

# Redistribution de données à travers un réseau à haut débit

## THÈSE

présentée et soutenue publiquement le 14 décembre 2005

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1

(spécialité informatique)

par

Wagner Frédéric

### Composition du jury

*Rapporteurs :* M. Pierre Manneback  
M. Evripidis Bampis

*Examineurs :* M. Olivier Beaumont  
M. Jens Gustedt (Directeur de thèse)  
M. Emmanuel Jeannot (Directeur de thèse)  
M. Dominique Mery

Mis en page avec la classe thloria.

## Remerciements

Cette thèse est le résultat de trois ans de travaux personnels, mais également de l'attention et de l'aide que j'ai reçues de la part d'un grand nombre de personnes. Je tiens à remercier ici tous ceux qui, de près ou de loin ont pris part à ce voyage de trois années, voyage tenant plus d'un parcours en haute mer que d'une simple promenade.

Je tiens ainsi à remercier, tous les gens dont l'aide technique m'a été précieuse, mes encadrants, Jens et Emmanuel, les membres du projet RedGrid, Christian Perez, André Ribes, Frédéric Desprez, . . .

Je remercie également les membres de mon jury de thèse d'avoir bien voulu consacrer une part de leur temps à la relecture de mon travail.

Je remercie aussi les administrateurs système de l'IUT Nancy-Brabois pour la mise à disposition des salles machines dont j'ai été le bénéficiaire, ainsi que Guillaume Mornet pour ses réactions rapides aux différents problèmes techniques que j'ai pu rencontrer.

Je n'oublie bien sûr pas tous les thésards qui m'ont épaulé par leur soutien psychologique. Merci donc à Yves Caniou, Mohammed Essaïdi, Hervé Muller, Szathmary Laszlo, Mathieu Daquin, Guillaume Doyen, et tous les autres que j'oublie sans doute.

Enfin, cette thèse n'aurait pas été possible sans le soutien de ma famille, et je remercie donc du fond du cœur mon frère, mes parents, et toi Kinda, pour ton aide de tous les jours.



*To our good old doctor Asimov.*



# Table des matières

Table des figures	9
-------------------	---

## Chapitre 1

### Introduction

1.1	Contexte . . . . .	13
1.1.1	Couplage de code . . . . .	13
1.1.2	Parallélisme et grilles de calcul . . . . .	14
1.2	Motivation . . . . .	14
1.3	Organisation de la thèse . . . . .	15

## Chapitre 2

### Grilles de calcul et redistribution de données

2.1	Calcul parallèle . . . . .	19
2.2	Grilles de calcul . . . . .	20
2.2.1	Distribution de ressources . . . . .	21
2.2.2	Agrégation de ressources . . . . .	21
2.2.3	Récupération de ressources . . . . .	22
2.2.4	Mise en commun de ressources . . . . .	23
2.3	Redistributions de données . . . . .	24
2.3.1	Couplage de code . . . . .	24
2.3.2	Optimisations de redistributions . . . . .	25

## Chapitre 3

### Redistribution de données à travers un réseau à haut débit

3.1	Problème . . . . .	33
3.1.1	Description . . . . .	34
3.1.2	Motivation . . . . .	37
3.2	Modélisation . . . . .	41
3.2.1	Paramètres . . . . .	42

3.2.2	Redistribution par étapes . . . . .	43
3.2.3	Le problème KPBS . . . . .	46
3.2.4	Problèmes en relation avec KPBS . . . . .	50
3.2.5	Expériences de validation du modèle . . . . .	58

**Chapitre 4**

**Algorithmes d'approximation pour le problème KPBS**

4.1	GGP . . . . .	61
4.1.1	Principe de fonctionnement . . . . .	62
4.1.2	Algorithme . . . . .	69
4.1.3	Propriétés . . . . .	72
4.2	OGGP . . . . .	80
4.2.1	Principe de fonctionnement . . . . .	80
4.2.2	Algorithme formel . . . . .	81
4.2.3	Propriétés . . . . .	82
4.3	Expériences . . . . .	84
4.3.1	Une bibliothèque de manipulation de graphes bipartis : <i>libkpbs</i> . . . . .	84
4.3.2	Simulations . . . . .	84
4.3.3	Tests locaux exécutés avec la bibliothèque <i>MPICH</i> . . . . .	89
4.3.4	Tests locaux exécutés manuellement . . . . .	92
4.3.5	Conclusion . . . . .	94

**Chapitre 5**

**Extensions au problème KPBS**

5.1	Redistribution de données sur un réseau hétérogène . . . . .	96
5.1.1	Problème . . . . .	96
5.1.2	Algorithme d'approximation . . . . .	102
5.1.3	Propriétés de l'algorithme DGGP . . . . .	103
5.1.4	Expériences . . . . .	108
5.2	Redistribution de données utilisant un réseau local . . . . .	111
5.2.1	Problème du routage . . . . .	111
5.2.2	Régime permanent . . . . .	112
5.2.3	Algorithme . . . . .	113
5.2.4	Étude . . . . .	123

**Chapitre 6**

**Conclusion**

---

6.1 Conclusion . . . . .	125
6.2 Perspectives . . . . .	127

<b>Annexes</b>
----------------

<b>Annexe A</b> <b>Notations</b>
-------------------------------------

<b>Annexe B</b> <b>Programmes</b>
--------------------------------------

B.1 Exemple d'utilisation de la bibliothèque libkpbs . . . . .	131
--	-----

<b>Annexe C</b> <b>Bibliographie</b>
---

<b>Bibliographie</b>	<b>135</b>
----------------------	------------



# Table des figures

1.1	Exemple de configuration réseau . . . . .	14
2.1	Exemple de redistribution de blocs cycliques à blocs cycliques . . . . .	26
2.2	Distribution cyclique, par blocs, ou bloc-cyclique d'un tableau à une dimension . . . . .	28
2.3	Optimisation de la quantité de données à redistribuer . . . . .	29
2.4	Déroulement de l'algorithme du Caterpillar . . . . .	30
2.5	Déroulement d'une heuristique gloutonne pour une redistribution sur un réseau hétérogène . . . . .	31
3.1	Exemple de configuration réseau . . . . .	34
3.2	Le réseau VTHD . . . . .	35
3.3	GRID 5000 . . . . .	35
3.4	Deux grappes de trois machines . . . . .	38
3.5	Matrice de communication . . . . .	38
3.6	Exécution de la redistribution par force brute . . . . .	39
3.7	Ordonnancement des communications en 2 étapes . . . . .	39
3.8	Tests de saturation sur le réseau VTHD (réalisés par Christian Perez et André Ribes) . . . . .	40
3.9	Deux grappes reliées par une dorsale lente . . . . .	40
3.10	Matrice de communication (homogène) . . . . .	41
3.11	Comparaison entre les deux approches . . . . .	41
3.12	Matrice de communication initiale . . . . .	43
3.13	Matrice normalisée et graphe de communication . . . . .	43
3.14	Décomposition d'un graphe de communication en couplages . . . . .	44
3.15	Déroulement d'une redistribution en plusieurs étapes . . . . .	44
3.16	Comparaison entre deux ordonnancements simples . . . . .	45
3.17	Comparaison entre ordonnancements sans et avec préemption . . . . .	46
3.18	Décomposition optimale en utilisant la préemption . . . . .	46
3.19	Graphe de communication simple, limité par $k$ . . . . .	49
3.20	Exécution de l'algorithme pseudo-polynomial de résolution de PBS . . . . .	51
3.21	Exécution de l'algorithme polynomial de résolution de PBS . . . . .	52
3.22	Étoile optique de diffusion et sélection . . . . .	54
3.23	Exemple d'ordonnancement sur un réseau à étoile optique . . . . .	54
3.24	Décomposition en 3 étapes de communication . . . . .	55
3.25	3 matrices de configuration . . . . .	56
3.26	Diminutions du nombre de matrices de configuration (nombre d'étapes) . . . . .	56
3.27	Création de $k$ sommets virtuels . . . . .	57

3.28	Une instance d'un problème d'open-shop . . . . .	57
3.29	Validation du modèle : acuité de l'estimation des temps pour des graphes aléatoires	59
3.30	Validation du modèle : acuité de l'estimation des temps pour une famille particulière de graphes . . . . .	60
4.1	Graphe de communication régulier . . . . .	63
4.2	Graphe étendu prenant en compte la contrainte sur $k$ . . . . .	63
4.3	Graphe simple, avant extension . . . . .	65
4.4	Déroulement de l'algorithme d'extension . . . . .	66
4.5	Graphe d'entrée illustrant la nécessité d'équilibrer latence et préemption . . . . .	67
4.6	Arrondi et décomposition pour $\beta = 2$ . . . . .	67
4.7	Arrondi et décomposition pour $\beta = 1$ . . . . .	67
4.8	Extension d'un graphe tel que $\frac{P}{k} \geq W$ . . . . .	68
4.9	Matrice de communication normalisée initiale . . . . .	69
4.10	Graphe $G$ initial, obtenu à partir de la matrice de communication 4.9 pour $\beta = 2$	69
4.11	Graphe $H$ , après l'arrondi des poids des arêtes de $G$ . . . . .	71
4.12	Graphes $I$ et $J$ , obtenus par ajouts d'arêtes virtuelles . . . . .	71
4.13	Décomposition de $J$ en deux couplages . . . . .	71
4.14	Solution finale . . . . .	72
4.15	Graphe problématique pour GGP ( $k = 3$ ) . . . . .	77
4.16	Graphe $I$ , solution intermédiaire à un facteur de $\frac{12}{5}$ et solution finale à un facteur de $\frac{9}{5}$ . . . . .	78
4.17	Second graphe problématique ( $k = 4$ ) . . . . .	78
4.18	Graphe problématique (seconde famille) pour GGP . . . . .	79
4.19	Ordonnancement à un facteur de $\frac{4}{3}$ . . . . .	79
4.20	Ordonnancement à un facteur de $\frac{3}{2}$ . . . . .	80
4.21	Simulations des heuristiques gloutonnes sur des graphes de poids faibles, moyenne des temps . . . . .	86
4.22	Simulations des heuristiques gloutonnes sur des graphes de poids faibles, maximum des temps . . . . .	86
4.23	Simulations des heuristiques gloutonnes sur des graphes de poids élevés, moyenne des temps. . . . .	87
4.24	Simulations des heuristiques gloutonnes sur des graphes de poids élevés, maximum des temps. . . . .	87
4.25	GGP et OGGP pour des graphes de poids faibles . . . . .	88
4.26	GGP et OGGP pour des graphes de poids élevés . . . . .	88
4.27	GGP et OGGP lorsque $\beta$ varie . . . . .	89
4.28	Temps obtenus pour des redistributions par force brute ou ordonnancées à l'aide de GGP pour $k = 3$ (basées sur <i>mpich</i> ) . . . . .	90
4.29	Temps obtenus pour des redistributions par force brute ou ordonnancées à l'aide de GGP pour $k = 5$ (basées sur <i>mpich</i> ) . . . . .	90
4.30	Temps obtenus pour des redistributions par force brute ou ordonnancées à l'aide de GGP pour $k = 7$ (basées sur <i>mpich</i> ) . . . . .	91
4.31	Temps obtenus sur des graphes de 25 arêtes pour une redistribution par force brute ou ordonnancée (basée sur la <i>libc</i> ) . . . . .	92
4.32	Temps obtenus sur des graphes de 35 arêtes pour une redistribution par force brute ou ordonnancée (basée sur la <i>libc</i> ) . . . . .	93

4.33	Temps obtenus sur des graphes de 45 arêtes pour une redistribution par force brute ou ordonnancée (basée sur la libc) . . . . .	93
4.34	Temps de transferts sur des graphes générant de mauvaises performances pour une redistribution par force brute . . . . .	94
5.1	Configuration réseau hétérogène de grappes homogènes . . . . .	96
5.2	Graphe de communication justifiant plusieurs communications simultanées sur un réseau hétérogène de grappes homogènes . . . . .	97
5.3	Configuration réseau de grappes hétérogènes . . . . .	97
5.4	Réseau de grappes hétérogènes avant utilisation de qualité de service . . . . .	98
5.5	Graphe de communication (les poids sont des quantités de données) . . . . .	98
5.6	Meilleure solution possible sans qualité de service et sans saturation . . . . .	98
5.7	Meilleure solution avec qualité de service . . . . .	99
5.8	Réseau de grappes hétérogènes, matrice de communication (en Mbit) . . . . .	99
5.9	Graphe de communication après calcul de la bande passante de base . . . . .	100
5.10	Ordonnancement solution, plusieurs communications simultanées par nœud . . . . .	101
5.11	Partage des sommets du graphe de communication de la figure 5.9 . . . . .	104
5.12	Graphe de communication problématique pour l'algorithme DGGP . . . . .	106
5.13	Graphes intermédiaires $H$ et $U$ pour le calcul de DGGP . . . . .	106
5.14	Solutions intermédiaire et finale pour le graphe de communication de la figure 5.12	106
5.15	Exemples problématiques pour $k = 3$ et $k = 4$ . . . . .	107
5.16	Évaluation de DGGP ( $k = 10$ ) . . . . .	108
5.17	Évaluation de DGGP ( $k = 3$ ) . . . . .	108
5.18	DGGP,OGGP : ratios d'évaluation . . . . .	109
5.19	DGGP : Comparaisons avec OGGP . . . . .	110
5.20	DGGP : Améliorations par rapport à OGGP . . . . .	110
5.21	Configuration réseau avec liens locaux . . . . .	112
5.22	Matrice de communication pour la redistribution en régime permanent . . . . .	115
5.23	Routes optimales . . . . .	116
5.24	Solution du problème multi-objectif . . . . .	117
5.25	Construction des nouvelles routes . . . . .	118
5.26	Graphes de communications locales et distantes . . . . .	119
5.27	Ordonnancement des communications sur la dorsale . . . . .	121
5.28	Ordonnancement final . . . . .	122
5.29	Routes et ordonnancement non réalisable . . . . .	122
5.30	Phase d'initialisation permettant d'atteindre le régime permanent . . . . .	123

## Liste des Algorithmes

1	Algorithme d'estimation . . . . .	59
2	Algorithme d'extension en graphe régulier sur les poids . . . . .	65
3	Algorithme GGP . . . . .	70
4	Algorithme par multigraphe . . . . .	73
5	Algorithme pour l'extraction d'un couplage parfait de poids minimal maximum . . . . .	81
6	Algorithme OGGP . . . . .	83
7	Heuristique sur les poids . . . . .	85
8	Algorithme de partage des sommets . . . . .	103

*LISTE DES ALGORITHMES*

---

9	Algorithme M2 . . . . .	104
10	Algorithme de regroupement de deux ordonnancements . . . . .	121

# Chapitre 1

## Introduction

### 1.1 Contexte

*De l'atome au noyau, du noyau aux particules, la physique de ce siècle a effectué une abrupte plongée vers l'infiniment petit. Ses instruments ont, eux, connu la progression inverse : de plutôt petits, ils sont devenus géants. C'est au cours des années trente que de nouveaux moyens techniques ont fait éclater les murs des laboratoires de physiques.*

*Jeffrey Hughes*

#### 1.1.1 Couplage de code

La citation de Jeffrey Hughes qui ouvre cette introduction nous rappelle à quel point l'évolution des instruments d'expérimentation scientifique a été spectaculaire tout au cours du siècle dernier. En physique des particules notamment, en faisant "éclater les murs des laboratoires", les instruments de recherche ont eu également comme effet de faire exploser les coûts financiers des différentes expérimentations. On songera par exemple au projet Manhattan [5] de recherche nucléaire aux États-Unis d'un coût total équivalent à 20 milliards de dollars.

Une telle évolution a conduit, avec le développement des techniques informatiques, à privilégier en premier lieu les simulations numériques, moins onéreuses, en particulier lors des phases initiales de nouveaux développements. De nombreux programmes de simulations numériques ont ainsi vu le jour en cosmologie, dynamique des fluides, biochimie, écologie, . . .

Depuis quelques années, les besoins de simuler des systèmes complexes dont le fonctionnement est dépendant de plusieurs lois physiques a amené le développement des applications de couplage de code. Une telle application vise à regrouper ensemble différents programmes ou codes agissant chacun sur un aspect particulier d'un problème. Par exemple, dans [62] Shankaran et al. étudient un couplage de code modélisant le fonctionnement d'une turbine d'avion. Un premier programme, nommé TFLO est utilisé pour une simulation des déplacements des différentes particules à l'aide d'une simulation de dynamique des fluides, tandis qu'un second programme, NCC, est chargé de la simulation de la combustion. TFLO et NCC s'échangent des données sur l'état global de la simulation à intervalles réguliers, les résultats de chacun étant nécessaire à tous pour la suite de la simulation.

La procédure d'échange de données s'effectue à l'aide d'une redistribution de données.

### 1.1.2 Parallélisme et grilles de calcul

Les simulations numériques se sont rapidement avérées de grandes consommatrices de puissance de calcul, au point de nécessiter dans bien des cas le recours au calcul parallèle.

Les développements des différentes techniques de parallélisation, ainsi que des différentes machines parallèles ont donc permis la mise en place de simulations de plus en plus complexes. En particulier, les progrès dans le domaine des interfaces matérielles de communication de plus en plus rapides, couplés à l'amélioration des performances des ordinateurs de bureau, ont permis l'apparition des grappes d'ordinateurs, couramment utilisées de nos jours. Ces grappes, composées d'un ensemble d'ordinateurs bas de gamme, reliés en réseau local ont ainsi abaissé le coût financier des calculateurs parallèles et démocratisé (dans une certaine mesure) leur accès.

Actuellement, les progrès dans les domaines de réseaux longues distances permettant d'atteindre des vitesses de communication de l'ordre de plusieurs Gbit/s ont rendu possible la mise en commun des ressources de calcul parallèle des différentes institutions de recherche scientifique au travers du réseau, formant ainsi la base des grilles de calcul.

Ces grilles, de topologies complexes et variées posent encore à l'heure actuelle de nombreux problèmes de recherche sur de nombreuses thématiques distinctes : sécurité, efficacité, résistance aux pannes, simplicité de programmation, ... L'étude des grilles de calcul nécessite donc le choix d'un problème particulier à étudier.

Nous proposons d'étudier une facette d'un problème d'exécution d'un couplage de code : nous considérons ici deux codes distincts requérant chacun une grappe distincte pour son exécution, chaque code étant un programme parallèle nécessitant une puissance de calcul élevée. Les deux programmes communiquent fréquemment à travers un réseau à haut débit permettant, en principe, d'atteindre un temps total de calcul acceptable. Une telle configuration est illustrée sur l'exemple de la figure 1.1.

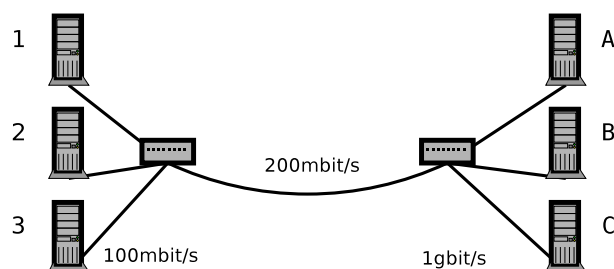


FIG. 1.1 – Exemple de configuration réseau

Nous étudions dans cette thèse comment effectuer au mieux les communications de chaque redistribution de données de ce couplage de code en tenant compte des caractéristiques du réseau.

## 1.2 Motivation

Différentes raisons motivent l'étude d'un tel problème.

Tout d'abord, rappelons que pour tout programme parallèle, une part plus ou moins importante du temps d'exécution est consacré aux transferts de données. Les temps de transferts ayant tendance à augmenter avec le nombre de processeurs alloués à l'exécution (et ce, à cause de la complexification du matériel d'interconnexion), un programme nécessitant de longs transferts de

données a souvent peu d'intérêt à être parallélisé au delà d'un certain nombre de processeurs. Les communications peuvent donc former une limite au parallélisme. Dans un environnement comme une grille, fortement dépendant du réseau, les temps de communications peuvent donc s'avérer cruciaux.

Nous cherchons donc à optimiser le temps de d'exécution de la procédure de communication complexe que représente une redistribution de données. Jusqu'à présent, la redistribution de données a été essentiellement étudiée dans le cadre de réseaux locaux sur lesquels les délais d'initialisations des différentes communications peuvent être relativement faibles et les liens entre sources et destinations relativement directs.

Nous entendons ici par délai d'initialisation la latence réseau augmentée de la durée nécessaire à la couche logicielle pour la mise en place d'une communication. L'ajout de différentes couches logicielles pour la mise en place des intergiciels sur les grilles de calcul a en effet une tendance à accroître les temps d'initialisation des différentes communications. La prise en compte de ce délai d'initialisation est une contribution importante de cette thèse. Elle nous permet de représenter d'une manière plus juste les communications réelles et d'atteindre ainsi des performances plus élevées.

Lorsque les chemins entre sources et destinations comprennent des routeurs, réémettre des paquets en cas de congestion du réseau peut avoir un coût important. Ce coût peut s'avérer d'autant plus élevé que la version la plus couramment utilisée du protocole TCP ne gère pas forcément au mieux une congestion sur un réseau à haut débit. En nous plaçant à un niveau plus élevé que la couche réseau, nous sommes en mesure de tirer partie des informations que possède l'application (ou l'intergiciel) sur les communications qu'elle a à effectuer pour éviter totalement la congestion du réseau.

Notons enfin, que les mécanismes d'ordonnancement que nous présentons peuvent également s'avérer utile pour limiter la quantité de bande passante consommée par une application ou le nombre de flux exécutés en parallèle. En maîtrisant les communications effectuées il est ainsi possible d'éviter d'accaparer une part trop importante des ressources réseau.

### 1.3 Organisation de la thèse

Nous commençons chapitre 2 par ré-introduire les notions de calcul parallèle, grappes d'ordinateurs et grilles de calcul en mettant l'accent sur le rôle joué dans chaque cas par les mécanismes de transferts de données. Nous présentons diverses visions des grilles de calcul, en indiquant comment certaines visions des grilles impliquent l'utilisation massive de transferts de données. Nous introduisons également le concept de redistribution de données, liée au développement des systèmes à mémoire distribuée, ainsi que différentes études visant à l'optimisation de redistributions locales.

Nous présentons ensuite, chapitre 3 une description précise du problème principal que nous étudions dans cette thèse : quels sont les paramètres précis attendus, quelles sont les différentes possibilités d'exécution des communications. Nous montrons également qu'il est possible d'obtenir des gains sur le temps de transferts des données, y-compris dans le cas théorique où le réseau répartit parfaitement la bande passante disponible entre chaque flux, et ce, sans aucune perte ou ré-émission de paquets.

Ce problème de redistribution est ensuite modélisé par un problème mathématique : le problème KPBS (*K-Preemptive Bipartite Scheduling*). Afin d'éviter l'engorgement des différents liens du réseau, nous cherchons ici à limiter le nombre de communications exécutées simultanément en les ordonnant. Nous montrons qu'il est possible de modéliser le motif de données à

redistribuer par un graphe biparti valué et un ordonnancement des communications par étapes en un ensemble de couplages valués, chacun d'entre eux représentant une étape différente. Le problème KPBS consiste ainsi à trouver une décomposition optimale de l'ensemble des arêtes du graphe de communication en un ensemble de couplages.

Afin de développer un algorithme résolvant KPBS, nous passons alors en revue différents problèmes similaires ou liés :

- des problèmes d'une thématique proche ;
- des problèmes modélisés d'une manière similaire ;
- des problèmes pour lesquels les méthodes de résolution présentent des similitudes à celles que nous utilisons ici.

Certaines idées seront ainsi reprises et modifiées pour s'adapter au problème que nous étudions.

Enfin, avant de passer à la résolution du problème posé, nous vérifions dans un premier temps l'adéquation du modèle à la réalité, à l'aide d'une série d'expériences et de simulations.

Le chapitre 4 suit le développement de deux algorithmes d'approximation résolvant le problème KPBS. Nous partons d'algorithmes existant, résolvant un problème plus simple : le problème PBS, et montrons comment étendre ces algorithmes pour prendre en compte toutes les contraintes de KPBS.

Nous étudions le premier algorithme proposé, GGP, et montrons que GGP est un algorithme d'approximation, garantissant pour tout ordonnancement un temps inférieur à  $\frac{8}{3}$  fois le temps optimal. Néanmoins, sur certains exemples, la qualité des résultats peut atteindre un facteur 2 par rapport à un temps optimal. Nous avons donc cherché à modifier l'algorithme dans l'espoir de diminuer son ratio d'approximation. Nous présentons ainsi l'algorithme OGGP, extension directe de GGP, présentant un bon comportement sur toute les familles d'exemples problématiques trouvées.

Pour chaque algorithme, nous effectuons un travail important d'analyse :

- du ratio d'approximation garantissant une qualité minimale sur les résultats ;
- de la complexité au pire cas.

Notons au passage qu'un grand nombre de symboles mathématiques sont utilisés pour toutes les parties théoriques. L'annexe A résume, dans un souci de clarté, toutes les notations utilisées.

Finalement, nous concluons le développement et l'analyse des propriétés de GGP et OGGP par une série de simulations et d'expérimentations réelles. Nous présentons différentes configurations expérimentales et montrons dans quelle mesure une approche ordonnancée peut s'avérer judicieuse.

Nous étudions ensuite, chapitre 5, des problèmes légèrement différents du problème KPBS :

- dans un premier temps, nous étudions une extension du problème KPBS, appelée d-KPBS, permettant d'effectuer une redistribution sur des grappes disposant d'interfaces réseau hétérogènes. Nous proposons une modification de l'algorithme OGGP, nommée DGGP, résolvant le problème posé. Nous montrons que DGGP est également un algorithme d'approximation et poursuivons par une étude théorique accompagnée de simulations permettant de vérifier une bonne qualité des résultats obtenus.
- dans un second temps, nous étudions une variation du problème de redistribution en ajoutant la possibilité d'utiliser un réseau local rapide lors de la redistribution. Ce réseau permettant de répartir la charge des données à envoyer entre les divers émetteurs, ce type de configuration peut permettre de diminuer effectivement les temps de transferts. Nous montrons que ce problème introduit des notions de routage, devenant ainsi particulièrement complexe à résoudre. Nous simplifions donc le problème étudié ici en négligeant les temps de latence, et en effectuant un ensemble de redistributions en régime permanent : plusieurs

redistributions aux motifs de données à redistribuer identiques. Nous introduisons alors un algorithme asymptotiquement optimal dans le cas du régime permanent, et nous montrons qu'il est également un algorithme d'approximation dans le cas où une seule redistribution est à effectuer.

Enfin, nous présentons au chapitre 6 les conclusions et perspectives de ces travaux.



## Chapitre 2

# Grilles de calcul et redistribution de données

### Sommaire

---

<b>2.1</b>	<b>Calcul parallèle</b> . . . . .	<b>19</b>
<b>2.2</b>	<b>Grilles de calcul</b> . . . . .	<b>20</b>
2.2.1	Distribution de ressources . . . . .	21
2.2.2	Agrégation de ressources . . . . .	21
2.2.3	Récupération de ressources . . . . .	22
2.2.4	Mise en commun de ressources . . . . .	23
<b>2.3</b>	<b>Redistributions de données</b> . . . . .	<b>24</b>
2.3.1	Couplage de code . . . . .	24
2.3.2	Optimisations de redistributions . . . . .	25

---

En 1941 la construction par l'équipe de Konrad Zuse du Z3 [8], souvent considéré comme la première machine réellement programmable, marque le début du développement des ordinateurs. Cette date marque également l'apparition chez les informaticiens et les scientifiques d'une quête sans fin vers une puissance de calcul sans cesse accrue. Chaque nouvelle génération de matériel informatique ne semble en effet qu'augmenter les besoins, en permettant le développement d'applications de plus en plus lourdes et complexes. Bien que la densité des transistors des processeurs, et de ce fait la puissance de calcul des ordinateurs double tous les 1,5 ans (suivant la très célèbre loi de Moore), certains logiciels nécessitent des ressources bien plus grandes que celles fournies par des machines classiques. Afin de satisfaire de telles demandes, il est nécessaire de recourir au calculateurs parallèles, et depuis quelques années, aux grilles de calculs.

### 2.1 Calcul parallèle

Le néophyte en informatique considère souvent (à tort) la puissance fournie par un ordinateur comme étant plus ou moins égale à sa fréquence d'horloge. En réalité plusieurs paramètres rentrent en ligne de compte pour estimer la puissance réelle d'une machine. La fréquence indique combien d'opérations peuvent être lancées chaque seconde, mais il est également nécessaire de prendre en compte la durée (en nombre de cycles) de chaque opération, ainsi que le nombre d'opérations que l'on peut exécuter simultanément. Ainsi sur un exemple simpliste où chaque opération ne prend qu'un seul cycle d'horloge, une machine calibrée à 1Ghz peut effectuer un

milliard d'opérations par seconde si elle n'autorise qu'une seule instruction simultanément ou deux milliards d'opérations par seconde si il lui est possible d'en exécuter deux simultanément. C'est cette idée simple de réaliser plusieurs choses en même temps (plutôt que de d'augmenter la vitesse de traitement) qui est au cœur du calcul parallèle.

Les débuts des architectures parallèles ont eu lieu dans les années 1960 avec le projet Solomon de Westinghouse. L'idée était la suivante : un processeur central contrôle un grand nombre de co-processeurs arithmétiques et envoie les mêmes instructions à chacun d'entre eux. Chaque co-processeur effectue donc le même travail que les autres, en parallèle, mais sur des données différentes. On voit ici apparaître le principe qui régit les architectures SIMD (*Single Instruction Multiple Data* : une instruction, plusieurs données). Le projet Solomon a été abandonné, mais ses idées ont été reprises par la suite par une équipe de l'université de l'Illinois, et ont mené au développement de la première machine vectorielle : l'ILLIAC IV en 1972. Il s'agissait alors de la machine la plus rapide au monde, disposant d'une puissance de 150 Mflops.

En 1982, l'entreprise américaine Cray commercialise le Cray X-MP [1], la première machine multi-processeurs. En effet, le X-MP contenait de un à quatre processeurs communiquant entre eux à travers une banque mémoire partagée. Chaque processeur disposait, de plus, de plusieurs unités de calcul arithmétique permettant de paralléliser les calculs à l'intérieur même de chacun d'entre eux. On commence ici à distinguer l'évolution des architectures parallèles : interconnecter des ressources de calcul de plus en plus puissantes, mais également de plus en plus distantes.

Les progrès des technologies d'interconnexion, notamment des interfaces réseau ont permis au début des années 1980 le développement de machines parallèles appelées grappes d'ordinateurs, constituées de plusieurs ordinateurs reliés entre eux par un réseau local. ARCNET en 1977, puis VAXcluster [50] quelques années plus tard vont introduire puis populariser ce type d'architecture en proposant des performances certes faibles au niveau des transferts de données, mais bonnes en terme de calcul, et surtout, un coût bien plus bas que les supercalculateurs classiques. A l'heure actuelle la plupart des grappes sont composées d'ordinateurs PC ou Macintosh standards, les performances élevées qu'elles sont capables de fournir étant dues à des tailles étendues : certaines grappes pouvant aller jusqu'à contenir plusieurs milliers de machines.

## 2.2 Grilles de calcul

Lors de la dernière décennie, les progrès dans les domaines des réseaux à longue distance ont permis d'envisager de distribuer les calculs à effectuer sur des grappes d'ordinateurs distantes reliées entre elles par internet, ou par des réseaux hautes performances tels VTHD [6]. On assiste ici encore à une augmentation des performances des communications permettant de regrouper des ressources de plus en plus éloignées, de manière similaire au mouvement qui a permis de mettre en place des grappes d'ordinateurs, mais ici à plus large échelle. Un tel ensemble de ressources ainsi mises en commun à travers le réseau forme une grille de calcul [36].

La définition exacte d'une grille, aussi bien au niveau du type de matériel utilisé (hétérogène / homogène, fiable / faillible), de l'organisation du réseau (fortement hiérarchique / faiblement hiérarchique), ou de la gestion de la sécurité varie fortement dans la littérature. Il existe de plus un grand nombre d'applications très différentes, dépendant fortement des paramètres choisis. Il est ainsi difficile de comparer des applications pair-à-pair avec des applications de calculs parallèles classiques.

Plutôt que de proposer une définition précise, mais sujette à caution, d'une grille de calcul, nous proposons de regarder quatre optiques différentes d'utilisation d'une grille. Nous verrons que les logiciels utilisés, les méthodes de développement ainsi que les différentes problématiques

associées à la notion de grille varient en fait fortement suivant les buts recherchés par l'utilisateur final. Aucun des choix proposés ici ne se veut plus pertinent qu'un autre ; nous considérons que toute grille poursuit en réalité l'ensemble des objectifs proposés ici.

### 2.2.1 Distribution de ressources

Le concept initial des grilles informatiques dérive du concept de réseau électrique (*electric grid*). D'une manière similaire à ceux-ci, les grilles informatiques sont imaginées comme un moyen de disposer de puissance de calcul (au lieu de puissance électrique) de manière pratique et transparente. Ici l'utilisateur se branche sur la grille, et peut utiliser directement ses logiciels (même très gourmands) sans avoir et s'occuper personnellement de chercher des ressources à même de satisfaire sa demande, les réserver, envoyer ses données sur place, préparer un script d'exécution, et autres tâches qui peuvent rendre très fastidieuse une exécution distante.

Bien entendu, dans la réalité, un seul ordinateur est déjà un objet relativement complexe, une grappe d'ordinateurs un objet difficile à utiliser et une grille de ressources hétérogènes est donc particulièrement délicate à utiliser. La solution à ce problème consiste en l'utilisation d'intergiciels pour développer et/ou soumettre des tâches sur la grille. Un intergiciel est un ensemble d'outils et de bibliothèques de développement qui ont pour tâches de simplifier le travail de l'utilisateur et de gérer les ressources de manière efficace.

Il existe différents intergiciels, de tailles variées, plus ou moins sophistiqués. L'ensemble des tâches prises en charge par un intergiciel est variable et peut comprendre : la détection des machines utilisables, le placement et l'ordonnancement des tâches de calcul, l'observation du réseau, la migration de tâches, la gestion des pannes, la gestion des données, la facturation du temps de calcul, l'exécution des communications, ... A l'heure actuelle, l'intergiciel le plus connu est Globus [35], développé par Globus Alliance, une collaboration internationale. Globus est sans doute l'intergiciel fournissant le plus grand nombre de services différents, mais il reste relativement délicat à mettre en place, et évolue rapidement.

On peut également citer des intergiciels plus faciles d'accès, mais permettant un usage plus restreint : les intergiciels de type client-agent-serveur par exemple. Leur principe de fonctionnement est le suivant : les serveurs de calcul s'enregistrent auprès d'un ou de plusieurs agents, chargés de répertoriés les ressources disponibles. Lorsqu'un client a une tâche à soumettre, il contacte un agent qui lui renverra l'identifiant de la machine ou soumettre sa tâche. Le client contacte alors directement le serveur. Plusieurs intergiciels utilisent ce modèle : Diet [2], Net-solve [21], Ninf [43].

### 2.2.2 Agrégation de ressources

Une autre manière d'envisager la grille consiste à la regarder comme un moyen de réaliser des calculs jusque-là impossibles à effectuer en agrégeant aussi bien les ressources CPU que les ressources mémoires (RAM et disques). Les exemples les plus connus d'applications atteignant cet objectif sont les applications pair-à-pair.

De telles applications sont surtout connues du grand public pour le transfert de fichiers distribué sur internet. On peut ainsi citer des applications comme Kazaa, Gnutella, Bittorrent, ... Elles sont capables de performances remarquables : par exemple, en janvier 2005 la bande passante consommée par Bittorrent était estimée par Cachelogic, une firme anglaise d'analyse de trafic, à un tiers de la bande passante totale d'internet, bien plus que la bande passante consommée par le web.

On peut également citer un certain nombre de projets de calculs distribués à travers internet, utilisant la puissance de calcul de machines de particuliers : SETI@home, folding@home, ... Le projet SETI@home vise à distribuer, à travers internet, des calculs de recherches de motifs dans des signaux radio sur des machines de particuliers. Le fonctionnement est le suivant : Un client SETI@home demande un jeu de données à un serveur central, effectue l'ensemble des calculs et renvoie les résultats, puis recommence la procédure. Le thème amusant du projet (la recherche de signaux extra-terrestres) attiré l'attention de plus de 4 millions de personnes, permettant d'obtenir une puissance de calcul de plus de 60 teraflops, approximativement la puissance dégagée par la machine la plus performante du TOP500 à l'heure actuelle (juillet 2005).

Bien entendu, il est clair que de telles performances ne peuvent être atteintes que dans le cas de problèmes simples où les communications ne représentent pas une charge élevée par rapport aux calculs. Il s'agit la plupart du temps de programmes data-parallèles : on dispose d'une masse importante de données à traiter, pouvant être scindés en plusieurs ensembles de données traitées indépendamment.

### 2.2.3 Récupération de ressources

Une troisième approche du concept des grilles de calculs consiste à récupérer une puissance inutilisée. Ici, de manière similaire au développement des grappes d'ordinateurs, le coût économique est un facteur important. L'idée principale est la suivante : certaines ressources informatiques ne sont pas utilisées en permanence, les besoins variant en fonction du temps. En reliant entre elles de telles ressources il devient donc possible d'équilibrer la charge de toutes les machines pour peu que les besoins de chacun ne tombent pas sur la même période.

Harmony [56] est un exemple d'intergiciel créé dans une telle optique. Harmony a été conçu pour fonctionner sur des machines de bureau standard. En effet, la plupart des ordinateurs utilisés à des fins de bureautique sont complètement sous-utilisés en journée, et souvent complètement inutilisés durant la nuit. En agrégeant les ressources ainsi gaspillées sur tous les postes d'une entreprise, il est possible d'obtenir une puissance non négligeable, et ce, à un coût relativement réduit.

Au niveau scientifique, la problématique diffère de celle posée par une grille visant à agréger le plus de puissance possible. Dans un cadre comme celui-ci, il devient plus important d'être un mesure de gérer efficacement les ressources plutôt que de terminer une application donnée au plus tôt. Ceci se ressent en particuliers sur des systèmes d'ordonnancement de tâches où le critère classique de Makespan peut ainsi se voir préférer d'autres critères visant plus à optimiser l'utilisation des ressources [20].

De plus, effectuer des transferts de données à longue distance, a ici un sens. Si les performances brutes sont visées, il est clair qu'il est vital d'éviter au maximum de transférer des données. Néanmoins, avec l'introduction d'une notion de coût il peut être plus rentable de lancer des calculs sur des sites lointains mais libres plutôt que sur des sites locaux, surchargés. Certains intergiciels visent de plus à mettre en place des mécanismes de facturation des calculs. Il serait donc possible de disposer de différents sites de calcul proposant des tarifs différents, et donc parfaitement envisageable de préférer des sites plus distants mais moins chers.

En poursuivant ce raisonnement jusqu'au bout, il est possible d'imaginer toute une classe d'applications étant capables de migrer au cours du temps. On peut ainsi autoriser des applications à s'exécuter à un endroit où une certaine puissance de calcul est disponible, puis les déplacer à un autre endroit si des applications plus prioritaires se mettent à requérir des machines. Une autre possibilité serait de concevoir des applications se déplaçant de sites en sites

sur Terre, afin de rester exécutées uniquement la nuit. On voit ici encore quelque chose qui n'a pas de sens au niveau performances à cause des surcoûts liées aux transferts de données, ainsi que mécanismes de reprises des calculs, mais qui peut tenir la route si des critères d'optimisation de l'utilisation des ressources sont utilisés.

Enfin, on peut noter que cette idée de migration de programmes parallèles se retrouve jusque dans les romans de science-fiction : Greg Egan [31] décrit ainsi un système de grilles où les calculs seraient facturés en fonction d'un prix proposé par le fournisseur de puissance. Dans son roman on assiste ainsi à des migrations d'applications distribuées, celles-ci se déplaçant vers les sites de calcul proposant les tarifs les plus intéressants.

#### 2.2.4 Mise en commun de ressources

Une dernière manière proposée pour l'utilisation d'une grille de calcul est la mise en commun de ressources hétérogènes.

Chaque laboratoire désirant effectuer des calculs scientifiques nécessite plusieurs type de ressources :

- un ensemble de machines où exécuter les calculs ;
- différents jeux de données en entrée (ou en sortie) des calculs ;
- les logiciels les réalisant.

À l'heure actuelle les projets de grilles de calculs se développent autour de laboratoires disposant déjà localement de toutes ces ressources. Le passage à une grille vise donc à définir de quelle manière celles-ci vont être interconnectées.

Les trois approches présentées précédemment sont très liées à la notion de puissance de calcul, et donc à la manière d'interconnecter entre elles les ressources *matérielles*. Pour cette dernière approche, nous considérons essentiellement les ressources que sont les logiciels et les données.

Les scientifiques du monde entier échangent depuis longtemps déjà logiciels et données. L'apparition d'internet a décuplé les possibilités de transferts de connaissances et c'est dans la continuation de cet effort que ce situe ici notre problématique.

Les logiciels de calcul scientifiques posent souvent différents problèmes. Souvent les pré-requis logiciels (dépendances) pour l'installation d'un code un peu lourd sont importants. Un programme peut dépendre d'une longue liste de bibliothèques, d'une version spéciale du compilateur, ainsi que de toute une suite d'outils non portables et dépendants d'un système d'exploitation particulier. Il faut ajouter à cela l'impossibilité pour certains logiciels de fonctionner sur certains matériels et systèmes et la possibilité que certains logiciels soient propriétaires et donc interdits à la copie. Ici l'échange de logiciels ne pose pas forcément de problème, mais la mise en place et la maintenance de ceux-ci peut se révéler coûteuse en temps et délicate. Il peut donc être particulièrement pratique et rentable de monter une infrastructure de grille commune à plusieurs institutions afin que chacun puisse disposer d'un accès aux ressources logicielles des autres.

En ce qui concerne les données, les grilles de calculs peuvent également permettre de résoudre de nombreux problèmes. Certaines données nécessitent ainsi un accès contrôlé : les données médicales par exemple. Il est clair qu'ici une authentification gérée au niveau de la grille pourrait permettre un accès plus large à de telles données tout en permettant le contrôle. Un autre problème concerne les données de taille très importante. Difficiles à stocker, dupliquer et transférer leur gestion par des sites spécialisés dans le stockage pourrait permettre un accès plus vaste, mais également des performances accrues. Enfin, il est important de noter que de plus en plus de données sont générées de manière distribuée sur de nombreux sites et nécessitent donc des infrastructures de stockages particulières.

Les problématiques scientifiques associées à ce type de grilles sont nombreuses et concernent aussi bien les performances brutes que la publication des différents services proposés par chaque entité, la recherche dans ces services, ainsi que leur interconnection.

Il est encore une fois important de noter qu'avec une telle optique d'utilisation des différentes ressources, des choses qui n'ont que peu de sens dans une optique d'optimisation de performances comme par exemple des transferts de données à longue distance peuvent se révéler très fréquentes et cruciales pour l'utilisateur.

## 2.3 Redistributions de données

Nous avons présenté aux sections précédentes l'importance que peuvent prendre les performances des communications pour une grille de calculs, aussi bien à un niveau pratique que fondamental : d'une part les progrès des technologies de communications sont un facteur important de progrès en parallélisme et calcul distribué, d'autre part il existe toute une classe d'applications susceptibles d'effectuer des transferts de données importants à longue distance.

Plusieurs travaux scientifiques issus des recherches en réseau visent à diminuer les temps de transferts suivant différentes approches. Il existe de nombreuses méthodes d'optimisation de la taille des fenêtres TCP aussi bien manuellement, que dynamiquement [34, 33], en fonction de la charge réseau. D'autres méthodes [57, 63] consistent à utiliser plusieurs sockets sur une interface réseau afin de bénéficier de plus de bande passante sur des réseaux chargés ou mal configurés. Enfin, il est également possible d'envisager la compression à la volée des communications [47], ce qui présente l'avantage de bien fonctionner sur des réseaux lents, mais uniquement sur des jeux de données compressibles.

Nous nous intéressons ici uniquement à des optimisations de plus haut niveau. Plutôt que de considérer des communications simples, en point à point, nous étudions ici un type de communications parallèles particulières : les redistributions de données. Ce type de communications est couramment utilisé dans les applications de couplage de code (*code coupling*).

### 2.3.1 Couplage de code

Le couplage de code consiste à assembler deux programmes différents en les modifiant légèrement pour qu'ils échangent régulièrement des informations. Ce type de logiciel est utilisé principalement pour des simulations scientifiques de haut niveau mettant en jeu plusieurs modèles physiques différents. Le projet Hydrogrid [4] est un bon exemple de ce type de programmes. Ce projet vise à simuler le déplacement de fluides et de solutés en milieu souterrain afin d'étudier le mouvement de produits dangereux ou gênants dans une nappe phréatique (sel, déchets toxiques ou nucléaires).

Simuler un tel phénomène est un problème complexe, qui nécessite plusieurs codes de simulations différents :

- un premier code de transport, traitant les équations sur la concentration et la masse volumique ;
- un second code d'écoulement, traitant les équations sur la pression et la vitesse ;
- un code de chimie, traitant les équations sur la concentration.

Deux applications différentes sont proposées : un couplage des codes d'écoulement et de transport, et un couplage des codes de transport et de chimie. Dans le cas du couplage écoulement-transport, les deux différentes simulations (parallèles) échangent régulièrement des informations : le code d'écoulement transférant au code de transport des informations sur la vitesse, tandis que le code de transport lui renvoie des informations de concentration.

Il existe à l'heure actuelle de nombreux projets de logiciels (bibliothèques et intergiciels) visant à permettre d'interfacer rapidement ensemble plusieurs codes différents en une application de couplage de code. Les projets les plus anciens ont été réalisés avec souvent un type particulier d'applications en vue.

