Étude des problèmes d'ordonnancement sur des plates-formes hétérogènes en modèle multi-port

Hejer Rejeb

THÈSE

présentée à

L'UNIVERSITÉ BORDEAUX I

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

Par **Hejer REJEB**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : **Informatique**

Étude des problèmes d'ordonnancement sur des plates-formes hétérogènes en modèle multi-port

---

Soutenue le : 30 août 2011

Après avis des rapporteurs :

Alexey Lastovetsky	Senior Lecturer	University College, Dublin
Laurent Philippe	Professeur	Université de Franche-Comté

Devant la commission d'examen composée de :

Alexey Lastovetsky	Senior Lecturer	Rapporteur
Laurent Philippe	Professeur	Rapporteur
Nicolas Bonichon	Maître de Conférences	Examinateur
Colette Johnen	Professeur	Examinateur
Miroslaw Korzeniowski	Assistant Professor	Examinateur
Olivier Beaumont	Directeur de Recherche	Directeur de thèse

– **2011** –

Résumé : Les travaux menés dans cette thèse concernent les problèmes d'ordonnancement sur des plates-formes de calcul dynamiques et hétérogènes et s'appuient sur le modèle de communication "multi-port" pour les communications. Nous avons considéré le problème de l'ordonnancement des tâches indépendantes sur des plates-formes maîtres-esclaves, dans les contextes statique et dynamique. Nous nous sommes également intéressé au problème de la redistribution de fichiers répliqués dans le cadre de l'équilibrage de charge. Enfin, nous avons étudié l'importance des mécanismes de partage de bande passante pour obtenir une meilleure efficacité du système.

Discipline : Informatique

Mots clefs : Ordonnancement
Systèmes distribués
Modèle multi-port
Qualité de service
Système de stockage distribué
TCP
Partage de bande passante
Re-construction des fichiers
Re-configuration de système
Tâches indépendantes
Diffusion

LaBRI, INRIA Bordeaux
Université Bordeaux 1
351 cours de la Libération,
33405 Talence Cedex (FRANCE)

Abstract :

The results presented in this document deal with scheduling problems on dynamic and heterogeneous computing platforms under the "multiport" model for the communications. We have considered the problem of scheduling independent tasks on master-slave platforms, in both offline and online contexts. We have also proposed algorithms for replicated files redistribution to achieve load balancing. Finally, we have studied the importance of bandwidth sharing mechanisms to achieve better efficiency.

Discipline : Computer-Science

Keywords : Scheduling
Distributed systems
Multiport Model
Quality of Service
Dstributed Storage System
TCP
Bandwidth Sharing
File Reconstruction
System Reconfiguration
Independant Tasks Scheduling
Broadcasting

LaBRI, INRIA Bordeaux
Université Bordeaux 1
351 cours de la Libération,
33405 Talence Cedex (FRANCE)

Table des matières

Chapitre 1

Introduction

L'accroissement des besoins en puissance de calcul dans divers domaines, tel que le traitement de signal et l'imagerie, associé à la fin de la loi de Moore, rend le recours au parallélisme indispensable. Le parallélisme apparaît à tous les niveaux et à une échelle nouvelle. D'un coté du spectre, les processeurs du commerce ont tous plusieurs coeurs et les prototypes en comportent aujourd'hui quelques centaines. De l'autre coté du spectre, les plates-formes de calcul volontaire, comme BOINC [5] ou Folding@home [65], regroupent des millions d'utilisateurs en permanence qui mettent leurs ressources de calcul à disposition. Ce changement d'échelle pose des problèmes de plusieurs types par rapport aux travaux classiques en parallélisme et en ordonnancement. Tout d'abord, il n'est plus raisonnable de supposer que la topologie de la plate-forme de calcul est complètement connue et on doit s'appuyer sur des modèles simplifiés du réseau. Dans le contexte qui nous intéresse, les modèles doivent être réalistes (c'est à dire capturer les caractéristiques essentielles du comportement du réseau), instanciables (les paramètres du modèles doivent pouvoir être évalués en un temps compatible avec le dynamisme de la plate-forme) et théoriquement pertinent (c'est à dire suffisamment simples pour conduire à une complexité algorithmique raisonnable pour des problèmes simples). D'autre part, la tolérance aux pannes devient incontournable puisque la multiplication du nombre de composants (calcul et réseau) rend incontournable la prise en compte des défaillances. Dans cette thèse, nous nous intéressons dans la partie I à des problèmes d'ordonnancement de tâches indépendantes, c'est à dire à des problèmes dans lesquels la tolérance aux pannes est prise en compte au niveau algorithmique, en éliminant les dépendances entre les tâches de sorte qu'il est seulement nécessaire de redémarrer les tâches qui ont échoué. Nous nous intéresserons également dans la partie II au placement de réplicats dans un réseau de manière à assurer la disponibilité des données. En plus du changement d'échelle, on assiste à une augmentation de l'hétérogénéité des composants de calcul (PCs, consoles de jeu, peut-être bientôt téléphones pour le calcul volontaire, CPU, GPU, GPGPU pour les processeurs classiques).

La situation est donc caractérisée par l'hétérogénéité des ressources de calcul et de communication et par le nombre et le dynamisme des ressources participantes. Le but de cette thèse est de démontrer que dans ces conditions, il est nécessaire de concevoir des ordonnancements, mais que ceux ci doivent avoir de nouvelles caractéristiques. Par exemple, nous montrons dans la partie I que le problème essentiel

est l'allocation de ressources (en l'occurrence de clients à des serveurs) plus que l'ordonnancement des tâches elles-mêmes, pour lequel on peut s'appuyer sur des mécanismes dirigés par la demande. Par contre, il est essentiel de concevoir des algorithmes ayant des garanties de performance y compris dans un cadre dynamique on-line et nous montrons dans la partie I que de telles garanties peuvent dans certains cas être maintenues à faible coût. La partie II étudie le placement de réplicats dans un réseau en se concentrant particulièrement sur le problème d'ordonnancement qui se pose quand des ressources sont ajoutées ou retirées au système. Un des apports important de ce chapitre et d'avoir mis en évidence le rôle joué par le partage de bande passante entre des communications concurrentes et d'avoir imposé une réflexion sur la nécessité de la conception d'algorithmes d'ordonnancements et la mise en place de mécanismes permettant un partage prescrit des ressources réseau entre des activités concurrentes. Ces constations motivent la dernière partie (partie III) de cette thèse, qui démontrent sur trois exemples classiques l'importance de la prise en compte de ce partage pour obtenir de bonnes performances.

Enfin, l'ensemble des études menées dans cette thèse s'appuie sur le modèle "multiport borné" pour les communications. Aujourd'hui, il n'est plus raisonnable, quelle que soit l'échelle à laquelle on se place (d'un processeur multicore à une plate-forme de calcul volontaire) de supposer que les communications se déroulent sans contentions et sans coûts. Le modèle 1-port a été largement étudié dans la littérature et il reste pertinent dans le cas d'applications MPI s'exécutant sur des machines dont la topologie est bien connue. Toutefois, il ne permet pas de modéliser de manière raisonnable les communications dans une plate-forme à plus grande échelle, dans laquelle Internet est le réseau sous-jacent (et donc dont la topologie ne peut pas être découverte). De plus, il n'est pas raisonnable de supposer, dans le cadre de ressources très hétérogènes, qu'un serveur à 1 GO/s sera bloqué pendant 1s pendant l'envoi de 1MO à un client ADSL dont la capacité de communication en entrée est de 1MO/s. Il est nécessaire dans un tel contexte de supposer qu'un noeud peut être engagé dans plusieurs communications simultanément. Toutefois, par souci de réalisme, il convient souvent d'ajouter une contrainte sur le nombre maximum de communications simultanées, par l'ajout d'une contrainte de degré aux ressources de communication.

Du point de vue des techniques utilisées, le lecteur trouvera dans cette thèse tous les ingrédients qu'il recherche en lisant des travaux d'ordonnancement : des preuves de NP-Complétude, des algorithmes d'approximation, des algorithmes offline et online, l'utilisation de l'augmentation de ressources, la comparaison de classes d'algorithmes en utilisant des simulations.

1.1 Conributions

La contribution principale de ce travail est la conception d'algorithmes d'ordonnancement pour des plates-formes à grande échelle. Dans la suite, nous résumons les trois grandes parties qui ont été étudiées durant cette thèse.

1.1.1 Distribution de tâches indépendantes

Dans cette partie I, nous avons considéré l'allocation statique (offline) et dynamique (online) d'un très grand nombre de tâches identiques et indépendantes sur une plate-forme maîtres-esclaves. Initialement, plusieurs nœuds maîtres possèdent ou génèrent les tâches qui sont ensuite transférées et traitées par des nœuds esclaves. L'objectif est de maximiser le débit (c'est-à-dire le nombre fractionnaire de tâches qui peut être traité en une unité de temps, en régime permanent, par la plate-forme). Nous considérons que les communications se déroulent suivant le modèle multi-port à degré borné, dans lequel plusieurs communications peuvent avoir lieu simultanément sous réserve qu'aucune bande passante ne soit dépassée et qu'aucun serveur n'ouvre simultanément un nombre de connections supérieur à son degré maximal. Sous ce modèle, la maximisation du débit correspond au problème Maximum-Througput-Bounded-Degree. Nous avons montré que le problème (offline) est NP-Complet au sens fort mais qu'une augmentation de ressources minimale (de 1) sur le degré maximal des serveurs permet de le résoudre en temps polynomial. Ensuite, nous avons considéré une extension de ce problème à la situation plus réaliste, dans le contexte des plates-formes de calcul à grande échelle, dans laquelle les nœuds rejoignent et quittent dynamiquement la plate-forme à des instants arbitraires (problème *online*). Nous avons montré qu'aucun algorithme complètement à la volée (c'est-à-dire qui n'autorise pas les déconnections) ne peut conduire à un facteur d'approximation constant, quelle que soit l'augmentation de ressources utilisée. Ensuite, nous montrons qu'il est en fait possible de maintenir à tout instant la solution optimale (avec une augmentation de ressource additive de 1) en ne réalisant à chaque modification de la plate-forme qu'une déconnexion et qu'une nouvelle connexion par maître. Ces travaux ont été acceptés et présentés dans deux conférences internationales ICPADS 2009 [4] en Chine et PDP 2010 [7] en Italie, un Workshop international avec comité de sélection au Chili [2] et une conférence nationale ALGOTEL [3]. Ils ont fait l'objet aussi de deux rapports de recherche [5].

1.1.2 Redistribution de fichiers avec Yahoo !

Cette partie de la thèse II a fait l'objet d'une publication dans la conférence IEEE ICPADS en décembre 2008 [1] et s'inscrit dans une collaboration de l'équipe CEPAGE avec Cyril Banino-Rokkones de Yahoo ! Research à Trondheim (Norvège). On s'est intéressé au système distribué de stockage de fichiers de Yahoo ! *Vespa*, qui est utilisé entre autre pour le stockage des mails et pour des applications comme Flickr. Les différents fichiers sont regroupés sous la forme de buckets, qui sont stockées (en utilisant des mécanismes d'équilibrage de charge inspirés des réseaux Pair à Pair) sur un ensemble de disques. En outre, ces fichiers sont répliqués de manière à assurer à la fois l'équilibrage de charge dynamique des requêtes et la tolérance aux pannes.

Nous avons particulièrement considéré le problème d'ordonnancement lié à l'ajout d'un disque dans le système. Lors d'une telle opération, un certain nombre de buckets doivent être transférées sur le nouveau disque et le problème consiste à choisir, parmi les réplicats existants dans le systèmes, lesquels devraient être utilisés. On

suppose la plate-forme complètement hétérogène (c'est-à-dire chaque disque est caractérisé par sa taille, sa bande passante entrante et sortante). Dans ces conditions, nous avons montré qu'il est possible de construire en temps polynomial une solution optimale dans laquelle certains fichiers peuvent être transférés depuis plusieurs disques. Pour des raisons d'implémentation, cette solution est difficile à mettre en place et nous avons donc considéré le problème dans lequel un fichier ne peut être transféré que depuis un seul disque source. Nous avons montré que ce problème est NP-Complet au sens fort et nous avons proposé un nouvel algorithme d'approximation qui s'appuie sur une relaxation de la solution fractionnaire optimale. Nous avons prouvé un facteur d'approximation général pour cet algorithme (dont le ratio dépend de l'hétérogénéité des bandes passantes et des tailles des buckets), mais qui est nettement meilleur que les résultats de la littérature (le meilleur facteur d'approximation est 2 par Shmoys et al. [**?**] dans les conditions d'utilisation de *Vespa*. Enfin, nous avons montré, par simulation en utilisant SimGrid, l'efficacité de l'approche proposée. Actuellement, nous travaillons à l'adaptation de ces résultats dans le cas de l'ajout simultané de plusieurs disques.

1.1.3 Mécanismes de contrôle de bande passante

Dans cette partie III, nous avons étudier les mécanismes de contrôle de bande passante sur les plates-formes hétérogènes à large échelle. Et nous avons souligné l'importance d'utiliser de tels mécanismes pour une meilleure performance et efficacité du système. En outre, nous avons montré que, dans ce cas, le modèle multi-port (où vous pouvez effectuer plusieurs communications simultanées) est plus efficace que le modèle traditionnel d'un port (1-port).
En particulier, nous avons considéré trois problèmes d'ordonnancement dans des systèmes hétérogènes à large échelle sous le modèle multi-port borné : l'ordonnancement des tâches indépendantes qui a été étudié dans la première partie, la redistribution des fichiers qui a été étudié dans la deuxième partie, et le problème classique de la diffusion.
Dans ce modèle, chaque nœud est caractérisé par une bande passante entrante et une bande passante sortante et il peut participer à plusieurs communications simultanées pourvu que ses bandes passantes ne soient pas dépassées. Ce modèle correspond bien aux technologies de réseaux modernes. Il peut être utilisé en programmant au niveau TCP et il est même implémenté dans les bibliothèques modernes à passage de messages telles que MPICH2.
Nous avons montré, en utilisant les trois problèmes d'ordonnancement cités ci-dessus, que ce modèle permet la conception d'algorithmes polynomiaux et que même des algorithmes distribués assez simples peuvent atteindre la performance optimale, pourvu qu'on puisse imposer les politiques de partage de bande passante.
Nous avons prouvé aussi que l'implémentation d'algorithmes optimaux peut conduire à des performances largement sous-optimales si seul le mécanisme de la gestion de contention de TCP est utilisé. Le travail a fait l'objet d'une publication dans une conférence internationale IPDPS 2010 [8] et d'un rapport de recherche [6].

Première partie

Tâches indépendantes en multi-port degré borné

Chapitre 2

Tâches indépendantes en multi-port degré borné

2.1 Introduction

Suite à l'accroissement des besoins en puissance de calcul dans divers domaines, tel que le traitement de signal et l'imagerie, une tendance actuelle en matière de calcul distribué est de chercher à fédérer un ensemble de machines, réparties à l'échelle d'un continent, voire de la planète entière, afin d'en agréger les puissances de calcul. Ce nouveau type de plate-forme de calcul est de nature très hétérogène, que ce soit au niveau des ressources de calcul (processeurs) ou au niveau des capacités de communication (réseau). La prise en compte de cette hétérogénéité est donc un enjeu majeur pour l'utilisation efficace de ces plates-formes, aujourd'hui et demain, en particulier, dans le cadre de l'émergence de certaines plates-formes de calcul à large échelle telle que BOINC [5] et Folding@home [65]. Ces plates-formes sont caractérisées par la taille des données traitées, leur hétérogénéité et la variation de la performance de leurs ressources, d'où l'importance d'un ordonnancement efficace des tâches [10, 18, 22]. Ces différentes caractéristiques ont un effet sur l'ensemble des applications qui peuvent s'y exécuter. D'abord, le temps d'exécution d'une application doit être suffisamment long pour diminuer l'influence du temps du démarrage. De plus, ce cadre de calcul est adapté aux applications qui permettent le partitionnement des tâches en des morceaux pouvant être traités indépendamment, afin de minimiser l'influence de la variation des performances des ressources et de limiter l'effet des pannes. Enfin, en raison du temps de déploiement et de la difficulté de mise en œuvre sur de telles plates-formes, il est seulement rentable d'y déployer de grosses applications. Ainsi, la difficulté vient autant de l'hétérogénéité de la plate-forme et des contraintes du problème que de la métrique que l'on cherche à minimiser (makespan). Notamment, en s'intéressant non pas à la minimisation du makespan mais plutôt à la maximisation du débit en régime permanent, les problèmes deviennent beaucoup plus simples à traiter, et on peut construire des ordonnancements dont le temps d'exécution est asymptotiquement optimal. Cette approche a été initiée par Bertsimas et Gamarnik [16] et a été, par la suite, étendue à l'ordonnancement des tâches [11] et des communications collectives [14].

Dans ce travail, nous allons donc nous placer en régime permanent, qui peut se produire lorsque le nombre de tâches devient important et nous allons considérer des

tâches de même taille. La plate-forme sera modélisée par un ensemble de serveurs et un ensemble de clients. Ces clients reçoivent les tâches à exécuter des serveurs. Les ressources sont hétérogènes avec différentes vitesses de calcul et de communications. Notre but sera de construire un graphe biparti pondéré connectant les clients et les serveurs. Le réseau sous-jacent pour le type de plates-formes auquel nous nous intéressons est, généralement, Internet et supposer connue la topologie de ce réseau est absurde. Même dans le cas de certaines plates-formes de calcul distribuées de plus petite taille, telles que les grilles, les outils de découverte de la topologie de réseau [40, 90], sont beaucoup plus lents que l'évolution dynamique des performances des ressources. En outre, les coeurs des réseaux sont généralement sur-dimensionnés, donc les contentions se produisent, le plus souvent, au niveau des interfaces réseau.

Pour modéliser les contentions, nous avons recours au modèle multi-port borné. Le modèle multi-port est un modèle dans lequel le serveur peut effectuer plusieurs communications en même temps. Dans [52], Hong et Prasanna ont proposé ce modèle pour l'allocation des tâches indépendantes dans les systèmes du calcul hétérogènes. Ce modèle diffère du modèle traditionnel, un-port, très utilisé dans la littérature [107]. Dans ce modèle, le serveur ne peut communiquer qu'avec un seul client en même temps. Il est simple à implémenter mais, dans certains cas, le maître pourrait obtenir un meilleur débit agrégé en envoyant des données à plusieurs esclaves simultanément. L'utilisation de connexions multiples dans ce cas permet d'augmenter le débit réseau apparent [49, 24].

Le modèle de tâches que nous considérons, est proche des tâches divisibles. Une tâche est dite divisible lorsqu'elle peut être divisée en un ensemble de sous-tâches indépendantes qui peuvent s'exécuter arbitrairement sur les processeurs. De telles tâches permettent l'élaboration de stratégies d'utilisation des ressources disponibles en termes de puissance de calcul et de bandes passantes. Ce modèle d'ordonnancement des tâches divisibles, a été largement étudié dans la littérature [17, 13]. Il a été appliqué dans plusieurs domaines tel que le traitement d'image, le graphisme, la diffusion de flux vidéo et multimédia [13, 4], la recherche d'informations dans une base de données, ou encore le traitement de gros fichiers distribués.

Dans la suite, nous allons nous intéresser au problème d'ordonnancement des tâches indépendantes sur des plates-formes hétérogènes en multi-port borné. Nous allons commencer par une modélisation du problème. Mais avant de commencer cette modélisation, nous allons présenter un ensemble d'applications auxquelles s'appliquent notre problème. Nous présentons, ensuite, une description formelle de l'exécution qui s'appuie sur le modèle clients-serveur, en justifiant au fur et à mesure nos différents choix. Enfin, nous présentons notre algorithme d'approximation Seq qui se base principalement sur une petite augmentation de ressource. En particulier, nous prouvons que si d_j exprime le nombre maximal de connexions que peut ouvrir simultanément sur un serveur S_j, alors le débit obtenu en utilisant cet algorithme et $d_j + 1$ connexions est au moins celui de l'optimal avec d_j connexions. Nous présentons, aussi, un algorithme dual pour minimiser le nombre maximal de connexions qui doivent être ouvertes pour assurer un débit donné. Enfin, nous terminons par une étude expérimentale de l'algorithme Seq en le comparant à d'autres heuristiques.

2.2 Motivation

Dans cette partie, nous allons présenter un ensemble d'applications qui peuvent s'adapter au cadre de calcul que nous avons présenté dans l'introduction. En particulier, les applications qui permettent le partitionnement des tâches (charge à traiter) en des fractions qui peuvent être traitées de façon indépendante, afin que les solutions partielles puissent être consolidées afin de construire une solution complète pour le problème. Le principe de ce partitionnement peut varier d'une application à une autre. Selon ce principe, les applications peuvent se diviser en deux grandes catégories.

La première catégorie comprend les applications qui nécessitent un découpage physique des tâches en petites unités indépendantes. C'est le cas des applications qui impliquent l'analyse et le traitement de grandes quantités de données, telles que les données provenant d'un télescope radio ou à partir d'un accélérateur de particules, et qui ont un parallélisme inhérent. Dans section 2.2.2, nous allons présenter un exemple de ces applications, à savoir SETI@home.

La deuxième catégorie comprend les applications qui nécessitent l'exécution d'un grand nombre de simulations indépendantes sur le même jeu de données avec un paramètre variable. Ceci correspond, en général, à un échantillonnage dans un espace de grande dimension. Nous trouvons ici, par exemple, les modèles de systèmes physiques qui ont une composante aléatoire et chaotique et dont le résultat est probabiliste. Pour étudier les statistiques de ce résultat, il faut exécuter un grand nombre de simulations avec des conditions initiales aléatoires. Nous trouvons également certains projets de calcul distribué qui disposent de la même séquence de données mais sous différentes configurations et chaque configuration correspond à une tâche indépendante. C'est le cas de Folding@home, que nous allons présenter dans la section 2.2.3. Une tâche, dans ce projet de calcul distribué qui traite le repliement des protéines, correspond à une configuration particulière de protéine.

2.2.1 Concept

Les deux applications SETI@home et Folding@home que nous avons choisies de détailler dans la suite se basent sur le même principe. Ce principe consiste en le découpage d'un calcul extrêmement complexe et long en des milliers d'unités, dont le traitement est confié à une multitude d'ordinateurs. L'intérêt majeur du calcul volontaire, est qu'il puise uniquement ses ressources de la puissance de calcul inutilisée des ordinateurs. L'utilisateur aura toujours la main car le logiciel client installé sur sa machine travaille en priorité basse. C'est ainsi que, dans ses travaux de tous les jours ou même avec des applications très gourmandes sur les machines plus modernes, il ne se rendra pas compte du calcul effectué en arrière-plan, tout simplement parce que le traitement se mettra de façon dynamique au diapason des ressources restantes de l'ordinateur.

2.2.2 SETI@home

Dans cette section nous allons décrire SETI@home [6], un des projets de distribution de calcul les plus connus. C'est une expérience scientifique en radio-astronomie exploitant la puissance inutilisée de millions des ordinateurs connectés via Internet dans un projet de Recherche d'un Intelligence Extra-Terrestre (Search for Extra-Terrestrial Intelligence ou SETI).

Ce projet, conduit par un groupe de chercheurs, de l'université de Californie à Berkeley, est la toute première tentative d'utilisation du calcul distribué à large échelle pour effectuer une recherche exhaustive et précise des signaux radios de civilisations extraterrestres.

2.2.2.1 Mécanisme

SETI@home effectue la recherche de transmissions radio extraterrestres à partir des observations faites par le radiotélescope d'Arecibo à Puerto Rico. Les données sont prises "passivement" pendant que le télescope est utilisé pour d'autres projets scientifiques. Elles sont numérisées, emmagasinées et expédiées aux installations de SETI@home en Californie. Par la suite, elles sont divisées en petites unités de fréquence et temps qui sont analysées à l'aide d'un logiciel afin d'y déceler un signal, c'est-à-dire une variation qui se distingue du bruit cosmique et qui contient de l'information [2]. Le point crucial de SETI@home est que les millions d'unités produites sont envoyées, par le serveur, à des ordinateurs personnels qui utilisent le logiciel et qui, une fois l'analyse terminée, retournent les résultats au serveur. Les données échangées avec le serveur de données sont semblables à la transmission des éléments d'un formulaire classique sur le web.

2.2.2.2 Statistiques

SETI@home distribue actuellement le logiciel client pour 47 combinaisons différentes de processeurs et de système d'exploitation. Les utilisateurs peuvent télécharger le logiciel depuis le site web SETI@home (http ://setiathome.ssl.berkeley.edu). Par exemple, le 22 septembre 2010, plus d'un million d'utilisateurs et plus de deux millions et demi d'ordinateurs ont exécuté le programme SETI@home (voir Figure 2.1). Parmi eux, 519725 exécutaient activement le programme et ont retourné au moins un résultat dans les deux semaines précédentes. Ces volontaires ont une moyenne de 1060 téra-flops (où flops dénote les opérations à virgule flottante traitées par seconde) (voir Figures 2.2).
Près de 93 millions unités de travail ont été générées pour les utilisateurs et près de 2/3 de millions d'années de calculs processeur ont été réalisées.

2.2.3 Folding@Home

Dans cette section, nous allons présenter Folding@home, un projet de calcul distribué qui étudie le repliement des protéines, l'agrégation des protéines, et les maladies liées. Cette étude se fait dans des configurations diverses de température et de pression afin de mieux comprendre ce processus et de comprendre le lien entre

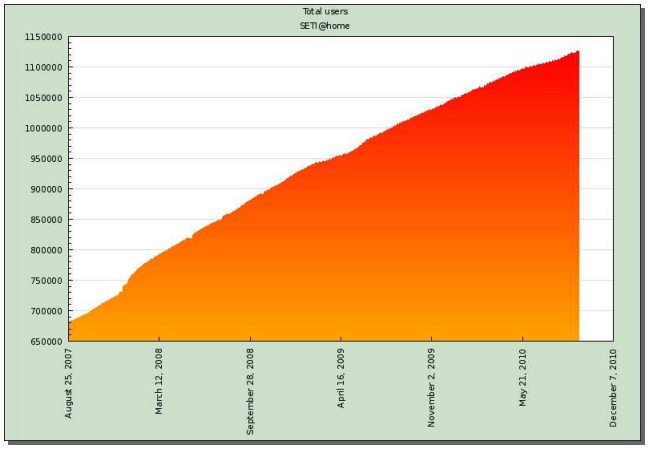

FIGURE 2.1: Nombre total des utilisateurs

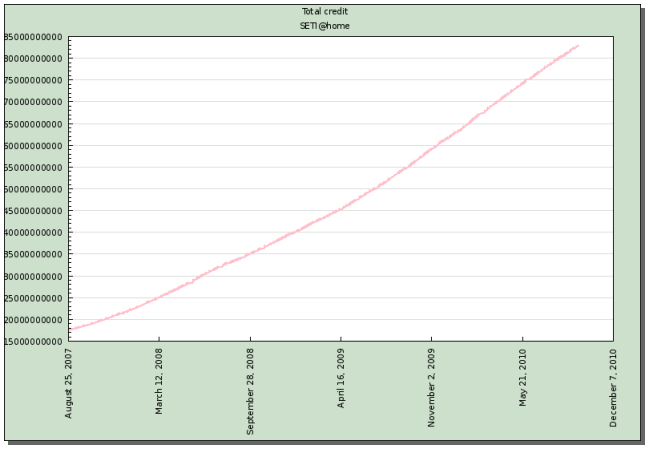

FIGURE 2.2: Nombre total des crédits

le repliement (forme tridimensionnel) des protéines humaines et certaines maladies (cancers, Parkinson, Alzeimer, etc).

Ce projet est organisé par une institution à but non-lucratif (le Pande Group de la Stanford University's Chemistry Department), ce qui est une garantie que les résultats des calculs seront accessibles aux chercheurs et autres scientifiques du monde entier et il est soutenu par Google, qui l'a intégré dans la Google Toolbar, sous Windows, un bouton permettant de lancer ou stopper le processus et informe sur son état d'avancement.

2.2.3.1 Principe

À l'Université de Stanford, il y a des serveurs qui contiennent les données initiales des molécules à étudier. Un client (utilisateur) doit installer un logiciel client sur sa machine (son ordinateur, sa PlayStation 3). Ce logiciel client va télécharger une unité de travail à partir du serveur et va calculer le repliement de la protéine qui lui a été confié. Chaque calcul occupe le processeur ou le GPU client quand il n'est pas utilisé, ce qui ne donne donc lieu à aucun ralentissement de la machine.

Chaque calcul dure de 4 à 200 heures environ, selon la configuration matérielle de l'ordinateur et la taille de l'unité traitée. Une fois son unité de travail terminée, le client la renvoie au serveur qui lui en confie une autre.

Une unité de travail définit un ensemble de paramètres pour la simulation de repliement de protéines. Elle correspond à un morceau de protéine à un moment précis. Par exemple, un client va calculer ce qui se passe entre la 20e et la 27e nanoseconde, pour la protéine X. En fonction de la protéine et de la propriété étudiée, il peut y avoir différentes tailles d'unités de travail. La page descriptif du projet "http ://fah-web.stanford.edu/cgi-bin/allprojects" donne des informations sur des tailles de protéines particulières et sur les dates limites allouées pour terminer l'étude de chacune.

2.2.3.2 Statistiques

Les utilisateurs peuvent télécharger le logiciel depuis le site web Folding@home (http ://folding.stanford.edu/).
Par exemple, au 22 septembre 2010 , 5600603 ordinateurs ont exécuté le programme Folding@home dont 288335 exécutaient activement le programme et ont retourné au moins un résultat dans les 50 jours précédents [65](voir Figure 2.3).

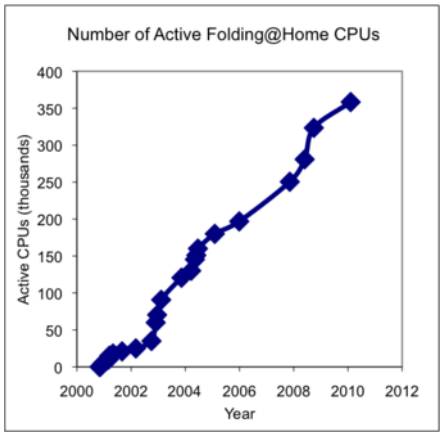

FIGURE 2.3: Nombre des CPUs actifs

2.3 Modélisation du problème

Dans cette section, nous allons commencer par une description de notre modèle se basant sur le modèle clients-serveurs. Puis, nous allons justifier le choix de ce modèle. Enfin, nous expliquerons pourquoi nous bornons le nombre de connexions aux serveurs.

2.3.1 Description du modèle

Pour modéliser notre problème, nous allons considérer un ensemble de serveurs $\{\mathcal{S}_j\}_j$ et un ensemble de de clients $\{\mathcal{C}_i\}_i$. Chaque serveur $\mathcal{S}_j$ a une capacité b_j qui présente le nombre de tâches par unité de temps que peut envoyer $\mathcal{S}_j$ aux clients. Chaque client $\mathcal{C}_i$ a une capacité w_i qui représente le minimum entre ce que $\mathcal{C}_i$ peut recevoir et ce qu'il peut traiter et est exprimé en nombre de tâches par seconde. On note donc par w_i^j le nombre de tâches qu'envoie un serveur $\mathcal{S}_j$ au client $\mathcal{C}_i$ par unité de temps. Un serveur peut se connecter simultanément à un nombre limité de clients. Ce nombre est borné par ce qu'on appelle le degré du serveur d_j. Chaque serveur maintient des connexions ouvertes avec les clients. Ce modèle peut se représenter par un graphe biparti pondéré entre les clients et les serveurs (voir exemple la figure 2.3.1). Notre but est de maximiser le nombre de tâches qui sont traitées par unité de temps par la plate-forme. Comme nous venons de le décrire dans la section précédente, notre modèle se base sur le paradigme maîtres-esclaves. Ce paradigme, consiste en l'exécution de tâches indépendantes sur un ensemble de processeurs, appelés esclaves, sous la houlette d'un processeur particulier, le maître [33].

