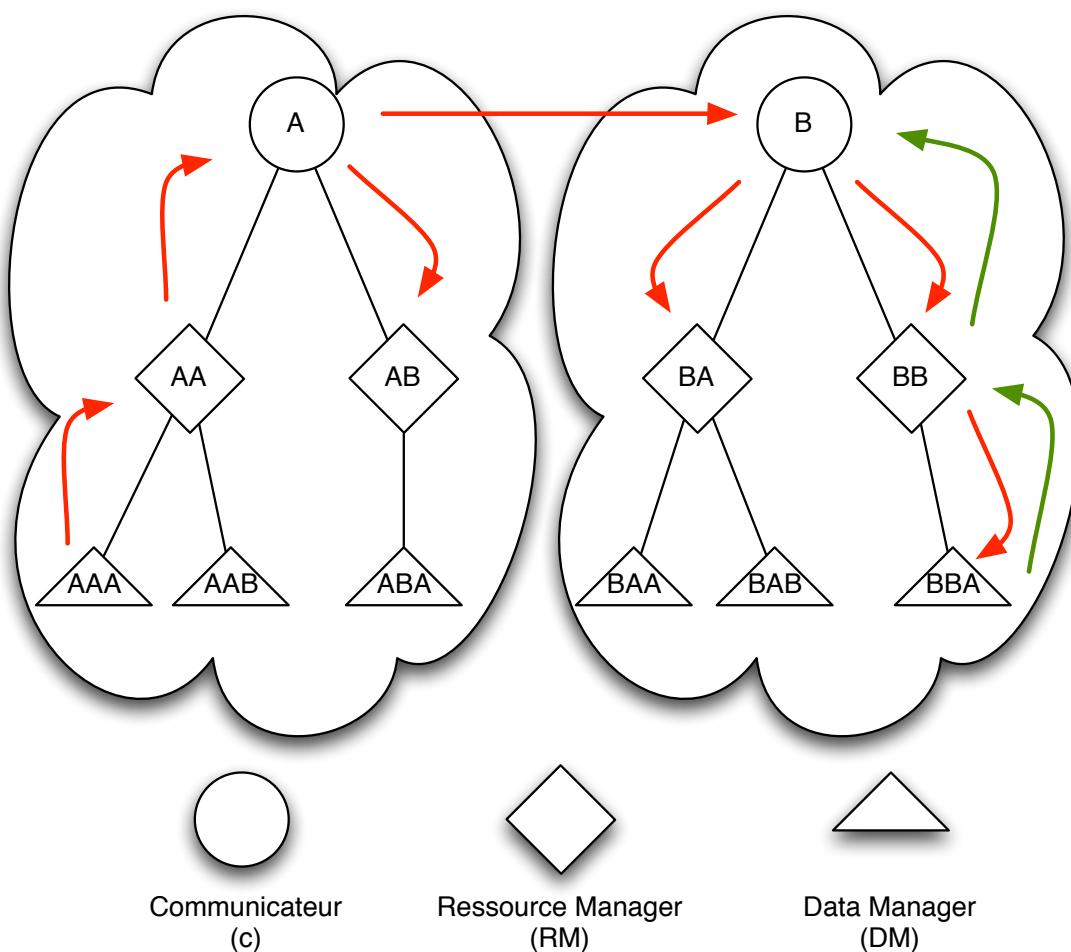


FIGURE 2.2 : Recherche d'une information dans GRAPP&S et mécanisme de routage préfixé

En cas de réussite, le client obtient l'identifiant du nœud DM_x responsable par la donnée. Dans ce cas, le client a deux possibilités pour accéder à la donnée, soit par connexion directe, soit par une connexion routée. Dans le cas de la connexion directe, le client fait une requête directe au nœud DM_x afin d'accéder à la donnée. Comme le client peut se trouver dans un autre réseau qui ne permet pas l'accès 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 intermédiaire d'un nœud DM_i , envoie une requête $\text{Req}(id(DM_x), Y, id(c_i))$ au nœud communicateur c_i .

- le nœud c_i , par routage préfixé, envoie la requête Req vers le nœud RM qui a indexé la donnée.
- Ce dernier fait suivre la requête vers le nœud DM_x responsable de la donnée.
- Une fois la donnée trouvée, le nœud DM_x retourne la bonne réponse au client en suivant le chemin inverse.

Ce mécanisme de recherche hiérarchique empêche l'inondation des liens du réseau. La hiérarchisation permet de définir des chemins par lesquels transitent les requêtes et comme la connectivité logique de notre architecture est par définition de $(n - 1)$, il suffit d'appliquer un algorithme de type PIF 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 1.2.

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 de 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'usage 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 par l'étude des interactions protéine-ligand, notamment à la recherche de petites molécules chimiques comme les 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 [?][?][?].

Souvent cette étude se fait à travers un "criblage virtuel" (*virtual screening*), qui consiste au déploiement à grande échelle permettant de tester un grand nombre de ligands (de centaines à plusieurs millions selon l'ampleur de la campagne) sur un nombre très restreint de cibles. En effet, nous trouvons des millions de composants catalogués dans des bases de données de chimie telles que la Cambridge Structural Database [?], PDBbind [?] [?], ZINC [?] et tant d'autres collections privées des groupes pharmaceutiques. De même, un large catalogue de protéines peut être obtenu à partir du Research Collaboratory for Structural Biology (RCSB) Protein Data Bank (PDB) [?], une base de données ouverte qui contient plus de 120 mille 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 bio-informatique, dirigée par le prof. Manuel Dauchez et co-encadré par Stéphanie Baud et moi-même) nous nous sommes penchés sur le développement et 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 métodologie de parallélisation

Le terme inverse docking a fait son apparition dans la littérature en 2001 avec les deux articles de Chen et al. [164][165]. 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 [172] ou AutoDock Vina [173] [174], et très peu d'outils sont dédiés au docking inversé (INVDOCK [164] [165]).

ou TarFisDock [177]). Toutefois, aucun de ces articles ne mentionne ni le déploiement ni 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 représente 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), et un algorithme génétique ira explorer le volume autour de la protéine afin d'évaluer l'énergie de liaison de différents points. Bien sûr, cette approche naïve est 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 de tours (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 de "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 de mieux utiliser les 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.

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 dans le fait de pouvoir déployer en parallèle les différentes sous-grilles comme des tâches de calcul indépendantes, toutefois un nombre trop important de découpages aura par 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

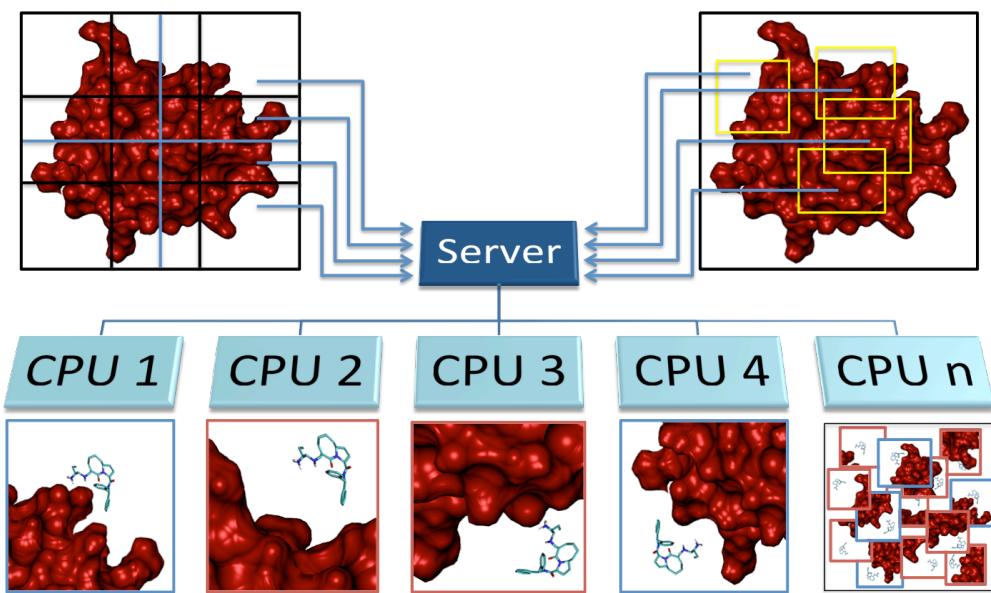


FIGURE 3.1 : Exemple d'un schéma de décomposition parallèle

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 à en considérant que le ligant se trouve totalement à l'intérieur de la sous-grille : n'importe quelle conformation où des atomes ligand dépassent la grille serait invalidée et donc ignorée.

1.2.2 Décomposition Géométrique avec superposition

Les inconvénients de la décomposition géométrique arbitraire cités dans le 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, $3 \times 2 \times 2$ (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 favorisée par la présence de cavités dans la surface de la protéine [34][35], 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 définit.

La détection des zones avec cavités se fait à partir d'un programme extérieur, Fpocket [36], spécialisé dans la recherche de cavités et de poches grâce à un algorithme géométrique basé sur les diagrammes de Voronoï. La prise en compte des zones avec un plus grand potentiel peut augmenter la performance du docking inversé car cela permet à la recherche de se concentrer sur une zone plus spécifique, mais 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 faisant une longueur entre un tiers et la moitié de la longueur totale de la protéine ont été conservés. 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 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 prise 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.

Ainsi, un ensemble de scripts Python a été crée pour automatiser toutes les étapes liées à la préparation et à l'exécution du docking inversé. Entre 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. Les esclaves doivent d'abord finaliser 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.

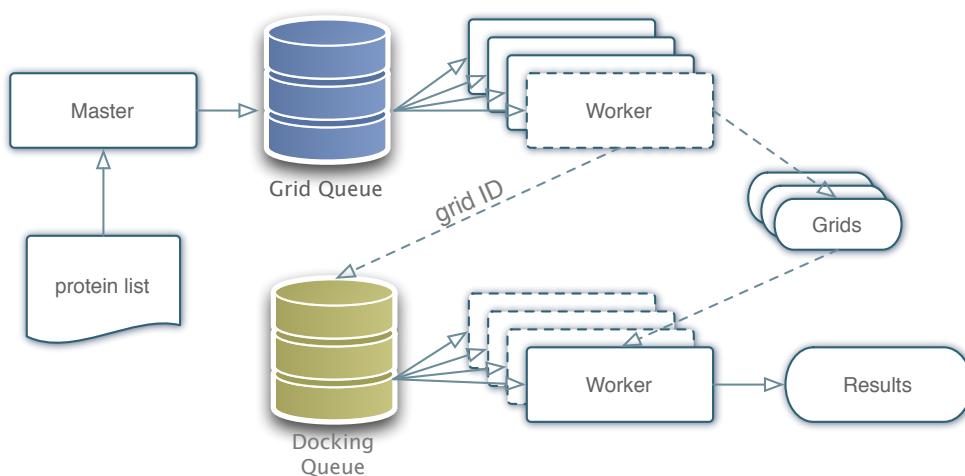


FIGURE 3.2 : Représentation du flot d'exécution dans l'architecture distribuée

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, OAR ou Slurm 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 à 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 géré 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, autant par rapport à la technique classique du blind docking que par rapport à des données expérimentales obtenues par cristallographie. Dans un deuxième moment nous avons conduit des tests de performance, où le docking inverse était effectué 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).

TABLE 3.1 : 3CM2 docking accuracy of decomposing strategies with different number of runs. RMSD is compared to the experimental pose and binding energies are compared to those obtained by blind docking.

ΔG blind docking ($kcal.mol^{-1}$)	Number of runs for cutting method	ΔG cutting ($kcal.mol^{-1}$)	RMSD ()
$\Delta G = -10.27$	20	$\Delta G_{12} = -10.09$	1.79
	50	$\Delta G_{pocket} = -10.11$	1.86
	70	$\Delta G_{pocket} = -10.39$	1.62

The space cutting with the better free energy binding is $n = 12$, as illustrated in Fig. 3.4. With only 20 docking runs, this type of decomposition gives the same free binding energy as the blind docking experiment and a RMSD pose of 1.79 Angstroms (see Table 3.1). Even if they succeed in replacing the ligand in the experimental cavity (as the blind docking), the other cuttings $n = 8, 27$ or 64 , do not give satisfying results in term of binding energy and RMSD value.

It is important to highlight that the pocket search derived from *Fpocket* experiment is able to predict the subspace and placing the ligand with $\Delta G = -10.11 \text{ kcal.mol}^{-1}$ and a RMSD value of 1.86 Angstroms for a 50 runs docking experiment.

In a second experiment, we carried out a larger experiment in which we tested our ligand (X23) on a set of 100 proteins from the PDB. Fig. 3.5 compares the raw performance of the different techniques when the number of runs performed is proportional to the 3D box volume.