CUMULVS [39] par exemple, développé à Oak Ridge National Laboratory. Cet intergiciel a pour but de permettre d'interfacer un code parallèle de simulation numérique avec un ou plusieurs clients de visualisation. Les données étant stockées sous forme de tableaux denses ou de particules associées à des régions de l'espace. CUMULVS est donc limité par ses choix de conceptions à un type de programmes particulier. PAWS [49] est un autre environnement, plus générique que CUMULVS, mais qui impose néanmoins de traiter les données sous forme de tableaux décomposés de manière rectilinéaire. Cette fois-ci, il y a moins de contraintes sur le type d'applications, mais la gestion des données est par contre plus limitée.

Plus récemment, différents travaux visent à fournir une interface de redistribution plus générique en s'appuyant sur la notion de composants logiciels. Ainsi, le projet CCA MxN [52, 15], issu des développements de PAWS et CUMULVS, poursuit cet objectif. Il est néanmoins à noter qu'à l'heure actuelle il ne gère que des types de données réguliers. PACO++ [27], développé à l'IRISA de Rennes, utilise lui des composants formés d'objets parallèles. Le but de ce projet est de permettre d'encapsuler chaque code (parallèle ou non) d'un couplage dans un composant afin de permettre un interfaçage clair et propre (grâce aux notions de programmation orientée objet) entre les différents composants. Cet intergiciel gère en outre l'échange de données entre les différents composants parallèles par une redistribution pouvant être contrôlée précisément. Une partie de nos travaux d'ordonnancement a été par ailleurs intégrée dans ce projet.

### 2.3.2 Optimisations de redistributions

#### Présentation

La redistribution de données est une opération complexe visant à transférer un jeu de données situé sur un ensemble de nœuds source vers un autre ensemble de nœuds destination. On peut noter que ces deux ensembles ne sont pas forcément disjoints. Aurélien Esnard a proposé [32] un découpage de cette opération en plusieurs phases :

1. description des données distribuées,
2. génération des messages,
3. calcul d'un ordonnancement,
4. exécution des transferts.

La première étape d'une redistribution consiste à fournir une description des données distribuées. Cette description fournit deux types d'informations différents : sur quel nœud sont placées quelles données, et pour un nœud donné, comment sont placées les données en mémoire. Beaucoup de travaux existant se restreignent ici au cas de données distribuées par blocs cycliques, mais en réalité il est possible d'envisager des redistributions pour tout type de placements, à condition d'être en mesure de les décrire précisément.

La seconde étape consiste à calculer quelles sont les données à envoyer entre deux nœuds particuliers, et à générer ensuite chaque message (en tirant parti des informations fournies à la première étape). Ce problème est loin d'être simple, et ce, pour deux raisons : premièrement le nombre de nœuds source et destination n'est pas forcément le même, ce qui implique une réorganisation du placement des données. De plus le placement cherché peut être complexe : par blocs cycliques, ou complètement irrégulier, par exemple lors de l'utilisation de matrices creuses.

L'exemple de la figure 2.1 illustre ainsi le placement des données en mémoire, ainsi que chacun des transferts à effectuer dans le cas d'une redistribution d'un ensemble de 3 nœuds utilisant une distribution en blocs cycliques de taille 3 vers 4 nœuds utilisant un placement des données en blocs cycliques de taille 2.

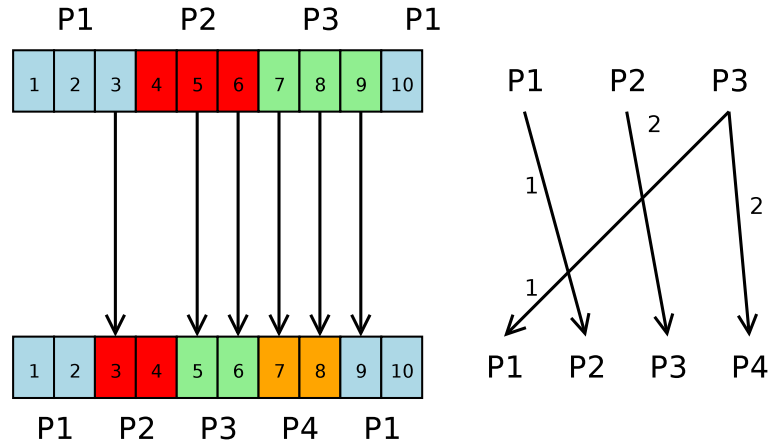


FIG. 2.1 – Exemple de redistribution de blocs cycliques à blocs cycliques

Pour résoudre ce problème, Aurélien Esnard a proposé [32] une méthode basée sur le principe suivant : l'ensemble des données que le processeur source 1 doit envoyer au processeur destination A est calculé en prenant l'intersection de l'ensemble des données présentes sur 1 et de l'ensemble des données requises sur A.

Dans le cas d'une redistribution simple, il est possible une fois les communications à effectuer connues de passer directement à l'exécution des transferts à proprement parler. Néanmoins, dans certains cas, l'exécution simultanée de tous les transferts peut se révéler un problème aussi bien pour les performances réseau que pour le programmeur. Afin de résoudre ces problèmes, il est nécessaire d'ajouter une étape supplémentaire en ordonnant les communications.

Lorsque l'ensemble des messages à effectuer est connu, il est possible de passer au calcul d'un ordonnancement. L'ordonnancement des communications vise, en trouvant un ordre pour chacune d'entre elles, à réaliser différents objectifs, dépendant des problèmes considérés par la communauté scientifique travaillant sur ce problème. Il s'agit souvent de limiter le nombre de communications simultanées (pour des raisons de performance) ou de limiter à une seule le nombre de communications simultanées sur chaque interface réseau, de manière à permettre un traitement séquentiel des communications.

Nous nous intéresserons, par la suite, essentiellement à ce problème de calcul d'un ordonnancement (et ce, pour différents cas de redistributions), sans nous préoccuper du calcul préalable du schéma de communications.

Enfin, une fois l'ordonnancement calculé, il ne reste plus qu'à passer à l'exécution des transferts en eux-mêmes. La encore, plusieurs méthodes ont été développées nécessitant une infrastructure plus ou moins complexe suivant l'approche choisie.

### Redistribution de données distribuées par blocs cycliques

La redistribution de données a été essentiellement étudiée lors de travaux de développement d'environnements de programmation sur des systèmes à mémoire distribuée. HPF [44] ou Sca-

LAPACK [17] fournissent ainsi chacun des mécanismes permettant de changer le placement des données. Par exemple, la directive *redistribute* du langage HPF permet de déplacer des données au sein d'un même programme parallèle. Ici, le but de l'opération est d'effectuer une réorganisation de la distribution des données sur les différents processeurs afin d'éviter par la suite un nombre trop important d'accès à des données non locales. Par exemple un algorithme comme une intégration par directions alternantes (*Alternating Direction Integration*) se décompose en deux phases distinctes [45] : dans un premier temps l'algorithme balaye un tableau à deux dimensions par lignes, et dans un second temps par colonnes. Une distribution où les lignes sont locales à chaque processeur, et les colonnes distribuées élimine toutes les communications dans la première phase, et inversement une distribution où toutes les colonnes sont locales à chaque processeur celles de la seconde phase. Il est important de noter également que les bibliothèques de calcul scientifique sont souvent optimisées pour une distribution particulière des données.

Les travaux initiaux sur la redistribution prennent donc essentiellement en compte les redistributions locales, souvent sans changer de jeu de processeurs. De plus seules des données distribuées en blocs cycliques sont considérées. Nous présentons ici un aperçu des recherches effectuées dans ce cadre.

Les langages de programmation data-parallèles comme HPF [44], Fortran D [37], Vienna Fortran [23], HPC [29] fournissent des directives de compilation permettant de spécifier la distribution des données de tableaux sur les différents processeurs utilisés lors de l'exécution. On peut distinguer deux types de distributions : régulières ou irrégulières. Les distributions irrégulières étant peu fréquemment utilisées, les distributions régulières ont fait l'objet d'études plus approfondies, et notamment les redistribution de données sont essentiellement étudiées dans la littérature entre deux distributions régulières de données.

On subdivise généralement les distributions régulières en trois catégories : cycliques, par blocs ou bloc-cyclique. Une distribution cyclique des données est effectuée en répartissant les données une par une sur chaque processeur, en recommençant au premier processeur une fois arrivé au dernier. Une redistribution par blocs est effectuée en divisant les données en blocs de tailles égales et en assignant un bloc à chaque processeur. Enfin une distribution bloc-cyclique( $n$ ) est réalisée en divisant le tableau en blocs de taille  $n$  et en distribuant ensuite de manière cyclique les blocs sur les processeurs. La figure 2.2 illustre ainsi ces différentes possibilités, par différents placements d'un tableau de 9 éléments (à une dimension) sur 3 processeurs P1, P2, P3.

On peut remarquer que les distributions cycliques ou par blocs sont des cas particuliers des distributions bloc-cycliques : dans le premier cas il suffit de considérer une distribution bloc-cyclique avec des blocs de taille 1 et dans le second cas une redistribution avec des blocs de taille  $\lceil \frac{M}{P} \rceil$  où  $M$  est la taille du tableau et  $P$  le nombre de processeurs. Tous les travaux présentés dans cette section concerneront des distributions bloc-cycliques mais peuvent donc parfaitement être appliqués aux deux autres types de distributions. On notera de plus que les tableaux à plusieurs dimensions peuvent être également distribués de manière similaire en prenant des blocs en plusieurs dimensions.

### Exécution d'une redistribution

Une redistribution interne à une grappe afin de changer le placement des données pose deux problèmes importants :

1. trouver efficacement quels sont les communications à effectuer ;
2. exécuter les communications le plus rapidement possible.

Le premier problème est certainement le plus étudié et il existe de nombreuses méthodes permettant de calculer le motif de redistribution (quel processeur doit envoyer quelles données à

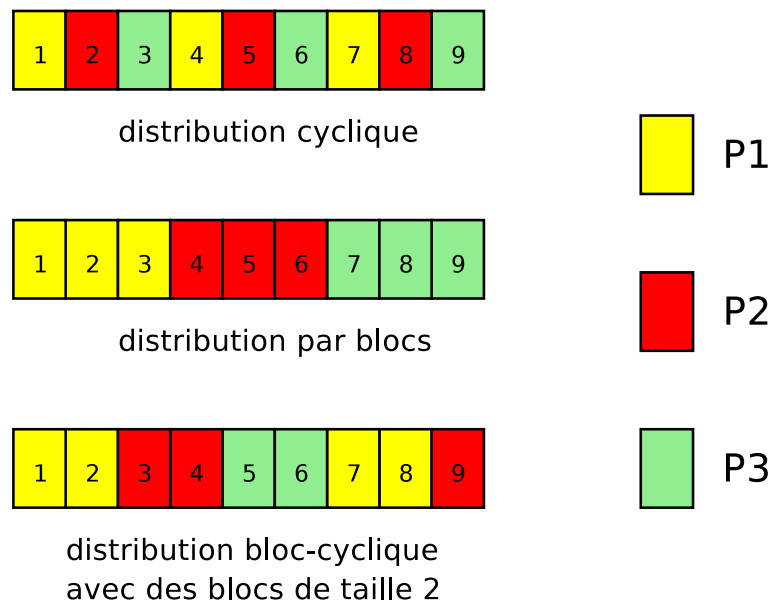


FIG. 2.2 – Distribution cyclique, par blocs, ou bloc-cyclique d’un tableau à une dimension

quel autre processeur). Nous présenterons par la suite un aperçu de quelques-unes des méthodes proposées, parmi les plus intéressantes. Nous porterons néanmoins plus d’attention sur le second problème, d’optimisation des communications.

Deux méthodes différentes et complémentaires existent pour minimiser le temps de communication :

- ordonnancer les communications afin d’éviter la contention ;
- diminuer la quantité de données à envoyer.

### Un placement intelligent des données

Hsu et al. [46] ont proposé une méthode permettant de réduire les quantités de données à redistribuer en fournissant un choix judicieux des numéros de processeurs : certains processeurs font ici partie de l’ensemble des processeurs sources et de l’ensemble des processeurs destinations. Certaines données ont ainsi juste besoin d’être copiées localement et non transférées à travers le réseau. Il est possible d’augmenter le nombre de données à transférer d’un processeur vers lui même en choisissant un bon ordre de distribution des données sur les différents processeurs.

La figure 2.3 illustre ainsi une redistribution d’un placement d’un tableau de 50 éléments en bloc-cyclique(10) vers un placement bloc-cyclique(5). Les données sont ici réparties sur 5 processeurs. On représente pour la distribution finale deux possibilités : une distribution des blocs sur les processeurs dans l’ordre P0, . . . , P4 et dans l’ordre P0, P3, P1, P4, P2. On peut voir que dans le premier cas seuls 2 blocs sont communiqués localement tandis que dans le second cas 5 blocs (soit 50% du total) sont déjà localisés sur le bon processeur.

Hsu et al. proposent une formule permettant d’obtenir un ordre optimal pour les processeurs dans le cas d’une redistribution d’une distribution bloc-cyclique( $kr$ ) vers une distribution bloc-cyclique( $r$ ) ou inversement. Ils concluent également par une extension pour le cas de tableaux multi-dimensionnels.

L’autre possibilité d’amélioration du temps de redistribution consiste à ordonnancer les com-

munications afin de minimiser non pas la quantité de données à transférer, mais bien directement les temps de transferts.

Pour chacune des méthodes présentées ici, le principe consiste à ordonnancer les communications en une série d'étapes, afin d'interdire à un nœud émetteur ou récepteur de traiter plus d'une communication simultanément. Il y a essentiellement deux raisons à cela :

- tout d'abord, se limiter à une seule communication simplifie grandement le travail d'implémentation logicielle ;
- de plus effectuer toutes les communications (ou plusieurs) simultanément nécessite d'une part de gérer des communications asynchrones, et d'autre part de conserver en mémoire simultanément tous les tampons nécessaires pour chacune des communications, ce qui peut impacter très fortement les performances.

index	01-10	11-20	21-30	31-40	41-50
processeur	P0	P1	P2	P3	P4

index	01-05	06-10	11-15	16-20	21-25	26-30	31-35	36-40	41-45	46-50
processeur	P0	P1	P2	P3	P4	P0	P1	P2	P3	P4
processeur	P0	P3	P1	P4	P2	P0	P3	P1	P4	P2

FIG. 2.3 – Optimisation de la quantité de données à redistribuer

### Un algorithme d'ordonnancement : le Caterpillar

L'algorithme d'ordonnancement utilisé actuellement dans ScaLAPACK est le *Caterpillar* [59] développé en 1996 par Prylli et al. On considère ici que l'ensemble des nœuds sont à la fois émetteurs et récepteurs. Le principe de l'algorithme est le suivant : on construit une liste de tous les  $n$  nœuds que l'on divise en deux listes de taille  $\frac{n}{2}$  : un ensemble d'émetteurs et un ensemble de récepteurs. Dans une première phase de redistribution, le  $i^{\text{ème}}$  nœud émetteur envoie toutes les données qu'il doit envoyer au  $i^{\text{ème}}$  nœud récepteur (pour  $i$  de 1 à  $\frac{n}{2}$ ). Aucun nœud ne reçoit ou n'émet donc plus d'une communication à la fois. On décale ensuite chaque liste d'un cran : le dernier nœud de la liste émettrice devient le premier de la liste réceptrice, et à l'intérieur de la liste émettrice (resp. réceptrice) les numéros de chacun des nœuds sont incrémentés (resp. décréments). S'ensuit une seconde étape de communication puis un nouveau décalage, l'algorithme continuant ainsi jusqu'à ce que chacun ait communiqué avec tout le monde. La figure 2.4 illustre ainsi le mouvement des processeurs entre les deux listes, à la manière d'une chenille.

On peut remarquer que le Caterpillar a une complexité très faible et donc que l'ordonnancement est très rapide à calculer. De plus le nombre de phases de redistribution est optimal dans le cas où chaque nœud a des données à redistribuer vers chacun des autres. Enfin, il est important de noter que chaque phase de redistribution a comme durée la durée de la communication la plus longue qu'elle contient. Ceci signifie que lorsque les longueurs des différentes communications ne sont pas identiques le Caterpillar peut fournir des résultats de performances variables : l'ordonnancement pourrait être moins efficace qu'un ordonnancement regroupant dans une même étape des communications de tailles sensiblement égales.

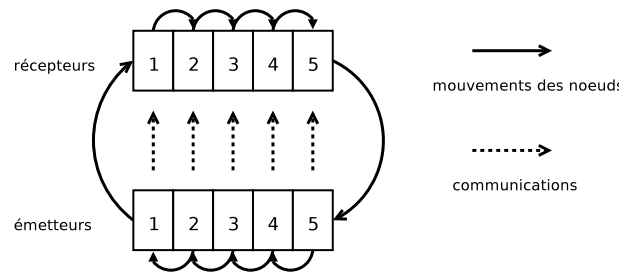


FIG. 2.4 – Déroulement de l’algorithme du Caterpillar

### De meilleurs algorithmes d’ordonnement

C’est ce problème qui a amené Desprez et al. à développer un autre algorithme d’ordonnement [28], en se basant sur des travaux initiaux de Walker et al. [67]. Ils proposent deux algorithmes d’ordonnement : le premier minimisant le nombre total d’étapes et le second cherchant à minimiser le coût total de toutes les étapes. Le coût d’une étape est défini par le temps pris par la communication la plus longue qu’elle contienne et le coût de toutes les étapes est simplement la somme des coûts de chacune.

Les deux algorithmes fonctionnent en modélisant le problème à résoudre à l’aide d’un graphe biparti valué  $G = (V_1, V_2, E, w)$  où  $V_1$  est l’ensemble des émetteurs,  $V_2$  celui des récepteurs,  $E \subset V_1 \times V_2$  l’ensemble de toutes les communications de taille non nulle et  $w : E \rightarrow \mathbb{Q}$  la taille de chaque communication. Les deux algorithmes décomposent le graphe initial en une série de couplages (ensemble d’arêtes tel qu’il n’existe pas un sommet incident à deux arêtes de l’ensemble) représentant chacun une étape de communication. L’utilisation de couplages permet de limiter à une seule le nombre de communications simultanées pour chaque processeur. Le nombre minimal d’étapes est obtenu en décomposant le graphe en un nombre minimal de couplages avec un algorithme similaire à celui décrit section 3.2.4. Nous rentrerons plus en détail sur cet algorithme dans le prochain chapitre consacré plus particulièrement aux travaux existants en théorie des graphes. L’heuristique cherchant à minimiser le coût des étapes fonctionne en prenant des couplages de poids maximal (on choisit à chaque étape le couplage donnant une somme des poids de chacune de ses arêtes qui est maximale).

Les expériences effectuées à l’aide des ces deux algorithmes montrent de meilleures performances qu’avec le Caterpillar sur des redistributions effectuées à l’aide de la bibliothèque MPI. On peut noter néanmoins que le temps de calcul de l’ordonnement n’est pas forcément négligeable, les deux heuristiques ayant une complexité en  $O(n^4)$ ,  $n$  étant le nombre total de processeurs utilisés. Un point fort de ces travaux est, comme pour le Caterpillar, qu’ils fonctionnent quelle que soit le type de redistribution et pas uniquement sur des redistributions de bloc-cyclique à bloc-cyclique : il n’y a en effet aucune contrainte sur le graphe biparti utilisé en entrée (ce qui n’est pas le cas, en revanche, de la phase de construction de la matrice exprimant les quantités de données à communiquer entre chaque processeur).

### Des algorithmes plus rapides

L’algorithme d’ordonnement par couplages sur un graphe biparti peut dans certains cas, lorsque les quantités de données à communiquer ne sont pas très élevées, avoir un temps d’exécution trop long pour qu’il soit utilisable. Ce problème a amené aux développements d’autres algorithmes, plus rapides (mais plus spécifiques) car prenant en compte les propriétés particulières de périodicité des distributions bloc-cyclique.

Park et al. [58] ont ainsi proposé un algorithme d'ordonnement pour une redistribution d'une distribution bloc-cyclique( $x$ ) sur  $P$  processeurs vers une distribution bloc-cyclique( $Kx$ ) sur  $Q$  processeurs. Cet algorithme, fonctionnant à l'aide de matrices circulantes minimise le nombre d'étapes de communications, fournit des étapes où toutes les communications ont une durée identique, et a une complexité en  $O(\max(P, Q))$ .

Guo et al. [42] ont proposé une méthode présentant les mêmes propriétés, mais pour le cas général d'une redistribution bloc-cyclique vers bloc-cyclique, y-compris les cas multi-dimensionnels.

**Une heuristique pour le cas hétérogène** Les développements ultérieurs sur la redistribution ont consisté à étendre le problème étudié. Bhat et al. [16] ont ainsi étudié le problème de la redistribution de données sur une machine parallèle à mémoire distribuée disposant de différentes interfaces réseau hétérogènes. Notons également que les redistributions considérées ici sont asynchrones.

L'ordonnement est ici effectué en deux étapes :

- on calcule tout d'abord le temps que prendra chaque communication : un coût de connexion (fixe) auquel il suffit de rajouter la quantité de données à envoyer divisée par la bande passante du lien ;
- plutôt que d'utiliser des étapes de communication, chaque émetteur choisit comme destination la prochaine destination qui sera libre (en se basant sur les durées calculées précédemment).

Il n'est ici plus possible d'utiliser les propriétés de régularité liées aux distributions bloc-cycliques car les différentes durées de communication dépendent à présent des paramètres réseau. Notons que le Caterpillar ou l'algorithme utilisant des couplages sur un graphe biparti pourrait fonctionner dans ce cas là, mais en présentant bien entendu les inconvénients liés à chacune de ces deux approches. La figure 2.5 illustre un exemple de déroulement de cette heuristique. Chaque barre du diagramme représente une communication (le numéro de la destination figurant sur la barre). On peut noter que le coût d'initialisation de chaque communication est ici compris directement dans la longueur de chacune d'entre elles.

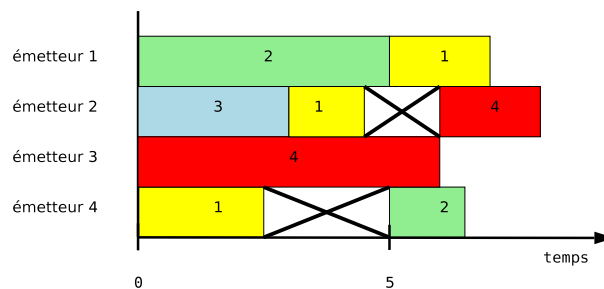


FIG. 2.5 – Déroulement d'une heuristique gloutonne pour une redistribution sur un réseau hétérogène



## Chapitre 3

# Redistribution de données à travers un réseau à haut débit

### Sommaire

---

<b>3.1</b>	<b>Problème</b>	<b>33</b>
3.1.1	Description	34
3.1.2	Motivation	37
<b>3.2</b>	<b>Modélisation</b>	<b>41</b>
3.2.1	Paramètres	42
3.2.2	Redistribution par étapes	43
3.2.3	Le problème KPBS	46
3.2.4	Problèmes en relation avec KPBS	50
3.2.5	Expériences de validation du modèle	58

---

### 3.1 Problème

Avec la grande hétérogénéité actuelle aussi bien des plateformes de calcul que des logiciels utilisés, le nombre de configurations différentes pour effectuer une redistribution est relativement élevé. De plus il est clair qu'une seule technique de redistribution n'est pas en mesure de fonctionner efficacement dans tous les cas, en particulier une méthode de redistribution conçue pour des réseaux rapides risque d'avoir une performance très mauvaise si les réseaux s'avèrent engorgés ou lents et inversement.

Nous pensons qu'il n'existe pas une méthode de redistribution fonctionnant pour chaque cas, mais au contraire que pour être efficace, un intergiciel doit être en mesure de gérer les différentes possibilités qui peuvent se présenter à lui. Il est donc important de disposer de plusieurs méthodes permettant de résoudre plusieurs problèmes, et tout aussi important d'être en mesure de choisir efficacement quelle solution prendre dans chaque cas.

Nous présentons dans cette section le problème *particulier* de redistribution que nous avons étudié initialement. Celui-ci dérive d'un cas concret qui nous a été présenté et forme le point de départ de nos travaux.

### 3.1.1 Description

#### Configuration matérielle

Nous considérons ici deux grappes d'ordinateurs reliées entre elles par un réseau à haut débit. Dans un soucis de clarté, nous assignerons par la suite un identifiant à chaque nœud d'une grappe : un nombre pour les nœuds de la première grappe, et une lettre pour ceux de la seconde. Ainsi parler du nœud "B" signifiera immédiatement que celui-ci appartient à la seconde grappe.

Chaque nœud d'une grappe dispose d'une interface réseau. Cette interface est connectée à un commutateur réseau. Ce dernier est lui même connecté à un réseau à *haut débit* reliant ainsi les deux grappes entre elles. On suppose ici que la vitesse de ce réseau, que nous appellerons dorsale (*backbone*) par la suite, est supérieure à la vitesse d'une interface. Nous supposons également que le commutateur est capable d'atteindre la bande passante de la dorsale. Dans le cas contraire, on considérera celle-ci comme égale (limitée) à celle du commutateur. A l'intérieur d'une grappe, les interfaces réseau sont homogènes : chacune fournit la même bande passante que les autres, mais les interfaces de la première grappe et seconde grappe peuvent être différentes. La figure 3.1 illustre ainsi une telle configuration.

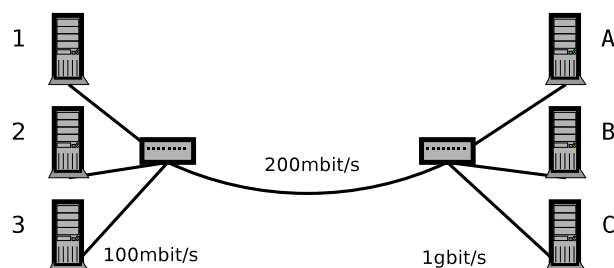


FIG. 3.1 – Exemple de configuration réseau

Il n'est pas rare à l'heure actuelle de disposer de grappes de calcul identiques à celles utilisées ici. En effet, les contraintes posées par nos hypothèses sont relativement faibles : une homogénéité des interfaces, une topologie simple (en étoile) et non une topologie hiérarchique, et enfin un commutateur rapide. Par contre, il est plus délicat d'obtenir une bande passante sur la dorsale plus élevée que la bande passante du réseau local. En particulier, la plupart des connections passant par internet ne sont pas en mesure de fournir une bande passante aussi élevée.

Le problème considéré n'est pas pour autant irréaliste. On peut considérer plusieurs possibilités différentes de rencontrer une telle configuration. Il est tout d'abord possible d'effectuer une redistribution entre deux grappes situées sur un réseau local et donc capable de soutenir un débit élevé. Il est également possible d'effectuer des calculs à travers un réseau à grande échelle haute performances. Il s'agit dans ce cas là souvent de réseaux dédiés à la recherche et au calcul scientifique et donc d'accès restreint. Le réseau VTHD [6] est un bon exemple de ce type d'infrastructure : de 1999 à 2001 (étendu de 2002 à 2004 par le projet VTHD++) ce réseau reliait les principaux sites de français de recherche en informatique suivant la configuration de la figure 3.2 et fournissait à des fins de recherche une bande passante de 2,5 Gbit/s.

Enfin, il est possible dans certaines configurations d'utiliser un réseau utilisé par plus de monde, mais avec une bande passante garantie par qualité de service. Par exemple, le projet GRID 5000 fédère à travers la France un ensemble de ressources informatiques afin de fournir une



FIG. 3.2 – Le réseau VTHD

plate-forme expérimentale de recherche sur les grilles. Ce projet, initié par l'ACI Globalisation des Ressources Informatiques et des Données, rassemble des ressources fournies par neuf laboratoires de recherche français (ID-IMAG, LRI, INRIA Sophia, IRISA, IRIT, LABRI, LIP, LIFL, LORIA) comme illustré sur la figure 3.3. L'interconnexion entre les différents sites est réalisée par le réseau Renater, qui est lui, partagé par toutes les universités. Néanmoins, 1 Gbit/s de bande passante a été dédié exclusivement à GRID 5000. De plus cette bande passante est destinée à augmenter en même temps que les besoins dans les années à venir.

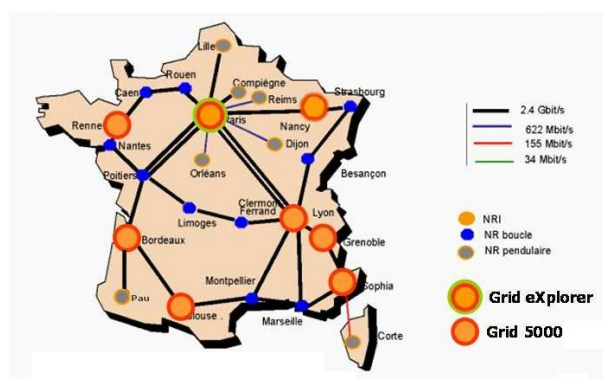


FIG. 3.3 – GRID 5000

### Pré-requis

En plus des contraintes sur la configuration matérielle requise, nous faisons ici un certain nombre d'hypothèses :

1. chaque machine peut communiquer vers l'extérieur.

Cette contrainte peut paraître comme allant de soi si l'on envisage de faire de la redistribution, mais toutes les plateformes ne permettent pas un accès externe à chaque machine d'une grappe. Pour des raisons de sécurité par exemple, il se peut que seule une machine

frontale dispose d'une connexion vers l'extérieur, toutes les machines internes ne disposant que d'une adresse IP locale. On peut néanmoins envisager de concilier sécurité et redistribution à l'aide d'un VPN (*Virtual Private Network*) et d'un routeur s'occupant du filtrage. C'est par exemple le cas pour les machines de GRID 5000 ou chaque site dispose d'un routeur s'occupant du filtrage conjointement avec la qualité de service.

2. Toutes les bandes passantes sont connues.

"Bande passante" est ici un terme un peu flou. En réalité la bande passante théorique maximale (i.e. délivrable en théorie) de chaque interface ne nous intéresse pas. On s'intéresse ici à savoir : quelle bande passante chaque interface, commutateur et enfin la dorsale est capable de soutenir réellement. Plusieurs paramètres rentrent ici en jeu : l'intergiciel utilisé, la configuration des systèmes d'exploitation, l'utilisation actuelle de la dorsale par d'autres utilisateurs, ...

Il existe plusieurs manières d'obtenir ces informations :

- elles peuvent être fournies directement par l'utilisateur. Dans ce cas on perd en transparence, mais on gagne sans doute en efficacité.
- Elles peuvent être fournies par un service de l'intergiciel. *Network Weather Service* (NWS) [70] par exemple permet de fournir une estimation des différentes bandes passantes disponibles.
- Enfin, il est possible d'effectuer un calibrage rapide juste avant d'effectuer la redistribution. Cette approche a l'avantage de fournir de bons résultats, mais prend bien entendu plus de temps que les deux autres approches.

Notons que nous supposons toujours par la suite ces informations fournies. Nous ne nous intéresserons pas plus à la manière de les obtenir.

3. La bande passante disponible ne varie pas au cours du temps.

Cette hypothèse est certainement l'hypothèse la plus délicate dont nous avons besoin. Il est clair que sur un réseau non dédié, il est impossible de garantir que la bande passante restera identique tout au long de la redistribution. Même l'utilisation de prédictions de charge à l'aide de statistiques (comme le permet NWS) ne fournit que des indications, et en aucun cas des garanties.

Cette hypothèse est néanmoins justifiable, pour plusieurs raisons :

- Il est possible dans certains cas d'avoir une bande passante réservée. Bien que ceci limite le nombre de cas réels d'application de ce problème, cette configuration est envisageable, notamment si les deux grappes appartiennent à une même organisation.
- Chaque redistribution ne prend qu'un temps relativement faible à s'exécuter (pas plus de quelques minutes). Il est donc probable que les variations de bande passante durant une redistribution ne soient donc pas très élevées. Cela limite donc l'impact que les changements peuvent avoir.
- Il est possible de segmenter chaque redistribution en redistributions de tailles plus petites. Nous verrons par la suite que cette technique accroît le coût total d'une redistribution, mais permet une adaptation plus grande aux variations.
- Seule l'augmentation de bande passante pose un réel problème. Ce point sera expliqué en détail dans la partie concernant le modèle utilisé. Il est possible de surévaluer légèrement la bande passante disponible afin de se prémunir d'une légère augmentation. Bien entendu cela ne fait que limiter le problème, et ne le supprime pas.
- En cas de variation trop élevée de la bande passante, il est toujours possible d'interrompre la redistribution et de recalculer un nouvel ordonnancement. Cette approche peut s'avérer délicate à mettre en place (elle nécessite de tester périodiquement quelle

est la bande passante disponible). De plus il n'est pas évident que les gains en temps obtenus soient conséquents car elle nécessite de recalculer un ordonnancement, ce qui coûte également du temps. Nous ne la mentionnons donc ici que pour être exhaustifs.

- Enfin, il est à noter que le cas réellement problématique n'est que lors de constants changements brusques de la bande passante disponible. Ce cas est mal géré par le modèle et les algorithmes proposés mais il est probable que même des algorithmes à grain très fin, laissant à TCP l'ordonnancement des communications, se comportent mal dans une telle situation.

4. Il n'y a pas de communications locales. Cette hypothèse semble limiter un peu la portée du problème étudié, mais en réalité il nous suffit de considérer que l'on effectue en premier une pré-redistribution concernant uniquement toutes les communications locales. Nous verrons plus tard comment éliminer cette contrainte.
5. Le motif de données à redistribuer est connu.

Cette hypothèse ne pose pas de problèmes particuliers (ce motif nous est fourni par la première étape de la redistribution, voir section 2.3.2). Elle nous interdit par contre de traiter le cas du problème on-line, c'est-à-dire lorsque les données à traiter apparaissent au fur et à mesure.

Nous supposons que les interfaces réseau supportent le full-duplex. Ainsi les communications dans les deux sens peuvent avoir lieu simultanément et on se réduit au problème d'effectuer deux redistributions (une dans chaque sens) simultanément. Nous prenons la convention de nommer la première grappe "grappe émettrice" et la seconde grappe "grappe réceptrice". Par la suite, la grappe réceptrice sera toujours représentée à gauche, la grappe émettrice à droite sur chaque illustration. Les données à envoyées seront par la suite représentées par une matrice de communication  $M = (m_{i,j})$  où  $m_{i,j}$  représente la quantité de données que le nœud  $i$  de la grappe émettrice doit envoyer nœud  $j$  de la grappe réceptrice.

### 3.1.2 Motivation

Avant de passer à l'étude du problème en lui même, nous proposons d'examiner plusieurs situations différentes, montrant chacune un problème posé par une exécution simpliste de la redistribution.

La manière la plus simple (au moins au niveau conceptuel) d'effectuer une redistribution est d'exécuter simultanément toutes les communications, en laissant au réseau la charge de gérer la contention. Nous appellerons une telle redistribution "redistribution par force brute". Cette approche présente différents avantages :

- aucun calcul n'est nécessaire ;
- les variations de la charge réseau ne nécessitent aucun réajustement ;
- l'approche est relativement simple à mettre en oeuvre.

Par la suite, nous allons illustrer l'exécution d'une telle redistribution à l'aide de diagrammes de Gantt. Nous supposons que lorsque la bande passante de la dorsale n'est pas suffisante pour laisser passer l'ensemble des flux à leur vitesse maximale, elle est répartie équitablement entre chacun d'entre eux. Cette hypothèse ne se vérifiera sans doute pas dans tous les cas. Néanmoins, nous supposons que dans la plupart des cas, la latence entre chaque source et chaque destination est identique. Les travaux existant sur les prédictions de performance réseau [22] indiquent que sous cette hypothèse la bande passante devrait être répartie équitablement. Bien entendu il existe sans doute des configurations où cette propriété ne se vérifie pas, mais il est alors délicat d'estimer les performances que l'on obtient.

### Utilisation non optimale de la bande passante

On considère la configuration de la figure 3.4 : deux grappes de trois machines reliées entre elles par un réseau à 200 Mbit/s. Chaque récepteur est capable de recevoir à 1 Gbit/s mais les émetteurs n'émettant qu'à 100 Mbit/s, la vitesse de chaque communication ne peut dépasser 100 Mbit/s. On voit ici que deux flux parallèles suffisent pour agréger la bande passante de la dorsale, en troisième flux supplémentaire réduisant la bande passante de chacun.

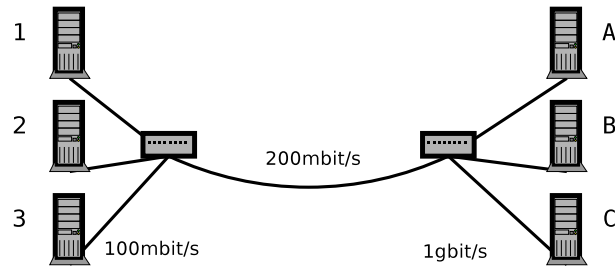


FIG. 3.4 – Deux grappes de trois machines

On cherche ici à effectuer une redistribution très simple : Le nœud 1 doit envoyer 100 Mbit au nœud A, le nœud 2 100 Mbit au nœud B, et le nœud 3 200 Mbit au nœud C, ce que l'on représente à l'aide de la matrice de communication de la figure 3.5.

nœuds	A	B	C
1	100	0	0
2	0	100	0
3	0	0	200

FIG. 3.5 – Matrice de communication

En exécutant une redistribution par force brute on voit deux cas de figure se présenter. Initialement, trois flux parallèles se partagent équitablement une bande passante de 200 Mbit/s. Chaque nœud envoie donc 100 Mbit durant la même durée de  $\frac{100}{200} = 1,5$  seconde. Ensuite, seul le nœud C a encore des données (100 Mbit) à envoyer. Étant à ce moment là seul à utiliser le réseau, il communique à la vitesse de sa carte soit 100 Mbit/s et termine donc sa communication en 1 seconde. Au total donc cette redistribution prend  $1,5 + 1 = 2,5$  secondes comme illustré sur le diagramme de Gantt de la figure 3.6. Nous prenons comme convention de noter les nœuds émetteurs sur l'axe des ordonnées et les nœuds récepteurs sur chaque communication.

On peut déjà sentir quel est le problème posé par cette redistribution rien qu'en regardant le diagramme de Gantt obtenu. Idéalement, pour minimiser le temps total de transfert, il est nécessaire d'éviter au maximum les "trous" (i.e. les moments où la bande passante est sous-utilisée) dans le diagramme. Or ici, il est clair que la seconde partie de la redistribution n'est pas optimale.

On se propose, pour remédier à cela, d'ordonnancer les communications de la manière suivante (figure 3.7) : dans une première étape de communication, les nœuds 1 et 3 émettront chacun 100 Mbit, puis dans une seconde étape, les nœuds 2 et 3. Ainsi le nœud 3 aura bien émis la totalité de ses 200 Mbit (en deux étapes de communication).

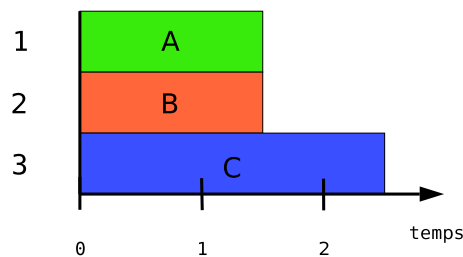


FIG. 3.6 – Exécution de la redistribution par force brute

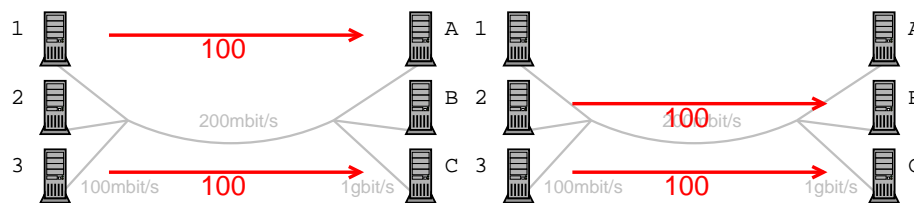


FIG. 3.7 – Ordonnancement des communications en 2 étapes

Chaque communication est ici effectuée à sa vitesse maximale de 100 Mbit/s et chaque étape ne prend donc qu'une seule seconde. La redistribution complète s'effectue donc en 2 secondes soit 80% du temps pris par une redistribution par force brute.

On voit donc ici que même sur un exemple très simple, alors qu'il n'y a aucune perte de paquets liée à une congestion, une approche par force brute n'est pas capable d'obtenir des performances optimales.

### Dégradation des performances

Un autre problème de l'approche par force brute se manifeste dans le cas où la redistribution dépasse les capacités du réseau. La congestion générée par l'utilisation de flux parallèles massifs peut se révéler importante et occasionner un surcoût en nécessitant par exemple la retransmission des paquets perdus.

Christian Perez et André Ribes [60] ont effectué une série de tests sur le réseau VTHD visant à estimer dans quelle mesure l'augmentation du nombre de communications dégradait la bande passante. Les tests ont eu lieu entre des machines de Rennes et Nice, disposant d'interfaces réseau à 100 Mbit/s, les routeurs supportant eux une bande passante de 1 Gbit/s. La matrice de redistribution utilisée dans ces tests est une matrice très simple : une matrice diagonale avec toutes les coefficients de la diagonale identiques. La figure 3.8 montre comment évolue la bande passante agrégée lorsque le nombre de nœuds croît. On voit qu'arrivé vers 12 flux parallèles la bande passante diminue pour une redistribution par force brute, tandis qu'une version ordonnancée des redistributions permet de maintenir un débit maximal.

### D'autres critères que le temps de terminaison

Enfin, nous nous proposons d'illustrer l'utilité de l'ordonnancement des communications pour une redistribution de donnée en utilisant des critères différents que le critère usuel de temps de

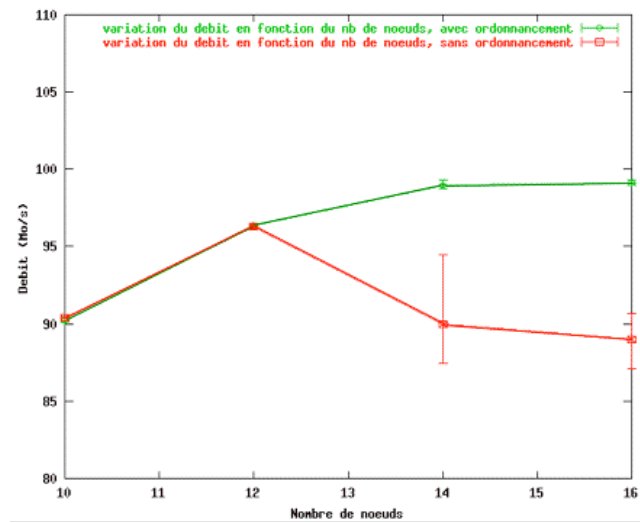


FIG. 3.8 – Tests de saturation sur le réseau VTHD (réalisés par Christian Perez et André Ribes)

terminaison (*Makespan*).

L'ordonnancement de tâches sur des systèmes parallèles dispose en effet d'autres critères d'évaluation permettant d'estimer les performances d'un système selon d'autres considérations que le point de vue de l'utilisateur :

- le *sum-flow* [11], qui est la somme des durées effectives de chaque tâche ;
- le *max-stretch* [13], le maximum des rapports entre la durée effective d'une tâche et sa durée minimale ;
- le *max-flow* [13], la durée effective maximale d'une tâche.

Ces critères, bien qu'utilisés moins couramment, peuvent s'avérer utiles, notamment si le but de l'ordonnancement est d'équilibrer et réduire la charge supportée par les ressources de calcul.

Prenons par exemple la configuration de la figure 3.9 avec les communications de la matrice de la figure 3.10.

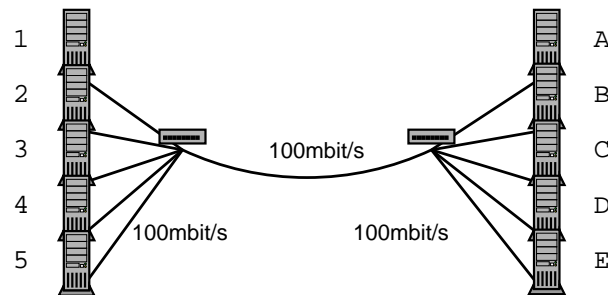


FIG. 3.9 – Deux grappes reliées par une dorsale lente

On voit ici qu'une seule communication est suffisante pour agréger toute la bande passante disponible. Il n'est donc pas déraisonnable d'effectuer les communications une-à-une séquentiellement (ce qui est bien sûr un ordonnancement). Le temps total de communication dans ce cas là est de  $5 \times \frac{100}{100} = 5$  secondes. Effectuer la même redistribution par force brute prend exac-

nœuds	A	B	C	D	E
1	100	0	0	0	0
2	0	100	0	0	0
3	0	0	100	0	0
4	0	0	0	100	0
5	0	0	0	0	100

FIG. 3.10 – Matrice de communication (homogène)

tement le même temps :  $\frac{100}{5} = 5$  secondes (sans prendre en compte les pertes dues à la gestion de la congestion). On pourrait donc dans le cas présent considérer les deux approches comme équivalentes en terme de performances.