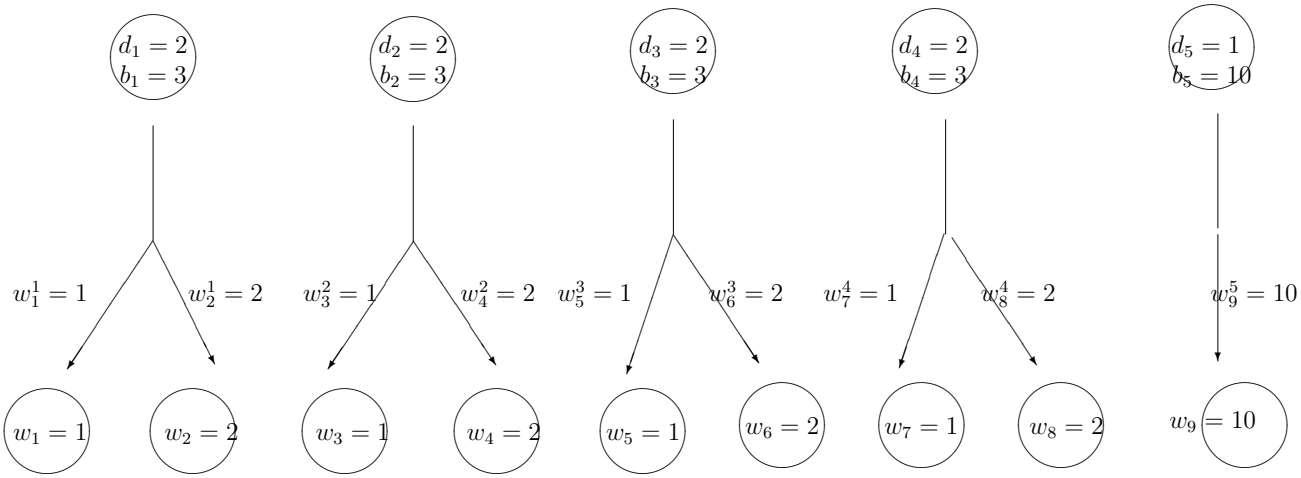

FIGURE 2.4: Solution Optimale avec 5 serveurs et 9 clients. Débit = 22

Notre modèle est formé d'une seule de couche de clients et de serveurs. Dans la sous section 2.3.1.1, nous discuterons brièvement ce choix.
En outre, ce modèle est caractérisé par le nombre limité de connexions que peut ouvrir un serveur avec les clients. Dans la section 2.3.2, nous allons expliquer notre choix de borner le nombre de connexions simultanées établies par un serveur avec les clients.

2.3.1.1 Modèle à une seule couche

Dans notre modélisation du système, nous nous appuyons sur le modèle maîtres-esclaves formé d'une seule couche (un seul niveau) de maîtres et d'esclaves.

Notre but est de concevoir un modèle permettant de maximiser le débit. Deux possibilités se présentent : soit un modèle à une seule couche formé d'un ensemble de serveurs connectés directement aux clients, soit un modèle à plusieurs couches où des nœuds intermédiaires jouant à la fois le rôle de client et celui de serveur secondaire s'insèrent entre le nœud initial (serveur principal) et la couche finale des clients.

Dans le cadre des tâches indépendantes, contrairement à celui des communications collectives par exemple, toutes les données associées à toutes les tâches sont différentes et doivent être envoyées depuis le serveur. Il est donc inutile, du point de vue de la consommation de la bande passante, d'ajouter une couche intermédiaire (même si, par contre, une telle couche pourrait permettre de minimiser le nombre de connexions nécessaires).

2.3.2 Degré borné

Plusieurs facteurs affectent la performance du système : la capacité des clients, la capacité des serveurs qui inclut leurs capacités de traitement et leurs capacités de gérer les différents clients auxquels ils sont connectés.

Selon le même principe et pour assurer une meilleure gestion du système, nous bornons le nombre de clients qui peuvent se connecter en même temps au serveur. Comme nous le verrons dans le chapitre 6 en particulier : la gestion dans le modèle multi-port de ressources hétérogènes nécessite l'utilisation de mécanismes de gestion de la qualité de service pour le partage de la bande passante, qui sont assez coûteux et qui limitent le nombre de connexions pouvant être gérées simultanément.

2.4 Travaux connexes

De nombreux travaux de recherche concernant l'ordonnancement de tâches indépendantes sur les plates-formes hétérogènes et distribuées ont été menés [102] notamment en "grid computing" .

Comme nous l'avons précisé au début de ce chapitre, les tâches que nous traitons sont indépendantes (c'est-à-dire sans relation de précédence). Donc, d'un point de vue système, lorsqu'un ensemble de tâches arrive, la stratégie commune est de les affecter selon la charge des ressources afin d'atteindre un débit maximal. Cette stratégie vise principalement l'équilibrage de charge entre les différentes ressources. Et elle est particulièrement utilisée dans les systèmes dont l'objectif primordial est de maximiser l'utilisation des ressources, plutôt que de minimiser le temps d'exécution des tâches individuelles [61]. Selon [37], il existe quatre approches pour effectuer cet équilibrage de charge : l'approche FIFO (premier arrivé premier servi), l'approche "balance-constrained" [28], l'approche "cost-constrained" et l'approche hybride [93]. L'approche FIFO permet de sélectionner, lorsqu'une tâche arrive, la ressource ayant la queue d'attente la plus courte [73]. L'approche "balance-constraint" cherche à équilibrer la charge dans tous les ressources en déplaçant périodiquement les tâches en attente d'une queue vers une autre [28]. Quant à l'approche "cost-constrained",

elle ne considère pas seulement l'équilibrage entre les ressources mais aussi elle tient compte du coût de communication des tâches. Avant chaque déplacement de tâche et au lieu de faire un échange périodique, le coût de celle ci est vérifiée. Si le coût de la communication suite à un déplacement d'une tâche est supérieur au temps d'exécution, la tâche ne sera pas déplacée ; sinon, elle le sera. Enfin, l'approche hybride permet de combiner l'ordonnancement dynamique et statique. L'ordonnancement statique est utilisé pour les tâches qui s'exécutent fréquemment, et l'ordonnancement dynamique pour les autres. Dans [93], Spring et al. présentent un exemple de ce scénario.

Le problème de minimisation du temps d'ordonnancement des tâches indépendantes sur des plates-formes hétérogènes est un problème qui a été déjà montré NP-complet [54]. Dans [60], les auteurs présentent un ensemble d'heuristiques traitant ce problème. Selon le moment où la décision d'ordonnancement est prise, l'ordonnancement est dit statique [20] ou dynamique [73]. Un ordonnancement est dit statique, ou encore "hors-ligne", lorsque le système dispose de toutes les tâches à allouer dès le début. Un ordonnancement est dit dynamique, ou encore "en-ligne", lorsque l'ordonnancement suit le changement du système ainsi que l'arrivée des tâches. Dans [19], Braun et al. présentent une étude comparative d'un ensemble de onze heuristiques, sélectionnées dans la littérature, pour traiter le problème d'ordonnancement statique des tâches indépendantes sur des plate-formes hétérogènes et distribuées.

Un résumé et une classification des heuristiques pour l'ordonnancement dynamique ont été détaillés dans [84]. Dans [66], Scott et al. développent deux approches dynamiques, une utilisant une politique distribuée et une autre utilisant une politique centralisée. Selon l'approche distribuée, à partir d'un noeud donné, ils considèrent la machine locale et les k voisins disposant des plus grandes bandes passantes. Par conséquent, l'allocation se fait à la meilleure machine parmi ces $k+1$ machines.

Dans [73], Maheswaran et al. traitent aussi le problème d'allocation dynamique d'une classe des tâches indépendantes dans les systèmes distribués hétérogènes mais ils considèrent deux modes : le mode immédiat et le mode différé. En mode immédiat, les tâches sont allouées aux machines dès qu'elles arrivent dans le système. En mode différé, les tâches sont regroupées en méta-tâches puis examinées pour l'allocation à certains instants, appelés moments d'allocation. Une méta-tâche peut inclure les nouvelles tâches qui arrivent au système ainsi que les anciennes qui ne sont pas encore exécutées.

Aucun de ces travaux, à notre connaissance, n'a traité le cas où les serveurs ont un degré borné. Sous un autre angle, notre problème est très proche du problème de "Bin Packing" (remplissage des sacs) avec articles divisibles et contraintes de cardinalité. Le but ici est de regrouper un ensemble donné d'articles dans un nombre minimum de sacs (bins). Les articles peuvent être divisés mais les sacs contiennent au plus k fractions. Ce problème est très proche du problème que nous traitons, avec toutefois deux différences principales. Selon notre modèle, le nombre de serveurs (qui correspondent aux sacs ou bins) est fixé à l'avance, et le but est de maximiser le débit des bandes passantes (qui correspond à la taille totale des paquets). Par contre, le

but de "Bin Packing" est de minimiser le nombre des boîtes utilisées pour contenir tous les articles. D'autre part, nous considérons des serveurs (bins) hétérogènes en capacité et en cardinalité.

Le problème "Bin Packing" avec des éléments divisibles et contraintes de cardinalité a été introduit dans le contexte de l'allocation de la mémoire sur des processeurs parallèles par Chung et al. [29], qui considèrent le cas particulier où $k = 2$. Ils prouvent que, même dans ce cas, le problème est NP-Complet et proposent un algorithme d'approximation de facteur 3/2. Dans [38], Epstein et Stee montrent que le problème de Bin Packing avec des éléments divisibles et des contraintes de cardinalité est NP-Complet pour tout valeur fixée de k, et que l'algorithme simple NEXT-FIT réalise un facteur d'approximation de $2 - \frac{1}{k}$. Ils proposent également un PTAS et un PTAS dual [39] pour le cas où k constant.

D'autres problèmes proches, ont été introduits par Shachnai et al. dans [89]. Ils ont traité le cas où la taille d'un article augmente lorsqu'il est fractionné, ainsi que le cas où une borne globale sur le nombre des fragments est imposée. Les auteurs ont prouvé qu'il n'existe pas de PTAS pour ces deux problèmes, mais proposent un PTAS dual et un PTAS asymptotique. Un autre problème, dans le contexte de l'ordonnancement de multi-processeurs, est très proche du notre. Il s'agit de l'ordonnancement avec des contraintes d'attribution et de parallélisme [88], où le but est d'ordonnancer un certain nombre de tâches, et où chaque tâche a une borne sur le nombre de machines sur lesquelles elle peut s'exécuter simultanément et une autre borne sur le nombre de machine qui peuvent participer à son exécution. Ce problème peut être aussi vu comme un problème de bin packing avec des articles divisibles mais, dans ce cas, avec une borne k_i sur le nombre de fois un article peut être fractionné. Dans [88], les auteurs proposent un algorithme d'approximation de facteur d'approximation $\max_i(1 + 1/k_i)$.

2.5 Algorithme Seq : Analyse de complexité et étude du problème

Avant de passer à l'analyse de complexité, rappelons la modélisation du problème.

Étant donné un ensemble de serveurs $\{\mathcal{S}_j\}_j$ et de clients $\{\mathcal{C}_i\}_i$.

On note par b_j la capacité d'un serveur $\mathcal{S}_j$ et d_j le nombre maximal de connexions qui peut maintenir simultanément. Et on note par w_i la capacité d'un client $\mathcal{C}_i$ (toutes les capacités sont exprimées en nombre de tâches par unité de temps). On note donc par w_i^j le nombre de tâches envoyées à partir du serveur $\mathcal{S}_j$ au client $\mathcal{C}_j$ par unité de temps. Chaque serveur se connecte avec un ensemble de clients en respectant les conditions suivantes :

$$\forall j, \quad \sum_i w_i^j \leqslant b_j \quad \text{contrainte de capacité du } \mathcal{S}_j \tag{2.1}$$

$$\forall j, \quad \text{Card}\{i, \ w_i^j > 0\} \leqslant d_j \quad \text{contrainte de degré du } \mathcal{S}_j \tag{2.2}$$

$$\forall i, \quad \sum_j w_i^j \leqslant w_i \quad \text{contrainte de capacité du } \mathcal{C}_i \tag{2.3}$$

Notre but est de maximiser le nombre de tâches qui peuvent être traitées en une unité de temps par la plate-forme. Ce qui correspond au problème MDDB suivant :
Maximiser-Débit-Degré-Borné (MDDB) :

$$\text{Maximiser} \sum_j \sum_i w_i^j \text{ sous les contraintes (2.1),(2.2) et (2.3).}$$

Une solution valide sera donc une affectation des w_i^j. Ce qui peut être représenté aussi par un graphe biparti pondéré entre les clients et les serveurs (voir Figure2.4) ayant pour poids sur les arêtes les w_i^j.

A ce problème peut correspondre le problème décisionnel DDB-Dec suivant :
Débit-Degré-Borné-Dec (DDB-DEC) :

- **Instance** : Un ensemble de m serveurs $\mathcal{S}_1, \ldots, \mathcal{S}_m$ de capacité b_j et degré d_j, un ensemble de n clients $\mathcal{C}_1, \ldots, \mathcal{C}_n$ de capacité w_i et une borne K.
- **Solution** : un graphe biparti pondéré entre les clients et les serveurs, le poids de l'arrête entre $server_j$ et $\mathcal{C}_i$ est donnée par w_i^j, où les w_i^j satisfont les contraintes (2.1),(2.2) et (2.3) et tel que $\sum_j \sum_i w_i^j \geqslant K$.

2.5.1 Complexité

Dans cette partie, nous allons montrer que le problème **DDB-Dec** est NP-Complet. Pour le prouver, nous allons réduire ce problème à un autre problème difficile, à savoir **3-Partition** [43] et nous allons montrer qu'une instance particulière de **DDB-Dec** peut se réduire à une instance générale de **3-Partition**.
Le problème de 3-Partition :

- **Instance** :
  un ensemble A de $3m$ éléments $\{ a_1 \ldots a_{3m} \}$ tel que :
  - $\sum a_i = mB$.
  - $\forall i, \frac{B}{4} < a_i < \frac{B}{2}$.

- **Solution** :
  Peut-on réaliser une partition de A en m classes de même poids ?
  Si la réponse est "oui", chaque classe contient exactement 3 éléments de A et est de taille B.

Soit une instance de **3-Partition** consistant en $3m$ éléments a_i tels que $\sum a_i = mB$ et $\forall i,\ \frac{B}{4} < a_i < \frac{B}{2}$ et soit $\forall j,\quad d_j = 3,\quad b_j = B$, $n = 3m$ (avec n nombre de clients et m celui de serveurs), $\forall i,\ w_i = a_i$ et $K = mB$. Comme la somme des degrés maximaux sortants des serveurs est $3m$ et que tous les clients doivent être connectés pour atteindre le débit mB, alors chaque serveur doit se connecter exactement à trois clients et chaque client doit être connecté à un unique serveur. En outre, la capacité totale des serveurs est mB, donc chaque serveur doit être connecté exactement à trois clients de capacité exactement B. Ceci achève la preuve de NP-complétude.

Nous avons prouvé donc que le problème **MDDB** est NP-Complet. L'idée sera donc d'essayer de relâcher une des trois contraintes. Dans la suite nous allons présenter un algorithme polynomial SEQ s'appuyant sur une relaxation de la contrainte de degré (2.2) ce qui correspond, dans notre cas, à une augmentation de ressources.

2.5.2 Algorithme SEQ

Dans cette partie nous allons commencer par une description de l'algorithme SEQ, qui s'appuie sur une augmentation de ressource. Ensuite nous prouvons que cette augmentation permet d'atteindre au minimum le débit optimal avec les degrés d_j. En effet, nous prouvons qu'une augmentation d'une unité (+1) au nombre maximal de connexions possibles (d_j) est suffisante pour calculer en temps polynomial une solution ayant pour débit au moins l'optimal avec d_j connexions.

2.5.2.1 Algorithme

SEQ permet de résoudre le problème en relâchant la contrainte sur le degré maximal des serveurs. Cette relaxation consiste en une petite augmentation de ressource. En particulier, il s'agit d'une augmentation d'une unité du nombre maximal de connexions simultanés qui peut maintenir chaque serveur. Une solution de SEQ permet donc à au plus $d_j + 1$ clients, de se connecter au serveur $\mathcal{S}_j$.

Dans la suite, nous allons considérer $\mathcal{C} = \{\mathcal{C}_i\}$ une liste triée des clients par capacités croissantes et $\mathcal{C}(l,k) = \sum_{i=l}^{k} w_i$ la somme des capacités des clients entre $\mathcal{C}_l$ et $\mathcal{C}_k$.

SEQ maintient une liste ordonnée des clients. Le choix des serveurs est arbitraire. En effet, à chaque fois qu'un serveur $\mathcal{S}_j$ est choisi, l'algorithme parcourt la liste des clients pour trouver un ensemble consécutif de clients convenable à $\mathcal{S}_j$. Le fait de considérer des clients consécutifs dans la liste permet à la fois de décroître la complexité de l'algorithme et de prouver sa correction.

Le choix des clients et leurs allocations suit le principe suivant :

Tout d'abord, l'algorithme essaie de trouver un intervalle $[l, l+d_j]$ tel que $\mathcal{C}(l, l+d_j) \geqslant b_j$ et $\mathcal{C}(l, l+d_j-1) < b_j$. Les d_j premiers clients sont alloués entièrement à $\mathcal{S}_j$ et le dernier (ici le $\mathcal{C}_{l+d_j}$) est alloué juste en partie (c.à.d qu'il peut être fractionné). Une fois que ce choix est réalisé, le dernier client (celui qui a été fractionné) est réinséré dans la liste des clients avec une nouvelle capacité égale à $\mathcal{C}(l, l+d_j) - b_j$ et la liste est triée de nouveau. Le même processus est ensuite repris avec le reste des serveurs.

Il reste à régler le cas où il n'existe pas un tel intervalle de $d_j + 1$ éléments.
Dans ce cas, on distingue deux situations différentes. La première situation, correspond au cas où la somme des capacités des $d_j + 1$ plus grands clients, c.à.d les derniers $d_j + 1$ clients dans la liste, n'est pas suffisante pour atteindre b_j. Dans ce cas nous affectons ces $d_j + 1$ clients à $\mathcal{S}_j$.
La deuxième situation, correspond au cas où les sous d_j premiers clients consécutifs ont des capacités dont la somme dépasse b_j. Dans ce cas nous prenons les k plus petits clients tel que $\mathcal{C}(1,k) \geqslant b_j$ et $\mathcal{C}(1,k-1) < b_j$.

Dans la suite, nous présentons une description plus formelle de l'algorithme.

Algorithm 1 Algorithme SEQ

Soit $\mathcal{S} = \{\mathcal{S}_j\}_{j=1}^{m}$ et $\mathcal{C} = \text{sort}(\{\mathcal{C}_i\}_{i=1}^{n})$;
Soit $\mathcal{A} = \{\mathcal{A}_j = \{\varnothing\}\}_{j=1}^{m}$ et $j = 1$;

for $j = 1$ à m **do**

if $\exists l$ tel que $\mathcal{C}(l, l + d_j - 1) < b_j$ et $\mathcal{C}(l, l + d_j) \geqslant b_j$ **then**
Fractionner $\mathcal{C}_{l+d_j}$ en $\mathcal{C}'_{l+d_j}$ et $\mathcal{C}''_{l+d_j}$ avec $w_{l+d_j} = w'_{l+d_j} + w''_{l+d_j}$ et $w''_{l+d_j} = b_j - \mathcal{C}(l, l + d_j - 1)$
Soit $\mathcal{A}_j = \{\mathcal{C}_l, \mathcal{C}_{l+1}, \ldots, \mathcal{C}_{l+d_j-1}, \mathcal{C}''_{l+d_j}\}$
Enlever $\mathcal{C}_l, \mathcal{C}_{l+1}, \ldots, \mathcal{C}_{l+d_j}$ de $\mathcal{C}$
Insérer $\mathcal{C}'_{l+d_j}$ dans $\mathcal{C}$
end if

if $\mathcal{C}(1, d_j) \geqslant b_j$ **then**
Rechercher le plus petit l tel que $\mathcal{C}(1, l) \geqslant b_j$
Fractionner $\mathcal{C}_l$ dans $\mathcal{C}'_l$ et $\mathcal{C}''_l$ avec $w_l = w'_l + w''_l$ et $w''_l = b_j - \mathcal{C}(1, l - 1)$
Soit $\mathcal{A}_j = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_{l-1}, \mathcal{C}''_l\}$
Enlever $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_l$ de $\mathcal{C}$
Insérer $\mathcal{C}'_l$ dans $\mathcal{C}$
end if

if $\mathcal{C}(n - d_j, n) < b_j$ **then**
Soit $\mathcal{A}_j = \{\mathcal{C}_{n-d_j}, \mathcal{C}_{n-d_j+1}, \ldots, \mathcal{C}_n\}$
Enlever $\mathcal{C}_{n-d_j}, \mathcal{C}_{n-d_j+1}, \ldots, \mathcal{C}_n$ de $\mathcal{C}$
end if

end for

RETOURNER $\mathcal{A} = \{\mathcal{A}_j\}_{j=1}^{m}$

2.5.2.2 Preuve

Dans cette section nous allons prouver que l'application de l'algorithme SEQ et le fait d'augmenter les degrés des serveurs d'une unité, permet d'atteindre au moins le débit optimal d'une solution valide de MDDB. En particulier nous allons prouver le théorème suivant :

Théorème 2.1 *Soit $\mathcal{A}$ une solution valide ayant I comme instance, et* SEQ(I) *la solution donnée par l'algorithme* SEQ. *Le débit obtenu par* SEQ(I) *est au moins égal à celui de $\mathcal{A}$.*

Pour prouver ce théorème, nous allons supposer que la liste de clients conservera n comme taille le long de la preuve, en affectant une capacité 0 aux clients déjà alloués par l'algorithme.

Nous allons définir une relation d'ordre $\preceq$ pour comparer deux listes de clients données, selon la définition suivante :

Définition 2.2 *Soient $\mathcal{C}$ et $\mathcal{R}$ deux listes de clients de même longueur n, triés selon des capacités croissantes. Nous disons que $\mathcal{C}$ est* plus facile *que $\mathcal{R}$ (noté par $\mathcal{C} \preceq \mathcal{R}$), ssi*

$$\forall k \leqslant n, \quad \mathcal{C}(1,k) \leqslant \mathcal{R}(1,k)$$

Comme nous l'avons déjà mentionné dans la section précédente, SEQ maintient en permanence une liste des clients non encore alloués aux serveurs. Soit la liste $\mathcal{C}$ des clients non encore alloués maintenue par SEQ et $\mathcal{R}$ celle maintenue par une solution valide (c'est-à-dire une solution au MDDB dans laquelle le degré de $\mathcal{S}_j$ est au plus d_j).

Soit $\mathcal{S}$ un serveur de degré d et capacité b. Considérons ce serveur et appliquons SEQ pour retrouver l'ensemble des clients à connecter avec $\mathcal{S}$.
Soit $\mathcal{C}'$ la liste des clients restant obtenue à partir de $\mathcal{C}$ après l'application de SEQ et soit $\mathcal{R}'$ la liste des clients restant obtenue à partir de $\mathcal{R}$ suite à une allocation valide du serveur $\mathcal{S}$. Cette opération préserve, selon le lemme 2.3 suivant, la relation d'ordre $\preceq$.

Lemme 2.3 *Si* $\begin{array}{ccc} \mathcal{C} & \xrightarrow{\text{SEQ}(d+1,b)} & \mathcal{C}' \\ \preceq & & \\ \mathcal{R} & \xrightarrow{valid(d,b)} & \mathcal{R}' \end{array}$ *alors* $\mathcal{C}' \preceq \mathcal{R}'$.

Preuve. Selon la Définition 2.2, pour montrer que $\mathcal{C}' \preceq \mathcal{R}'$, il suffit de montrer que $\forall k \leqslant n, \quad \mathcal{C}'(1,k) \leqslant \mathcal{R}'(1,k)$. Pour comparer $\mathcal{C}'(1,k)$ et $\mathcal{R}'(1,k)$ nous avons besoin de les évaluer ou de les borner. Nous allons commencer par $\mathcal{R}'(1,k)$ et nous allons essayer de le borner inférieurement de deux manières différentes.

Comme la liste $\mathcal{R}'$ est obtenue à partir de $\mathcal{R}$ par une allocation valide, il existe un sous ensemble $\mathcal{C}$ inclus dans $[1 \dots n]$ de clients choisis et de valeurs affectées v_i tel que $Card(C) \leqslant d, \forall i \; v_i \leqslant \mathcal{R}_i$ (où $\mathcal{R}_i$ représente la capacité du i^{me} client dans $\mathcal{R}$) et $\sum v_i \leqslant b$.
Soit $i \leqslant n$ un index donné. Si $i \notin C$, alors ce i^{me} client ne fait pas partie des clients qui ont été alloués au serveur $\mathcal{S}$. Du coup, il garde la même capacité dans $\mathcal{R}$, et le seul changement qu'il puisse subir est le changement de sa position dans la liste. Donc il existe une permutation σ tel que $\mathcal{R}'_{\sigma(i)} = \mathcal{R}_i$. Si $i \in C$, alors ce i^{me} client a bien été impliqué dans l'allocation. Sa nouvelle capacité dans $\mathcal{R}'$ a diminué par rapport à celle dans $\mathcal{R}$, et $\mathcal{R}'_{\sigma(i)} = \mathcal{R}_i - v_i$. Ainsi $\mathcal{R}$'(1,k) peut s'exprimer de la manière suivante :

$$\begin{aligned} \mathcal{R}'(1,k) &= \sum_{i:i\notin C \wedge \sigma(i) \leqslant k} \mathcal{R}_i + \sum_{i:i\in C \wedge \sigma(i) \leqslant k} \mathcal{R}_i - v_i \\ &= \sum_{i:\sigma(i) \leqslant k} \mathcal{R}_i - \sum_{i:i\in C \wedge \sigma(i) \leqslant k} v_i \end{aligned}$$

D'autre part, nous avons, pour $k > d$, au moins $k - d$ clients d'indice i tel que $i \notin C$ et tel que le nouveau indice $\sigma(i)$ dans $\mathcal{R}$ est inférieur ou égal à $k \quad \sigma(i) \leqslant k$.

Comme $\mathcal{R}(1, k-d)$ représente la somme des capacités des $(k-d)$ plus petits clients (en terme de capacité disponible) alors $\sum_{i:i\notin c\wedge\sigma(i)\leqslant k}\mathcal{R}_i \geqslant \mathcal{R}(1, k-d)$. De plus, comme $\mathcal{R}_i - v_i \geqslant 0$, nous obtenons la borne suivante :

$$\mathcal{R}'(1,k) \geqslant \mathcal{R}(1,k-d), \quad \forall k > d. \tag{2.4}$$

De même, puisqu'il existe k indices i tels que $\sigma(i) \leqslant k$, alors $\sum_{i:\sigma(i)\leqslant k}\mathcal{R}_i \geqslant \mathcal{R}(1,k)$ et $\sum_{i\in C} v_i \leqslant b$, Nous obtenons donc la borne suivante (la deuxième) :

$$\mathcal{R}'(1,k) \geqslant \mathcal{R}(1,k) - b. \tag{2.5}$$

Une fois $\mathcal{R}'(1,k)$ est borné, nous allons maintenant considérer le deuxième membre de la comparaison, à savoir le $\mathcal{C}'(1,k)$, et nous allons l'évaluer.

Face à l'ajout d'un nouveau serveur, l'algorithme SEQ peut se trouver dans trois situations différentes. $\mathcal{C}'(1,k)$ doit donc être évalué dans chacun de ces trois possibles cas.

Cas 1 : $\exists l$ tel que $\mathcal{C}(l, l+d-1) < b$ et $\mathcal{C}(l, l+d) \geqslant b$. C'est le cas le plus général. Dans ce cas (voir les lignes de 4 à 8 dans l'Algorithme 1), l'algorithme alloue en entier les clients $\mathcal{C}_l, \mathcal{C}_{l+1}, \ldots, \mathcal{C}_{l+d-1}$ au serveur $\mathcal{S}$, et en partie seulement le client $\mathcal{C}_{l+d}$ dont la capacité restante est w'_{l+d}. Les d clients alloués entièrement vont être insérés au début de la liste $\mathcal{C}'$ avec la capacité zéro. Le client $\mathcal{C}'_{l+d}$ va être inséré dans $\mathcal{C}'$ avec une nouvelle capacité plus petite w'_{l+d}, donc dans une position avant $\mathcal{C}_{l+d+1}$. Disons, qu'il va être inséré entre $\mathcal{C}_p$ et $\mathcal{C}_{p+1}$. La nouvelle liste $\mathcal{C}'$ sera :

$$\begin{aligned}&\{\mathcal{C}'_1 = 0, \ldots, \mathcal{C}'_d = 0, \mathcal{C}_1, \ldots, \mathcal{C}_p, \mathcal{C}'_{l+d}, \mathcal{C}_{p+1}, \ldots\\ &\ldots, \mathcal{C}_{l-1}, \mathcal{C}_{l+d+1}, \ldots, \mathcal{C}_n\}.\end{aligned}$$

Considérons maintenant différents cas selon la position de k.

$k \leqslant d$ **:** Dans ce cas, $\mathcal{C}'(1,k)$ est la somme de k clients réinsérés et complètement alloués, donc $\mathcal{C}'(1,k) = 0$.

$d < k \leqslant p+d$ **:** Dans ce cas, $\mathcal{C}'(1,k)$ est la somme de $(k-d)$ premières capacités dans $\mathcal{C}$, puisqu'ils ont été poussés de d positions (à cause de l'insertion de d clients au début de la liste), et donc $\mathcal{C}'(1,k) = \mathcal{C}(1,k-d)$.

$p+d < k \leqslant l+d$ **:** Dans ce cas, la somme est la même que celle dans l'intervalle précédent sauf le dernier élément dans la somme, qui est remplacé par la taille du client fractionné qui a été réinséré, et $\mathcal{C}'(1,k) = \mathcal{C}(1,k-d-1) + w'_{l+d}$.

$l+d < k$ **:** Dans ce cas, la somme est égale à celle dans la liste $\mathcal{C}$, diminuée de la capacité totale allouée à $\mathcal{S}$ c'est-à-dire b, et nous avons $\mathcal{C}'(1,k) = \mathcal{C}(1,k) - b$.

Maintenant, en utilisant les équations 2.4 et 2.5, et puisque $\mathcal{C} \preceq \mathcal{R}$, nous avons :

$$\begin{aligned}
\mathcal{C}'(1,k) &= 0 \leqslant \mathcal{R}'(1,k) \quad \text{pour} \quad k \leqslant d \\
\mathcal{C}'(1,k) &= \mathcal{C}(1,k-d) \leqslant \mathcal{R}(1,k-d) \\
&\leqslant \mathcal{R}'(1,k) \quad \text{pour} \quad d < k \leqslant p+d \\
\mathcal{C}'(1,k) &= \mathcal{C}(1,k-d-1) + w'_{l+d} \\
&\leqslant \mathcal{C}(1,k-d) \leqslant \mathcal{R}(1,k-d) \\
&\leqslant \mathcal{R}'(1,k) \quad \text{pour} \quad p+d < k \leqslant l+d \\
\mathcal{C}'(1,k) &= \mathcal{C}(1,k) - b \leqslant \mathcal{R}(1,k) - b \\
&\leqslant \mathcal{R}'(1,k) \quad \text{pour} \quad l+d < k.
\end{aligned}$$

Cas 2 : $A(1,d) \geqslant b$**.** C'est le cas où la somme des d plus petites capacités dépasse b (voir les lignes de 9 à 14 de l'Algorithme 1). Comme SEQ utilise les premiers $l \leqslant d$ clients, il n'y aura pas de ré-ordonnancement de la liste.
La nouvelle liste $\mathcal{C}'$ aura donc la forme suivante $\{\mathcal{C}'_1, \ldots, \mathcal{C}'_{l-1}, \mathcal{C}'_l, \mathcal{C}_{l+1}, \ldots, \mathcal{C}_n\}$, où $\mathcal{C}'_i$ a capacité zéro pour $i < l$. Or, la capacité totale allouée au serveur est égale à b. Donc, nous avons $\mathcal{C}'(1,k) = 0$ quand $k \leqslant l-1$, et $\mathcal{C}'(1,k) = \mathcal{C}(1,k) - b$ for $k > l-1$.

En considérant le fait que $\mathcal{C} \preceq \mathcal{R}$ et en tenant compte de l'équation 2.5, nous avons $\mathcal{C}'(1,k) \leqslant \mathcal{R}'(1,k)$.

Cas 3 : $A(n-d,n) < b$**.** C'est le cas où la somme des $(d+1)$ plus grosses capacités est strictement inférieure à b (voir les lignes de 15 à 18 de l'Algorithme 1). Dans ce cas, SEQ alloue complètement les $d+1$ derniers clients au $\mathcal{S}$, et donc tous les clients réinsérés dans $\mathcal{C}'_i$ auront une capacité zéro et seront insérés au début de la liste.
La nouvelle liste $\mathcal{C}'$ aura donc la forme suivante $\{\mathcal{C}'_{n-d}, \ldots, \mathcal{C}'_n, \mathcal{C}_1, \ldots, \mathcal{C}_{n-d-1}\}$. Donc, $\mathcal{C}'(1,k) = 0$ quand $k \leqslant d+1$, et $\mathcal{C}'(1,k) = \mathcal{C}(1, k-(d+1))$ pour $k > d+1$.

De même, en tenant compte de l'équation 2.4 et en considérant le fait que $\mathcal{C} \preceq \mathcal{R}$, nous avons $\mathcal{C}'(1,k) \leqslant \mathcal{R}'(1,k)$. La preuve du lemme 2.3 s'achève ici.