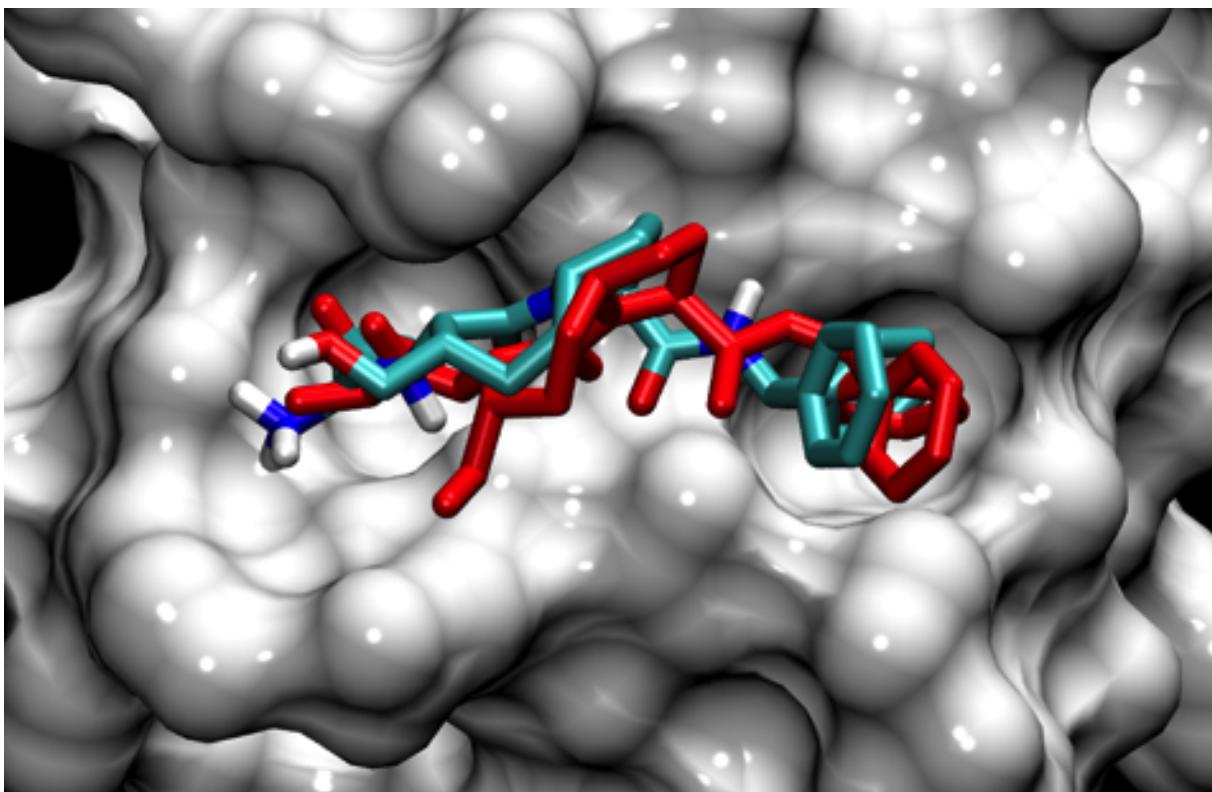


FIGURE 3.3 : Placement du ligand selon la méthode cristallographique (rouge) et celle obtenue par blind docking (vert)

Hence, if we consider that the space cutting respects the box proportions, the ideal number of runs for a cutting $n = 8$ without overlapping should be $256 / 8 = 32$, for example. Not all decomposition techniques are prone to this proportional rule, for example the pocket strategy that targets cavities with variable dimensions. In this case, we present the computational cost for a fixed number of runs (50, for instance).

By using a proportional number of runs for the blind strategy and $n=8$, 12 and 27, we are able to conduct the same number of evaluations in less time, while improving distribution. Nonetheless, an indivisible preprocessing cost at the beginning of each docking execution prevents a linear acceleration, and it is also important to consider that the accuracy of the different techniques is not the same : as shown in the previous section, a strategy with a 12-part cutting with 21 runs is much more precise than a 8-part cutting with 32 runs due to the use of overlapping areas. We can point out that the cost for the pocket strategy also depends on the number of identified cavities and the size of the pocket box. In the case of the 3CM2 protein, only one cavity was identified but this number may be higher or cover a larger portion of the protein.

In spite of the overall computational cost, the use of decomposition methods present clear advantages when regarding fault tolerance and load balancing. By dividing the docking of a protein in several tasks, we limit the losses in the case of a failure. For example, a computer failure in a blind docking that takes more than 5h30 requires the re-execution of the entire docking ; this is not the case in a 12-part decomposition, where at most half-hour is lost. In addition, the use of a bag-of-tasks queue mechanism improves the load balancing, as illustrated in Fig. 3.6.

In the case when all the available resources are less important than the number of tasks, com-

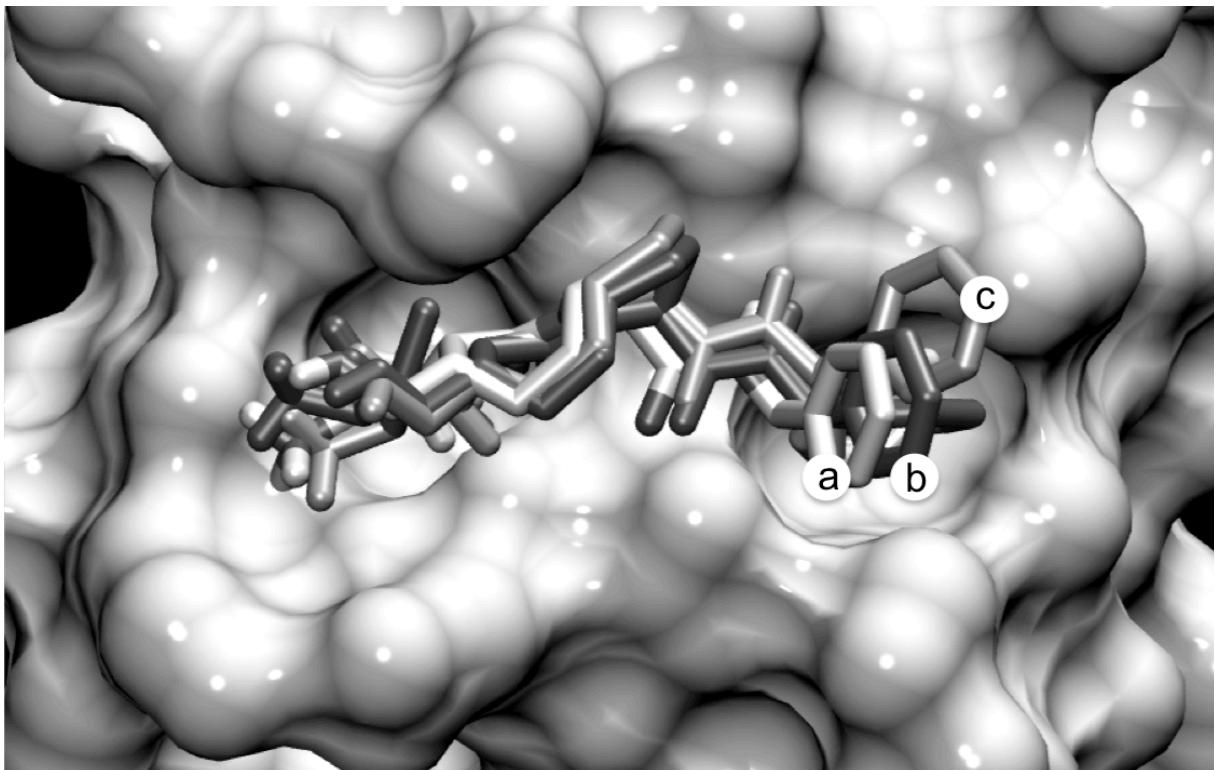


FIGURE 3.4 : Ligand pose from the blind docking compared to methodology best results. X23 pose from the blind docking in light grey (a), X23 pose from the $n = 12$ cutting in dark grey (b) and X23 pose from the pockets search in medium grey (c).

putation time increases irremediably whatever the method employed. Nevertheless, we proved on a large set of proteins, that our method performed a better volume exploration and gave better results than the blind docking. In addition, we can better distribute the load, better manage the resources and improve fault-tolerance.

2 Hétérogénéité des Ressources de Calcul

Apache Hadoop is a popular framework for distributed and parallel computing. It implements the MapReduce programming paradigm, which aims at processing big datasets [28] and to scale up from a single server to thousands of machines.

Without specific configuration by the administrator, Apache Hadoop supposes the use of dedicated homogeneous clusters for executing MapReduce applications. As the overall performance depends on the task scheduling, Hadoop performance may be seriously impacted when running on heterogeneous and dynamic environments it was not designed for.

This is a special concern when deploying Hadoop over pervasive grids. Pervasive grids are an interesting alternative to costly dedicated clusters, as the acquisition and maintenance of a dedicated cluster remain high and dissuasive for many organizations. According to [66], pervasive grids represent the extreme generalization of the grid concept, in which the resources are pervasive. Pervasive grids use resources embedded in pervasive environments to perform computing tasks in a distributed way. Concretely, they can be seen as computing grids formed by existing resources (desktop machines, spare servers, etc.) that occasionally contribute to the computing

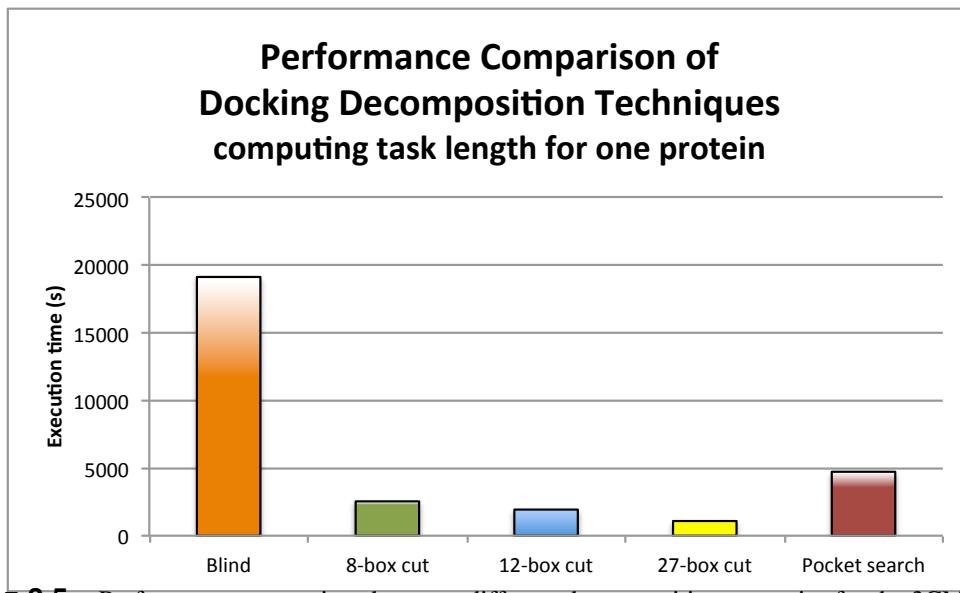


FIGURE 3.5 : Performance comparison between different decomposition strategies for the 3CM2 protein.

grid power. These resources are inherently heterogeneous and potentially mobile, dynamically joining and leaving the grid. Knowing that, in essence, pervasive grids are heterogeneous, dynamic, shared and distributed environments, their efficient management becomes a very complex task [63]. Task scheduling is thus severely affected by the environment complexity.

Many works proposed to improve the Hadoop framework on environments that diverge from the original working specifications ([52, 95, 72, 76]). The PER-MARE project ([84]), in which this work was developed, aims at adapting Hadoop to pervasive environments [81].

Therefore, adapting the execution to dynamic environments is a necessity as Hadoop is based on static configuration files that do not adapt to resources variations. Attempting install on heterogeneous clusters imply for the administrators to manually set the characteristics for each resource, a repetitive and time consuming task. All these factors prevent deploying Hadoop on more volatile environments, and our objective is to improve Hadoop so that it could adapt itself to the execution context and therefore be deployed over pervasive grids.

In order to adapt Hadoop to a pervasive grid environment, supporting context-awareness is essential. Context-awareness is the capacity of an application or software to detect and respond to environment changes [56]. A context-aware system is able to adapt its operations without human intervention, therefore improving the usability and efficiency of the system [4]. In pervasive grids, context-aware data may help task schedulers to make better decisions based on real feedback from the system.

This work focuses on our developments to introduce context-awareness capabilities on Hadoop task scheduling mechanisms. Through a context collection procedure and minimal changes on Hadoop's resource manager, we are able to update the information about the availability of resources in each node of the grid and then influence the scheduler tasks assignments. It also extends the preliminary observations from [19] through the analysis of additional performance benchmarks.

The rest of the paper is organized as follows : Section 2.1 presents Apache Hadoop architecture and scheduling mechanisms. Section 2.2 discusses related work, focusing on context-awareness and on other works that try to improve Hadoop schedulers. Section 2.3 presents our proposal of context-aware scheduling, while Section 2.4 presents the experiments conducted

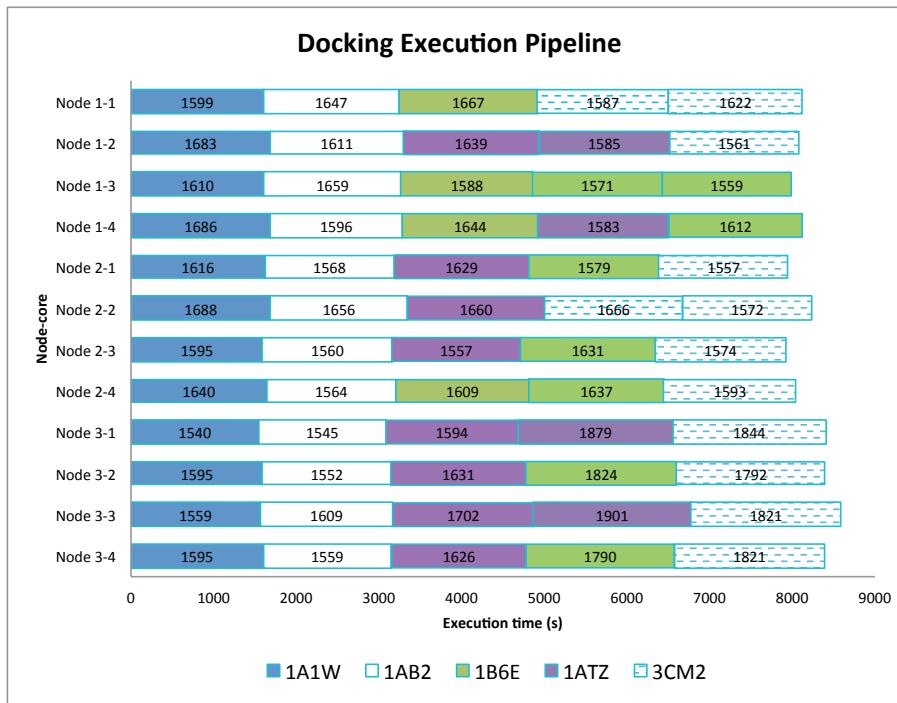


FIGURE 3.6 : Load balancing observed during a docking with 5 proteins (the individual task times are shown in the bars).

and the results achieved. Section 2.5 introduces general discussion about the results and future challenges. We finally conclude this paper in Section 2.6.

2.1 About Hadoop Scheduling

Before discussing related works and presenting our proposal, it is worth introducing some basic concepts about the Hadoop framework and its schedulers.

2.1.1 Hadoop Framework Architecture

The current Apache Hadoop framework is organized as a master and slave architecture, with two main services : storage (HDFS) and processing (YARN). Both services have their own master and slave components, as presented on Fig. 3.7 : the *NameNode* and *ResourceManager* services are the masters of the HDFS and YARN respectively, and the *DataNode* and *NodeManager* their slave counterparts. It is also possible to note the *ApplicationMaster*, the component responsible for internal application management (task scheduling), while *ResourceManager* is responsible for job scheduling. Each node also runs a set of *Containers*, where the execution of Map and Reduce tasks takes place.