En réalité, l'approche ordonnancée est ici encore la meilleure : en ordonnant les communications dans cet ordre :  $(1, A)(2, B)(3, C)(4, D)(5, E)$  on remarque que au bout d'une seconde les processeurs 1 et A ont terminé toutes leurs communications et peuvent donc éventuellement commencer leurs calculs. Au bout de deux secondes, c'est au tour des processeurs 2 et B, puis 3 et C, etc... Au contraire, dans le cas d'une redistribution par force brute, la bande passante étant équitablement répartie entre chaque communication, elles se terminent toutes en même temps : à 5 secondes. On voit donc ici que bien que la durée totale de communication soit identique dans les deux cas, le sum-flow n'est que de 5 secondes dans le cas ordonnancé contre 25 secondes dans le cas par force brute. De plus, le temps moyen avant la fin d'une communication n'est que de 3 secondes ( $\frac{1+2+3+4+5}{5}$ ). Ceci est particulièrement visible sur les diagrammes de Gantt de la figure 3.11.

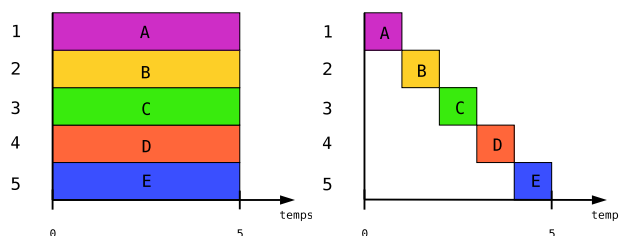


FIG. 3.11 – Comparaison entre les deux approches

Ainsi, une approche par communications ordonnancées peut être justifiée y compris lorsque une approche par force brute se révèle performante.

## 3.2 Modélisation

Le calcul d'un ordonnancement efficace pour exécuter une redistribution de données nécessite une modélisation précise du problème étudié. Nous présentons ici un modèle introduit par Crescenzi et. al. [26] pour le traitement du problème PBS, plus simple que celui considéré ici. Ce modèle a ensuite été légèrement modifié, et réétudié par Cohen et al. afin de s'adapter à nos besoins [25].

Nous présentons tout d'abord le modèle complet accompagné de quelques propriétés intéressantes pour nos travaux ultérieurs. La présentation du modèle s'effectuera en deux étapes :

nous détaillerons section 3.2.1 les différents paramètres impliqués par le réseau et l'application ainsi qu'une modélisation du motif de données à redistribuer sous forme de graphe biparti, puis section 3.2.2 une modélisation d'une redistribution effectuée par étapes synchronisées.

Enfin nous présentons une description formelle du problème étudié, intitulé KPBS, suivi par une classification de différents problèmes en relation directe avec KPBS.

Nous concluons ce chapitre par une série d'expérience permettant de vérifier une bonne adéquation du modèle à la réalité.

### 3.2.1 Paramètres

Sous les conditions citées section 3.1, une redistribution de données est essentiellement soumise à deux types de paramètres :

- les paramètres réseau,
- le motif de données à redistribuer.

La gestion des paramètres réseau ne nécessite que quelques variables :

- $b_1$ , la bande passante locale de la grappe émettrice ;
- $b_2$ , la bande passante locale de la grappe réceptrice ;
- $b_b$ , la bande passante de la dorsale ;
- $\beta$ , la latence d'une communication.

Il est de plus important de noter que  $\beta$  n'est pas obligé de prendre uniquement en compte la latence réseau. On peut ajouter à  $\beta$  les surcoûts en temps liés aux logiciels établissant les communications ainsi que d'éventuels coûts de synchronisations.

Dans le modèle choisi, le motif de données à redistribuer va être représenté à l'aide d'un graphe biparti valué  $G = (V_1, V_2, E, w)$  appelé *graphe de communication*.  $V_1$  est l'ensemble des nœuds de la grappe émettrice,  $V_2$  l'ensemble des nœuds de la grappe réceptrice,  $E \subseteq V_1 \times V_2$  l'ensemble de toutes les communications et  $w : E \rightarrow \mathbb{Q}$  une fonction associant à chaque communication son temps d'exécution à vitesse maximale (i.e. si elle était exécutée toute seule sur le réseau). En réalité cette modélisation n'est pas très éloignée de ce que l'on peut voir représenté sur la figure 3.7 de la section précédente.

Le calcul de la fonction  $w$  se fait très facilement à partir de la matrice de communication et des différentes bandes passantes. Il suffit de diviser chaque quantité de données par la bande passante minimale (celle à laquelle s'effectuera chaque communication). Ainsi à partir d'une matrice de communication  $\mathcal{M}$  on calcule une matrice  $\mathcal{M}' = (m'_{i,j})$ , appelée matrice de communication normalisée, où  $m'_{i,j} = \frac{m_{i,j}}{\min(b_1, b_2, b_b)}$ . La matrice  $\mathcal{M}'$  connue il est toujours possible de retrouver la matrice  $\mathcal{M}$  par l'opération inverse.

Nous considérerons ultérieurement (sauf si précisé autrement) uniquement des matrices de communications déjà normalisées par la bande passante disponible que nous nommerons  $\mathcal{M}$ .

Le calcul des arêtes de l'ensemble  $E$  est effectué à partir de  $\mathcal{M} : \forall i \in V_1, \forall j \in V_2, (i, j) \in E \Leftrightarrow m_{i,j} \neq 0$ .

Pour illustrer cette modélisation, nous considérons la matrice de communication de la figure 3.12 sur la configuration de la section précédente illustrée par la figure 3.4. Chaque case  $m(i, j)$  de la matrice contient la quantité de données en Mbit que le nœud  $i$  doit transférer au nœud  $j$ . On suppose de plus une latence (exagérée, mais permettant un exemple simple) égale ici à un dixième de seconde :  $\beta = 0,1s$ .

A partir de ces données on calcule la matrice de communication normalisée ainsi que le graphe biparti de la figure 3.13. Notons que chaque case de la matrice exprime désormais un temps en secondes.

Nous définissons de plus quelques valeurs importantes à partir du graphe  $G$  :

nœuds	A	B	C
1	100	300	0
2	0	200	500
3	0	150	100

FIG. 3.12 – Matrice de communication initiale

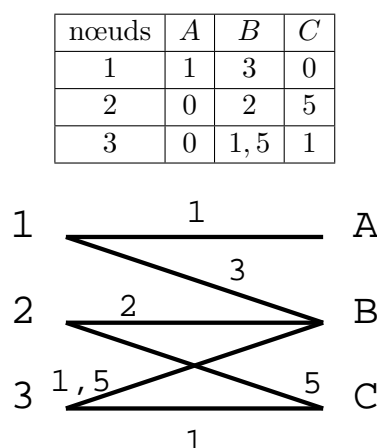


FIG. 3.13 – Matrice normalisée et graphe de communication

- $m = |E|$  : le nombre d'arêtes du graphe,
- $n = |V_1 \cup V_2|$  : le nombre de sommets du graphe,
- $\Delta(G)$  : le degré du graphe,

L'ensemble des notations utilisées par l'auteur tout au long de cette thèse est résumé Annexe A afin de permettre au lecteur de retrouver rapidement toutes les définitions.

### 3.2.2 Redistribution par étapes

L'ensemble des données à envoyer est représenté par le graphe de communication. Afin de disposer d'un mécanisme de redistribution qui ne soit pas trop complexe à implanter, nous souhaitons décomposer la redistribution à effectuer en une série d'étapes synchronisées, il nous faut à présent être capable de décomposer le graphe lui correspondant. On souhaite également qu'à chaque étape un nœud récepteur ne puisse recevoir au plus qu'une seule communication, et de manière similaire qu'un nœud émetteur ne puisse émettre qu'un seul flux de données.

De plus, nous autorisons la préemption, c'est à dire la possibilité d'exécuter chaque communication en plusieurs étapes comme illustré figure 3.7.

Avec ces hypothèses, une décomposition en  $h$  étapes d'un graphe de communication  $G = (V_1, V_2, E, w)$  est un ensemble  $D$  de  $h$  couplages valués  $M_1 = (E_1, w_1), \dots, M_h = (E_h, w_h)$  avec chaque  $E_l$  l'ensemble des arêtes du couplage, et chaque  $w_l$  une fonction de valuation de  $E$  dans  $\mathbb{Q}$ . Les différentes fonctions  $w_l$  permettent ainsi d'associer un poids différent à chaque arête de chaque couplage. On rappelle qu'un couplage est défini comme un ensemble d'arêtes telles qu'il n'existe aucun sommet adjacent (i.e. en contact) avec deux arêtes du couplage. Ainsi, pour un couplage donné, le nombre de communications pour chaque nœud est d'au plus une. Une

décomposition doit pour être valide vérifier les propriétés suivantes :

- toute arête d'un couplage  $M_l$  est une arête de  $G$  :

$$\forall l \in 1, \dots, h, E_l \subseteq E$$

- toute arête qui n'appartient pas à un couplage a un poids nul :

$$\forall l \in 1, \dots, h, \forall e \in E | e \notin E_l, w_l(e) = 0$$

- les poids sont conservés :

$$\forall i \in V_1, \forall j \in V_2, m_{i,j} = \sum_{l=1}^h w_l(i,j)$$

On peut ainsi décomposer le graphe de la figure 3.13 en 3 couplages, comme illustré sur la figure 3.14. Nous considérons dans un premier temps cet exemple avec comme paramètre  $\beta = 0,1$ . On peut ainsi lire sur cette figure,  $w_1((1, B)) = 3$ ,  $w_1((2, C)) = 5$ ,  $w_2((1, A)) = 1$ , etc...

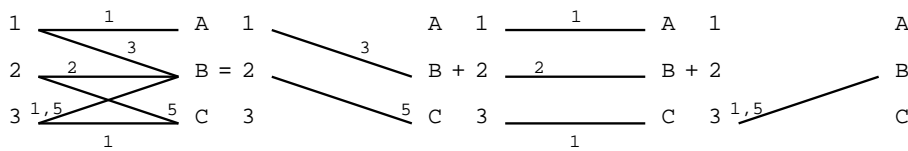


FIG. 3.14 – Décomposition d'un graphe de communication en couplages

On peut noter que sur cet exemple, aucune communication ne se trouve effectuée en plusieurs étapes : nous n'utilisons pas la possibilité de préempter les communications. On a ainsi :  $\forall l \in 1, \dots, h, \forall e \in E_l, w_l(e) = w(e)$  ou  $w_l(e) = 0$ .

Un des premiers intérêts d'une décomposition de ce type est qu'il est possible de lui associer un coût en temps à l'exécution. Ainsi, toujours sur l'exemple de la figure 3.14, la première étape de la redistribution va durer cinq secondes : durant les trois premières secondes de cette étape, les communications entre les nœuds 1-B et 2-C prennent place simultanément. A la fin de ces trois secondes, la communication 2-C dure encore deux autres secondes jusqu'à sa fin. La seconde étape dure deux secondes, etc ... On peut représenter le déroulement de cette redistribution par le diagramme de Gantt de la figure 3.15.

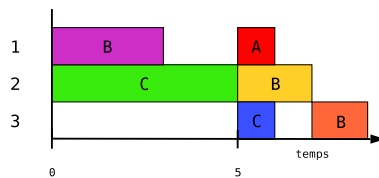


FIG. 3.15 – Déroulement d'une redistribution en plusieurs étapes

Au temps pris pour les communications en elles-même il convient néanmoins d'ajouter également un surcoût fixe pour chaque étape, lié au délai nécessaire pour leur établissement. Ce coût vaut  $\beta$  pour chacune des 3 étapes soit  $3 \times \beta$  pour l'ensemble de la redistribution. On arrive

ainsi sur notre exemple à un coût final pour notre décomposition  $D$  de  $5 + 2 + 1,5 + 3\beta = 8,5 + 3 \times 0,1 = 8,8$  secondes.

Dans le cas général, le temps estimé pour une étape d'une redistribution  $c(M_l)$ , représentée par un couplage  $M_l = (E_l, w_l)$  a pour valeur la durée de sa communication la plus longue, soit  $\max_{e \in E_l}(w_l(e))$ . Ainsi le temps total  $c(D)$  aura pour valeur la somme des temps de chaque étape augmenté des délais d'initialisation :  $c(D) = \sum_{l=1}^n c(M_l) + n \times \beta = \sum_{l=1}^n (\max_{e \in E_l}(w_l(e))) + n \times \beta$ .

Le temps pris par chaque étape ne dépendant que de la durée de la communication la plus longue, ceci signifie qu'une décomposition regroupant des communications de taille identique dans une même étape se révélera meilleure à l'exécution qu'une redistribution regroupant ensemble des communications de tailles différentes. La figure 3.16 permet d'illustrer facilement cette idée. On peut voir sur la partie gauche de la figure un diagramme de Gantt correspondant à l'exécution d'une redistribution en deux étapes, en regroupant ensemble à chaque fois deux communications de tailles différentes. Le temps total pris par une telle exécution est de 10 secondes. La partie droite de la figure montre la même redistribution en regroupant cette fois-ci les communications de taille identique lors d'une même étape. Le temps total est alors de 6 secondes. La différence de 4 secondes entre les deux temps obtenus s'explique par une utilisation sous-optimale de la bande passante disponible dans le premier cas : lorsqu'une communication se termine, la bande passante qu'elle utilisait ne peut être réutilisée que lors de l'étape suivante.

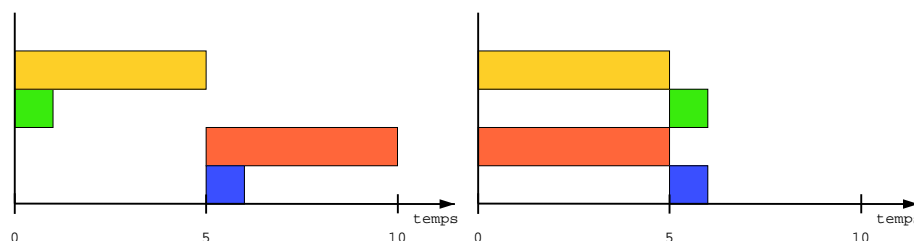


FIG. 3.16 – Comparaison entre deux ordonnancements simples

Ce phénomène pose un problème dans le cas où toutes les communications ont des longueurs différentes, signifiant qu'une exécution par étapes implique forcément des attentes et donc une perte de performances. C'est ici que rentre en jeu la préemption. Afin d'éviter des attentes trop longues entre fin d'une communication et fin d'une étape, nous autorisons le découpage d'une communication en plusieurs communications plus courtes. Ainsi on peut voir figure 3.17 deux découpages différents d'une redistribution. On considère ici  $\beta = 0$ . De plus, il n'est, pour des raisons de bande passante, pas possible d'effectuer plus de deux communications simultanées. Sur la gauche de la figure, un ordonnancement n'utilisant pas la préemption, d'un temps optimal de 7 secondes, et sur la droite un ordonnancement utilisant la préemption en découplant la communication de durée 5 en 5 communications de durée 1. La version avec préemption ne prend qu'un temps de 5 secondes. L'écart en temps entre version préempté ou non peut ainsi monter jusqu'à un facteur 2. On peut remarquer au passage que l'utilisation de la préemption a tendance à faire augmenter le nombre d'étapes de la décomposition.

L'utilisation de la préemption s'effectue à l'aide de la fonction de valuation associée à chaque couplage de la décomposition. Ainsi, en reprenant l'exemple de la figure 3.14 (toujours avec  $\beta = 0,1$ ) on peut obtenir la décomposition optimale de la figure 3.18 en découplant la communication entre 2 et C en deux communications de durées  $w_1((2, C)) = 3$  et  $w_3((2, C)) = 2$ . Le temps pris ici n'est cette fois plus que de 7,3 secondes.

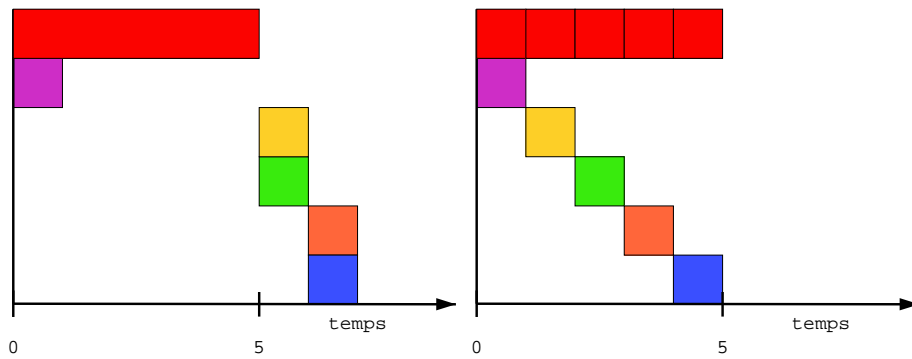


FIG. 3.17 – Comparaison entre ordonnancements sans et avec préemption

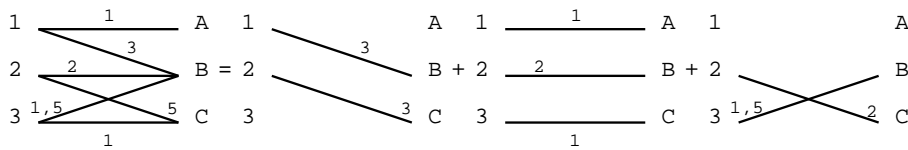


FIG. 3.18 – Décomposition optimale en utilisant la préemption

### 3.2.3 Le problème KPBS

Ayant défini l'approche utilisée pour modéliser une redistribution de données exécutée en plusieurs étapes, et vérifié la validité du modèle choisi, nous présentons maintenant de manière formelle le problème KPBS, au cœur de nos travaux.

#### Énoncé du problème

On cherche ici à atteindre deux objectifs différents :

- minimiser le temps pris par une redistribution ;
- ne pas dépasser les capacités du réseau.

KPBS se formule donc ainsi : étant donné  $\beta, b_1, b_2, b_b$ , ainsi qu'un graphe de communication, trouver une décomposition de la redistribution en plusieurs étapes telle que son temps d'exécution soit minimal et qu'aucun lien ne soit jamais saturé.

On peut plus précisément distinguer deux types de contraintes sur les différents liens du réseau :

- contraintes sur les liens locaux ;
- contraintes sur la dorsale.

La contrainte posée par le réseau sur les liens locaux, est résolue en limitant le nombre de communications pour chaque nœud à une seule. On se place ici dans le cadre du modèle 1-port. Exprimer cette contrainte sur un graphe de communications est effectué à l'aide de couplages. En effet, le degré de chacun des sommets touché par un couplage ne peut dépasser 1. On peut noter que certaines configurations physiques se prêtent mal à l'utilisation du modèle 1-port : par exemple lorsque les bandes passantes locales à chaque grappe sont différentes comme sur la figure 3.4. Dans ce dernier cas, en particulier les liens à 1Gbit/s de la grappe réceptrice peuvent recevoir 10 communications à 100Mbit/s en parallèle. Ces cas particuliers seront traités chapitre 5.

Ne pas saturer la dorsale signifie éviter que la somme des bandes passantes de tous les flux la traversant ne dépasse la bande passante disponible. Les flux disposant tous de la même vitesse, ce problème correspond à limiter le nombre de flux parallèles afin de ne jamais dépasser les ressources disponibles. On appelle  $k$  le nombre maximal de communications autorisées en parallèle.  $k$  est calculé à partir des paramètres réseau en divisant la bande passante de la dorsale par la bande passante de chaque flux :

$$k = \frac{b_b}{\min(b_1, b_2, b_b)}$$

Notons que lorsque  $k$  n'est pas entier il est nécessaire de l'arrondir à l'entier inférieur ou supérieur. Choisir l'entier inférieur signifie ne pas utiliser toute la bande passante disponible, tandis que l'entier supérieur génère de la contention sur le réseau. Il est possible (et même souvent souhaitable) de déterminer expérimentalement la valeur de  $k$  en l'augmentant progressivement jusqu'à ce que la bande passante agrégée reste constante. Les expériences de saturation de la figure 3.8 illustrent ainsi la marche à suivre : la valeur de  $k$  est ici fixée à 12. Pour l'ensemble des exemples de ce chapitre, le problème d'arrondi ne se posera pas.

La contrainte de non saturation de la dorsale impose donc une nouvelle contrainte pour chaque couplage d'une redistribution par étapes en limitant à  $k$  le nombre maximal d'arêtes qu'il contient.

**Définition 1** *Le problème KPBS est défini comme suit :*

Entrée :  $G$ , un graphe de communication,  $\beta \in \mathbb{Q}$  la latence d'une communication,  $k \in \mathbb{N}$  le nombre maximal de communications autorisées simultanément ;

Sortie :  $D = \{M_1 = (E_1, w_1), \dots, M_h = (E_h, w_h)\}$  un ensemble de  $h$  couplages validant les contraintes présentées section 3.2.2. De plus, le nombre d'arêtes de chaque couplage est d'au plus  $k$  :  $\forall l \in 1, \dots, h, |M_l| \leq k$ . Objectif : Minimiser le temps total pris par l'exécution de la décomposition valant  $c(D) = \sum_{l=1}^h (\max_{e \in E_l} (w_l(e))) + h \times \beta$ .

### Propriétés de complexité

Cohen et al. ont prouvé dans [25] que le problème KPBS est NP-complet, au sens fort, par réduction avec le problème de 3-partition.

### Borne inférieure sur le temps de redistribution

Un des premiers intérêts d'une formulation du problème KPBS en terme de graphes est de pouvoir obtenir une borne inférieure sur  $c(G)$ , le temps minimal pris par l'exécution de la redistribution du graphe  $G$ . Cette borne  $\eta(G)$  va, par la suite, nous permettre d'évaluer la qualité des résultats obtenus par nos différents algorithmes.

L'étude des bornes inférieures sur le temps de redistribution va nécessiter de définir de nouvelles notations mathématiques que nous utiliserons également ultérieurement : Nous prenons pour convention de noter  $p : V_1 \cup V_2 \rightarrow \mathbb{Q}$  la fonction qui à un sommet  $s$  associe la somme des poids de toutes les arêtes incidentes à  $s$  :  $\sum_{s' \in V_2} w((s, s'))$  si  $s \in V_1$  et  $\sum_{s' \in V_1} w((s', s))$  si  $s \in V_2$ . Avec cette notation, on peut définir  $W(G) = \max_{s \in V_1 \cup V_2} p(s)$ , la plus grande des sommes des communications pour chaque sommet. Enfin, nous définissons  $P(G)$  qui est la somme des poids de toutes les arêtes du graphe  $G$  :  $P(G) = \sum_{e \in E} w(e)$ .

Le calcul de  $\eta(G)$  est effectué à partir de deux bornes plus simples :

- $\eta_d(G)$  : une borne sur le temps pour effectuer les communications (sans coût d'initialisation),

- $\eta_s(G)$  : une borne sur le nombre d'étapes.
- On définit alors simplement  $\eta(G) = \eta_d(G) + \beta \times \eta_s(G)$ .

**Proposition 1**  $W(G)$  est une borne inférieure sur le temps pris par les communications.

**Preuve de la Proposition 1** Quelle que soit  $D = \{M_1, \dots, M_h\}$  une décomposition en couplages solution du problème KPBS, le temps de transfert des données pour  $D$  est égal à la somme des temps de transfert de chacun de ses couplages soit  $\sum_{l=1}^h c(M_l) = \sum_{l=1}^h \max_{e \in E} (w_l(e))$ . De plus, la décomposition conservant les poids des arêtes on a :  $\forall e \in E, w(e) = \sum_{l=1}^h w_l(e)$ . Soit  $s \in V_1$ . On a  $\sum_{(s,j) \in E} w(s,j) = p(s) = \sum_{(s,j) \in E} \sum_{l=1}^h w_l((s,j)) = \sum_{l=1}^h \sum_{(s,j) \in E} w_l((s,j))$  par commutativité de l'addition. Chaque  $M_l$  est un couplage, par conséquent, pour  $l$  fixé, la somme  $\sum_{(s,j) \in E} w_l((s,j))$  ne contient qu'un seul terme non nul, et donc  $\sum_{(s,j) \in E} w_l((s,j)) \leq \max_{e \in E} (w_l(e))$ . On en tire donc immédiatement  $p(s) \leq \sum_{l=1}^h c(M_l)$ . Le même raisonnement tient de manière similaire pour  $s \in V_2$ . L'inégalité trouvée étant valable quel que soit  $s$ , on a en particulier :  $\max_{s \in V_1 \cup V_2} (p(s)) \leq \sum_{l=1}^h c(M_l)$  soit  $W(G) \leq \sum_{l=1}^h c(M_l)$ . ■

Reprenons par exemple le graphe de communication de la figure 3.18. Si l'on examine le sommet 2, on voit qu'il a deux communications à effectuer :

- (2,B) de durée 2,
- (2,C) de durée 5.

D'après la contrainte posée par le modèle 1-port, ces deux communications ne peuvent avoir lieu en même temps et doivent donc être effectuées en séquence, pour un temps total de 7 secondes. On sait donc que quelle que soit l'ordonnancement choisi, le temps total pris par la redistribution ne pourra pas prendre moins de 7 secondes.

**Proposition 2**  $\Delta(G)$  est une borne inférieure sur le nombre d'étapes.

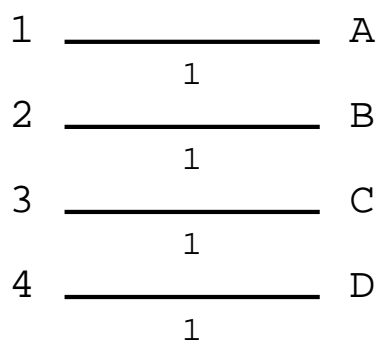
**Preuve de la Proposition 2** Si l'on effectue la décomposition en couplages sans utiliser la préemption, le problème est équivalent à un problème de coloriage d'arêtes. Le coloriage d'arêtes dans un graphe biparti  $G$  est un résultat classique [61] de la théorie des graphes et nécessite  $\Delta(G)$  couleurs. L'utilisation de la préemption ne peut que rajouter des arêtes supplémentaires, et donc augmenter le nombre d'étapes. ■

Toujours sur le même exemple, un examen du sommet B permet de déduire que toute décomposition comprendra un minimum de 3 étapes : en effet, le degré de ce sommet est de 3. En utilisant la même hypothèse du modèle 1-port que précédemment cela signifie que ces 3 communications doivent être exécutées séquentiellement et donc que la redistribution comporte un minimum de 3 étapes. Ceci se traduit en terme de temps par un temps minimal de  $3\beta$ .

En cumulant les deux temps minimaux on obtient un temps de  $7 + 3\beta = 7 + 3 \times 0,1 = 7,3$  soit le temps pris par la décomposition de la figure 3.18. On en déduit donc immédiatement que cette dernière est optimale.

On peut remarquer que les bornes ainsi obtenues ne prennent pas en compte  $k$ . Par exemple, considérons le graphe de la figure 3.19, avec  $k = 2$  et  $\beta = 1$ .

Une utilisation directe de nos formules nous donne  $\eta_d(G) = 1$ ,  $\eta_s(G) = 1$  d'où  $\eta(G) = 2$ . Ceci forme bien une borne inférieure sur le temps de redistribution : il est clair qu'il n'est pas possible d'effectuer toutes les communications en moins de 2 secondes. Néanmoins, il est possible d'affiner cette borne.

FIG. 3.19 – Graphe de communication simple, limité par  $k$ 

En effet, on dénombre au total 4 arêtes dans le graphe de communication. La limite de deux communications simultanées posée par  $k$  signifie clairement qu'il est nécessaire d'effectuer au moins  $\frac{4}{k} = 2$  étapes de communication. On peut donc augmenter  $\eta_s(G)$  à 2. D'une manière similaire, la somme des poids de chaque communication, notée  $P(G) = \sum_{e \in E} w(e)$  vaut ici 4. Sachant qu'au plus  $k$  communications peuvent avoir lieu à chaque seconde, il faudra un minimum de  $\frac{4}{k} = 2$  secondes pour effectuer toutes les communications. Une valeur de 2 pour  $\eta_d(G)$  nous donne ainsi une borne inférieure plus proche de la valeur de  $c(G)$ .

On obtient ainsi  $\eta(G) = \eta_d(G) + \beta\eta_s(G) = 2 + 2 = 4$  secondes comme borne inférieure sur la durée de la redistribution. Notons au passage qu'il s'agit ici de la valeur exacte de  $c(G)$ , cette borne étant atteinte par un ordonnancement de deux fois deux communications.

**Proposition 3**  $\frac{P(G)}{k}$  est une borne inférieure sur le temps pris par les communications.

**Preuve de la Proposition 3** De manière similaire à la preuve de la Proposition 1 le temps de transfert des données pour une redistribution  $D$  est égal à  $\sum_{l=1}^h c(M_l) = \sum_{l=1}^h \max_{e \in E} (w_l(e))$ . On a de plus,  $P(G) = \sum_{e \in E} w(e) = \sum_{e \in E} \sum_{l=1}^h w_l(e) = \sum_{l=1}^h \sum_{e \in E} w_l(e)$ . Comme tout couplage contient un maximum de  $k$  arêtes, la somme des poids des arêtes qu'il contient vaut au plus  $k$  fois le poids de son arête de poids le plus élevé :  $\forall l, \sum_{e \in E} w_l(e) \leq k \times \max_{e \in E} (w_l(e))$ . Par conséquent  $P(G) \leq k \times \sum_{l=1}^h c(M_l)$  et donc  $\frac{P(G)}{k} \leq \sum_{l=1}^h c(M_l)$  ■

**Proposition 4**  $\left\lceil \frac{m(G)}{k} \right\rceil$  est une borne inférieure sur le nombre d'étapes.

**Preuve de la Proposition 4** Si l'on effectue la décomposition en couplages sans utiliser la préemption, chaque couplage contenant au plus  $k$  arêtes, il faut au moins  $\left\lceil \frac{m}{k} \right\rceil$  couplages pour effectuer une décomposition. L'utilisation de la préemption ne peut, encore une fois, que rajouter des arêtes supplémentaires, et donc augmenter le nombre d'étapes. ■

**Proposition 5**  $\eta_d(G) = \max(W(G), \frac{P(G)}{k})$  est une borne inférieure sur le temps pris par les communications.

**Preuve de la Proposition 5**  $W(G)$  et  $\frac{P(G)}{k}$  sont deux bornes inférieures sur le temps de communication, par conséquent leur maximum également. ■

**Proposition 6**  $\eta_s(G) = \max(\Delta(G), \lceil \frac{m(G)}{k} \rceil)$  est une borne inférieure sur le nombre d'étapes.

**Preuve de la Proposition 6**  $\Delta(G)$  et  $\lceil \frac{P(G)}{k} \rceil$  sont deux bornes inférieures sur le nombre d'étapes, par conséquent leur maximum également. ■

**Théorème 1**  $\eta(G) = \eta_d(G) + \beta \times \eta_s(G)$  est une borne inférieure sur le coût  $c(D)$  de toute décomposition  $D$  de  $G$ .

**Preuve du Théorème 1** Par définition,  $c(D) = \sum_{l=1}^n c(M_l) + \beta \times n$ . Or d'après la Proposition 5  $\eta_d(G) \leq \sum_{l=1}^n c(M_l)$  et d'après la Proposition 6  $\eta_s(G) \leq n$ . ■

### 3.2.4 Problèmes en relation avec KPBS

Le problème KPBS a la particularité d'étendre de nombreux problèmes existants dans la littérature. Nous présentons ici quelques-uns d'entre eux. Cette sous-section poursuit deux objectifs différents : tout d'abord KPBS étant un problème plus général que les problèmes plus anciens que nous rappelons ici, tout algorithme permettant de résoudre KPBS peut également être utilisé pour les autres problèmes. De plus, une présentation des algorithmes utilisés dans la littérature va nous permettre d'introduire des notions que nous utiliserons ultérieurement.

Nous présentons dans un premier temps le problème PBS, dont KPBS dérive directement (et tire son nom), puis différents problèmes liés aux commutateurs satellitaires à multiplexage temporel (*SS/TDMA : Satellite Switched / Time Division Multiple Access*) et à des réseaux optiques utilisant un multiplexage par fréquences (*Wavelength Division Multiplexing*). Nous terminons en montrant les liens entre KPBS et un problème classique d'ordonnancement : le problème d'ordonnancement d'atelier sur ateliers à cheminements libres (*Open Shop*).

Notons au passage que KPBS est suffisamment générique pour toucher différents domaines de l'informatique.

#### Le problème PBS

Crescenzi et al. ont introduit le problème PBS en 1998 [26]. Il s'agit d'optimiser des communications entre un ensemble d'émetteurs et un ensemble de récepteurs étant soumis à la contrainte 1-port, la préemption étant autorisée. Chaque étape a un coût d'initialisation  $\beta$ ; initialement tous les temps de communication sont normalisés par  $\beta$ . L'ensemble des communications est modélisé ici à l'aide d'un graphe biparti  $G$ . Le problème consiste donc à décomposer un graphe de communication en une série de couplages correspondant chacun à une étape de communication. On cherche bien entendu à minimiser le temps total de communication.

En fait, la différence entre PBS et KPBS est donnée par la contrainte de KPBS sur le nombre limite de  $k$  communications simultanées. Ainsi toute instance de PBS peut être résolue par n'importe quel algorithme résolvant KPBS en prenant  $k = n$ .

Crescenzi et al. montrent qu'approximer le problème PBS à moins de  $\frac{7}{6}$  est un problème NP-complet. Ils introduisent ensuite une borne inférieure de  $\Delta(G) + W(G)$  pour le coût d'une solution. On peut remarquer que cette borne est semblable à la borne pour KPBS introduite en section 3.2.3 où l'on enlèverait les termes utilisant  $k$ . Finalement ils présentent un algorithme d'approximation de ratio 2, suivi par une famille d'algorithmes de même ratio, mais supposés d'une efficacité croissante.

Nous rappelons qu'un algorithme d'approximation de ratio 2 (ou 2-approximation) est un algorithme garantissant que toute solution qu'il donne a un coût inférieur ou égal au double du coût d'une solution optimale.

Nous présentons rapidement le principe derrière le premier algorithme proposé car il sera réutilisé dans la suite de nos travaux. Notons au passage, qu'Afrati et al. ont proposé [10] un algorithme d'approximation légèrement différent, d'un ratio de  $2 - \frac{1}{\beta+1}$  pour le problème PBS.

Dans un premier temps Crescenzi et al. introduisent un algorithme pseudo-polynomial sur lequel ils démontrent la propriété de 2-approximation. Ensuite, ils dérivent de cet algorithme un algorithme polynomial équivalent. L'algorithme pseudo-polynomial est le suivant :

1. construire un multigraphe  $H$  en divisant chaque arête de poids  $w(e)$  en  $\lceil w(e) \rceil$  arêtes de poids 1 ;
2. ajouter des arêtes pour obtenir un multigraphe  $\Delta(H)$  régulier ;
3. décomposer en  $\Delta(H)$  couplages parfaits, chacun de poids 1.

On rappelle qu'un graphe est dit  $a$ -régulier si chacun de ses sommets a le même degré  $a$ . Nous reviendrons plus en détail section 4.1 sur la manière d'obtenir un multigraphe régulier, ainsi que sur la manière d'obtenir la décomposition.

La figure 3.20 illustre sur un exemple la construction de  $H$  ainsi que la solution obtenue. On peut remarquer que le degré de  $H$  valant 4, la décomposition comprend 4 étapes de communication.

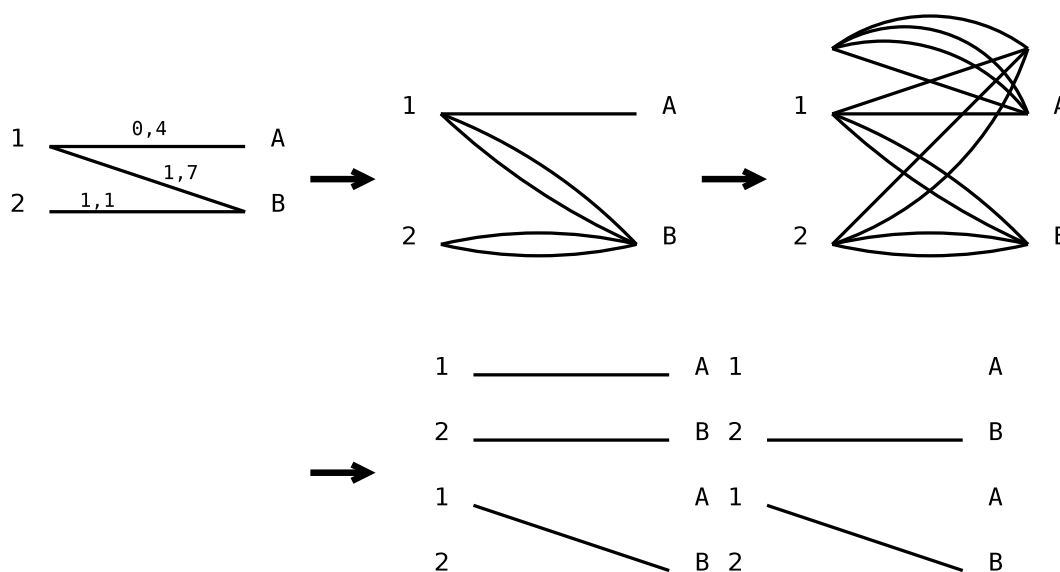


FIG. 3.20 – Exécution de l'algorithme pseudo-polynomial de résolution de PBS

On obtient ainsi une solution de  $\Delta(H)$  étapes de coût 2 : une durée de 1 + un coût d'initialisation de 1. Le coût total de la solution est donc  $2\Delta(H)$ .

De plus, pour tout sommet  $v$ , son degré dans  $H$  est  $\sum_{e=(v,u)} \lceil w(e) \rceil$  ce qui est inférieur à la somme des poids de tous les arcs incidents plus le degré de  $v$  (le degré apparaît ici car il y a au plus un arrondi de 1 pour chaque arête incidente à  $v$ ). On a donc  $\Delta(H) \leq \Delta(G) + W(G)$  ce qui nous donne un coût pour la solution inférieur à  $2(\Delta(G) + W(G))$  soit deux fois la borne inférieure. Cet algorithme est donc bien une 2-approximation.

Cet algorithme est pseudo-polynomial car le graphe  $H$  contient un nombre d'arêtes fonction des poids des arêtes du graphe  $G$ . Dans un soucis d'utilisation réelle il est donc important de réduire sa complexité. Pour ce faire, Crescenzi et al. proposent l'algorithme suivant :

1. construire un graphe  $G' = (G, w')$  en arrondissant le poids de chaque arête à l'entier supérieur ;
2. ajouter des arêtes pour obtenir un graphe  $W(G')$  régulier sur les poids ;
3. décomposer en couplages parfaits de poids identiques.

Il s'agit ici en fait d'utiliser exactement le même algorithme que précédemment, mais sans effectuer la construction d'un multigraphe. Un graphe est dit  $\alpha$ -régulier sur les poids (*weight-regular*) si pour chacun de ses sommets la somme des poids de toutes les arêtes incidentes est égale à  $\alpha$  (autrement dit, dans le cas présent, si le multigraphe obtenu en scindant toute arête de poids  $w(e)$  en  $w(e)$  arêtes de poids 1 est régulier). Les étapes de l'algorithme sont identiques (arrondi des poids, extension du graphe, décomposition en couplages parfaits), mais dans un cas on raisonne sur un multigraphe régulier et dans l'autre sur un graphe régulier sur les poids. La figure 3.21 illustrant l'exécution de l'algorithme polynomial sur le même exemple que précédemment permet de comparer les deux exécutions. On remarquera que le graphe étendu avec de nouvelles arêtes correspond au multigraphe étendu de l'algorithme pseudo-polynomial. On peut également remarquer que les couplages identiques de la solution se retrouvent ici fusionnés en un seul couplage de taille double.

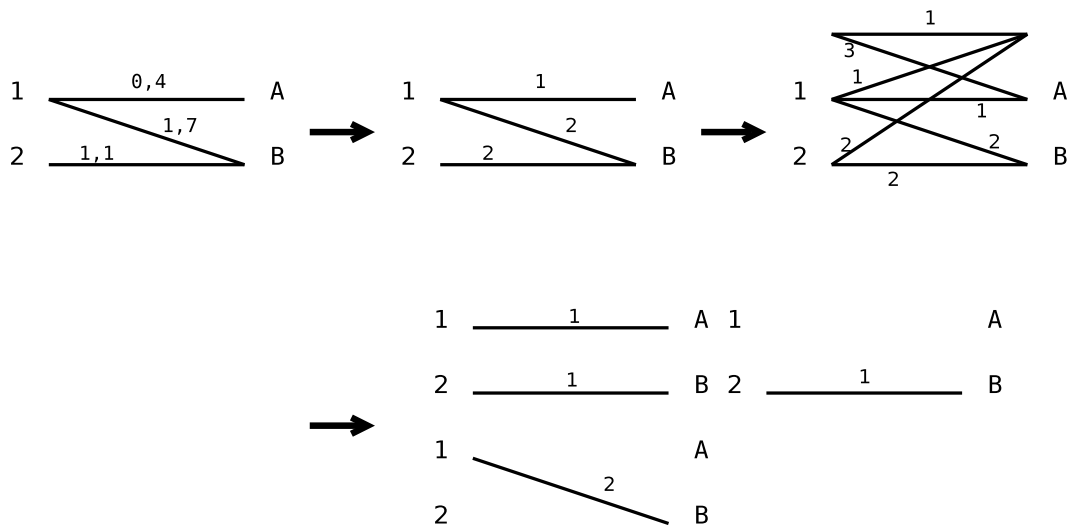


FIG. 3.21 – Exécution de l'algorithme polynomial de résolution de PBS

### Différents problèmes de communications dans les systèmes SS/TDMA

KPBS est un problème rencontré pour l'ordonnancement des communications dans les commutateurs satellitaires. Bongiovanni et al. [19] ont ainsi étudié le problème suivant : on considère un satellite traitant des flux de données qui lui sont émis depuis la Terre, et les redirigeant vers le sol. Le satellite doit être capable de réémettre le plus rapidement possible les flux qu'il reçoit. Les données sont divisées en paquets, et un multiplexage temporel permet d'interlacer les différentes communications étant destinées à la même cible. Les reconfigurations des routes entre

sources et destinations se font de manière synchronisées et coûte une latence  $\beta$ . Enfin, le nombre de flux retransmis simultanément est limité par les capacités du commutateur à un nombre  $k$  de transmissions. On cherche à minimiser la durée des communications (sans les délais  $\beta$ ), ainsi que le nombre total d'étapes de communication. Une modélisation à l'aide de graphes bipartis permet donc bien de se retrouver avec une instance du problème KPBS.

Bongiovanni et al. présentent un algorithme fonctionnant directement sur les matrices de communication et permettant d'atteindre  $\eta_d$  : la borne inférieure sur la durée des communications (toujours sans prendre en compte les coûts d'initialisation des différentes étapes). Ils calculent également une borne sur le nombre maximal d'étapes obtenues et concluent par des simulations montrant les bonnes performances de leur algorithme,  $\beta$  étant relativement faible.

Gopal et al. proposent [41] une extension de ce problème au cas où les différents émetteurs et récepteurs à bord du satellite ne disposent pas d'une bande passante identique, et peuvent donc supporter plus d'une communication simultanément. La résolution se fait par la création de sommets virtuels supplémentaires, en divisant un sommet disposant de  $n$  fois plus de bande passante que les autres en  $n$  sommets virtuels. La résolution s'effectue ensuite avec l'algorithme de Bongiovanni et al. Nous étudierons également section 5.1 cette extension du problème KPBS en utilisant une technique similaire.

### Différents problèmes de communications dans les réseaux optiques

Il existe toute une série de problèmes d'ordonnement de communications dans des réseaux optiques qui présentent des similarités avec le problème KPBS. Nous décrivons ici plusieurs de ces problèmes et montrons comment tout algorithme de résolution de KPBS peut être également utilisé pour les résoudre.

On s'intéresse ici à des réseaux optiques, et plus particulièrement les réseaux basés sur une étoile optique de diffusion et sélection (*optical broadcast and select star*). Une étoile optique de diffusion et sélection est un équipement réseau situé à mi-chemin entre un concentrateur et un commutateur. Elle dispose d'un ensemble d'entrées et de sorties et se charge de transmettre chaque flux réseau vers la bonne destination, le plus rapidement possible. Pour ce faire l'étoile utilise un multiplexage par longueur d'onde (*wavelength division multiplexing*) : chaque entrée émet son signal sur une certaine longueur d'onde, et toutes les sorties configurées pour cette longueur d'onde reçoivent le signal. Il est possible de reconfigurer certaines entrées ou sorties afin de changer de longueur d'onde, au prix d'un délai relativement important pouvant atteindre quelques millisecondes. Une entrée/sortie n'est pas obligée d'être configurable et peut être bloquée sur une certaine fréquence. Dans tous les problèmes présentés ici, tous les émetteurs ont une fréquence configurable. On distinguera donc deux configurations différentes : lorsque les récepteurs sont également configurables ou lorsqu'ils sont à fréquence fixe. Un exemple d'une telle étoile est illustré figure 3.22.

Notons de plus que les communications sont divisées en paquets et qu'il est possible d'utiliser sur chaque longueur d'onde un multiplexage temporel (*Time Division Multiplexed*). Enfin, certains équipements permettent à un émetteur ou un récepteur de transmettre plusieurs flux simultanément.

On voit déjà ici apparaître les points communs qui vont se présenter entre les problèmes d'ordonnements de communications dans de tels réseaux et le problème KPBS : les changements de configuration impliquent des délais importants qu'il est nécessaire de prendre en compte (ce qui correspond au paramètre  $\beta$  pour KPBS) et le nombre de communications parallèles est limité par le nombre de longueurs d'onde disponibles que l'on nommera ici  $k$ , comme pour KPBS. Enfin la division des communications en paquet est similaire à l'hypothèse autorisant la préemption

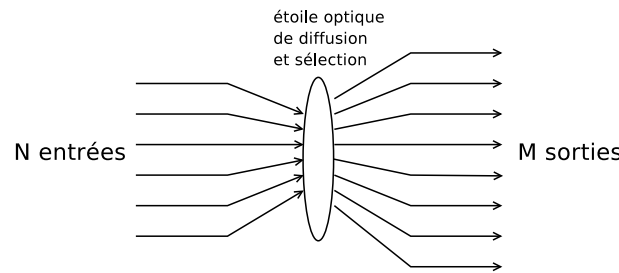


FIG. 3.22 – Étoile optique de diffusion et sélection

des communications pour KPBS.