Preuve. (du Théorème 2.1)
Soit $\mathcal{LC}_0$ et $\mathcal{LR}_0$ deux listes initiales de clients tel que $\mathcal{LC}_0 = \mathcal{LR}_0 = \mathcal{L}$. Soit $\mathcal{LC}_j$ (resp. $\mathcal{LR}_j$) la liste des clients restants après les j premières étapes de l'algorithme SEQ (resp. les clients non totalement alloués au serveurs $\mathcal{S}_1, \ldots, \mathcal{S}_j$ par l'allocation valide $\mathcal{A}$).

Il suffit d'appliquer successivement du lemme 2.3, pour prouver par induction que $\mathcal{LC}_m \preceq \mathcal{LR}_m$ c'est-à-dire $\forall k \leqslant n$, $\mathcal{LC}_m(1,k) \leqslant \mathcal{LR}_m(1,k)$. En particulier, nous avons $\mathcal{LC}_m(1,n) \leqslant \mathcal{LR}_m(1,n)$. Or $\mathcal{LC}_m(1,n)$ et $\mathcal{LR}_m(1,n)$ représentent respectivement la capacité totale non encore utilisée dans la solution calculée par SEQ et dans la solution $\mathcal{A}$.

Par conséquent, le débit obtenu par l'algorithme SEQ est plus grand que celui obtenu par une solution valide $\mathcal{A}$, ce qui termine la preuve du Théorème 2.1.

2.6 Résultats d'approximation

Dans cette partie , nous allons prouver deux résultats d'approximation. Le premier concerne notre problème principal MDDB. Nous allons montrer qu'avec une petite modification, SEQ peut devenir un algorithme d'approximation valide pour MDDB. Le deuxième, concerne le problème dual MDGT (Minimisation de Degré d'un Débit Donné). Nous allons montrer que SEQ fournit, aussi, une bonne approximation à ce problème dual.

Nous commençons par l'approximation de MDDB, en s'appuyant sur l'algorithme SEQ, d'un facteur de $\rho = \frac{d_{\min}}{d_{\min}+1}$ (où $d_{\min}$ correspond au plus petit degré des serveurs).

Le principe est le suivant, nous retirons, à la fin de l'algorithme SEQ, un client de chaque serveur dont le degré sortant a été dépassé. Le fait de retirer le plus petit client (en terme de capacité) ne fait pas décroître la moyenne de la bande passante sortante. Donc, si nous notons par w^j la bande passante sortante du serveur $\mathcal{S}_j$ à la fin de l'algorithme SEQ, et par w'^j sa bande passante sortante après la modification (c.a.d après avoir déconnecté le plus petit client), nous aurons $\frac{w'^j}{d_j} \geqslant \frac{w^j}{d_j+1}$. Par conséquent, $w'^j \geqslant \frac{d_j}{d_j+1} w^j \geqslant \rho w^j$. D'autre part, le débit total T est égal à la somme de tous les w^j. Donc T est plus grand que le débit optimal T^*. Il en résulte que $T' \geqslant \rho T^*$.

Ensuite, nous passons au problème dual MDGT pour lequel nous allons aussi fournir une approximation. En effet, si nous avons une borne $T \leqslant \min(\sum_j b_j, \sum_i w_i)$ sur le débit, une simple recherche dichotomique permet de retrouver la valeur minimale α_{SEQ} de α telle que le débit de $\text{SEQ}(I(\alpha))$ est au moins B, dans l'instance modifiée $I(\alpha)$ où chaque serveur $\mathcal{S}_j$ a un degré $d_j + \alpha$.

En revanche, d'après le Théorème 2.1, si le débit d'une solution $\mathcal{A}$ de l'instance $I(\alpha - 1)$ est B, alors $\text{SEQ}(I(\alpha - 1))$ fournit une solution valide à l'instance $I(\alpha)$ de débit au moins B. Donc $\alpha_{\text{SEQ}} \leqslant \alpha^* + 1$, où α^* est la valeur optimale du problème MDGT de l'instance I. Comme MDGT est NP-complet, alors ce résultat est le meilleur résultat d'approximation possible.

2.7 Résultats expérimentaux

2.7.1 Heuristiques de comparaison

Pour assurer l'évaluation pratique de SEQ, nous présentons trois algorithmes (heuristiques) gloutons (LCLS, LCBC, OBC).

LCLS (Largest Client Largest Server) : À chaque étape, le client avec la plus grande capacité w_i est associé avec le serveur avec la plus grande capacité disponible $b'_j = b_j - \sum_i w_i^j$. Le client est divisé, si c'est nécessaire, en plusieurs morceaux. Dans ce cas, la capacité restante $w'_i = w_i - b'_j$ est insérée dans la liste ordonnée.

LCBC (Largest Client Best Connection) : Dans cette heuristique, on considère aussi le client le plus grand, en terme de capacité, mais les serveurs sont ordonnés selon leur *capacité par connexion* restante, qui est définie par le rapport entre la capacité restante b'_j et le degré restant d'_j. Le serveur avec la plus grande capacité

par connexion est sélectionné. Ici aussi, nous pouvons fractionner le client, si c'est nécessaire.

OBC (Online Best Connection) : Cette heuristique est une version en ligne de la précédente. Tous les serveurs sont supposés être connus au début de l'exécution, mais les clients arrivent arbitrairement. On choisit un serveur avec la capacité (par connexion) restante la plus proche de la capacité du client. En particulier, on choisit le serveur avec la grande capacité b'_j/d'_j tel que $b'_j/d'_j \leqslant w_i$.

2.7.2 Simulations

L'étude expérimentale se divise en deux parties. La première partie concerne la résolution de l'algorithme Seq. La deuxième concerne la résolution du problème dual. En particulier, dans la première partie, nous avons mesuré et comparé le débit des solutions proposées par chaque algorithme.

Dans la deuxième partie, nous avons calculé, pour chaque algorithme, la valeur minimale α^*, que nous devons ajouter aux degrés de chaque serveur pour que l'algorithme $\mathcal{A}$ atteigne la borne maximale $B = \min(\sum_j b_j, \sum_i w_i)$.

– Dans cette partie, nous avons lancé l'exécution des différents algorithmes et nous avons calculé le débit de chaque algorithme. Toutes les valeurs sont normalisées par rapport à la borne supérieure $\min(\sum_j b_j, \sum_i w_i)$.
Les capacités des serveurs et des clients ont été simulées en utilisant une distribution réaliste basée sur GIMPS [45]. De même, la comparaison avec les heuristiques gloutonnes a été réalisée en utilisant des distributions de degré réelles. Le nombre des clients, est de dix fois ($\times 10$) le nombre des serveurs.
Dans la Figure 2.5, nous trouvons les résultats moyens de 250 instances, où le nombre des serveurs varie de 20 à 140.

  Nous remarquons que pour l'ensemble des instances, Seq obtient les meilleurs résultats et qu'il atteint toujours la borne supérieure. Nous remarquons aussi que la performance de l'algorithme LCBC est, à peu près 4% inférieure à Seq, et LCLS 10-12% inférieure à Seq. Enfin, nous remarquons que la performance de OBC, n'est pas aussi mauvaise en moyenne, mais nous pouvons (via d'autres simulations) noter que la variabilité est plus importante que pour les trois autres algorithmes.

  En outre, nous constatons que la performance des algorithmes glouton est un peu sensible à la distribution des capacités des clients, leur performance est pire lorsque les clients sont assez hétérogènes. En revanche, la garantie théorique de Seq rend sa performance assez stable sous toutes ces conditions.
– Dans cette partie, nous avons calculé, la valeur minimale α^* dont a besoin chaque algorithme pour atteindre la borne $B = \min(\sum_j b_j, \sum_i w_i)$ avec $d_j + \alpha^*$ connexions.
Dans la Figure 2.6, nous trouvons les résultats moyens de tous les algorithmes en fonction de m.
D'après les figures, nous pouvons voir que l'algorithme Seq permet une bonne utilisation des degrés ajoutés. En plus, la borne supérieure est atteinte dans

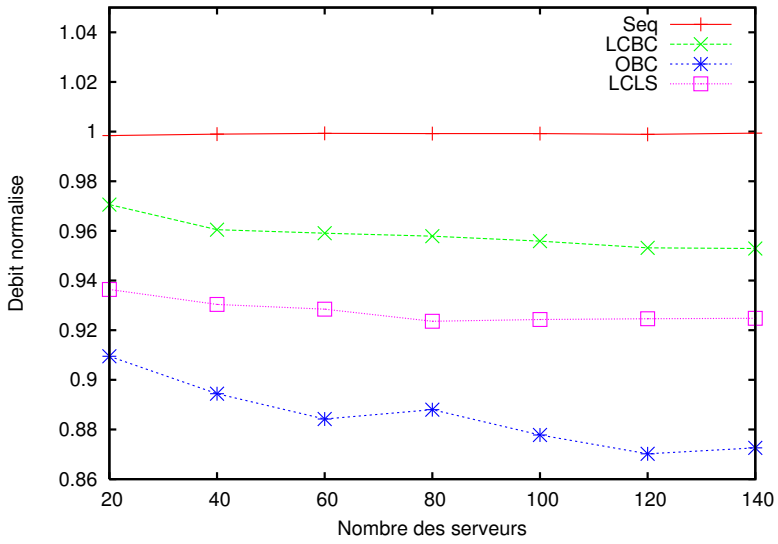

FIGURE 2.5: Débit Moyen Normalisé

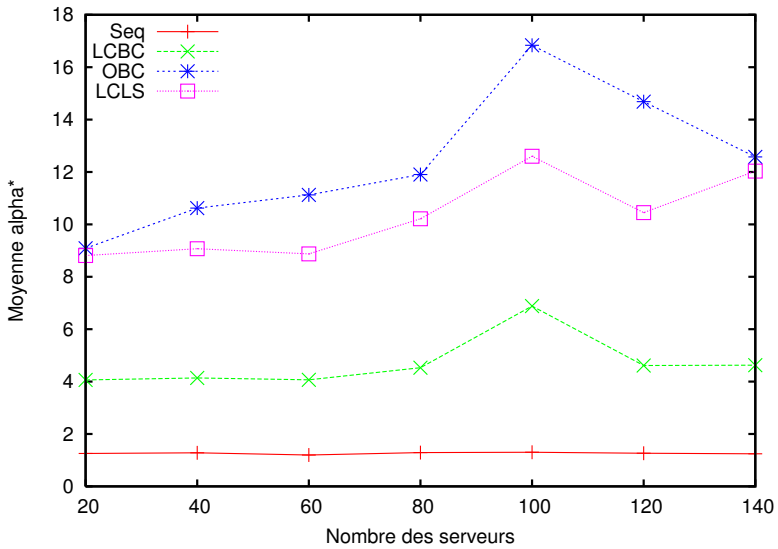

FIGURE 2.6: La moyenne des α^* pour 250 instances.

la majorité des cas en ajoutant une unité (+1) ou deux (+2).
Pour les autres algorithmes, nous remarquons qu'il y a une conservation du même classement que celui observé pour le débit total. Nous observons que LCBC a besoin, à peu près, de quatre connexions en plus pour atteindre la borne, LCLS a besoin, à peu près, de dix connexions, et OBC de douze connexions. En fait, si nous examinons bien les résultats (voir [15] pour pus de détails), nous remarquons bien que la majorité des valeurs de LCBC sont entre deux et cinq. Par contre, ces valeurs peuvent être, de l'ordre de 80 pour des instances avec une grande distribution des capacités des clients. Ainsi ces valeurs assez élevées augmentent la moyenne. Le même comportement, peut être observé en examinant LCLS et OBC, c'est-à-dire des valeurs élevées de α^* pour des instances assez hétérogènes. Ceci explique la valeur moyenne assez élevée. Ainsi, pour ce cas des instances hétérogènes, nous pouvons voir le profit de la garantie prouvée dans la section 2.5 de l'algorithme Seq. En plus, dans l'ensemble des simulations, la valeur moyenne de α est de dix, donc une valeur de α^* plus élevée de cinq par exemple peut dégrader significativement la performance des serveurs. Par conséquent, les algorithmes gloutons, n'arrivent pas à bien utiliser la capacité totale de la plate-forme dans les cas très hétérogènes, par contre, une ou deux connexions en plus sont suffisantes en utilisant Seq.

2.8 Conclusion

Dans ce chapitre, nous avons traité le problème d'ordonnancement des tâches indépendantes sur une plate-forme hétérogène formée de serveurs et de clients. Nous avons prouvé que si on borne le nombre de connexions qu'un serveur peut ouvrir simultanément avec les clients, le problème devient NP-Complet dans le sens fort. Néanmoins, nous avons proposé un algorithme polynomial Seq permettant d'atteindre le débit optimal en utilisant une très petite augmentation de ressources. En particulier, nous avons prouvé qu'une augmentation d'une unité (+1) du nombre maximal de connexions possibles (d_j) est suffisante pour calculer en temps polynomial une solution ayant pour débit au moins l'optimal avec d_j connexions. Enfin, nous avons effectué un ensemble de simulations pour prouver la performance de l'algorithme proposé (Seq).

L'approche présentée dans ce chapitre consiste à déterminer une contrainte faible qui rend le problème d'ordonnancement des tâches indépendantes NP-Complet et procéder ensuite à une augmentation de ressources sur ce paramètre. Nous considérons cette approche très prometteuse dans le contexte de l'ordonnancement en régime permanent, puisqu'elle permet de considérer des modèles de communications plus réalistes sans s'appuyer sur des algorithmes d'approximation qui limitent le débit espéré.

Une extension naturelle de ce travail consiste à considérer la version "en ligne" de ce problème où l'ensemble des clients n'est pas connu à l'avance. Ce nouveau problème sera traité dans le chapitre suivant.

Chapitre 3
OSEQ : Problème "en ligne"

Dans ce chapitre, nous allons présenter la version "en ligne" de SEQ (OSEQ) dans laquelle l'ensemble de clients n'est pas connu à l'avance (les clients arrivent et partent à tout moment). Nous commençons par une modélisation du problème. Puis nous décrivons l'algorithme OSEQ. Ensuite, nous prouvons qu'il n'existe pas d'algorithme d'approximation totalement "en ligne" pour ce problème. Par contre, nous montrons qu'il est possible d'atteindre l'optimal avec un coût d'au plus quatre changements par serveur à chaque fois qu'un nouveau nœud s'ajoute au système ou le quitte. Enfin, nous présentons quelques résultats expérimentaux où nous comparons la performance et le coût de OSEQ avec ceux d'autres heuristiques gloutonnes.

3.1 Modélisation du problème et coûts de maintenance

Résoudre la version "en ligne" de MDDB doit satisfaire deux objectifs. Le premier est de chercher à maximiser le débit (obtenir le débit optimal est possible en utilisant l'augmentation des ressources du degré comme nous l'avons vu dans le cas hors ligne). Le deuxième est d'effectuer un nombre minimal de changements dans les connexions existantes à chaque serveur.

Afin de pouvoir comparer les solutions "en ligne" des différents algorithmes, nous devons définir précisément le coût des modifications apportées à l'allocation des clients aux serveurs.

Nous allons commencer, par présenter le modèle de la plate-forme sur lequel nous allons travailler. Puis nous définirons et nous calculerons le coût des changements d'allocation des clients aux serveurs.

3.1.1 Modélisation du problème

Ce problème correspond à la version "en ligne" du problème MDDB. Nous gardons les mêmes notations que dans la première partie, en particulier celles concernant la description des clients $\mathcal{C}_i$ et des serveurs $\mathcal{S}_j$. Nous introduisons la notion de rondes. Une nouvelle ronde commence lorsqu'un client arrive ou quitte le système. Il n'y a pas de durée associée à une ronde. Soit $\mathcal{C}^t$ l'ensemble des clients

présents à la ronde t. Le client C *rejoint* (resp. *quitte*) le système à la ronde t si $C \in \mathcal{C}^t \backslash \mathcal{C}^{t-1}$ (resp. $C \in \mathcal{C}^{t-1} \backslash \mathcal{C}^t$). L'arrivée et le départ des clients ne peut se produire qu'au début d'une ronde et $\forall t,\ |\mathcal{C}^t \backslash \mathcal{C}^{t-1}| + |\mathcal{C}^{t-1} \backslash \mathcal{C}^t| \leqslant 1$.

3.1.2 Coûts de maintenance

Soit $w_i^j(t)$ la fraction de bande passante sortante du serveur S_j affectée au client C_i à la ronde t. On dit que :

- Un client C_i est *connecté* au serveur S_j à la ronde t si $w_i^j(t) > 0$.
- La connexion entre le serveur S_j et le client C_i *change* à la ronde t si $w_i^j(t-1) \neq w_i^j(t)$. Soit $\mathcal{N}_j^t = |\{i, w_i^j(t-1) \neq w_i^j(t)\}|$ le nombre des changements du serveur $\mathcal{S}_j$ à la ronde t.

Les changements peuvent correspondre à trois situations différentes :

$w_i^j(t-1) = 0$ **et** $w_i^j(t) > 0$ **:** Une nouvelle connexion au serveur S_j.

$w_i^j(t-1) > 0$ **et** $w_i^j(t) = 0$ **:** Le client $\mathcal{C}_i$ est déconnecté du serveur $\mathcal{S}_j$.

$w_i^j(t-1)$ **et** $w_i^j(t)$ **positifs et différents :** Changement de la qualité de service entre le client $\mathcal{C}_i$ et le serveur $\mathcal{S}_j$.

Dans notre contexte, nous utilisons des mécanismes de contrôle de la bande passante complexes pour assurer la bande passante prévue entre les clients et les serveurs, donc tout changement dans l'allocation de la bande passante induit un coût. Si un nouveau client se connecte à un serveur, une nouvelle connexion TCP doit être ouverte, ce qui induit également un coût. Par ailleurs, toutes les modifications de bande passante des connexions faites par des serveurs différents peuvent se dérouler en parallèle. Par conséquent, nous introduisons la définition suivante nécessaire pour mesurer et comparer les algorithmes qui résolvent le problème MDDB "en ligne".

Définition 3.1 *Soit $\mathcal{A}$ un algorithme résolvant la version "en ligne" de* MDDB. *On dit que $\mathcal{A}$ produit au plus l* changements *de connexions par ronde si :*

$$\max_t \max_{S_j \in \mathcal{S}} \mathcal{N}_j^t \quad \leqslant l$$

On dira alors que le coût *de l'algorithme $\mathcal{A}$ est l.*

3.1.3 Algorithme OSeq

Nous pouvons voir, dans un premier temps, OSeq comme un algorithme *pseudo en ligne*, dans le sens où il correspond au calcul de la nouvelle solution optimale au début de chaque événement. Il est donc équivalent à l'application de l'algorithme Seq à l'instance en cours (en considérant les serveurs dans le même ordre). Nous commençons par présenter, ci-après, une vue globale de l'algorithme.

Dans la suite, nous allons considérer la liste des clients $\mathcal{C} = \{\mathcal{C}_i\}$ ordonnée selon les capacités croissantes des clients. Nous allons noter $\mathcal{C}(l,k) = \sum_{i=l}^{k} w_i$ la somme des capacités des clients entre $\mathcal{C}_l$ and $\mathcal{C}_k$ (où $\mathcal{C}_l$ et $\mathcal{C}_k$ sont deux éléments de la liste).

Dans la section 2.5.2, nous avons décrit l'algorithme Seq qui présente une étape de OSeq. Cette procédure d'allocation va être la base de notre analyse. Une étape

Algorithm 2 Algorithm OSEQ

Soit $\mathcal{S} = \mathcal{S}$ la liste des serveurs ;
Soit $\mathcal{C} = \text{sort}(\mathcal{C}^t)$ l'ensemble des clients disponibles et ordonnés à la ronde t ;

for $j = 1$ à $|\mathcal{S}|$: **do**
Calculer $S_j(\mathcal{C})$ l'ensemble des clients dans $\mathcal{C}$ à connecter avec $\mathcal{S}_j$ en appliquant l'algorithmeSEQ
Soit $\mathcal{C} = \mathcal{C}_{\mathcal{S}_j}$, la liste actualisée des clients
end for

RETOURNER $S_1(\mathcal{C}), S_2(\mathcal{C}), \ldots, S_{|\mathcal{S}|}(\mathcal{C})$, la liste des clients alloués pendant la ronde t.

d'un serveur particulier de capacité b et degré d, sera noté par OSEQ(d, b). À partir d'une liste des clients ordonnée, il calcule une allocation du serveur et une liste $\mathcal{C}'$ actualisée des clients.

L'algorithme complet consiste simplement à des applications successives de OSEQ(d_j, b_j) à tous les serveurs $\mathcal{S}_j$. En s'appuyant sur les propriétés de SEQ, on obtient donc directement le résultat suivant.

Lemme 3.2 *Le débit fourni par* OSEQ *à chaque ronde est au moins le débit optimal quand la contrainte de degré est satisfaite (avec une augmentation de ressources d'au plus 1 sur le degré des serveurs).*

3.2 Résultats d'inapproximation

Dans cette section, nous allons prouver qu'il n'existe pas d'algorithme d'approximation (avec un rapport constant) totalement "en ligne" polynomial pour MDDB. Ce résultat reste vrai quelle que soit l'augmentation de ressources, contrairement, au cas "hors ligne", où le débit optimal peut être atteint en permettant une augmentation seulement d'une unité du degré de chaque serveur. La preuve va s'appuyer sur un contre exemple.

Nous savons qu'un algorithme $\mathcal{A}_\alpha$ utilise *un facteur d'augmentation de ressources additif* $\alpha \geqslant 1$, lorsque le degré maximal utilisé par $\mathcal{S}_j$ est $d_j + \alpha$ alors que son degré original est d_j. Notons par $OPT(I)$ le débit optimal pour une instance I et par $\mathcal{A}_\alpha(I)$, le débit calculé par $\mathcal{A}_\alpha$ sur I.

Théorème 3.3 *Etant donnés le facteur d'augmentation de ressources α et une constante k, il existe une instance I de la version en ligne de* MDDB*, telle que pour tout algorithme $\mathcal{A}_\alpha$ de coût inférieur à 2,*

$$\mathcal{A}_\alpha(I) < \frac{1}{k} OPT(I).$$

Preuve. La preuve consiste, tout simplement, à présenter un contre exemple sous forme d'une instance I pour laquelle tout algorithme en ligne de coût inférieur à 2 ne va pas réussir à atteindre le facteur d'approximation requis.

Instance :

- Un serveur $\mathcal{S}$ de bande passante $b = (2k)^{\alpha+1}$ et degré $d = 1$.
- un ensemble de clients $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_{\alpha+1}$ de capacités respectives $1, 2k, (2k)^2, \ldots, (2k)^{\alpha+1}$.
- les clients arrivent un par un et par capacité croissante. En particulier, à l'étape j, avec $0 \leqslant j \leqslant \alpha+1$, le nouveau client C_j de capacité $(2k)^j$ est ajouté.

Puisque le degré du serveur est 1, seulement un client peut être connecté au serveur. De plus, comme les clients arrivent par capacité croissante, alors la solution optimale consiste, à l'étape j, à connecter $\mathcal{C}_j$ au serveur. Dans ce cas, le coût de maintenance de la solution optimale, à chaque étape, est égal à 2 (à l'étape j, le client $\mathcal{C}_j$ se connecte et le client $\mathcal{C}_{j-1}$ se déconnecte).

En particulier, tout algorithme "en ligne", qui peut s'autoriser un rapport d'approximation au plus k, doit connecter $\mathcal{C}_j$ au serveur à l'étape j puisque la capacité de $\mathcal{C}_j$ est $\frac{3}{2}k$ fois la capacité de tous les clients précédents ($\sum_0^{j-1}(2k)^i < (\frac{3}{2}k)(2k)^j$). Par conséquent, un algorithme en ligne dont le facteur d'approximation est au plus k doit connecter un nouveau client à chaque étape. Donc, si son coût est inférieur à 2, il ne peut pas déconnecter des clients. Par conséquent, le degré du serveur va devenir arbitrairement grand, ce qui ne permet pas d'avoir une augmentation des ressources bornée.

3.3 Résultats d'approximation

Dans la section précédente, nous avons prouvé qu'il n'existe pas d'algorithme totalement en ligne et polynomial, dont le coût est inférieur à 2, qui peut approximer MDDB. Par contre, dans cette section, nous allons montrer qu'il existe un algorithme polynomial dont le coût est au plus 4, avec une augmentation d'une unité (+1) de ressources et qui peut atteindre le débit optimal.

Nous savons, d'après le lemme 3.2, que le débit atteint par OSEQ à chaque ronde est au moins le débit optimal quand la contrainte de degré est satisfaite. Nous allons montrer, dans la suite, que les solutions calculées par OSEQ entre deux événements, c'est-à-dire quand un client rejoint ou quitte la plate-forme, diffèrent par, au plus, quatre changements au niveau d'un serveur. Pour cela, nous allons observer les différences entre les listes des clients restants au cours de l'exécution de OSEQ.

Définition 3.4 *Soient $\mathcal{C} = (\mathcal{C}_i)_{1\leqslant i\leqslant n}$ et $\mathcal{LD} = (\mathcal{LD}_i)_{1\leqslant i\leqslant n+1}$ deux listes ordonnées de clients. On dit que $\mathcal{LD}$ est une version* augmentée *de $\mathcal{C}$ si elle est obtenue à partir de $\mathcal{C}$ par l'insertion d'un nouveau client et la possibilité de l'augmentation de la capacité du client immédiatement suivant. Plus formellement, $\mathcal{C}$ est* augmentée *de $\mathcal{LD}$ s'il existe un entier $p \leqslant n$, un nouveau client $\mathcal{X}$ et une valeur $y \geqslant 0$ tels que $\mathcal{LD} = (\mathcal{C}_1, \ldots, \mathcal{C}_{p-1}, \mathcal{X}, \mathcal{C}_p^{(i)}, \mathcal{C}_{p+1}, \ldots, \mathcal{C}_n)$, avec $w(\mathcal{X}) \leqslant w_p$ et $w(\mathcal{C}_p^{(i)}) = w_p + y$.*

Le lemme suivant montre qu'une liste de clients et sa version augmentée, allouées au même serveur, produisent presque la même allocation. Soient une liste $\mathcal{C}'$ mise à jour et une allocation $\mathcal{A}$, le résultat de l'application de OSEQ(d, b) à la liste des

clients $\mathcal{C}$. De même, soient $\mathcal{LD}'$ et $\mathcal{B}$ le résultat de l'application de OSeq(d, b) à une autre liste $\mathcal{LD}$.

Lemme 3.5 *Si $\mathcal{LD}$ est une version augmentée de $\mathcal{C}$, alors $\mathcal{LD}'$ est une version augmentée de $\mathcal{C}'$, et les allocations $\mathcal{A}$ et $\mathcal{B}$ diffèrent par au plus 4 changements.*

Preuve. Soient $\mathcal{LD} = (\mathcal{C}_1, \ldots, \mathcal{C}_{p-1}, \mathcal{X}, \mathcal{C}_p^{(i)}, \mathcal{C}_{p+1}, \ldots, \mathcal{C}_n)$, où $w(\mathcal{C}_{p+1}^{(i)}) = w_p + y$ et $w(\mathcal{X}) = x$.
Nous allons commencer par le calcul des sommes partielles $\mathcal{LD}(u, v)$ pour tout $u \leqslant v \leqslant n$. Une étude de cas nous montre que :

$$\mathcal{LD}(u,v) = \begin{cases} \mathcal{C}(u-1, v-1) & \text{Si } p < u-1 \\ \mathcal{C}(u-1, v-1) + y & \text{Si } p = u-1 \\ \mathcal{C}(u, v-1) + x + y & \text{Si } u \leqslant p < v \\ \mathcal{C}(u, v-1) + x & \text{Si } p = v \\ \mathcal{C}(u, v) & \text{Si } p > v \end{cases}.$$

En particulier, vu que, par hypothèse, $x \leqslant w_p$ et $x + y \leqslant w_{p+1}$, alors nous aurons dans tous les cas, $\mathcal{LD}(u, v) \leqslant \mathcal{C}(u, v)$. En plus, vu que $x \geqslant w_{p-1}$, nous aurons $\mathcal{LD}(u, v) \geqslant \mathcal{C}(u-1, v-1)$.

Soit OSeq(d, b) une application de OSeq à $\mathcal{C}$. Soit $[l, l+d]$ un intervalle tel que $\mathcal{C}(l, l+d-1) < b$ et $\mathcal{C}(l+1, l+d) \geqslant b$ et tel que l'allocation $\mathcal{A}$ est $(\mathcal{C}_l, \ldots, \mathcal{C}_{l+d-1}, \mathcal{C}_{l+d}^{(a)})$ et la liste mise à jour $\mathcal{C}'$ est $(\mathcal{C}_1, \ldots, \mathcal{C}_{l-1}, \mathcal{C}_{l+d}^{(b)}, \ldots, \mathcal{C}_n)$, où $\mathcal{C}_{l+d}^{(a)}$ et $\mathcal{C}_{l+d}^{(b)}$ présentent les deux parties du client fractionné $\mathcal{C}_{l+d}$.

1. Si le changement de $\mathcal{C}$ à $\mathcal{LD}$ est hors intervalle, alors ce changement n'affecte pas notre algorithme. Et donc $\mathcal{A}$ et $\mathcal{B}$ sont les mêmes
2. Sinon, l'allocation résultante $\mathcal{B}$ dépend de la valeur de $\mathcal{LD}(l+1, l+d)$. Or, les bornes précédentes de $\mathcal{LD}(u, v)$ impliquent que $\mathcal{LD}(l, l+d-1) < b$, et $\mathcal{LD}(l+2, l+d+1) \geqslant b$. Donc, ni $[l, l+d]$ ni $[l+1, l+d+1]$ ne sont des intervalles adéquats pour l'application de OSeq à $\mathcal{LD}$.
   - Si $\mathcal{LD}(l+1, l+d) \geqslant b$, alors l'allocation $\mathcal{B}$ sera $(\mathcal{C}_l, \ldots, \mathcal{X}, \mathcal{C}_p^{(i)}, \ldots, \mathcal{C}_{l+d-1}^{(a)})$. Cette liste diffère de $\mathcal{A}$ exactement par quatre changements. Il s'agit de deux modifications, un ajout et une suppression : l'ajout de $\mathcal{X}$, la suppression de $\mathcal{C}_{l+d}^{(a)}$, et les modifications de $\mathcal{C}_p$ et $\mathcal{C}_{l+d-1}$. D'autre part, $\mathcal{LD}'$ est une version augmentée de $\mathcal{C}'$. En effet, $\mathcal{LD}'$ est formée de $\mathcal{C}_1, \ldots, \mathcal{C}_{l-1}, \mathcal{C}_{l+d-1}^{(b)}, \mathcal{C}_{l+d}, \ldots, \mathcal{C}_n$ et la capacité du client $\mathcal{C}_{l+d-1}^{(b)}$, inséré entre $\mathcal{C}_{l-1}$ et $\mathcal{C}_{l+d}^{(b)}$, est augmentée à w_{l+d}. En effet, selon le principe du processus de fragmentation, $w(\mathcal{C}_{l+d-1}^{(b)}) = \mathcal{LD}(l, l+d) - b$ et $w(\mathcal{C}_{l+d}^{(b)}) = \mathcal{C}(l, l+d) - b$, ce qui implique $w(\mathcal{C}_{l+d-1}^{(b)}) \leqslant w(\mathcal{C}_{l+d}^{(b)})$.
     Dans le cas où $p = l+d$, $\mathcal{X}$ est le client fractionné et seulement deux changements sont produits (un ajout et une suppression). Il s'agit de l'ajout d'une partie de $\mathcal{X}$ et de la suppression de $\mathcal{C}_{l+d}^{(a)}$. La liste $\mathcal{LD}'$ est, donc, une version augmentée de $\mathcal{C}'$, avec l'insertion de la fraction de $\mathcal{X}$ et l'augmentation de la capacité du client $\mathcal{C}_{l+d}^{(b)}$ à $w_{l+d} + y$.

– Dans le cas où $\mathcal{LD}(l+1, l+d) < b$, l'intervalle convenable est $[l+1, l+d+1]$ et l'allocation résultante $\mathcal{B}$ sera $(\mathcal{C}_{l+1}, \ldots, \mathcal{X}, \mathcal{C}_p^{(i)}, \ldots, \mathcal{C}_{l+d}^{(a')})$. Cette nouvelle allocation diffère de $\mathcal{A}$ par quatre changements aussi (un ajout, une suppression et deux modifications). Il s'agit de l'ajout de $\mathcal{X}$, la suppression de $\mathcal{C}_l$, et la modification de deux clients $\mathcal{C}_p$ et $\mathcal{C}_{l+d}^{(a)}$. La liste, mise à jour des clients, $\mathcal{LD}'$, devient $(\mathcal{C}_1, \ldots, \mathcal{C}_l, \mathcal{C}_{l+d}^{(b')}, \ldots, \mathcal{C}_n)$, ce qui correspond à une version augmentée de $\mathcal{C}'$. En effet, le client $\mathcal{C}_l$ est inséré entre $\mathcal{C}_{l-1}$ et $\mathcal{C}_{l+d}^{(b)}$, et sa capacité est augmentée à $w(\mathcal{C}_{l+d}^{(b')})$. Le fait que $w_l \leqslant w(\mathcal{C}_{l+d}^{(b)})$ vient de la propriété de localité de OSeq (voir section 3.1.3.).
Dans le cas où $p = l$, $\mathcal{X}$ ne sera pas inclus dans $\mathcal{B}$. Donc, dans ce cas, nous n'avons que deux changement : la modification de la capacité de $\mathcal{C}_p$ et le fait que le client $\mathcal{C}_{l+d}$ est fractionné dune autre façon. $\mathcal{LD}'$, dans ce cas aussi, est une version augmentée de $\mathcal{C}'$, avec l'insertion de $\mathcal{X}$ et l'augmentation de la capacité de $\mathcal{C}_{l+d}^{(b)}$.