2.1.2 Hadoop Schedulers

The Hadoop framework has two execution layers that depend on schedulers, according to their granularity. The coarse-grain instance is the job, while the fine-grain instance is represented by the tasks that compose a job.

Job-level scheduling is performed by the *ResourceManager*, an entity that has a global view of the available resources and can, for instance, arbitrate available resources among the com-

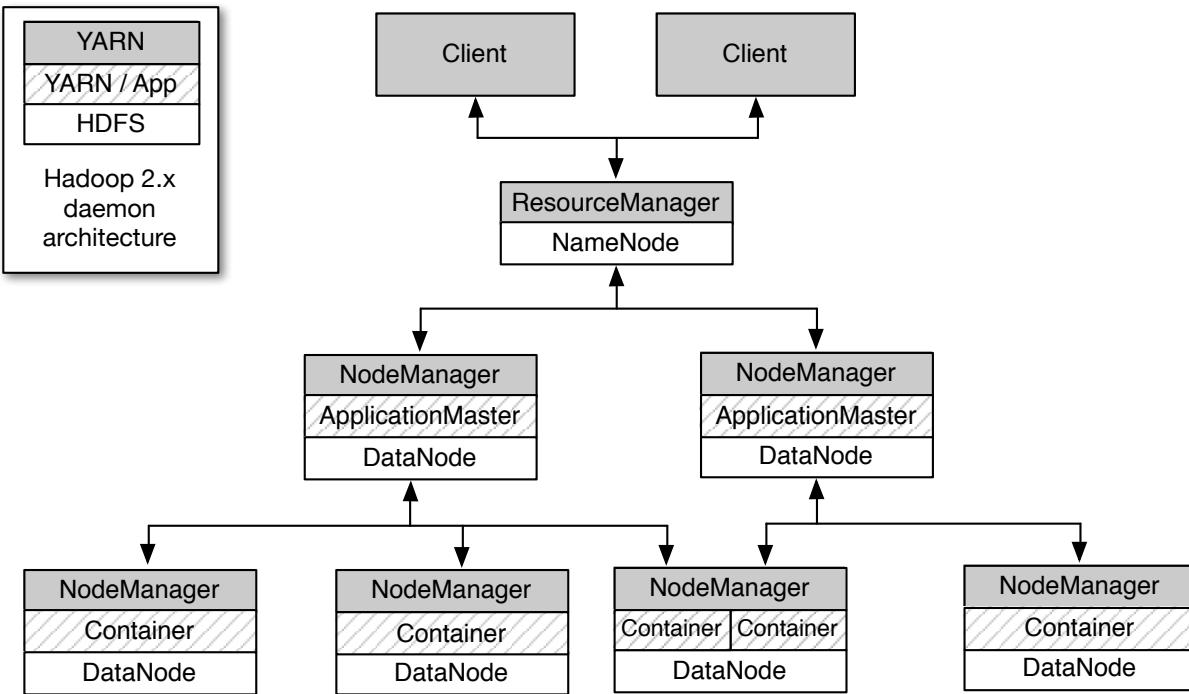


FIGURE 3.7 : General Apache Hadoop architecture

peting applications. Each application is associated to an *ApplicationMaster* that tries to acquire the right to use a given number of resource units (*containers*) from the *ResourceManager*; the latter must optimize the resources utilization according to different constraints such as capacity guarantees, fairness, and SLAs.

The *ResourceManager* can be optimized to different constraints and parameters through the use of pluggable schedulers. The simplest scheduler algorithm, *Internal Scheduler*, processes all jobs in the order of arrival (FIFO). This scheduler performs well in dedicated clusters where the competition for resources is not a problem. Another scheduler available is the *Fair Scheduler*, which uses a two level scheduling to fairly distribute the resources among batches of small jobs [2]. A third widely known scheduler is the *Capacity Scheduler*, which is designed to run Hadoop MapReduce in a shared, multi-tenant cluster. The Capacity Scheduler is designed to allow sharing a large cluster while giving each organization a minimum capacity guarantee. The central idea is that the available resources in the Hadoop MapReduce cluster are partitioned among multiple organizations that collectively fund the cluster based on computing needs [2].

These schedulers allows a flexible management of the framework at the job level. Yet, the available schedulers neither detect nor react to the dynamicity and heterogeneity of the computing environment, a requirement for pervasive grids. Indeed, the design of Hadoop YARN passes on this responsibility to the *Application Master*, which can better adapt to the application needs.

At a fine-grain, scheduling is performed by the *ApplicationMaster* according to the number of tasks and assigned resources. There is almost no documentation on the default task scheduler, but we can assume that it implements a progressive filling approach where tasks are fed to all granted containers in one node before starting filling the next node. This behavior was experimentally observed in [20].

It is also worth noting that the *ApplicationMaster* is not aware of the real execution context as it relies on the resources granted by the *ResourceManager*, i.e., it has only a limited view on

the computing platform. Modifying the *ApplicationMaster* scheduling algorithms to consider real context-awareness requires changes on the *ResourceManager* itself. After reviewing some related work on the next session, we will introduce our approach to bring context-awareness to Hadoop.

2.2 Related Work

Over the years, different works proposed improvements to the Hadoop scheduler mechanisms in order to respond to specific needs. These contributions may be classified as new scheduling methods or as improvements for the resource distribution.

Works like [52], [87] and [72] assume that most applications are periodic and demand similar resources regarding CPU, memory, network and hard disk load. These assumptions allow the applications and nodes to be analyzed regarding the CPU and I/O potential, enabling the optimization of execution through matching of nodes and applications with the same characteristics. [46] focus on a new scheduling method proposing the usage of a capacity-demand graph that assists the calculation of optimal scheduling based on an overall cost function.

While previous works focus on performance improvement using static information about resources and applications, other works sought to incorporate task specific information into their proposals. For example, [95] and [25] attempted to improve tasks distribution as a way to reduce the response time in large clusters. [95] use heuristics to infer the estimated task progress and to make a decision about the launching of speculative tasks. Speculative tasks are launched when there is a possibility that the original task is on a node either faulty or too slow node. Another work ([25]) proposes the use of historical execution data to improve decision making.

Both scheduling mechanics and resource distribution methods result in a load rebalancing, forcing faster nodes to process more data and slower nodes to process less data. [76] try to achieve that through a system based on resource supply and demand, allowing each user to directly influence scheduling through spending rates. The main objective is to allow a dynamic resource sharing based on preferences set by each user.

There are also works such as [94], which attempt to provide a performance boost in jobs through better data placement, mainly using data location as information to decision making. The performance gain is achieved through data rebalancing on nodes, increasing the load on faster nodes. This proposal reduces the number of speculative tasks and data transfers over the network. A similar proposal is observed by [22], which address scheduling and data distribution issues on geographically distributed clusters. These authors present a new hierarchical scheduling mechanism based mainly on throughput and application data profiling. As for [94], [22] also focus on optimizing data transfer through the network.

[57] uses a P2P structure to arrange the cluster. In this approach, nodes can change their function (master/slave) over time and can have both functions at the same time, the functions being tied to the applications and not to the cluster. The objective of this work is the adaptation of MapReduce paradigm to a P2P environment, which given the natural volatility of P2P environments, would offer support to pervasive grids. However, this proposal focuses on providing a resilient infrastructure and does not explore the scheduling of jobs and tasks. Another work relying on a P2P overlay, [82] offers MapReduce-like computing for pervasive environments on top of a general distributed computing platform. However, this platform focuses on fault-tolerance and volatility aspects through a fully distributed task scheduling mechanism, which for the moment does not implement context-aware scheduling optimizations.

Indeed, most of previously cited works do not actually consider the evolving state of the

available resources. Resources are described, not observed. However, works on context-aware computing ([4, 56, 71, 62]) have demonstrated that this observation is possible and that the execution environment may influence application behavior. This raises a question : can we improve MapReduce scheduling by observing current execution environment ? The next sections will try to answer this question.

2.3 Context-Aware Scheduling

The main goal of this work is to improve the scheduling of Hadoop by adding support to dynamic changes at the *Resource Manager* level. Unlike the works on Section 2.2 we opted to feed dynamic context information to an existing scheduler (*Capacity Scheduler*) and therefore modifying the Hadoop source code the least possible.

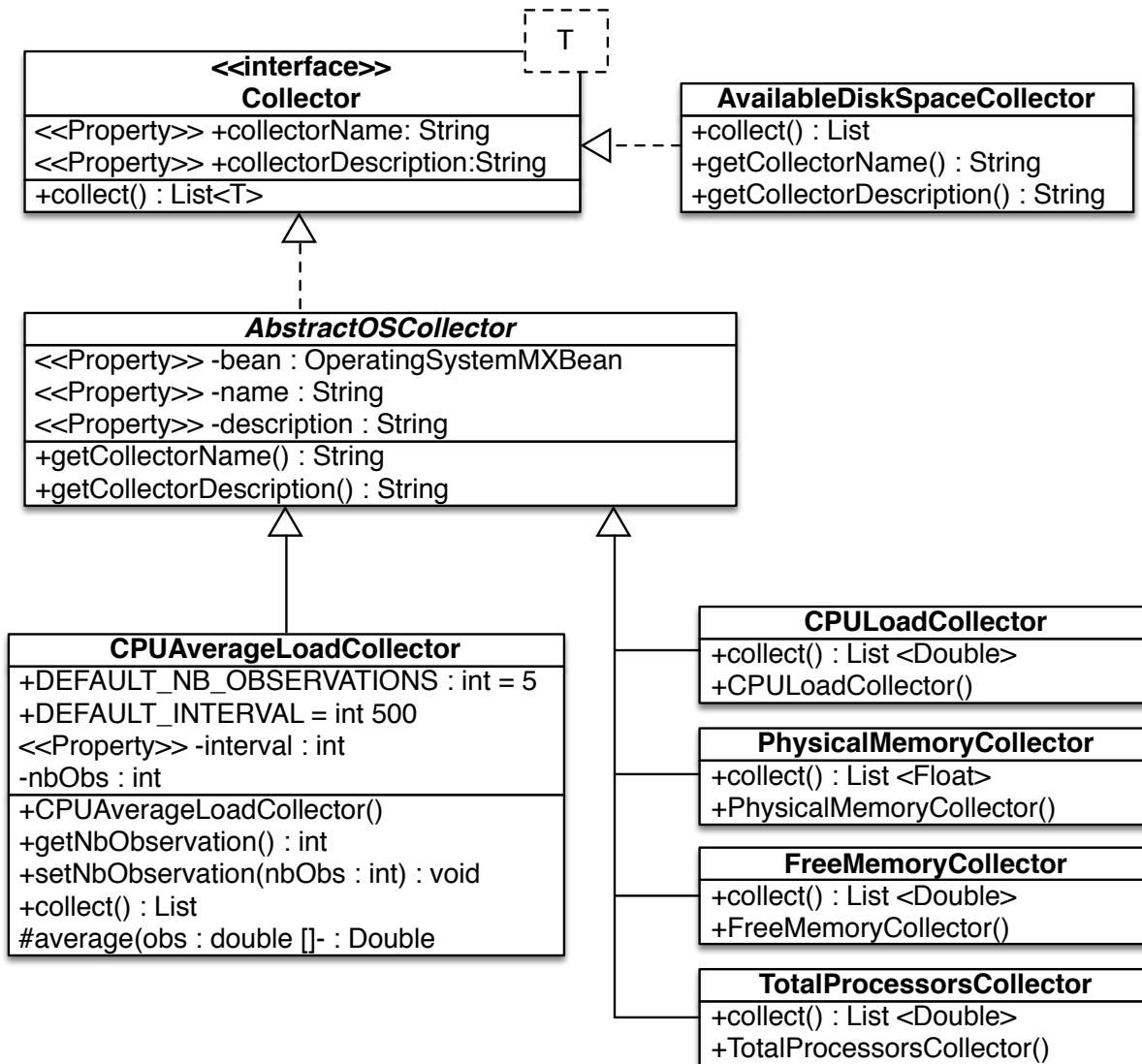
In the default Hadoop implementation, a *NodeManager* declares its computing resources to the *ResourceManager* when joining the Hadoop network, this information being usually obtained from static configuration files. In order to detect dynamic changes, the scheduler must collect context information that, in this case, refer to available resources on the nodes. Then, a *NodeManager* communicates periodically with the *ResourceManager* in order to keep information updated and let the scheduler adapt to the new context. In the following section we present a more detailed explanation of the changes implemented in Apache Hadoop.

Context collector

By default, Hadoop reads information about the nodes from XML configuration files. These files contain many configuration parameters, including the resource capacity of each node. Once loaded, the information will not be updated until the service is restarted. As pervasive environments may face performance changes during the execution of an application, we need a mechanism that collects contextual information at runtime and subsequently updates the *ResourceManager* knowledge base.

Therefore, we integrate into Hadoop a collector module, allowing to observe contextual information about the available resources. The collector was developed for the PER-MARE project [84], and its class diagram is presented in Fig. 4.3 [20]. The collector module is based on the standard Java monitoring API ([64]), which allows to easily access the real characteristics of a node. It allows collecting different context information such as the number of processors (cores) and the system memory using a set of interface and abstract/concrete classes that generalize the collecting process. Due to its design, it is easy to integrate new collectors and improve the resources available for the scheduling process, providing data about the CPU load or disk usage, for example.

Nevertheless, replacing XML configuration files by the context collector is not enough, since a global knowledge of the available resources is necessary in order to adapt scheduling to runtime conditions. Indeed, to allow *ResourceManager* to adapt scheduling to the current available resources, the context collector associated to each node must be able to communicate its current state to the *ResourceManager* during the execution. In order to do so, we have improved communication capabilities of both *ResourceManager* and *NodeManager* as explained in the following section.

**FIGURE 3.8 :** Context collector structure

Communication

Gathering the context information requires to feed the Hadoop scheduler requires transmitting this information through the network from slave nodes (*NodeManager*) to the master node (*ResourceManager*) which is in charge of the scheduling. Instead of relying on a separate service, we chose to use the ZooKeeper API [44] that provides efficient, reliable, and fault-tolerant tools for the coordination of distributed systems. In our case, ZooKeeper services are used to distribute context information.