La figure 3.23 illustre par un diagramme de Gantt ce à quoi peut ressembler un ordonnancement des paquets dans un réseau optique. Chaque ligne correspond ici à un émetteur, et on note sur chaque paquet le récepteur correspondant. On considère 6 émetteurs nommés 1, . . . , 6 et 6 récepteurs nommés A, . . . , F. Trois fréquences sont disponibles :  $\omega_1, \omega_2, \omega_3$  et les récepteurs ont une configuration fixe. A et B ne peuvent ainsi recevoir que sur la fréquence  $\omega_1$ , C et D uniquement sur  $\omega_2$  et E et F sur  $\omega_3$ . Les paquets grisés signifie que l'émetteur ne peut pas émettre. On remarque qu'à tout instant seuls trois paquets peuvent être transmis simultanément. De plus il n'y a jamais deux paquets transitant simultanément sur la même fréquence.

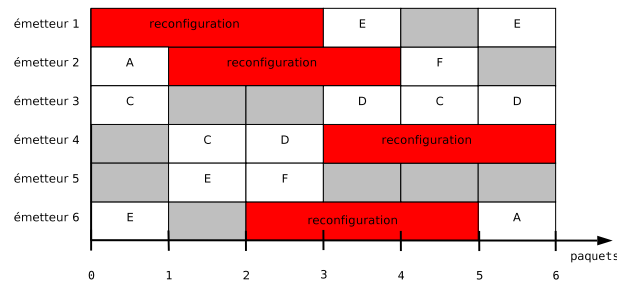


FIG. 3.23 – Exemple d'ordonnancement sur un réseau à étoile optique

La figure 3.23 montre également la différence entre le problème KPBS et les problèmes d'ordonnancement sur une étoile optique. En effet, le changement de configurations des longueurs d'onde des différents transmetteurs n'est pas obligatoirement synchronisé, ce qui signifie une possibilité de masquer les temps de configurations par d'autres communications. Pour le problème KPBS, les temps d'initialisation ( $\beta$ ) ne sont pas recouverts par des communications. Il est important de faire la distinction entre deux catégories d'études des réseaux optiques : les travaux les plus anciens n'utilisant pas de recouvrement configurations / communications, appelé dans la littérature pipelining des latences des configurations (*pipelining tuning latencies*). Ces travaux se focalisent donc sur des problèmes très similaires à KPBS. Les temps de configurations étant néanmoins particulièrement conséquents, l'utilisation des recouvrements s'est imposé comme une bonne manière d'accélérer les communications. Une grande part des travaux sur les commutateurs optiques traitent donc de problèmes légèrement différents de KPBS.

**Algorithme d'ordonnancement de Ganz et al.** Ganz et al. ont proposé [38] une méthode d'ordonnancement pour les communications dans les réseaux optiques pour un problème en

partie identique à KPBS. Ils considèrent une étoile optique de diffusion et sélection où chaque émetteur (resp. récepteur)  $i$  dispose de  $t_i$  (resp.  $r_i$ ) transmetteurs capables chacun d'émettre sur une longueur d'onde différente. On sort donc ici du modèle 1-port étant donné que chaque nœud est ici capable d'émettre ou recevoir plusieurs communications simultanément. Néanmoins, pour  $t_i = r_i = 1$ , le problème étudié par Ganz et al. est identique à KPBS. Notons au passage que nous étudions une extension du problème KPBS éliminant la contrainte 1-port au chapitre 5.

Les communications sont représentées ici par une matrice contenant pour chaque couple émetteur / récepteur le nombre de paquets à transmettre. Le nombre total de longueurs d'ondes disponibles sur le système est de  $\omega$ . Le but recherché est de décomposer la matrice de communication en une série de matrices validant les contraintes posées par  $t_i$ ,  $r_i$  et  $\omega$ . On cherche de plus en ensemble de matrices de configurations spécifiant quelles longueurs d'ondes utiliser pour chacun des nœuds. Chaque matrice de configuration correspond à une étape de reconfiguration d'une durée de  $\beta$ .

L'algorithme proposé fonctionne en deux étapes :

- une étape permettant de générer les matrices résultats ;
- une étape diminuant le nombre de matrices en configurations en réordonnant les étapes de communication.

La première étape de l'algorithme fonctionne de la manière suivante :

1. construction d'un graphe biparti correspondant à la matrice de communication ;
2. ajout d'arêtes virtuelles permettant de prendre en compte  $\omega$  ;
3. extractions de couplages tant qu'ils valident les contraintes sur  $t_i$  et  $r_i$  ; l'union de ces couplages forme une matrice de communication du résultat ;
4. boucle sur l'étape 3. jusqu'à ce que le graphe soit vide.

Nous ne rentrerons pas ici dans les détails des différentes étapes. Notons néanmoins que la seconde étape de l'algorithme est détaillée section 4.1.1 car utilisée également dans nos algorithmes. Enfin, nous pouvons noter également que  $\beta$  n'entre pas en jeu dans le fonctionnement de cet algorithme.

La seconde étape consiste à réordonner les matrices obtenues afin de minimiser le nombre de reconfigurations du réseau. Considérons par exemple les trois matrices de communication de la figure 3.24. On considère ici  $t_i = r_i = 1$  et  $\omega = 2$ . Effectuées dans cet ordre ces communications nécessitent trois matrices de configuration comme illustré figure 3.25 : on peut ainsi lire que lors de la première étape de communication, le nœud 1 dispose pour émettre vers le nœud A de la première des deux longueurs d'ondes disponibles.

nœuds	A	B
1	1	0
2	0	1

nœuds	A	B
1	0	2
2	3	0

nœuds	A	B
1	1	0
2	0	1

FIG. 3.24 – Décomposition en 3 étapes de communication

Il apparaît clairement que la première étapes sont identiques et pourraient être fusionnées si elles étaient exécutées de manière contiguë comme sur la figure 3.26. La seconde partie de l'algorithme de Ganz et al. permet de diminuer le nombre de reconfigurations en groupant ensemble un nombre maximum d'étapes de communications compatibles. Deux remarques paraissent importantes sur cette partie de l'algorithme : tout d'abord cette étape de fusion est utilisable quelle que soit la méthode utilisée pour obtenir les résultats initiaux. En particulier, elle est directement

nœuds	A	B
1	1	0
2	0	2

nœuds	A	B
1	0	1
2	2	0

nœuds	A	B
1	1	0
2	0	2

FIG. 3.25 – 3 matrices de configuration

réutilisable avec les résultats fournis par nos algorithmes. Une seconde remarque est que cette étape a pour but de minimiser l'impact des délais de configuration. En effet, le facteur  $\beta$  n'est pris en compte nulle part dans l'algorithme ce qui peut éventuellement impliquer de mauvaises performances dans le cas où sa valeur est élevée.

nœuds	A	B
1	0	2
2	3	0

nœuds	A	B
1	1	0
2	0	1

nœuds	A	B
1	1	0
2	0	1

⇒

nœuds	A	B
1	0	2
2	3	0

nœuds	A	B
1	2	0
2	0	2

FIG. 3.26 – Diminutions du nombre de matrices de configuration (nombre d'étapes)

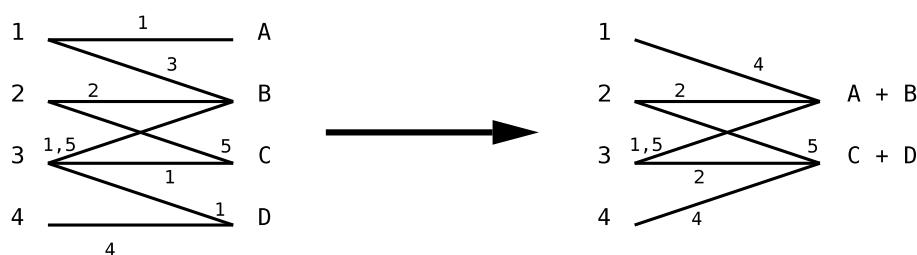
**Algorithme d'ordonnement de Choi et al.** Choi et al. ont également proposé [24] le calcul d'un ordonnancement sans pipelining en se basant sur des graphes bipartis. Leur algorithme n'est pas envisagé pour une utilisation réelle, mais sert au calcul de bornes inférieures sur la qualité des résultats.

Le problème étudié par Choi et al. est différent du problème de Ganz et al. car ils considèrent ici des récepteurs à fréquence fixe (*fixed tune*), ce qui signifie que seuls les émetteurs peuvent choisir leur fréquence d'émission. De plus les  $k$  différentes fréquences disponibles sont supposées réparties équitablement entre les différents récepteurs. Les communications sont modélisées par un graphe biparti similaire à celui utilisé pour KPBS, néanmoins, ne figurent sur le graphe que  $k$  récepteurs *virtuels* correspondant chacun à une longueur d'onde différente : en effet, les récepteurs ayant une fréquence fixe, toute émission vers une destination donnée touchera également tous les autres récepteurs ayant la même fréquence. Deux émetteurs ne peuvent donc pas émettre simultanément vers deux récepteurs sur la même fréquence, même si ceux-ci sont distincts. La figure 3.27 illustre ainsi la comparaison avec un graphe de communication classique.

La suite de l'algorithme ressemble à l'algorithme de Crescenzi et al. pour PBS décrit section 3.2.4 : le graphe de communication est transformé en multigraphe, puis un coloriage du graphe permet d'obtenir les différentes étapes de communication.

### Problèmes d'ordonnement sur ateliers à cheminements libres

Le problème d'ordonnement sur ateliers à cheminements libres, plus connu sous son nom anglais d'*open-shop*, est un problème classique d'ordonnement qui a donné lieu à différentes

FIG. 3.27 – Création de  $k$  sommets virtuels

variantes.

Le problème standard est le suivant : on considère ici un ensemble de  $m$  machines  $M_1, \dots, M_m$  sur lequel doivent être exécutées  $n$  tâches  $T_1, \dots, T_n$ . Chaque tâche  $T_i$  est composée de  $m$  sous-tâches  $O_{i,1}, \dots, O_{i,m}$  où la sous-tâche  $O_{i,j}$  ne peut être exécutée que sur la machine  $M_j$ . A chaque sous-tâche  $O_{i,j}$  est associé un temps d'exécution  $t_{i,j}$  sur la machine  $M_j$ . Le but recherché est de trouver un ordonnancement des sous-tâches tel qu'à un instant donné une seule machine exécute une tâche, et une seule tâche est exécutée par machine. Le problème consiste à trouver un ordonnancement de durée minimale.

Remarquons au passage que ce problème peut-être simplement modélisé par un graphe biparti où un ensemble de sommets représente les machines, l'autre ensemble les tâches et chaque arête représentant une sous-tâche est évaluée par sa durée. La figure 3.28 illustre ainsi une telle configuration.

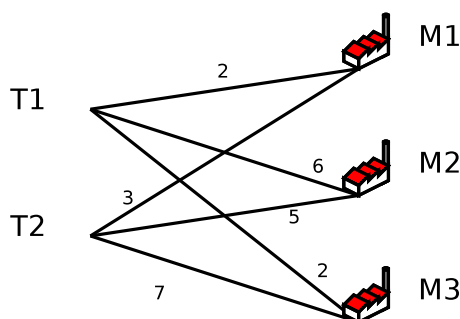


FIG. 3.28 – Une instance d'un problème d'open-shop

Avec une telle modélisation, le problème d'open-shop est donc fortement semblable au problème KPBS, mais notons néanmoins plusieurs différences majeures :

- il est ici impossible de préempter l'exécution des différentes sous-tâches ;
- il n'y a aucune limite au nombre de sous-tâches exécutables simultanément ;
- l'ordonnancement obtenu peut ne pas être découpé en étapes.

Ce problème a été montré NP-difficile pour un nombre de machines égal à 3 [40] et NP-difficile au sens fort [54] dans le cas général. De plus, le problème a été prouvé par Williamson et al. [69] non approximable à un facteur inférieur à  $\frac{5}{4}$  à moins que  $P=NP$ . Remarquons également qu'un algorithme glouton très simple, choisissant dès que possible la sous-tâche la plus longue restante (et exécutable) garanti un ratio d'approximation de 2 [64].

Notons que différentes variantes du problème ont été étudiées, certaines se rapprochant encore

un peu du problème KPBS en prenant notamment en compte la possibilité de préempter les différentes sous-tâches. Néanmoins, dans ces travaux [68, 40], aucun coût n'est associé à la préemption, ce qui rend le problème solvable en temps polynomial.

### 3.2.5 Expériences de validation du modèle

Afin de valider le modèle utilisé pour le problème KPBS, nous avons effectué une série d'expériences sur des redistributions entre deux grappes locales.

Ces expériences représentent nos premières contributions directes à l'étude de KPBS. Il nous apparaît en effet des plus importants de commencer nos travaux par une vérification de l'adéquation entre le modèle étudié et la réalité. L'ensemble des travaux théoriques développés par la suite ne sont en effet utilisables en pratique que si le modèle reflète correctement les comportements réels du réseau.

Ces tests ont été réalisés dans les mêmes conditions que les tests de la section 4.3.4. Nous considérons ici deux grappes locales, chacune machine disposant d'une interface réseau d'une bande passante de 100Mbit/s. Ces grappes sont reliées entre elles par deux commutateurs réseau capable également de soutenir 100Mbit/s. Dans ces conditions, une seule communication arrive à saturer la dorsale, et l'ordonnancement optimal est obtenu en sérialisant les communications. Afin de pouvoir effectuer des redistributions plus intéressantes, nous avons limité la bande passante de chaque interface réseau à 20Mbit/s à l'aide d'un module de qualité de service appelé : *rshaper* [7]. Dans ces conditions, 5 communications parallèles peuvent prendre place simultanément sans congestionner le réseau.

Nous avons considéré deux types de redistributions :

- des redistributions par étapes ;
- des redistributions par force brute, où toutes les communications sont exécutées simultanément.

Les temps de redistributions sont estimés dans le cas d'une redistribution par étape par le calcul du coût de la décomposition du graphe de communication choisie. Dans le cas d'une redistribution par force brute, nous proposons un algorithme (figure 1) permettant le calcul du temps de redistribution.

Cet algorithme nous permet d'estimer rapidement à partir d'un graphe de communication s'il est utile ou non d'ordonner les communications : il nous suffit de comparer le temps estimé pour une redistribution par force brute au temps estimé pour une redistribution ordonnancée.

L'algorithme, une simulation à événements discrets, fonctionne en simulant la redistribution étape par étape. Le principe est le suivant : comme la fraction de bande passante d'une interface dédiée à un flux de données ne dépend que du motif de redistribution, nous calculons pour chacun des flux la quantité de bande passante dont il dispose (en répartissant équitablement la bande passante d'une interface entre tous ses flux). Nous calculons ensuite le temps nécessaire pour terminer chaque communication. Le minimum  $t$  d'entre tous ces temps détermine quelle(s) communication(s) termine(nt) en premier. Nous simulons alors pour chaque flux  $t$  secondes de communication en diminuant de manière correspondante la quantité de données qu'il a à émettre. A ce point de la simulation le motif de communication a changé, et nous re-calculons donc la part des différentes bandes passantes allouées à chaque flux. Nous bouclons sur cette opération jusqu'à ce que toutes les communications aient été effectuées. Notons que cet algorithme est semblable à celui utilisé dans le simulateur de grille *Simgrid* [22] pour la simulation du réseau.

Afin de vérifier la validité de cet algorithme, tout comme celle du modèle de redistribution par étapes, nous avons effectué une série d'expériences visant à comparer dans différents cas, temps estimés et temps obtenus.

**Entrées :** un graphe de communication  $G = (V_1, V_2, E, w)$ ,  $k$  le nombre maximal de communications simultanées autorisées.

**Sorties :**  $\eta_o$  l'estimation du temps de redistribution.

$\eta_o = 0$ ;

**tant que** *il reste une arête dans  $E$*  **faire**

associer à chaque arête  $e$  de  $E$  une fraction de bande passante  $bw(e)$  d'une valeur de 0;  
construire  $l$  la liste de tous les sommets, ordonnée par degrés décroissants;

**pour chaque** *sommet  $n$  de  $l$*  **faire**

$bw$  représentant la bande passante de l'interface est égal à 1;

enlever de  $bw$  la part de bande passante  $bw(e_n)$  de chaque arête  $e_n$  incidente à  $n$ ;

calculer  $a$ , le nombre d'arêtes incidentes à  $n$  d'une bande passante de 0;

pour chaque arête  $e_n$  incidente à  $n$ , avec  $bw(e_n) = 0$ ,  $bw(e_n) = \frac{bw}{a}$ ;

**fin**

calculer pour chaque arête  $e$  le temps  $t(e) = \frac{w(e)}{bw(e)}$  nécessaire pour effectuer sa communication;

calculer  $t = \min_{e \in E}(t(e))$  le temps nécessaire pour la communication terminant en premier;

$\eta_o = \eta_o + t \times \max\left(\frac{\sum_{e \in E} bw(e)}{k}, 1\right)$ ;

pour chaque  $e$  dans  $E$  changer  $w(e)$  à  $w(e) - t \times bw(e)$ ;

enlever toutes les arêtes de poids 0 de  $E$ ;

**fin**

retourner  $\eta_o$ ;

**Algorithme 1 :** Algorithme d'estimation

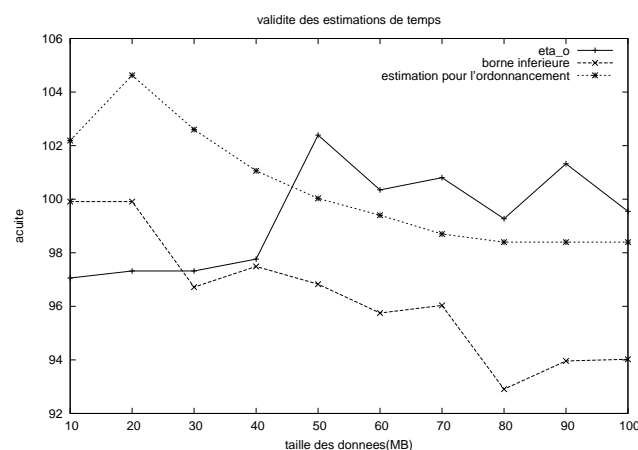


FIG. 3.29 – Validation du modèle : acuité de l'estimation des temps pour des graphes aléatoires

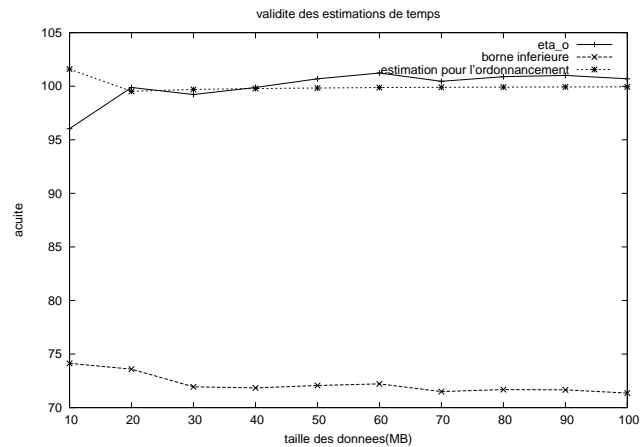


FIG. 3.30 – Validation du modèle : acuité de l'estimation des temps pour une famille particulière de graphes

Nous étudions pour chaque expérience trois rapports différents :

- le ratio entre  $\eta_o$  le temps estimé pour la redistribution par force brute et son temps d'exécution ;
- le ratio entre la borne inférieure sur les temps de communication et le temps d'exécution de la redistribution par force brute ;
- le ratio entre le temps estimé pour une redistribution ordonnancée et son temps d'exécution.

Chacun de ces ratios est ici multiplié par 100 pour permettre de visualiser directement un pourcentage d'acuité. Les figures 3.29 et 3.30 présentent les résultats obtenus dans le cas de redistributions purement aléatoire et dans le cas particulier d'une famille de graphes connue pour donner de mauvais résultats dans le cas d'une redistribution par force brute (telle la redistribution de la figure 3.7). Dans le cas de la famille de graphes, les graphes ne sont pas générés aléatoirement, mais leurs poids sont progressivement augmentés, ce qui explique une variation linéaire des temps de redistribution.

On observe aussi bien pour une redistribution ordonnancée que pour une redistribution par force brute une bonne acuité des prédictions de temps (avec quelques variations aléatoires allant jusqu'à 5 % du temps). En revanche, les bornes inférieures ne sont bien sûr pas suffisantes pour estimer un temps de redistribution par force brute.

Ces expériences nous permettent ainsi de conclure que le modèle de redistribution par étapes (ainsi que le modèle de coût associé) reflète à peu de choses près le comportement réel d'une redistribution de données ordonnancée. De plus, l'algorithme 1 de simulation d'une redistribution par force brute modélise également avec une bonne précision le comportement du réseau pour une redistribution cette fois-ci non ordonnancée.

## Chapitre 4

# Algorithmes d'approximation pour le problème KPBS

### Sommaire

---

<b>4.1</b>	<b>GGP</b> . . . . .	<b>61</b>
4.1.1	Principe de fonctionnement . . . . .	62
4.1.2	Algorithme . . . . .	69
4.1.3	Propriétés . . . . .	72
<b>4.2</b>	<b>OGGP</b> . . . . .	<b>80</b>
4.2.1	Principe de fonctionnement . . . . .	80
4.2.2	Algorithme formel . . . . .	81
4.2.3	Propriétés . . . . .	82
<b>4.3</b>	<b>Expériences</b> . . . . .	<b>84</b>
4.3.1	Une bibliothèque de manipulation de graphes bipartis : <i>libkpbs</i> . . . . .	84
4.3.2	Simulations . . . . .	84
4.3.3	Tests locaux exécutés avec la bibliothèque <i>MPICH</i> . . . . .	89
4.3.4	Tests locaux exécutés manuellement . . . . .	92
4.3.5	Conclusion . . . . .	94

---

Ce chapitre présente l'ensemble de nos travaux en relation directe avec le problème KPBS introduit au chapitre précédent. Nous commençons section 4.1 par décrire un algorithme polynomial offrant une garantie minimale de qualité sur les redistributions obtenues nommé GGP. Cet algorithme se trouve ensuite étendu Section 4.2 par l'algorithme OGGP, de complexité supérieure, mais fournissant des résultats de meilleure qualité. Enfin, la dernière section de ce chapitre traite des travaux réalisés en pratique, et notamment des tests réalisés en vue de comparer nos algorithmes avec une approche par force brute ou diverses heuristiques.

### 4.1 GGP

L'algorithme GGP (pour *Generic Graph Peeling* : épiluchage générique de graphe) est le premier algorithme de calcul d'une série de couplages que nous avons proposé. Cet algorithme étant relativement délicat à comprendre, nous commencerons, dans une première sous-section, par présenter les concepts et les mécanismes qu'il utilise. Nous introduirons également un algorithme d'extension de graphes en graphes réguliers sur les poids qui sera utilisé par GGP. Cette

première approche de l'algorithme nous permettra d'en mettre au clair une description formelle, illustrée par une exécution pas à pas sur un exemple. Enfin section 4.1.3 nous conclurons la présentation de GGP par une analyse de la complexité de l'algorithme, d'une borne sur la qualité des résultats, ainsi que par une présentation de quelques cas problématiques.

#### 4.1.1 Principe de fonctionnement

L'idée principale derrière GGP est tirée de l'algorithme de Crescenzi et al. [26] pour la résolution du problème PBS présenté section 3.2.4. Les deux algorithmes auront donc de fortes similitudes, notamment pour l'étude d'un ratio d'approximation où nous utiliserons également un algorithme travaillant sur des multigraphes.

L'algorithme GGP prend en entrée un graphe de communication  $G$ , ainsi qu'un paramètre  $k$  indiquant le nombre maximal de communications simultanées autorisées.  $G$  est obtenu à partir d'une matrice de communication  $M$  normalisée par la bande passante. La matrice  $M$  est normalisée une seconde fois par la latence  $\beta$ , puis transformée en graphe de communication  $G$  par la procédure explicitée au chapitre précédent. Cette première étape de normalisation de chaque poids par  $\beta$  va nous permettre de ne considérer par la suite que des valeurs de  $\beta$  égales à  $\frac{\beta}{\beta} = 1$ . Notons qu'il est facile de transformer toute solution du problème normalisé par  $\beta$  en une solution du problème initial en remultipliant chacun des poids des couplages solutions par  $\beta$ .

GGP se décompose en deux phases :

- une phase d'extension du graphe pour le rendre régulier sur les poids ;
- une phase d'épluchage du graphe en un ensemble de couplages parfaits.

Nous rappelons qu'un graphe  $G$  est régulier sur les poids si pour tout sommet  $s$  de  $G$ ,  $p(s) = a$  où  $a$  est une constante. Autrement dit la somme des poids des arêtes incidentes à un sommet est la même pour tous les sommets du graphe. Crescenzi et al. ont montré [26] que tout graphe régulier sur les poids admet un couplage parfait. Nous utilisons cette propriété afin de décomposer notre graphe régulier sur les poids en un ensemble de couplages parfaits : la propriété garantit qu'il existe un tel couplage. Il suffit de modifier ses arêtes de sorte qu'elles soient toutes de poids identiques et d'enlever ce couplage du graphe à décomposer. Le graphe restant étant de nouveau régulier sur les poids (car chaque sommet du graphe a vu son poids diminuer de la même quantité), il suffit pour le décomposer de boucler sur cette procédure jusqu'à ce qu'il ne contienne plus aucune arête.

La différence entre PBS et KPBS tient en une contrainte supplémentaire sur les couplages : quel que soit un couplage solution d'une instance de KPBS, le nombre d'arêtes qu'il contient ne peut pas être supérieur à  $k$ . Afin de prendre en compte cette contrainte, nous allons modifier la phase d'extension du graphe.

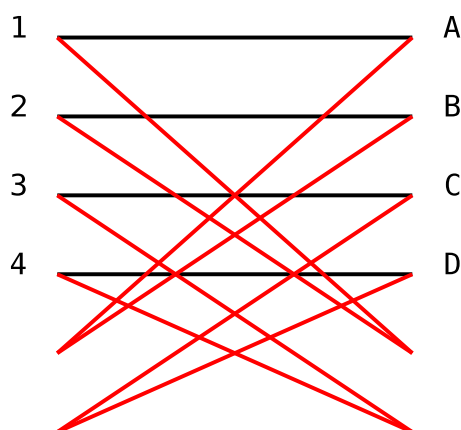
Considérons par exemple le graphe de communication de la figure 4.1, avec un paramètre  $k$  autorisant un maximum de deux arêtes par couplage. On considère ici chaque arête ayant un poids de 1.

Ce graphe étant à la fois régulier et régulier sur les poids, une application directe de l'algorithme résolvant PBS retourne simplement le graphe passé en entrée. En effet, celui-ci forme bien un couplage tout ce qu'il y a de plus valide en regard des contraintes imposées par PBS. En revanche, il contient 4 arêtes, soit 2 de plus que la limitée imposée pour cette instance de KPBS.

La solution à ce problème va consister à étendre le graphe initial par un ensemble d'arêtes judicieusement placées. Nous appellerons par la suite ces arêtes *arêtes virtuelles* par opposition aux *arêtes réelles* du graphe initial. De manière similaire les sommets ajoutés sont nommés sommets virtuels, par opposition aux sommets initiaux du graphe. La figure 4.2 montre le résultat



FIG. 4.1 – Graphe de communication régulier

FIG. 4.2 – Graphe étendu prenant en compte la contrainte sur  $k$ 

obtenu pour notre exemple. On peut voir de plus qu'il est impossible de choisir sur ce graphe un couplage parfait contenant plus de 2 arêtes réelles. Remarquons au passage que le graphe est désormais 2-régulier, ce qui signifie une décomposition en au plus 2 couplages parfaits.

### Algorithme d'extension

- L'algorithme d'extension du graphe initial a donc pour but de répondre à 2 objectifs distincts :
- d'une part, obtenir un graphe régulier sur les poids permettant une décomposition aisée en couplages parfaits ;
  - d'autre part, obtenir un graphe garantissant que tout couplage parfait contient au plus  $k$  arêtes réelles.

Notons qu'il est possible de découper n'importe quel graphe régulier sur les poids en un ensemble de couplages parfaits. Néanmoins, dans un soucis de minimiser les temps de communications il est nécessaire de minimiser également les quantités de données ajoutées lors de la phase d'extension. Nous cherchons donc ici à obtenir un graphe garantissant un nombre maximal de  $k$  arêtes réelles dans tout couplage parfait, mais également de poids des sommets minimal. Nous montrerons par la suite que l'extension réalisée garantie à la fois de bonnes performances et le respect de la contrainte posée par  $k$ .

Les 2 objectifs fixés étant relativement complexes à atteindre, nous nous restreignons, dans un premier temps, à une classe particulière de graphes initiaux. On suppose ici que le poids de

chaque arête du graphe initial est dans  $\mathbb{N}$ , tout comme  $\frac{P}{k}$ . De plus, on vérifie également  $\frac{P}{k} \geq W$ . On définit une fonction  $mw : V_1 \cup V_2 \rightarrow \mathbb{N}$  qui à un sommet  $s$  donné associe le poids manquant à ses arêtes pour que  $d(s) = \frac{P}{k}$ . L'algorithme va fonctionner en itérant sur chaque sommet du graphe initial et en ajoutant à chaque sommet  $s$  de nouvelles arêtes (virtuelles) dont la somme des poids vaudra  $mw(s)$ . Les arêtes virtuelles seront connectées à de nouveaux sommets virtuels. Notons également qu'aucune arête ne reliera deux sommets virtuels quelconques. Notons que le graphe final devant être régulier sur les poids, les sommets virtuels devront également avoir une somme des poids égale  $\frac{P}{k}$ . Enfin, on va chercher à minimiser le nombre d'arêtes virtuelles ajouter dans un soucis de décroître la complexité des algorithmes utilisés ultérieurement.

L'algorithme d'extension, décrit formellement figure 2 repose sur la construction séquentielle des sommets virtuels. On utilise pour ce faire une variable  $cn$  représentant le sommet virtuel en cours de construction. Le principe est le suivant : pour chaque sommet  $s$  réel, on calcule  $mw(s)$  et  $mw(cn)$  et on ajoute une arête entre  $s$  et  $cn$  de poids  $\max(mw(s), mw(cn))$  : c'est-à-dire une arête de poids le plus grand possible (dans un soucis de minimiser le nombre d'arêtes virtuelles) sans jamais dépasser pour chaque sommet la somme des poids recherchée. Après cet ajout, soit  $cn$  soit  $s$  a atteint son poids désiré (ou éventuellement même les deux) et l'on passe au sommet  $s$  suivant ou l'on crée un nouveau sommet virtuel  $cn$ .

Notons que deux hypothèses sont à vérifier sur le graphe  $I$  :

- $\frac{P(I)}{k} \in \mathbb{N}$  sinon  $\forall s \in V_{1_I} \cup V_{2_I}, mw(s) = \frac{P(I)}{k} - p(s) \notin \mathbb{N}$  et on ajoute donc des arêtes de poids non entier, ce que nous voulons éviter ;
- $\frac{P(I)}{k} \geq W(I)$  sinon  $\forall s \in V_{1_I} \cup V_{2_I}, mw(s) = \frac{P(I)}{k} - p(s) \leq \frac{P(I)}{k} - W(I) < 0$  et l'algorithme ne peut pas fonctionner.

**Proposition 7** *Le graphe  $J$  solution de l'algorithme 2 est  $\frac{P(I)}{k}$ -régulier sur les poids.*

**Preuve de la proposition 7** On doit prouver l'assertion suivante :  $\forall s \in V_{1_J} \cup V_{2_J}, p(s) = \frac{P(I)}{k}$ . On considère deux cas : le cas où  $s$  appartient à  $I$  et le cas où  $s$  est un sommet virtuel.

Si  $s$  appartient à  $I$ , l'algorithme ajoute des arêtes incidentes à  $s$  jusqu'à ce que  $mw(s) = 0$  et donc  $p(s) = \frac{P(I)}{k}$ .

Si  $s$  est un sommet virtuel : si l'on considère tous les sommets virtuels d'un côté du graphe biparti, la somme de leurs poids est égale par construction à la somme de tous les poids manquants ( $mw$ ) de tous les sommets réels de l'autre ensemble de sommets : respectivement  $\sum_{t \in V_{1_I}} mw(t)$  et  $\sum_{t \in V_{2_I}} mw(t)$ . On sait d'après la définition de  $mw$  que  $\sum_{t \in V_{1_I}} mw(t) = \sum_{t \in V_{1_I}} \frac{P(I)}{k} - p(t)$ . Ce qui est égal à  $|V_{1_I}| \times \frac{P(I)}{k} - P(I) = (|V_{1_I}| - k) \frac{P(I)}{k}$  puisqu'il n'existe aucune arête reliant deux sommets du même ensemble de sommets. Cette valeur divisée par  $\frac{P(I)}{k}$  donne une valeur entière :  $(|V_{1_I}| - k)$  ce qui signifie qu'il est possible d'ajouter des arêtes virtuelles à  $(|V_{1_I}| - k)$  sommets virtuels  $s$  tels qu'au final  $p(s) = \frac{P(I)}{k}$ . Le même raisonnement tient de manière symétrique pour  $V_{2_I}$ . Comme les sommets virtuels sont construits séquentiellement, en ne laissant jamais un sommet  $s$  avec  $p(s) < \frac{P(I)}{k}$  on a  $\forall s \in V_{1_J} \cup V_{2_J}, p(s) = \frac{P(I)}{k}$ . ■

Nous illustrons le déroulement de l'algorithme d'extension sur le graphe biparti de la figure 4.3. On considère ici une valeur de  $k = 2$ . Les poids de chaque arête sont entiers,  $W = 3 \leq \frac{P}{k} = \frac{2+1+2+1}{2} = 3$  il est donc possible d'utiliser l'algorithme d'extension pour obtenir un graphe 3-régulier sur les poids.

Tous les sommets de  $V_1$  ayant d'ors et déjà un poids de 3, on passe directement à l'extension des sommets de  $V_2$ . Les différentes étapes du déroulement de l'algorithme peuvent être suivies sur la figure 4.4. On choisit à la première étape le sommet  $A$ . Il manque à celui-ci un poids de

**Entrées** : un entier  $k$ , un graphe biparti  $I = (V_{1_I}, V_{2_I}, E_I, w_I)$  tel que  $\frac{P(I)}{k} \in \mathbb{N}$ ,  
 $W(I) \leq \frac{P(I)}{k}$  et  $w_I : E_I \rightarrow \mathbb{N}$ .

**Sorties** : un graphe biparti  $J$  tel que  $J$  est  $\frac{P(I)}{k}$  régulier sur les poids.  
 effectuer une copie de  $I$  en  $J : V_{1_I} = V_{1_J}, V_{2_I} = V_{2_J}, E_I = E_J$ , et  $\forall e \in E_J, w_J(e) = w_I(e)$ ;  
 on utilise une variable  $cn$  qui représente un sommet ;  
 $cn$  est initialement indéfinie ;  
**pour chaque** sommet  $s \in V_{1_I}$  ( $s$  également dans  $V_{1_J}$ ) **faire**  
   calculer  $mw(s)$  ;  
   **si**  $cn$  est indéfini ou  $mw(cn) = 0$  **alors**  
     ajouter un nouveau sommet  $n$  à  $V_{2_J}$  ;  
      $cn = n$  ;  
     ajouter une arête  $(s, cn)$  à  $E_J$  avec  $w_J(s, cn) = mw(s)$  ;  
   **fin**  
   **sinon**  
     **si**  $mw(cn) \geq mw(s)$  **alors** ajouter une arête  $(s, cn)$  à  $E_J$  avec  $w_J(s, cn) = mw(s)$  ;  
     **sinon**  
       ajouter une arête  $(s, cn)$  à  $E_J$  avec  $w_J(s, cn) = mw(cn)$  ;  
       ajouter un nouveau sommet  $n$  à  $V_{2_J}$  ;  
        $cn = n$  ;  
       ajouter une arête  $(s, cn)$  à  $E_J$  avec  $w_J(s, cn) = mw(s)$  ;  
     **fin**  
**fin**  
 effectuer les mêmes opérations pour les sommets de  $V_{2_J}$  ;

**Algorithme 2** : Algorithme d'extension en graphe régulier sur les poids

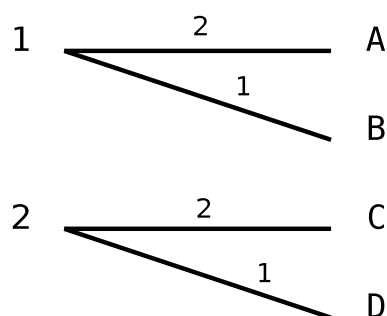


FIG. 4.3 – Graphe simple, avant extension

1. On crée donc le sommet virtuel 3 ainsi qu'une arête joignant 3 et A, de poids 1. On passe alors à un sommet suivant, ici le sommet C et on réitère l'opération en connectant une nouvelle arête de poids 1 au sommet 3. Notons au passage qu'aucune contrainte n'est posée sur l'ordre de traitement des différents sommets et que celui-ci peut donc être parfaitement aléatoire. On choisit le sommet B qui lui, nécessite un ajout d'un poids de 2. Néanmoins, le sommet 3 a déjà un poids de 2 et ne peut plus voir son poids augmenter que d'une seule unité. On crée donc une arête entre 3 et B avec un poids de 1 seulement. L'ajout suivant continue toujours avec le sommet B, et crée un nouveau sommet virtuel 4. Enfin, on termine en connectant le sommet D au sommet 4 avec une arête d'un poids de 2. Chaque sommet du graphe final a donc bien un poids de 3.

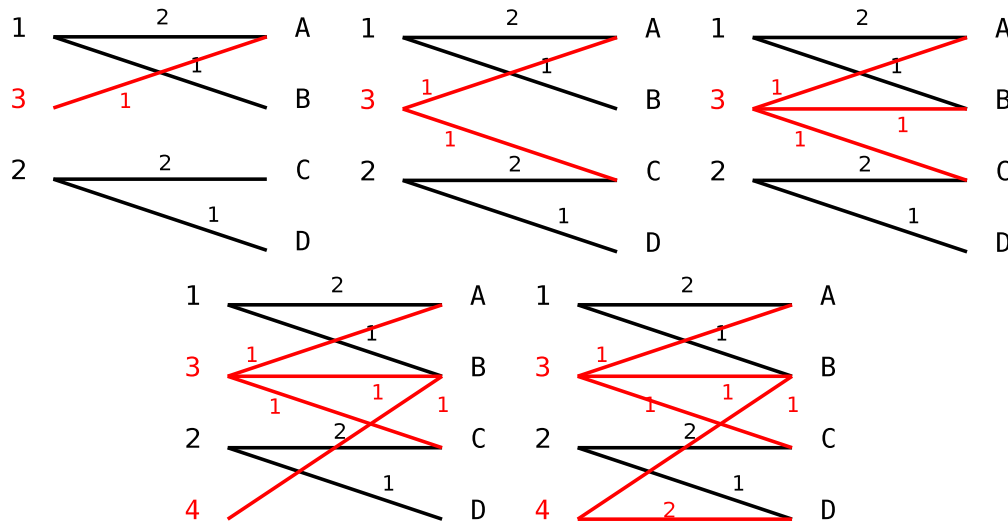


FIG. 4.4 – Déroulement de l'algorithme d'extension

### Initialisation

Avant de modifier le graphe  $G$  passé en entrée on vérifie que la valeur du paramètre  $k$  n'est pas inutilement trop élevée. Si  $k$  est supérieur à  $n(G)$ , le nombre de sommets de  $G$ , alors on autorise l'émission simultanée d'un plus grand nombre de communications qu'il n'est possible d'en effectuer en réalité. Afin de réduire au maximum la complexité de l'algorithme utilisé par la suite, on commence donc par affecter à  $k$  la valeur minimale entre  $k$  et  $n(G)$ . Cette opération n'affecte pas la validité des résultats obtenus.

L'algorithme d'extension ne fonctionnant que sur des graphes  $I$  vérifiant  $\frac{P(I)}{k} \in \mathbb{N}$  et  $\frac{P(I)}{k} \geq W(I)$ , il est nécessaire de commencer par modifier les graphes passés en entrée à GGP pour pouvoir les transformer en graphes réguliers sur les poids.

La première étape consiste à arrondir les poids de chaque arête à l'entier supérieur. Le but de cette manœuvre, outre d'obtenir des poids entiers, est d'équilibrer latence et préemption. En effet, l'utilisation de la préemption des communications permet de mieux répartir la bande passante entre les différents nœuds communicants. Malheureusement, préempter les communications a pour effet d'augmenter le nombre d'étapes de communication, ce qui augmente le temps total de transfert des données. Afin de limiter l'usage de la préemption aux cas où elle est le plus plausible d'être efficace, on utilise une règle simple : on ne découpe pas une communi-

cation en plusieurs étapes si le gain en temps que l'on pourrait espérer par cette décomposition est inférieur à  $\beta$ . En l'occurrence on ne découpe pas des arêtes de poids inférieur à 1 : en effet, les poids des arêtes du graphe d'entrée ont été normalisés par  $\beta$ . En faisant en sorte que le reste de l'algorithme ne manipule que des arêtes de poids entiers on limite donc tout usage de la préemption de la manière voulue. Notons au passage que ceci n'est qu'une heuristique et en garanti en aucun cas un choix optimal. Néanmoins, nous prouverons section 4.1.3 que ce choix permet d'obtenir de bons résultats.

Les figures 4.6 et 4.7 représentent ainsi l'exécution de la procédure d'arrondi sur le graphe de la figure 4.5, ainsi que les résultats fournis par GGP pour deux valeurs différentes de  $\beta$  (respectivement  $\beta = 2$  et  $\beta = 1$ ). On voit bien ici que dans le cas où  $\beta = 1$  la décomposition comporte 3 étapes, contre 2 pour  $\beta = 2$ . Le temps pris pour les communications (sans les latences) est lui, inversement plus faible alors que le nombre d'étapes augmente.

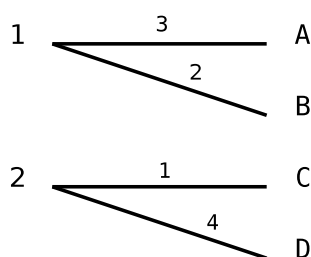


FIG. 4.5 – Graphe d'entrée illustrant la nécessité d'équilibrer latence et préemption

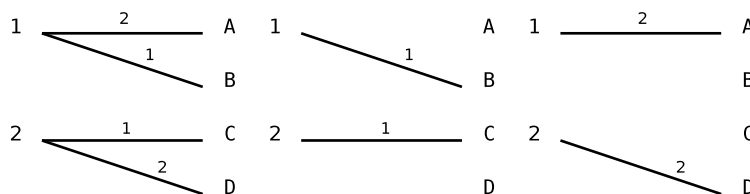


FIG. 4.6 – Arrondi et décomposition pour  $\beta = 2$

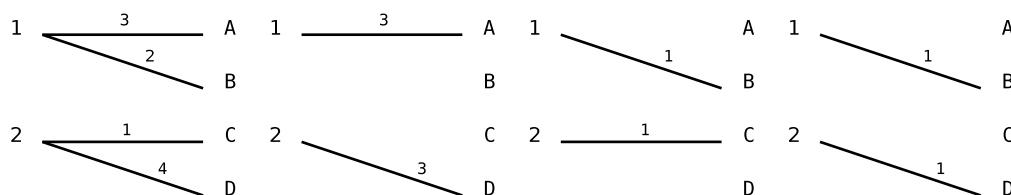


FIG. 4.7 – Arrondi et décomposition pour  $\beta = 1$

Le graphe disposant de poids entiers, il reste néanmoins pour pouvoir utiliser l'algorithme d'extension à valider les propriétés que  $\frac{P}{k} \geq W$  et  $\frac{P}{k} \in \mathbb{N}$ . Ceci se fait simplement en rajoutant des arêtes virtuelles reliant des sommets virtuels, jusqu'à ce que  $\frac{P}{k}$  soit entier et supérieur à  $W$ . Encore une fois, dans un soucis de minimiser le nombre d'arêtes ainsi rajoutées, on rajoute

des arêtes de poids le plus élevé possible. Comme l'on va chercher à ne pas augmenter la valeur de  $W$  pour le graphe (car elle rentre en jeu dans le calcul des bornes inférieures de KPBS), on rajoute ici des arêtes de poids  $W$  (sauf éventuellement la dernière arête rajoutée). Par exemple, le graphe de la figure 4.8 est étendu avec deux arêtes virtuelles de poids respectifs 4 et 2 ce qui amène à une valeur de  $\frac{P}{k}$  de  $\frac{4+1+1+4+2}{3} = 4 = W$ .

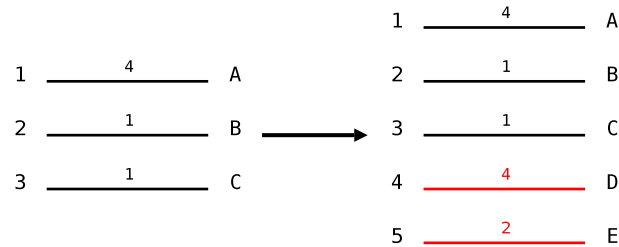


FIG. 4.8 – Extension d'un graphe tel que  $\frac{P}{k} \geq W$

### Décomposition du graphe

Une fois le graphe régulier sur les poids construit, il reste alors à la décomposer en une série de couplages. On utilise pour cela l'algorithme de Crescenzi et al. [26] :

1. choisir un couplage parfait  $M$  qui est garanti d'exister car le graphe est régulier sur les poids ;
2. calculer  $q$ , le poids le plus faible de chaque arête de  $M$  ;
3. extraire du graphe un couplage valué composé de toutes les arêtes de  $M$ , mais chaque arête ayant un poids de  $q$  ;
4. le graphe restant étant de nouveau régulier sur le poids, recommencer la procédure à l'étape 1. jusqu'à ce que le graphe soit vide.