Théorème 3.6 *Le coût de l'algorithme* OSeq *est au plus 4.*

Preuve. Nous allons prouver que si deux listes données $\mathcal{C}$ et $\mathcal{LD}$ de OSeq diffèrent seulement par l'ajout d'un nouveau client, alors les allocations résultantes à chaque serveur diffèrent par au plus quatre changements.

Soit $\mathcal{C}^j$ la liste courante des clients après l'application des j premières étapes de OSeq, qui commence à partir du $\mathcal{C}^0 = \mathcal{C}$, et de même pour $\mathcal{LD}$. Il est clair que $\mathcal{LD}$ est une version augmentée de $\mathcal{C}$ et que selon le lemme 3.5, si $\mathcal{LD}^j$ est une version augmentée de $\mathcal{C}^j$, alors $\mathcal{LD}^{j+1}$ est une version augmentée de $\mathcal{C}^{j+1}$. Puis par induction, le lemme 3.5 s'applique le long de l'exécution de OSeq, et toutes les allocations fournies diffèrent par au plus quatre changements.

Pour le cas de départ d'un client, nous pouvons simplement inverser les rôles dans les étapes précédentes.

3.4 Résultats expérimentaux

3.4.1 Heuristiques de comparaison

Dans la suite, nous allons présenter les algorithmes (heuristiques) que nous allons comparer avec Seq (OBC, OLS).

OBC (Online Best Connection) : les serveurs sont ordonnés selon leur *capacité par connexion* restante, qui est définie comme le rapport entre la capacité restante b'_j et le degré restant d'_j. Quand un nouveau client arrive, il se connecte au serveur dont la capacité par connexion est la plus proche de celle du client. S'il n'a pas de serveur disponible, OBC cherche un serveur ayant encore de la capacité disponible même s'il utilise tout son degré, et il échange le nouveau client avec le plus petit client connecté à ce serveur. Le serveur choisi, en cas de l'existence de plusieurs choix possibles, est celui qui réalise le plus de gain au niveau du débit global.

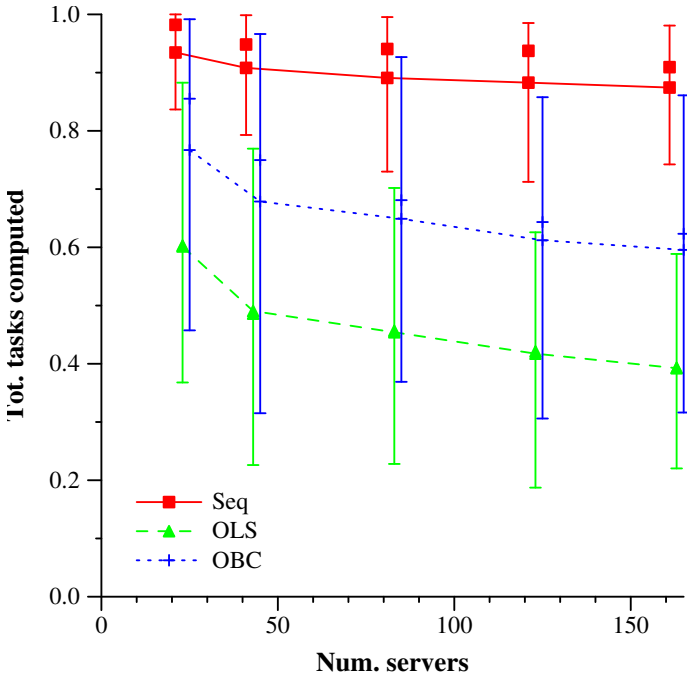

FIGURE 3.1: Moyenne normalisée des tâches calculées

Quand un client $\mathcal{X}$ quitte le système, OBC essaie d'utiliser la nouvelle bande passante disponible pour réduire le degré entrant des autres clients. Nous supposons que $\mathcal{X}$ s'est connecté au serveur $\mathcal{S}$, et que le client $\mathcal{Y}$ est connecté à la fois à $\mathcal{S}$ et $\mathcal{S}'$. Lorsque $\mathcal{X}$ quitte le système, le serveur $\mathcal{S}$ pourrait ré-allouer la bande passante correspondante au client $\mathcal{Y}$. Cette opération peut être intéressante si elle permet de déconnecter $\mathcal{Y}$ de $\mathcal{S}'$, puisqu'elle baisserait le degré sortant du $\mathcal{S}'$. OBC choisit, tant que possible, de nombreuses connexions incidentes de ce genre. S'il existe des clients non connectés, OBC réagit comme s'ils venaient d'arriver et essaie de les connecter avec la procédure décrite avant.

OLS (Online Largest Server OLS) : cette heuristique est très similaire à la précédente. La seule différence est l'ordre dans lequel les serveurs sont triés. Dans OLS, quand un nouveau client arrive, il se connecte au serveur ayant la plus grande bande passante disponible. Le reste de l'heuristique est identique.

3.4.2 Simulations

L'étude expérimentale s'appuie sur un ensemble de simulations pour des instances différentes où nous varions le nombre m des serveurs. Pour chaque valeur de m, 250 instances sont générées. Sur les figures, nous trouvons les valeurs moyennes, la médiane, le premier et le dernier décile de ces 250 instances. Pour chaque algorithme, le trait connecte les valeurs moyennes. La barre d'erreur supérieure montre le dernier décile, ce que signifie que dans 10% d'instances, la valeur est plus élevée. La barre d'erreur inférieure montre le premier décile, la valeur était inférieure dans 10% des instances. La seule valeur entre les deux est la médiane (la moitié des instances a des valeurs inférieures).

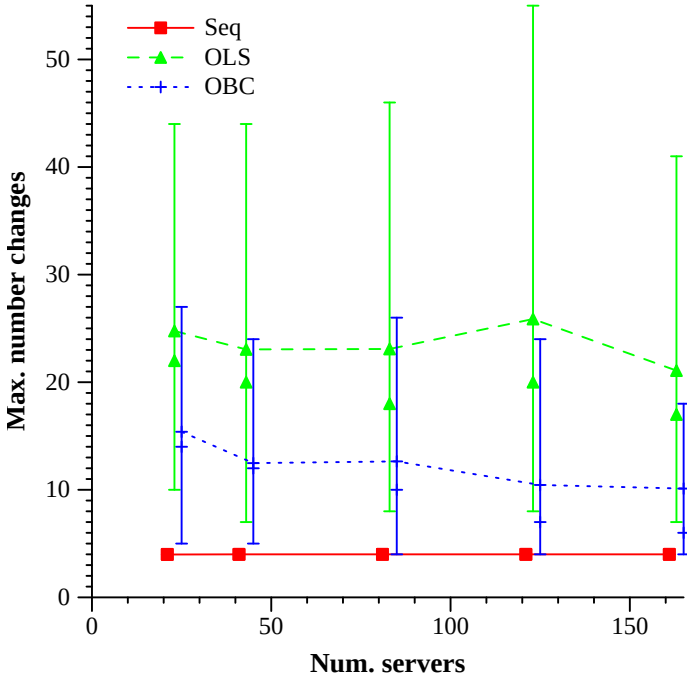

FIGURE 3.2: Coût maximal.

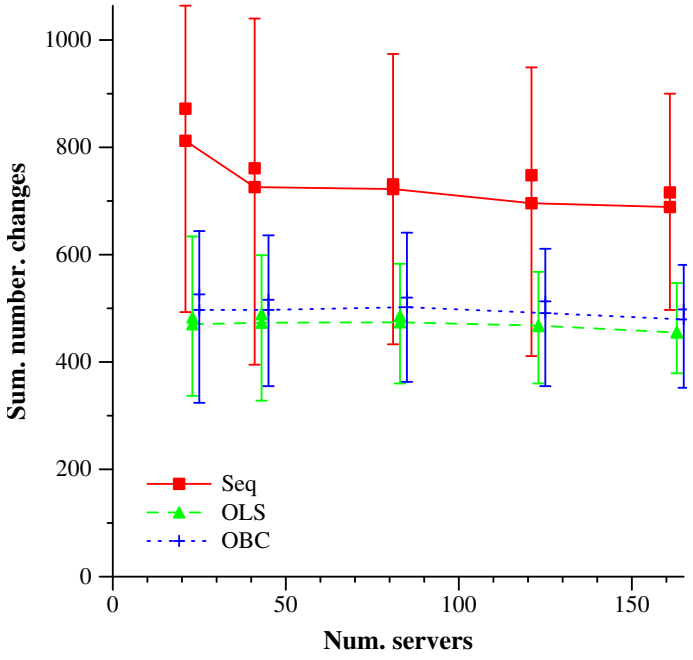

FIGURE 3.3: Coût total des 300 événements.

Dans la figure 3.1, nous décrivons le nombre total des tâches calculées, tout au long de l'exécution, qui représente simplement l'intégrale en temps du flux instantané. Pour des raisons de simplicité, nous allons supposer que le changement d'une solution à une autre ne nécessite pas de temps. La valeur obtenue est normalisée selon la borne supérieure $\min(\sum_j b_j, \sum_i w_i)$. Nous pouvons remarquer (alors que la borne supérieure n'est pas toujours accessible à cause de la contrainte de degré) que OSeq réussit dans 90% des cas à l'atteindre. D'autre part, la performance de OBC se dégrade en moyenne lorsque la taille de l'instance augmente, et est en plus très variable (elle peut être à seulement 30% de la borne supérieure, même pour des petites instances). Pour OLS, la situation est encore pire, avec une performance inférieure en moyenne et à peu près la même variabilité.

Dans la figure 3.2, nous observons le coût des algorithmes. Nous pouvons voir que le coût de OSeq est toujours de 4, celui de OBC est autour de 10 et en moyenne 15. Par contre, il reste aussi très variable et peut atteindre 25 dans 10% des instances. Le comportement de OLS est pire ici aussi, avec un coût de 25 en moyenne, et plus de 40 dans 10% de ces instances, pour toutes les tailles. Or nous avons le degré moyen sortant des serveurs est à peu près 50 (les instances avec m serveurs contiennent $50m$ clients), ce qui signifie qu'il est très probable, en utilisant OLS ou OBC et à un point de l'exécution, que le serveur change plus de la moitié des clients auxquels il est connecté.

Dans la figure 3.3, nous trouvons la somme des coûts de tous les événements (300 événements). Le coût moyen d'un événement de OSeq est à peu près 2.3 (il est au dessus de 3 dans 10% des instances). Pour OBC et OLS, il est autour de 1.6 (variant de 1.3 et 2). Ceci montre que les événements qui demandent beaucoup de changements en OBC et OLS sont relativement rares, et sont entrelacés aux beaucoup d'événements qui génèrent très peu de changements. Par contre, nous sommes persuadés que le coût pour maintenir les garanties est justifié par une performance plus élevée, et pas la stabilité de OSeq.

3.5 Conclusion

Ce chapitre illustre la nécessité de concevoir des algorithmes "en ligne", c'est-à-dire susceptibles de s'adapter à des changements dans l'ensemble des ressources participants ainsi que dans leurs performances. Il est également à noter que dans le cas de OSeq, le surcoût lié au maintien de la solution optimale est tout à fait raisonnable puisqu'aucun algorithme ne peut assurer un facteur d'approximation constant à un coût plus de deux fois inférieur. Toutefois, pour assurer un faible coût de maintenance de OSeq, il est nécessaire de choisir avec précaution la solution optimale de Seq parmi toutes celles qui sont acceptables puisque d'après notre étude, parmi tous les intervalles de clients valides pour être alloués à un serveur, seuls le plus à droite permet d'avoir un coût de maintenance faible.

Deuxième partie

Stockage et redistribution des données

Chapitre 4

Techniques de redistribution des données

4.1 Introduction

Au cours de ces dernières décennies, le volume des contenus numériques a été en constante expansion, à un rythme important (photographie numérique, vidéo *etc.*), d'où le besoin important de la gestion de ces données numérisées. D'autre part, les réseaux informatiques ont connu eux aussi une grande expansion, ce qui a permis aux utilisateurs d'accéder à leurs données depuis différentes machines (cloud computing). Cette tendance est à la source de nombreux défis technologiques, en particulier le stockage et la gestion des données, généralement répartis sur plusieurs unités.

Ceci constitue, aussi, l'un des enjeux majeurs de *Yahoo !*, qui développe une plate-forme distribuée appelée *Vespa* pour l'acquisition, la recherche, le traitement et le placement de données et un système de fichiers distribué pour assurer ces fonctionnalités.

Récemment, de nombreux efforts ont été consacrés aux algorithmes de placement de données pour les systèmes de stockage distribués, afin d'assurer une localisation efficace des données à stocker tenant compte de l'évolution du système et de l'hétérogénéité des composants [64, 53, 103, 101, 87].

Par ailleurs, lorsque le système change de configuration, les données doivent être migrées ou répliquées [53, 94, 44, 101] entre les nœuds de stockage afin de maintenir le niveau de redondance et la cohérence du système ou encore, pour augmenter ses performances.

Bien que plusieurs systèmes de stockage distribués traitent le problème de reconfiguration [100, 44, 53, 94, 101, 83], il y a encore un manque de preuves théoriques sur la complexité des problèmes d'ordonnancement liés à la reconfiguration du système, qui cherchent à déterminer à partir de quels nœuds les données doivent être récupérées. En effet, au cours d'une reconfiguration, la charge de travail, suite au stockage des données sur les différents nœuds et à l'hétérogénéité des liens du réseau, peut dégrader fortement la performance globale du système. À notre connaissance, dans la littérature, ces problèmes d'ordonnancement, dont l'origine est un changement dans la configuration du système, n'ont pas été étudiés.

Dans ce travail, nous faisons un premier pas dans cette direction. En particulier,

nous nous intéressons au problème d'expansion du système. Ce problème est, en fait, inévitable, vu que de nouveaux dispositifs de stockage doivent être ajoutés afin de suivre le rythme de croissance des données et des informations. En revanche, les nouveaux périphériques de stockage ont généralement des capacités supérieures aux anciens. Il est donc souhaitable de transférer les données des anciens noeuds vers les nouveaux dans le but d'améliorer les performances du système. D'ailleurs, il est, généralement, intéressant de mélanger les vieilles et les nouvelles données afin d'éviter les "hot spots" dans le système, car les nouvelles données ont tendance à être consultées plus fréquemment que les anciennes.

Dans la suite, nous allons considérer la situation dans laquelle un seul nœud est ajouté au système. Bien que, l'expansion du système est plus susceptible de se produire dans des lots de plusieurs nœuds, l'étude du cas d'un seul disque permet de réduire la complexité du problème et fournit donc une première étape naturelle pour traiter le cas de plusieurs nœuds.

4.2 Contexte

Notre travail s'inscrit dans le cadre d'une collaboration avec Cyril Babino, de *Yahoo !* Research à Trondheim. Cette collaboration vise la conception d'algorithmes permettant d'optimiser l'ordonnancement et la redistribution des données dans le cas d'une reconfiguration du système. En particulier, nous avons considéré le problème d'ordonnancement lié à l'ajout d'un disque au système de stockage de fichiers distribué *Vespa*.

4.2.1 *Yahoo !*

Yahoo ! est une compagnie qui offre de nombreux autres services gratuits et payants, dont un moteur de recherche [105], des boîtes à courrier, de la messagerie instantanée, de l'hébergement web et des portails (nouvelles, finances, etc). Ces sites Web hébergés servent plus de 500 millions d'utilisateurs uniques par mois et génèrent plusieurs milliards de dollars de revenus par la publicité et les autres services. En raison de cet écosystème, *Yahoo !* génère plus d'une dizaine de téra-octets par jour des données sous diverses formes. Le stockage et la gestion de cet ensemble de données représente un enjeu majeur et pose plusieurs défis.

4.2.2 La plate-forme Vespa

Yahoo ! Technologies Norway développe une plate-forme de stockage de données générique et évolutive *Vespa*. À part le stockage, cette plate-forme permet l'acquisition, le traitement et la recherche dans des grandes quantités de données. À l'heure actuelle, *Vespa* est déployée dans de nombreuses applications *Yahoo !* telles que *Flickr* [41], *Yahoo ! Mail* [106] and *del.icio.us* [34] *etc.* La recherche présentée dans cette partie est motivée par les défis survenus dans le système de stockage de données de *Vespa*, qui est un système de stockage de données distribué destiné au stockage d'énormes quantités de données.

4.3 Systèmes de gestion de fichiers distribués (SGF distribués)

Le système de gestion de fichiers (SGF) permet l'organisation et le stockage des fichiers (où un fichier est une collection nommée d'informations apparentées, enregistrées sur un stockage secondaire). Il permet notamment de gérer la correspondance entre un fichier et les blocs du périphérique qui lui sont associés. On parle de méta-données, pour désigner les informations que le SGF stocke à cet effet. Les systèmes de gestion de fichiers, qu'ils soient centralisés ou répartis, ont toujours occupé une place prépondérante au sein de la thématique des systèmes d'exploitation.

Les principales opérations rendues possibles par un SGF sont,

- la création, la suppression, le déplacement et le renommage d'un fichier,
- la lecture, l'écriture et le positionnement dans un fichier,
- la troncature et la gestion des attributs d'un fichier (droit d'accès, date de modification, taille,...).

Un SGF assure en outre la cohérence du contenu d'un fichier dans le cas d'accès concurrents.

En général, un système de fichiers centralisé offre une sémantique de cohérence forte, dite cohérence séquentielle. Dans le cas des systèmes répartis, maintenir une cohérence forte pour les différents clients est plus difficile et très coûteux, et n'est donc généralement pas offert par tous les SGF distribués. Un SGF implémente également des techniques afin de tolérer les pannes et l'échec de certains composants et assurer la fiabilité du système. Ces techniques s'appuient généralement sur le principe de réplication des données.

Les systèmes de fichiers distribués sont évalués par plusieurs facteurs : réactivité, transparence, disponibilité ou encore capacité de passage à l'échelle. Le terme de réactivité représente la quantité de temps nécessaire pour satisfaire une requête (latence réseau, analyse de la demande et formulation de la réponse). Les autres caractéristiques sont développées ci-après. Dans la suite, nous allons, tout d'abord, présenter les caractéristiques d'un SGF, nous donnerons ensuite un aperçu des quelques SGFs de la littérature.

4.3.1 Évaluation

4.3.1.1 Transparence

Une des propriétés fondamentales d'un système de fichiers est sa transparence [47, 1] vis à vis du réseau. Les clients doivent pouvoir accéder à des ressources distantes en utilisant les mêmes primitives d'opérations sur les fichiers que celles applicables aux fichiers locaux.

Nous rappellerons ici une notion importante dans l'implémentation d'un système de fichiers, le nommage, qui est la relation entre le nom logique et l'identifiant physique. Un usager référence un fichier par un nom textuel. Une correspondance est alors réalisée avec un identifiant numérique (appelé inode ou bloc de méta-données) qui est lui même une interprétation d'une portion (bloc) d'un disque physique. Ces abstractions permettent de cacher à l'utilisateur final les détails de l'implantation

physique du fichier sur le périphérique de stockage. Pour un système de fichiers, il est impératif d'ajouter une nouvelle dimension afin de rendre transparente la localisation du fichier au sein du réseau. Ainsi deux niveaux de transparence ont été définis pour caractériser un système de fichiers ,

- transparence de situation : le nom du fichier ne comporte aucune information sur son emplacement physique,
- indépendance à la mobilité : le nom du fichier n'a pas à être modifié lorsque sa situation géographique au sein du réseau évolue.

4.3.1.2 Tolérance aux pannes

La tolérance aux pannes est un problème essentiel dans les systèmes informatiques (toute une littérature y est d'ailleurs consacrée [82, 30, 35]). Perte de réseau, dysfonctionnement du programme, arrêt d'une des machines, altération d'une unité de stockage, doivent être pris en considération afin de permettre au système de fichiers de continuer à s'exécuter dans un état cohérent. Un système qui requiert un arrêt lors d'une quelconque de ces erreurs n'est pas tolérant aux fautes. De manière simplifiée, toute les solutions, permettant d'améliorer ce facteur, gravitent autour de la redondance matérielle et/ou logicielle.

Dans le cadre la tolérance aux fautes, un fichier est qualifié de

- récupérable : si en cas d'erreur il est possible de retrouver son dernier état cohérent,
- robuste : s'il est en mesure de réparer une éventuelle détérioration du support de stockage.

Un fichier récupérable n'est pas obligatoirement robuste et vice-versa. Une journalisation de chaque opération en cours permet de basculer entre les différents états cohérents. La robustesse des fichiers est réalisée par la mise en place de techniques de redondance (RAID [81], mirroring *etc.*).

La disponibilité des ressources est une autre propriété de la tolérance aux pannes (une information est disponible si quelle que soit la panne, elle est accessible quand un noeud en a besoin). Le fait de ne pas pouvoir accéder à une ressource pour un client peut être vu comme une erreur qu'il est nécessaire de prévoir (défaillance matérielle ou verrou bloquant selon la sémantique).

Enfin, l'ensemble des contraintes générées par l'utilisation des solutions de réplication est à prendre en compte. Du point de vue de l'utilisateur, cette gestion doit êtres totalement invisible. Cependant, il faut que chaque réplicat d'un fichier sur le système soit cohérent selon la sémantique employée. La cohérence entre toutes les images physiques du fichier a un coût non négligeable. Il est parfois obligatoire d'utiliser des solutions effectuant les mises à jour sur un nombre limité d'images seulement.

4.3.1.3 Passage à l'échelle

La capacité d'un système à s'adapter à une forte augmentation de son taux de charge est appelé la scalabilité(ou passage à l'échelle) [59]. Ajouter de nouvelles ressources peut résoudre le problème mais génère parfois un coût additionnel, voire

une remise en cause de la conception du système. Un système de fichiers scalable doit pouvoir croître sans engendrer de tels phénomènes.

Dans un premier temps, afin d'éviter ce type de problèmes, les architectures ont été bornées. Chaque grandeur décrivant l'architecture (débit réseau maximum, puissance d'analyse *etc.*) était fixée à une constante qui limitait le nombre de ressources possibles. Par la suite, l'implémentation de serveurs avec ou sans états plus "intelligents" a permis de repousser certaines limites [62]. Le serveur stocke les informations concernant chaque client pendant une période déterminée. Si au bout de celle-ci, le client n'a pas donné signe de son existence, les données sont effacées. Le serveur n'a plus à s'occuper de l'état de l'ensemble des clients.

4.3.2 Exemples de SGFs distribués

Dans cette section, nous allons présenter trois principaux systèmes de fichiers représentant trois technologies différentes.
Nous présentons le système de fichiers AFS [55] qui fournit un concept général de système de fichiers distribué. En fait, il est impossible de parler des systèmes de fichiers distribués sans présenter ce que nous appellerons la "famille" AFS. En effet, ce système est un des projets les plus anciens parmi les DFS (Distributed File System).

Quant à PVFS [71, 23], il est représentatif des systèmes de fichiers parallèles et introduit le concept de séparation entre les gestionnaires de méta-données et les serveurs de stockage. Les systèmes de fichiers parallèles, qui sont généralement construits à partir d'un grand nombre de serveurs de stockage, cherchent à servir des demandes parallèles.
Le système de fichiers GoogleFS [44], qui est utilisé pour le moteur de recherche Google, est très distribué et assure une haute performance. Ce système de fichiers est très proche de celui utilisé par *Yahoo !* et il cherche à répondre pratiquement aux mêmes besoins.

4.3.2.1 AFS (Andrew File System)

AFS [55] a été développé à l'université de Carnegie Mellon. Il est fondé sur une architecture clients/serveur structurée en sous clusters inter-connectés par un routeur à un réseau fédérateur. Chaque sous-groupe est constitué d'un nombre de postes "clients" et d'un unique serveur. La charge est, de cette manière, répartie sur une collection de serveurs dédiés, fournissant chacun le même ensemble de services à un groupe de clients.

La prise en charge de la dynamicité, qui se fait par la possibilité d'ajouter ou de retirer des nœuds clients, est un avantage considérable qui est apparu dès la version 2. Plusieurs versions apportant des gains de performances se sont succédées. Actuellement, IBM propose une version OpenSource15 d'AFS depuis la fin de l'année 2000 et de nombreux organismes utilisent encore ce logiciel (notamment CMU et le CERN).

4.3.2.2 PVFS (Parallel Virtual File System)

PVSF [71, 23] est un système de gestion de fichiers qui a été développé à la Clemson University et au laboratoire Argonne. Son objectif est de fournir un système de fichiers parallèle pour les grappes de calcul. Il est conçu en priorité pour les besoins des applications de calcul scientifique (qui manipulent d'énormes quantités de données) et pour lesquelles la vitesse d'exécution est la contrainte majeure. Les données manipulées sont généralement facilement reproductibles et/ou archivées sur un support de stockage.

4.3.2.3 XFS(eXtended File System)

XFS a été développé à l'université de Berkeley [104]. L'idée principale est d'éviter les entités centralisées dans l'architecture du SGF et de fournir aux clients une cohérence forte pour l'accès aux données.

Les clients coopèrent et stockent les données selon le modèle pair-à-pair. L'architecture de XFS est définie à travers trois types d'entités que sont les clients, le gestionnaire de méta-données (managers) et les serveurs de données. Chaque nœud endosse ces trois rôles. Il peut gérer et stocker n'importe quelle partie de donnée ou de méta-donnée.

Chaque gestionnaire du système est responsable de la gestion d'un ensemble de fichiers, il maintient une liste des fichiers alloués. Cette liste peut changer au cours du temps suite à l'ajout ou au retrait de nœuds. Un gestionnaire est également chargé d'assurer la cohérence de l'accès aux données et la mise à jour des méta-données. Chaque fichier est associé à un groupe de serveurs de stockage. Utiliser plusieurs SGF permet notamment d'optimiser les Entrées/Sorties concurrentes sur différents fichiers et d'équilibrer la charge. Chaque gestionnaire garde la liste des clients possédant une copie d'un bloc dans sa mémoire, ce que lui permet d'assurer la cohérence des fichiers.

Afin d'optimiser les accès aux serveurs de stockage, chaque client écrit ses données dans un tampon local. Ces informations écrites sont ensuite fragmentées et envoyées aux différents serveurs de stockage du SGF concerné. Cette méthode fournit de bonnes performances d'accès aux données. Par contre, les mécanismes de reprise sur panne et de reconfiguration sont complexes et coûteux, ce qui peut limiter le passage à l'échelle.

4.3.2.4 GoogleFS

Le système de gestion de fichiers GoogleFS [44] a été conçu en 2003 pour les besoins de la société Google. Il a quelques similitudes avec PVFS mais présente également plusieurs caractéristiques atypiques découlant de son contexte d'utilisation. L'architecture de GoogleFS est assez proche de celle de PVFS. Elle est basée sur trois types d'entités logicielles, à savoir clients, serveur principal (maître) et chunkservers. Chaque entité est implémentée sous forme d'un processus utilisateur communiquant avec les autres via TCP/IP.

Les fichiers sont divisés en unités de taille fixe (64 Mo) appelés chunks, allouées par le serveur principal et stockées par les chunkservers. Chaque chunk, désigné par

un identifiant global, est répliqué sur plusieurs chunkservers (3 par défaut). C'est le maître qui est en charge de la gestion des méta-données, de la cohérence des accès et de la surveillance des chunkservers. Pour accéder au contenu d'un fichier, il faut commencer par contacter le maître pour la récupération des méta-données nécessaires. Puis, on peut communiquer directement avec un chunkserver pour les transferts de données. Pour ne pas encombrer le maître, la responsabilité de la synchronisation des exemplaires d'un même chunk est confiée à un chunkserver.

Les méta-données décrivant le contenu d'un fichier sont stockées sur le disque et conservées en mémoire pour accélérer le traitement des requêtes. La correspondance entre un identifiant de chunk et les serveurs qui en stockent une copie n'est pas conservée de manière permanente par le maître. Cette liste est reconstituée à chaque redémarrage du système. Le maître est responsable de l'ajout, de la suppression ou du déplacement d'un chunk.

Les chunkservers choisis par le maître, pour le stockage d'un chunk donné, sont éloignés les uns des autres. Ceci permet de fournir, d'un coté, des performances d'accès équitables à l'ensemble des clients, et de l'autre, d'éviter les scénarios catastrophes (tels qu'une anomalie électrique paralysant toutes les machines d'un même rang). Enfin, le maître maintient à jour des copies de ses méta-données sur différents nœuds.

4.4 Les protocoles de réplications

Un protocole de placement désigne la stratégie du placement des données (copies) sur les différentes unités de stockages. Ces protocoles de placement peuvent avoir un grand impact sur la fiabilité du système [70] et affecter la vitesse de sa réparation en cas de panne. Ces protocoles diffèrent principalement par leur stratégie de placement des différentes répliques. Principalement, nous trouvons deux types : le placement séquentiel et le placement aléatoire.

Avant de détailler les différentes stratégies de réplication, nous allons présenter en bref le problème et une modélisation plus précise sera détaillée dans la suite.

4.4.1 Placement séquentiel vs aléatoire

Le principe de ce type de placement est simple pour une réplication de degré k des données. Une unité (de stockage) sera choisie comme unité principale, et les k répliques seront placées sur cette unité principale et les $(k-1)$ unités suivantes. C'est le principe utilisé par la dissociation chaînée (chained declustering [56, 46], qui a été employé par Petal [67], ainsi que, par plusieurs systèmes de stockage utilisant une table de hachage distribuée (DHT) tel que CFS [32] et PAST [85], où l'unité principale est identifiée à travers une fonction de hachage.

Dans le placement aléatoire, comme son nom l'indique, les répliques sont placées aléatoirement sur les N unités disponibles. Ce principe a été employé par GFS (Google File System, voir section 4.3.2.4) et dans [97].

4.4.2 Crush : placement pseudo-aléatoire

Dans les paragraphes précédents, nous avons présenté les deux principales stratégies de placement de données utilisées par les systèmes de stockage de données. Une alternative est de combiner les deux stratégies en une "pseudo-aléatoire". Cette stratégie est utilisée par Crush (Controlled Replication Under Scalable Hashing) [99], un algorithme de distribution de données pseudo-aléatoire utilisé par *Vespa*. Il est implémenté comme une fonction pseudo-aléatoire et déterministe qui fait correspondre à une valeur d'entrée, désignant un identifiant d'un objet ou d'un groupe d'objets, une liste d'unités sur lesquelles les répliques vont être stockées, en tenant compte de l'organisation des grappes (clusters) et des capacités des serveurs.

Cette distribution est contrôlée, en fait, par une carte des clusters qui représente les ressources de stockage disponibles. La politique de distribution est définie selon des règles de placement, qui spécifient le nombre d'unités de stockage du même cluster qui peuvent être choisies pour stocker des répliques du même fichier ainsi que les restrictions imposées sur le placement de cette réplique.

Crush résout simultanément le problème de répartition de données en répondant à la question "où doit-on placer les données ?" et le problème de localisation de données en répondant à la question "où a-t-on placé les données ?", en s'appuyant sur une unique fonction pseudo-aléatoire.

Afin d'obtenir une distribution uniforme des données, chaque dispositif de stockage est caractérisé par un poids. Cette distribution est contrôlée par une carte de cluster qui représente hiérarchiquement les ressources de stockage disponibles. La carte de cluster est composée de deux types d'unités de stockage : les "disques de stockage" et les "unités intermédiaires". Les disques de stockage se trouvent sur les feuilles. Les unités intermédiaires, quant à eux, représentent les noeuds intermédiaires et sont de quatre types différents (des unités uniformes, des unités liste, des unités arbre, des unités en paille). Chaque type de ces unités a une structure interne différente et utilise une fonction de sélection pseudo-aléatoire différente lors du choix des unités de stockage pendant le processus de placement des copies.