As illustrated in Fig. 3.9, all slaves (*NodeManager*) run an instance of the *NodeStatusUpdater* service, which collects data about the real resources availability (for example, every 30 seconds). If the amount of available resources changes, the DHT on ZooKeeper will be updated. Since the Operating System produces some variations on the resources, this information will only be changed if the variation is big enough to raise/lower the maximum capacity of containers on that node. This small change will spare a lot of executions where the information changed, but the variation was not enough to impact the scheduling. Similarly, the master (*Re-*

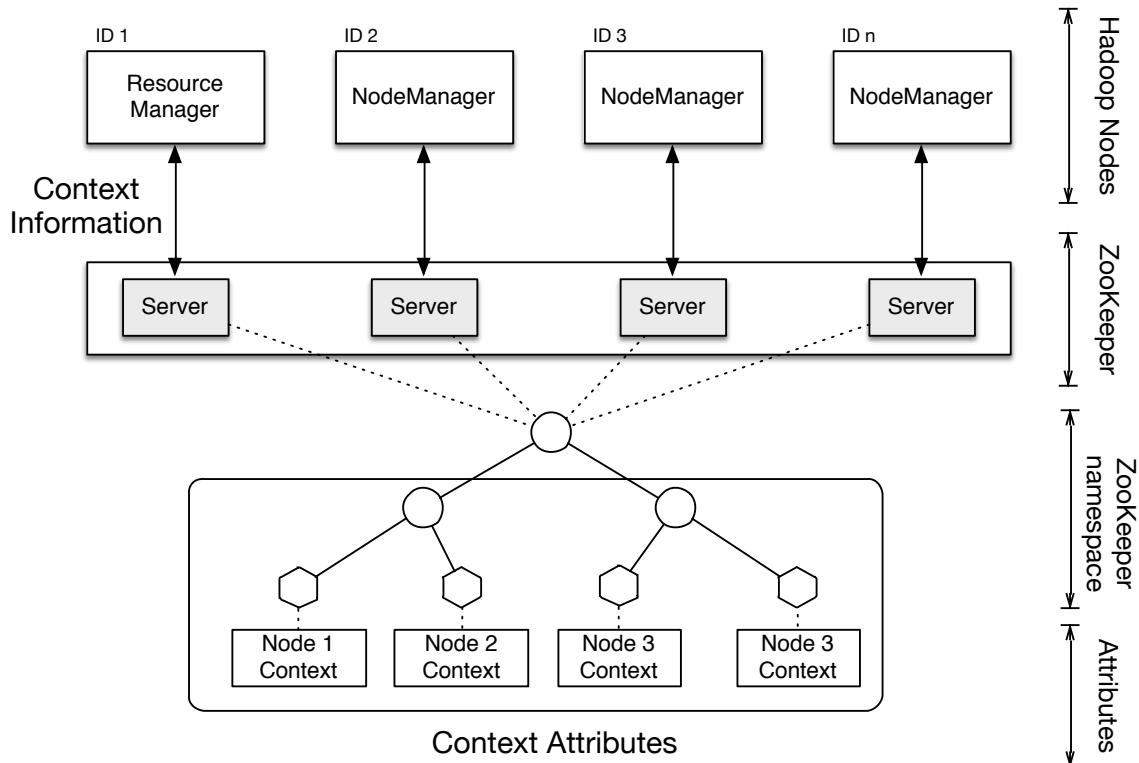


FIGURE 3.9 : Using ZooKeeper to distribute context information

sourceManager) also creates a service to watch ZooKeeper's zNodes. If the Zookeeper node detects a DHT change, the master will be notified and will update the scheduler information based on the new information. This solution extends a previous one presented in [20] by offering a real time observation of available nodes. Indeed, our previous solution only updates information regarding the resources on service initialization, replacing the XML configuration file, while this one updates resource information whenever the availability changes. As a result, scheduling is performed based on the current resource state.

2.4 Experiments and Results

In order to evaluate the impact of context awareness on the job scheduler mechanism, we performed two sets of experiments. In the first one, presented on Sections 2.4.3 and 2.4.3, we considered a small dedicated cluster so that the stress of incorrect resources estimation could easily be controlled and analyzed. On the second set of experiences, presented on Sections 2.4.3 and 2.4.3, we run the TeraSort benchmark in a larger number of nodes but a real shared environment (i.e., with nodes shared with other users and applications). Both the first and the second experiments considered different execution scenarios as well as different application profiles, as presented in the following section.

2.4.1 Execution scenarios

In the context of this work we compare the Hadoop behavior under different execution scenarios using the available memory and the number of nodes (*v-cores*) as resource metrics. These parameters are reported to the *ResourceManager* and represent the main elements considered by

the Capacity Scheduler algorithm. In addition, the available memory parameter can be considered as closely related to the node real environment (contrarily to the *v-cores*) and is easily configurable through the *yarn.nodemanager.resource.memory-mb* property.

The execution scenarios were tailored to reproduce different combinations of reported resources, available resources and context awareness. Indeed, the execution performance is closely related to the assignment of computing resources to the applications. Incorrect configuration parameters or external factors may heavily impact the nodes performance, especially when these parameters lead to the resources overload. Similarly, changes on the available resources during the execution may drive a well-configured environment to an overloaded state if the changes are not reported to the scheduler. The four scenarios described below try to reproduce these situations, thus providing sufficient information for the analysis and discussions at the end of this paper.

Scenario A : in this scenario we simulate a dedicated Hadoop cluster so that the reported memory always correspond to the available memory, which can be considered as the "best case" scenario. We consider the reported memory as the information that the scheduler use in the scheduling process, while available memory is the free memory of the node or cluster. Using a direct notation, the reported memory is 100% and the available memory is also 100% all through the execution.

Scenario B : in this case nodes can be used for other purposes than running Hadoop, so the reported memory may at some point differ from the available memory initially configured for Hadoop usage. This case corresponds to the default behavior of Hadoop, where memory resources are provided through a property *yarn.nodemanager.resource.memory-mb*. Because the scheduler never adapts to the reduced resources, we consider it the "worst case" scenario. On this scenario, using a direct notation, the reported memory is 100%, but the available memory is 50%.

Scenario C : in this case, the nodes are shared with other applications (like in Scenario B) but the context-awareness collector is active, updating context information every 30 seconds. When the MapReduce application is launched, the scheduler is aware of the execution context and can assign tasks according to the available resources. Using a direct notation, the reported memory and the available memory correspond to 50% of the values reported on Scenario A.

Scenario D : this scenario presents an extension of Scenario C where the MapReduce application starts before the context collector update. The scheduler starts with wrong available resource information and must therefore adapt during the execution with the help of the context collector. Using a direct notation, the reported memory at the beginning of the execution is 100% of the resources from Scenario A (wrong information), but at runtime this information is updated to 50%.

Scenario A corresponds to the "best case" for Hadoop framework, in which all resources are available during the entire execution. Scenario B simulates Hadoop execution in a heterogeneous environment where resources fluctuate during the execution but the scheduler is never aware of the changes (a "worst case"). If we consider context information, Scenario C illustrates a situation in which the context collector is able to detect environment changes before application execution starts while scenario D suffers from late information update but can still adapt to the changes. These two scenarios allow us to evaluate the impact of the context-aware scheduler during the deployment of an application in a heterogeneous environment.

2.4.2 Application benchmarks

While big-data applications are expected to rely on memory, other factors like CPU usage and I/O may also impact the execution of the applications. For this reason, this work uses three different benchmarks, each characterized by different memory, CPU and I/O usages [53]. The three benchmarks are the following :

- TeraSort : The goal of TeraSort benchmark [40] is to sort a given amount of data as fast as possible. It is a benchmark that combines testing the HDFS and MapReduce layers of a Hadoop cluster. Because of the sorting algorithms, this benchmark stress memory and CPU ;
- WordCount : the WordCount benchmark is the basic MapReduce example. Its objective is to count the number of occurrences of each word in a given text. Because WordCount memory and I/O usage are limited (both memory and output structures are small in comparison to the input file), the main performance is notably determined by the CPU ;
- TestDFSIO : The TestDFSIO benchmark is a read and write test for HDFS. It is helpful for tasks such as stress testing HDFS, to discover performance bottlenecks in the network, OS and Hadoop setup ; it aims at giving a first impression of how fast a cluster is in terms of I/O. Memory and CPU are less solicited.

For the experiments we use the HiBench benchmarks suite, which is detailed in [43]. The TeraSort benchmark was run using a data set of 15 GB, TestDFSIO was run with 90 files of 250 MB and WordCount used a 10 GB file as input.

2.4.3 Environment setup and configuration for the controlled experiments

In order to test the new behavior of the framework, we conducted experiments with the Grid'5000 platform [37]. We configured a dedicated network with 4 slaves, each having the following configuration : 2 Intel Xeon CPU E5420 @ 2.50 GHz (totalizing 8 cores per node) and 8 GB of RAM. All nodes run Ubuntu-x64-12.04, with JDK 1.7 installed, and the Apache Hadoop 2.5.1 distribution.

The resources evaluated in the experiments were the memory and number of cores, which have a direct impact on the amount of Map tasks allocated. In addition to the overall performance of each benchmark on the different scenarios, we performed an in-depth investigation on the tasks (containers) distribution throughout the execution. The information about the execution of each container was obtained from the Hadoop logs. These information allow us to present detailed Gantt charts of the tasks placement during the experiments.

To emulate the scenarios with reduced resources (Scenarios B, C and D), we opted to halve the number of nodes while reporting the same amount of global available memory as in Scenario A. While this approach is unrealistic (resources from failing nodes should be withdrawn from the available resources set), it provides a clear reference for the analysis of the scheduler decisions.

Results from the controlled experiments

The results from experiments are presented in both Tables 3.2, 3.3, 3.4 and Figures 3.10, 3.11, 3.12, respectively for TeraSort, TestDFSIO and WordCount. All Tables summarize the experiments presenting the total time used by all map tasks, the average execution time, the standard deviation and the number of speculative tasks launched on each scenario and for each benchmark.

[h !]

TABLE 3.2 : Summary of TeraSort results expressed in seconds.

Scenario	A	B	C	D
Total Map Time (s)	149	788	348	477
Avg. Map Time (s)	39.47	222.97	38.38	68.42
Standard Deviation	15.73	59.86	18.09	29.91
# Map Tasks	76	76	76	76
# Speculative Tasks	2	1	3	1

[h !]

TABLE 3.3 : Summary of TestDFSIO results expressed in seconds.

Scenario	A	B	C	D
Total Map Time (s)	139	444	239	364
Avg. Map Time (s)	38.95	85.01	32.20	81.62
Standard Deviation	17.20	69.08	8.30	73.60
# Map Tasks	90	90	90	90
# Speculative Tasks	0	9	0	1

[h !]

TABLE 3.4 : Summary of WordCount results expressed in seconds.

Scenario	A	B	C	D
Total Map Time (s)	155	1009	309	805
Avg. Map Time (s)	43.76	208.39	41.73	175.80
Standard Deviation	15.61	128.90	10.99	151.59
# Map Tasks	90	90	90	90
# Speculative Tasks	1	15	1	10

Figures 3.10, 3.11 and 3.12 present the Gantt charts for each benchmark and scenario. For a given benchmark, each scenario is composed by either 2 or 4 lines, one for each node in the cluster during the experiment. As stated in the description of the scenarios, Scenarios B, C and D run on half of the nodes in Scenario A to simulate a reduced amount of resources. On each line we consolidate the resources according to a color scale where the darker the tone, the more containers run simultaneously and thus, more overloaded the node is. For instance, white means no container executing, and black means 16 containers. Additionally, the lines are segmented along the chart to indicate that a container has either finished or started at that moment. The charts are scaled in seconds and all charts for a given benchmark have the same scale. Hence, TeraSort charts go from 0 to 790 seconds, TestDFSIO go from 0 to 450 seconds and WordCount from 0 to 1010 seconds.

While the Gantt charts represent the tasks (containers) execution, some containers are not affected by scheduling, such as the *ApplicationMaster* or the Reduce tasks. For this reason, we ignore these tasks and concentrate the analysis on the elements that can be affected by the context-aware scheduling.

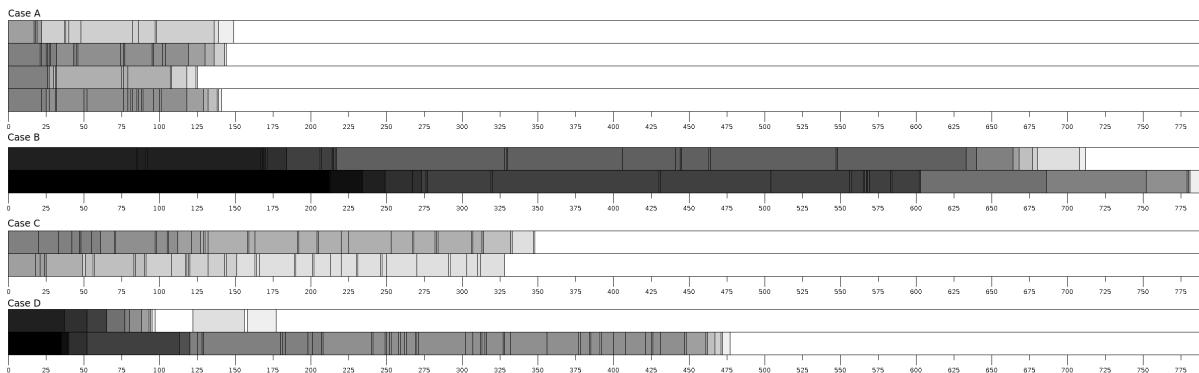


FIGURE 3.10 : Gantt charts of the TeraSort experiments

From the analysis of the Tables, it is possible to identify some common trends. Indeed, all experiments present a similar behavior when we consider the Total Map Time : case A was the fastest, followed by C, D, and finally B. In addition, scenarios A and C showed not only the smallest Average Map Times and Standard Deviations among their applications, but also very similar results in these categories, regardless the benchmark. This is due to the fact that the nodes were never overloaded since the scheduler had an updated information before the start of the application. The charts also corroborate this analysis : in all experiments, cases A and C have much lighter tones than the others, meaning that less containers are running simultaneously. It is also worth noting that case C takes approximately twice the time case A used to complete the execution, a result to be expected since case C had half the resources of case A.