Nous prouvons section 4.1.3 que grâce à la phase d'extension, tout couplage obtenu contient au plus  $k$  arêtes réelles.

L'ensemble de couplages ainsi obtenu ne peut néanmoins être utilisé tel quel, en effet, il est nécessaire d'éliminer toutes les arêtes virtuelles rajoutées au cours des différentes phases d'extension. De plus, la première étape d'arrondi des poids a modifié les poids des arêtes et il est donc nécessaire de les modifier une seconde fois ici.

Par la suite nous nommerons *solution intermédiaire* la solution obtenue une fois les arêtes virtuelles éliminées et avant la modification des poids et *solution finale* la solution retournée par GGP une fois les poids des couplages modifiés.

Pour effectuer cette dernière modification, on itère sur le graphe initial en extrayant chaque couplage obtenu, mais sans jamais enlever plus de poids que disponible. Les couplages obtenus en réalisant cette décomposition forment alors la solution recherchée. Notons au passage que si les poids de  $G$  sont tous entiers, alors la décomposition obtenue est directement valide.

Par exemple, un graphe d'une seule arête, de poids 0,6 avec une valeur de  $k = 1$  va être étendu par GGP en un graphe d'une arête de poids  $[0, 6] = 1$  qui sera décomposé en un seul couplage, d'une arête de poids 1. On enlève ce couplage au graphe initial : l'arête du graphe initial ne disposant que d'un poids de 0,6 et le couplage à enlever d'un poids de 1, on ne retire ici du graphe qu'un couplage formé d'une arête avec un poids de 0,6 qui forme la solution recherchée.

### 4.1.2 Algorithme

En remettant les différents morceaux de l'algorithme dans l'ordre, on obtient la procédure suivante :

1. normaliser les poids par  $\beta$  et arrondir à l'entier supérieur ;
2. étendre le graphe afin que  $\frac{P}{k} \geq W$  et  $\frac{P}{k} \in \mathbb{N}$  ;
3. étendre le graphe avec l'algorithme d'extension ;
4. décomposer le graphe en une série de couplages ;
5. éliminer les arêtes virtuelles afin d'obtenir la solution intermédiaire ;
6. recalculer les poids des arêtes de la solution intermédiaire pour éliminer l'augmentation de poids liée à l'arrondi lors de la construction du graphe  $H$  afin d'obtenir la solution finale.

La figure 3 décrit l'algorithme GGP de façon formelle. Une telle description va en particulier nous être utile pour l'étude de la complexité et de la qualité de GGP.

#### Exemple de déroulement de l'algorithme

On considère le graphe de communication de la figure 4.10, avec un paramètre  $k$  d'une valeur de 3. Celui-ci est obtenu à partir de la matrice de communication de la figure 4.9 avec une valeur de  $\beta$  de 2. Remarquons au passage que certains poids ne sont pas entiers, et qu'ils ont été normalisés par  $\beta$ .

nœuds	A	B	C
1	4	0	0
2	0	1,5	0,8
3	0	2	0

FIG. 4.9 – Matrice de communication normalisée initiale

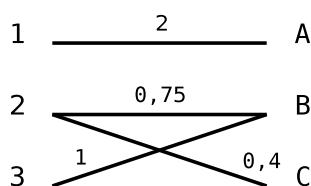


FIG. 4.10 – Graphe  $G$  initial, obtenu à partir de la matrice de communication 4.9 pour  $\beta = 2$

Lors de la première étape de GGP on normalise tous les poids par  $\beta$  puis les arrondit à l'entier supérieur, comme le montre le graphe  $H$  obtenu figure 4.11.

On calcule alors  $W(H) = 2$  et  $\frac{P(H)}{k} = \frac{2+1+1+1}{2} = 2,5$ .  $\frac{P(H)}{k}$  étant supérieur à  $W(H)$  mais non entier, on ajoute une arête virtuelle reliant deux nouveaux sommets virtuels 4 et  $D$  de poids 1 pour obtenir le graphe  $I$ . On a bien désormais  $\frac{P(I)}{k} = \frac{2+1+1+1+1}{2} = 3 = \left\lceil \frac{P(H)}{k} \right\rceil$ .

Le graphe  $I$  est alors étendu en un graphe  $J$  qui soit  $\frac{P(I)}{k}$ -régulier sur les poids. On procède en itérant sur les sommets de  $V_{1_I}$  et  $V_{2_I}$  en ajoutant une arête de poids 1 à chaque sommet de poids inférieur à 2. La figure 4.12 illustre ainsi les graphes  $I$  et  $J$  obtenus.

```

Entrées : un graphe biparti  $G = (V_1, V_2, E, w_G)$ , un entier  $k$ .
Sorties : un ensemble  $S$  de couplages pondérés.
1 construire un graphe  $H = (V_1, V_2, E, w_H)$  tel que  $\forall e \in E, w_H(e) = \lceil w_G(e) \rceil$ ;
 $k = \min(k, n(G))$ ;
/* construire un graphe  $I$  tel que  $\frac{P(I)}{k} \geq W(I)$  et  $\frac{P(I)}{k} \in \mathbb{N}$  */
2 construire  $I = (V_{1I}, V_{2I}, E_I, w_I)$  de la manière suivante :
  si  $\frac{P(H)}{k} < W(H)$  alors  $p = kW(H) - P(H)$ ;
  sinon  $p = k \lceil \frac{P(H)}{k} \rceil - P(H)$ ;
/*  $p$  a comme valeur le total des poids des arêtes à rajouter */
/* le nombre d'arêtes à rajouter est alors  $n = \lceil \frac{p}{W(H)} \rceil$  */
/* on construit donc : */
 $V_{1I} = \{s_1, \dots, s_{|V_1|}, s'_1, \dots, s'_{n-1}, s'_n\}$ ;
 $V_{2I} = \{r_1, \dots, r_{|V_2|}, r'_1, \dots, r'_{n-1}, r'_n\}$ ;
 $E_I = \{e_1, \dots, e_m, e'_1, \dots, e'_{n-1}, e'_n\}$  avec
 $\{s_1, \dots, s_{|V_1|}\} = V_1, \{r_1, \dots, r_{|V_2|}\} = V_2, \{e_1, \dots, e_m\} = E, \forall e \in E, w_i(e) = w(e)$ ;
pour  $i \in \{1, \dots, n-1\}$  faire
  |  $(s'_i, r'_i) = e'_i \in E_I$ , et,  $w_I(e'_i) = W(H)$ 
fin
 $(s'_n, r'_n) = e'_n \in E_I$ , et  $w_I(e'_n) \leq W(H)$ ;
 $\sum_{i=1}^n w_I(e'_i) = p$ ;
/* transformer  $I$  en un graphe régulier sur les poids */
3 construire un graphe  $J = (V_{1J}, V_{2J}, E_J, w_J)$ , qui soit  $\frac{P(I)}{k}$ -régulier sur les poids en
  utilisant l'algorithme de la figure 2;
4  $R = \emptyset$ 
5 tant que  $E_J \neq \emptyset$  faire
  | choisir un couplage parfait  $M$  dans  $J$ ;
  | changer  $w_M$  de sorte que  $\forall e \in M, w_M(e) = s(M)$  le plus petit poids de toutes les
  | arêtes de  $M$ ;
  | ajouter  $M$  à  $R$ , l'ensemble des couplages formant la solution intermédiaire;
  |  $\forall e \in M$  changer  $w_J(e)$  en  $w_J(e) - w_M(e)$ ;
  | enlever de  $E_J$  toutes les arêtes de poids 0;
fin
6 enlever de  $R$  toute arête  $e$  telle que  $e \notin E$ ;
7  $R$  étant composé des couplages  $M_1, \dots, M_h$ , pour  $i \in \{1, \dots, h\}$  faire
  |  $M'_i = \emptyset$ ;
  | pour chaque arête  $e$  de  $M_i$  faire
  | | ajouter  $e$  dans  $M'_i$  avec  $w_{M'_i}(e) = \min(w_i(e), w_G(e))$ ;
  | |  $w_G(e) = \max(w_G(e) - w_{M'_i}(e), 0)$ ;
  | fin
  | ajouter  $M'_i$  dans  $S$ ;
fin

```

Algorithme 3 : Algorithme GGP

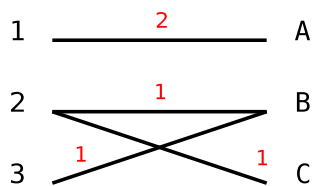


FIG. 4.11 – Graphe  $H$ , après l'arrondi des poids des arêtes de  $G$

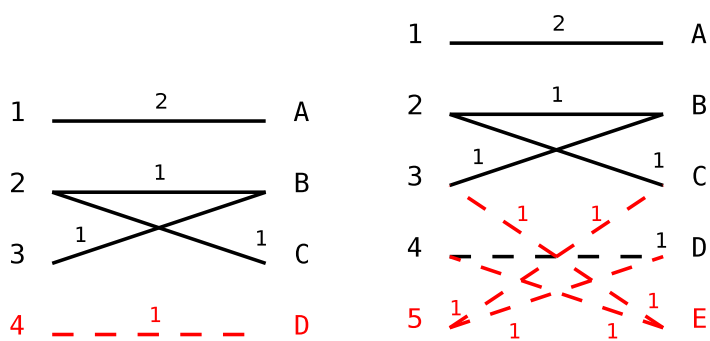


FIG. 4.12 – Graphes  $I$  et  $J$ , obtenus par ajouts d'arêtes virtuelles

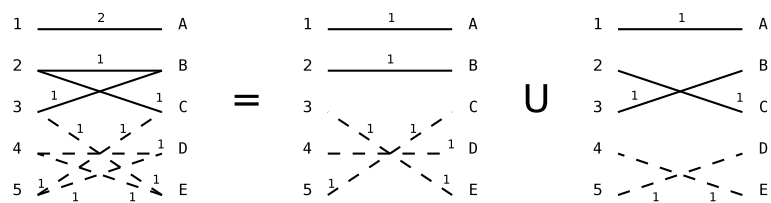


FIG. 4.13 – Décomposition de  $J$  en deux couplages

Le graphe  $J$  que l'on obtient est 2-régulier sur les poids. On le décompose simplement en deux couplages parfaits comme l'illustre la figure 4.13.

Enfin, la solution finale est obtenue en éliminant toutes les arêtes virtuelles, et en diminuant les poids des arêtes afin d'avoir une décomposition du graphe  $G$  initial. La figure 4.14 contient la décomposition finale du graphe en deux couplages.

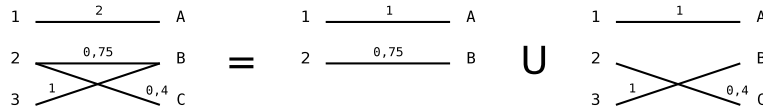


FIG. 4.14 – Solution finale

### 4.1.3 Propriétés

L'étude des différentes propriétés de l'algorithme GGP que nous proposons va se structurer en trois parties distinctes : dans un premier temps, une étude de la complexité au pire cas de GGP, puis une étude sur la qualité des solutions obtenues, où nous prouvons que toute décomposition fournie par GGP ne peut avoir un coût plus élevé que  $\frac{8}{3}$  fois la borne optimale  $\eta$  introduite section 3.2.3 c'est-à-dire que GGP est un algorithme d'approximation. Nous prouvons enfin que toute solution obtenue par GGP valide bien la contrainte posée par  $k$  et nous terminons par une mise en avant de différents cas problématiques.

#### Complexité

**Proposition 8** *GGP a une complexité au pire cas en  $O(\sqrt{n(G)}(m(G) + n(G))^2)$*

Les étapes 1,2,3,4 et 6 de GGP sont effectuées en temps linéaire. La complexité au pire cas de l'algorithme est donc donnée par la complexité de l'étape 5, plus élevée. En effet l'étape 5 requiert de trouver un couplage parfait, ce qui est d'une complexité de  $O(\sqrt{n(J)}m(J))$  en utilisant la méthode hongroise [51]. Comme à chaque itération au moins une arête est enlevée du graphe (l'arête du couplage obtenu de poids minimal), le nombre total d'itérations sur l'étape 5 est au pire cas de  $m(J)$ . L'étape 5 prend donc dans le pire des cas  $O(\sqrt{n(J)}m(J)^2)$  opérations.

On cherche maintenant à exprimer  $n(J)$  et  $m(J)$  en fonction de  $n(G)$  et  $m(G)$ . Lors de la construction du graphe  $H$  aucun sommet et aucune arête ne sont ajoutés et donc  $n(H) = n(G)$  et  $m(H) = m(G)$ . Pour construire  $I$  il est nécessaire d'ajouter au plus  $k$  arêtes et donc  $2k$  sommets. On a donc  $n(I) \leq n(G) + 2k$  et  $m(I) \leq m(G) + k$ . Comme  $k$  est inférieur à  $n(G)$  on en déduit que  $n(I) \leq 3n(G)$  et  $m(I) \leq m(G) + n(G)$ .

Lors de l'étape d'extension en graphe régulier sur les poids, pour chaque sommet de  $I$  (chaque itération) on rajoute au plus un nouveau sommet à  $J$ . On a donc  $n(J) \leq 2n(I)$ . De manière similaire, pour chaque sommet du graphe  $I$  on rajoute au plus 2 nouvelles arêtes à  $J$  et donc  $m(J) \leq m(I) + 2n(I)$ . On en tire,  $n(J) \leq 6n(G)$  et  $m(J) \leq m(G) + 7n(G)$ .

La complexité au pire cas de l'algorithme GGP est donc en  $O(\sqrt{n(G)}(m(G) + n(G))^2)$ . ■

#### Ratio d'approximation

Nous prouvons ici que toute solution de GGP a au pire un coût  $\frac{8}{3}$  fois supérieur à la borne du coût de toute redistribution  $\eta$  étudiée au chapitre précédent. La preuve nécessite d'introduire un

nouvel algorithme d'approximation appelé algorithme par multigraphe fonctionnant de manière similaire à GGP et disposant du même ratio d'approximation. Nous prouvons que l'algorithme par multigraphe est un algorithme d'approximation d'un ratio d'approximation de  $\frac{8}{3}$  et que pour tout jeu de données en entrée, GGP fournit une décomposition de coût inférieur.

On définit la fonction *trans* qui prend en argument un graphe biparti valué  $G = (V_1, V_2, E, w_G)$  et retourne le multigraphe  $G' = (V'_1, V'_2, E')$  obtenu en découpant toute arête tel que toute arête  $e \in E$  de poids  $w_G(e)$  soit transformée en  $w_G(e)$  arêtes de poids 1.

Nous utilisons cette fonction pour définir l'algorithme par multigraphe (figure 4). L'algorithme commence par étendre le graphe  $G$  fourni en entrée en un graphe  $J$  de manière similaire à GGP mais diffère sur la manière d'éplucher le graphe : On construit ici  $J' = \text{trans}(J)$  qui est un graphe régulier que l'on épluche en un ensemble de couplages parfaits (où chaque arête à un poids 1) en utilisant la propriété qu'il existe toujours un couplage parfait sur un multigraphe régulier. Le coût de chacun des couplages ainsi obtenus est donc de 1. On peut noter que le nombre d'arêtes de  $J'$  dépend des poids des arêtes de  $J$  et donc cet algorithme n'est que pseudo-polynomial et donc inutilisable en pratique. Cohen et al. ont proposé [25] une version modifiée de cet algorithme le rendant polynomial. Néanmoins sa complexité reste trop élevée pour une utilisation réelle.

**Entrées** : un graphe biparti  $G = (V_1, V_2, E, w_G)$ , un entier  $k$ .

**Sorties** : un ensemble de couplages  $S'$ .

- 1 construire le graphe  $J$  comme décrit dans les étapes 1, 2 et 3 de GGP;
  - 2 construire  $J' = \text{trans}(J)$ ;
  - 3  $S' = \emptyset$ ;
  - 4 **tant que**  $E_{J'} \neq \emptyset$  **faire**
    - choisir un couplage parfait  $M'$  dans  $J'$ ;
    - ajouter  $M'$  à  $S'$ , l'ensemble de couplages cherché;
    - enlever de  $E_{J'}$  toutes les arêtes de  $M'$ ;
- fin**

**Algorithme 4** : Algorithme par multigraphe

Afin de prouver le ratio d'approximation de  $\frac{8}{3}$  de cet algorithme, on commence par prouver que la borne inférieure sur le temps de transmissions des données du graphe  $I$  est égale à  $\max\left(\left\lceil \frac{P(H)}{k} \right\rceil, W(H)\right)$

**Lemme 1**  $\eta_d(I) = \max\left(\left\lceil \frac{P(H)}{k} \right\rceil, W(H)\right)$

**Preuve** : Considérons les trois cas possibles lors de la construction de  $I$  :

1.  $\frac{P(H)}{k} \geq W(H)$  et  $\frac{P(H)}{k} \in \mathbb{N}$  :  $I = H$  et donc

$$\begin{aligned} \eta_d(I) &= \max\left(\frac{P(I)}{k}, W(I)\right) \\ &= \max\left(\frac{P(H)}{k}, W(H)\right) \\ &= \max\left(\left\lceil \frac{P(H)}{k} \right\rceil, W(H)\right) \end{aligned}$$

2.  $\frac{P(H)}{k} \geq W(H)$  et  $\frac{P(H)}{k} \notin \mathbb{N}$  : on n'ajoute aucune arête de poids supérieur à  $W(H)$  et  $\frac{P(I)}{k} = \left\lceil \frac{P(H)}{k} \right\rceil$  d'où  $\eta_d(I) = \max\left(\left\lceil \frac{P(H)}{k} \right\rceil, W(H)\right)$ .
3.  $\frac{P(H)}{k} < W(H)$  : comme on n'ajoute aucune arête de poids supérieur à  $W(H)$ ,  $W(I) = W(H)$ . De plus on ajoute des arêtes jusqu'à ce que  $\frac{P(I)}{k} = W(H)$  d'où  $\eta_d(I) = \eta_d(H) = W(H)$ .

■

**Theorème 2** *L'algorithme par multigraphe est un algorithme d'approximation d'un ratio d'approximation de  $\frac{8}{3}$ .*

**Preuve :** On nomme  $I'$  le graphe  $\text{trans}(I)$ .

Avec  $G = (V_1, V_2, E, w_G)$  un graphe biparti valué, et  $k$  un entier, nos paramètres d'entrée : On applique l'algorithme par multigraphe sur  $G$  pour obtenir  $S'$  l'ensemble des couplages recherché. Comme  $J$  est, par construction, un graphe  $\frac{P(I)}{k}$ -régulier sur les poids et que tous les poids sont entiers, on sait que  $J' = \text{trans}(J)$  est un multigraphe  $\frac{P(I)}{k}$ -régulier. Comme  $P(I) = m(I')$  on a  $J'$  un graphe  $\frac{m(I')}{k}$ -régulier. Comme à chaque étape de l'itération, on enlève un couplage parfait de  $J'$  et donc une arête sur chaque sommet on sait que  $|S'| = \frac{m(I')}{k} = \eta_s(I')$ . Le coût de la solution  $S'$  est donc  $c(S') = \eta_s(I') + \eta_s(I') \times 1 = 2\eta_s(I')$  car chaque étape à une durée de 1, et un coût d'initialisation de 1. Comme  $\frac{P(I)}{k} = \frac{m(I')}{k}$  et  $W(I) = \Delta(I')$  on sait que  $\eta_s(I') = \max\left(\Delta(I'), \frac{m(I')}{k}\right) = \max\left(W(I), \frac{P(I)}{k}\right) = \eta_d(I)$ . Par conséquent  $c(S') = 2\eta_d(I)$ . On utilise alors le lemme 1 pour en déduire la formule de l'équation 4.1.

$$c(S') = 2\max\left(\left\lceil \frac{P(H)}{k} \right\rceil, W(H)\right) \quad (4.1)$$

Supposons dans un premier temps

$$\left\lceil \frac{P(H)}{k} \right\rceil > W(H) \quad (4.2)$$

L'équation 4.1 devient alors :  $c(S') = 2\left(\left\lceil \frac{P(H)}{k} \right\rceil\right)$ .

D'après l'algorithme de construction de  $H$ , aucune arête ne voit son poids augmenter de plus d'une unité. Par conséquent on a :  $P(H) \leq P(G) + m(G)$  et  $W(H) \leq W(G) + \Delta(G)$ . On peut ainsi majorer  $c(S')$  :

$$\begin{aligned} c(S') &\leq 2\left(\left\lceil \frac{P(G) + m(G)}{k} \right\rceil\right) \\ &\Rightarrow c(S') \leq 2\left(\left\lceil \frac{P(G)}{k} \right\rceil + \left\lceil \frac{m(G)}{k} \right\rceil\right) \\ &\Rightarrow c(S') \leq 2\left(\left\lceil \frac{P(G)}{k} \right\rceil + \max\left(\left\lceil \frac{m(G)}{k} \right\rceil, \Delta(G)\right)\right) \\ &\Rightarrow c(S') \leq 2\left(\frac{P(G)}{k} + 1 + \eta_s(G)\right) \\ &\Rightarrow c(S') \leq 2(\eta_d(G) + 1 + \eta_s(G)) \end{aligned}$$

On peut donc en tirer un ratio d'approximation de :

$$\frac{c(S')}{\eta(G)} \leq \frac{2(\eta_d(G) + 1 + \eta_s(G))}{\eta(G)} = 2 + \frac{2}{\eta(G)} \quad (4.3)$$

Comme  $W(H)$  est un nombre entier, et que nous avons supposé par l'équation 4.2 que  $\left\lceil \frac{P(H)}{k} \right\rceil > W(H)$  on peut en déduire  $\frac{P(H)}{k} > W(H)$ . Ceci signifie que  $m(H) > k$  car si le nombre d'arêtes du graphe  $H$  était inférieur à  $k$ , alors  $P(H)$  serait d'une valeur inférieure à  $k \times W(H)$  par la définition de  $W(H)$  et par conséquent  $\frac{P(H)}{k}$  serait inférieur à  $W(H)$ .  $k$  étant au moins égal à 1, on en déduit donc  $m(H) \geq 2$ .

Par construction  $m(H) = m(G)$  et donc  $\left\lceil \frac{m(G)}{k} \right\rceil \geq 2$ . On en déduit  $\eta_s(G) \geq 2$ .

Si on suppose de plus que le poids d'au moins une arête de  $G$  est supérieur ou égal à 1, on a  $W(G) \geq 1$  et donc  $\eta_d(G) = \max\left(\frac{P(G)}{k}, W(G)\right) \geq 1$ .

et par conséquent, dans ce cas,  $\eta(G) \geq 2 + 1 = 3$ .

On peut alors majorer le ratio d'approximation de l'équation 4.3 :

$$\frac{c(S')}{\eta(G)} \leq 2 + \frac{2}{\eta(G)} \leq 2 + \frac{2}{3} = \frac{8}{3} \quad (4.4)$$

Considérons maintenant l'hypothèse opposée, où toutes les arêtes de  $G$  ont un poids inférieur à 1. On sait alors que toutes les arêtes de  $H$  ont un poids de 1 et donc  $W(H) = 1$ . Par conséquent les arêtes ajoutées à  $H$  pour construire  $I$  ont également un poids de 1. On a donc :  $\eta_s(I') = \eta_s(I)$  car  $I$  et  $I'$  sont alors deux graphes identiques. Comme le poids de chaque arête de  $I$  vaut 1, on a  $P(I) = m(I)$  et  $W(I) = \Delta(I)$ . On a donc :  $\eta_s(I) = \max\left(\left\lceil \frac{P(I)}{k} \right\rceil, W(I)\right) = \max\left(\left\lceil \frac{P(H)}{k} \right\rceil, W(H)\right)$  d'après la construction de  $I$ . Comme le poids de chaque arête de  $H$  vaut également 1, on en conclut que  $\eta_s(I) = \eta_s(H)$ . Finalement comme  $m(H) = m(G)$  et  $\Delta(H) = \Delta(G)$  on a :  $\eta_s(I') = \eta_s(I) = \eta_s(H) = \eta_s(G)$  et donc  $c(S') \leq 2\eta_s(G) \leq 2\eta(G)$ . Ce qui nous donne pour ce cas un ratio d'approximation de 2 :

$$\frac{c(S')}{\eta(G)} \leq \frac{2\eta(G)}{\eta(G)} = 2 \quad (4.5)$$

Il nous reste maintenant à étudier le cas où  $\left\lceil \frac{P(H)}{k} \right\rceil \leq W(H)$ .

L'équation 4.1 devient alors :

$$c(S') = 2W(H) \quad (4.6)$$

D'après la construction de  $H$ , on sait que  $W(H) \leq W(G) + \Delta(G)$ . On en tire donc :

$$\begin{aligned} c(S') &\leq 2(W(G) + \Delta(G)) \\ &\leq 2\left(\max\left(\frac{P(G)}{k}, W(G)\right) + \max\left(\left\lceil \frac{m(G)}{k} \right\rceil, \Delta(G)\right)\right) \\ &\leq 2(\eta_d(G) + \eta_s(G)) = 2\eta(G) \end{aligned}$$

Ce qui nous amène à un ratio d'approximation de :

$$\frac{2\eta(G)}{\eta(G)} = 2 \quad (4.7)$$

Par conséquent les équations 4.3 4.5 4.7 déterminent que l'algorithme par multigraphe a un ratio d'approximation d'au plus  $\max(\frac{8}{3}, 2, 2) = \frac{8}{3}$  ■

Notons au passage que ce ratio ne concerne qu'un nombre réduit de cas : le ratio de  $2 + \frac{2}{\eta}$  tend en effet rapidement vers 2 lorsque les temps de redistribution augmentent et de plus, même pour des valeurs basse de  $\eta$ , seules certaines sous-catégories de graphes peuvent espérer atteindre ce ratio.

**Theorème 3** *GGP est un algorithme d'approximation d'un facteur d'approximation de  $\frac{8}{3}$ .*

**Preuve :** Pour prouver le théorème 3, on montre que pour tout ordonnancement  $S$  obtenu par GGP, de coût  $c(S)$  à partir d'une solution intermédiaire  $R$  de coût  $c(R)$  il existe un ordonnancement  $S'$  obtenu par l'algorithme par multigraphe, de coût  $c(S')$  tel que  $c(S) \leq c(S')$ .

On sait par construction que  $c(S) \leq c(R)$ ,  $S$  étant obtenue en diminuant les poids des arêtes de  $R$ . Il nous suffit par conséquent de montrer que  $c(R) \leq c(S')$ .

Par construction, une solution intermédiaire  $R$ , obtenue par GGP, peut être décomposée en une solution  $S'$  telle que  $\forall M_i \in R$  de coût  $c(M_i)$  il existe  $c(M_i)$  couplages identiques  $M'_j$  de coût 1 appartenant à  $S'$ . On a donc  $\sum_{i=1}^{|R|} c(M_i) = \sum_{j=1}^{|S'|} c(M'_j)$  et  $|R| \leq |S'|$ .

Le coût de  $R$  est :  $|R| \times 1 + \sum_{i=1}^{|R|} c(M_i)$ . De manière similaire, le coût de  $S'$  est :  $|S'| \times 1 + \sum_{j=1}^{|S'|} c(M'_j)$ . Ainsi  $c(R) \leq c(S')$  et donc  $c(S) \leq c(S')$ . ■

## Validité

Pour obtenir un ordonnancement valide, il est nécessaire de vérifier que la contrainte du modèle 1-port ainsi que la contrainte des  $k$  communications simultanées sont vérifiées. La contrainte du modèle 1-port est validée automatiquement par GGP, la solution étant constituée d'un ensemble de couplages. Nous prouvons dans cette section par la proposition 9 le respect de la contrainte sur  $k$ . Nous avons pour cela besoin du lemme suivant :

**Lemme 2** *Tout couplage parfait du graphe  $J$  contient  $k$  arêtes du graphe  $I$*

**Preuve :** Soit  $M$  un couplage parfait sur  $J$ . On sait d'après la preuve de la proposition 7 que  $V_{1_J}$  contient tous les sommets de  $V_{1_I}$ , ainsi que  $|V_{2_I}| - k$  nouveaux sommets. De manière similaire,  $V_{2_J}$  est formé de  $V_{2_I}$  augmenté de  $|V_{1_I}| - k$  nouveaux sommets. Nous avons donc  $|V_{1_J}| = |V_{2_J}| = |V_{1_I}| + |V_{2_I}| - k$ , ce qui est également le nombre d'arêtes de  $M$ . On sait de plus qu'aucune arête ne connecte les sommets ajoutés pour construire  $J$  ainsi que toute arête incidente à un d'entre eux n'appartient pas à  $E_I$ . Par conséquent, chacun de ces sommets est incident à une arête de  $M$  qui n'est donc pas dans  $E_I$ . Comme nous avons ajouté  $|V_{1_I}| - k$  et  $|V_{2_I}| - k$  sommets, le nombre d'arêtes de  $M$  appartenant également à  $E_I$  est de  $|V_{1_I}| + |V_{2_I}| - k - (|V_{1_I}| - k) - (|V_{2_I}| - k) = k$ . ■

**Proposition 9**  $\forall M \in S$ , un couplage de la solution  $S$  donnée par GGP,  $|M| \leq k$ .

**Preuve :** On sait d'après le lemme 2 que tout couplage parfait  $M$  de  $J$  contient  $k$  arêtes appartenant à  $I$ . Comme  $I$  est obtenu en ajoutant des arêtes à  $H$ ,  $M$  contient au plus  $k$  arêtes appartenant à  $G$ . Comme  $S$  est formé d'un ensemble de couplages parfaits sur  $J$  auxquels

sont enlevées toutes les arêtes n'appartenant pas à  $G$ , nous pouvons conclure directement que  $|M| \leq k$ . ■

### Cas problématiques

Disposant d'une borne supérieure sur le rapport temps obtenu / temps optimal pour les ordonnancements réalisés avec GGP, nous avons naturellement cherché à obtenir une borne inférieure sur le ratio d'approximation de GGP, c'est-à-dire trouver un ensemble de graphes particuliers pour lesquels les résultats fournis par l'algorithme font preuve de mauvaises performances. En effet, nous avons prouvé qu'il n'est pas possible d'obtenir avec GGP un ordonnancement fournissant des temps d'exécution plus grands que  $\frac{8}{3}$  fois une borne inférieure sur le temps de redistribution. Rien à ce stade ne nous indique que  $\frac{8}{3}$  est le meilleur ratio d'approximation que l'on puisse prouver pour GGP. Le meilleur moyen de montrer qu'il est impossible d'améliorer le ratio d'approximation en dessous d'une valeur de  $b$  consiste tout simplement à trouver un exemple de graphe sur lequel le temps de redistribution obtenu à l'aide de GGP a une valeur de  $b$  fois la borne inférieure.

Nous montrons tout d'abord que le ratio d'approximation de  $\frac{8}{3}$  ne peut pas être amélioré sans prendre en considération la dernière étape de GGP diminuant les poids des arêtes des couplages obtenus.

Pour ce faire, nous présentons ici une famille d'exemple pour lesquelles le ratio temps d'exécution d'une solution intermédiaire obtenue par GGP divisé par la borne inférieure tend vers le ratio d'approximation de  $\frac{8}{3}$ .

Considérons le graphe de la figure 4.15, associé à une valeur de  $k$  égale à 3.

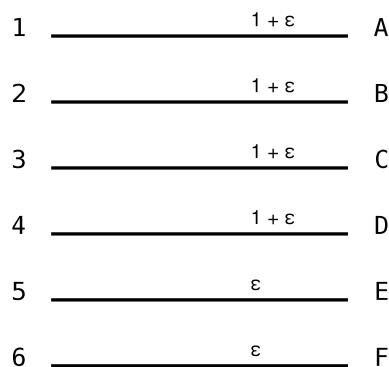


FIG. 4.15 – Graphe problématique pour GGP ( $k = 3$ )

Pour ce graphe, la borne inférieure a comme valeur :  $\eta = \eta_d + \eta_s = \max\left(\frac{4+6\epsilon}{3}, 1 + \epsilon\right) + \max\left(\lceil \frac{6}{3} \rceil, 1\right) = \frac{4+6\epsilon}{3} + 2 \approx \frac{10}{3}$ .

L'exécution de GGP n'étant pas déterministe (à cause de l'étape de choix d'un couplage), il est ici possible d'obtenir différentes solutions. Nous supposons pour notre exemple obtenir la solution intermédiaire de la figure 4.16, accompagnée ici par le graphe  $I$  ainsi que la solution finale. On obtient ici une solution en 4 étapes de communications, chacune de durée 1. Le coût total de l'ordonnancement est donc de  $4 + 4 = 8$  soit un temps à un facteur de  $\frac{8}{\frac{10}{3}} = \frac{12}{5} = 2,4$  de la borne inférieure.

On peut construire toute une famille d'exemples problématiques suivant le modèle de ce

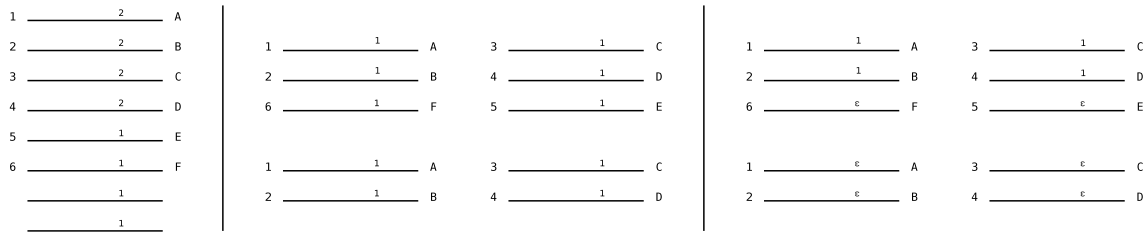


FIG. 4.16 – Graphe  $I$ , solution intermédiaire à un facteur de  $\frac{12}{5}$  et solution finale à un facteur de  $\frac{9}{5}$

graphe en augmentant le nombre d'arêtes en même temps que  $k$ . La figure 4.17 illustre ainsi le second graphe de la famille, pour  $k = 4$ . La borne inférieure est ici de  $\frac{5+8\epsilon}{4} + 2 \approx \frac{13}{4}$  tandis que l'ordonnancement le plus mauvais que l'on puisse obtenir par GGP en tant que solution intermédiaire a un coût de  $4 + 4 = 8$  ce qui nous amène à un facteur  $\frac{8}{\frac{13}{4}} = \frac{32}{13} \approx 2,46$  de l'optimum.

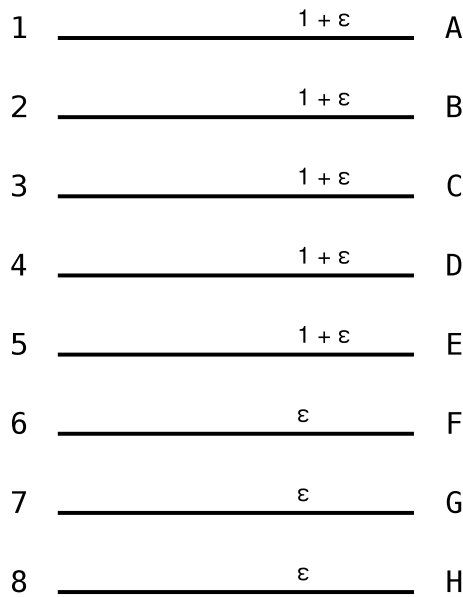


FIG. 4.17 – Second graphe problématique ( $k = 4$ )

En continuant cette famille de graphe, on peut calculer la borne inférieure sur le temps de redistribution en fonction de  $k$  :  $\eta_s = 2$  et  $\eta_d = \frac{(k+1)(1+\epsilon)}{k} \approx \frac{k+1}{k}$

$$\rightarrow \eta = 2 + \frac{k+1}{k}$$

Il est possible d'obtenir à l'aide de GGP une solution intermédiaire d'un temps de  $4 + 4 = 8$ .

On se retrouve donc à un facteur  $\frac{8}{2 + \frac{k+1}{k}}$  tendant vers  $\frac{8}{3}$  lorsque  $k$  tend vers  $\infty$ .

Nous avons, dans un second temps, cherché une autre série d'exemples pour laquelle une mauvaise borne se conserverait après la réduction finale des poids. Nous utilisons cette famille

pour prouver que GGP ne peut en aucun cas avoir un ratio d'approximation inférieur à 2 : il est impossible de prouver que pour tout graphe de communication  $G$  et toute valeur de  $k$  le ratio entre le temps d'exécution d'un ordonnancement obtenu par GGP et le temps optimal d'exécution de la redistribution est inférieur à un ratio  $b$  pour toute valeur de  $b$  inférieure à 2.

Considérons le graphe de la figure 4.18, associé à une valeur de  $k$  égale à 2.

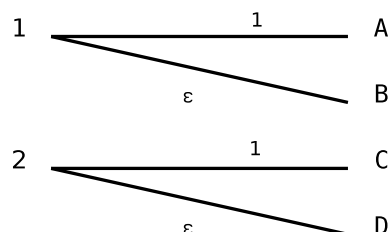


FIG. 4.18 – Graphe problématique (seconde famille) pour GGP

Pour ce graphe, la borne inférieure a comme valeur :  $\eta = \eta_d + \eta_s = 1 + \epsilon + 2 \approx 3$ . L'exécution de GGP n'étant pas déterministe (à cause de l'étape de choix d'un couplage), il est ici possible d'obtenir différentes solutions. Nous supposons pour notre exemple obtenir la solution intermédiaire de la figure 4.19, accompagnée ici de la solution finale de GGP. On remarque un coût de  $2 + 1 + 1 = 4$  pour les deux versions, c'est-à-dire y compris lorsque l'augmentation des poids liées à l'arrondi est éliminée. Ceci nous amène à un facteur de  $\frac{4}{3}$  de l'optimum.

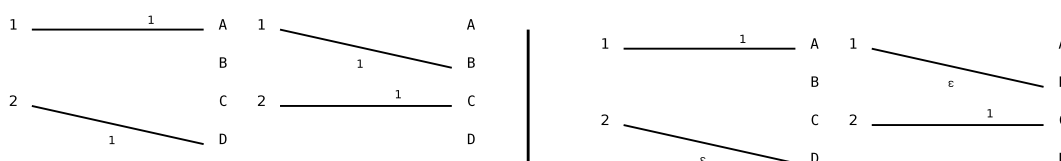


FIG. 4.19 – Ordonnancement à un facteur de  $\frac{4}{3}$

On peut construire toute une famille d'exemples problématiques suivant le modèle de ce graphe en augmentant le nombre d'arêtes en même temps que  $k$ . La figure 4.20 illustre ainsi le second graphe de la famille, pour  $k = 3$ , accompagné de l'ordonnancement obtenu par GGP. La borne inférieure est ici de  $3 + 1 + \epsilon + \epsilon \approx 4$  tandis que l'ordonnancement a lui un coût de  $3 + 1 + 1 + 1 = 6$  ce qui nous amène à un facteur  $\frac{6}{4} = \frac{3}{2}$  de l'optimum.

En continuant cette famille de graphe, le temps optimal a comme valeur  $k + 1 + (k - 1) \times \epsilon \approx k + 1$  tandis qu'il est possible d'obtenir un ordonnancement d'une durée de  $k + k = 2k$ . On se retrouve donc pour chaque cas à un facteur  $\frac{2k}{k+1}$  de l'optimal, ce qui tend vers 2 lorsque  $k$  tends vers l'infini.

Par conséquent l'algorithme GGP ne peut avoir un ratio d'approximation inférieur à 2.

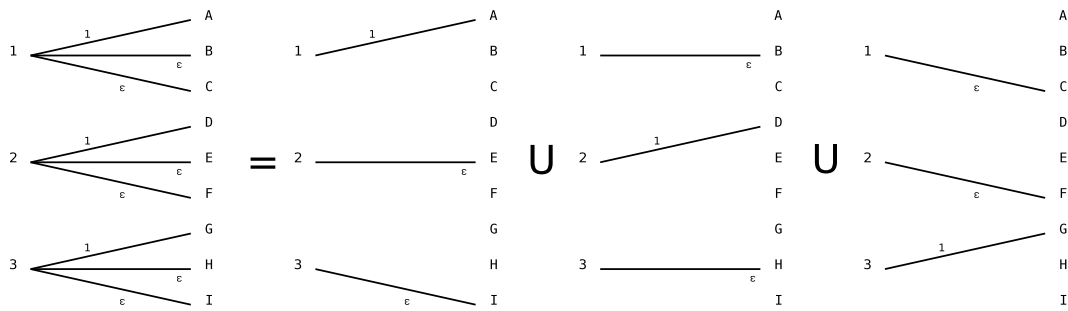


FIG. 4.20 – Ordonnancement à un facteur de  $\frac{3}{2}$

## 4.2 OGGP

L'étude de GGP a permis de montrer que cet algorithme garantissait un ratio d'approximation de  $\frac{8}{3}$ . Néanmoins, nous avons pu trouver une série d'exemples où la qualité des résultats se rapprochait d'un facteur 2 de l'optimum, ce qui signifie qu'il est impossible de prouver pour GGP un ratio d'approximation inférieur à 2. En d'autres termes la garantie sur la qualité des solutions que nous avons nous assure que toute solution ne peut avoir un temps plus long que  $\frac{8}{3}$  fois le temps optimal. Nous savons de plus qu'il est impossible d'améliorer cette garantie pour obtenir un facteur inférieur à 2. Nous avons donc cherché à améliorer le comportement de GGP de manière à augmenter les performances, au minimum sur la famille d'exemples particulière de la section précédente. OGGP est l'algorithme résultant de cette amélioration.

Nous présentons dans cette section l'algorithme OGGP (*Optimized Generic Graph Peeling*). Nous introduisons dans un premier temps l'idée principale derrière cet algorithme, puis comme pour GGP, une version formelle permettant une analyse de ses différentes propriétés.

### 4.2.1 Principe de fonctionnement

Le problème de GGP sur la famille d'exemples des figures 4.18, 4.19 et 4.20 vient du fait que lors de l'étape d'arrondi, l'algorithme perd sa connaissance des poids réels de chaque arête. Il n'est donc plus en mesure de grouper ensemble des arêtes de poids proches et peut donc dans certains cas regrouper des arêtes de poids ayant jusqu'à une unité d'écart (la valeur maximale de l'arrondi). Nous allons donc pour OGGP travailler en associant à chaque arête non pas une, mais deux fonctions de poids. La première fonction sera celle de GGP (où le poids se retrouve arrondi), tandis que la seconde fonction fonctionnera de manière identique, mais sans la première étape d'arrondi des poids. Nous nommerons dans cette section "*poids réels*" les poids donnés par cette seconde fonction et "*poids virtuels*" les poids fournies par la première de ces fonctions. Ces fonctions seront appelées respectivement "*fonction de poids réels*" et "*fonction de poids virtuels*".

Mise à part l'introduction de la fonction de poids réels, OGGP est quasi-identique à GGP. OGGP est obtenu en modifiant GGP lors de l'étape de sélection d'un couplage. L'idée est la suivante : dans GGP, lors de l'étape de l'épluchage du graphe en un ensemble de couplages, on itère en sélectionnant à chaque itération un couplage que l'on élimine ensuite du graphe. Dans GGP, aucune contrainte n'est posée sur le choix du couplage mis à part le fait que le couplage doit être un couplage parfait. Nous allons pour OGGP imposer un choix plus intelligent. On va chercher à éliminer en un seul couplage  $M$  le maximum de poids sur les arêtes du graphe. Le poids éliminé dans le graphe sur chaque arête de  $M$  dépend dans GGP du poids le plus faible

de toutes ses arêtes. On cherche donc ici parmi tous les couplages parfaits possibles celui dont le poids (réel) le plus faible de ses arêtes est maximal. On espère ainsi d'une part réduire le nombre total d'étapes pour l'ordonnancement, d'autre part éviter de regrouper ensemble des communications de tailles trop différentes.

Revenons par exemple sur le graphe problématique de la figure 4.15. Les phases d'extensions de GGP le laissent inchangé (mis à part l'arrondi des poids). On dispose pour le choix du premier couplage de 4 possibilités, respectivement :  $\{(1, A), (2, C)\}$ ,  $\{(1, A), (2, D)\}$ ,  $\{(1, B), (2, C)\}$ ,  $\{(1, B), (2, D)\}$  de poids minimaux  $1, \epsilon, \epsilon, \epsilon$ . OGGP choisit donc à coup sûr le premier couplage qui permet d'obtenir un ordonnancement plus efficace tandis que GGP a une chance sur deux de regrouper une arête de poids  $\epsilon$  avec une arête de poids 1.

### 4.2.2 Algorithme formel

Nous présentons ici dans un premier temps, un algorithme permettant d'obtenir un couplage parfait de poids minimal maximum, puis une description formelle de OGGP.

#### Couplage de poids minimal maximum

L'algorithme de couplage est basé sur un algorithme de Bongiovanni et al. [19] qui maximise le poids minimal d'un couplage. Il fonctionne de manière simple : on classe l'ensemble des arêtes de l'arête de poids le plus élevé à l'arête de poids le plus faible. On essaie alors de construire un couplage parfait avec uniquement la première de ces arêtes. Si c'est impossible, on recommence avec les deux premières arêtes. On itère ainsi en ajoutant à chaque étape l'arête restante de poids le plus élevé et en cherchant de nouveau un couplage parfait. Le premier couplage parfait obtenu est le couplage recherché. La figure 5 décrit cet algorithme de manière plus formelle.