4.4.2.1 Carte de cluster hiérarchique

La carte de cluster est composée des disques de stockage et des unités intermédiaires. Une unité intermédiaire peut contenir d'autres unités intermédiaires ou des disques. En outre, si on représente la carte de cluster par une hiérarchie ou un arbre pondéré, les unités intermédiaires forment les nœuds intermédiaires et les disques forment les feuilles. Le poids de ces différentes unités de stockage est alloué par l'administrateur pour contrôler la quantité relative des données qu'elles sont capables de stocker. Bien qu'un grand système puisse contenir des disques avec des capacités et des caractéristiques de performance variées, une distribution des données aléatoire corrèle statistiquement l'utilisation des disques avec la charge des données, ce qui résulte en une métrique unidimensionnelle de placement des données, le poids, qui doit être dérivé des capacités des disques.

4.4.2.2 Principe de placement des données

Étant donné un entier x et une carte de clusters, on cherche une description compacte et hiérarchique des différents dispositifs constituant le cluster de stockage, et des règles de placement. Crush permet de générer une liste ordonnée de n destinations de stockage distinctes en utilisant une fonction de hachage entière à plusieurs sorties. La répartition est pseudo-aléatoire donc il n'y a pas de corrélation apparente entre les sorties résultantes d'entrées proches. On dit que Crush génère une répartition "non clusterisée" des différentes copies dans le sens où les ensembles de disques stockant un même élément sont indépendants des autres.
x représente l'identifiant d'un objet ou d'un groupe d'objets dont les copies vont être placées sur les mêmes unités de stockage. Ensuite, CRUSH applique un ensemble de règles de placement permettant de choisir les dispositifs de stockage des différentes copies de x, via l'usage consécutif d'une fonction (select(n,t)) qui permet de sélectionner, à chaque niveau de la hiérarchie, n unités de type t.

4.4.2.3 Équilibrage de charge

Un élément crucial au niveau de la distribution des données dans les systèmes de fichiers est l'équilibrage de la charge lors du changement du système. L'équilibrage de charge couvre l'ensemble des techniques permettant une distribution équitable de la charge de travail entre les ressources disponibles d'un système, afin d'optimiser le temps de réponse moyen d'un ensemble de tâches. Ceci nécessite, en général, une réallocation des données pour assurer l'équilibrage, notamment lors des changements dynamiques de la charge du serveur ou de la capacité du stockage. Ce changement est généralement provoqué par le départ ou l'ajout d'une nouvelle unité.

Pour répondre à une telle problématique et éviter tout déséquilibre asymétrique de la charge, CRUSH maintient, en permanence, une distribution uniforme de la charge et des données. Lorsqu'un disque tombe en panne, CRUSH marque ce disque et le laisse dans la hiérarchie. Puis il redistribue son contenu sur les autres disque via l'algorithme de placement. Un tel changement résulte en une fraction $\frac{w_{fail}}{W}$ (où w_{fail} = le poids du dispositif en panne, W = poids total de tous les dispositifs) du total des données à ré-allouer aux nouveaux disques destinataires.

La situation est plus compliquée lorsque la hiérarchie du cluster est modifiée, avec l'ajout ou le départ des ressources de stockage c'est-à-dire lorsque le changement se produit au niveau des nœuds intermédiaires de la hiérarchie. Ce changement entraîne le changement du poids des sous arbres contenant ce nœud. Lorsque ce changement affecte la distribution, des données doivent être déplacées des sous-arbres avec un poids en baisse vers les sous-arbres avec un poids en augmentation.

L'équilibrage de charge passe, principalement, par trois grandes étapes. Une première étape qui découpe l'ensemble des ressources en sources (surchargées), destinations (sous-chargées) ou neutres (équilibrées). Une deuxième étape de sélection qui choisit les données à faire migrer à partir des ressources sources vers les ressources destinations. Et une troisième étape de correspondance qui détermine les paires (source,destination) convenables.

CRUSH, qui est utilisé par *Vespa* pour assurer le placement des données et l'équi-

librage de charge, assure la première et la deuxième étapes. Par contre la troisième étape (étape de correspondance sources destination) n'est pas encore optimisée.

Dans la suite nous allons traiter cette problématique et nous allons considérer les deux cas de transferts possibles : transfert à partir d'un seul disque source (où une buquette (unité de stockage regroupant un ensemble de fichiers) donnée doit être transférée en entier à partir d'un seul disque) et transfert à partir de plusieurs disques sources (où une buquette donnée peut être transférée en morceaux à partir de plusieurs disques sources).

4.4.3 Équilibrage de charge : correspondance sources-destinations

Le problème de correspondance entre les disques sources et les disques destinations lors du placement des données dans les systèmes de stockage, n'a pas été étudié dans la littérature malgré son importance, alors même que le problème de la re-configuration de systèmes et de l'équilibrage de charge a été largement étudié [31, 12, 95, 72]. En particulier, la complexité théorique du problème de re-configuration est inconnue.

Ce problème de correspondance est, en fait, un problème d'ordonnancement consistant à déterminer les répliques des données à partir desquelles le transfert va se faire pour générer la nouvelle copie. Le choix de la réplique et par conséquent le choix de disque source peut influencer la performance du système en provocant la surcharge de certains disques et ainsi un ralentissement du système. En effet, même si la quantité de données à transférer est fixée, la charge des différents nœuds de stockage et les liens de communication réseaux peuvent être très hétérogènes, ce qui peut dégrader considérablement la performance de système durant cette étape de re-configuration.

Par contre, ce problème d'ordonnancement est très proche du problème de "l'ordonnancement multi-processeurs" (Minimum Multiprocessor Scheduling) [8]. Si nous considérons les disques comme des processeurs et les buquettes à transférer comme des processus à migrer, notre problème peut être considéré comme une instance particulière de ce problème. Dans la littérature, beaucoup d'intérêt a été montré pour le problème de migration de processus dans le cadre de l'ordonnancement des tâches sur plusieurs processeurs. Dans [79], Nuttal donne un aperçu des avantages qui découlent de la migration des processus, et dans [50], les auteurs ont fait des simulations pour prouver l'importance pratique de la migration dans des situations réelles.

Tous ces auteurs, ont essayé de répondre à la question de savoir si l'utilisation d'un nouveau processeur ou d'un processeur moins chargé peut justifier le coût du transfert. Rodulph et al. ont proposé dans [86] de réduire le nombre de processus à migrer et de choisir parmi les processus à migrer un nombre limité. Dans [3], Aggarwal et al. aussi ont adopté la même technique et ont considéré la distance entre les processeurs comme paramètre de choix de migration. Hu et al. [57] proposent une technique de diffusion pour l'équilibrage de charge dans ce modèle.

Ces travaux, sont très liés à notre objectif, mais le problème consiste à faire un

choix parmi les données à transférer, alors que dans notre contexte, les données à transférer sont fournies par CRUSH et notre objectif est de déterminer les disques (parmi ceux qui contiennent une réplique) à partir desquels il faut transférer les données.

4.5 conclusion

Après cette brève présentation des caractéristiques des systèmes de fichiers distribués, nous allons considérer le problème de reconfiguration de système dans le chapitre suivant, qui se pose quand une nouvelle ressource de stockage est ajoutée au système. Dans ce cas, il est nécessaire de prévoir le transfert des données qu'il doit contenir, et donc de décider quelles répliques doivent être utilisées pour le transfert.

Chapitre 5

Optimisation du transfert des données

Comme nous l'avons vu dans le chapitre précédent, l'ensemble des fichiers à transférer est calculé par un mécanisme de placement externe (CRUSH). Notre but sera donc d'identifier les répliques qui doivent être choisies pour optimiser l'opération de transfert et minimiser son temps de traitement total (makespan). Dans ce chapitre, nous allons considérer ce problème (voir section 5.1).

Puis, dans 5.2 nous allons étudier la complexité de ce problème et proposer des solutions dans les deux cas du transfert fractionnaire et entier. La section 5.3 sera consacrée pour les résultats expérimentaux. Enfin, nous présentons une conclusion et des perspectives.

5.1 Modélisation du problème

Nous considérons le cas où un disque $\mathcal{D}$, ayant une bande passante entrante B, s'ajoute au système. Les fichiers à transférer sur ce disque sont regroupés dans des unités de stockage logique appelées "buquettes" (buckets). Cette organisation et ce regroupement des fichiers sous forme des buquettes permettent d'avoir des unités de stockage plus homogènes. Cette homogénéité des buquettes est assurée par un mécanisme externe. Mais pour des raisons de généralité, nous allons considérer que s_i désigne la taille de la buquette bu_i et noter N le nombre total des buquettes à transférer.

Chaque buquette bu_i est répliquée r_i fois dans le système, sur des disques différents. Dans le contexte de *Vespa*, ce nombre de répliques est constant pour toutes les buquettes (pourtant nous allons considérer un contexte plus général dans la suite). Soit n le nombre total de disques qui peuvent s'impliquer dans le transfert c'est-à-dire les disques qui possèdent au moins une copie d'un des fichiers à transférer sur le disque destinataire (*resp.* les disques destinataires). Nous posons $\delta_i^j = 1$ si le disque $\mathcal{D}_j$ possède une copie de la buquette bu_i et $\delta_i^j = 0$ sinon.

Chaque disque participant $\mathcal{D}_j$ est caractérisé par une bande passante entrante, notée B_j^{in}, et une sortante, notée B_j^{out}. La bande passante entrante (*resp.* bande passante sortante) regroupe, à la fois, la bande passante entrante de son interface réseau et sa vitesse d'écriture (*resp.* bande passante sortante et vitesse de lecture).

Un disque peut être impliqué dans plusieurs transferts en parallèle, pourvu que la bande passante totale des communications entrantes (*resp.* sortantes) ne dépasse pas sa capacité. Pour bien modéliser les interactions entre les communications, nous allons utiliser le modèle "multi-port". Ce modèle est proche de celui présenté par *Hong* et *Prasanna* dans [51] et reflète bien les capacités de réseaux modernes. Dans la première partie de cette thèse (voir chapitre 2), ce modèle a été présenté et sera discuté en détail dans le chapitre suivant qui sera consacré à l'étude de l'importance des mécanismes de partage de bande passante dans de telles plates-formes. En particulier, l'étude portera sur le problème étudié dans ce chapitre et celui étudié dans le chapitre 2.

D'autre part, nous considérons que la contention ne peut se produire qu'au niveau des interfaces des disques mais, par contre, que les capacités du réseau cœur sont généralement très importantes. Cette situation, en fait, s'applique bien sur la plate-forme de *Yahoo !* où les réseaux de communication sont sur-dimensionnés.

En respectant ces contraintes, deux types du transfert sont possibles. Le premier type correspond au cas le plus général où les buquettes peuvent être transférées en morceaux à partir de plusieurs disques. Le deuxième correspond au cas particulier où chaque buquette doit être transférée intégralement à partir d'un seul disque.

Cette contrainte supplémentaire d'intégrité des buquettes complexifie le problème et le rend NP-Complet. Une étude de la complexité de ce problème avec la preuve de NP-complétude sera menée plus loin dans de ce chapitre (voir 5.2). Par contre, le transfert fractionnaire des données est plus difficile à assurer au niveau de l'implémentation. Il demande une forte consistance des données, pour assurer que toutes les répliques sont identiques, qui n'est pas évidente à mettre en œuvre.

5.1.0.1 Cas du transfert fractionnaire

Comme nous venons de le préciser dans le paragraphe précédent, ce cas correspond au cas où les buquettes peuvent être transférées en fractions à partir de plusieurs disques. En respectant les différentes autres contraintes citées ci-dessus, la façon la plus générale de définir le problème d'optimisation correspondant au problème d'ordonnancement est donc la suivante.

$$\text{(Pb-Frac-NonConst)} \left\{ \begin{array}{lll} \textsc{Minimiser} \quad T_{max} & & \\ \forall t, i, j \quad f_{ij}(t) \leqslant \delta_i^j B_j^{out} & & (1) \\ \forall t, \quad \sum_i f_{ij}(t) \leqslant B_j^{out} & & (2) \\ \forall t, \quad \sum_j \sum_i f_{ij}(t) \leqslant \mathcal{B} & & (3) \\ \forall i, \quad \sum_j \int_{t=0}^{T_max} f_{ij}(t) = s_i & & (4) \end{array} \right\}$$

où

- $f_{ij}(t)$ représente la bande passante utilisée par le disque $\mathcal{D}_j$ au temps t pour transférer $\mathcal{B}_i$ et B la bande passante entrante de disque destinataire,

- la condition (1) montre que $f_{ij}(t)$ doit être plus petite que B_j^{out} et doit être nulle si $\mathcal{D}_j$ ne possède pas la buquette $\mathcal{B}_i$,

- les contraintes (2) et (3) assurent le respect des bandes passantes entrantes du disque destinataire D et sortantes des disques $\mathcal{D}_j$,

- la condition (4) assure que toutes les buquettes $\mathcal{B}_i$ sont transférées en T_{max} unités de temps.

L'implémentation des transferts de données basée sur la formulation ci-dessus peut être difficile. Notamment, le transfert d'une buquette donnée à partir de plusieurs sources peut être difficile, car dans le contexte de *Vespa*, les fichiers (et donc les buquettes) peuvent être mis à jour dynamiquement et donc assurer que toutes les répliques sont exactement les mêmes est impossible (même si un protocole de cohérence est utilisée pour s'assurer que les répliques d'un même fichier sont régulièrement mises à jour). Par conséquent, dans la formulation suivante, nous imposons que chaque buquette soit transférée exactement à partir d'un disque et nous montrons qu'il est possible d'assurer que la bande passante allouée au transfert d'une buquette donnée ne change pas au cours du temps (encore une fois pour des raisons d'implémentation) sans perte d'efficacité.

5.1.1 Cas du transfert entier

Dans ce contexte, chaque buquette $\mathcal{B}_i$ est associée à un disque $\mathcal{D}(\mathcal{B}_i)$, une date de début $t(\mathcal{B}_i)$ et une bande passante $b(\mathcal{B}_i)$ allouée au transfert. Ainsi, ce transfert prend place entre $t(\mathcal{B}_i)$ et $t(\mathcal{B}_i) + \frac{s_i}{b(\mathcal{B}_i)}$, et le problème d'optimisation devient, en notant par $[T_j = \{\mathcal{B}_i, \mathcal{D}(\mathcal{B}_i) = \mathcal{D}_j \text{ et } t(\mathcal{B}_i) \leqslant t \leqslant t(\mathcal{B}_i) + \frac{s_i}{b(\mathcal{B}_i)}\}]$ l'ensemble des buquettes à transférer à partir de $\mathcal{D}_j$ à l'instant t.

$$\text{(Pb-Ent-Const)} \left\{ \begin{array}{ll} \textsc{Minimiser} & \max_i (t(\mathcal{B}_i) + \dfrac{s_i}{b(\mathcal{B}_i)}) \\ \forall t, & \displaystyle\sum_{i \in T_j(t)} b(\mathcal{B}_i) \leqslant B_j^{out} \\ \forall t, & \displaystyle\sum_{j} \sum_{i \in T_j(t)} b(\mathcal{B}_i) \leqslant B \end{array} \right\}.$$

Après avoir modélisé les deux problèmes "fractionnaire" et "entier", l'étape suivante sera d'étudier la complexité de ces problèmes.

5.2 Résultats de complexité

5.2.1 Cas du transfert fractionnaire

Dans le cas général, quand une buquette donnée peut être transférée à partir de plusieurs disques et quand la bande passante allouée à un tel transfert peut varier dans le temps, optimiser le temps de transfert total correspond à la résolution du problème d'optimisation suivant.

$$\text{(Pb-Frac-NonConst)} \left\{ \begin{array}{lll} \text{MINIMISER} \quad T_{max} & & \\ \forall t, i, j \quad f_{ij}(t) \leqslant \delta_i^j B_j^{out} & & (1) \\ \forall t, \quad \sum_i f_{ij}(t) \leqslant B_j^{out} & & (2) \\ \forall t, \quad \sum_j \sum_i f_{ij}(t) \leqslant B & & (3) \\ \forall i, \quad \sum_j \int_{t=0}^{T_max} f_{ij}(t) = s_i & & (4) \end{array} \right\}.$$

Même si cette formulation paraît compliquée, on peut, en fait, restreindre la recherche à des ordonnancements réguliers (où le transfert des buquettes ne change pas au cours du temps). Pour prouver ce résultat, nous allons considérer le programme linéaire suivant, où la bande passante allouée au transfert de la buquette $\mathcal{B}_i$ à partir du disque $\mathcal{D}_j$ est notée par x_{ij} et où l'objectif est de maximiser la fraction f de chaque buquette $\mathcal{B}_i$ transférée par unité de temps.

$$\text{(Pb-Frac-1)} \left\{ \begin{array}{l} \text{MAXIMISER} \quad f \\ \forall j, \quad \sum_i x_{ij} \leqslant B_j^{out} \\ \forall t, \quad \sum_i \sum_j x_{ij} \leqslant B \\ \forall i, j, \quad x_{ij} \leqslant \delta_i^j B_j^{out} \\ \forall i, \quad \sum_j x_{ij} = s_i f \end{array} \right\}$$

Notons par f_{opt} la solution optimale de Pb-Frac-1 et T_{opt} une solution optimale de Pb-Frac-NonConst .

Théorème 5.1 *Soit $T_{opt} = \frac{1}{f_{opt}}$, notons par x_{ij}^{opt} la bande passante allouée au transfert de $\mathcal{B}_i$ à partir de $\mathcal{D}_j$ dans la solution optimale de Pb-Frac-1 , alors*

$$f_{ij}(t) = x_{ij}, \quad \forall t \in [0, \frac{1}{f_{opt}}]$$

est une solution optimale de Pb-Frac-NonConst .

Preuve. Prouvons que $f_{opt} \geqslant \frac{1}{T_{opt}}$ et considérons une solution optimale de Pb-Frac-NonConst . Si on pose

$$x_{ij} = \frac{\int_0^{T_{opt}} f_{ij}(t)dt}{T_{opt}},$$

alors

$$\forall t, \quad f_{ij}(t) \leqslant \delta_i^j B_j^{out},$$

$$\int_0^{T_{opt}} f_{ij}(t)dt \leqslant \delta_i^j B_j^{out} T_{opt}$$

et donc

$$x_{ij} \leqslant \delta_i^j B_j^{out}.$$

Les inégalités (2) et (3) peuvent être prouvées en utilisant la même technique. En outre, on a

$$\forall i, \quad \sum_j \int_0^{T_{opt}} f_{ij}(t) = s_i,$$

$$\Rightarrow \forall i, \quad \sum_j x_{ij} T_{opt} = s_i,$$

$$\Rightarrow \forall i, \quad \sum_j x_{ij} = \frac{s_i}{T_{opt}}.$$

Donc $(x_{ij}, \frac{1}{T_{opt}})$ est une solution valide de Pb-Frac-1 , ce qui termine la preuve.
Prouvons maintenant que $T_{opt} \leqslant \frac{1}{f_{opt}}$.
Considérons une solution optimale de Pb-Frac-1 et posons :

$$\forall i, j \quad 0 \leqslant t \leqslant T_{opt} \quad \left\{ \begin{array}{l} f_{ij}(t) = x_{ij} \\ f_{ij}(t) = 0 \quad \text{SINON} \end{array} \right\}$$

alors, les conditions (1), (2) and (3) sont vérifiées et puisque

$$\forall i, \quad \sum_j x_{ij} = f_{opt} s_i,$$

$$\text{et alors} \quad \forall i, \quad \sum_j \int_0^{\frac{1}{f_{opt}}} f_{ij}(t)dt = f_{opt} s_i.$$

Ainsi $(f_{ij}(t), \frac{1}{f_{opt}})$ est une solution valide de Pb-Frac-NonConst et $T_{opt} \leqslant \frac{1}{f_{opt}}$, ce qui termine la preuve du théorème.

Par conséquent, Pb-Frac-NonConst peut être résolu en temps polynomial en utilisant un programme linéaire en nombres rationnels auxiliaire Pb-Frac-1 et il existe un ordonnancement optimal qui est régulier (c'est-à-dire dans lequel tous les transferts commencent au temps 0 et s'arrêtent au temps T_{opt} et où la bande passante allouée à un transfert donné ne change pas au cours du temps).

5.2.2 Cas du transfert entier

Pour étudier la complexité de Pb-Ent-Const nous allons considérer seulement le cas où B $\geqslant \sum_j B_j^{out}$, c'est-à-dire quand la bande passante entrante de $\mathcal{D}$ ne peut pas être saturée par les communications sortantes des autres disques. Dans ce cas, Pb-Ent-Const devient

$$\text{(Pb-Ent-Const)} \left\{ \begin{array}{ll} \text{MINIMISER} & \max_i(t(\mathcal{B}_i) + \frac{s_i}{b(\mathcal{B}_i)}) \\ \forall t, & \sum_{i \in T_j(t)} b(\mathcal{B}_i) \leqslant B_j^{out} \end{array} \right\},$$

où $T_j(t) = \{\mathcal{B}_i, D(\mathcal{B}_i) = \mathcal{D}_j \text{ et } t(\mathcal{B}_i) \leqslant t \leqslant t(\mathcal{B}_i) + \frac{t(s_i)}{b(\mathcal{B}_i)}\}$.

Notons $S_j = \{\mathcal{B}_i, D(\mathcal{B}_i) = \mathcal{D}_j\}$. La quantité de données totale transférée de $\mathcal{D}_j$ est donnée par $\sum_{i \in S_j} s_i$. Par conséquent, le temps de transfert ne peut pas être inférieur $\sum_{i \in S_j} \frac{s_i}{B_j^{out}}$ et d'autre part, l'ordre des communications n'a pas d'importance, puisque la bande passante de $\mathcal{D}$ ne peut pas être saturé) et que $b(\mathcal{B}_i) = B_j^{out}$.

En conséquence, dans ce cas, la question est de déterminer quelles buquettes doivent être transférées à partir de chaque disque (sélection de ressources), et non quand elles doivent être transférées (ordonnancement).

Dans le cas où $B \geqslant B_j^{out}$, Pb-Ent-Const devient Pb-Ent-LargeB .

- **I**nstance : Un ensemble $\mathcal{B}_i$ de buquettes de taille s_i, n disques $\{\mathcal{D}_j\}$ de bandes passantes sortantes B_j^{out} (transférer $\mathcal{B}_i$ de $\mathcal{D}_j$ prend $\frac{s_i \delta_i^j}{B_j^{out}}$, où δ_i^j indique si une réplique de $\mathcal{B}_i$ est stockée sur $\mathcal{D}_j$) et une borne B.
- **S**olution : Une fonction $f[1,n] \to [1,n]$ telle que $\max_{j \in [1,n]} \frac{\sum_{i, f(i)=j} \mathcal{B}_i}{B_j^{out}} \leqslant B$.

Le problème ci-dessus est clairement NP-Complet. Par réduction au problème de 3-Partition, avec n disques homogènes ($B_j^{out} = 1 \quad \forall j$), $3n$ fichiers (buquettes) de tailles a_i répliqués sur tous les disques, et une interrogation sur la possibilité de réaliser le transfert en K unités de temps, où $(a_i \dots a_{3n})$ est une instance de 3-Partition (c'est-à-dire $\sum a_i = n.K$ et $\frac{K}{4} < a_i < \frac{K}{2}, \quad \forall i$).

Le problème reste NP-complet même si le nombre de disques (et le nombre de répliques) reste borné. La réduction au problème de 2-Partition est également triviale, en considérant n buquettes de tailles a_i répliquées sur deux disques homogènes ($B_j^{out} = 1$) et une interrogation sur la possibilité de réaliser le transfert en K unités de temps, où $\sum a_i = 2K$.

Pb-Ent-LargeB est plus général que le problème d'ordonnancement "Minimum Multiprocessor Scheduling With Related Resources", puisque dans notre cas, certaines buquettes (resp tâches) ne peuvent pas être transférées à partir de (resp. exécutées sur) certains disques (resp. processeurs). D'un autre coté, Pb-Ent-LargeB est une instance spéciale du problème d'ordonnancement "Minimum Multiprocessor Scheduling" [7] où le temps pour exécuter une tâche t sur le processeur j est donnée par $l(t, j)$ (au lieu de $\delta_i ts_i/b_j$ dans notre cas). Par conséquent, la solution de Pb-Ent-LargeB peut être approximée à un facteur 2 [68]. Par contre, il n'est pas clair que ce facteur d'approximation reste vrai lorsque la bande passante entrante de $\mathcal{D}$ ne satisfait pas la condition $\sum_j B_j^{out} \leqslant$B.

5.2.3 Facteur d'approximation de Pb-Ent-Const

5.2.3.1 Algorithme d'approximation basé sur la programmation linéaire

Nous allons proposer un algorithme d'approximation pour résoudre Pb-Ent-Const . Cet algorithme va se baser sur une analyse détaillée de la solution Pb-Frac-1 qui a été introduite dans la section 5.2.1.

Le principe, en fait, est que dans Pb-Frac-NonConst une buquette donnée peut être transférée à partir de plusieurs disques. Ainsi, la valeur objective de Pb-Frac-NonConst (qui est son makespan) correspond au temps du transfert du dernier morceau, de la dernière buquette. Par contre, dans Pb-Ent-Const , ce temps correspond au temps du transfert de la dernière buquette entière. Comme les transferts peuvent se faire en parallèle (modèle multi-port), la valeur objective de Pb-Frac-NonConst est donc plus petite que la valeur objective de Pb-Ent-Const . L'idée sera d'essayer de compléter l'analyse de ce temps de transfert via le programme linéaire auxiliaire Pb-Frac-1 .

Pour faciliter l'analyse détaillée de la solution optimale de Pb-Frac-1 , nous avons modifié un peu sa formulation en introduisant l'ensemble $S = \{(i, j) \quad / \quad \delta_i^j = 1\}$ et nous traitons seulement les variables x_{ij} potentiellement non nulles (c'est-à-dire

tel que $(i,j) \in S$).

$$\text{(Pb-Frac-1)} \left\{ \begin{array}{ll} \text{MAXIMISER} \quad f & \\ \forall (i,j), \quad 0 \leqslant x_{ij} & (1) \\ \forall j, \quad \sum_{i,(i,j)\in S} x_{ij}(t) \leqslant B_j^{out} & (2) \\ \sum_{i,j\in S} x_{ij}(t) \leqslant B & (3) \\ \forall i, \quad \sum_{i,j\in S} x_{ij}(t) = s_i f & (4) \end{array} \right\}$$

Ce programme linéaire est clairement équivalent au précédent. Les variables x_{ij}, où $\delta_i^j = 0$, ont été contraintes à être nulles à cause de la contrainte $0 \leqslant x_{ij} \leqslant \delta_i^j B_j^{out}$ et la contrainte $x_{ij} \leqslant \delta_i^j B_j^{out}$ est automatiquement satisfaite par la contrainte (2) si $\delta_i^j = 1$.
Dans le programme linéaire ci-dessus, il y a $|S| + 1$ variables (les x_{ij} et f) et $n + 1 + |S| + m$ contraintes, parmi lesquelles n sont des équations).
D'après la théorie de la programmation linaire [78, 91], on sait qu'il existe une solution en un sommet du polyèdre défini par les contraintes linéaires. En ce sommet, il existe au moins $|S| + 1$ contraintes qui sont des égalités. Vu qu'il existe $n + m + 1$ contraintes de type (2), (3) et (4), alors au plus $m + n + 1$ parmi elles sont des égalités. Par conséquent, au moins $|S| + 1 - m - n - 1$ contraintes de type (1) sont des égalités, et donc il existe, au plus, $|S| - (|S| - m - n) = m + n$ variables x_{ij} qui sont strictement positives. En outre, puisque $\forall i$, $\sum_{(i,j)\in S} x_{ij} = s_i f$ et $s_i f > 0$ (sauf dans le cas où $f_{opt} = 0$), il existe, $\forall i$, au moins une des variables x_{ij} qui est positive. Par conséquent, il existe, au plus, m buquettes $\mathcal{B}_i$ tel que $\exists j_1, j_2 / \quad x_{ij_1} > 0$ et $x_{ij_2} > 0$, et donc au moins $n - m$ buquettes sont complètement transférées à partir d'un seul disque.

Cette remarque est particulièrement importante dans le contexte de *Vespa*. En effet, dans ce contexte, le nombre de buquettes (n) est beaucoup plus grand que le nombre des disques (m). En conséquence, la solution de ce programme linéaire fournit un ordonnancement où la plus grande partie des buquettes ($n - m$ parmi n) sont transférées à partir d'un seul disque. En plus, nous pouvons nous appuyer sur la technique développée dans [68] pour prouver le lemme suivant

Lemme 5.2 *Soit $\mathcal{B} = \{\mathcal{B}_{i1} \dots \mathcal{B}_{ik}\}$ l'ensemble des buquettes transmises à partir de plusieurs disques dans la solution du programme linéaire et $\mathcal{D} = \{D_{j1} \dots D_{jm}\}$ l'ensemble des disques utilisés pour la transmission de ces buquettes. Il existe une parfaite correspondance entre $\mathcal{B}$ et $\mathcal{D}$ dans le graphe biparti G_B où les arêtes sont associées aux couples (buquette, disque) tel que $x_{ij} > 0$ dans la solution du programme linéaire.*

En particulier, il est possible d'organiser le transfert des buquettes restantes (celles qui n'ont pas été transférées à partir d'un seul disque dans la solution fractionnelle optimale) de manière à ce que chaque disque transfère au plus une buquette.
L'algorithme Algo-Transf-Ent qu'on propose pour transférer toutes les buquettes est :

Algo-Transf-Ent :

- Résoudre le programme linéaire et transférer toutes les buquettes qui sont associées à un seul disque dans la solution du programme linéaire LP.
- Construire un graphe biparti G_B et trouver une correspondance parfaite (perfect matching) entre B et D.
- Transférer toutes les buquettes restantes en utilisant cette correspondance.

5.2.4 Calcul du facteur d'approximation

Nous allons commencer par énoncer le théorème permettant de fournir un facteur d'approximation sur le temps nécessaire pour transférer toutes les buquettes (que nous prouverons ensuite).

Théorème 5.3 *Algo-Transf-Ent fournit une solution qui transfère toutes les buquettes en temps T, tel que*

$$1 \leqslant \frac{T}{T_{opt}} \leqslant 1 + \frac{m}{n}(\frac{s_{max}}{s_{mean}})(\frac{b_{mean}}{b_{min}}),$$

où

- n est le nombre total des buquettes,
- m est le nombre des disques impliqués dans le transfert,
- $s_{max} = \max_i s_i \quad i \leqslant n$,
- $s_{mean} = \dfrac{\sum_{i=1}^{n} s_i}{n}$,
- $b_{mean} = \dfrac{\sum_{j=1}^{m} B_j^{out}}{m}$,
- $b_{min} = \min_{j \leqslant n} \quad B_j^{out}$
- T_{opt} indique le makespan optimal pour Pb-Ent-Const .

La preuve se décompose en deux cas, selon la bande passante utilisée au niveau du disque destinataire dans la solution du programme linéaire.

Cas1 :

La bande passante utilisée au nœud destinataire dans la solution du programme linéaire est plus petite que B.

- Dans ce cas, tous les disques D_j qui transfèrent partiellement une buquette $\mathcal{B}_i$ dans la solution du programme linéaire utilisent leur bande passante sortante B_j^{out} maximale. En effet, si ce n'est pas le cas, la bande passante allouée peut être augmentée pour transférer B_i (et celle allouée aux autres disques transférant B_i peut être diminuée).

- Ce processus s'arrête lorsque la bande passante totale au nœud destinataire devient >B (et dans ce cas on passe au cas 2), soit quand $\mathcal{B}_i$ est complètement transférée à partir de D_j (et dans ce cas, on peut assurer la même opération avec toute autre buquette transférée partiellement), soit quand la bande passante utilisée en $\mathcal{B}_i$ devient B_j^{out} (et dans ce cas, on peut assurer la même opération avec toute autre buquette transférée partiellement).
  En conséquence, si b_j^{out} représente la bande passante effectivement utilisée par $\mathcal{D}_j$ dans la solution du programme linéaire, alors $\forall j$ tel que $\mathcal{D}_j$ transfère partiellement au moins une buquette, alors

$$b_j^{out} = B_j^{out}$$

  et

$$\sum_{j \in \mathcal{P}} B_j^{out} < B,$$

  où $\mathcal{P}$ représente l'ensemble des indices j tel que D_j transfère partiellement au moins une buquette.
- Dans ce cas, le fonctionnement de l'algorithme est comme suit. Pendant T_{opt}^{LP}, toutes les buquettes transférées complètement à partir d'un seul disque sont effectivement transférées.
  Au temps T_{opt}^{LP}, le transfert des buquettes transférées partiellement commence. Chaque disque participant transfère exactement une buquette, avec sa bande passante maximale. Le transfert de la buquette $\mathcal{B}_i$ à partir de $\mathcal{D}_j$ prend ainsi $\frac{s_i}{B_j^{out}} \leqslant \frac{s_{max}}{b_{min}}$.