The analysis of the number of speculative tasks also gives interesting insights. In the case of TeraSort, the number of speculative tasks is reduced and almost similar in all scenarios. TestDFSIO and WordCount, on the other hand, deploy many more speculative tasks when the system is overloaded (scenarios B and D). This may be due to the factors that trigger a speculative task : a speculative task is launched only after all the tasks for a job have been launched, and then only for tasks that have been running for some time (at least a minute) and have failed to make as much progress, on average, as the other tasks from the job. In the case of TeraSort, tasks evenly require both memory, CPU and I/O, so their execution time is less subject to specific resource

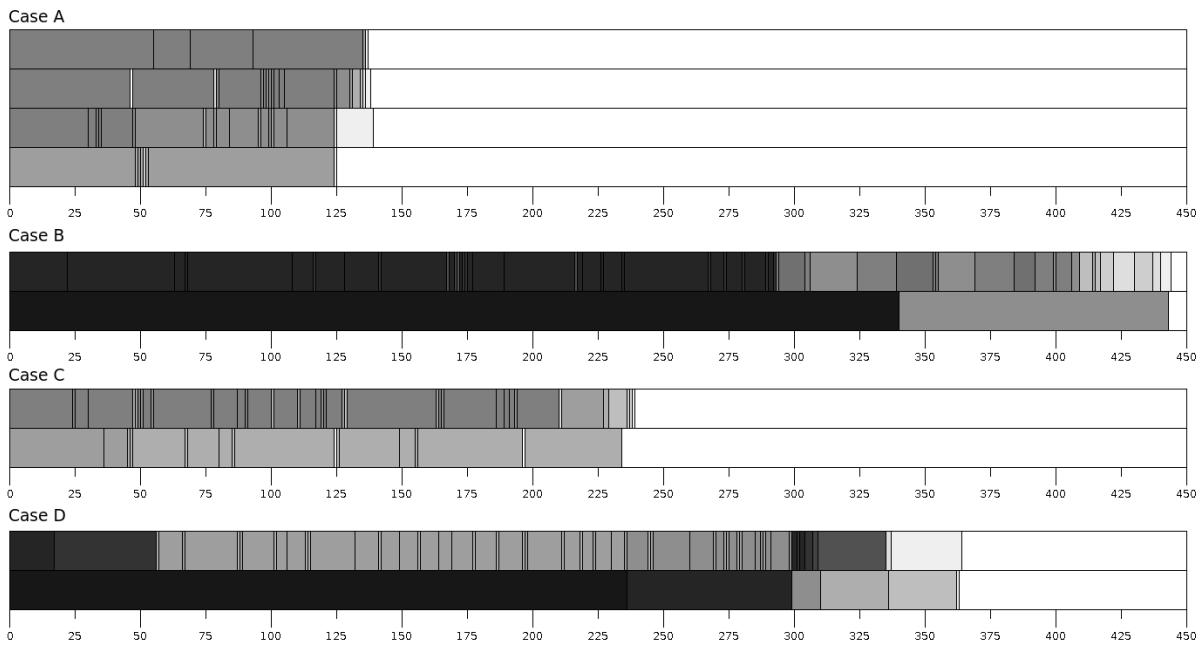


FIGURE 3.11 : Gantt charts of the TestDFSIO experiments

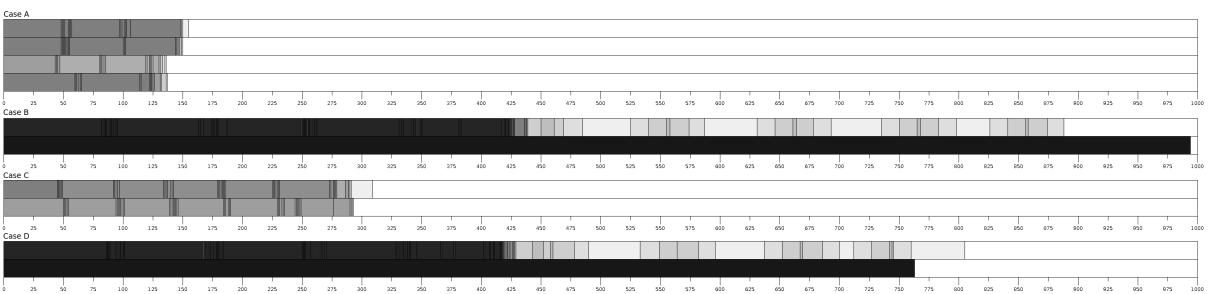


FIGURE 3.12 : Gantt charts of the WordCount experiments

depletion. TestDFSIO and WordCount, on the contrary, rely on more specific resources and thus are more subject to resource overload. In all the cases, the use of context-awareness on scenario D helps minimizing the number of speculative tasks when comparing to the "context-unaware" scenario B.

Concerning the execution flow, as illustrated by the Gantt charts, it is possible to note that both cases B and D have a dark tone at the beginning, meaning that 16 containers are running (twice the real capacity). Also, the first containers took 20 to 50 seconds to complete execution in scenarios A and C in all experiments (cf. the first segment line). On the opposite side, during most of scenario B (and to a lesser extent D) 70 or more seconds are required, evidencing an overload on the nodes. The exception here is the TestDFSIO scenarios B and D, which have segment lines at about 25 seconds. Through further analysis, it is possible to note that all TestDFSIO experiments had at least one segment line near the 25 seconds mark, meaning this was a task very easily processed and somehow not affected by the node overload.

Although both B and D scenarios had the same initial conditions (50% available resources and 100% reported resources), case D took less time to complete in all experiments (20% to 40% faster than B). The reason for this is that the context collector updates the reported resources

in D, allowing the scheduler to reorganize tasks after the first tasks complete. This behavior is easily noted on TeraSort and TestDFSIO charts. Indeed, all D scenarios had high concentration of executing containers at the start of the job but lessened the load over time, while B scenarios had the nodes overloaded until the end due to the absence of updated information. Although the scheduler does not preempt excess containers, it is possible to observe a performance improvement of about 40% on TeraSort and 20% on TestDFSIO experiments as the scheduler avoids overloading the nodes.

There are some specificities that might be pointed out to better understand the charts. On all benchmarks, the first node is seemingly less charged in scenarios B and D after the initial executions. This is mostly due to the fact that after the initial computations this node hosts the Reduce containers, which were not included in the chart. It is important to remember that reducer tasks may start before the end of map tasks and that a reducer may even complete before all maps have finished processing.

Although other specificities about the characteristics of the proposed jobs can be discussed, scenarios C and D show that regular context updates contribute to reduce the execution time on a dynamic Hadoop cluster. We demonstrated that even when starting with the same circumstances as in the worst case (Scenario B), updating the information helps the scheduler to minimize the execution time. Our solution contributes therefore to both provide correct information before the execution starts (Scenario C) and adapt the execution to resources changes (Scenario D).

Environment setup for the shared execution environment

As stated previously, this second set of experiments considered a more realistic environment, where we use 10 slave nodes in which part of the nodes are shared with another user. Indeed, most resource management systems (Slurm, Grid'5000 OAR) allow users to reserve only part of a node (for example, 8 from 16 cores). As the resources are not always evenly shared and some applications don't respect the limits from the allocated resources, this makes an interesting environment for the context-aware scheduler. Also, by increasing the number of slaves nodes, we aim to acquire more data about the solution scalability.

Hence, this second environment presents a Scenario A with fully dedicated nodes, while in Scenarios B, C and D the first half of the nodes are shared. more exactly, each node has a total of 8 GB of memory and 8 cores but in a shared node only 2 GB and 2 cores are available for Hadoop (the remaining 3/4 of the resources are assigned to another user).

Results from shared environment experiments

The results from the TeraSort benchmark on this shared environment is presented in Table 3.5 and Figure 3.13. Once again, we compared the scheduler behavior under four different Scenarios : A, B, C and D, as described in Section 2.4.1. Table 3.5 summarizes the experiments, presenting the total time used by all map tasks, the average execution time, the number of successful map tasks and the number of speculative tasks launched.

Figure 3.13 presents the Gantt charts for each scenario, where each line represents one slave node in the cluster during the experiment. As stated before, Scenarios B, C and D have 5 nodes with shared resources. Hence, Hadoop only access 1/4 of the overall resources on these nodes (for a total of 40 GB and 40 cores on these five nodes, 30 GB and 30 cores are allocated to another user in the cluster). Each line portraits the consolidated resources by that node on a color scale, where the darker the tone, the more containers are executing simultaneously. The

[h]

TABLE 3.5 : Summary of Scaled TeraSort results expressed in seconds.

Scenario	A	B	C	D
Total Map Time (s)	273	2138	743	1421
Avg. Map Time (s)	55.69	322.66	51.49	85.35
# Map Tasks	298	298	298	298
# Speculative Tasks	2	26	1	1

maximum number of containers in these experiments is 8 (present on scenario A), consequently, the darkest tone on the Gantts corresponds to 8 containers. The same way as before, the lines are segmented along the chart to indicate that a container has either finished or started at that moment.

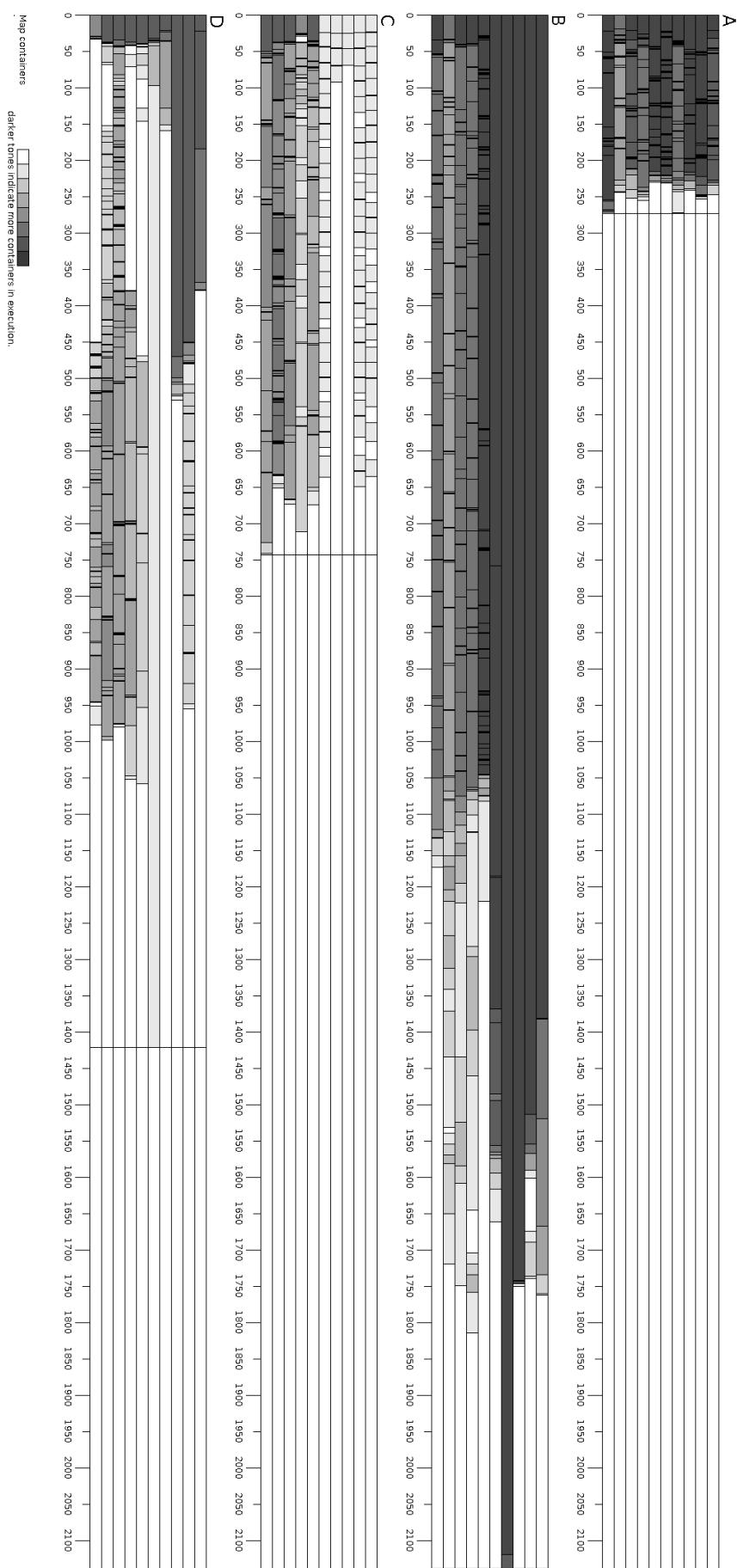


FIGURE 3.13 : Gantt charts for TeraSort on the shared environment experiment

Both Figure 3.13 and Table 3.5 show that the general behavior observed in the previous experiment is maintained. Scenario A has the fastest execution time, followed by Scenarios C, D then B. Scenarios A and C have similar averages on Map Time, followed by Scenarios D then B.

One may wonder why a small number of containers being executed on the first 5 nodes from Scenario C while the second half has a higher load. Actually, the nodes have all resources fully used but the Gantt only shows the Map tasks to simplify the view. If we included the Reduce tasks, the average load would be the same on all these nodes.

Nevertheless, we can observe some anomalies on Scenario D. While it was expected that the containers take a longer time to complete on the shared nodes, sometimes the second half of nodes (the dedicated ones) presented an abnormal small load in some nodes, sometimes even with zero running containers. This behavior can be explained by the way the Capacity Scheduler handles resource information. As explained in Section 2.3, our context-awareness collector updates the resource capabilities from each node to the Capacity Scheduler. More specifically, this information is stored in a global *totalResources* variable that is used by the Capacity Scheduler along with a second *usedResources* variable. If an update reduces the amount of global available resources, there is a transition period in which *usedResources* has a higher value than *totalResources*, leading to a halt on containers deployment as the scheduler believes that there are no free resources to command. Eventually the running containers finish their work and the Capacity Scheduler will be able to start new tasks, normalizing the situation.

There are several strategies to minimize such anomaly. The simpler strategy considers that the update algorithm gradually reduces the available resources in order to prevent a halt on the containers scheduling. Indeed, the reduction pace according to the average execution time of containers, instead of drastically modifying the available resources. The best solution, however, requires a change on the Capacity Scheduler algorithm to allow distinct lists of *usedResources* and *totalResources* by node. Using these detailed lists, the scheduler knows when to stop launching a container in an overloaded node while keeping the deployment on other nodes.