**Entrées** : un graphe biparti  $G = (V_1, V_2, E, w)$  sur lequel il est possible de trouver un couplage parfait.

**Sorties** :  $M$  : un couplage parfait valué, de poids minimal maximum.

- 1  $G' = (V_1, V_2, E' = \emptyset), M = \emptyset, G'' = (V_1, V_2, E'' = E);$
  - 2 construire la liste  $L = \{e_1, \dots, e_m\}$  des arêtes triées par poids décroissants;
  - 3 **tant que**  $M$  n'est pas un couplage parfait de  $G$  **faire**
  - 4     construire  $A = \{e_1, \dots, e_h\}$  l'ensemble des  $h$  premiers éléments de  $L$ , de poids identiques :  
 $\forall e_i \in A, w(e_i) = c$  où  $c$  est une constante et  $w(e_{h+1}) < w(e_h);$
  - 5      $E' = E' \cup A$
  - 6      $E'' = E'' \setminus A;$
  - 7      $M =$  un couplage maximal de  $G';$
  - 8      $L = \{e_{h+1}, \dots, e_m\};$
- fin**

**Algorithme 5** : Algorithme pour l'extraction d'un couplage parfait de poids minimal maximum

**Proposition 10** *L'algorithme de la figure 5 retourne un couplage de poids minimal maximum.*

**Preuve** : Soit  $M$  le couplage retourné par l'algorithme. Soit  $A$  l'ensemble des arêtes ajoutées à  $G'$  à l'étape 5. On sait qu'au moins une arête  $l$  de  $A$  appartient à  $M$  car il était impossible de

trouver un couplage parfait quant  $A$  n'appartenait pas à  $G'$ . On sait de plus que  $\forall e \in G, w(e) > w(l) \Rightarrow e \in G'$ . Supposons pour une contradiction que  $M'$  soit un couplage parfait meilleur que  $M$  (son poids minimal est plus grand que celui de  $M$ ).  $M'$  est tel que :  $\forall e \in M', w(e) > w(l)$ . On en déduit donc que  $M' \subset G'$ , ce qui est une contradiction car tout couplage de  $G'$  ne peut être parfait que s'il contient au moins une arête de  $A$  et donc de poids égal à  $w(l)$ .

$M$  est donc bien un couplage parfait de poids minimal maximum. ■

## OGGP

La figure 6 décrit étapes par étapes l'algorithme OGGP. Remarquons que les étapes 1,2 et 3 sont identiques à celles de GGP.

### 4.2.3 Propriétés

Les propriétés de OGGP découlant pour la plupart des propriétés de GGP, leur étude sera beaucoup moins complexe. Nous commençons ici également par une étude de la complexité de l'algorithme au pire cas, suivie par une étude de son ratio d'approximation.

#### Complexité

Dans un premier temps, nous devons calculer la complexité du nouvel algorithme de couplage. L'algorithme fonctionne en itérant sur l'ensemble des arêtes, et en cherchant un couplage à chaque itération. Sa complexité est donc de :  $O(m(J) \times \sqrt{n(J)}m(J)) = O(m(J)^2 \sqrt{n(J)})$ .

On en déduit immédiatement la complexité de OGGP :  $O(\sqrt{n(G)}(m(G) + n(G))^3)$ .

#### Facteur d'approximation

OGGP ne différant de GGP que dans le choix du couplage parfait, toute solution obtenue par OGGP est également obtainable par GGP. Par conséquent on en conclut directement que OGGP, tout comme GGP, un algorithme d'approximation d'un ratio d'approximation de  $\frac{8}{3}$ .

En revanche, contrairement à GGP, il ne nous a pas été possible de trouver un exemple impliquant que le ratio d'approximation de OGGP est au moins 2. Nous supposons donc que bien que doté du même facteur d'approximation que GGP, OGGP se comporte mieux en pratique. Afin de vérifier une telle supposition, nous proposons d'effectuer une série d'expériences et de simulations.

**Entrées** : un graphe biparti  $G = (V_1, V_2, E, w_G)$ , un entier  $k$ .

**Sorties** : un ensemble  $S$  de couplages pondérés.

- 1 construire un graphe  $H = (V_1, V_2, E, w_H)$  tel que  $\forall e \in E, w_H(e) = \lceil w_G(e) \rceil$ ;  
 $k = \min(k, n(G))$ ;  
 /\* construire un graphe  $I$  tel que  $\frac{P(I)}{k} \geq W(I)$  et  $\frac{P(I)}{k} \in \mathbb{N}$  : \*/
- 2 construire  $I = (V_{1_I}, V_{2_I}, E_I, w_I)$  de la manière suivante :  
**si**  $\frac{P(H)}{k} < W(H)$  **alors**  $p = kW(H) - P(H)$ ;  
**sinon**  $p = k \left\lceil \frac{P(H)}{k} \right\rceil - P(H)$ ;  
 /\*  $p$  a comme valeur le total des poids des arêtes à rajouter. \*/  
 /\* le nombre d'arêtes à rajouter est alors  $n = \left\lceil \frac{p}{W(H)} \right\rceil$ . \*/  
 /\* on construit donc : \*/  
 $V_{1_I} = \{s_1, \dots, s_{|V_1|}, s'_1, \dots, s'_{n-1}, s'_n\}$ ;  
 $V_{2_I} = \{r_1, \dots, r_{|V_2|}, r'_1, \dots, r'_{n-1}, r'_n\}$ ;  
 $E_I = \{e_1, \dots, e_m, e'_1, \dots, e'_{n-1}, e'_n\}$  avec  
 $\{s_1, \dots, s_{|V_1|}\} = V_1, \{r_1, \dots, r_{|V_2|}\} = V_2, \{e_1, \dots, e_m\} = E, \forall e \in E, w_i(e) = w(e)$ ;  
**pour**  $i \in \{1, \dots, n-1\}$  **faire**  
 |  $(s'_i, r'_i) = e'_i \in E_I$ , et  $w_I(e'_i) = W(H)$   
**fin**  
 $(s'_n, r'_n) = e'_n \in E_I$ , et  $w_I(e'_n) \leq W(H)$ ;  
 $\sum_{i=1}^n w_I(e'_i) = p$ ;  
 /\* transformer  $I$  en un graphe régulier sur les poids : \*/
- 3 construire un graphe  $J = (V_{1_J}, V_{2_J}, E_J, w_J)$ , qui soit  $\frac{P(I)}{k}$ -régulier sur les poids en utilisant l'algorithme de la figure 2.;
- 4  $R = \emptyset, S = \emptyset$  ;  
 Soit  $w_p$  la fonction de  $E_J$  dans  $\mathbb{Q}$  définie par :  
 -  $\forall e \in E, w_p(e) = w(e)$ ,  
 -  $\forall e \in E_J, e \notin E, w_p(e) = w_J(e)$
- 5 **tant que**  $E_J \neq \emptyset$  **faire**  
 | choisir un couplage parfait  $M$  de poids minimal maximum dans  $J$  avec  $w_p$  comme fonction de valuation et l'algorithme de la figure 5;  
 | changer  $w_M$  de sorte que  $\forall e \in M, w_M(e) = s(M)$  le plus petit poids de toutes les arêtes de  $M$ ;  
 | ajouter  $M$  à  $R$ , l'ensemble des couplages formant la solution;  
 |  $\forall e \in M$  changer  $w_J(e)$  en  $w_J(e) - w_M(e)$ ;  
 |  $\forall e \in M$  changer  $w_p(e)$  en  $\max(w_p(e) - w_M(e), 0)$ ;  
 | enlever de  $E_J$  toutes les arêtes de poids  $w_J$  égal à 0;  
**fin**
- 6 enlever de  $R$  toute arête  $e$  telle que  $e \notin E$ ;
- 7  $R$  étant composé des couplages  $M_1, \dots, M_h$ , **pour**  $i \in \{1, \dots, h\}$  **faire**  
 |  $M'_i = \emptyset$ ;  
 | **pour chaque arête**  $e$  **de**  $M_i$  **faire**  
 | | ajouter  $e$  dans  $M'_i$  avec  $w_{M'_i}(e) = \min(w_i(e), w_G(e))$ ;  
 | |  $w_G(e) = \max(w_G(e) - w_{M'_i}(e), 0)$ ;  
 | **fin**  
 | ajouter  $M'_i$  dans  $S$ ;  
**fin**

Algorithme 6 : Algorithme OGGP

## 4.3 Expériences

Afin de comparer et valider nos différents algorithmes, nous avons effectué une série d'expériences et de simulations. Nous présentons dans cette section l'ensemble des tests effectués à l'aide de GGP et OGGP. Les différents tests sont ici présentés dans leur ordre d'exécution chronologique qui correspond également à l'ordre de la plus faible à la plus forte complexité de mise en œuvre.

Nous présentons tout d'abord notre bibliothèque de manipulation de graphes bipartis, *libkpbs*, spécialement développée pour implanter GGP et OGGP. Cette bibliothèque nous a permis d'effectuer des simulations permettant de comparer entre eux nos deux algorithmes, ainsi que de les comparer à des heuristiques existantes. Les résultats étant concluants, nous avons effectué une série de redistributions réelles, exécutées sur un réseau local : un premier jeu de tests a été implanté à l'aide de la bibliothèque *MPICH* [30], tandis qu'un second jeu de tests a été implanté entièrement à la main au dessus de la *glibc* afin de disposer d'un plus grand contrôle sur l'exécution de la redistribution.

### 4.3.1 Une bibliothèque de manipulation de graphes bipartis : *libkpbs*

Afin d'automatiser l'exécution de GGP et OGGP, nous avons rapidement implanté ces algorithmes sur ordinateur. Afin de minimiser les dépendances requises nous avons choisi de développer notre propre bibliothèque de manipulation de graphes bipartis. De plus, bien qu'il existe de nombreuses bibliothèques permettant de manipuler des graphes, à notre connaissance, aucune d'entre elles ne se focalise sur la gestion de graphes bipartis. Le développement d'un outil propre nous a donc permis de limiter la surcharge de la bibliothèque utilisée par un nombre élevé de fonctions inutiles, et nous a également donné la possibilité de garder un contrôle sur son développement.

La *libkpbs* a été implanté en C++, avec comme seules dépendances les bibliothèques *glibc* et *STL*. Le choix du C++ a été effectué pour des raisons de performances. De plus, l'interface de programmation fournie par la *libkpbs* a été développée de manière à être fortement orientée objet. Notons enfin, que le choix d'implanter nos algorithmes sous forme d'une bibliothèque a été réalisé afin de faciliter une intégration de ceux-ci dans d'autres projets. Nous avons notamment collaboré avec l'équipe PARIS de l'IRISA de Rennes pour l'intégration de la *libkpbs* à PACO++ [27].

Nous donnons un exemple d'utilisation de notre bibliothèque en annexe, section B.1. Ce programme fonctionne en lisant un fichier contenant une matrice de communication. Il construit le graphe biparti correspondant à celle-ci puis calcule et affiche la borne inférieure sur le temps de redistribution ainsi que l'ordonnancement obtenu par l'utilisation de OGGP. Il permet donc de comparer rapidement sur tout exemple le temps pris par OGGP et le temps optimal. Les classes *bigraph*, *edge*, *node*, *matching* représentent respectivement un graphe biparti, une arête, un sommet, et un couplage et leur utilisation se fait de manière très intuitive. Un ordonnancement solution du problème est lui représenté par la classe *kpbs\_approximation*. La documentation relative à la bibliothèque est intégrée avec toute distribution de celle-ci, étant générée à partir du code source à l'aide de *doxygen* [3]. Nous ne rentrerons donc pas plus dans les détails d'utilisation ici.

### 4.3.2 Simulations

Les premiers tests réalisés ont visé à comparer l'efficacité de différents algorithmes au niveau théorique, c'est-à-dire en travaillant uniquement sur des graphes. Nous utilisons ici, en plus de

GGP et OGGP, deux heuristiques gloutonnes. Celles-ci sont basées sur une heuristique développée par Desprez et al. [28] et légèrement modifiées pour prendre en compte la contrainte sur  $k$ . Ces heuristiques, en  $\mathcal{O}(m(G)^2 \times \sqrt{n(G)})$ , sont de complexité moindre, mais ne sont pas des algorithmes d'approximation et n'apportent donc aucune garantie sur la qualité de leurs résultats. Nous avons en particulier prouvé [48] qu'elles peuvent dans certains cas fournir des temps d'exécutions jusqu'à  $k$  fois plus longs que le temps optimal.

Elles fonctionnent toutes deux suivant le même principe : On choisit un couplage *maximal* dans le graphe et on ne retient dans ce couplage que les  $k$  meilleures arêtes suivant un certain critère. On élimine ensuite le couplage du graphe et on recommence jusqu'à ce qu'il n'y ait plus aucune arête dans le graphe initial.

La première heuristique, appelée "heuristique sur les poids" garde dans le couplage les  $k$  arêtes de poids le plus élevé, tandis que la seconde, "heuristique sur les degrés" garde dans le couplage les arêtes incidentes aux sommets de plus haut degré. Le but étant pour la première d'essayer de réduire la quantité de données à envoyer, et pour la seconde d'essayer de réduire le nombre d'étapes. La figure 7 décrit en détail le fonctionnement de l'heuristique sur les poids.

**Entrées** : un graphe biparti  $G$ , un entier  $k$ .  
**Sorties** : un ensemble de couplages valués  $S$ .  
**tant que**  $E \neq \emptyset$  **faire**  
    | trouver un couplage maximal  $M$ ;  
    | enlever des arêtes de  $M$  en gardant les  $k$  arêtes de poids le plus élevé;  
    | retirer  $M$  du graphe  $G$ ;  
**fin**

**Algorithme 7** : Heuristique sur les poids

Nous avons exécuté ces heuristiques ainsi que les algorithmes d'approximation sur des graphes générés aléatoirement et comparé les temps des ordonnancements obtenus entre eux.

### Simulation des heuristiques

Chaque heuristique a été testée sur un échantillon de 100000 graphes aléatoires (pour  $k$  fixé, et  $\beta = 1$ ) avec 20 sommets de chaque côté du graphe. Les graphes ont été choisis aléatoirement de la manière suivante : le nombre d'arêtes, leurs positions, ainsi que leurs poids ont été tirés aléatoirement avec une loi de probabilité uniforme. Deux cas ont été considérés : des graphes de poids élevés tirés entre 1 et 100000 et des graphes de poids faibles tirés entre 1 et 20. Le résultat chaque heuristique est calculé puis le coût de la solution obtenue est divisé par la borne inférieure  $\eta$  sur le temps de redistribution. Nous appelons ce ratio *ratio d'évaluation*.

Les courbes des figures 4.21, 4.22, 4.23 et 4.24 montrent l'évolution de la moyenne des ratio d'évaluation ainsi que du ratio d'évaluation maximal (sur les 100000 tests) lorsque la valeur de  $k$  augmente.

Pour les heuristiques, sur les graphes de poids faibles, le ratio maximal est toujours inférieur à 2,6 et le ratio moyen à 1,8 ; les résultats étant sensiblement de même ordre sur les graphes de poids élevés.

On remarque que pour une valeur de  $k$  égale à 1, les deux heuristiques obtiennent la solution optimale qui consiste à sérialiser les communications. Les performances se dégradent ensuite avec un pic autour de  $k = 5$  puis s'améliorent légèrement pour rester presque constantes à partir de  $k = 10$ .

Il semblerait donc que ces heuristiques soient plus efficaces pour les problèmes où  $W > \frac{P}{k}$  et se comportent moins bien lorsque le nombre de flux parallèles autorisés est relativement limité.

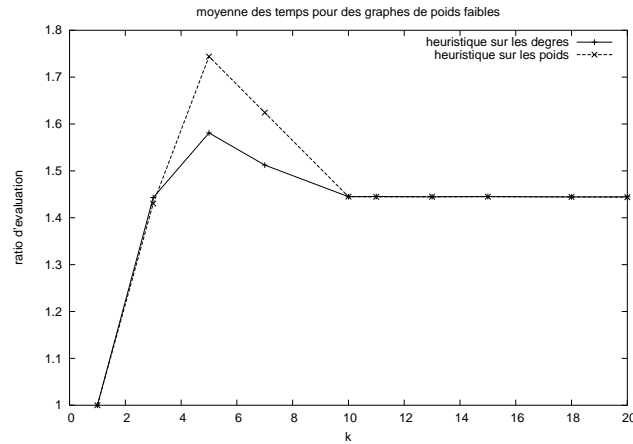


FIG. 4.21 – Simulations des heuristiques gloutonnes sur des graphes de poids faibles, moyenne des temps

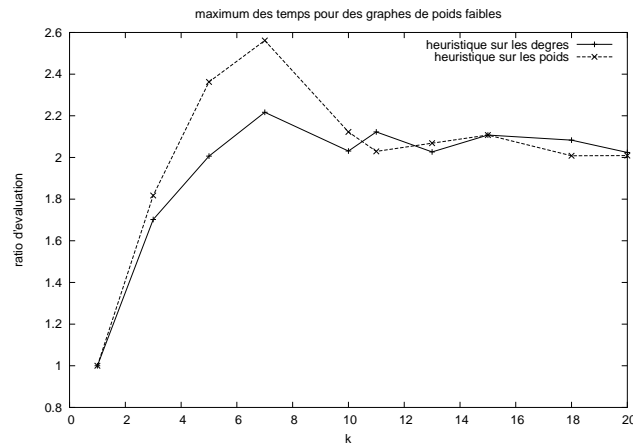


FIG. 4.22 – Simulations des heuristiques gloutonnes sur des graphes de poids faibles, maximum des temps

### Simulation de GGP et OGGP

Les simulations de GGP et OGGP ont été conduites dans les mêmes conditions que celles sur les heuristiques. En particulier, les graphes aléatoires utilisés ici sont les mêmes que ceux utilisés précédemment afin de permettre une comparaison entre nos algorithmes et les heuristiques.

**Comparaison entre GGP et OGGP** La figure 4.25 montre le comportement du ratio d'évaluation lorsque  $k$  varie, pour les graphes de poids faibles.

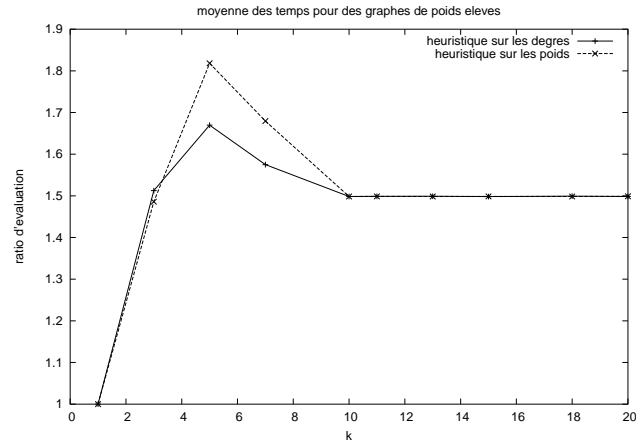


FIG. 4.23 – Simulations des heuristiques gloutonnes sur des graphes de poids élevés, moyenne des temps.

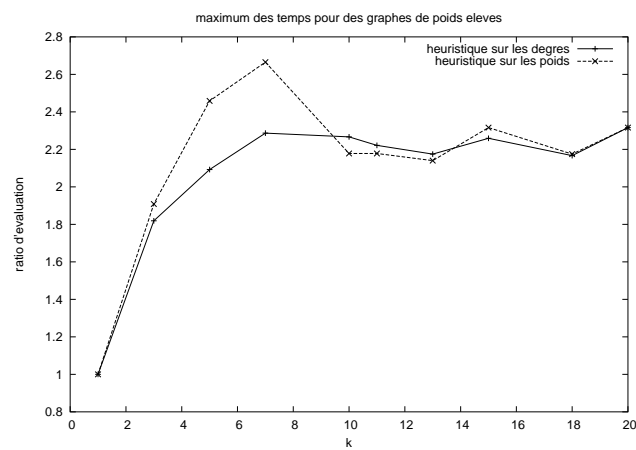


FIG. 4.24 – Simulations des heuristiques gloutonnes sur des graphes de poids élevés, maximum des temps.

On peut remarquer que lorsque  $k$  augmente, le ratio d'évaluation augmente puis se stabilise. OGGP donne de meilleurs résultats que GGP y compris en comparant le plus mauvais ratio obtenu par OGGP au ratio moyen obtenu par GGP.

Les figures 4.25 et 4.26 montrent le comportement du ratio d'évaluation lorsque  $k$  varie, pour les graphes de poids élevés.

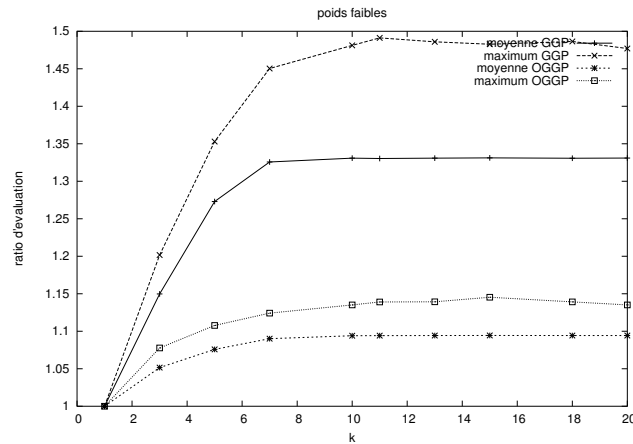


FIG. 4.25 – GGP et OGGP pour des graphes de poids faibles

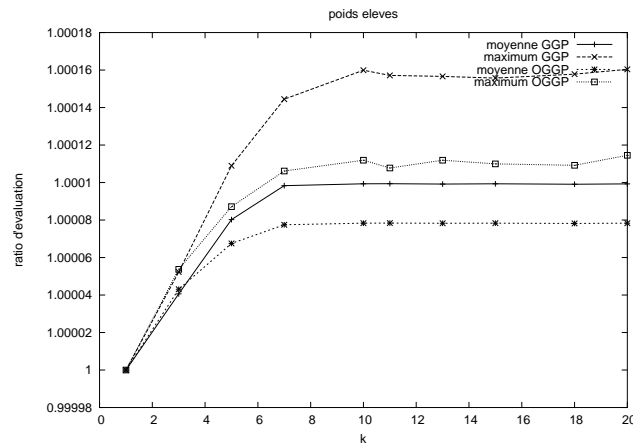


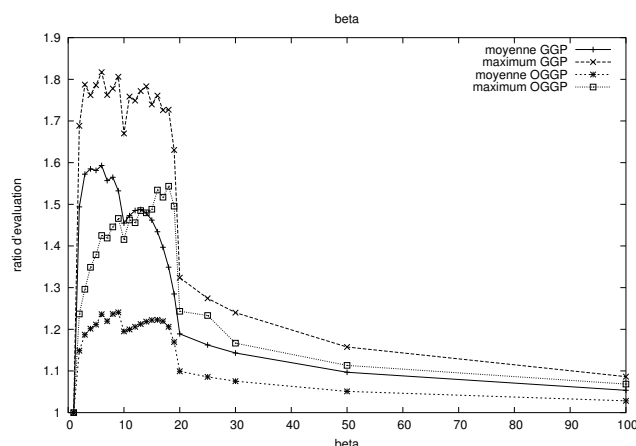
FIG. 4.26 – GGP et OGGP pour des graphes de poids élevés

Les courbes obtenues évoluent de manière semblable, mais le ratio d'évaluation est ici beaucoup plus proche de 1. Sur ces cas, la différence entre OGGP et GGP est beaucoup plus petite.

Nous avons également cherché à exécuter nos deux algorithmes pour des cas plus problématiques, notamment lorsque  $\beta$  est plus élevé que les poids des arêtes du graphe d'entrée.

Nous considérons ici le cas non normalisé où  $\beta \neq 1$ . Le graphe de la figure 4.27 montre comment le ratio d'évaluation varie lorsque  $\beta$  augmente. Les poids des arêtes sont ici générés aléatoirement entre 1 et 20, et, pour chaque point du graphe,  $k$  varie aléatoirement entre 1 et 20.

On peut voir que OGGP donne encore de meilleurs résultats que GGP. En augmentant,  $\beta$

FIG. 4.27 – GGP et OGGP lorsque  $\beta$  varie

diminue la qualité des résultats obtenus jusqu'à ce qu'il devienne plus grand que tous les poids du graphe (supérieur à 20). En effet, en augmentant  $\beta$  augmente également la borne inférieure sur le temps de redistribution, ce qui explique une amélioration du ratio d'évaluation.

Les résultats sont ici plus mauvais que pour les autres cas testés avec le ratio d'évaluation de OGGP montant jusqu'à une valeur de 1,5.

Ces simulations valident donc la supériorité de OGGP sur GGP. Ainsi, bien que les deux algorithmes aient le même ratio d'approximation, en moyenne OGGP se comporte mieux que GGP, ce qui conforte les choix réalisés pour son développement.

**Comparaison entre GGP et les heuristiques** Lors de l'exécution des différents algorithmes sur des graphes de poids faibles, on peut voir que GGP donne de meilleurs résultats que les heuristiques. En particulier, bien que le ratio d'évaluation moyen soit sensiblement identique, le ratio d'évaluation maximal est jusqu'à 1,5 fois plus élevé pour les heuristiques que pour GGP.

En comparant GGP avec les heuristiques sur des graphes de poids plus élevés, le fossé entre eux s'élargit. La grande différence de qualité des résultats que l'on observe ici s'explique par l'utilisation par GGP de la préemption des communications, ce qui n'a pas un impact fort lorsque le délai d'initialisation a une valeur élevée. C'est la préemption qui nous permet d'obtenir ici des temps quasi optimaux.

### 4.3.3 Tests locaux exécutés avec la bibliothèque *MPICH*

L'étape suivante de notre série d'expériences a consisté à effectuer une série de redistributions, afin de tester les comportements des différentes méthodes de redistribution dans un cadre plus réaliste.

Nous avons utilisé pour cette expérience deux grappes de Pentium 2 cadencés à 1,5 Ghz fonctionnant sous linux. Les cartes réseaux sont des cartes ethernet à 100Mbit/s, chaque grappe étant relié à un commutateur réseau. Le lien entre les deux commutateurs est également à 100Mbit/s.

L'utilisation d'un réseau local répond à deux besoins :

- la facilité de mise de œuvre : l'obtention des machines, ainsi que l'installation des logiciels requis s'avère plus simple que sur des machines distantes ;

- l'absence de perturbations : les tests sur ce réseau ayant été effectués dans des salles de TP d'informatique de l'IUT Nancy-Brabois durant les week-ends, le réseau peut être considéré comme un réseau dédié à ces expériences.

L'utilisation d'un tel réseau nous pose néanmoins le problème que  $k$  vaut 1 ( $\frac{100\text{Mbit/s}}{100\text{Mbit/s}}$ ). Dans un tel cas, aussi bien OGGP que GGP séquentialisent toutes les communications. Afin de tester les cas qui nous intéressent le plus, c'est-à-dire lorsque  $k \neq 1$ , nous avons limité la bande passante entrante et sortante de chaque interface à  $\frac{100}{k}$  Mbit/s. Pour ce faire, nous avons utilisé le module *rshaper* [7] implantant un mécanisme de qualité de service sur le principe du seuil à jeton, nous permettant ainsi d'effectuer les contrôles voulus sur la bande passante. Les expériences ont été conduites pour des valeurs de  $k$  de 3, 5 et 7.

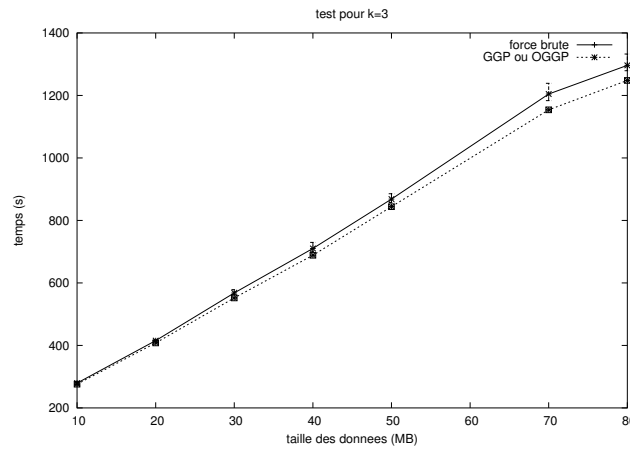


FIG. 4.28 – Temps obtenus pour des redistributions par force brute ou ordonnancées à l'aide de GGP pour  $k = 3$  (basées sur *mpich*)

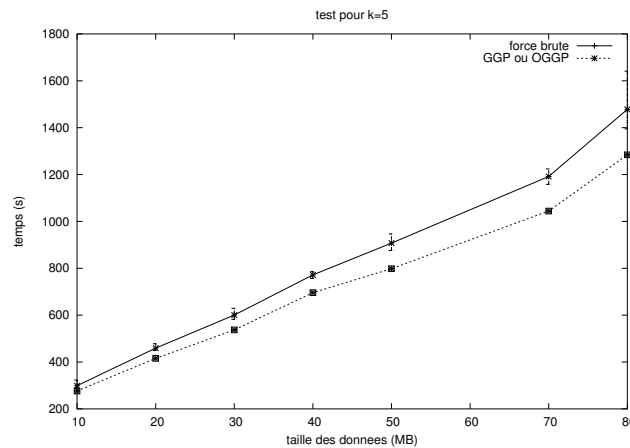


FIG. 4.29 – Temps obtenus pour des redistributions par force brute ou ordonnancées à l'aide de GGP pour  $k = 5$  (basées sur *mpich*)

Nous avons implémenté deux types différents de redistributions. Nous avons choisi d'effectuer

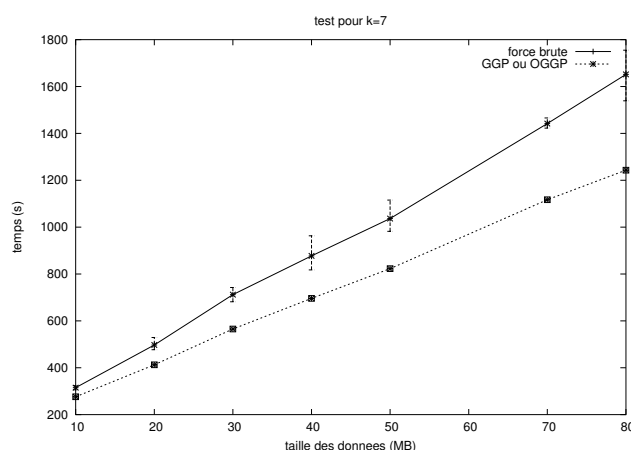


FIG. 4.30 – Temps obtenus pour des redistributions par force brute ou ordonnancées à l’aide de GGP pour  $k = 7$  (basées sur *mpich*)

ces tests en langage C, à l’aide de la bibliothèque *MPICH* [30], cette bibliothèque étant utilisée couramment.

Dans le première approche nous avons considéré une redistribution par force brute, identique à celle décrite section 3.1.2 : toutes les communications sont lancées simultanément à l’aide de la fonction *MPIsend*, le programme se terminant en même temps que le dernier transfert. Le protocole TCP s’occupe alors de contrôler la congestion du réseau.

Dans la seconde approche, nous divisons les communications en différentes étapes, synchronisées à l’aide de barrières (*MPIBarrier*). Seules des communications synchrones prennent place et pas plus d’une communication n’est effectuée par chaque nœud à chaque étape.

L’ensemble des temps de communication a été mesuré à l’aide de la fonction *ntp\_gettime* de la glibc.

Dans ces tests, les 10 nœuds émetteurs de la première grappe communiquent vers chacun des 10 nœuds receveurs (le graphe de communication est un graphe complet). La quantité de données à envoyer ou recevoir est tirée uniformément entre 10 et  $n$  méga-octets. Les figures 4.28, 4.29 et 4.30 montrent l’augmentation des temps de communication suivant les deux approches de redistribution lorsque  $n$  augmente. Chaque point du graphe représente la moyenne des temps obtenus pour une vingtaine de graphes aléatoires différents. Est également noté pour chaque point le temps minimal et maximal réalisé.

Plusieurs observations peuvent être faites :

- Les temps des communications ordonnancées sont de 5 et 20% moins longs que les temps des redistributions par force brute. La bibliothèque *MPICH* que nous avons utilisé semble liée à ce gain de performances. Afin de vérifier néanmoins cette hypothèse, nous avons effectué par la suite une nouvelle série d’expériences réalisées en utilisant uniquement des appels systèmes standards. Cette série d’expérience ainsi que les conclusions que nous en avons tiré sont présentées dans la section suivante.
- Les barrières de synchronisations coûtent extrêmement peu de temps. Bien que *OGGP* fournisse des ordonnancements ayant en moyenne 50% d’étapes de communication que les ordonnancements fournis par *GGP*, les deux algorithmes fournissent des résultats prenant un temps quasi-identique. Nous pensons que l’écart entre les deux algorithmes pourrait se creuser sur un réseau non dédié où les perturbations réseau pourraient augmenter le coût

des synchronisations.

- L'algorithme par force brute ne se comporte pas de manière déterministe. En lançant plusieurs fois les mêmes expériences, des variations de temps pouvant aller jusqu'à 10% du temps moyen peuvent être observées. Les variations sont beaucoup plus faibles dans le cas de l'approche ordonnancée.
- Lorsque la bande passante diminue (i.e. lorsque  $k$  augmente) l'écart entre les deux types de redistributions augmente.

#### 4.3.4 Tests locaux exécutés manuellement

Une partie des résultats précédents dépendant directement du fonctionnement interne de *MPICH*, nous avons décidé de reprogrammer l'ensemble des tests effectués en utilisant uniquement des appels standards unix, afin de garder une très bonne vue du déroulement du programme. Nous avons pour ce faire utilisé des processus légers pour la programmation des envois simultanés de l'algorithme par force brute. Les conditions matérielles ne varient pas de celles des expériences précédentes avec deux grappes de 10 ordinateurs reliées entre elles par un réseau à 100Mbit/s. Notons au passage que nous avons également effectué ces tests en utilisant le programme *tc* pour la gestion de la qualité de service, les résultats étant identiques à ceux obtenus avec *rshaper*.

Une fois encore, nous comparons les temps obtenus entre une redistribution ordonnancée et une redistribution par force brute lorsque la taille des données varie.  $k$  est fixé à une valeur de 5. Deux motifs de redistribution différents ont été considérés ici :

- Nos premiers tests ont été effectués comme précédemment sur des graphes aléatoires. Nous avons néanmoins procédé différemment sur la manière de générer chacun d'entre eux. Dans le cas présent, nous avons fixé arbitrairement le nombre total d'arêtes du graphe, puis tiré leurs positions et poids à l'aide d'une loi uniforme. Le but étant ici de voir l'influence du nombre d'arêtes sur la qualité des résultats.

Les figures 4.31, 4.32 et 4.33 montrent les temps obtenus pour des graphes de 25, 35 et 45 arêtes.

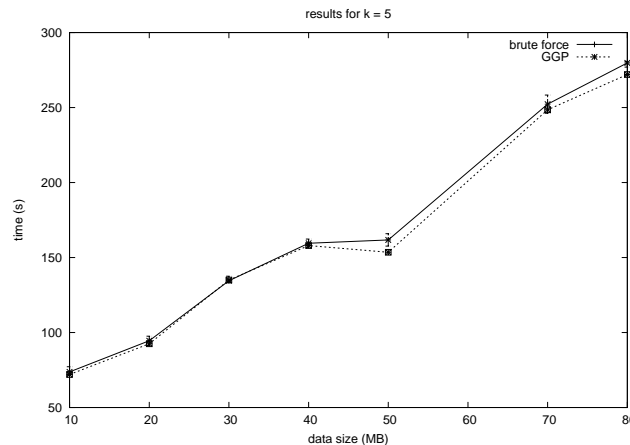


FIG. 4.31 – Temps obtenus sur des graphes de 25 arêtes pour une redistribution par force brute ou ordonnancée (basée sur la libc)

On remarque que la distribution ordonnancée fait preuve de meilleures performances que l'approche par force brute, avec des pics pouvant aller jusqu'à 7% d'amélioration. Les

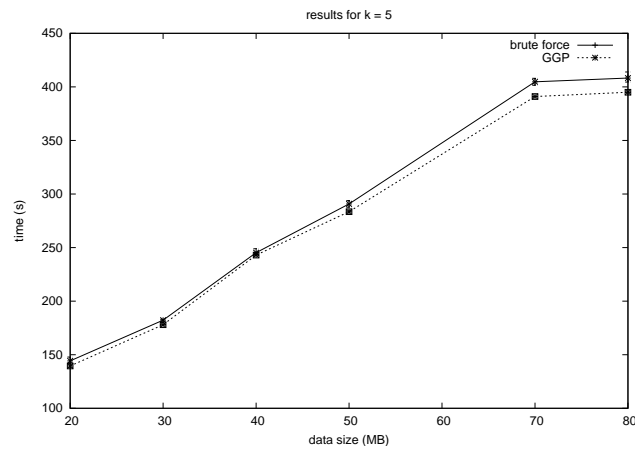


FIG. 4.32 – Temps obtenus sur des graphes de 35 arêtes pour une redistribution par force brute ou ordonnancée (basée sur la libc)

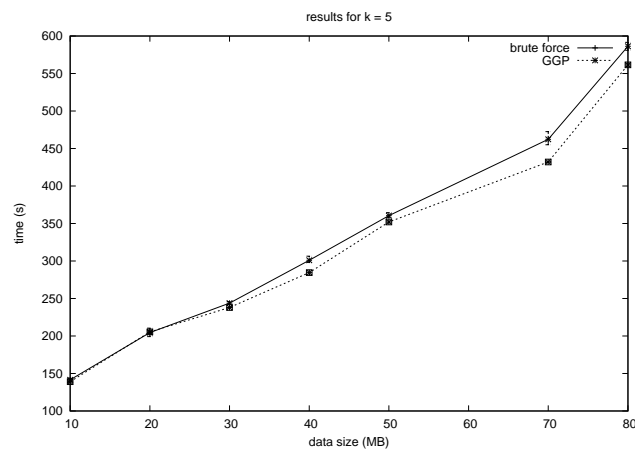


FIG. 4.33 – Temps obtenus sur des graphes de 45 arêtes pour une redistribution par force brute ou ordonnancée (basée sur la libc)

gains réalisés par l'approche ordonnancée sont par contre nettement moins importants que sur les expériences réalisées avec MPICH. On voit également que l'écart entre les deux approches se creuse lorsque le nombre d'arêtes augmente.

- Un second type de tests a été effectué sur un ensemble de graphes particuliers. Il s'agit du graphe de communication présenté section 3.1.2 et illustré par la figure 3.6. Le graphe est ici modifié pour s'adapter au facteur  $k = 5$ , mais le principe reste le même : d'un point de vue purement théorique, la redistribution par force brute fournit de moins bons résultats. Les temps obtenus, illustrés figure 4.34, confirment encore une fois la théorie. La redistribution ordonnancée est en moyenne 30% plus rapide que la redistribution par brute force. On remarque de plus que, le graphe étant déterministe, les temps de transferts augmentent linéairement avec les quantités de données à transférer.

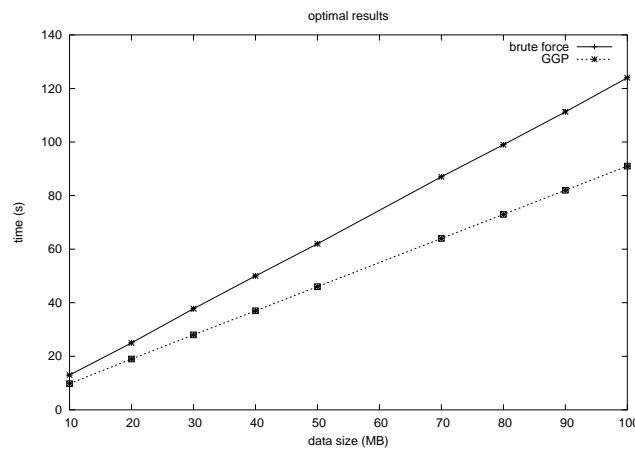


FIG. 4.34 – Temps de transferts sur des graphes générant de mauvaises performances pour une redistribution par force brute

### 4.3.5 Conclusion

L'ensemble de ces expériences montrent qu'il peut y avoir un avantage à utiliser GGP plutôt que d'exécuter une redistribution par force brute. De plus, nous avons vérifié que les temps obtenus sont relativement proches des temps prédits par la théorie.

Nous avons vu que l'utilisation d'une approche ordonnancée peut s'avérer utile lorsque l'intergiciel utilisé offre des primitives de communications asynchrones imposant un surcoût en temps. Une redistribution ordonnancée s'avère en effet plus simple à programmer et n'utilise que des primitives de communication synchrones.

Les simulations semblent confirmer l'idée que le ratio d'approximation de GGP pourrait être descendu à un facteur 2 et laissent ouverte la possibilité pour le ratio de GGP de descendre encore plus bas.

Nous pouvons également conclure que les différences de performance entre l'approche par force brute et l'approche ordonnancée sont liées essentiellement au motif de données à redistribuer plus qu'aux pertes liées à la congestion du réseau. Bien entendu, il est probable que les pertes liées directement au fonctionnement du réseau soient plus élevées sur un réseau d'une taille plus grande, particulièrement pour un nombre de routeurs entre la grappe source et la grappe destination plus élevé.

## Chapitre 5

# Extensions au problème KPBS

### Sommaire

---

<b>5.1</b>	<b>Redistribution de données sur un réseau hétérogène . . . . .</b>	<b>96</b>
5.1.1	Problème . . . . .	96
5.1.2	Algorithme d'approximation . . . . .	102
5.1.3	Propriétés de l'algorithme DGGP . . . . .	103
5.1.4	Expériences . . . . .	108
<b>5.2</b>	<b>Redistribution de données utilisant un réseau local . . . . .</b>	<b>111</b>
5.2.1	Problème du routage . . . . .	111
5.2.2	Régime permanent . . . . .	112
5.2.3	Algorithme . . . . .	113
5.2.4	Étude . . . . .	123

---

Nous avons vu au chapitre 3 la formalisation du problème KPBS à partir d'un problème concret de redistribution de données par étapes de communications.

Le modèle développé à partir de ce problème présuppose plusieurs propriétés sur la configuration physique des machines utilisées ainsi que sur le déroulement des communications. L'ensemble des hypothèses supposées vraies pour permettre l'utilisation du modèle limite donc la formalisation du problème de redistribution par le problème KPBS à un ensemble de configurations physiques les validant.

Afin de permettre une utilisation plus étendue du modèle de redistributions par étapes, nous avons étudié différentes variations du problème initial en supprimant ou affaiblissant différentes hypothèses afin d'étendre le nombre de configuration physiques différentes utilisables.

Nous présentons dans ce chapitre deux modifications par rapport au problème KPBS initial :

- section 5.1 nous présentons le problème obtenu en supprimant la contrainte du modèle 1-port limitant chaque nœud à une seule communication simultanée ;
- section 5.2 nous présentons le problème obtenu en complexifiant la topologie du réseau en ajoutant à chaque grappe un réseau local rapide.

Pour chaque cas, nous présentons en quoi l'hypothèse modifiée influe sur le problème KPBS et nous adaptons les algorithmes du chapitre 4 afin de fournir une solution performante à ces nouveaux problèmes.

## 5.1 Redistribution de données sur un réseau hétérogène

### 5.1.1 Problème

Nous proposons d'étudier ici le problème obtenu en partant du problème KPBS et en éliminant la contrainte du modèle 1-port.

Cette extension a pour but de permettre une redistribution efficace y-compris sur un réseau hétérogène. Enfin d'introduire et de justifier cette problématique, nous présentons tout d'abord plusieurs configurations physiques justifiant l'assouplissement du modèle.

#### Réseau hétérogène de grappes homogènes

On considère dans un premier temps une configuration réseau où la bande passante  $b_1$  du lien reliant chaque nœud de la grappe émettrice à son commutateur réseau est différente de  $b_2$ , celle de la grappe réceptrice. La figure 5.1, déjà rencontrée au chapitre 3 illustre sur un exemple simpliste une telle configuration,  $b_1$  ayant ici une valeur de 100Mbit/s et  $b_2$  une valeur de 1Gbit/s.

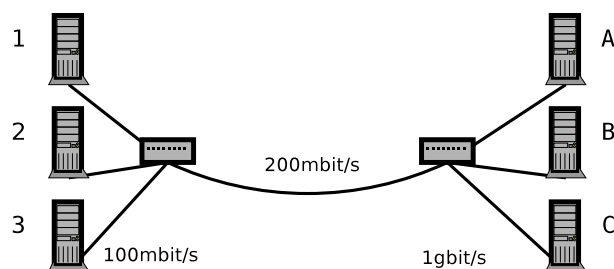


FIG. 5.1 – Configuration réseau hétérogène de grappes homogènes

Il est clair que toute communication ne peut disposer ici d'une bande passante supérieure à 100Mbit/s, les interfaces émettrices formant ici le goulot d'étranglement. On peut donc remarquer que les interfaces réseau de la grappe réceptrice sont donc capables de recevoir  $\frac{1000\text{Mbit/s}}{100\text{Mbit/s}} = 10$  communications simultanément. Le modèle 1-port nous impose donc ici une restriction inutile. Bien entendu l'utilisation du modèle 1-port pour ce problème peut s'avérer un choix judicieux, notamment du fait que la programmation d'un programme acceptant plusieurs communications simultanément s'avère plus complexe que celle d'un programme n'acceptant qu'une communication à chaque étape. On peut néanmoins trouver certains motifs de redistribution pour lesquels autoriser plusieurs communications simultanées sur certaines interfaces présente un intérêt fort. Si l'on considère le graphe de communication de la figure 5.2 par exemple, on peut voir qu'une redistribution permettant trois communication simultanées s'avère ici 3 fois plus rapide à l'exécution qu'une redistribution classique.