Le temps d'exécution totale T_{alg} est par conséquent

$$T_{opt}^{LP} \leqslant T_{alg} \leqslant T_{opt}^{LP} + \frac{s_{max}}{b_{min}}.$$

et on a donc

$$1 \leqslant \frac{T_{alg}}{T_{opt}^{LP}} \leqslant 1 + \frac{1}{T_{opt}} \frac{s_{max}}{b_{min}}, \tag{5.1}$$

$$\frac{T_{alg}}{T_{opt}^{LP}} \leqslant 1 + \frac{1}{T_{opt}^{LP}} \frac{s_{max}}{b_{min}}. \tag{5.2}$$

$$\text{De plus, comme} \quad T_{opt}^{LP} \geqslant \frac{\sum s_i}{\sum B_j^{out}}, \tag{5.3}$$

$$1 \leqslant \frac{T_{alg}}{T_{opt}} \leqslant 1 + \frac{mb_{mean}}{ns_{mean}} \frac{s_{max}}{b_{min}} \tag{5.4}$$

$$\text{et finalement} \frac{T_{alg}}{T_{opt}} \leqslant 1 + (\frac{m}{n})(\frac{s_{max}}{s_{mean}})(\frac{b_{mean}}{b_{min}}). \tag{5.5}$$

Cas2 :

La bande passante totale utilisée au niveau du disque destinataire dans la solution du programme linéaire est égale à B.
Dans ce cas, nous commençons par le transfert des buquettes transférées partiellement au temps T_{opt}^{LP}. Soit b_j^{out} la bande passante utilisée pour transférer une buquette à partir de $\mathcal{D}_j$ (c'est-à-dire la bande passante allouée à $\mathcal{D}_j$ dans la solution du programme linéaire).
Par la suite, on peut toujours choisir les bandes passantes utilisées pour transférer les buquettes restantes de telle façon que le transfert qui termine le dernier se déroule en utilisant la bande passante maximale de ce disque source, ou que tous les transferts des buquettes restantes s'arrêtent en même temps. En outre, considérons qu'il existe un ordonnancement optimal des buquettes restantes tel que le dernier transfert de la dernière buquette n'utilise pas toute la bande passante du disque source. Par la suite, s'il y a un transfert qui termine plus tôt, il est possible d'augmenter la bande passante allouée au dernier transfert (en décroissant la bande passante allouée au transfert terminant tôt).
De plus, si la bande passante utilisée pour transférer la dernière buquette $\mathcal{B}_i$ transmise est B_j^{out} pour un disque donné $\mathcal{D}_j$, alors la preuve du facteur d'approximation du "cas1" reste vraie. D'autre part, si tous les transferts des buquettes restantes se terminent au même moment, alors $T_{alg} \leqslant T_{opt}^{LP} + \frac{m.s_{max}}{B}$ où $\frac{m.s_{max}}{B}$ représente le temps maximal pour transférer les m buquettes restantes avec toute la bande passante B et ainsi

$$\frac{T_{alg}}{T_{opt}} \leqslant 1 + \frac{m.s_{max}}{B} \frac{B}{\sum s_i}.$$

En effet, si

$$\left\{ \begin{array}{c} T_{opt} \geqslant \frac{B}{\sum s_i} \\ et \\ \frac{T_{alg}}{T_{opt}} \leqslant 1 + (\frac{m}{n})(\frac{s_{max}}{s_{mean}}) \end{array} \right\}, \text{ alors } \quad \frac{T_{alg}}{T_{opt}} \leqslant 1 + (\frac{m}{n})(\frac{s_{max}}{s_{mean}})(\frac{b_{mean}}{s_{min}}).$$

En conséquence, dans tous les cas,

$$\frac{T_{alg}}{T_{opt}} \leqslant 1 + (\frac{m}{n})(\frac{s_{max}}{s_{mean}})(\frac{b_{mean}}{s_{min}}).$$

Dans le cas de *Vespa*, le facteur d'approximation ci-dessus est en fait tout petit. En effet, dans la majorité des cas $\frac{m}{n}$ est $\leqslant \frac{1}{100}$ et $\frac{s_{max}}{s_{mean}}$ et $\frac{b_{mean}}{b_{min}}$ sont de l'ordre de 2 ou 3, de sorte que la facteur d'approximation est plus petit que 1.1.

5.3 Résultats expérimentaux

5.3.1 Paramètres de simulation

Toutes les simulation présentées dans cette section ont été réalisées en utilisant SIMGRID [92]. Nous comparons les différentes heuristiques en utilisant les paramètres suivants :

Hétérogénéité des buquettes : Les buquettes sont des unités de stockage qui regroupent un ensemble des fichiers de telle sorte qu'elles sont à peu près homogènes. Ainsi, les simulations ont été réalisées en utilisant des buquettes homogènes (de même taille) et des buquettes légèrement hétérogènes (facteur 3).

Rapport entre le nombre des buquettes et le nombre de disques : Dans toutes les simulations, le nombre de disques qui contiennent au moins une réplique est fixé à 50 ; Par contre, nous considérons deux cas différents pour le nombre des buquettes : un nombre relativement petit comparé au nombre de disques, à savoir 100, et un nombre plus grand, à savoir 1000.

Hétérogénéité de la bande passante sortante : Le facteur entre la bande passante maximale et la bande passante minimale sortante d'un disque est 1, 3 ou 10.

Facteur entre la bande passante entrante totale et la bande passante sortante : La bande passante entrante en un nœud est généralement plus petite que la bande passante sortante, puisqu'elle comprend l'opération d'écriture sur le disque. Par contre, le nœud destinataire utilise toute sa bande passante pour l'opération de reconstruction, alors que les autres nœuds dédient seulement une (petite) fraction de leur bande passante à l'opération de reconstruction. Par conséquent, dans toutes les équations ci-dessus, B_j^{out} ne représente pas la bande passante sortante de $\mathcal{D}_j$ mais la bande passante sortante maximale de ce disque qui peut être utilisée pour la reconstruction. En effet, pendant la reconstruction, les disques $\mathcal{D}_j$ ont besoin de continuer à exécuter les requêtes usuelles. En conséquence, nous avons considéré trois cas différents selon la valeur du facteur entre B et $\sum B_j^{out}$.

- Cas 1, $B = \sum B_j^{out}$. Dans ce cas, on n'aura pas de limitation au niveau de l'utilisation des bandes passantes.
- Cas 2, $B = \frac{1}{3} \sum B_j^{out}$. Dans ce cas, il y aura une contention au niveau du nœud destinataire.
- Cas 3, $B = \frac{1}{10} \sum B_j^{out}$. Dans ce cas, il y aura une forte contention au niveau du nœud destinataire.

Enfin, chaque résultat de la simulation est obtenu à partir de 10 simulations avec les mêmes caractéristiques pour les paramètres. Sur toutes les figures, la valeur moyenne et l'écart type sont présentés.

5.3.2 Les heuristiques

Tout au long des simulations, nous comparons les cinq heuristiques suivantes :

PL. Pour cette heuristique, une fraction des buquettes peut être transféré à partir de chaque disque et la bande passante utilisée par le disque $\mathcal{D}_j$ pour transférer la buquette $\mathcal{B}_i$ est donné par x_{ij}, où les x_{ij} sont les solutions du programme linéaire Pb-Frac-1 .

PL adapté (PL-ad). En SIMGRID, un mécanisme sophistiqué pour modéliser les communications TCP et le partage de bande passante a été implémenté, sur lequel nous reviendrons au chapitre suivant. Pour toutes les heuristiques, le simulateur prend comme donnée une liste ordonnée des buquettes à transférer au disque destinataire. Il est possible que la bande passante b_j^{out} allouée au disque $\mathcal{D}_j$ dans la solution du programme linéaire soit plus petite que B_j^{out}, en particulier quand $B < \sum B_j^{out}$. Dans ce cas, la bande passante allouée à $\mathcal{D}_j$ est une fonction complexe vu que la bande passante allouée à certains disques peut être très élevée au début de leurs transferts. Ainsi, dès que ces disques ont transféré toutes leurs buquettes, la bande passante sortante des disques restants peut être plus petite que B. Par conséquent, le résultat de la simulation dans ce cas peut être très différent de ce qu'on attend théoriquement puisque les contraintes n'ont pas été respectées. Pour contourner ce problème, nous avons exécuté les simulations sous une plate-forme "adaptée", où B_j^{out} est forcée à être égale à b_j^{out}. Ceci correspond à une introduction du mécanisme de contrôle de la qualité de service pour fixer la bande passante sortante de tous les disques. Il est important de noter ici, que dans le contexte de *Vespa*, ce mécanisme est nécessaire dans tous les cas, puisque comme nous l'avons vu, la bande passante allouée aux transferts à partir de $\mathcal{D}_j$ est une fraction fixée de la bande passante totale.

Cette heuristique, qui permet de transférer les buquettes d'une manière fractionnaire à partir de plusieurs disques, réalise toujours le meilleur résultat et est utilisée comme référence pour la comparaison avec les autres heuristiques.

PL entier (LPH). Cette heuristique correspond à l'algorithme d'approximation décrit dans la section 5.2.3. En utilisant cet algorithme, chaque buquette est transférée exactement à partir d'un seul disque.

PL entier adapté (LPH-ad). Cette heuristique est identique à LPH mais en adaptant la plate-forme comme pour PL-ad.

Glouton. C'est un algorithme glouton qui prend une liste triée selon la taille des buquettes. Il transfère chaque buquette à partir du disque la contenant et qui est le moins chargé.

5.3.3 Présentation des résultats

Comme il a été expliqué dans la section précédente, il y a quatre paramètres. Pour chaque figure, on fixe trois de ces paramètres et on fait varier le quatrième. Par conséquent, dans ce contexte nous ne pouvons pas présenter toutes les figures, et nous avons choisi d'en décrire précisément 3. La valeur des paramètres dans ces figures est détaillée dans le tableau suivant.

Dans les trois figures suivantes, le nombre des buquettes est $n = 100$ et $n = 1000$, par contre l'hétérogénéité des buquettes et l'hétérogénéité des bandes passantes sont fixées.

- Hétérogénéité de la taille des buquettes : le facteur entre les tailles maximale et minimale des buquettes est 3.
- Hétérogénéité des bandes passantes sortantes des nœuds sources : le facteur entre les bandes passantes sortantes maximale et minimale est 2.

Enfin, la bande passante entrante du disque destinataire diffère dans chaque figure selon le tableau suivant.

Figures	Figure 5.1	Figure 5.2	Figure 5.3
Bande passante entrante de disque destinataire	$\sum B_j^{out}$	$\frac{\sum B_j^{out}}{3}$	$\frac{\sum B_j^{out}}{10}$

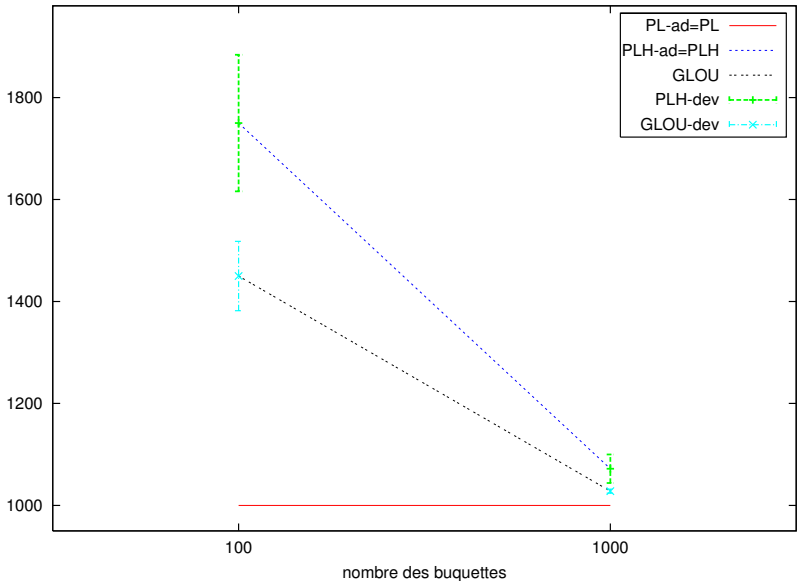

FIGURE 5.1: $B = \sum B_j^{out}$

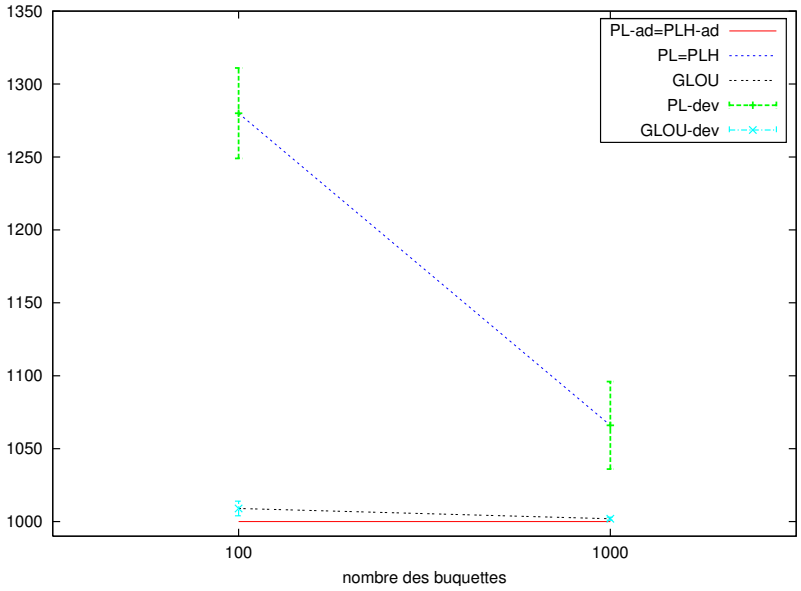

FIGURE 5.2: $B = \frac{\sum B_j^{out}}{3}$

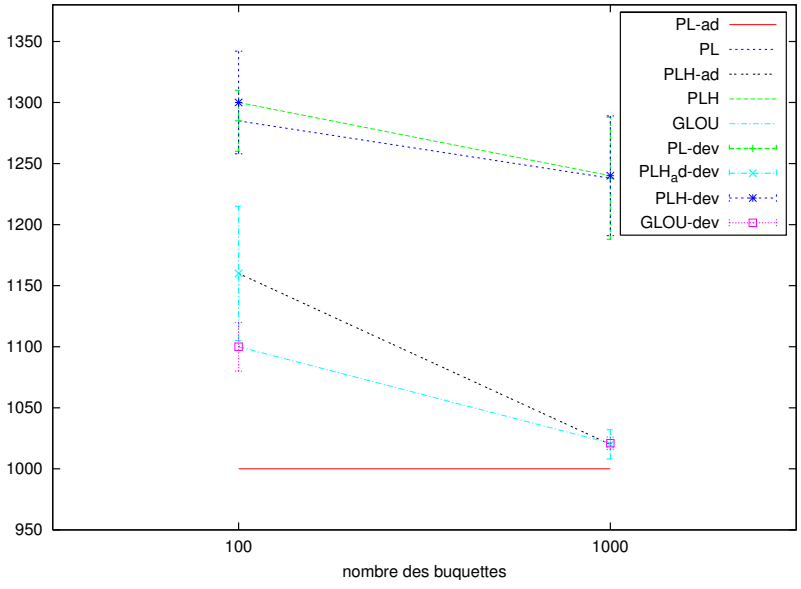

FIGURE 5.3: $B = \frac{\sum B_j^{out}}{10}$

5.3.4 Analyse des résultats

5.3.4.1 Remarques générales

Tout d'abord, il est important de noter que dans les conditions de nos expériences, toutes les heuristiques obtiennent de bons résultats. Dans le cas où le nombre des buquettes est 1000, le facteur entre le temps d'exécution total et la borne inférieure (programme linéaire) est plus petit que 1.1. Dans les cas où le nombre des buquettes est proche de celui des disques (100 vs 50), le facteur peut être proche

de 2, mais dans ce cas la borne inférieure (où une buquette peut être transférée de plusieurs disques sources) est sans doute très optimiste.

5.3.4.2 Influence de l'adaptation de la plate-forme

La remarque la plus importante à propos des résultats expérimentaux est liée à l'adaptation de la plate-forme. Puisque la bande passante maximale utilisée dans le programme linéaire peut être beaucoup plus petite que la bande passante qu'on peut allouer, on exécute une des heuristiques basée sur le programme linéaire sur la plate-forme initiale et une sur la plate-forme adaptée où les bandes passantes sont celles calculées par le programme linéaire Pb-Frac-1 .

Dans le cas où les bandes passantes originales sont utilisées, les disques avec les bandes passantes élevées peuvent finir leur transfert rapidement (en utilisant toutes la bande passante disponible). Par la suite, les disques ayant des petites bandes passantes commencent leur transfert plus tard et peuvent ainsi utiliser toute la bande passante disponible au nœud destinataire (mais la somme de leurs bandes passantes sortantes B_j^{out} peut être beaucoup plus petite que B).

Cette situation ne se produit pas quand B = $\frac{1}{10}\sum B_j^{out}$ (voir Figure 5.3) puisque dans ce cas, même un petit sous ensemble de disques est en effet capable d'utiliser toute la bande passante disponible du nœud.

Par contre, cette situation se produit quand B = $\frac{\sum B_j^{out}}{3}$. Dans ce cas, (Figure 5.2), il est important de noter que les solutions fractionnaires du programme linéaire exécutent moins de tâches que les solutions entières (PLH-ad et Glouton), même dans le cas où le nombre des buquettes est petit (Figure 5.1) Par conséquent, pour bien partager la bande passante entre les différents transferts, il est intéressant de limiter la bande passante allouée à chaque transfert.

5.3.4.3 Influence du facteur entre la bande passante entrante et la bande passante sortante

Quand la bande passante entrante au nœud destinataire est grande (c'est-à-dire $B = \sum B_j^{out}$), l'algorithme d'approximation produit des mauvais résultats en comparaison avec Glouton, particulièrement quand le nombre des buquettes est petit par rapport au nombre de disques. De plus, les buquettes qui sont transférées à partir de plusieurs disques sont les plus grandes et Glouton prend avantage du tri des buquettes par taille décroissante. Par contre, dans le cas de l'algorithme d'approximation PLH-ad, les plus grandes buquettes sont transmises en dernier !

Néanmoins, comme déjà mentionné, la bande passante entrante B est beaucoup plus petite que $\sum B_j^{out}$ en pratique. Dans notre ensemble d'expériences, le plus grand facteur qu'on considère est 10, mais la valeur réelle peut être plus grande. D'autre part, un facteur 10 est suffisant pour faire l'observation suivante : quand $B = \frac{\sum B_j^{out}}{10}$, l'algorithme d'approximation qu'on propose est, en fait, optimal, quelque soit le nombre des buquettes.

Ce phénomène peut être expliqué en analysant le programme linéaire "Pb2-lin". Dans le cas où $B \ll \sum B_j^{out}$, aucun disque n'utilise toute sa bande passante disponible, donc $b_j^{out} < B_j^{out}$ dans la solution optimale du programme linéaire. Puisqu'il

existe au plus n variables x_{ij} strictement positives, et puisque $\forall i, \exists j / x_{ij} > 0$, alors $\forall i, \exists! j$ tel que $x_{ij} = 1$. Ceci signifie que dans la solution factionnaire optimale, toutes les buquettes sont en fait transférées à partir d'une seule source !

5.3.5 Conclusion

Ce chapitre est divisé en deux sous-parties, une première sous-partie théorique avec un ensemble de preuves et une deuxième pratique avec un ensemble de simulations.

1. Le problème de reconstruction d'un disque que nous traitons ici, peut être formulé comme un programme linéaire dans le cas général du transfert des buquettes en fractions. À partir de cette formulation, une solution valide pour le cas du transfert entier de chaque buquette à partir d'un seul disque, est dérivée. En outre, le facteur d'approximation de cette solution est proche de 1 dans le cadre de *Vespa* où le nombre des buquettes transférées est beaucoup plus important que le nombre des disques participants dans le transfert et où l'hétérogénéité des bandes passantes et de la taille des buquettes est faible.
2. À l'aide de simulations réalisées en utilisant SIMGRID, nous avons montré que l'algorithme proposé, est aussi optimal en pratique, quand la bande passante entrante du disque destinataire est beaucoup plus petite (dans les simulations le facteur 10 était suffisant) que la somme des bandes passantes sortantes des disques sources. En pratique, cette hypothèse est satisfaite dans les conditions d'utilisation de *Vespa*.

   Un autre résultat très important que nous avons pu affirmer à l'aide de ces simulations est qu'il est fondamental d'utiliser des mécanismes de QoS pour adapter la plate-forme et contrôler les bandes passantes allouées aux différents transferts. Cette approche permet, en outre, d'utiliser beaucoup moins de bande passante qu'avec les approches gloutonnes habituelles. L'étude précise de l'influence des mécanismes de contrôles de bandes passantes sera l'objet du chapitre suivant.

Troisième partie

Sur l'importance des mécanismes de contrôle de bande passante

Chapitre 6

Sur l'importance des mécanismes de contrôle de bande passante

6.1 Introduction

Dans ce chapitre, nous considérons trois problèmes différents pour prouver l'importance des mécanismes de contrôle de bande passante et évaluer leur influence sur la performance des algorithmes d'ordonnancement sur des plates-formes hétérogènes.

Le premier problème d'ordonnancement que nous considérons est le problème qui a été introduit et traité dans le chapitre précédent (voir Chapitre 5) et qui se situe dans le contexte des systèmes de stockage à large échelle tel que *Vespa* [9], développée par *Yahoo !*, quand une re-configuration du système prend place. Dans le contexte de *Vespa*, le système de stockage réplique les données pour tolérer des pannes ou échecs du système et le placement des données est assuré par un mécanisme externe basé sur CRUSH [99]. Dans la suite nous allons considérer le cas où un disque est ajouté au système.

Le deuxième problème que nous cosidérons est le problème d'ordonnancement des tâches indépendantes qui a été traité dans le Chapitre 2. Nous supposons qu'initialement, un nœud maître génère une grande quantité de tâches de même taille, comme dans le cas des applications de calcul volontaire qui s'exécutent sur des plates-formes telles que BOINC [5] ou Folding@home [65]. Ces tâches vont être exécutées par des nœuds esclaves, dont les capacités de communication (en termes de latences et bande passante) et de calcul sont très hétérogènes. Dans le contexte des applications de calcul volontaire, le nombre de tâches est énorme et la minimisation de makespan n'a pas donc vraiment de sens. Par conséquent nous allons considérer la maximisation du débit, et le but sera de maximiser le nombre de tâches qui peuvent s'exécuter pendant une unité de temps une fois l'état stable de système atteint, comme dans le Chapitre 2.

Le troisième problème d'ordonnancement que nous allons considérer sera le problème de diffusion. C'est un problème classique, très utilisé dans les systèmes à large échelle et qui a été largement étudié dans la littérature [63, 98, 96]. La diffusion un-à-plusieurs est le modèle de communication collective le plus primaire : initialement, un nœud source génère les données qui doivent être diffusées ; à la fin, une copie des données doit se trouver sur chacun des nœuds cibles. Les algorithmes

parallèles nécessitent, assez souvent, l'envoi d'une même donnée à tous les autres processeurs, pour disséminer une information globale telle que la taille du problème ou les paramètres de l'application. Ce même cadre s'applique pour la diffusion une donnée "live stream", telle qu'un film. Dans ce travail, nous allons considérer un scénario simple, où les nœuds sont organisés sous la forme d'une étoile (le nœud source au centre), et où toutes les communications prennent place entre le nœud source et les clients.

Les conditions décrites dans les chapitres précédents restent valables pour les deux premiers problèmes d'ordonnancement. En particulier, on ne peut pas supposer connue la topologie du système à l'avance, puisque les mécanismes de découverte automatiques tel que ENV [90] et AlNEM [40] sont trop lents pour être utilisés dans des contextes dynamiques et à large échelle. Par conséquent, nous associons à chaque nœud des propriétés locales (bande passante entrante et sortante et capacité de calcul), dont les valeurs peuvent être déterminées facilement pendant l'exécution. Ainsi, les topologies des réseaux que nous considérons sont plutôt logiques que physiques. De même, pour modéliser les contentions, nous allons utiliser le modèle multi-port borné, où le serveur peut communiquer avec plusieurs clients simultanément, à la place du modèle 1-port. L'importance et l'utilité du modèle ont été évoquées dans les précédents chapitres. Dans le contexte des plates-formes à large échelle, le facteur d'hétérogénéité peut être élevé, et il n'est pas acceptable de supposer qu'un serveur de 100MB/s peut rester occupé pendant 10 secondes pendant qu'il communique un fichier de 1M à un nœud DSL de 100kB/s. Nous supposons également que toutes connexions sont gérées au niveau TCP. Il est intéressant de noter ici qu'au niveau TCP, plusieurs mécanismes de contrôle des qualités de services (QoS) tel que qdisc, sont disponibles dans les systèmes modernes, ce qui permet de contrôler le partage de la bande passante [21, 58]. En particulier, il est possible de gérer simultanément plusieurs connexions et de fixer la bande passante allouée à chaque connexion. Dans notre contexte, ces mécanismes sont particulièrement utiles vu que dans les ordonnancements optimaux, la bande passante allouée à une connexion entre deux nœuds P_i et P_j peut être (très) inférieure à B_i^{out} et B_j^{in}. Par conséquent, le modèle que nous proposons comprend les bénéfices des deux modèles multi-port et 1-port. Il permet à plusieurs communications de se produire simultanément, ce qui est nécessaire dans le contexte des plates-formes distribuées à large échelle, et l'implémentation pratique est assurée en utilisant les mécanismes de contrôle de bande passante.

Nous allons prouver en utilisant les trois problèmes d'ordonnancement mentionnés ci-dessus que ce modèle est intéressant et que des algorithmes d'ordonnancement peuvent atteindre une performance très sous-optimale, si on n'applique pas les politiques de QoS pour le partage de la bande passante. Notre but est de montrer la nécessité de tels mécanismes de qualité de service pour obtenir de bonnes performances (c'est-à-dire conformes à ce qui est prévu) des algorithmes d'ordonnacement. Ces mécanismes sont maintenant disponibles dans les noyaux des systèmes d'exploitation modernes. Pour montrer l'importance des mécanismes de contrôle de bande passante, nous fournirons à la fois des analyses théoriques au pire cas et des simulations en utilisant SimGRID.

Le reste du chapitre est organisé comme suit. Nous allons présenter les trois

problèmes d'ordonnancement. Pour chacun d'eux, nous allons modéliser le problème, présenter deux implémentations (l'une utilisant les mécanismes de contrôle de bande passante, l'autre non) et analyser théoriquement le pire cas dans les paragraphes 6.4, 6.5 et 6.6. Dans la deuxième partie (voir section 6.7) nous présentons les résultats expérimentaux pour les trois problèmes. Nous terminerons par une conclusion dans le paragraphe 6.8.

6.2 Motivation

Ce problème, ou plutôt ce constat sur l'influence des mécanismes de partage de bande passante sur la performance des algorithmes d'ordonnancement nous a été inspiré par les précédents travaux présentés dans les chapitres précédents, en particulier le chapitre 5. En effet, lorsque nous avons observé les résultats des simulations dans le cadre de la redistribution des fichiers, nous avons constaté que le fait de fixer à priori les débits maximaux, en utilisant une plate-forme de simulation "adaptée" permettait d'améliorer grandement les résultats. Ce même phénomène a été également observé lors du traitement du problème d'ordonnancement de tâches indépendantes 2, d'où la généralisation proposée dans ce chapitre.

6.3 Le contrôle de congestion dans TCP

Le contrôle de congestion consiste à essayer de gérer les problèmes d'embouteillages dans le réseau. Ces embouteillages se manifestent par un flux de paquets plus important en entrée qu'en sortie (serveur, routeur *etc.*). Les cas d'embouteillages se matérialisent par des pertes de paquets dans le réseau. Dans TCP, des mécanismes permettant de contrôler les trafics ont été mis en place afin de palier aux congestions.
Le contrôle de congestion peut être vu comme un contrôle de flux à l'aide d'une fenêtre de congestion. Les mécanismes de contrôle de congestion pour TCP ont été longtemps étudiés et de nombreux modèles ont été proposés dans la littérature, pour comprendre comment TCP traite la congestion [76, 80, 74]. Dans la suite, nous allons modéliser la congestion en utilisant la méthode "RTT-aware Max-Min Flow-level" qui a été proposée dans [25] et validée par le simulateur de réseaux NS-2 [77] dans [26].

Considérons la plate-forme basique décrite dans la Figure 6.1, qui va être utilisée dans la suite. Notons par B^{out} la bande passante sortante du nœud S, par b_i^{in} la bande passante entrante du nœud P_i et par λ_i la latence entre S et P_i. Considérons le cas où S envoie simultanément des messages à tous les P_i (le cas où tous les P_i envoient simultanément un message à S donne les mêmes résultats). La bande passante allouée à la communication entre S et P_i en utilisant le modèle "RTT-aware Max-Min Flow-level" est donnée par l'algorithme suivant (où B^{rem} présente la bande passante restante).

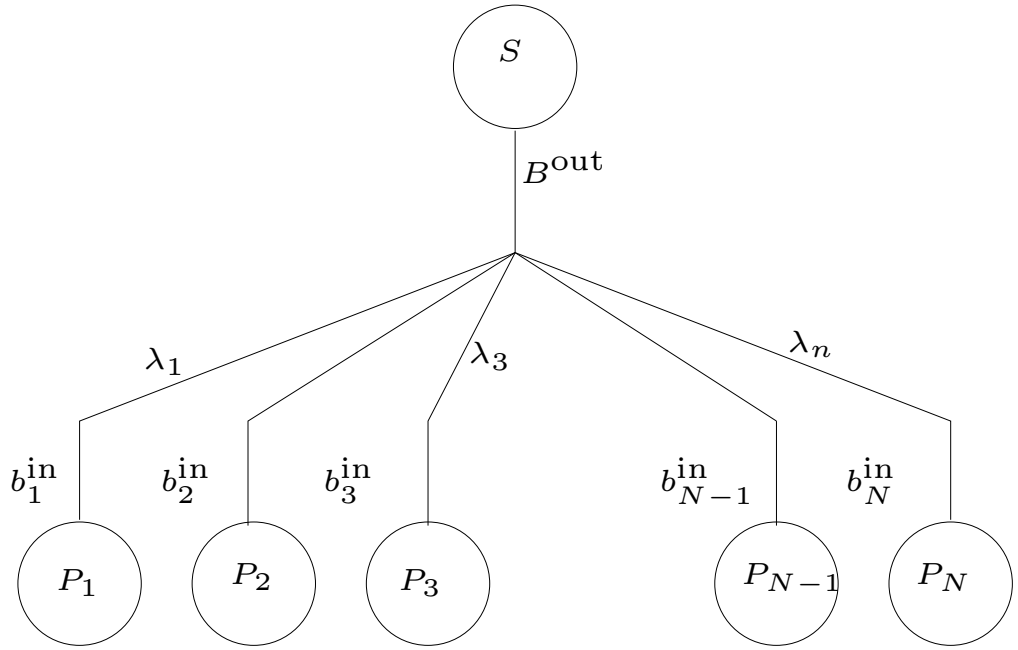

FIGURE 6.1: Partage de bande passante en présence de congestions

Soit	$\text{marked}_i = 0 \ \ \forall i; \quad B^{\text{rem}} = B^{\text{out}}$
Tant que $\exists i$,	$\text{marked}_i = 0$ et $b_i^{\text{in}} \leqslant \frac{\frac{1}{\lambda_i}}{\sum_{j,\ \text{marked}_j=0} \frac{1}{\lambda_j}} B^{\text{rem}}$
	$c_i = b_i^{\text{in}}; \text{marked}_i = 1; \quad B^{\text{rem}} = B^{\text{rem}} - b_i^{\text{in}};$
Fin Tant que	
Pour Tous i,	Si $\text{marked}_i = 0$ alors $c_i = \frac{\frac{1}{\lambda_i}}{\sum_{j,\ \text{marked}_j=0} \frac{1}{\lambda_j}} B^{\text{rem}}$
Fin Pour Tous	

En utilisant ce modèle, dans le cas où toutes les valeurs b_i^{in} sont grandes (par exemple plus grande que B^{out}), la bande passante allouée entre S et P_i dépend seulement du lien et elle est inversement proportionnelle à la latence de ce lien. D'autre part, s toutes les valeurs b_i^{in} sont petites alors la bande passante allouée à la communication entre S et P_i est b_i^{in}.

On peut donc démontrer les deux lemmes suivants.

Lemme 6.1 *Si* $\sum_i b_i^{in} \leqslant B^{out}$, *alors* $\forall i, \ c_i = b_i^{in}$.

Preuve.

Prouvons tout d'abord qu'il $\exists i, \ \ \text{marked}_i = 0$ et $b_i^{\text{in}} \leqslant \frac{\frac{1}{\lambda_i}}{\sum_{j,\ \text{marked}_j=0} \frac{1}{\lambda_j}} B^{\text{out}}$.

Tous les nœuds sont initialement non marqués. Supposons que $\forall i, \ \ b_i^{\text{in}} > \frac{\frac{1}{\lambda_i}}{\sum_j \frac{1}{\lambda_j}} B^{\text{out}}$.