2.5 Discussions

Results presented in Section 2.4 demonstrate that, by observing context information during application execution, it is possible to improve job scheduling on Hadoop framework. Indeed, the experiments prove that by tracking the changes in the available resources (core CPUs and memory), we could keep Hadoop scheduler aware of the current status of these resources. Hadoop scheduler could then base its decisions on more accurate information, corresponding to the real execution context and not to a standard one extracted from configuration files. This allowed us to obtain important performance gains, about 40% in the TeraSort experiment, for instance. This ability to observe context information is particularly important when executing on heterogeneous or non-dedicated clusters, in which resources may fluctuate during the application execution. Offering accurate information about the environment allows Hadoop scheduler to adapt itself to the current environment even if the execution conditions change after the job starts.

These results, although positive, have been based on a limited set of context information, the number of available cores and memory. These criteria correspond to those traditionally used by scheduling, and in this work we focus on changing only the parameter feeding of the scheduler, keeping unchanged the scheduling policy itself. This process allows the assessment of the impact of real-time context information on the performance. Even though, other context

information can be acquired thanks to our context collector and could be used for scheduling purposes. For instance, information such as network bandwidth or latency, available storage and average CPU load could be used.

For instance, researches on Cloud Computing ([59, 3]) propose different scheduling policies to optimize resources use and to respect signed SLA (Service Level Agreement). [59] underlines the potential effects of external factors such as workload and environmental changes on VM allocation and performance. These authors [59] consider workload volatility on resource allocation and reconfiguration in order to avoid under- or over-utilization of resources. In addition to workload information, which can be estimated using average CPU load and memory consumption, other context information can be considered. [3] mention, for instance, the time of the day, other executing applications or the user's location. These authors suggest using context information for an adaptive job scheduling in order to rationalize resource consumption in the cloud. They assume that context-aware job scheduling is necessary in pervasive environments, in which limited mobile-based devices may delegate processing components to a cloud infrastructure. In this case, an adaptive job scheduling can be used to rationalize the consumption of cloud resources (and to avoid job violation events) based on the possible variations of the end user's local context.

These works ([59, 3, 22]) illustrate the possible use of different context information on job scheduling. However, they also underline the necessity of an appropriate scheduling policy, specifically designed in order to explore context information. In the case of a MapReduce application in heterogeneous environments, several context parameters can be considered as relevant, and notably network conditions and data locality in addition to CPU and memory related information (e.g. available cores, CPU load, available memory, etc.). The challenge here is to figure out an adaptive scheduling policy capable of exploring this context information in an appropriate and advantageous way.

Finally, our results (cf. Section 2.4) demonstrate that it is possible to adapt Hadoop scheduler for efficient execution on heterogeneous environments. The possibility of executing MapReduce applications in such environments underlines the potential of using pervasive grids for MapReduce, and more generally, for Big Data.

Indeed, MapReduce applications, and particularly those based on the Hadoop framework, have been heavily used for Big Data to analyze large volumes of data, often unstructured, obtained from various heterogeneous sources (IoT devices, social media, etc.). These applications are mostly executed on homogeneous computing environments such as cloud computing platforms. Different reasons motivate the use of cloud computing for MapReduce applications, among them the elasticity and the cost-effectiveness. Indeed, thanks to cloud computing platforms such as Amazon Elastic MapReduce¹, it is possible to analyze an increasing volume of data, without the need of a local (and costly) infrastructure like a dedicated cluster that remains often underutilized in many organizations. Nevertheless, as underlined by [42], cloud computing also has several trade offs, including security, performance instability and limited latency/throughput network performances. [42], [79] also underline security and privacy issues of cloud platforms. As indicated, several cloud providers find it hard to meet standards for auditability or to comply with legislation in the domain of security and privacy, raising the question of data sensibility.

In other words, although interesting, homogeneous cloud platform are not the only possible solution for data analysis. Several other solutions exist, such as using off the shelf GPU-based

1. <http://aws.amazon.com/elasticmapreduce/>

architectures ([31]) or pervasive grids ([82]). In this paper, we demonstrate that, with an appropriate scheduling mechanism, pervasive grids can be used for Hadoop applications. Indeed, by introducing context-awareness capabilities, pervasive grids may represent a complementary alternative to costly dedicated clusters or to cloud infrastructures, particularly when these cannot be used due to data constraints.

2.6 Conclusion

In this work, we aimed at developing the ability to detect and adapt to resource availability changes on the environment of Apache Hadoop Capacity Scheduler. The improvements were implemented with a light-weight context collector and communication provided by ZooKeeper. These improvements go further than our previous work [20] by including a continuous observation of node capabilities. The results show that this solution can positively impact the execution performance of MapReduce applications, especially in situations where the available resources drop after the beginning of the execution.

While Hadoop does not perform preemption/migration of tasks, the association of a context-aware scheduler and speculative tasks may contribute to circumvent the bottlenecks caused by the resources variability. Our future works will concentrate on modifying the scheduler algorithms, in order to consider a wider collection of context information than the current memory and cores parameters. We believe that additional parameters such as CPU speed, network speed and even battery capacity are essential factors in a pervasive grid. Finally, we intend to evaluate our context-aware scheduler under realistic situations with both MapReduce applications and pervasive grid environments composed of heterogeneous off-the-shelf volunteer computers.

Chapitre 4

CloudFIT : un intergiciel pour le edge-computing et les réseaux IoT

les besoins des architectures (from IoT to the cloud)-> présenter différents aspects genre object to object / object to gateway / object to cloud et réciproquement

Résumé

Dans l'ensemble des éléments qui composent les systèmes informatiques, les réseaux informatiques sont 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.

Dans certains cas, cependant, juste la connaissance des profils de performance peut s'avérer insuffisante, notamment lorsque les applications ont des besoins de fiabilité accrus. Ainsi, l'hétérogénéité dans les réseaux doit aussi s'intéresser à l'observation des ressources afin de détecter des éventuelles défaillances, et de les signaler aux applications et systèmes, qui pour-

ront ainsi réagir et prendre des mesures nécessaires pour assurer ses engagements de fiabilité.

Dans cette première partie, la Section 1 présente les travaux qui ont été menés sur la modélisation et l'évaluation des performances de communication des réseaux. La Section ?? présente les travaux sur la découverte de la topologie et la subséquente classification des ressources. Finalement, la Section ?? présente les travaux sur la détection de défaillances. Tous ces éléments sont connectés aux travaux qui seront présentés dans les chapitres suivants.

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. Cette augmentation est liée, entre autres, à 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 d'informations à grande échelle.

Plusieurs facteurs ont contribué au développement de l'Internet des Objets, dont l'augmentation de la bande passante et de la puissance de calcul des appareils, couplés avec un coût décroissant de capteurs [49]. De nos jours, il est possible d'équiper la quasi-totalité des dispositifs quotidiens d'une interface sans fil, rendant ainsi possible l'interaction entre eux [67]. De telles capacités de communication ouvrent d'innombrables possibilités dans plusieurs domaines tels que la santé et les villes intelligentes, mais soulève également plusieurs défis concernant l'évolutivité, l'hétérogénéité et la dynamicité du réseau. Les principales limitations d'utilisation de ces dispositifs à nos jours ne sont pas liées à leur capacité de communication (WiFi, Bluetooth, etc.), mais surtout à la difficulté d'une application à mieux tirer profit de cette interconnexion à d'autres appareils. Le potentiel de l'IoT ne sera pleinement atteint que si les données issues de l'IoT puissent être analysées et explorées convenablement par les applications.

Actuellement, l'agrégation et l'analyse des données IoT sont effectuées principalement sur les infrastructures déportées de type *cloud computing*. Plusieurs travaux [60, 39, 33] comptent sur ces infrastructures car elles offrent de la puissance de calcul et de la flexibilité pour l'exécution de services et applications [80]. Malgré ses avantages, les plates-formes de type *cloud* ont aussi quelques inconvénients importants. En effet, le transfert de données peut être considérablement coûteux et prendre du temps. En outre, les applications qui dépendent entièrement des services distants peuvent échouer si la connexion est défectueuse ou trop lente.

Par conséquent, nous devons repenser la façon de transmettre, de stocker et d'analyser les données dans ces environnements. Les architectures 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 [66], le *mobile edge computing* [29, 32, 78], le *fog computing* [18] ou bien le *edge-centered computing* [36]. 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.

Pour répondre aux défis de l'IoT, ce travail préconise le développement de plates-formes multi-échelle capables d'assurer la coordination et la redistribution des tâches entre les appareils (en fonction de leurs capacités), améliorant ainsi l'utilisation des ressources et la qualité de service. Nous présentons des solutions architecturales et pratiques à mettre en œuvre pour la création d'une plate-forme multi-échelle adaptée à l'IoT, en utilisant comme base une plate-forme de dispositifs reliés par un réseau P2P.

La suite de cet article est organisée comme suit : la Section 2 présente l'état de l'art sur le calcul dans les environnements pervasifs, ainsi que la définition de l'informatique multi-échelle. La Section 3 analyse les principales exigences pour un cadre multi-échelle et illustre comment une telle approche est mise en œuvre sur la plate-forme de calcul CloudFIT. Finalement, la Section 2.6 présente nos conclusions et travaux futurs.

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.

Les travaux sur l'*edge computing* et le *fog computing* partagent souvent les mêmes définitions [89]. En effet, le *fog computing* a été défini par CISCO [26] 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" [89]. Des exemples de *edge/fog* comprennent des services *fog* [18] et les *cloudlets* [78], tous les deux proposant le déploiement des serveurs de proximité offrant des services avec une latence réduite. A quelques exceptions près, comme [29], 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. [36] explorent une autre facette de l'*edge computing* en se concentrant sur le rôle de l'homme dans la boucle de contrôle. En effet, ces auteurs affirment la nécessité de recentrer le contrôle sur les équipements situés au bord du réseau, au lieu de simplement les considérer comme une première couche de calcul reliée à un réseau plus grand et plus puissant. Malheureusement, dans cette définition les dispositifs IoT ne contribuent pas non plus aux efforts de calcul, étant considérés comme des capteurs/actionneurs pilotés par les interactions entre l'homme et l'*edge*.

Bien que ces travaux préconisent la nécessité d'un environnement informatique de proximité, ils oublient souvent de détailler l'interconnexion ou les exigences de coordination entre les processus. Cela est particulièrement nécessaire dans l'optique de l'IoT, qui impose des défis importants pour l'évolutivité, la dynamicité et l'hétérogénéité des ressources. Afin de gérer efficacement les environnements pervasifs IoT, nous proposons l'utilisation de stratégies multi-échelles pour la gestion et la coordination des périphériques.

Dans la littérature nous trouvons, par exemple, la notion de grille pervasive [66], 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 [83]. 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 datacenter. 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.

Afin de répondre aux différents besoins des architectures et applications IoT, les grilles pervasives doivent 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 fournit un niveau de service supplémentaire qui peut être consulté en fonction du contexte de l'appareil [74, 75]. 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 externalisées. 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.

Dans la section suivante nous énumérons certaines exigences pour le développement d'une plate-forme informatique multi-échelle.

3 Calcul Multi-Échelle

Comme indiqué dans la section précédente, 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. Contrairement au *fog/edge computing*, les grilles pervasives supposent que tous les dispositifs peuvent contribuer au calcul des tâches selon leurs propres capacités et ressources disponibles. Les dispositifs IoT peuvent alors devenir des acteurs actifs de l'environnement et non seulement des clients passifs. Les concepts de grilles pervasives et de calcul multi-échelle peuvent ainsi être utilisés pour intégrer les dispositifs IoT dans l'environnement informatique.

Cependant, afin que les applications puissent profiter pleinement de ces environnements, différents challenges doivent être relevés. Dans ce travail, nous analysons un ensemble d'exigences qui nous paraissent essentielles pour cette intégration et nous illustrons leur prise en charge par une plate-forme de calcul multi-échelle basée sur CloudFIT [83]. CloudFIT est un middleware de calcul P2P basé sur le paradigme FIIT (*Finite Independent Irregular Tasks*).

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

La version actuelle prend en charge le système TomP2P² ainsi que son API DHT (basée sur Kademlia), offrant des possibilités intéressantes pour la gestion des données. CloudFIT est développé en Java et peut donc être déployé sur une large gamme de dispositifs (une version Android est en cours de développement). Afin de rendre son déploiement simple et rapide, nous avons pris soin de minimiser les dépendances envers les bibliothèques tiers, ce qui permet le lancement de la plate-forme avec un simple fichier *jar*.

La flexibilité de lancement de CloudFIT permet son exécution directement sur certains appareils IoT récents, souvent localisés au bord du Edge. Dans le cas des capteurs isolés ou de micro-contrôleurs de type Arduino, incapables de tourner le code Java, CloudFIT peut servir de *gateway*, grâce à une interface front-end permettant le stockage des données et le déclenchement d'actions, comme illustré sur le côté gauche de la Fig. 4.1.

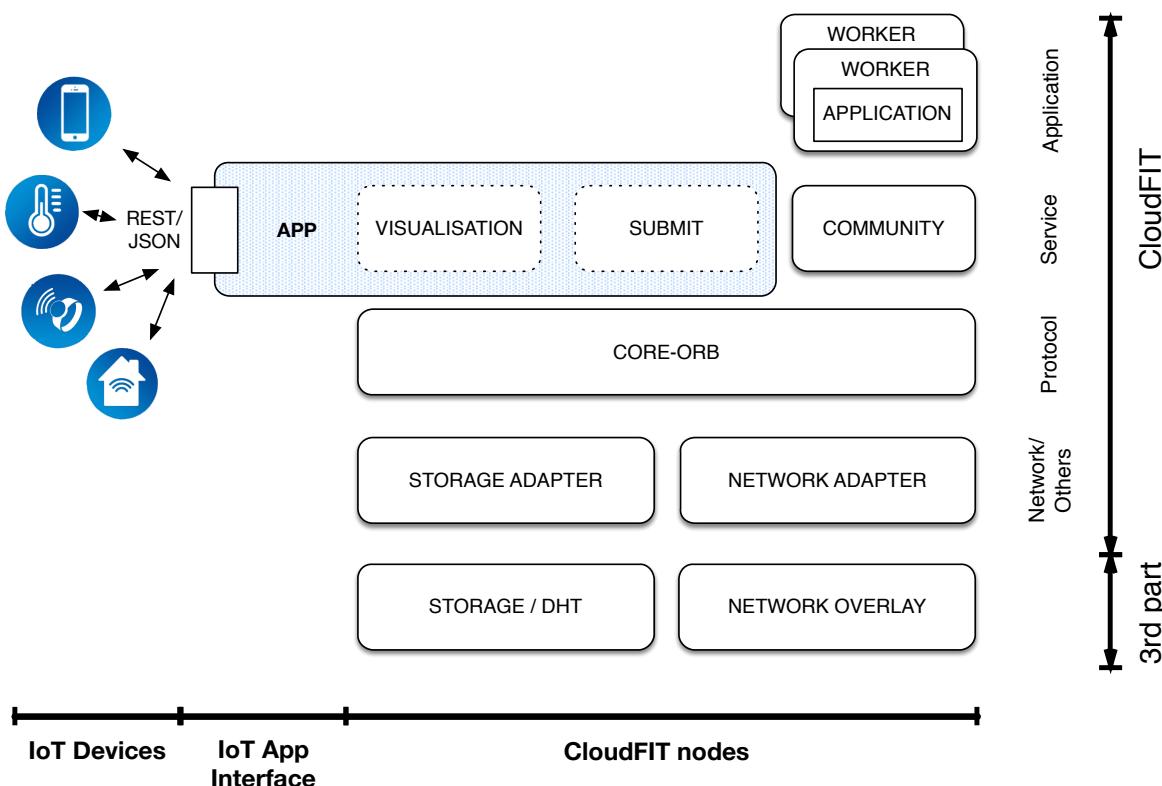


FIGURE 4.1 : La pile d'exécution CloudFIT et l'interface pour les dispositifs IoT

Les prochaines sections décrivent ainsi les défis pour la mise en place d'une plate-forme multi-échelle et comment nous avons adapté CloudFIT à ces fins.