#### Réseau de grappes hétérogènes

Nous considérons ici également la possibilité d'un réseau de grappes hétérogènes où toute interface réseau peut disposer d'une bande passante quelconque. La figure 5.3 illustre une telle configuration. Bien entendu l'exemple de cette figure est une configuration irréaliste mais elle présente néanmoins deux intérêts. Tout d'abord, il n'est pas tellement rare de trouver des

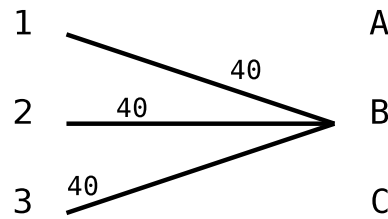


FIG. 5.2 – Graphe de communication justifiant plusieurs communications simultanées sur un réseau hétérogène de grappes homogènes

grappes d'ordinateurs disposant de deux types d'interfaces différentes (100Mbit/s et 1Gbit/s par exemple). Ce cas se présente souvent dans le cas de grappes mises en place au fil du temps et donc naturellement plus hétérogènes que des grappes où toutes les machines ont été achetées à la même date. Nous verrons par la suite qu'une telle configuration présente plus de ressemblances avec la configuration jouet de la figure 5.3 qu'avec le réseau hétérogène de grappes homogènes de la figure 5.1. Enfin une deuxième raison d'étudier ce type de configuration vient du fait qu'elle pose beaucoup plus de difficultés pour le calcul d'un ordonnancement. En effet, il est par exemple difficile d'utiliser directement une variable  $k$  limitant le nombre total de communications simultanées, chaque communication s'effectuant à une vitesse propre.

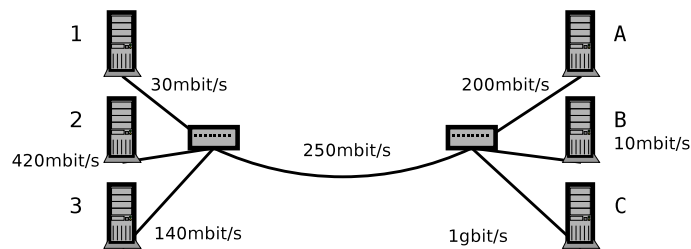


FIG. 5.3 – Configuration réseau de grappes hétérogènes

### Autre configuration

Une autre configuration peut également justifier le choix d'assouplir la contrainte du modèle 1-port :

sur certaines configurations, il est possible de trouver plusieurs interfaces réseau sur chaque machine (souvent sur des machines multiprocesseurs). Il est alors possible d'effectuer simultanément une communication sur chaque interface ;

### Qualité de service

De tous les exemples de configurations présentés on distingue deux cas différents :

- un cas pour lequel toutes les communications sont effectuées à la même vitesse et donc il est facile de calculer une limite  $k$  au nombre de communications simultanées autorisées ;
- un cas où chaque communication est effectuée à une vitesse différente (réseau de grappes hétérogènes), et où il est donc impossible de fixer une valeur à  $k$ .

En réalité il est, dans le cas de grappes hétérogènes, très difficile de trouver un ordonnancement des communications qui évite toute saturation du réseau, tout en étant efficace. Pour gaspiller le moins possible de bande passante sur ce type de configuration, il est nécessaire de recourir à une gestion plus souple de la bande passante de chaque interface à l'aide de qualité de service.

Prenons par exemple le réseau de la figure 5.4 sur lequel sont à effectuer les communications du graphe de la figure 5.5. Sans utiliser de qualité de service, il est ici au mieux possible d'obtenir l'ordonnancement en trois étapes de la figure 5.6, pour un coût total de  $2 + 3\beta$  secondes. En limitant à 500Mbit/s la bande passante de l'interface réseau du nœud 1 il est en revanche possible d'obtenir l'ordonnancement de la figure 5.7, pour un coût total de  $1 + 2\beta$  secondes où toute la bande passante de la dorsale est utilisée. Remarquons au passage qu'on se retrouve alors avec une instance classique du problème KPBS associée à une valeur de  $k$  de 2.

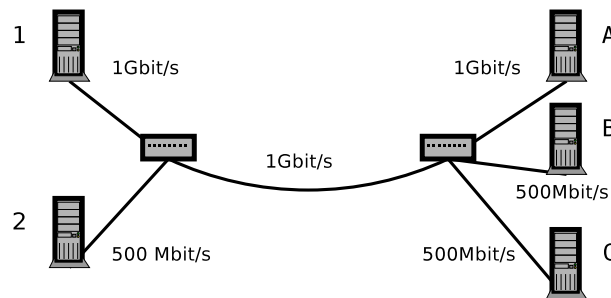


FIG. 5.4 – Réseau de grappes hétérogènes avant utilisation de qualité de service

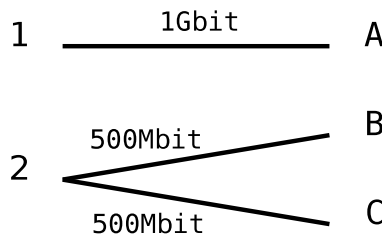


FIG. 5.5 – Graphe de communication (les poids sont des quantités de données)

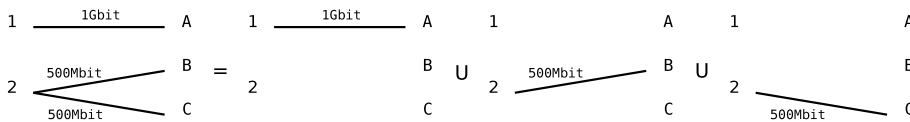


FIG. 5.6 – Meilleure solution possible sans qualité de service et sans saturation

Nous allons utiliser la qualité de service pour nous ramener dans le cas d'un réseau de grappes hétérogènes à un problème plus simple. Le principe est le suivant : on calcule une bande passante  $b_r$  de base qui va servir à calculer aussi bien  $k$  que le nombre de communications simultanées

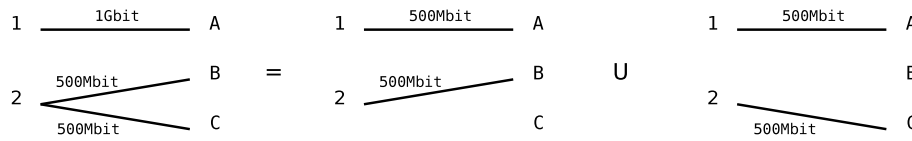


FIG. 5.7 – Meilleure solution avec qualité de service

autorisées pour chaque nœud. Idéalement  $b_r$  a comme valeur le plus grand commun diviseur entre toutes les bandes passantes de chaque interface. Dans le cas pratique, on évitera néanmoins de prendre une valeur de  $b_r$  trop petite, quitte à perdre un peu d'efficacité.

Une fois  $b_r$  calculée, il est facile de voir que désormais, chaque nœud  $s$  de la grappe admet sur son interface locale  $\frac{b_s}{b_r}$  communications simultanées, où  $b_s$  est la bande passante de l'interface. De plus, cette valeur est entière car  $b_r$  est, par construction, un diviseur de  $b_s$ . De même, on peut calculer  $k$ , le nombre total de communications autorisées simultanément comme  $k = \frac{b_b}{b_r}$ . Bien entendu, ici, "communication" est entendu ici dans le sens "communication effectuée à une vitesse  $b_r$ ".

Afin d'illustrer cette procédure, on considère le réseau de la figure 5.8 accompagné d'une matrice de communication contenant la quantité de données à transmettre lors d'une redistribution.

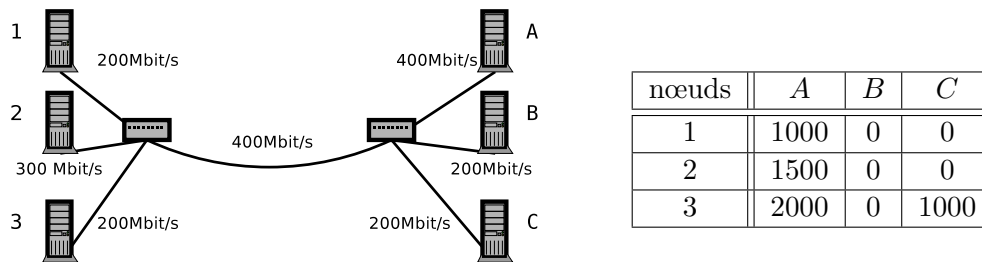


FIG. 5.8 – Réseau de grappes hétérogènes, matrice de communication (en Mbit)

On commence par calculer la valeur de la bande passante de base :  $b_r = \text{pgcd}(200, 300, 400) = 100\text{Mbit/s}$ . Sachant désormais que toute les communications seront effectuées à  $100\text{Mbit/s}$ , il est possible de calculer  $k = \frac{b_b}{b_r} = \frac{400}{100} = 4$  communications simultanées. Le nœud 1 se voit autorisé  $\frac{200}{100} = 2$  communications simultanées, les nœuds 2, 3, A, B et C respectivement 3, 2, 4, 2 et 2 communications simultanées. On divise alors les quantités de données de la matrice de communication par  $b_r$  pour obtenir la matrice de communication normalisée, puis le graphe de communication de la figure 5.9.

D'une manière similaire à la résolution du problème KPBS, on va chercher à décomposer ce graphe en une série d'étapes de communications, chacune représentée par un ensemble d'arêtes valuées. Dans le cas présent, néanmoins, les ensembles d'arêtes obtenus ne seront pas des couplages, et ce pour deux raisons :

- tout d'abord, la même arête peut être contenue plusieurs fois à l'intérieur d'un même ensemble (en réalité un multi-ensemble donc) ;
- de plus, le nombre d'arêtes incidentes à un même sommet peut désormais être différent de 1.

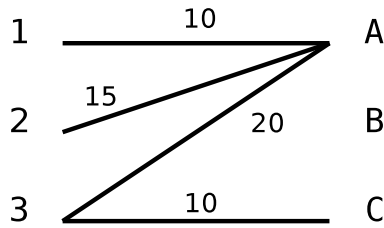


FIG. 5.9 – Graphe de communication après calcul de la bande passante de base

Nous prenons la convention de nommer les multi-ensembles d’arêtes recherchés  $\delta$ -ensembles. Notons qu’il serait sans doute possible d’utiliser ici des b-couplages [14], dans le cas où la fonction  $\delta$  serait constante.

Afin de définir formellement un  $\delta$ -ensemble, il nous est en premier lieu nécessaire d’introduire une nouvelle notation. Pour tout graphe de communication  $G = (V_1, V_2, E, w)$ , nous notons  $\delta : V_1 \cup V_2 \rightarrow \mathbb{N}$  la fonction qui à chaque sommet de  $G$  associe le nombre de communications autorisées sur ce sommet. Nous avons ainsi sur l’exemple précédent  $\delta(A) = 4$ , le sommet  $A$  disposant d’une bande passante de 400Mbit/s, 4 fois supérieure à la bande passante de base de 100Mbit/s. Notons au passage que pour tout sommet  $s$  du graphe  $\delta(s) \leq k$ . En effet, il ne sert à rien d’autoriser plus de communication qu’il n’est possible d’en réaliser sur la dorsale.

**Définition 2** Un  $\delta$ -ensemble valide sur un graphe  $G = (V_1, V_2, E, w, \delta)$  est défini comme un multi-ensemble d’arêtes  $C \subset E^*$  tel que :

$$\forall s \in V_1, \sum_{v \in V_2} occ_C(s, v) \leq \delta(s)$$

$$\forall s \in V_2, \sum_{v \in V_1} occ_C(v, s) \leq \delta(s)$$

où la fonction occurrence  $occ_C : E \rightarrow \mathbb{N}$  retourne le nombre de fois qu’une arête  $e$  est présente dans le multi-ensemble  $C$ .

Avec cette définition, nous pouvons obtenir pour notre exemple la solution de la figure 5.10, composée de 4  $\delta$ -ensembles. Notons que la manière d’exécuter la redistribution est différente lorsqu’il y a utilisation de qualité de service. Pour tout ordonnancement obtenu, si deux communications identiques sont à exécuter simultanément alors en pratique, on exécute une seule communication mais avec une bande passante deux fois plus grande. Ainsi, lors des deux premières étapes de redistribution deux flux à 200Mbit/s sont émis et lors de la troisième un flux à  $3 \times 100 = 300$ Mbit/s, pour un temps total de  $5 + 5 + 5 + 3\beta$ .

### Le problème d-KPBS

Le problème d-KPBS suit le même principe que le problème KPBS dont il dérive son nom : on cherche à décomposer un graphe de communication en un ensemble d’ensembles d’arêtes de coût minimal. Ici le graphe  $G$  à décomposer est lié à la fonction  $\delta$  indiquant le nombre maximal de communications autorisé pour chaque sommet et l’on n’obtient pas de couplages, mais des  $\delta$ -ensembles.

Avec cette fonction, on peut définir le problème d-KPBS qui nous intéresse :

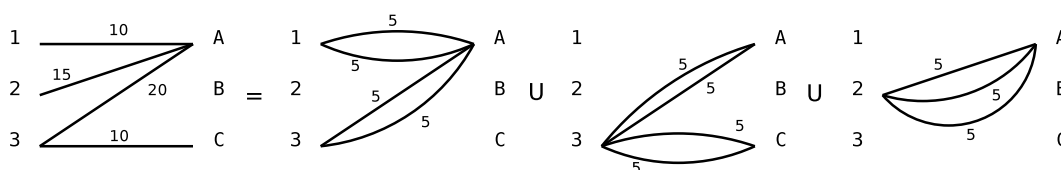


FIG. 5.10 – Ordonnancement solution, plusieurs communications simultanées par nœud

**Définition 3** Le problème  $d$ -KPBS est défini comme suit :

Entrée :  $G = (V_1, V_2, E, w, \delta)$  un graphe de communication,  $\beta$  la latence d'une communication,  $k$  le nombre maximal de communications autorisées simultanément ;

Sortie :  $D = \{M_1 = (E_1, w_1), \dots, M_h = (E_h, w_h)\}$  où chaque  $E_i$  est un multi-ensemble d'arêtes :  $\forall i \in \{1, \dots, h\}, E_i \subset E^*$  ;  $w_i$  est une fonction de valuation associant un poids à chaque arête de  $E_i$  :  $w_i : E_i \rightarrow \mathbb{Q}$ .

L'ensemble  $D$  valide les contraintes suivantes :

- chaque ensemble d'arêtes  $M_i$  est un  $\delta$ -ensemble valide ;
- de manière similaire à une décomposition pour KPBS, les poids des arêtes du graphe sont conservés (voir section 3.2.2) ;
- le nombre d'arêtes de chaque couplage est d'au plus  $k$  :  $\forall l \in 1, \dots, h, |M_l| \leq k$ .

Objectif : On cherche à minimiser le temps total pris par l'exécution de la décomposition valant :

$$c(D) = \sum_{l=1}^h (\max_{e \in E_l} (w_l(e))) + h \times \beta$$

### Bornes inférieures sur le temps de redistribution

En examinant l'exemple de la section précédente, on remarque que la borne inférieure  $\eta$  sur le temps de redistribution d'un graphe de communication développée pour le problème KPBS n'est plus valide pour le problème étendu. En effet, on a  $\eta(G) = \eta_d(G) + \eta_s(G) \times \beta$ . En prenant pour  $\beta$  une valeur de 1 on obtient donc ici  $\eta(G) = \max\left(\frac{P(G)}{k}, W(G)\right) + \max\left(\left\lceil \frac{m(G)}{k} \right\rceil, \Delta(G)\right) = \max\left(\frac{55}{4}, 45\right) + \max\left(\left\lceil \frac{4}{4} \right\rceil, 3\right) = 45 + 3 = 48$ . Or la solution obtenue figure 5.10 a un coût total de 18 secondes, ce qui est inférieur à  $\eta(G)$ , par conséquent  $\eta(G)$  n'est pas une borne inférieure sur le temps de redistribution pour le problème KPBS hétérogène.

Nous prenons comme convention de noter  $d : V_1 \cup V_2 \rightarrow \mathbb{N}$  la fonction associant à chaque sommet  $s$  de  $G$  son degré  $d(s)$ .

On peut voir que les communications du sommet  $A$  influent fortement sur le calcul des bornes inférieures. En effet,  $W(G) = \max_{s \in V_1 \cup V_2} (p(s)) = p(A) = 45$  et  $\Delta(G) = d(A) = 3$ . Néanmoins, en prenant en compte le fait que le sommet  $A$  est capable de recevoir 4 communications simultanément, il est clair que le nombre d'étapes nécessaires pour ce sommet n'est que de  $\left\lceil \frac{d(A)}{\delta(A)} \right\rceil = \left\lceil \frac{3}{4} \right\rceil = 1$  étape. De même le temps nécessaire pour transmettre toutes les données de ce sommet n'est que de  $\frac{p(A)}{\delta(A)} = \frac{45}{4}$ . Nous définissons donc une nouvelle borne inférieure  $\eta'(G)$ , de la manière suivante :  $\eta'(G) = \eta'_d(G) + \eta'_s(G) \times \beta$  où  $\eta'_d(G)$  est le temps minimal nécessaire pour

transmettre toutes les données et  $\eta'_s(G)$  le nombre minimal d'étapes. La prise en compte de la fonction  $\delta$  sur le nombre de communications autorisées nous amène aux formules suivantes :

$$\eta'_d(G) = \max \left( \max_{s \in V_1 \cup V_2} \left( \frac{p(s)}{\delta(s)} \right), \frac{P(G)}{k} \right) \quad (5.1)$$

$$\eta'_s(G) = \max \left( \max_{s \in V_1 \cup V_2} \left( \left\lceil \frac{d(s)}{\delta(s)} \right\rceil \right), \left\lceil \frac{m(G)}{k} \right\rceil \right) \quad (5.2)$$

Remarquons au passage que si  $\forall s \in V_1 \cup V_2, \delta(s) = 1$  alors :

$$\max_{s \in V_1 \cup V_2} \left( \frac{p(s)}{\delta(s)} \right) = \max_{s \in V_1 \cup V_2} (p(s)) = W(G)$$

et

$$\max_{s \in V_1 \cup V_2} \left( \left\lceil \frac{d(s)}{\delta(s)} \right\rceil \right) = \max_{s \in V_1 \cup V_2} (d(s)) = \Delta(G)$$

et par conséquent  $\eta'_d(G) = \eta_d(G)$  et  $\eta'_s(G) = \eta_s(G)$ .

Une dernière remarque concerne la possibilité d'utiliser un calcul rapide des nouvelles bornes inférieures afin de déterminer quel algorithme utiliser. En effet si  $\eta' \approx \eta$  alors il est clair que DGGP ne peut offrir qu'un gain de temps modéré par rapport à GGP ou OGGP et qu'il convient alors plutôt de lui préférer un de ces deux algorithmes.

### 5.1.2 Algorithme d'approximation

Nous proposons, pour résoudre le problème d-KPBS un algorithme DGGP (*Delta GGP*) développé à partir de OGGP.

L'algorithme fonctionne en ajoutant deux étapes supplémentaires à GGP, tel que décrit formellement sur la figure 3 :

- une première étape de création de sommets virtuels située entre l'étape 1 et l'étape 2 de OGGP ;
- une seconde étape d'élimination de ces sommets située après la terminaison de OGGP.

Le principe de DGGP est le suivant : chaque sommet  $s$  du graphe initial est capable d'effectuer  $\delta(s)$  communications simultanées. DGGP fonctionne en remplaçant chaque sommet  $s$  par  $\delta(s)$  sommets virtuels  $s_1, \dots, s_{\delta(s)}$  et répartit le plus équitablement possible les arêtes incidentes à  $s$  entre chacun de ces nouveaux sommets. Cette répartition est effectuée après l'étape d'arrondi des poids, car nous conservons la propriété que toutes les arêtes manipulées par OGGP ont des poids entiers. A la fin de OGGP, les sommets virtuels sont à nouveau fusionnés, transformant ainsi les couplages obtenus par OGGP en  $\delta$ -ensembles, solution de d-KPBS.

La figure 8 montre l'algorithme formel utilisé pour créer les sommets virtuels. L'algorithme fonctionne en créant en premier lieu tous les sommets virtuels de  $V_1$ , puis, dans un second temps, ceux de  $V_2$ . Nous nommons ici  $G' = (V'_1, V'_2, E', w')$  le graphe intermédiaire et  $G'' = (V''_1, V''_2, E'', w'')$  le graphe final obtenu. Pour chaque sommet  $s$  divisé en  $\delta(s)$  sommets, on calcule  $p(s)$  et on répartit à chaque sommet virtuel des arêtes pour un poids de  $\frac{p(s)}{\delta(s)}$  si  $\frac{p(s)}{\delta(s)} \in \mathbb{N}$  et  $\left\lceil \frac{p(s)}{\delta(s)} \right\rceil$  ou  $\left\lfloor \frac{p(s)}{\delta(s)} \right\rfloor$  (voir la description formelle de l'algorithme pour la répartition exacte de ces poids) sinon.

Reprenons l'exemple du graphe de communication de la figure 5.9. Le découpage des sommets se déroule en deux étapes, comme illustré figure 5.11. Dans un premier temps tous les sommets de  $V_1$  sont découpés équitablement car pour chaque sommet  $s$ ,  $\delta(s)$  divise  $p(s)$ . A la seconde

**Entrées :** Un graphe  $G = (V_1, V_2, E, w, \delta)$ , un entier  $k$ .

**Sorties :** Un graphe  $G'' = (V_1'', V_2'', E'', w'')$  obtenu en partageant équitablement chaque sommet  $s$  de  $G$  en  $\delta(s)$  sommets.

$V_2' = V_2$ ;

**pour chaque**  $s \in V_1$  **faire**

calculer  $p(s) = \sum_{(s, s_2) \in E} w((s, s_2))$ ;

construire  $(l_i)_{0 \leq i < \delta(s)}$  la liste représentant la part de toutes les communications de  $s$  donnée à chaque sommet virtuel :  $\forall i \in \{1, \dots, p(s) \text{ modulo } \delta(s)\}, l_i = \left\lceil \frac{p(s)}{\delta(s)} \right\rceil$  ;

$\forall i \in \{(p(s) \text{ modulo } \delta(s)) + 1, \dots, \delta(s)\}, l_i = \left\lfloor \frac{p(s)}{\delta(s)} \right\rfloor$ . Bien entendu,  $\sum_i l_i = p(s)$  ;

ajouter à  $V_1'$  les sommets obtenus en découpant  $s : s_{i0 \leq i < \delta(s)}$  ;

découper les arêtes **pour chaque**  $i \in \{0, \dots, \delta(s) - 1\}$  **faire**

**tant que**  $l_i \neq 0$  **faire**

prendre une arête  $a = (s, a_2) \in E$ , telle que  $a$  soit de poids maximal;

**si**  $w(a) \leq l_i$  **alors**  $l_i = l_i - w(a)$ , enlever  $a$  de  $E$ , ajouter  $(s_i, a_2)$  de poids  $w(a)$  à  $E'$ ;

**sinon**  $w(a) = w(a) - l_i$ , ajouter  $(s_i, a_2)$  à  $E'$  avec  $w'((s_i, a_2)) = l_i, l_i = 0$

**fin**

**fin**

**fin**

effectuer les mêmes opérations pour découper les sommets de  $V_2$  pour former  $G''$ ;

**Algorithme 8 :** Algorithme de partage des sommets

étape, on remarque que  $A$  a été partagé en 4 sommets  $A, A', A'', A'''$  et que  $p(A) = 12, p(A') = p(A'') = p(A''') = 11$ . On vérifie que l'on a bien  $p(A) + p(A') + p(A'') + p(A''') = 45$ , le poids de  $A$  sur le graphe de communication initial.

Une fois les sommets partagés, OGGP est utilisé pour fournir un ensemble de couplages solutions. Les sommets virtuels sont alors re-fusionnés ensemble sur le sommet réel leur correspondant.

### 5.1.3 Propriétés de l'algorithme DGGP

L'étude de DGGP est divisée en 3 parties : une étude de complexité au pire cas, une preuve d'un ratio d'approximation de 4, ainsi qu'une étude de quelques exemples sur lesquels DGGP fournit de mauvais résultats.

#### Complexité

**Proposition 11** *DGGP a une complexité au pire cas de  $\sqrt{nk}(m + kn)^3$ .*

**Preuve :** On rappelle que  $\forall n \in V_1 \cup V_2, \delta(n) \leq k$ .

Pour le découpage des arêtes, l'algorithme itère sur chaque sommet  $s$ . Pour chacun d'entre eux,  $\delta(s)$  nouveaux sommets sont créés. La création d'un nouveau sommet se fait en parcourant l'ensemble des arêtes en passant au plus une fois par arête. Par conséquent, le partage des arêtes est effectué au plus en  $O(n \times k \times m)$  car  $k$  est supérieur à toute valeur de  $\delta(s)$ . La boucle principale de OGGP générant les couplages solutions étant d'une complexité plus élevée, c'est encore elle qui va déterminer la complexité de l'ensemble de l'algorithme.

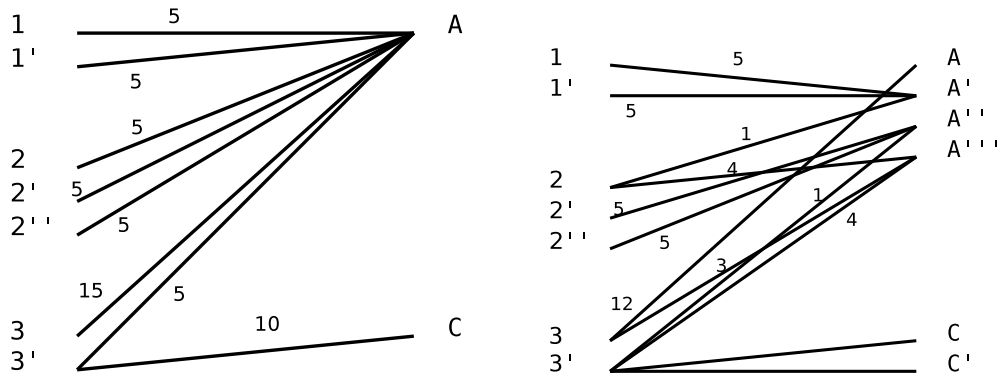


FIG. 5.11 – Partage des sommets du graphe de communication de la figure 5.9

On peut voir que le découpage d’une arête en deux arêtes distinctes signifie que la variable  $l_i$  lui correspondant est mise à zéro. Par conséquent, le partage du sommet  $s$  génère au plus  $\delta(s)$  découpages d’arêtes. Ceci signifie que  $|E'| \leq |E| + k \times (|V_1| + |V_2|)$  par conséquent  $m' \leq m + kn$ . Enfin, il est clair que  $n' \leq kn$ . OGGP étant en  $O(\sqrt{n}(m+n)^3)$  on conclut directement que la complexité au pire cas de DGGP est de l’ordre de  $O(\sqrt{kn}(m+kn+kn)^3) = O(\sqrt{kn}(m+kn)^3)$ . ■

### Ratio d’approximation

L’étude d’un ratio d’approximation pour DGGP se fait de manière similaire à celle réalisée pour GGP, et est donc relativement complexe.

Le principe est identique à celui de GGP : nous introduisons un nouvel algorithme, remplaçant le graphe décomposé dans DGGP par un multigraphe. Nous montrons alors que cet algorithme a un ratio d’approximation de 4 et que, pour une instance donnée du problème d-KPBS, tout ordonnancement obtenu par DGGP est plus rapide qu’un ordonnancement obtenu par l’algorithme par multigraphe.

L’algorithme par multigraphe, que nous appellerons ici M2, est décrit formellement figure 9. Cet algorithme fonctionne en utilisant plusieurs parties des algorithmes développés section 4.1. Notons que les noms graphes employés sont identiques à ceux des graphes intermédiaires de l’algorithme GGP lorsque les deux graphes ont été construits de la même manière.

**Entrées** : un graphe biparti  $G = (V_1, V_2, E, w, \delta)$  et un entier  $k$ .

**Sorties** : un ensemble  $S'$  de couplages valués.

construire le graphe  $H$  en réalisant la première étape de GGP;

construire le graphe  $U$  en partageant les arêtes de  $H$  à l’aide de l’algorithme de la figure 8;

construire à partir de  $U$  les graphes  $I$  et  $J$  en utilisant les étapes 2 et 3 de GGP ;

construire  $J' = trans(J)$  le multigraphe obtenu en transformant toute arête de poids  $w$  en  $w$  arêtes de poids unitaire ;

décomposer  $J'$  dans l’ensemble de couplages  $S'$  en utilisant l’étape 4 de l’algorithme de la figure 4.

#### Algorithme 9 : Algorithme M2

**Theorème 4** *L'algorithme M2 est un algorithme d'approximation d'un ratio d'approximation de 4.*

**Preuve :** La décomposition de  $J'$  en l'ensemble de couplage  $S'$  étant la même que pour l'algorithme de la figure 4, ainsi que la construction des graphes  $I$  et  $J$ , nous utilisons la preuve du théorème 2 pour en déduire :

$$c(S') = 2\eta_d(I)$$

L'algorithme de construction de  $I$  impliquant  $W(I) = W(U)$  et  $\frac{P(I)}{k} = \max\left(\left\lceil \frac{P(U)}{k} \right\rceil, W(U)\right)$  on en déduit donc :

$$c(S') = 2\max\left(\left\lceil \frac{P(U)}{k} \right\rceil, W(U)\right)$$

L'algorithme de partage des sommets construisant le graphe  $U$  à partir du graphe  $H$  implique pour tout sommet  $s$  :

$$p_U(s) \leq \left\lceil \frac{p_H(s)}{\delta(s)} \right\rceil$$

De plus, la construction de  $U$  conservant la somme des poids, on sait que  $P(H) = P(U)$  Nous avons donc par conséquent :

$$c(S') = 2\max\left(\left\lceil \frac{P(H)}{k} \right\rceil, \max_{s \in V_{1_H} \cup V_{2_H}} \left(\left\lceil \frac{w(s)}{\delta(s)} \right\rceil\right)\right)$$

On effectue la même opération que dans la preuve du théorème 2 pour en déduire :

$$c(S') \leq 2\max\left(\left\lceil \frac{P(G) + m(G)}{k} \right\rceil, \max_{s \in V_{1_H} \cup V_{2_H}} \left(\left\lceil \frac{w(s) + d(s)}{\delta(s)} \right\rceil\right)\right)$$

$$c(S') \leq 2\left(\max\left(\left\lceil \frac{P(K)}{k} \right\rceil, \max_{s \in V_{1 \cup V_2}} \left(\left\lceil \frac{w(s)}{\delta(s)} \right\rceil\right)\right) + \max\left(\left\lceil \frac{m(G)}{k} \right\rceil, \max_{s \in V_{1 \cup V_2}} \left(\left\lceil \frac{d(s)}{\delta(s)} \right\rceil\right)\right)\right)$$

$$\Rightarrow c(S') \leq 2(\eta'_d(G) + \eta'_s(G) + 1)$$

Sachant  $\eta_s \geq 1$ , toute solution nécessitant au minimum une étape de redistribution, on en déduit :

$$\frac{c(S')}{\eta'(G)} \leq 2 + \frac{2}{\eta'(G)} \leq 4$$

■

**Theorème 5** *L'algorithme DGGP est un algorithme d'approximation d'un ratio d'approximation de 4.*

**Preuve :** La preuve est identique à celle du théorème 3.

■

Cas problématiques

Le ratio d'approximation de 4 obtenu, plus élevé que celui de GGP, nous a amené à rechercher différents jeux d'exemples susceptibles de s'approcher de cette borne, le but de cette approche étant de déterminer la possibilité ou non de réduire la valeur du ratio d'approximation.

Nous présentons ici, une famille d'exemple par laquelle il est possible de s'approcher d'un ratio temps d'exécution / temps optimal de 4 lorsque l'on considère encore une fois les solutions intermédiaires (voir pour plus de détails la section 4.1.3).

Considérons le graphe de communication de la figure 5.12 pour une valeur de  $k$  égale à 3, et une valeur de  $\delta$  égale à 3 pour chaque sommet.

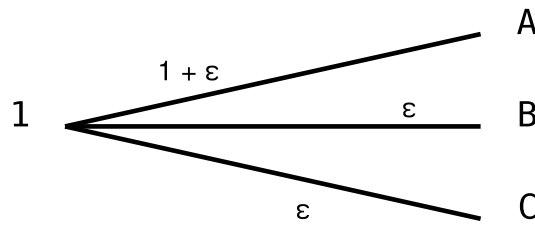


FIG. 5.12 – Graphe de communication problématique pour l'algorithme DGGP

L'exécution de DGGP construit dans un premier temps le graphe  $H$  obtenu en arrondissant les poids de chaque arête, puis le graphe  $U$  obtenu en effectuant le partage des arêtes. Ces deux graphes intermédiaires sont représentés ici figure 5.13.

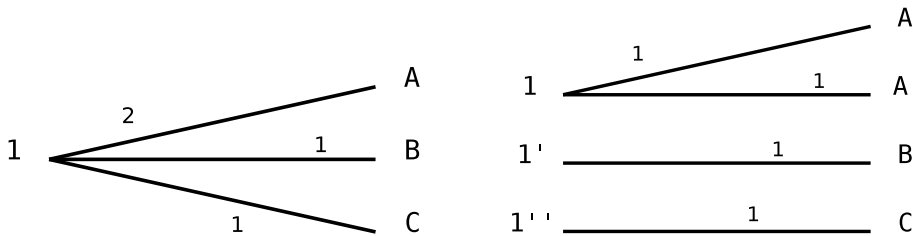


FIG. 5.13 – Graphes intermédiaires  $H$  et  $U$  pour le calcul de DGGP

L'algorithme DGGP fournit donc la solution intermédiaire  $S'$  représentée sur la partie gauche de la figure 5.14 accompagnée sur la partie droite de la figure par la solution finale.

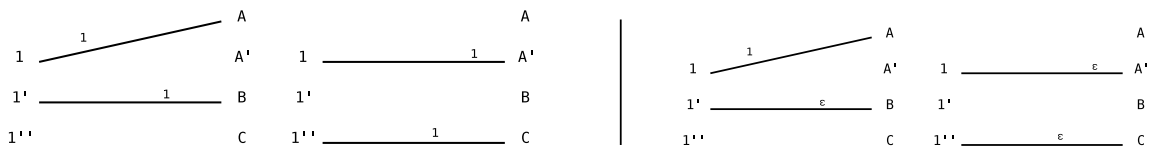


FIG. 5.14 – Solutions intermédiaire et finale pour le graphe de communication de la figure 5.12

Le coût de la solution intermédiaire est ici de 4, soit 2 étapes de coût unitaire. La borne

inférieure  $\eta'$  sur le temps de redistribution a, elle, pour valeur

$$\eta' = \max\left(\frac{1+3\epsilon}{k}, \frac{1+3\epsilon}{\delta(1)}\right) + \max\left(\left\lceil \frac{3}{k} \right\rceil, \left\lceil \frac{3}{\delta(1)} \right\rceil\right)$$

$$\Rightarrow \eta' \cong \frac{1}{3} + 1 = \frac{4}{3}$$

Le ratio temps d'exécution / temps optimal a donc ici une valeur de  $\frac{4}{\frac{4}{3}} = 3$ . Notons que même la solution finale fait preuve d'un mauvais ratio temps d'exécution / temps optimal avec une valeur de  $\frac{9}{4}$ .

Nous construisons à partir de cet exemple toute une famille de communications en augmentant à chaque fois les valeurs de  $k$  et de  $\delta$ . La figure 5.15 illustre les deux membres suivant de cette famille, pour les valeurs de  $k$  de 3 et 4.

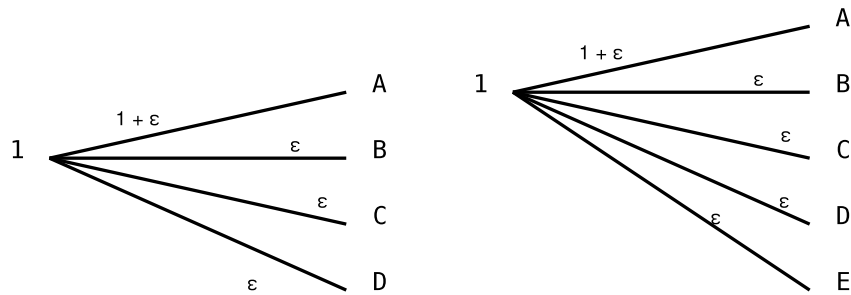


FIG. 5.15 – Exemples problématiques pour  $k = 3$  et  $k = 4$

Dans chaque cas, le temps d'exécution de la solution intermédiaire a une valeur de 4. En revanche, la borne inférieure décroît lorsque  $k$  et  $\delta$  augmentent, sa valeur étant de  $1 + \frac{1}{k}$ . Ceci nous amène à un ratio temps d'exécution / temps optimal d'une valeur de  $\frac{4}{1+\frac{1}{k}}$ , ce qui tend vers 4 lorsque  $k$  tend vers l'infini. Notons au passage que la solution finale voit, elle, son ratio tendre vers une valeur de 3.

Avant de tirer des conclusions des résultats obtenus, il est important de remarquer que la borne inférieure sur le temps d'exécution n'est pas atteignable. En effet, soit l'on scinde les arêtes pour atteindre  $\eta'_d$ , et le nombre d'étapes augmente alors, soit l'on cherche à atteindre  $\eta'_s$  et il n'est alors tout de même pas possible d'atteindre  $\eta'$ . Le meilleur résultat que l'on puisse obtenir sur cet exemple étant réalisé en une étape, en exécutant toutes les communications pour un temps de  $2 + \epsilon$ . Si l'on compare les résultats obtenus par DGGP à ce temps "optimal" on ne dépasse jamais un facteur de 2.

Nous concluons de tout ceci que pour la borne inférieure sur les temps de communication  $\eta'$  choisie, DGGP ne peut pas dans tous les cas fournir des résultats pour un temps inférieur à  $3\eta'$ . De plus, d'une manière similaires aux résultats obtenus section 4.1.3, il n'est pas possible de prouver que l'on peut borner ces temps par un ratio inférieur à 4 sans prendre en compte la phase finale de modification des poids de l'algorithme. Il reste toujours possible de trouver pour DGGP un ratio d'approximation inférieur à 3, mais ceci nécessiterai d'affiner la borne inférieure  $\eta'$  en choisissant différentes bornes en fonction de différents graphes de communication, ce que nous considérons comme un problème particulièrement difficile.

### 5.1.4 Expériences

Afin d'étudier le comportement de DGGP sous différentes situations nous avons effectué des simulations similaires à celles effectuées pour GGP et OGGP section 4.3.2. Ces simulations consistent en l'exécution de DGGP ou GGP sur des graphes d'entrée aléatoires, puis en un calcul de la borne inférieure  $\eta'$  sur les temps de communications ainsi qu'un calcul pour chaque algorithme d'un ratio d'évaluation défini encore une fois comme le temps de l'exécution de l'ordonnancement obtenu pour un algorithme divisé par  $\eta'$ .

Un ratio d'évaluation d'une valeur de 1 signifie donc un ordonnancement optimal.

Pour chaque graphe généré, la valeur de  $\delta$  pour chaque sommet est générée aléatoirement entre 1 et 5.

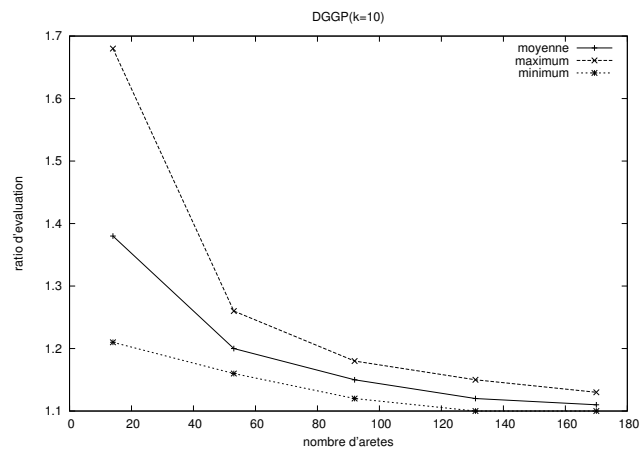


FIG. 5.16 – Évaluation de DGGP ( $k = 10$ )

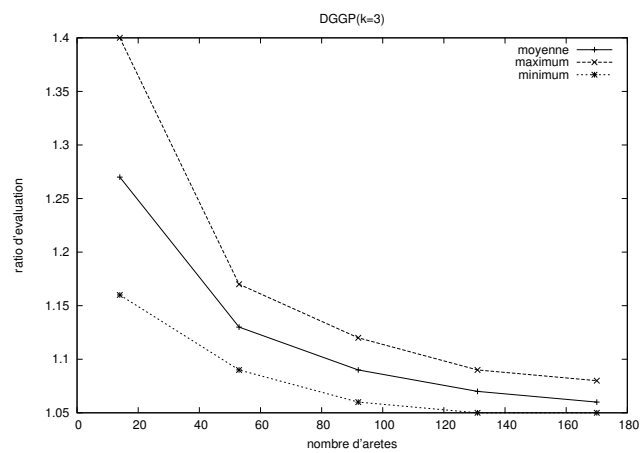


FIG. 5.17 – Évaluation de DGGP ( $k = 3$ )

Nous présentons plusieurs graphes détaillant les résultats obtenus. Chaque point du graphe représente la moyenne des ratios obtenus sur une centaine de graphes aléatoires. Le nombre de paramètres de l'algorithme étant particulièrement élevé (nombre de sommets, nombre d'arêtes,

génération des poids, ...), pour chaque graphe, presque tous les paramètres sont fixés, afin d'obtenir un graphe en 2 dimensions. Les résultats sont de nature semblable en faisant varier différents paramètres.

Dans un premier temps, nous avons évalué la performance de DGGP. Nous avons choisi d'exécuter DGGP sur des graphes aléatoires de 14 sommets dans chaque ensemble, avec  $k = 10$  et  $k = 3$ . Les figures 5.16 et 5.17 affichent la variation du ratio d'évaluation lorsque le nombre d'arêtes du graphe augmente. On distingue une amélioration des performances liée à l'augmentation du nombre d'arêtes, la variance, lui étant inversement proportionnelle.

Le plus mauvais résultat est obtenu avec un ratio d'approximativement 1,7. de l'optimal. Aucun ratio n'est ici supérieur à 2 car les valeurs de  $\beta$  choisies sont inférieures aux poids. De plus les valeurs de  $\eta'$  étant élevées, la borne de  $2 + \frac{2}{\eta'}$  se rapproche rapidement de 2.

Notons également que des graphes de taille élevée donnent des résultats proches de l'optimal. Ceci s'explique car il est plus facile pour DGGP de créer des étapes de communications de durées semblables lorsque les poids des arêtes sont relativement homogènes : Le tirage aléatoire des poids des arêtes implique en effet une disparité élevée lorsque le nombre d'arêtes est bas. Les temps d'exécution obtenus font preuve de la même variance quelle que soit le nombre d'arêtes du graphe. Néanmoins, la borne inférieure augmentant avec le nombre d'arête, il est donc logique de voir la variance des ratios d'évaluation diminuer.

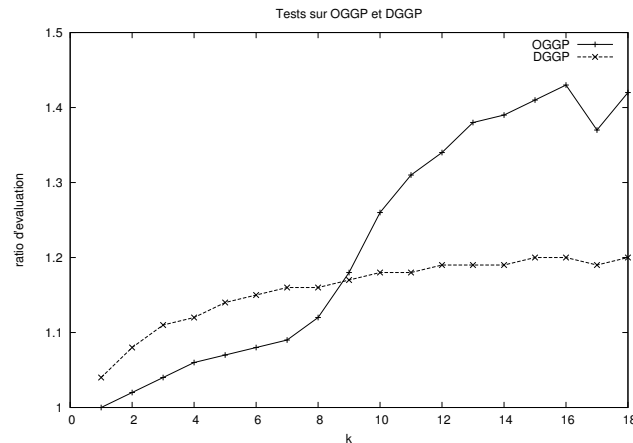


FIG. 5.18 – DGGP,OGGP : ratios d'évaluation

Nous avons également comparé DGGP avec l'algorithme OGGP. La figure 5.18 montre les ratios d'évaluations pour OGGP et DGGP lorsque  $k$  augmente. Les graphes d'entrées sont ici des graphes aléatoires de 18 sommets dans chaque ensemble, au nombre d'arêtes aléatoire. On peut voir que OGGP commence par fournir des résultats légèrement meilleurs que DGGP, mais lorsque  $k$  augmente les résultats obtenus par OGGP se dégradent avec des pics allant jusqu'à un temps 20 % plus long que pour DGGP.

Le comportement initial de DGGP, plus mauvais que OGGP s'explique en examinant les formules de chaque borne inférieure : en effet,  $\eta'_s$  et  $\eta'_d$  ont comme valeur le maximum entre 2 termes, un diminuant avec  $k$ , et l'autre restant constant quelle que soit la valeur de  $k$ . Par conséquent, lorsque  $k$  augmente, le terme indépendant de  $k$  a plus de chance d'être le maximum, et donc le facteur influant sur les résultats ; or DGGP permet justement des gains de performance lorsque le terme indépendant de  $k$  est le maximum.