Par conséquent, $\sum_i b_i^{\text{in}} > \frac{\sum_i \frac{1}{\lambda_i}}{\sum_j \frac{1}{\lambda_j}} B^{\text{out}} = B^{\text{out}}$, ce qui est absurde. Il existe donc au moins un nœud P_{i_1} tel que $b_{i_1}^{\text{in}} \leqslant \frac{\frac{1}{\lambda_{i_1}}}{\sum_{j,\ \text{marked}_{j=0}} \frac{1}{\lambda_j}} B^{\text{out}}$.

L'algorithme marque ce nœud P_{i_1} et alloue une bande passante $b_{i_1}^{\text{in}}$ à P_{i_1} et $B^{\text{rem}} = B^{\text{out}} - b_{i_1}^{\text{in}}$. Puisque initialement $\sum_i b_i^{\text{in}} \leqslant B^{\text{out}}$, alors $\sum_{i \neq i_1} b_i^{\text{in}} \leqslant B^{\text{rem}}$ et on peut terminer la preuve par induction.

Lemme 6.2 *Si* $\sum_i b_i^{in} \geqslant B^{out}$, *alors* $\sum_i c_i = B^{out}$.

Preuve.

À la fin de la boucle "Tant que", notons par $\mathcal{S}$ l'ensemble des nœuds qui ont été marqués. Par conséquent, $\forall P_i \in \mathcal{S},\ \ c_i = b_i^{\text{in}}$ et $B^{\text{rem}} = B^{\text{out}} - \sum_{P_i \in \mathcal{S}} c_i$.

À la fin de la boucle "Pour Tous", $\forall P_i \notin \mathcal{S},\ \ c_i = \frac{\frac{1}{\lambda_i}}{\sum_{P_j \notin \mathcal{S}} \frac{1}{\lambda_j}} B^{\text{rem}}$,

ainsi $\sum_{P_i \notin \mathcal{S}} c_i = B^{\text{rem}}$ et

$\sum_i c_i = \sum_{P_i \in \mathcal{S}} c_i + \sum_{P_i \notin \mathcal{S}} = B^{\text{out}} - B^{\text{rem}} + B^{\text{rem}} = B^{\text{out}}$.

6.4 Redistribution des fichiers

6.4.1 Modélisation du problème

Ce problème a été déjà traité dans le chapitre 5. Dans le contexte des systèmes de stockage à large échelle tel que Vespa [9], développé par Yahoo !, nous avons considéré le cas où un disque est ajouté au système. L'ensemble des fichiers qui doivent être transférés sur le nouveau disque est connu à l'avance et le problème d'ordonnancement consiste à trouver, pour chaque fichier, dans l'ensemble de ses répliques, celle qui doit être utilisée pour le transfert afin de minimiser le temps total de transfert. Chaque disque source est caractérisé par une bande passante sortante B_j^{out} et le disque destinataire D est caractérisé par une bande passante entrante B^{in}. Notons par $S = \{F_1, F_2, \ldots, F_k\}$ l'ensemble des fichiers qui doivent être transférés au disque destinataire D et notons par x_k^i la fonction indicatrice telle que $x_k^j = 1$ si le disque source contient une réplique du fichier F_k et 0 sinon. Chaque fichier F_k est caractérisé par taille s_k, il peut être transféré en fractions à partir de plusieurs disques ou entièrement à partir d'un seul disque. Les deux cas ont été étudiés dans le chapitre 5. En particulier, dans le cas du transfert entier, le problème a été montré NP-Complet et un algorithme d'approximation a été proposé. Par contre, dans le cas fractionnaire, un simple programme linéaire peut résoudre le problème et fournir les transferts des fichiers qui minimisent le makespan. Par conséquent, pour des raisons de simplicité, nous allons considérer le cas du transfert fractionnaire. Soit le programme linéaire suivant qui fournit une borne inférieure sur le temps nécessaire pour terminer tous les transferts.

$$\text{Minimiser } T \quad \begin{cases} \forall j,k & z_k^j \leqslant x_k^j B_j^{\text{out}} \text{ et } z_k^j \geqslant 0 \\ \forall k, & \sum_j z_k^j = s_k \\ \forall j & \sum_k z_k^j \leqslant B_j^{\text{out}} T \\ & \sum_j \sum_k z_k^j \leqslant B^{\text{in}} T \end{cases} .$$

dans lequel z_k^j présente la taille de la fraction du fichier F_k transférée à partir de disque source au disque destinataire D. Il est clair que toute solution doit satisfaire les conditions ci-dessus (si on fait la moyenne des bandes passantes utilisées au fil du temps), de sorte que la valeur optimale du programme linéaire T_{opt} est une borne inférieur du makespan réalisable. D'autre part, considérons une implémentation telle que chaque disque source envoie une fraction du fichier F_k au disque D au débit constant $\frac{z_k^j}{T_{\text{opt}}}$. Une telle implémentation doit finir tous les transferts des fichiers en temps T_{opt}.
Dans la Section 6.4.2, nous allons montrer comment on peut atteindre l'optimal en utilisant les mécanismes du contrôle de bande passante et nous allons prouver que sans de tels mécanismes, c'est-à-dire en considérant seulement les mécanismes de contrôle de congestion de TCP, la performance d'une telle implémentation peut être aussi mauvaise que $2T_{\text{opt}}$.

6.4.2 Implémentation

Dans cette section, nous allons définir deux implémentations du problème décrit dans la section précédente 6.4.1. Dans le cas où les fichiers peuvent être fractionnés et envoyés à partir de plusieurs nœuds sources au nœud destinataire, nous avons vu qu'un simple programme linéaire peut fournir l'ensemble des transferts qui minimisent le makespan. Plus précisément, la solution du programme linéaire fournit pour chaque fichier F^k la taille z_k^i de la fraction de F_k qui doit être transférée de S_i à D.
Pour évaluer l'impact des mécanismes de partage de bande passante sur la performance des algorithmes d'ordonnancement, nous allons considérer les deux implémentations suivantes de l'algorithme de redistribution de données.

1. **Implémentation 1** : Chaque disque source S_i possède la liste des morceaux des fichiers F_k qu'il doit envoyer au disque destinataire D. Il les envoie de manière synchrone au nœud destinataire.
2. **Implémentation 2** : L'**Implémentation 2** est exactement la même que l'**Implémentation 1** sauf qu'on borne la bande passante sortante de S_i à $\frac{\sum_k z_k^i}{T}$.

6.4.3 Analyse du pire cas

Dans cette section, nous allons prouver que le rapport entre le makespan en utilisant les mécanismes de contrôle de contention de TCP en présence de contentions et le makespan optimal en utilisant les mécanismes de contrôle de la bande passante

est borné supérieurement par 2. Nous prouvons, aussi, que cette borne est fine en exhibant une plate-forme dans laquelle le rapport peut être arbitrairement proche de 2.

6.4.3.1 Borne supérieure sur la perte de la performance

Considérons une plate-forme avec plusieurs sources disques S_i et un disque destinataire D et considérons le makespan M_2 pour terminer tous les transferts en utilisant **Implémentation 2**. Pour modéliser les contentions, nous allons utiliser le modèle "RTT-aware Max-Min Flow-level" qui a été introduit dans la Section 6.3. Nous allons prouver que le makespan M_2 en utilisant **Implémentation 2** ne peut pas être plus grand que deux fois le makespan M_1 obtenu en utilisant **Implémentation 1**. La preuve est basée sur les mêmes idées que la borne classique de Graham [48].
Nous allons distinguer deux types d'instants pendant l'exécution de **Implémentation 2**. La première phase est l'ensemble des instants tels que la bande passante entrante de D est totalement utilisée et la deuxième phase correspond au reste. Nous allons noter par T_1 la durée de la première phase et par T_2 la durée de la deuxième phase, alors on a $T_1 + T_2 = M_2$.

Théorème 6.3 *$T_1 \leqslant M_1$ et $T_2 \leqslant M_1$ et par conséquent $T_1 + T_2 \leqslant 2M_1$*

Preuve. Considérons, tout d'abord, la première phase. Pendant cette phase, la bande passante entrante de D est totalement utilisée donc la taille totale des données transmises de S_1 pendant la Phase 1 est exactement $S_1 = B^{\text{in}} \times T_1$. Par construction, on a $S_1 \leqslant S$, où S désigne la taille totale des données qui doivent être transférées à D et par conséquent $T_1 \leqslant \frac{S}{B^{\text{in}}}$. De plus, $\frac{S}{B^{\text{in}}}$ est une borne inférieure du temps de terminaison de tous les transferts, donc $T_1 \leqslant T_{\text{opt}} \leqslant M_1$.

Considérons maintenant la deuxième phase et plus précisément le disque source S_{last} qui est impliqué dans un transfert de fichiers à la fin de la Phase 2. À un instant t de la Phase 2, notons par $\mathcal{U}(t)$ l'ensemble des nœuds qui envoient effectivement des données à D. En utilisant les notations du lemme 6.2, $\sum_{S_i \in \mathcal{U}(t)} c_i(t) < B^{\text{in}}$ donc $\sum_i B_i^{\text{out}} < B^{\text{in}}$ et, suite au lemme 6.1, $\forall S_i \in \mathcal{U}(t), \quad c_i(t) = B_i^{\text{out}}$. Par conséquent, puisque S_{last} envoie encore des données à la fin de la Phase 2, alors il a envoyé des données à D à tous les instants de la Phase 2 avec un débit $B_{\text{last}}^{\text{out}}$. En conclusion, la quantité totale des données envoyée par S_{last} pendant Phase 2 est au moins $B_{\text{last}}^{\text{out}} \times T_2$. Clairement, la quantité totale de données envoyée par S_{last} est au plus $B_{\text{last}}^{\text{out}} \times M_1$, donc $T_2 \leqslant M_1$, ce qui termine la preuve du théorème.

6.4.3.2 Exemple de pire cas

Dans cette section, nous allons décrire un exemple dans lequel le rapport entre les deux makespan (M2 et M1) peut être arbitrairement proche de 2.

Théorème 6.4 *$\frac{M_2}{M_1}$ peut être arbitrairement proche de 2.*

Preuve. Nous allons prouver que la borne 2 dans le Théorème 6.3 est fine. Pour obtenir ce résultat, considérons la plate-forme suivante, constituée de deux sources disques S_1 et S_2 et d'un disque destinataire D avec les caractéristiques suivantes

$$S_1: \quad \lambda_1 = \epsilon^3, \quad B_1^{\text{out}} = 1; \qquad S_2: \quad \lambda_2 = \epsilon, \quad B_2^{\text{out}} = \epsilon; \qquad D: \ B^{\text{in}} = 1,$$

où ϵ présente une quantité arbitrairement petite et λ_1 et λ_2 représentent respectivement les latences entre S_1 et D et entre S_2 et D. Puisque les latences sont arbitrairement petites, nous n'allons pas considérer les délais introduits par ces latences mais plutôt nous concentrer sur leur impact sur contrôle de bande passante en utilisant l'algorithme RTT-aware Max-Min Flow-level présenté dans la Section 6.3.

Supposons que S_1 doit envoyer à D un fichier de taille 1 et que S_2 doit envoyer un fichier de taille ϵ. Dans la solution optimale, S_1 envoie en continu des données pendant le temps $1+\epsilon$ à D en utilisant la bande passante $\frac{1}{1+\epsilon}$ et S_2 envoie en continu des données pendant le temps $1+\epsilon$ à D en utilisant la bande passante $\frac{\epsilon}{1+\epsilon}$. Ainsi au temps $1+\epsilon$, D a reçu les deux fichiers.

Regardons maintenant ce qui se passe si on s'appuie uniquement sur les mécanismes de TCP de partage de bande passante pour traiter les contentions.

Au départ,

$$\begin{cases} B_1^{\text{out}} = 1 > \frac{\frac{1}{\epsilon^3}}{\frac{1}{\epsilon}+\frac{1}{\epsilon^3}} = 1 - \epsilon^2 + o(\epsilon^2) \\ B_2^{\text{out}} = \epsilon > \frac{\frac{1}{\epsilon}}{\frac{1}{\epsilon}+\frac{1}{\epsilon^3}} = \epsilon^2 + o(\epsilon^2) \end{cases},$$

ainsi (les valeurs c_i sont attribuées dans la boucle Forall) $c_1 = 1 - \epsilon^2 + o(\epsilon^2)$ et $c_2 = \epsilon^2 + o(\epsilon^2)$. Par conséquent, S_1 termine son transfert au temps $1 + \epsilon^2$. À cet instant, S_2 a transféré $\epsilon^2 + o(\epsilon^2)$ données donc il a besoin d'un supplément du temps $1-\epsilon$ pour terminer son transfert en utilisant sa bande passante maximale ϵ. Le temps total nécessaire donc pour transférer les deux fichiers en utilisant les mécanismes de TCP de partage de bande passante est $(2 - \epsilon)$, c'est-à-dire $(2 - 3\epsilon)$ fois le temps nécessaire pour faire ces transferts de façon optimale, ce qui termine la preuve de ce théorème.

6.5 Ordonnancement des tâches indépendantes

6.5.1 Modélisation du problème

Ce problème a été déjà introduit et traité dans le Chapitre 2. Dans ce problème, on considère une plate-forme maître-esclave élémentaire pour exécuter un grand nombre de tâches de tailles égales. Initialement, le nœud maître M possède (ou génère à un débit donné) un grand nombre de tâches qui vont être exécutées par un ensemble de nœuds esclaves $\{P_i\}$. Le nœud maître est caractérisé par sa bande passante sortante B^{out}. Par contre, un nœud esclave est caractérisé par sa bande passante entrante B_i^{in} et sa capacité de calcul w_i. Étant donné que toutes les tâches sont de même taille, on normalise toutes les quantités B^{out}, B_i^{in} et w_i en terme de tâches (transférées ou exécutées) par unité de temps. Pour modéliser ce problème, considérons le programme linéaire suivant

$$\text{Maximiser } \rho_{\text{opt}} = \sum_i \rho_i \quad \begin{cases} \forall i & \rho_i \leqslant \min(B_i^{\text{in}}, w_i) \text{ et } \rho_i \geqslant 0 \\ & \sum_i \rho_i \leqslant B^{\text{out}} \end{cases},$$

où ρ_i représente le nombre de tâches que le nœud maître alloue à P_i par unité de temps. Si on considère une solution valide de ce problème pendant une longue période T et si on note par x_i le nombre moyen des tâches exécutées par P_i par unité de temps, c'est-à-dire $x_i = N_i(T)/T$, alors les x_i satisfont les conditions du programme linéaire et par conséquent $\sum_i x_i \leqslant \rho_{\text{opt}}$ et $\sum_i N_i(T) \leqslant \rho_{\text{opt}} T$, ce qui prouve que ρ_{opt} est une borne supérieure du débit réalisable. D'autre part, comme nous l'avons montré dans le chapitre 2 consacré à l'étude de ce problème, la durée de la phase d'initialisation (période pendant laquelle les nœuds esclaves reçoivent leurs premières tâches) peut être négligée (puisque la durée de cette phase est constante) et les conditions du programme linéaire restent satisfaites après cette phase, donc cette solution est valide et elle exécute $\sum_i \rho_i$ tâches par unité de temps. Le débit réalisable tend vers ρ_{opt}.
Dans la Section 6.5.2, nous allons montrer comment on peut atteindre l'optimal en utilisant les mécanismes de contrôle de bande passante et nous allons prouver que sans de tels mécanismes, c'est-à-dire en considérant seulement les mécanismes de contrôle de congestion TCP, la performance d'une telle implémentation peut être arbitrairement proche de $3/4\rho_{\text{opt}}$.

6.5.2 Implémentation

Dans cette section, nous allons définir deux implémentations du problème décrit dans la section précédente 6.5.1 pour évaluer l'effet de l'utilisation des mécanismes de partage de bande passante sur la performance de système. Nous avons vu dans la section précédente, qu'un simple programme linéaire fournit pour chaque nœud esclave P_i le débit ρ_i avec lequel le maître doit envoyer les tâches au P_i pour maximiser le débit total, c'est-à-dire le nombre total des tâches qui peuvent être exécutées en utilisant cette plate-forme par unité de temps. Comme dans le cas de la redistribution des fichiers, nous allons proposer deux implémentations différentes pour évaluer l'impact de l'utilisation des mécanismes de partage de bande passante sur la performance du problème d'ordonnancement.

1. **Implémentation 1**. Pour éviter l'attente sans rien faire (la famine), chaque nœud esclave commence avec deux tâches dans le tampon local. À chaque instant où P_i commence l'exécution d'une nouvelle tâche, il demande une autre tâche et le nœud maître initialise la communication immédiatement.

2. **Implémentation 2**. L'**implémentation 2** est exactement la même que l'**implémentation 1** sauf qu'on borne la bande passante utilisée par M pour envoyer les tâches de P_i à ρ_i.

Dans la suite, on notera par T_1 le débit réalisable en utilisant **Implémentation 1** et par T_2 le débit réalisable en utilisant **Implémentation 2**.

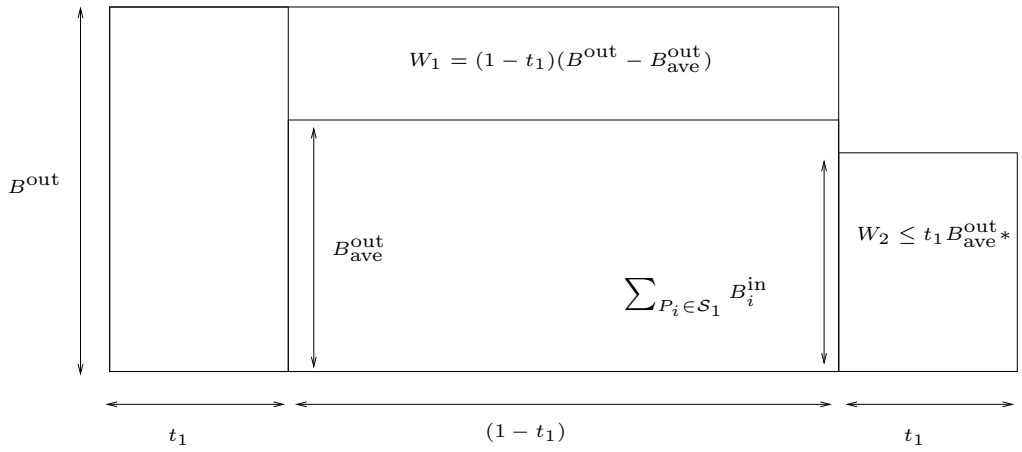

FIGURE 6.2: Partage de bande passante via TCP et **Implémentation 1**

6.5.3 Analyse du pire cas

Dans cette section, nous allons prouver que le rapport entre le makespan en utilisant les mécanismes de contrôle de contention de TCP en présence de contentions et le makespan optimal en utilisant les mécanismes de contrôle de la bande passante est inférieur à $\frac{4}{3}$. Nous prouvons, aussi, que cette borne est fine en exhibant une plate-forme dans laquelle le rapport peut être arbitrairement proche de $\frac{4}{3}$.

6.5.3.1 Borne supérieure sur la dégradation du débit

Théorème 6.5 $T_2 \leqslant \frac{4}{3} T_1$

Preuve. Considérons le résultat obtenu en utilisant **Implémentation 1** pendant une longue période de temps et notons par x_i le nombre moyen des tâches exécutées par P_i pendant une unité de temps. Si $\sum x_i = B^{\text{out}}$, alors l'**Implémentation 1** obtient asymptotiquement le débit optimal et le théorème est vrai. Sinon, notons par t_1 la fraction du temps pendant laquelle la bande passante du maître est entièrement utilisée. D'autre part, notons par $B^{\text{out}}_{\text{ave}}$ la moyenne de la bande passante utilisée quand la bande passante du maître n'est pas entièrement utilisée, c'est-à-dire pendant la fraction du temps $(1 - t_1)$ (voir Figure 6.2).

En utilisant ces notations, on peut trouver une première borne supérieure du débit W perdu en utilisant **Implémentation 1** : $W \leqslant (1 - t_1)(B^{\text{out}} - B^{\text{out}}_{\text{ave}})$.

Considérons maintenant l'ensemble $\mathcal{S}_1$ des processeurs esclaves qui ne sont pas utilisés à leur meilleur débit, c'est-à-dire tels que $x_i < \min(w_i, B^{\text{in}}_i)$ et par $\mathcal{S}_2$ l'ensemble des processeurs tel que $x_i = \min(w_i, B^{\text{in}}_i)$. De plus, notons par $\rho^{(k)}_{\text{opt}}$, $k = 1, 2$ le débit total obtenu par l'ensemble des esclaves $\mathcal{S}_k$ dans la solution optimale. On peut noter que $\sum_{P_i \in \mathcal{S}_2} x_i \geqslant \rho^{(2)}_{\text{opt}}$. Puisque les processeurs de $\mathcal{S}_1$ ne sont pas utilisés à leur vitesse de calcul maximale, alors ils demandent continuellement des tâches en utilisant **Implémentation 1**. Par conséquent, à chaque instant quand la bande

passante de M n'est pas totalement utilisée, l'esclave $P_i \in \mathcal{S}_1$ reçoit des tâches au débit B_i^{in}. Par la suite, $B_{\text{ave}}^{\text{out}} \geqslant \sum_{P_i \in \mathcal{S}_1} B_i^{\text{in}}$ et $\sum_{P_i \in \mathcal{S}_1} x_i \geqslant (1-t_1)B_{\text{ave}}^{\text{out}}$. En outre, par définition, $\rho_{\text{opt}}^{(1)} \leqslant \sum_{P_i \in \mathcal{S}_1} B_i^{\text{in}}$. En conséquence,

$$\begin{array}{rcl} \rho_{\text{opt}}^{(1)} - \sum_{P_i \in \mathcal{S}_1} x_i & \leqslant & \sum_{P_i \in \mathcal{S}_1} B_i^{\text{in}} - (1-t_1)B_{\text{ave}}^{\text{out}} \\ & \leqslant & B_{\text{ave}}^{\text{out}} - (1-t_1)B_{\text{ave}}^{\text{out}} \\ & \leqslant & t_1 B_{\text{ave}}^{\text{out}}. \end{array}$$

$$\text{et ainsi, } W = \rho_{\text{opt}}^{(2)} - \sum_{P_i \in \mathcal{S}_2} x_i + \rho_{\text{opt}}^{(1)} - \sum_{P_i \in \mathcal{S}_1} x_i \leqslant t_1 B_{\text{ave}}^{\text{out}}.$$

En utilisant les deux bornes supérieures de W, on obtient $W \leqslant f(t_1, B_{\text{ave}}^{\text{out}}) = \min((1-t_1)(B^{\text{out}} - B_{\text{ave}}^{\text{out}}), t_1 B_{\text{ave}}^{\text{out}})$.

Si on considère, en premier, $f(t_1, B_{\text{ave}}^{\text{out}})$ comme une fonction de $B_{\text{ave}}^{\text{out}} \in [0, B^{\text{out}}]$, on observe que $f(t_1, B_{\text{ave}}^{\text{out}})$ est minimal quand $B_{\text{ave}}^{\text{out}} = (1-t_1)B^{\text{out}}$ et $f(t_1, (1-t_1)B^{\text{out}}) = t_1(1-t_1)B^{\text{out}}$ donc $\forall t_1, B_{\text{ave}}^{\text{out}}, \quad W \leqslant \frac{B^{\text{out}}}{4}$, ce qui termine la preuve du théorème.

6.5.3.2 Exemple de pire cas

Théorème 6.6 *$\frac{T_2}{T_1}$ peut être arbitrairement proche de $\frac{4}{3}$.*

Preuve. Prouvons maintenant que la borne de $\frac{4}{3}$ est fine. Pour obtenir ce résultat, considérons la plate-forme suivante, constituée de deux nœuds esclaves P_1 et P_2 et d'un nœud maître M avec les caractéristiques suivantes

$$P_1 :\ \lambda_1 = \epsilon^3,\ \ w_1 = 1,\ \ B_1^{\text{in}} = 2; \quad P_2 :\ \lambda_2 = \epsilon,\ \ w_1 = 1,\ \ B_2^{\text{in}} = 1; \quad D :\ B^{\text{out}} = 2,$$

où ϵ représente une quantité arbitrairement petite et λ_1 et λ_2 représentent respectivement les latences entre M et P_1 et entre M et P_2. Comme précédemment, puisque les latences sont arbitrairement petites, on ne va pas considérer les délais introduits par ces latences mais plutôt se concentrer sur leur impact sur le partage de la bande passante en utilisant l'algorithme RTT-aware Max-Min Flow-level présenté dans la Section 6.3. En utilisant **Implémentation 2**, P_1 commence l'exécution sa première tâche au temps 0 et termine au temps 1. Le maître commence l'envoi de la nouvelle tâche au temps 0 en utilisant la bande passante 1 et la communication se termine au temps 1. Le même processus s'applique sur P_2 donc exactement deux tâches s'exécutent à chaque unité de temps, vu que $T_2 = 2$.

Essayons maintenant de voir ce qui se passe si on s'appuie uniquement sur les mécanismes de partage de bande passante de TCP pour traiter les congestions . Au départ,

$$\left\{ \begin{array}{l} B_1^{\text{out}} = 2 > \frac{\frac{1}{\epsilon^3}}{\frac{1}{\epsilon}+\frac{1}{\epsilon^3}} \times 2 = 2 - 2\epsilon^2 + o(\epsilon^2) \\ B_2^{\text{out}} = \epsilon > \frac{\frac{1}{\epsilon}}{\frac{1}{\epsilon}+\frac{1}{\epsilon^3}} \times 2 = 2\epsilon^2 + o(\epsilon^2) \end{array} \right. ,$$

donc (les valeurs c_i sont attribuées sans la boucle "Forall") $c_1 = 2 - \epsilon^2 + o(\epsilon^2)$ et $c_2 = 2\epsilon^2 + o(\epsilon^2)$. Par conséquent, P_1 reçoit sa première tâche au temps $\frac{1}{2} + 2\epsilon^2 + o(\epsilon^2)$

et P_2 reçoit seulement $\epsilon^2 + o(\epsilon^2)$ tâches au temps $\frac{1}{2} + 2\epsilon^2 + o(\epsilon^2)$. Entre le temps $\frac{1}{2} + 2\epsilon^2 + o(\epsilon^2)$ et le temps 1, P_2 reçoit les tâches au débit 1 puisque il est le seul qui demande des tâches. Ainsi, au temps 1, P_2 a reçu $\frac{1}{2} + O(\epsilon^2)$ tâches. Au temps 1, le même schéma s'applique puisque P_1 demande une nouvelle tâche et va la recevoir au temps $\frac{3}{2} + O(\epsilon^2)$ par contre P_2 reçoit $O(\epsilon^2)$ extra tâches. Par conséquent, P_2 va terminer la réception de sa première tâche au temps $2 - \epsilon^2$. Par la suite, le même schéma s'applique pendant chaque période de taille 2.
En conclusion, **Implémentation 1** exécute deux tâches chaque unité de temps par contre **Implémentation 2** exécute 3 tâches à chaque 2 unités de temps, ce qui termine la preuve du théorème.

6.6 Diffusion

6.6.1 Modélisation du problème

Dans le contexte de la diffusion, un nœud source possède (ou génère à un débit donné) un grand fichier qui doit être envoyé à tous les nœuds clients. De nombreux algorithmes de diffusion ont été conçus pour des machines parallèles telles que les "meshs" et les hypercubes (voir parmi d'autres [63, 98]). D'autre part, ce problème est au cœur des systèmes de distribution de contenu, comme les caches web et les réseaux d'échanges de fichiers, qui doivent pouvoir servir une population de clients à la fois très grande et fortement dynamique (temps de connexion très courts), en particulier les systèmes de distribution "live streaming" tel que CoolStreaming [108] et SplitStream [27]. Dans les deux cas, on s'intéresse à la distribution d'un grand message à tous les nœuds d'une plate-forme à large échelle. Par conséquent, on ne s'intéresse pas à la minimisation du makespan mais plutôt à la maximisation du débit (c'est-à-dire le débit de diffusion maximal, une fois l'état stable du système atteint).
Dans ce contexte, le nœud source S est caractérisé par sa bande passante sortante B^{out}. Par contre un nœud client P_i est caractérisé par sa bande passante entrante B_i^{in} et sa bande passante sortante B_i^{out} puisqu'il peut être utilisé comme une source intermédiaire pour les autres nœuds une fois qu'il a reçu une partie du message diffusé. Dans le cas le plus général, le but sera de concevoir un réseau de recouvrement $G = (P, E, c)$ tel que le nœud P_i envoie des messages au nœud P_j à un débit $c(P_i, P_j)$. Le débit de diffusion optimal en G peut être caractérisé en utilisant les flux. En effet, les théorèmes dans [36, 42] relient le débit de diffusion optimal avec la coupe minimale d'un graphe pondéré. $\forall j$, on peut noter par $\text{cut}(j)$ la valeur minimale d'une coupe de G en deux ensembles de clients C_1 et C_2 tel que $C_1 \bigcup C_2 = P$, $C_1 \bigcap C_2 = \varnothing$, $S \in C_1$ et $P_j \in C_2$. $\forall j$, $\text{cut}(j)$ désigne la valeur maximale d'un flot entre le nœud source S et P_j et donc représente une borne supérieure du débit de diffusion. En outre, il est prouvé dans [36] que cette borne est en fait fine, c'est-à-dire que le débit optimal de diffusion d'un graphe G est égal à $\text{mincut}(G) = \min_j \text{cut}(j)$. Des algorithmes efficaces [42] ont été conçus pour calculer un ensemble des arbres pondérés qui atteint ce débit de diffusion optimal à partir des valeurs $c(P_i, P_j)$. Par conséquent, on peut utiliser l'approche fondée sur la programmation linéaire pro-

posée dans [69] pour calculer le débit de diffusion optimal $\rho*$ et $\forall i, j, \quad c(P_j, P_i)$, la bande passante totale utilisée entre les nœuds P_j et P_i. Une fois toutes les valeurs $c(P_i, P_j)$ déterminées, Massoulié et al. [75] ont proposé récemment un algorithme randomisé décentralisé pour implémenter une diffusion qui atteint un débit arbitrairement proche de $\rho*$, dans le cas où toutes les bandes passantes entrantes ont une capacité infinie. Dans ce contexte, une communication entre P_i et P_j peut atteindre la bande passante sortante maximale de P_i, ainsi on peut utiliser totalement la bande passante disponible sans prendre en compte les contentions. Dans la suite, nous allons considérer un contexte simple, dans lequel les nœuds sont organisés en étoile avec le nœud source au centre et les nœuds clients en périphérie (sans bandes passantes sortantes). Par contre, on ne fait aucune hypothèse sur la bande passante entrante des nœuds clients. En particulier, les bandes passantes entrantes peuvent être inférieures à B^{out} ce qui nécessite de faire plusieurs communications simultanément pour agréger la bande passante B^{out}, et ainsi exige la prise en compte des contentions.
Dans la Section 6.6.2, nous allons montrer comment on peut atteindre l'optimal en utilisant les mécanismes du contrôle de bande passante et nous allons prouver que sans de tels mécanismes, c'est-à-dire en considérant seulement les mécanismes de contrôle de congestion TCP, la performance d'une telle implémentation peut être arbitrairement inférieure à $\rho*$.

6.6.2 Implémentation

Dans cette section, nous considérons le contexte simple d'une plate-forme en forme d'étoile où le maître est au centre. Dans ce cas, si N présente le nombre de clients, le débit réalisable $\rho*$ est donné par $\rho* = \min(\frac{B^{\text{out}}}{N}, \min_i B_i^{\text{in}})$. Considérons les deux implémentations suivantes :

1. **Implémentation 1**. À chaque unité de temps, la source S engage simultanément une communication avec chaque nœud destinataire, et envoie un message de taille $\rho*$ contenant les derniers paquets générés à chaque client.
2. **Implémentation 2**. L'**implémentation 2** est la même que l'**implémentation 1** sauf qu'on borne la bande passante utilisée par S pour envoyer le message à P_i à $\rho*$.

Pour comparer les deux implémentations, on va exécuter la diffusion pendant une longue durée T. Soit $x_i^k(T)$ la taille du message reçu au temps T par P_i en utilisant l'implémentation k. La performance de l'implémentation k est donnée par $\rho_k = \lim_{T\to+\infty} \frac{\min_i x_i^k(T)}{T}$.. Dans la suite on va prouver que ρ_1 peut être arbitrairement plus petit que $\rho*$.