3.1 Communication, coordination et clustering

L'un des principaux défis avec le calcul multi-échelle dans un environnement IoT est celui de permettre à la fois un contrôle précis sur les ressources et une interaction simplifiée avec le reste du réseau. En effet, les environnements IoT se caractérisent par leur échelle très variable, pouvant varier de quelques objets à plusieurs milliards. En concevant chaque couche comme

2. <http://tomp2p.net>

un cluster, nous pouvons utiliser un réseau P2P soutenant le déploiement de chaque couche. Cependant, les réseaux P2P les plus connus organisent les nœuds indistinctement de leur emplacement réel, ce qui empêche l'établissement d'un regroupement de nœuds pour fournir des services à faible latence. Pour contourner ces inconvénients, nous considérons que le réseau P2P doit être enrichi par l'utilisation de 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.

Plusieurs approches de clustering sont proposées dans la littérature [48] 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 comme celui de l'IoT, vu qu'elles ne permettent pas d'exprimer les besoins du calcul multi-échelle. Au contraire, le regroupement des nœuds doit être défini à la fois de manière à co-localiser les données et les ressources de calcul nécessaires et aussi à définir des couches de calcul reliant les dispositifs proches et plus éloignés (jusqu'au cloud). Ainsi, afin de s'adapter à l'hétérogénéité des environnements pervasifs et l'IoT, les métriques de clustering doivent être étendues pour 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.

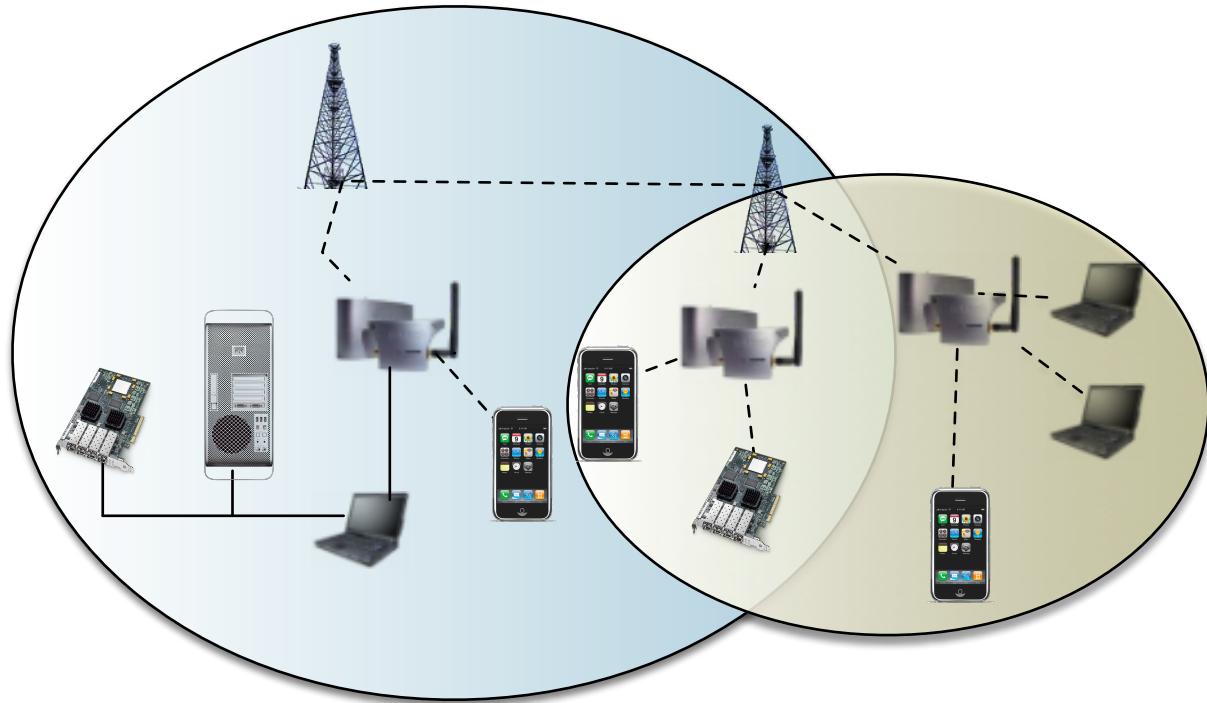


FIGURE 4.2 : Des communautés interconnectées dans une plate-forme multi-échelle

Dans le cas de CloudFIT, ce clustering peut être mis en œuvre grâce au concept de *communautés*, des sous-ensembles de nœuds qui peuvent être adressés séparément et donc utilisés pour répartir/cloisonner les opérations. Dans le cas d'un clustering automatique, la liste de membres est gérée par le *cluster-head*. Les membres de la liste peuvent donc être contactés

en utilisant des procédures spécifiques telles que les diffusions structurées de TomP2P. 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 Fig. 4.2. 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.

3.2 Contexte et ordonnancement

Dans les environnements pervasifs, la sensibilité au contexte est essentielle pour la planification des tâches vu que la performance varie beaucoup d'un appareil à l'autre [29]. Les informations contextuelles telles que la puissance de calcul, la mémoire disponible et l'espace de stockage peuvent être utiles pour améliorer l'exécution en assignant des tâches aux dispositifs les plus adéquats. Avec d'autres paramètres tels que la proximité géographique, la latence réseau et le niveau de fiabilité d'un nœud, la sensibilité au contexte est un outil nécessaire à la clusterisation des nœuds et à la composition de services multi-échelle.

Bien que la collecte d'informations de contexte et sa mise en œuvre est au-delà de la portée de cet article, il faut noter que les algorithmes d'ordonnancement doivent néanmoins être assez permissives, avec l'exclusion des nœuds uniquement lorsque les exigences spécifiques de l'application ne sont pas respectées (par exemple, un minimum de capacité mémoire). En effet, la dynamité des environnements pervasifs due entre autres à la mobilité de certains appareils, rend particulièrement difficile d'anticiper quelles ressources seront disponibles et dans quelle mesure elles le seront. Dans ces conditions, il est préférable d'avoir des nœuds lents mais qui contribuent à l'effort que d'avoir un blocage par manque de ressources ou une surcharge trop importante des nœuds rapides [21, 90].

Les applications CloudFIT étant décrites par un ensemble fini de tâches qui peuvent être réparties entre les nœuds, des informations de base telles que le nombre de tâches parallèles qu'un nœud peut supporter sont déjà utilisées par l'ordonnanceur. Si l'ordonnanceur par défaut utilise une approche totalement distribuée (exécution *best-effort* associée à la diffusion des tâches finies), il peut être enrichi avec d'autres éléments tels que la mémoire disponible ou la vitesse relative des CPUs. La collecte de ces éléments peut se faire grâce au collecteur de contexte que nous avons proposé et utilisé dans [21], présenté sur la figure 4.3.

3.3 Accès aux données

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 [83] 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. 4.5. 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*.

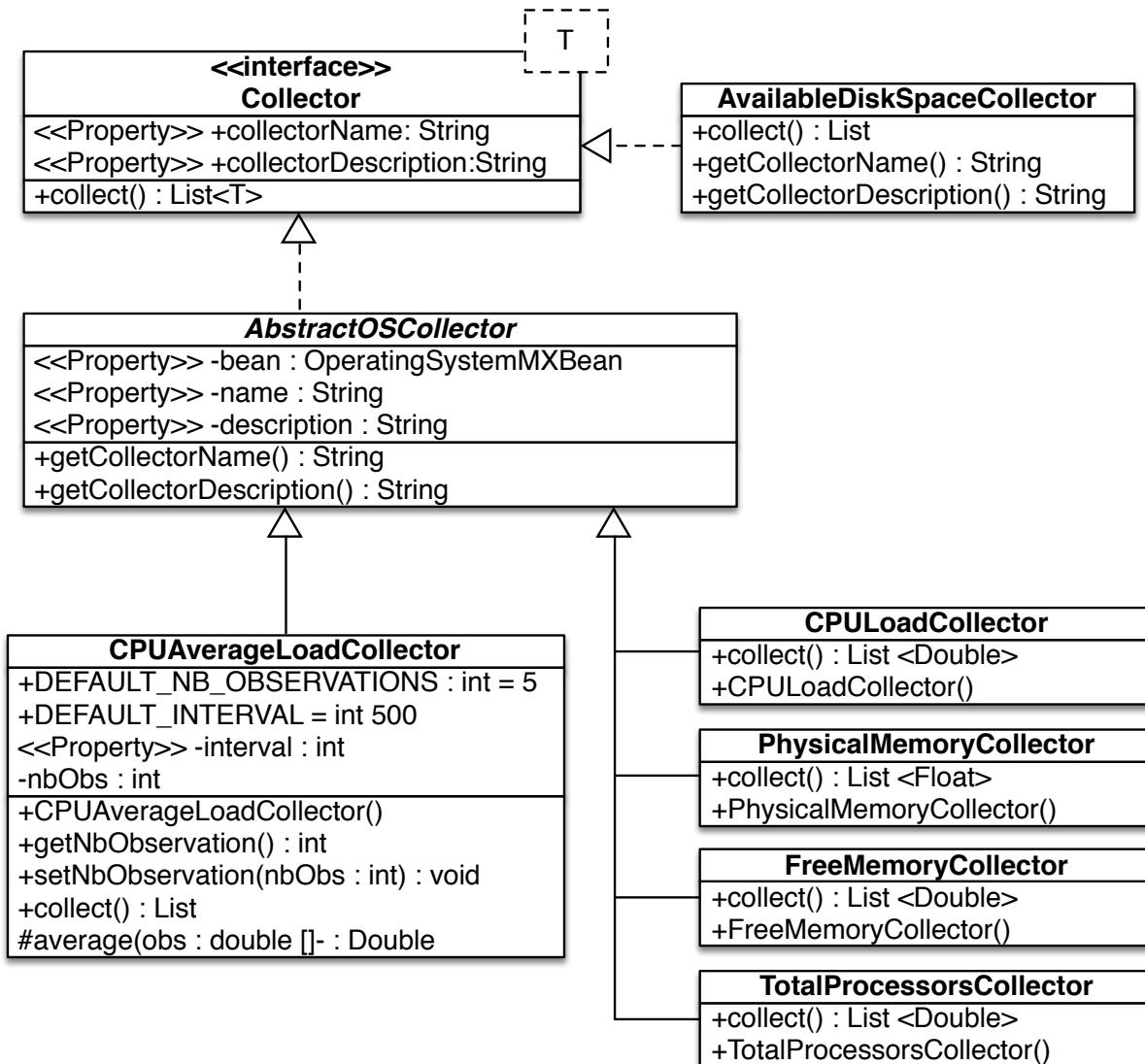


FIGURE 4.3 : Architecture pour un collecteur de contexte [19]

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 situés à la frontière de l'Edge Computing. Ainsi, par exemple, un Raspberry Pi est fortement penalisé 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 nœuds puissent choisir d'agir seulement en tant que clients distants. Ces nœuds peuvent donc interroger la surcourche 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

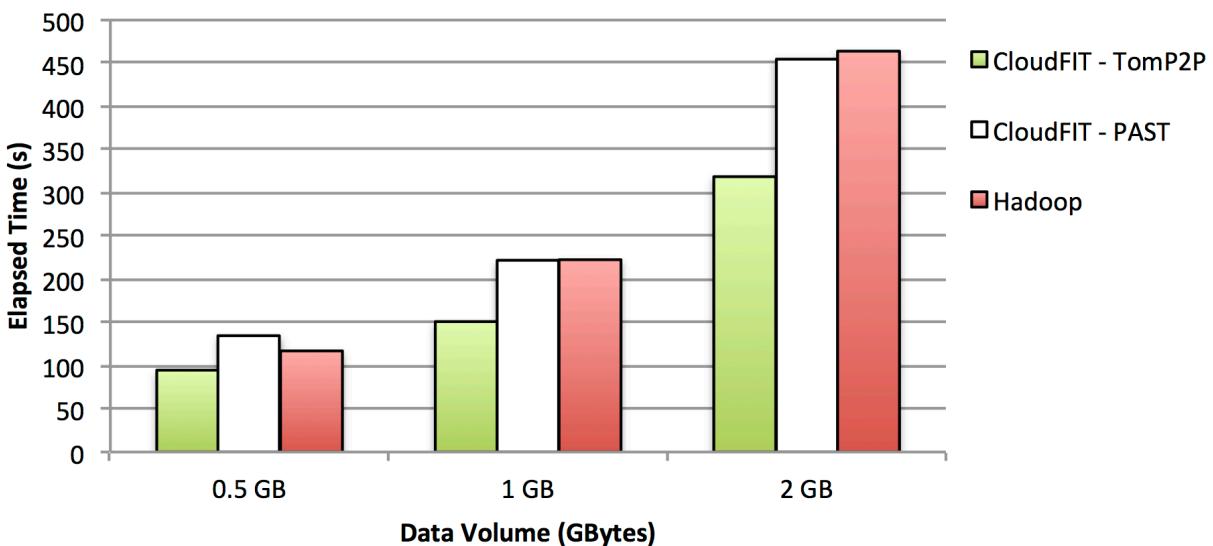


FIGURE 4.4 : Comparaison des temps d'exécution de WordCount avec CloudFIT et Hadoop

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 [92], rendant difficile l'optimisation des transferts réseau.

A l'instar d'autres systèmes de P2P, La bibliothèque Tom P2P utilisée par CloudFIT ne permet pas encore à un nœud de faire la différence entre les données locaux ou distants. Nous sommes en contact avec l'équipe de développement TomP2P afin d'intégrer une telle fonctionnalité dans les futures versions, mais en attendant nous avons développé une stratégie pour renforcer la proximité des données grâce à un calcul personnalisé de la clé de localisation des ressources.

Ceci est possible car 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

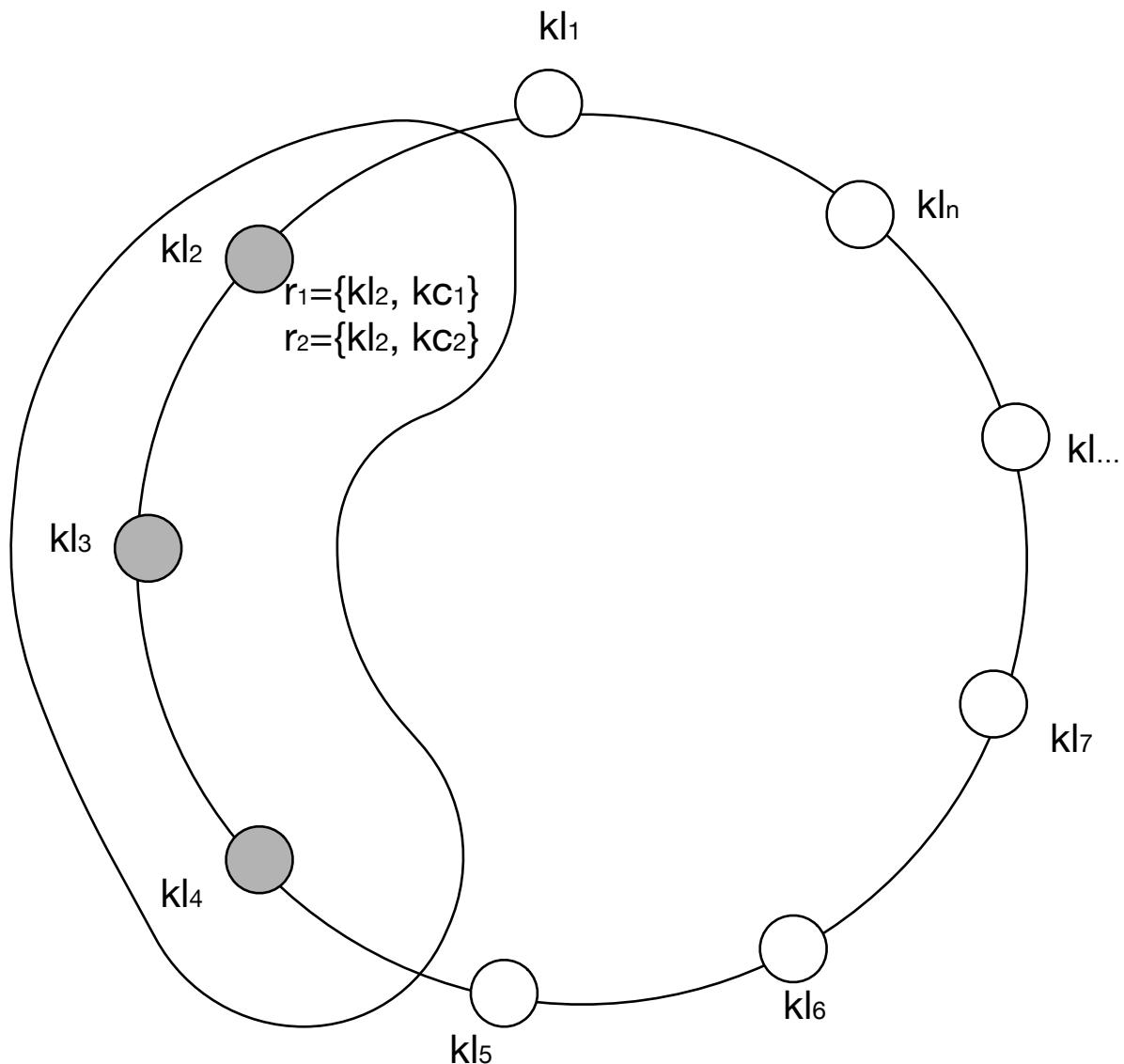
La clé de localisation est celle qui s'approche le plus des clés DHT traditionnelles, ayant par fonction l'association d'une ressource (copie primaire ou index) au nœud avec l'ID le plus proche. Sans aucune instruction supplémentaire, la clé de localisation et la clé de contenu sont les mêmes, mais peuvent 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. ?? 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 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.



$$C_1 = \{kl_2, kl_3, kl_4\}$$

$$kc_3 = \text{hash}(r_3)$$

$$kl' = \text{hash}(kc_3, C_1)$$



$$r_3 = \{kl', kc_3\}$$

FIGURE 4.5 : Cartographie des ressources renforçant la data-locality

Conclusion et perspectives

Publications personnelles

Articles dans des revues avec comité de lecture

Internationales

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

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

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 Parallélisme, 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 LNCS 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), Compiègne, 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, Morocco, 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 CoreGRID 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

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 Parallelisme (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 Anag-

nostopoulos, 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 interactions 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

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. Cérin, S. Petiton, J. Gustedt, C. Blanchet, C. Randriamaro, S. Tixeuil. "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] 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.
- [2] Apache. Apache hadoop, 2014. <http://hadoop.apache.org/docs/r2.6.0/index.html>. Last access : November 2014.
- [3] M.D. Assuncao, M.A.S. Netto, F. Koch, and S. Bianchi. Context-aware job scheduling for cloud computing environments. In *Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on*, pages 255–262, Nov 2012.
- [4] 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.
- [5] 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.
- [6] 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.
- [7] 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.
- [8] 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.
- [9] 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.
- [10] Olivier Beaumont and Loris Marchal. Pipelining broadcasts on heterogeneous platforms under the one-port model. Technical report, LIP, ENS Lyon, France, 2004.
- [11] Olivier Beaumont, Loris Marchal, and Yves Robert. Broadcast trees for heterogeneous platforms. Technical report, LIP, ENS Lyon, France, 2004.
- [12] 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.
- [13] Massimo Bernaschi and Giulio Iannello. Collective communication operations : Experimental results vs. theory. *Concurrency : Practice and Experience*, 10(5) :359–386, April 1998.

- [14] 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.
- [15] 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.
- [16] Manav Bhatia, Vishwas Manral, and Yasuhiro Ohara. IS-IS and OSPF Difference Discussion. IETF Internet Draft, January 2006.
- [17] G. Blair and P. Grace. Emergent middleware : Tackling the interoperability problem. *Internet Computing*, 16(1) :78–82, January 2012.
- [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] Guilherme W. Cassales, Andrea S. Charao, Manuele Kirsch Pinheiro, Carine Souveyet, and Luiz A. Steffenel. Context-aware scheduling for apache hadoop over pervasive environments. *Procedia Computer Science*, 52 :202 – 209, 2015. The 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015).
- [20] 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.
- [21] Guilherme W. Cassales, Andrea Schwertner Charão, Manuele Kirsch-Pinheiro, Carine Souveyet, and Luiz-Angelo Steffenel. Improving the performance of apache hadoop on pervasive environments through context-aware scheduling. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–13, 2016.
- [22] 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.
- [23] 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.
- [24] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2) :225–267, 1996.
- [25] 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.
- [26] 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.
- [27] 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.

- [28] Jeffrey Dean and Sanjay Ghemawat. Mapreduce : Simplified data processing on large clusters. *Commun. ACM*, 51(1) :107–113, January 2008.
- [29] 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.
- [30] EW. Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the ACM*, 17 :643–644, 1974.
- [31] TiagoAugusto Engel, AndreaSchwertner Charão, Manuele Kirsch-Pinheiro, and Luiz-Angelo Steffenel. Performance improvement of data mining in weka through multi-core and gpu acceleration : opportunities and pitfalls. *Journal of Ambient Intelligence and Humanized Computing*, 6(4) :377–390, June 2015.
- [32] ETSI. Mobile-edge computing - introductory technical white paper. https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobile-edge_Computing_-_Introductory_Technical_White_Paper_V1%2018-09-14.pdf.
- [33] 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).
- [34] 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.
- [35] L. Garcés-Erice, P. Felber, E. W. Biersack, G. Urvoy-Keller, and K. W. Ross. Data indexing in peer-to-peer dht networks. In *ICDCS 2004*, pages 200–208, 2004.
- [36] 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.
- [37] Grid'5000. Grid 5000, 2013. <https://www.grid5000.fr/>, Last access : July 2014.
- [38] Duncan Grove. *Performance Modelling of Message-Passing Parallel Programs*. PhD thesis, University of Adelaide, 2003.
- [39] Jayavardhana 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.
- [40] James Hamilton. Hadoop wins terasort, Jul 2008. <http://perspectives.mvdirona.com/2008/07/hadoop-wins-terasort/>, Last access : September 2015.
- [41] R.W. Hockney. The communication challenge for MPP : Intel Paragon and Meiko CS-2. *Parallel Computing*, 20 :389–398, 1994.
- [42] P. Hofmann and D. Woods. Cloud computing : The limits of public clouds for business applications. *Internet Computing, IEEE*, 14(6) :90–93, Nov 2010.
- [43] 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.
- [44] 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.
 - [45] Lars Paul Huse. Collective communication on dedicated clusters of workstations. In *Proceedings of the European PVM/MPI Users' Group Meeting*, pages 469–476, 1999.
 - [46] 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.
 - [47] Minghui Ji. Hierarchical bidirectional chord. In *2010 International Conference on Educational and Information Technology (ICEIT 2010)*, pages 486–489. IEEE Xplore, 2010.
 - [48] 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.
 - [49] Michelle Jones. Internet of things : Shifting from proprietary to standard. <http://www.valuewalk.com/2014/07/internet-of-things-iot/>.
 - [50] M. Kessis, C. Roncancio, and A. Lefebvre. Dasima : A flexible management middleware in multi-scale contexts. In *Sixth International Conference on Information Technology : New Generations*, (ITNG'09), pages 1390–1396, April 2009.
 - [51] 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.
 - [52] 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.
 - [53] 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.
 - [54] Pangfeng Liu and Tzu-Hao Sheng. Broadcast scheduling optimization for heterogeneous cluster systems. In *Proceedings of 12th SPAA*, pages 129–136, 2000.
 - [55] N. Lynch. *Distributed algorithms*. Morgan Kaufman, 1996.
 - [56] Zakaria Maamar, Djamel Benslimane, and Nanjangud C. Narendra. What can context do for web services ? *Commun. ACM*, 49(12) :98–103, December 2006.
 - [57] 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.
 - [58] Gabriel Mateescu. A method for MPI broadcast in computational grids. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2005)*, 2005.

- [59] M. Maurer, I. Brandic, and R. Sakellariou. Self-adaptive and resource-efficient sla enactment for cloud computing infrastructures. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 368–375, June 2012.
- [60] 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.
- [61] J. Moy. OSPF Version 2. RFC 2328 (INTERNET STANDARD), April 1998. Updated by RFCs 5709, 6549, 6845, 6860.
- [62] 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.
- [63] Aline P. Nascimento, Cristina Boeres, and Vinod E. F. Rebello. Dynamic self-scheduling for parallel applications with task dependencies. In *Proceedings of the 6th International Workshop on MGC*, MGC ’08, pages 1 :1–1 :6, New York, NY, USA, 2008.
- [64] 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.
- [65] D. Oran. OSI IS-IS Intra-domain Routing Protocol. RFC 1142 (Informational), February 1990.
- [66] 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.
- [67] Koosha Paridel, Engineer Bainomugisha, Yves Vanrompay, Yolande Berbers, and Wolfgang De Meuter. Middleware for the internet of things, design goals and challenges. *ECEASST*, 28, 2010.
- [68] Zhuo Peng, Zhenhua Duan, Jian-Jun Qi, Yang Cao, and Ertao Lv. Hp2p : A hybrid hierarchical p2p network. In *ICDS’07*, pages 18–, Washington, DC, USA, 2007. IEEE.
- [69] James Lyle Peterson. *Petri Net Theory and Modeling of Systems*. Prentice-Hall, Englewood-Cliffs, New Jersey, 1981.
- [70] 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.
- [71] 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.
- [72] 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.

- [73] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. *ACM SIGCOMM Computer Communication Review*, 31(4) :161–172, 2001.
- [74] 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.
- [75] 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.
- [76] 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.
- [77] Mahadev Satyanarayanan. Mobile computing : the next decade. *SIGMOBILE Mobile Computing and Communication Review*, 15(2) :2–10, 2011.
- [78] 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.
- [79] Eric E. Schadt, Michael D. Linderman, Jon Sorenson, Lawrence Lee, and Garry P. Nolan. Computational solutions to large-scale data management and analysis. *Nature Reviews Genetics*, 11(9) :647–657, Sept 2010.
- [80] 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.
- [81] 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.
- [82] 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).
- [83] 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.
- [84] STIC-AmSud. PER-MARE project, 2014. <http://cosy.univ-reims.fr/PER-MARE>, Last access : July 2014.
- [85] LAM-MPI Team. Lam/mpi version 7. <http://www.lam-mpi.org>, 2004.
- [86] 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.

-
- [87] 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.
 - [88] Sathish Vadhiyar, Graham Fagg, and Jack Dongarra. Automatically tuned collective communications. In *Proceedings of the Supercomputing 2000*. IEEE Computer Society, 2000.
 - [89] 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.
 - [90] Weinan Wang, Matthew Barnard, and Lei Ying. Decentralized scheduling with data locality for data-parallel computation on peer-to-peer networks. In *Proceedings of Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Monticello, IL, USA, 2015.
 - [91] Rich Wolski, Neil Spring, and Chris Peterson. Implementing a performance forecasting system for metacomputing : The network weather service. In *Proceedings of the Supercomputing*, 1997.
 - [92] 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.
 - [93] 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.
 - [94] 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.
 - [95] 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.