6.6.2.1 Analyse du pire cas

Théorème 6.7 *ρ_1 peut être arbitrairement plus petit que ρ_2 et $\rho*$.*

Preuve. Considérons la plate-forme suivante se composant de N clients et d'un nœud source avec une bande passante sortante $B^{\text{out}} = N$. Les $N - 1$ premiers

clients $P_i,\ i = 1 \ldots N-1$ ont une bande passante $B_i^{\text{in}} = \frac{N}{N-1}$ et la latence entre eux $(P_i,\ i = 1 \ldots N-1)$ et S est donnée par $\lambda_i = \epsilon^2$. Concernant le client P_N, il a une bande passante entrante 1 et la latence entre lui et S égale à ϵ. En fin, on suppose que ϵ est arbitrairement petit et en particulier $\epsilon \times N << 1$.

En utilisant cette plate-forme, **Implémentation 2** réalise un débit optimal $\rho_2 = \rho* = 1$. En effet, à chaque étape de temps, tous les clients sont servis simultanément avec une bande passante $\rho*$ et tous les transferts se terminent en une unité de temps.

En utilisant **Implémentation 1**, la somme des bandes passantes des nœuds clients impliqués dans les communications avec S au temps 0 est donnée par $N+1$, donc des congestions ont lieu au niveau du nœud source. En utilisant l'algorithme présenté dans la Section 6.3 pour modéliser le partage de bande passante dans TCP en présence de congestions, on obtient $\forall i = 1, \ldots, N-1,\ c_i = \frac{N}{N-1}(1+O(\epsilon))$ et $c_N = \frac{N\epsilon}{N-1} + o(\epsilon)$.
Par conséquent, tous les clients P_i, $i \leqslant N-1$ reçoivent le premier message au temps $1 - 1/N + O(\epsilon)$. Par contre, pendant ce temps, P_N a reçu seulement un message de taille $O(\epsilon)$. Pendant l'intervalle de temps entre $1 - 1/N$ et 1 (instant auquel un nouveau message est diffusé à tous les clients), P_N est l'unique nœud communiquant avec S et $C_N = 1$. Par conséquent, au temps 1, P_N a reçu un message de taille $1/N + O(\epsilon)$. Le même schéma s'applique entre le temps 1 et 2 et il va prendre un temps N à P_N pour recevoir complètement le premier message. En conséquence, la performance totale est $\rho_1 = 1/N$, ce qui termine la preuve du théorème.

6.7 Résultas

Dans cette section, nous allons présenter les résultats des simulations, pour les trois problèmes, obtenus avec des instances aléatoires (mais réalistes) avec SimGRID.

6.7.1 Redistribution

Dans cette section, nous allons présenter les résultats de simulation relatifs au problème de redistribution. Dans ce cas, comme le cas du problème d'ordonnancement des tâches indépendantes (Section 6.5.2) et le problème de diffusion (Section 6.6.2), nous considérons une simple plate-forme en étoile. Dans ce contexte, la simulation du modèle "multi-port borné" dans SimGRID a été validée dans [26] en utilisant le simulateur de réseaux NS-2. Vu que nous nous intéressons à l'impact de l'utilisation des mécanismes de partage de bande passante de TCP, nous considérons les cas où $\sum_i B_i^{\text{out}} = 2B^{\text{in}}$ et $\sum_i B_i^{\text{out}} = 1.2B^{\text{in}}$, qui correspondent respectivement à des niveaux haut et bas de congestion. D'autre part, vu que la latence a un impact majeur sur le partage de la bande passante en utilisant TCP, nous allons considérer la cas où les latences sont assez homogènes (valeurs aléatoires entre 10^{-5} et 3×10^{-5}) et le cas où les latences sont très hétérogènes (10^{-x}, où x est une valeur aléatoire entre 3 et 7). Enfin, afin d'évaluer l'impact du nombre des nœuds, nous considérons les cas où $N = 10$ et $N = 20$.

Le tableau suivant présente le ratio entre le makespan obtenu avec l'**Implémentation 2** et celui obtenu en utilisant l'**Implémentation 1**. Toutes les valeurs correspondent à 20 simulations différentes, et dans tous les cas, nous calculons le minimum, le maximum et la moyenne des 20 simulations.

	$\sum_i B_i^{\text{out}} = 1.2B^{\text{in}}$				$\sum_i B_i^{\text{out}} = 2B^{\text{in}}$			
Homogène	rapport	min.	max.	moy.	rapport	min.	max.	moy.
	$N = 10$	1.18	1.45	1.26	$N = 10$	1.02	1.17	1.11
	$N = 20$	1.25	1.40	1.30	$N = 20$	1.05	1.15	1.11
Hétérogène	rapport	min.	max.	moy.	rapport	min.	max.	moy.
	$N = 10$	1.30	1.57	1.45	$N = 10$	1.09	1.26	1.16
	$N = 20$	1.22	1.64	1.51	$N = 20$	1.06	1.29	1.13

Les résultats de simulation prouvent l'impact du contrôle de la bande passante sur la performance des algorithmes de redistribution des fichiers. On remarque que, comme attendu, l'impact est plus important quand l'hétérogénéité est élevée (dans ce cas, la bande passante allouée à certains nœuds en présence de congestion peut être très petite, retardant ainsi leurs transferts) et quand le niveau de la congestion est relativement bas. En effet, dans le cas où $\sum_i B_i^{\text{out}} = 2B^{\text{in}}$, même si certains transferts sont presque complètement retardé au début, une fois le premier ensemble des transferts est terminé, $\sum_i B_i^{\text{out}}$ reste importante et ainsi la bande passante B^{in} n'est pas gaspillée.

6.7.2 Ordonnancement des tâches indépendantes

Comme dans la section précédente, nous allons présenter ici des résultats des simulations obtenus avec SimGRID pour des instances aléatoires mais réalistes. Le tableau suivant présente le facteur entre le débit obtenu avec l'**Implémentation 2** et celui obtenu en utilisant l'**Implémentation 1**. Toutes les valeurs correspondent à 20 simulations différentes, et dans tous les cas, nous calculons le minimum, le maximum et la moyenne des 20 simulations.
Pour chaque simulation pour estimer le débit, nous exécutons les deux implémentations sur 200 tâches. Afin d'estimer l'impact des communications plutôt que le traitement, les vitesses de traitement des processeurs sont fixées et très rapides et donc dans la solution optimale, les processeurs sont limités par leurs capacités de communication.

	$\sum_i B_i^{\text{out}} = 1.2B^{\text{in}}$				$\sum_i B_i^{\text{out}} = 2B^{\text{in}}$			
Homogène	rapport	min.	max.	moy.	rapport	min.	max.	moy.
	$N = 10$	1.01	1.03	1.02	$N = 10$	1.01	1.04	1.02
	$N = 20$	1.00	1.02	1.01	$N = 20$	1.00	1.03	1.01
Hétérogène	rapport	min.	max.	moy.	rapport	min.	max.	moy.
	$N = 10$	1.01	1.04	1.03	$N = 10$	1.01	1.04	1.03
	$N = 20$	1.01	1.03	1.02	$N = 20$	1.00	1.03	1.02

La différence entre les deux implémentations est beaucoup moins importante que pour le cas de la redistribution de fichiers (et de la diffusion). Ceci est dû au fait que contrairement aux autres situations, il peut y avoir une compensation entre les processeurs. Les processeurs avec des petites latences exécutent plus de tâches avec l'**Implémentation 1** qu'avec l'**Implémentation 2**. Pour obtenir une différence plus significative, on peut saturer les processeurs en calcul dans la solution optimale, et former deux groupes de capacités d'exécutions agrégées équivalentes, un groupe avec des petites latences et un autre avec des grandes latences. Dans ce cas, le rapport est proche de la borne $\frac{4}{3}$ prouvée dans la Section 6.5.

6.7.3 Diffusion

Comme dans les deux précédents cas, nous allons présenter ici des résultats des simulations obtenus avec SimGRID sur des instances aléatoires mais réalistes. Le tableau suivant présente le ratio entre ρ_1 et ρ_2, le débit obtenu avec l'**Implémentation 2** et celui obtenu en utilisant l'**Implémentation 1**. Toutes les valeurs correspondent à 20 simulations différentes, et dans tous les cas, nous calculons le minimum, le maximum et la moyenne des 20 simulations.
Pour chaque simulation pour estimer le débit, nous exécutons les deux implémentations 500 fois.

	$\sum_i B_i^{\text{out}} = 1.2B^{\text{in}}$				$\sum_i B_i^{\text{out}} = 2B^{\text{in}}$			
Homogène	rapport	min.	max.	moy.	rapport	min.	max.	moy.
	$N = 10$	1.01	1.03	1.02	$N = 10$	1.01	1.22	1.07
	$N = 20$	1.00	1.02	1.01	$N = 20$	1.00	1.09	1.03
Hétérogène	rapport	min.	max.	moy.	rapport	min.	max.	moy.
	$N = 10$	1.01	1.09	1.04	$N = 10$	1.01	1.79	1.47
	$N = 20$	1.00	1.04	1.03	$N = 20$	1.00	1.33	1.19

Les résultats de simulations prouve que le débit obtenu avec l'**Implémentation 1** peut être plus petit que celui obtenu par l'**Implémentation 2**, particulièrement quand les latences sont très hétérogènes. En effet, dans le cas où plusieurs communications se déroulent simultanément, les processeurs avec des grandes latences prennent une très petite partie de la bande passante. Puisque les nouvelles communications sont lancées à chaque étape, alors la taille des données reçues par ces processeurs est significativement petite, particulièrement dans le cas des fortes congestions $\sum_i B_i^{\text{out}} = 2B^{\text{in}}$.

6.8 Conclusion

Dans ce chapitre, nous avons étudié l'influence des mécanismes de contrôle de bande passante sur la performance de trois problèmes d'ordonnancement sur des plates-formes à large échelle. Deux de ces problèmes (la redistribution des fichiers et l'ordonnancement des tâches indépendantes) ont été déjà étudiés dans les précédents chapitres et ce chapitre a été consacré à l'étude de l'importance des mécanismes de

contrôle de bande passante sur la performance de ces algorithmes. Le troisième algorithme étudié, la diffusion, est un problème d'ordonnancement classique et considéré comme une brique de base pour plusieurs autres problèmes plus sophistiqués.
Dans ce contexte, le réseau d'interconnexion est basé sur internet donc la topologie ne peut pas être considérée connue à l'avance. Les coûts de communications et les contentions sont modélisés à travers d'un nombre limité de paramètres, qui peuvent être déterminés pendant l'exécution (les latences et les bandes passantes entrantes et sortantes). Concernant la modélisation des communications, nous avons adopté le modèle multi-port borné au lieu du modèle classique 1-port qui ne convient pas bien à notre cas où les ressources sont très hétérogènes. Le modèle multi-port permet à plusieurs communications de se produire simultanément à condition que les capacités des bandes passantes ne soient pas dépassées .

En particulier, nous avons comparé pour les trois problèmes traités (la redistribution des fichiers, l'ordonnancement des tâches indépendantes et la diffusion) les performances obtenues avec deux différentes implémentations, une implémentation utilisent les mécanismes de contrôle de bande passante et une s'appuyant uniquement sur TCP pour le partage de bande passante. Pour chaque problème nous avons prouvé une borne supérieure sur la perte maximale de performance qui peut être induite par le partage de bande passante avec TCP. Nous avons prouvé aussi, pour chaque problème, que cette borne est fine en exhibant des instances qui atteignent ces bornes.

En conclusion, ce travail montre que dans le contexte des plates-formes à large échelle, où les latences sont très hétérogènes, l'utilisation des mécanismes de contrôle de bande passante, qui sont disponibles dans les systèmes d'exploitations modernes, est nécessaire pour atteindre des bonnes performances.

Chapitre 7
Conclusion

Dans cette thèse, nous avons analysé quelques problèmes liés à l'exécution d'applications sur des plates-formes hétérogènes, distribuées et à grande échelle. Plus précisément, nous nous sommes concentrés sur la ditribution de tâches indépendantes (avec un maître et des esclaves), la redistribution de données (en modèle All-To-All personnalisé) et la diffusion de contenus (en modèle One-To-All). Cette étude couvre donc l'essentiel des primitives de communications qui sont effectivement utilisées dans ce type de plates-formes puisque, pour des raisons de sécurité essentiellement, il n'existe pas d'applications complexes modélisées par des DAGs et s'exécutant à grande échelle.

La première conclusion que je propose porte sur la pertinence du modèle multi-port. En effet, le modèle multi-port permet de prendre en compte les contentions entre les communications, avec un jeu de paramètres relativement restreint (les bandes passantes entrante et sortante en chaque nœud) et instanciables à l'exécution. De plus, si on ajoute une contrainte sur le degré des nœuds participants, c'est à dire sur le nombre maximal de communications dans lesquels ils peuvent être engagés simultanément, il contient également les qualités du modèle 1-port. Enfin, nous avons vu que même si la contrainte sur le degré rend les problèmes d'ordonnancement NP-Complets, cette difficulté peut être facilement contournée en utilisant l'augmentation de ressources.

La deuxième conclusion porte sur la prise en compte du dynamisme et de la variabilité des performances de ressources de calcul. Nous avons proposé dans le cas de l'allocation de tâches indépendantes un algorithme "on-line" dont le coût est raisonnable, puisqu'il permet de conserver le débit optimal, à un coût au plus 2 fois celui d'un algorithme permettant d'assurer un facteur d'approximation constant. Il est donc possible, dans certains cas, de maintenir l'optimalité même en présence de changements dans les performances. Nous trouvons ici un autre avantage du modèle multi-port, qui est plus robuste à des estimations erronées des paramètres : en effet, si la bande passante d'un nœud se révèle être très inférieure à la prévision, l'influence de cette mauvaise estimation sera dramatique dans le cas du 1-port (puisque l'émetteur et le récepteur seront bloqués) et négligeables dans le cas multi-port (puisque seulement une petite partie des ressources de l'émetteur sera consommée). Toutefois, la dernière partie de la thèse sur l'importance du contrôle de bande passante démontre qu'il est important de ne pas laisser le système s'auto-gérer, mais

bien d'apporter un maximum de données issues de l'ordonnancement statique (pour mettre en place les mécanismes de contrôle de bande passante). Dans ces conditions, de mauvaises estimations peuvent donc également avoir une influence négative sur les performances.

Chapitre 7
Liste des publications

[1] *"Scheduling Techniques for Effective System Reconfiguration in Distributed Storage System"*, Babino, Cyril ; Beaumont, Olivier ; Rejeb, Hejer, *ICPADS* (2008)

[2] *"Online and Offline Independent Tasks Allocation Schemes on Large Scale Master-Worker Heterogeneous Platforms"*, Beaumont, Olivier ; Dubois, Lionel ; Rejeb, Hejer ; Thraves, Christopher, *Workshop on New Challenges in Distributed Systems* (Avril 2009)

[3] *"Online Allocation of Splitable Clients to Multiple Servers on Large Scale Heterogeneous Platforms"*, Beaumont, Olivier ; Dubois, Lionel ; Rejeb, Hejer ; Thraves, Christopher, *Algotel* (juin 2009)

[4] *"Allocation of Clients to Multiple Servers on Large Scale Heterogeneous Platforms"*, Beaumont, Olivier ; Dubois, Lionel ; Rejeb, Hejer ; Thraves, Christopher, *ICPADS* (2009)

[5] *"Online Allocation of Splitable Clients to Multiple Servers on Large Scale Heterogeneous Platforms"*, Beaumont, Olivier ; Dubois, Lionel ; Rejeb, Hejer ; Thraves, Christopher, *Raport Recherche* (2009)

[6] *"On the Importance of Bandwidth Control Mechanisms for Scheduling on Large Scale Heterogeneous Platforms"*, Beaumont, Olivier ; Rejeb, Hejer, *Rapport de recherche* (Decembre 2009)

[7] *"Online Allocation of Clients to Multiple Servers on Large Scale Heterogeneous Platforms"*, Beaumont, Olivier ; Dubois, Lionel ; Rejeb, Hejer ; Thraves, Christopher, *PDP* (2010)

[8] *"On the Importance of Bandwidth Control Mechanisms for Scheduling on Large Scale Heterogeneous Platforms"*, Beaumont, Olivier ; Rejeb, Hejer, *IPDPS* (Avril 2010)

Références bibliographiques

[1] Transparency. http://serverfault.com/questions/124595/how-do-i-get-transparent-efficient-file-system-snapshotting-or-versioning-on-

[2] WikipÈdia. http://www.wikipedia.fr/.

[3] G. Aggarwal, R. Motwani, and A. Zhu. The load rebalancing problem. *Journal of Algorithms*, 60(1) :42–59, 2006.

[4] D.T. Altilar and Y. Paker. Optimal scheduling algorithms for communication constrained parallel processing. *Lecture notes in computer science*, pages 197–206.

[5] D.P. Anderson. BOINC : A System for Public-Resource Computing and Storage. In *5th IEEE/ACM International Workshop on Grid Computing*, pages 365–372, 2004.

[6] D.P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@ home : an experiment in public-resource computing. *Communications of the ACM*, 45(11) :56–61, 2002.

[7] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity And Approximation*, volume the compendium. Springer.

[8] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and approximation : Combinatorial optimization problems and their approximability properties*. Springer Verlag, 1999.

[9] R. Baeza-Yates and R. Ramakrishnan. Data challenges at Yahoo ! In *Proceedings of the 11th international conference on Extending database technology : Advances in database technology*, pages 652–655. ACM New York, NY, USA, 2008.

[10] K.R. Baker. *Introduction to sequencing and scheduling*. John Wiley & Sons, 1974.

[11] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Scheduling Strategies for Master-Slave Tasking on Heterogeneous Processor Platforms. *IEEE Transactions on Parallel and Distributed Systems*, pages 319–330, 2004.

[12] M.E. Baran and F.F. Wu. Network reconfiguration in distribution systems for loss reduction and load balancing. *Power Delivery, IEEE Transactions on*, 4(2) :1401–1407, 1989.

[13] O. Beaumont, H. Casanova, A. Legrand, Y. Robert, and Y. Yang. Scheduling divisible loads on star and tree networks : results and open problems. *IEEE Transactions on Parallel and Distributed Systems*, pages 207–218, 2005.

[14] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Pipelining Broadcasts on Heterogeneous Platforms. *IEEE Transactions on Parallel and Distributed Systems*, pages 300–313, 2005.

[15] Olivier Beaumont, Lionel Eyraud Dubois, Hejer Rejeb, and Christopher Thraves. Allocation of Clients to Multiple Servers on Large Scale Heterogeneous Platforms. Research Report RR-6767, INRIA, 2008.

[16] D. Bertsimas and D. Gamarnik. Asymptotically optimal algorithms for job shop scheduling and packet routing. *Journal of Algorithms*, 33(2) :296–318, 1999.

[17] V. Bharadwaj. *Scheduling divisible loads in parallel and distributed systems*. Wiley-IEEE Computer Society Pr, 1996.

[18] J. Blazewicz, M. Drabowski, J. Weglarz, I. Automatyki, and P. Poznańska. Scheduling multiprocessor tasks to minimize schedule length. *IEEE Transactions on Computers*, 100(35) :389–393, 1986.

[19] T.D. Braun, HJ Siegal, N. Beck, L.L. Boloni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, D. Hensgen, et al. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. volume 61, pages 810–837. Elsevier, 2001.

[20] T.D. Braun, HJ Siegal, N. Beck, L.L. Boloni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, D. Hensgen, et al. A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems. In *Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth*, pages 15–29. IEEE, 2002.

[21] Martin A. Brown. Traffic Control HOWTO. Chapter 6. Classless Queuing Disciplines. *http ://tldp.org/HOWTO/Traffic-Control-HOWTO/classless-qdiscs.html*, 2006.

[22] P. Brucker. *Scheduling algorithms*. Springer Verlag, 2007.

[23] P.H. Carns, W.B. Ligon III, R.B. Ross, and R. Thakur. PVFS : A parallel file system for Linux clusters. In *Proceedings of the 4th annual Linux Showcase & Conference-Volume 4*, page 28. USENIX Association, 2000.

[24] H. Casanova. Modeling large-scale platforms for the analysis and the simulation of scheduling strategies. 2004.

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

[26] H. Casanova and L. Marchal. A network model for simulation of grid application. *RR-40-2002*, 40, 2002.

[27] M. Castro, P. Druschel, A.M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream : high-bandwidth multicast in cooperative environments. *ACM SIGOPS Operating Systems Review*, 37(5) :298–313, 2003.

[28] H. Chen and M. Maheswaran. Distributed dynamic scheduling of composite tasks on grid computing systems. In *Proceedings of the 11th IEEE Heterogeneous Computing Workshop*. Citeseer, 2002.

[29] F. Chung, R. Graham, J. Mao, and G. Varghese. Parallelism versus Memory Allocation in Pipelined Router Forwarding Engines. *Theory of Computing Systems*, 39(6) :829–849, 2006.

[30] F. Cristian. Understanding fault-tolerant distributed systems. *Communications of the ACM*, 34(2) :56–78, 1991.

[31] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Journal of parallel and distributed computing*, 7(2) :279–301, 1989.

[32] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS.

[33] W. Day and S. Hill. Farming : towards a rigorous definition and efficient transputer implementation. page 49, 1992.

[34] del.icio.us. http://del.icio.us/.

[35] P.J. Denning. Fault tolerant operating systems. *ACM Computing Surveys (CSUR)*, 8(4) :359–389, 1976.

[36] J. Edmonds. Edge disjoint branchings. In *Combinatorial Algorithms : Courant Computer Science Symposium 9 : January 24-25, 1972*, page 91. Algorithmics Press, 1972.

[37] H. El-Rewini, T.G. Lewis, and H.H. Ali. *Task scheduling in parallel and distributed systems*. 1994.

[38] L. Epstein and R. van Stee. Improved results for a memory allocation problem. *WADS 2007*, pages 362–373, 2007.

[39] Leah Epstein and Rob van Stee. Approximation schemes for packing splittable items with cardinality constraints. In Christos Kaklamanis and Martin Skutella, editors, *WAOA*, volume 4927 of *Lecture Notes in Computer Science*, pages 232–245. Springer, 2007.

[40] L. Eyraud-Dubois, A. Legrand, M. Quinson, and F. Vivien. A First Step Towards Automatically Building Network Representations. *Lecture Notes in Computer Science*, 4641 :160, 2007.

[41] Flickr. http://flickr.com/.

[42] H.N. Gabow and KS Manu. Packing algorithms for arborescences (and spanning trees) in capacitated graphs. *Mathematical Programming*, 82(1) :83–109, 1998.

[43] M.R. Garey and D.S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-completeness*. WH Freeman San Francisco, 1979.

[44] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. In *SOSP '03 : Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, New York, NY, USA, 2003. ACM Press.

[45] The great internet mersenne prime search (gimps). http://www.mersenne.org/.

[46] L. Golubchik, J.C.S. Lui, and R.R. Muntz. Chained declustering : load balancing and robustness to skew and failures. In *Second International Workshop on Research Issues on Data Engineering, 1992 : Transaction and Query Processing*, pages 88–95, 1992.

[47] S. Gorn. Transparent-mode control procedures for data communication, using the American standard code for information interchange : a tutorial. *Commun. ACM*, 8 :203–206, April 1965.

[48] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.R. Kan. Optimization and approximation in deterministic sequencing and scheduling : A survey. *Annals of Discrete Mathematics*, 5(2) :287–326, 1979.

[49] T. Hacker, B. Athey, and B. Noble. The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network. pages 434–443, 2002.

[50] M. Harchol-Balter and A.B. Downey. Exploiting process lifetime distributions for dynamic load balancing. *ACM Transactions on Computer Systems (TOCS)*, 15(3) :253–285, 1997.

[51] B. Hong and V. K. Prasanna. Distributed Adaptive Task Allocation in Heterogeneous Computing Environments to Maximize Throughput. In *International Parallel and Distributed Processing Symposium IPDPS'2004*, page 52b. IEEE Computer Society Press, 2004.

[52] B. Hong and V.K. Prasanna. Distributed adaptive task allocation in heterogeneous computing environments to maximize throughput. *International Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, 2004.

[53] R. J. Honicky and Ethan L. Miller. Replication under scalable hashing : A family of algorithms for scalable decentralized data distribution. In *IPDPS*. IEEE Computer Society, 2004.

[54] E. Horowitz and S. Sahni. Exact and approximate algorithms for scheduhng nomdentlcal processors. *Journal of the ACM*, 23(2) :317–327, 1976.

[55] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, and M.J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems (TOCS)*, 6(1) :51–81, 1988.

[56] H. Hsiao and D.J. DeWitt. Chained Declustering : A New Availability Strategy for Multiprocessor Database Machines. In *6th International Conference on Data Engineering*, page 456. IEEE Computer Society Press, 1990.

[57] YF Hu, RJ Blake, and DR Emerson. An optimal migration algorithm for dynamic load balancing. *Concurrency - Practice and Experience*, 10(6) :467–483, 1998.

[58] B. Hubert et al. Linux Advanced Routing & Traffic Control. Chapter 9. Queueing Disciplines for Bandwidth Management. *http ://lartc.org/lartc.pdf*, 2002.

[59] K. Hwang. *Advanced computer architecture : parallelism, scalability, programmability*, volume 348. McGraw-Hill New York, 1993.

[60] O.H. Ibarra and C.E. Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the ACM (JACM)*, 24(2) :280–289, 1977.

[61] H.A. James. Scheduling in metacomputing systems. 1999.

[62] P. Jogalekar and M. Woodside. Evaluating the scalability of distributed systems. *Parallel and Distributed Systems, IEEE Transactions on*, 11(6) :589–603, 2002.

[63] S.L. Johnsson and C.T. Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Transactions on Computers*, 38(9) :1249–1268, 1989.

[64] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent Hashing and Random Trees : Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *STOC '97 : Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663, New York, NY, USA, 1997. ACM Press.

[65] S.M. Larson, C.D. Snow, M. Shirts, and V.S. Pande. Folding@ Home and Genome@ Home : Using distributed computing to tackle previously intractable problems in computational biology. *Computational Genomics*, 2002.

[66] C. Leangsuksun, J. Potter, and S. Scott. Dynamic task mapping algorithms for a distributed heterogeneous computing environment. In *4th IEEE Heterogeneous Computing Workshop (HCW95*, pages 30–34, 1995.

[67] E.K. Lee and C.A. Thekkath. Petal : Distributed virtual disks. *ACM SIGOPS Operating Systems Review*, 30(5) :84–92, 1996.

[68] Jan Karel Lenstra, David B.Shmoys, and Eva Tardos. Approximation Algorithms for scheduling unreleated parallel machines. *Mathematical Progamming*, 46.

[69] Z. Li, B. Li, D. Jiang, and L.C. Lau. On achieving optimal throughput with network coding. In *Proceedings IEEE INFOCOM*, 2005.

[70] Q. Lian, W. Chen, and Z. Zhang. On the impact of replica placement to the reliability of distributed brick storage systems. 2005.

[71] W.B. Ligon III and R.B. Ross. An overview of the parallel virtual file system. In *Proceedings of the 1999 Extreme Linux Workshop*. Citeseer, 1999.

[72] M. Livny and M. Melman. Load balancing in homogeneous broadcast distributed systems. In *Proceedings of the Computer Network Performance Symposium*, pages 47–55. ACM, 1982.

[73] M. Maheswaran, S. Ali, HJ Siegal, D. Hensgen, and R.F. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth*, pages 30–44. IEEE, 2002.

[74] L. Massoulie and J. Roberts. Bandwidth sharing : objectives and algorithms. In *IEEE INFOCOM*, 1999.

[75] L. Massoulie, A. Twigg, C. Gkantsidis, and P. Rodriguez. Randomized decentralized broadcasting algorithms. In *IEEE INFOCOM*, pages 1073–1081, 2007.

[76] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *ACM SIGCOMM Computer Communication Review*, 27(3) :67–82, 1997.

[77] S. McCanne, S. Floyd, K. Fall, K. Varadhan, et al. Network simulator ns-2, 2000.

[78] M. Minoux. *Programmation mathématique : théorie et algorithmes*, volume 1. Dunod, 1983.

[79] M. Nuttall. A brief survey of systems providing process or object migration facilities. *ACM SIGOPS Operating Systems Review*, 28(4) :64–80, 1994.

[80] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput : A simple model and its empirical validation. *ACM SIGCOMM Computer Communication Review*, 28(4) :303–314, 1998.

[81] David A. Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (raid). *SIGMOD Rec.*, 17 :109–116, June 1988.

[82] DK Pradhan. Fault-tolerant computing. *Computer*, pages 6–7, 1980.

[83] Sean Rhea, Patrick Eaton, Dennis Geels, Hakim Weatherspoon, Ben Zhao, and John Kubiatowicz. Pond : The OceanStore Prototype. In *FAST '03 : Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 1–14, Berkeley, CA, USA, 2003. USENIX Association.

[84] H.G. Rotithor. Taxonomy of dynamic task scheduling schemes in distributed computing systems. *IEE Proceedings : Computers and Digital Techniques*, 141(1) :1–10, 1994.

[85] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. *ACM SIGOPS Operating Systems Review*, 35(5) :188–201, 2001.

[86] L. Rudolph, M. Slivkin-Allalouf, and E. Upfal. A simple load balancing scheme for task allocation in parallel machines. In *Proceedings of the third annual ACM symposium on Parallel algorithms and architectures*, pages 237–245. Citeseer, 1991.

[87] Christian Schindelhauer and Gunnar Schomaker. Weighted distributed hash tables. In *SPAA '05 : Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 218–227, New York, NY, USA, 2005. ACM Press.

[88] H. Shachnai and T. Tamir. Multiprocessor Scheduling with Machine Allotment and Parallelism Constraints. *Algorithmica*, 32(4) :651–678, 2002.

[89] Hadas Shachnai, Tami Tamir, and Omer Yehezkely. Approximation schemes for packing with item fragmentation. *Theory Comput. Syst.*, 43(1) :81–98, 2008.

[90] G. Shao, F. Berman, and R. Wolski. Using effective network views to promote distributed application performance. In *International Conference on Parallel and Distributed Processing Techniques and Applications*. CSREA Press, June 1999.

[91] Elizabeth A. M. Shriver, Bruce Hillyer, and Abraham Silberschatz. Performance Analysis of Storage Systems. In Günter Haring, Christoph Lindemann, and Martin Reiser, editors, *Performance Evaluation*, volume 1769 of *Lecture Notes in Computer Science*, pages 33–50. Springer, 2000.

[92] SIMGRID. http://simgrid.gforge.inria.fr/.

[93] N. Spring and R. Wolski. Application level scheduling of gene sequence comparison on metacomputers. In *Proceedings of the 12th international conference on Supercomputing*, pages 141–148. ACM, 1998.

[94] Hong Tang, Aziz Gulbeden, Jingyu Zhou, William Strathearn, Tao Yang, and Lingkun Chu. A Self-Organizing Storage Cluster for Parallel Data-Intensive Applications. In *SC '04 : Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 52, Washington, DC, USA, 2004. IEEE Computer Society.

[95] A.N. Tantawi and D. Towsley. Optimal static load balancing in distributed computer systems. *Journal of the ACM (JACM)*, 32(2) :445–465, 1985.

[96] Y.C. Tseng, S.Y. Wang, and C.W. Ho. Efficient broadcasting in wormhole-routed multicomputers : anetwork-partitioning approach. *IEEE TPDS*, 10(1) :44–61, 1999.

[97] R. van Renesse. Efficient reliable internet storage. In *Workshop on Dependable Distributed Data Management*. Citeseer, 2004.

[98] J. Watts and R.A. Geijn. A pipelined broadcast for multidimensional meshes. *Parallel Processing Letters*, 5(2) :281–292, 1995.

[99] S.A. Weil, S.A. Brandt, E.L. Miller, and C. Maltzahn. CRUSH : Controlled, scalable, decentralized placement of replicated data. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 122. ACM, 2006.

[100] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. Ceph : A Scalable, High-Performance Distributed File System. In *OSDI*, pages 307–320. USENIX Association, 2006.

[101] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, and Carlos Maltzahn. CRUSH : controlled, scalable, decentralized placement of replicated data. In *SC '06 : Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 122, New York, NY, USA, 2006. ACM Press.

[102] M. Wieczorek, R. Prodan, and T. Fahringer. Scheduling of scientific workflows in the ASKALON grid environment. *ACM SIGMOD Record*, 34(3) :62, 2005.

[103] Changxun Wu and Randal Burns. Achieving Performance Consistency in Heterogeneous Clusters. In *HPDC '04 : Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing*, pages 140–149, Washington, DC, USA, 2004. IEEE Computer Society.

[104] XFS Filesystem Structure. `oss.sgi.com/projects/xfs/papers/xfs_filesystem_structure.pdf`.

[105] Yahoo site. `http://www.yahoo.com/`.

[106] Yahoo ! Mail. `http://mail.yahoo.com`.

[107] Y. Yang and H. Casanova. Multi-round algorithm for scheduling divisible workload applications : analysis and experimental evaluation. *University of California, San Diego, Dept. of Computer Science and Engineering, Technical Report CS2002-0721*, 2002.

[108] X. Zhang, J. Liu, B. Li, and Y.S.P. Yum. CoolStreaming/DONet : A data-driven overlay network for peer-to-peer live media streaming. In *Proceedings IEEE INFOCOM*, 2005.