Notons au passage que la qualité des résultats de OGGP n'est plus bornée par 2 pour ce

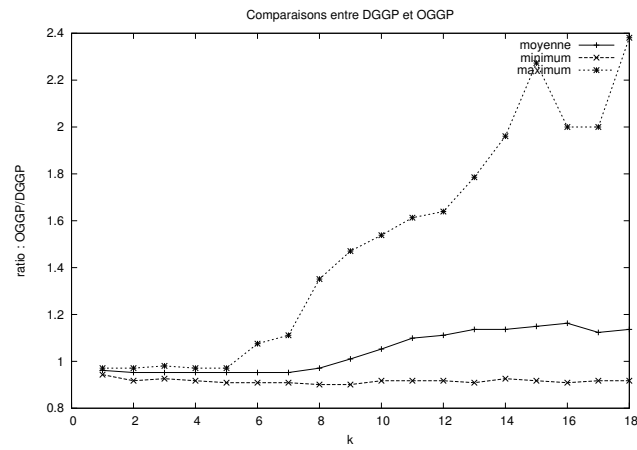


FIG. 5.19 – DGGP : Comparaisons avec OGGP

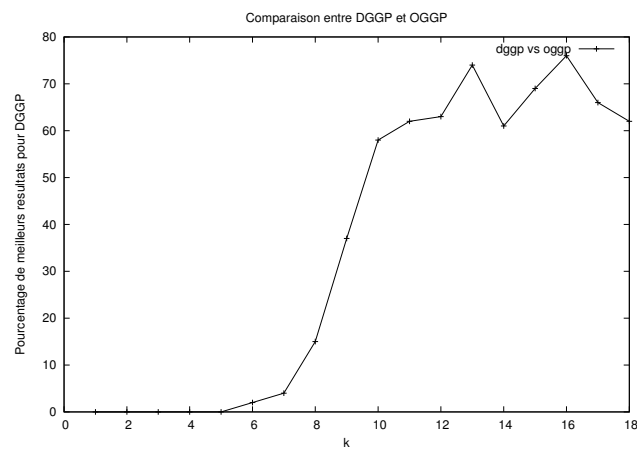


FIG. 5.20 – DGGP : Améliorations par rapport à OGGP

problème (en considérant  $\eta'$  bien sûr) : sur certains graphes, le ratio atteint des valeurs de 2,94.

La figure 5.19 affiche le ratio entre les temps obtenus par OGGP et les temps obtenus par DGGP. 3 courbes montrent le cas médian, le pire cas et le meilleur cas sur chaque échantillon de graphes aléatoires. Un ratio supérieur à 1 signifie donc que DGGP fournit de meilleurs résultats que OGGP. On voit qu'en moyenne DGGP est légèrement plus performant que OGGP, une fois atteinte une valeur de  $k$  suffisamment grande. Sur certains cas les ordonnancements obtenus par DGGP sont jusqu'à 2,4 fois plus rapide que ceux de OGGP (ce facteur augmentant avec  $\delta$ ). De l'autre côté, OGGP ne bat jamais DGGP d'un facteur supérieur à 10%.

Finalement, nous avons calculé, figure 5.20, le nombre de fois que DGGP est meilleur que OGGP sur un échantillon de graphes aléatoire, lorsque  $k$  varie. La courbe obtenue confirme les résultats précédents. DGGP fournit de meilleurs résultats dans 65% des cas pour les valeurs élevées de  $k$ .

## 5.2 Redistribution de données utilisant un réseau local

Nous étudions dans cette section une seconde modification du problème de redistribution du chapitre 3. La différence réside cette fois-ci dans l'ajout d'un réseau local à chaque grappe, pouvant être utilisé en parallèle à l'interface réseau connectant chaque nœud vers la dorsale. Différentes configurations matérielles peuvent en effet présenter cette particularité :

- les grappes disposant d'une seconde interface en Myrinet [18] ;
- les grappes disposant de deux interfaces réseau ethernet identiques par machine ;
- les grappes disposant d'une seule interface, mais dont l'architecture réseau implique une bande passante plus élevée pour les communications locales que les communications distantes.

L'étude de ce problème se décompose en plusieurs parties. Nous commençons par montrer section 5.2.1 que le problème étudié ici est différent d'un simple problème d'ordonnement : il est en effet nécessaire de calculer un routage pour les communications. La notion de routage complexifiant grandement le problème, nous présentons section 5.2.2 différents problèmes similaires de la littérature résolus dans le cas de communications en régime permanent. Nous décrivons ensuite section 5.2.3 un algorithme de routage et d'ordonnement basé sur le même principe que les algorithmes de la littérature. Enfin, nous étudions section 5.2.4 la complexité au pire cas, ainsi qu'un ratio d'approximation.

### 5.2.1 Problème du routage

L'introduction d'un réseau local rapide utilisable en parallèle avec le réseau longue distance peut permettre par une utilisation judicieuse de décharger d'une partie de ses communications une machine plus chargée que les autres.

Considérons par exemple la configuration réseau de la figure 5.21.

Supposons que le nœud 1 soit le seul nœud à devoir transférer des données. Ce nœud disposant d'une bande passante de 100Mbit/s, le temps total d'une communication directe avec chacun de ses nœuds destinations prendra comme temps la somme  $d$  (en Mbit) de toutes les quantités de données à transmettre divisée par 100. On remarque que seule la moitié de la bande passante de la dorsale, de 200Mbit/s, est utilisée dans ce cas. De ce fait, une manière plus efficace d'effectuer les communications consiste à transférer la moitié des données du nœud 1 à travers le réseau local vers le nœud 2, puis d'effectuer les communications distantes à l'aide de deux flux parallèles de 100Mbit/s chacun, saturant ainsi la dorsale.

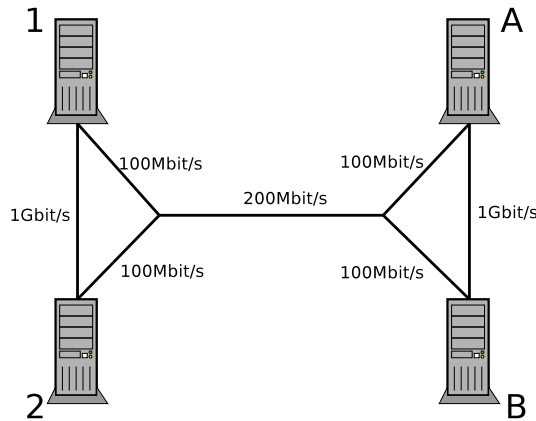


FIG. 5.21 – Configuration réseau avec liens locaux

Ainsi, atteindre un temps de transfert le plus court possible nécessite désormais non plus un simple ordonnancement des communications, mais une détermination d'un routage efficace. Notons que le routage obtenu peut être relativement complexe : dans certains cas en effet, plusieurs chemins peuvent être empruntés pour une source donnée et une destination donnée. La question n'est donc pas simplement de déterminer "par où passer" mais de déterminer quelle quantité de données passe par quel chemin.

Remarquons au passage qu'une répartition des données sur des nœuds locaux a pour but de diminuer une des bornes du problème KPBS introduite section 3.2.3 :  $\eta_d$ . En effet, étant donné un graphe de communication  $G$ ,  $\eta_d(G) = \max\left(\frac{P(G)}{k}, W(G)\right)$ . On sait que si  $\frac{P(G)}{k} > W(G)$  alors, un ordonnancement des communications atteignant un temps de redistribution égale à la borne inférieure sature la dorsale en utilisant en permanence  $k$  communications simultanées. Inversement si  $W(G) > \frac{P(G)}{k}$  alors la dorsale n'est pas saturée, et donc l'optimisation déroulée sur l'exemple précédent visant à décharger le nœud ayant le plus de données à transférer d'une partie de ses communications en utilisant le réseau local, peut être utilisée. Bien entendu, il est ainsi possible d'abaisser  $W(G)$ , mais en aucun cas  $\frac{P(G)}{k}$ , la quantité de données à transférer d'un bord à l'autre du réseau restant la même.

Les bornes inférieures sur le temps de redistribution pour le problème KPBS peuvent donc être utilisées comme un moyen rapide d'estimer s'il y a lieu ou non d'utiliser un routage par le réseau local. En examinant simplement le graphe de communication  $G$ , on peut conclure immédiatement que si  $\frac{P(G)}{k} > W(G)$  ou si  $W(G) > \frac{P(G)}{k}$  mais que l'écart entre  $W(G)$  et  $\frac{P(G)}{k}$  est très faible, il n'est alors pas nécessaire de chercher un routage. Un simple ordonnancement des communications à l'aide des algorithmes du chapitre 4 fournissant un résultat suffisamment proche de l'optimal.

### 5.2.2 Régime permanent

Plusieurs problèmes de la littérature sont proches du problème que nous étudions ici. Nous présentons rapidement le principe de fonctionnement de deux algorithmes développés par Legrand et al. visant à optimiser des temps de transferts pour des communications parallèles dans des réseaux hétérogènes. Ces travaux visent des communications de groupe plus simples telles des communications de type *scatter, reduce* [55] ou *broadcast* [12] mais forment un bon point de

départ pour être étendus à notre problème.

Le problème étudié étant particulièrement complexe, plusieurs hypothèses supplémentaires sont ajoutées, permettant d'atteindre de très bonnes solutions. L'hypothèse la plus importante est l'exécution des communications en *régime permanent* : on considère ici non pas une seule communication, mais un ensemble de grande taille de communications identiques à réaliser séquentiellement. Par "communications identiques" nous entendons des communications pour lesquels le motif de données à transférer est identique, les données en elles-mêmes pouvant être quelconques. Chaque fois qu'une communication est terminée, la communication suivante commence immédiatement. Le nombre total de communications à réaliser étant infini, le but ici n'est pas d'optimiser le temps total de terminaison, mais le débit d'exécution. Notons que pour notre problème, la notion de régime permanent est légèrement différente : en effet, on ne dispose pas d'un ensemble de communications identiques, mais d'un ensemble de *redistributions* identiques. Le grain considéré est donc moins fin dans notre cas.

L'utilisation du régime permanent permet aux différents algorithmes de Legrand et al. d'atteindre un débit asymptotiquement optimal (i.e. optimal lorsque le nombre de communications à effectuer tend vers l'infini). Ces algorithmes se découpent en 3 parties distinctes :

- une phase de calcul d'un routage optimal à l'aide de programmation linéaire ;
- une phase d'ordonnancement des communications à l'aide d'une modélisation sous forme de graphes ;
- une dernière phase, résolvant certains problèmes liés à l'ordonnancement obtenu.

Enfin, notons que pour ce problème, le temps d'établissement d'une communication  $\beta$ , introduit section 3.2.1 n'est pas pris en compte.

### 5.2.3 Algorithme

L'algorithme que nous avons développé pour résoudre le problème de redistribution en régime permanent fonctionne de même manière en 3 étapes de calcul de routage, ordonnancement et une dernière étape que nous appellerons "étape d'initialisation".

Nous décrivons ici en détail chacune de ces étapes, accompagnée d'un exemple sur lequel nous déroulerons chaque phase de l'algorithme.

#### Routage

Le calcul d'un routage optimal se fait, de manière similaire aux algorithmes de Legrand et al. en utilisant une modélisation sous forme de programmation linéaire. Notons au passage qu'elle n'est néanmoins pas identique.

Nous commençons par proposer une algorithmes fournissant un routage minimisant le temps de communication, puis nous affinons la modélisation afin de choisir entre tous les routages optimaux celui qui minimise la bande passante consommée.

**Modèle** L'idée principale derrière la modélisation utilisée consiste à éviter de faire apparaître le temps dans les équations. Pour ce faire, nous *fixons* le temps total de communication à 1 seconde et essayons de maximiser la quantité de données transférée durant cette seconde. Une fois le résultat obtenu, il suffit alors juste de multiplier toutes les quantités de données obtenues (ainsi que leurs temps de transferts) pour ré-obtenir les quantités initiales au problème.

Nous posons les notations suivantes :

- $S$  l'ensemble des émetteurs ;
- $R$  l'ensemble des destinataires ;

- $N = S \cup R$  l'ensemble de tous les nœuds ;
- $x_{i,j,k}$  la quantité de données que le nœud  $i$  envoie au nœud  $j$  avec le nœud  $k$  comme destination finale.  $x_{i,j,k}$  forment les variables de notre problème de programmation linéaire ;
- $r_{i,k}$  le ratio obtenu en divisant la quantité de données à envoyer du nœud  $i$  au nœud (final)  $k$  par la quantité totale de données à recevoir par toutes les destinations finale. On a donc :  $\forall i \in N, \forall k \in S, r_{i,k} = 0, \forall i \in R, \forall k \in N, r_{i,k} = 0$ , et  $\sum_{i \in S} \sum_{k \in R} r_{i,k} = 1$ .  $r_{i,k}$  sont nommés *ratios de communication* et vont permettre de s'assurer que les proportions de données entre les différents flux restent identiques aux quantités de la matrice de communication ( $m_{i_1, i_2}$ ) (un nœud devant envoyer deux fois plus de données qu'un autre d'après la matrice de communication le fera également dans la solution obtenue). Ces ratios sont obtenus à l'aide de la formule suivante :  $r_{i,k} = \frac{m_{i,k}}{\sum_{i'} \sum_{k'} m_{i',k'}}$  ;
- $b$  la bande passante de la dorsale ;
- $b_{i,j}$  la bande passante locale entre les nœuds  $i$  et  $j$  ;
- $b_i, i \in N$  la bande passante à laquelle le nœud  $i$  accède à la dorsale ;
- $O$  la quantité totale de données à réceptionner (au final).  $O = \sum_{i \in N} \sum_{j \in R} x_{i,j,j}$ .

Le nombre de variables utilisées est relativement élevé, mais certaines d'entre elles ont automatiquement une valeur de 0 et seront donc éliminées des contraintes du problème :

- il n'est pas possible d'envoyer des données d'un destinataire vers un émetteur :

$$\forall i \in R, \forall j \in S, \forall k \in N, x_{i,j,k} = 0$$

- il n'est pas possible d'envoyer des données à destination finale d'un émetteur :

$$\forall i \in N, \forall j \in N, \forall k \in R, x_{i,j,k} = 0$$

- un nœud ne peut pas envoyer des données avec lui même comme destination finale :

$$\forall i \in N, \forall j \in N, x_{i,j,i} = 0$$

- un nœud ne peut pas s'envoyer des données :

$$\forall i \in N, \forall k \in N, x_{i,i,k} = 0$$

Il nous est désormais possible de définir les contraintes du problème :

- Étant donné un nœud  $i$  et un destinataire  $k$ , la quantité de données que  $i$  envoie vers  $k$  est égale à la quantité qu'il a lui-même à envoyer, augmentée de la quantité que les autres nœuds envoient à  $k$  à travers  $i$  :

$$\forall i \in N, \forall k \in R, k \neq i, \sum_{j \in N} x_{i,j,k} = r_{i,k} \times O + \sum_{j \in N} x_{j,i,k}$$

Nous appellerons cette contrainte *contrainte de conservation* ;

- il n'est pas possible d'excéder la bande passante de la dorsale :

$$\sum_{i \in S} \sum_{j \in R} \sum_{k \in R} x_{i,j,k} \leq b$$

- il n'est pas possible d'excéder la bande passante d'un lien local :

$$\forall i \in R, \forall j \in R, \sum_{k \in R} x_{i,j,k} \leq b_{i,j}$$

$$\forall i \in S, \forall j \in S, \sum_{k \in R} x_{i,j,k} \leq b_{i,j}$$

– il n'est pas possible d'excéder la bande passante d'un lien de connection à la dorsale :

$$\forall i \in S, \sum_{j \in R} \sum_{k \in R} x_{i,j,k} \leq b_i$$

$$\forall j \in R, \sum_{i \in S} \sum_{k \in R} x_{i,j,k} \leq b_j$$

Les contraintes définies, il nous reste à formuler la fonction objectif qui est de maximiser la quantité de données arrivant à destination :

$$\max(O)$$

**Exemple** Considérons comme exemple une redistribution donnée par la matrice de communication de la figure 5.22 devant prendre place sur le réseau de la figure 5.21.

nœuds	A	B
1	400	20
2	0	40

FIG. 5.22 – Matrice de communication pour la redistribution en régime permanent

Le nœud 1 doit ainsi envoyer 400Mbit à destination du nœud A et 20Mbit à destination du nœud B ; le nœud 2 a juste 40 Mbit à envoyer à B.

Nous commençons par calculer les ratios de communication :  $r_{1,A} = \frac{400}{400+20+0+40} = \frac{20}{23}$ ,  $r_{1,B} = \frac{20}{460} = \frac{1}{23}$ ,  $r_{2,A} = 0$ ,  $r_{2,B} = \frac{40}{460} = \frac{2}{23}$ , les autres  $r_{i,k}$  sont nuls.

Nous pouvons maintenant écrire les contraintes :

– contrainte de conservation :

$$i = 1, k = A : x_{1,1,A} + x_{1,2,A} + x_{1,A,A} + x_{1,B,A} = \frac{20}{23}O + x_{1,1,A} + x_{2,1,A} + x_{A,1,A} + x_{B,1,A}$$

$$\Rightarrow x_{1,2,A} + x_{1,A,A} + x_{1,B,A} = \frac{20}{23}O + x_{2,1,A}$$

$$i = 1, k = B : x_{1,2,B} + x_{1,A,B} + x_{1,B,B} = \frac{1}{23}O + x_{2,1,B}$$

$$i = 2, k = A : x_{2,1,A} + x_{2,A,A} + x_{2,B,A} = x_{1,2,A}$$

$$i = 2, k = B : x_{2,1,B} + x_{2,A,B} + x_{2,B,B} = \frac{2}{23}O + x_{1,2,B}$$

$$i = A, k = B : x_{A,1,B} + x_{A,2,B} + x_{A,A,B} + x_{A,B,B} = 0O + x_{1,A,B} + x_{2,A,B} + x_{A,A,B} + x_{B,A,B}$$

$$\Rightarrow x_{A,B,B} = x_{1,A,B} + x_{2,A,B}$$

$$i = B, k = A : x_{B,A,A} = x_{1,B,A} + x_{2,B,A}$$

– contrainte de la dorsale :

$$x_{1,A,A} + x_{1,A,B} + x_{1,B,A} + x_{1,B,B} + x_{2,A,A} + x_{2,A,B} + x_{2,B,A} + x_{2,B,B} \leq 200$$

– contrainte des bandes passantes locales :

$$i = 1, j = 2 : x_{1,2,A} + x_{1,2,B} \leq 1000$$

$$i = 2, j = 1 : x_{2,1,A} + x_{2,1,B} \leq 1000$$

$$i = A, j = B : x_{A,B,B} \leq 1000$$

$$i = B, j = A : x_{B,A,A} \leq 1000$$

– contraintes des bandes passantes distantes :

$$i = 1 : x_{1,A,A} + x_{1,A,B} + x_{1,B,A} + x_{1,B,B} \leq 100$$

$$i = 2 : x_{2,A,A} + x_{2,A,B} + x_{2,B,A} + x_{2,B,B} \leq 100$$

$$j = A : x_{1,A,A} + x_{1,A,B} + x_{2,A,A} + x_{2,A,B} \leq 100$$

$$j = B : x_{1,B,A} + x_{1,B,B} + x_{2,B,A} + x_{2,B,B} \leq 100$$

L'objectif est, sous toutes ces contraintes, de maximiser :

$$O = x_{1,A,A} + x_{1,B,B} + x_{2,A,A} + x_{2,B,B} + x_{A,B,B} + x_{B,A,A}$$

En résolvant ce système, nous obtenons le routage optimal sur la partie gauche de la figure 5.23 transférant une quantité de 200Mbit en une seconde. Pour obtenir les 460Mbit du problème initial, il suffit de multiplier toutes les quantités de données par  $\frac{460}{200}$ . Nous obtenons ainsi le routage optimal situé sur la droite de la figure 5.23 d'un temps de 2,3 secondes.

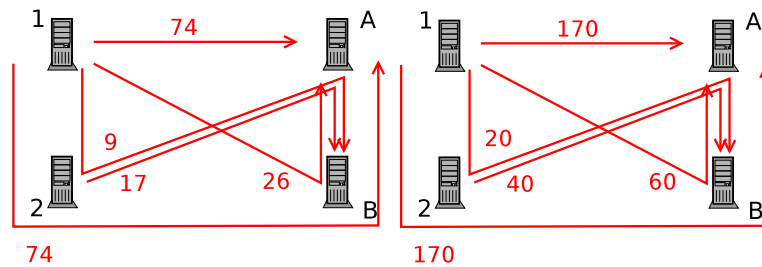


FIG. 5.23 – Routes optimales

**Objectif secondaire** L'analyse des résultats obtenus sur notre exemple montre aisément que le temps obtenu est optimal : en effet la quantité de données traversant le réseau en une seconde est de  $74 + 26 + 9 + 17 + 74 = 200$ Mbit, ce qui correspond à la bande passante de la dorsale. Néanmoins, on remarque également que les routes obtenues paraissent relativement complexe. Par exemple, les données devant transiter de 1 à B suivent le chemin  $(1, 2, A, B)$ , passant ainsi par tous les nœuds du réseau. On voit donc que rien n'impose aux routes obtenues d'être le plus simple possible, seule l'optimalité étant garantie.

Afin d'obtenir des routes plus simples, tout en gardant un résultat optimal, nous ajoutons un second objectif à notre problème de programmation linéaire.

Nous cherchons à maximiser  $O$ , tout en minimisant  $BP = \sum_{i \in N} \sum_{j \in N} \sum_{k \in N} x_{i,j,k}$ . Ceci signifie que nous cherchons la solution optimale consommant la plus petite quantité possible de

bande passante. Pour résoudre ce problème, nous commençons par résoudre le problème initial de maximisation, puis nous rajoutons une contrainte fixant  $O$  à la valeur obtenue et nous résolvons le problème de minimisation de  $BP$ .

Nous obtenons ainsi sur notre exemple les routes de la figure 5.24. On peut ainsi remarquer que le routage obtenu ici est beaucoup plus facile à mettre en œuvre que le routage obtenu initialement.

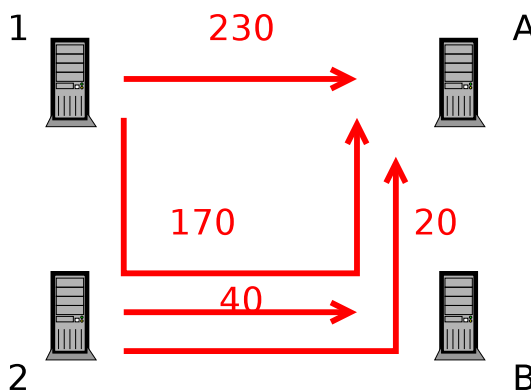


FIG. 5.24 – Solution du problème multi-objectif

### Ordonnancement

Toutes les routes étant connues, il reste nécessaire d'ordonner toutes les communications afin d'éviter de saturer le réseau. Pour ce faire, de manière similaire à la résolution du problème KPBS, nous réalisons toutes les communications en un ensemble d'étapes synchronisées fortement. Notons que les contraintes de bande passante utilisées pour le calcul du routage nous garantissent qu'il existe un ordonnancement capable d'atteindre le temps calculé à l'étape précédente.

Afin d'atteindre cet objectif, nous utilisons des techniques de décomposition de graphes en un ensemble de couplages. L'ensemble des communications à réaliser est modélisé à l'aide d'un graphe, que nous découpons en plusieurs graphes sur lesquels nous appliquons des algorithmes d'ordonnancement. Chaque étape obtenue vérifie la propriété que l'on ne dépasse à aucun moment la capacité d'un lien du réseau.

**Graphes d'entrée** Le graphe de routage obtenu à la première étape de l'algorithme va nous servir à construire trois graphes différents à ordonner. De manière similaire à la modélisation proposée chapitre 3 chaque sommet du graphe représente une interface réseau d'un nœud d'une grappe (2 sommets dans le cas d'une carte full duplex). Chaque arête représente une communication évaluée par son temps d'exécution optimal obtenu en divisant la quantité de données à transférer sur un lien par sa bande passante maximale.

Trois graphes représentent les communications à effectuer :

- un graphe représentant toutes les communications locales à la grappe émettrice ;
- un graphe représentant toutes les communications distantes à travers la dorsale ;
- un graphe représentant toutes les communications locales à la grappe réceptrice.

**Lemme 3** Les deux graphes de communications locales sont des graphes bipartis valués de type  $G = (V_1, V_2, E, w)$  où  $V_1$  est l'ensemble des sommets qui ne font qu'émettre des données,  $V_2$  l'ensemble des sommets qui ne font que recevoir des données,  $E$  l'ensemble des arêtes, et  $w$  leur fonction de valuation.

**Preuve :** Considérons le graphe représentant toutes les communications locales à la grappe émettrice (la démonstration fonctionne de manière symétrique pour la grappe réceptrice).

Supposons pour une contradiction qu'il existe un sommet que nous appellerons arbitrairement 1 qui soit à la fois émetteur et récepteur. Il reçoit une communication d'un sommet 2, à destination finale du sommet  $A$  et émet une communication vers un sommet 3 à destination finale du sommet  $B$ .

Notons  $y_{2,1,A}$  et  $y_{1,3,B}$  les poids de ces deux arêtes. Supposons  $y_{1,3,B} > y_{2,1,A}$  (la démonstration fonctionne de manière symétrique dans le cas contraire) : Nous pouvons modifier le graphe de communication afin d'envoyer la communication de 2 vers  $A$  à travers le sommet 3 plutôt que le sommet 1 : on remplace pour cela l'arête  $(2, 1)$  par une arête  $(2, 3)$  de poids identique (toutes les interfaces locales à une grappe ayant la même bande passante). Nous diminuons ensuite le poids de l'arête  $(1, 3)$  de  $y_{2,1,A}$ , et modifions le graphe de communications à travers la dorsale afin que toutes les données arrivent bien à destination. La figure 5.25 illustre cette opération.

Les nouvelles routes obtenues sont encore valides car chaque nœud source ou destination continue à émettre ou recevoir toutes ses données. De plus l'optimalité du routage est conservée.

On peut remarquer que le coût total de la redistribution en terme de quantité totale de données transférées (déterminée par la variable  $BP$  de l'étape de calcul du routage) a désormais diminué d'une valeur de  $y_{2,1,A}$ .

Ceci est une contradiction car les routes obtenues après la première étape de l'algorithme ont d'ors et déjà minimisé la valeur de  $BP$ .

Le graphe de communications locales à la grappe émettrice est donc bien un graphe biparti. La preuve est identique pour la grappe réceptrice. ■

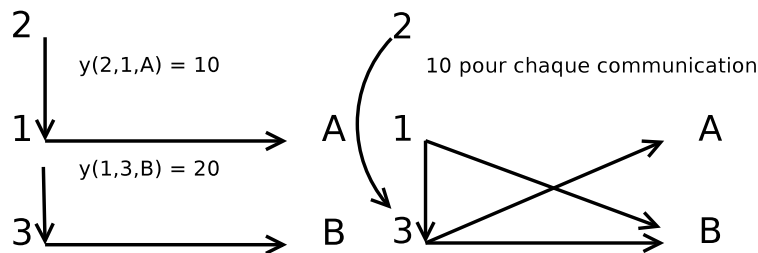


FIG. 5.25 – Construction des nouvelles routes

La figure 5.26 illustre la construction de chacun des graphes, à partir des routes obtenus dans la première partie figure 5.24. On voit ainsi par exemple que la communication de 190Mbit entre les nœuds 1 et 2 à 1Gbit/s est représentée par une arête de poids 0,19. Les communications à travers la dorsale étant réalisées à une vitesse de 100Mbit/s, la communication de 40Mbit entre 1 et  $A$  est représentée par une arête de poids 0,4.

**Calcul de l'ordonnancement** Afin d'ordonner les communications, plusieurs possibilités s'offrent à nous :

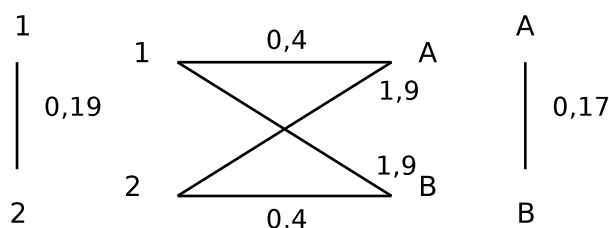


FIG. 5.26 – Graphes de communications locales et distantes

- tout d’abord, il est possible d’utiliser l’algorithme de Bongiovanni et al. [19] déjà présenté section 3.2.4;
- nous pouvons également utiliser l’algorithme GGP du chapitre précédent, moyennant un choix intelligent du paramètre  $\beta$  qu’il utilise. Afin de ne pas confondre dans cette section la latence réelle d’établissement d’une communication et le paramètre  $\beta$  utilisé par l’algorithme GGP, nous choisissons de nommer ce dernier  $\beta'$  par la suite.  $\beta'$  sera donc ici une valeur choisie et non dépendante d’une latence réelle. Le paramètre  $k$  prendra la valeur du nombre de communications autorisées pour les communications distantes, et la valeur de  $\min(|V_1|, |V_2|, m)$  pour les communications locales ( $k$  n’étant pas une contrainte dans ce cas).

Afin d’obtenir un ordonnancement optimal en utilisant GGP, nous prenons comme valeur fictive pour  $\beta'$  le plus grand diviseur commun entre les poids de toutes les arêtes divisé par  $k$  :

$$\beta' = \frac{\text{pgcd}_{e \in E} w(e)}{k}.$$

L’utilisation de l’algorithme GGP plutôt que de celui de Bongiovanni et al. permet d’obtenir un peu de souplesse à l’utilisation : en effet, pour une redistribution réelle, où il peut s’avérer utile d’éviter de subdiviser les communications en un nombre trop important de messages, une valeur plus élevée de  $\beta'$  pourrait être acceptable, le temps perdu par rapport au temps optimal étant compensé par le gain de temps résultant d’un nombre plus faibles de synchronisations. Estimer l’impact d’un tel choix sur les performances est néanmoins délicat et nécessite une étude plus expérimentale.

**Theorème 6** Pour tout graphe  $G = (V_1, V_2, E, w)$  Si  $\beta' = \frac{\text{pgcd}_{e \in E} w(e)}{k}$  GGP retourne un ordonnancement d’un coût égal à  $\eta_d(G)$ .

L’ensemble des démonstrations que nous utilisons ici, nécessite l’utilisation de toutes les notations développées section 4.1, ainsi que la description formelle de GGP, et celle de l’algorithme par multigraphe. Nous commençons par prouver un lemme intermédiaire :

**Lemme 4**  $P(H) = P(G)$  et  $\frac{P(H)}{k} \in \mathbb{N}$ .

**Preuve :**  $\beta' = \frac{\text{pgcd}_{e \in E} w(e)}{k}$  par conséquent,  $\forall e \in E, \beta' \times k$  divise  $w(e)$  et donc  $\beta'$  divise  $w(e)$ . Par conséquent la phase d’arrondi des poids construisant le graphe  $H$  n’a aucun effet et donc  $G = H$ . Par conséquent  $P(G) = P(H)$ .

De plus,  $\forall e \in E, k$  divise  $\frac{w(e)}{\beta'}$  on en déduit immédiatement que  $\forall e \in E_H, k$  divise  $\frac{w_H(e)}{\beta'}$  et par conséquent  $k$  divise  $P(H)$  et  $\frac{P(H)}{k} \in \mathbb{N}$ . ■

**Preuve du théorème 6 :**  $S'$  étant la solution obtenue par GGP, on a  $c(S') = 2\eta_s(J') = \eta_s(J') + \eta_s(J') \times \frac{\beta'}{\beta'}$ . Néanmoins la moitié des coûts de cette solution est due uniquement au délai d'initialisation des communications qui a ici une valeur réelle de  $\beta = 0$ . On a donc en réalité :  $c(S') = \eta_s(J') + \eta_s(J') \times \frac{\beta}{\beta'} = \eta_s(J') + \eta_s(J') \times 0 = \eta_s(J')$ .

$$\begin{aligned} c(S') &= \eta_s(J') \\ \Rightarrow c(S') &= \eta_d(J) \\ \Rightarrow c(S') &= \eta_d(I) \\ \Rightarrow c(S') &= \max\left(\left\lceil \frac{P(H)}{k} \right\rceil, W(H)\right) \end{aligned}$$

en suivant un raisonnement identique à celui de la preuve du théorème 2. On sait d'après le lemme 4 que  $\left\lceil \frac{P(H)}{k} \right\rceil = \frac{P(H)}{k}$  et par conséquent :

$$c(S') = \max\left(\frac{P(H)}{k}, W(H)\right)$$

On en déduit toujours d'après le lemme 4 que :

$$\begin{aligned} c(S') &= \max\left(\frac{P(G)}{k}, W(G)\right) \\ \Rightarrow c(S') &= \eta_d(G) \end{aligned}$$

■

Une fois les 3 ordonnancements obtenus, il reste à les rassembler en un seul. Pour ce faire nous utilisons l'algorithme de la figure 10. On commence par utiliser cet algorithme pour regrouper les ordonnancements des communications locales et la grappe émettrice et les communications distantes. Ensuite le regroupement de l'ordonnancement obtenu et de l'ordonnancement des communications locales à la grappe réceptrice se fait en fusionnant les deux ordonnancements dans l'ordre inverse, à partir des étapes à exécuter en dernier. De cette manière, nous essayons d'éloigner le plus possible dans le temps les deux redistributions locales. Le temps de l'ordonnancement obtenu est donné par le maximum des temps de chacun des trois ordonnancements.

**Exemple** Toujours en restant sur le même exemple, nous prenons  $\beta' = \frac{0,01}{2}$ . Nous obtenons pour les ordonnancements des communications locales deux ordonnancements d'une seule étape de temps respectifs de 0,17 seconde et 0,19 seconde. Les deux ordonnancements obtenus sont identiques aux graphes de communication leurs correspondant.

Pour les communications de la dorsale, nous obtenons deux étapes de durée 0,4 seconde et 1,9 seconde représentées sur la figure 5.27.

Enfin, nous rassemblons l'ensemble des ordonnancements dans un seul ordonnancement. Nous obtenons ainsi les 4 étapes de communication de la figure 5.28. On remarque le nombre plus élevé d'étapes, obtenu lors du découpage des étapes les plus longues. On peut également vérifier que toutes les communications à l'intérieur d'une étape sont bien de tailles identiques.

**Entrées** : deux listes de couplages  $S_1$  et  $S_2$  contenant chacune un ordonnancement solution.

**Sorties** : une liste  $S$  de couplages contenant la fusion de  $S_1$  et  $S_2$  en un seul ordonnancement.

soit  $M_1$  le premier couplage de  $S_1$  et  $M_2$  le premier couplage de  $S_2$  ;

**tant que** *il reste un couplage dans  $S_1$  ou  $S_2$*  **faire**

**si**  $M_1$  ou  $M_2$  est vide **alors** retourner  $S$  ;

soit  $p_1$  le poids des arêtes de  $M_1$ ,  $p_2$  le poids des arêtes de  $M_2$  et  $p = \min(p_1, p_2)$  (on rappelle que dans chaque couplage, toutes les arêtes sont de poids identiques) ;

construire  $C$  le couplage contenant toutes les arêtes de  $M_1$  et  $M_2$  avec pour chaque arête un poids  $p$  et ajouter  $C$  à  $S$  ;

**si**  $p_1 < p_2$  **alors**

éliminer  $M_1$  ;

prendre à sa place le prochain couplage de  $S_1$  ;

changer le poids de chaque arête de  $M_2$  à une valeur de  $p_2 - p$  ;

**fin**

**sinon**

**si**  $p_1 > p_2$  **alors**

éliminer  $M_2$  ;

prendre à sa place le prochain couplage de  $S_2$  ;

changer le poids de chaque arête de  $M_1$  à une valeur de  $p_1 - p$  ;

**fin**

**sinon**

**si**  $p_1 = p_2$  **alors**

éliminer  $M_1$  et  $M_2$  ;

prendre à leur place les prochains couplages de  $S_1$  et  $S_2$  ;

**fin**

**fin**

**fin**

**Algorithme 10** : Algorithme de regroupement de deux ordonnancements

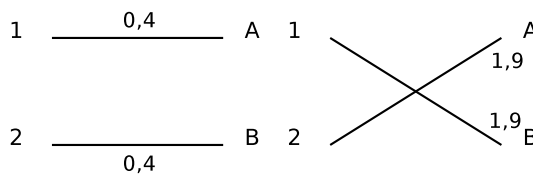


FIG. 5.27 – Ordonnement des communications sur la dorsale

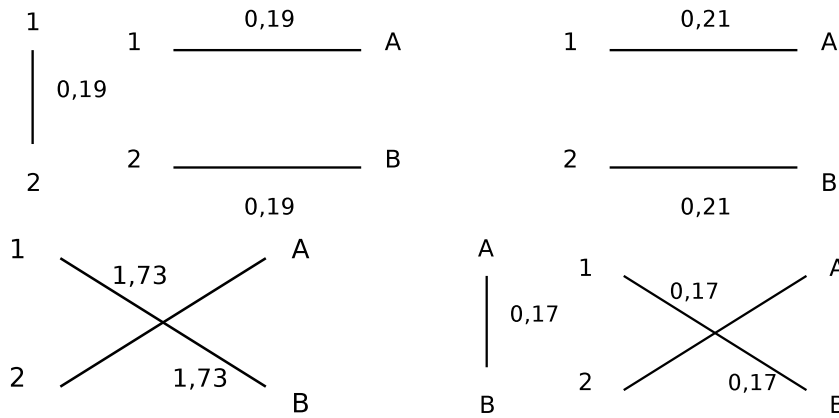


FIG. 5.28 – Ordonnancement final

### Initialisation

L'ordonnancement obtenu à ce niveau n'est malheureusement pas toujours réalisable : il est en effet possible qu'un nœud doive envoyer des données avant même de les avoir reçues. Ceci vient du fait qu'à aucun moment dans le calcul de l'ordonnancement n'apparaît une contrainte impliquant l'attente de données, les ordonnancements étant calculés séparément pour chaque graphe.

De plus, il est parfaitement possible d'avoir une chaîne de nœuds s'attendant l'un l'autre. Ceci signifie que des données ayant à parcourir un chemin composé de  $n$  nœuds peuvent parfaitement avoir à subir  $n - 1$  attentes.

C'est à ce niveau de l'algorithme que rentre en jeu l'hypothèse du régime permanent : puisque plusieurs redistributions identiques sont exécutées d'affilée, il est possible de procéder de la manière suivante :

- on stocke les données arrivant pour les transmettre à la redistribution suivante ;
- si des données de la redistribution précédente sont disponibles, elles sont envoyées à la place de celles de la redistribution courante.

Considérons par exemple, l'ordonnancement des communications du nouvel exemple de la figure 5.29. On considère ici un lien distant deux fois plus lent que les liens locaux.

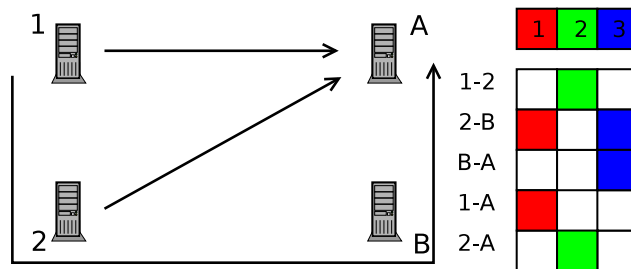


FIG. 5.29 – Routes et ordonnancement non réalisable

L'ordonnancement considéré prend place en trois étapes de redistribution. Sur cet exemple, les données envoyées du nœud 1 au nœud 2 prennent deux fois plus de temps à arriver que pour transiter du nœud 2 au nœud B. Les communications entre 2 et B nécessitent donc 2 étapes.

On remarque que les données de 2 à  $B$  sont envoyées avant d'être reçues. De manière similaire, les données de  $B$  à  $A$  sont envoyées pendant qu'elles sont reçues, ce qui n'est pas réalisable.

Pour résoudre ce problème, nous exécutons plusieurs redistributions une après l'autre. Lors de chaque redistribution, seules les données qu'il est possible de transmettre sont transmises. Les données impossibles à transmettre sont gardées pour être transférées lors de la redistribution suivante. La figure 5.30 montre l'exécution d'une telle opération sur notre exemple. On peut voir que les données envoyées de 2 à  $B$  nécessitent 2 redistributions complètes avant d'arriver à destination. On remarque également qu'après une phase d'initialisation non optimale, le débit maximal de régime permanent est atteint.

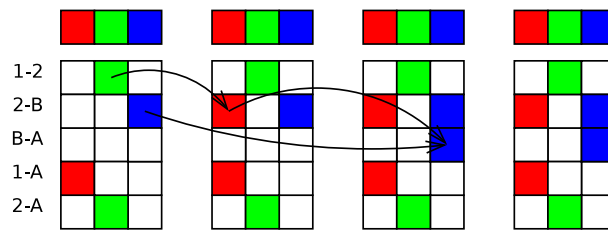


FIG. 5.30 – Phase d'initialisation permettant d'atteindre le régime permanent

### 5.2.4 Étude

Nous présentons à présent une étude de l'algorithme développé pour résoudre le problème de redistribution en régime permanent pour deux grappes distantes disposant chacune d'un réseau local.

Nous étudions la complexité au pire cas de notre algorithme et prouvons qu'il est asymptotiquement optimal utilisé pour des redistributions effectuées en régime permanent, et une 3-approximation utilisé pour l'exécution d'une seule redistribution.

#### Complexité

**Proposition 12** *L'algorithme de redistribution à une complexité au pire cas de  $n^9$ .*

**Preuve :** L'algorithme GGP utilisé pour le calcul de l'ordonnancement a une complexité au pire cas en  $O(\sqrt{n(G)}(m(G) + n(G))^2)$ . La phase d'initialisation ne nécessite que de parcourir 2 fois toutes les communications à effectuer et à donc une complexité linéaire. La partie la plus complexe de l'algorithme est donc la phase de calcul de routage.

La résolution d'un problème de programmation linéaire en nombre rationnels est un problème polynomial pouvant être résolu en utilisant l'algorithme de Vaidya [66] en un temps  $t^3$  où  $t$  est le nombre de variables utilisées. Les variables utilisées ici sont de la forme  $x_{i,j,k}$  où chaque indice  $i, j$  et  $k$  désigne un sommet du graphe de communication. Par conséquent le nombre de variables est en  $O(n^3)$  et la complexité de l'algorithme de calcul d'un routage optimal est donc en  $O((n^3)^3)$  soit  $O(n^9)$ , ce qui représente une complexité particulièrement élevée.

La complexité globale de l'algorithme est donc déterminée par celle de la phase de calcul du routage et en  $O(n^9)$ . ■

Il est clair qu'une complexité aussi élevée rend notre algorithme difficile d'utilisation en pratique. Nous espérons pouvoir trouver par la suite une méthode de calcul plus efficace (et peut-être

moins générique) ou bien des algorithmes de complexité plus faible mais garantissant néanmoins une certaine qualité des résultats. L'ensemble de ces possibilités fait partie des problèmes ouverts que nous étudions actuellement.

### Ratio d'approximation

L'hypothèse de redistribution en régime permanent a été introduite afin de faciliter l'étude du problème de redistribution sur des réseaux où la notion de routage est importante pour les performances.

Nous étudions donc dans un premier temps les performances de notre algorithme dans le cas de redistributions en régime permanent et montrons qu'il est asymptotiquement optimal.

Nous montrons de plus que même dans le cas simple où une seule redistribution est à effectuer, l'algorithme nous garantit des temps de redistribution inférieurs à 3 fois le temps optimal.

Nous commençons par prouver que la solution obtenue au problème de redistributions en régime permanent est asymptotiquement optimale.

**Lemme 5** *Le transfert de toutes les données de  $n$  redistributions nécessite au plus  $n + 2$  redistributions.*

**Preuve :** Il s'agit de prouver ici que seules 2 redistributions supplémentaires sont nécessaires. Considérons pour ce faire, un jeu de données à transférer :

Lors de la première redistribution, toutes les communications locales à la grappe émettrice sont exécutées : en effet, comme le graphe de ces communications est biparti, il n'existe aucun nœud qui soit à la fois émetteur et récepteur. Ceci signifie qu'aucune communication à l'intérieur de cette grappe ne passe par un nœud intermédiaire, et donc qu'il n'y a aucune possibilité de blocage et d'attente.

Lors de la seconde redistribution, toutes les communications traversant la dorsale peuvent être exécutées. En effet, dans le pire des cas, toutes les données sont arrivées à leur destination dans la grappe émettrice lors de la redistribution précédente.

Une fois ces deux redistributions complétées, plus aucune attente n'est nécessaire, chaque nœud de la grappe réceptrice disposant des données dont il a besoin. ■

**Théorème 7** *L'algorithme de routage et d'ordonnement pour des redistributions effectuées en régime permanent est asymptotiquement optimal.*

**Preuve :** Nous avons prouvé aux différentes étapes de présentation de l'algorithme que le routage ainsi que l'ordonnement étaient tous deux optimaux. Chaque redistribution est donc exécutée en un temps optimal et par conséquent  $n$  redistributions nécessitent au plus un temps  $\frac{n+2}{n}$  fois plus élevé que le temps optimal. Lorsque  $n$  tend vers l'infini, ce ratio tend vers 1 et par conséquent notre algorithme est asymptotiquement optimal. ■

**Théorème 8** *L'algorithme de routage et d'ordonnement utilisé pour une seule redistribution est une 3-approximation.*

**Preuve :** Nous avons prouvé pour le théorème 7 que le temps de redistribution était au pire cas  $\frac{n+2}{n}$  fois plus élevé que le temps optimal. Pour une valeur de  $n = 1$ , on obtient donc un temps 3 fois plus élevé que le temps optimal. Cet algorithme est donc bien une 3-approximation. ■

# Chapitre 6

## Conclusion

*Il n'y a pas des problèmes qu'on se pose, il y a des problèmes qui se posent. Il n'y a pas de problèmes résolus, il y a seulement des problèmes plus ou moins résolus.*

*Henri Poincaré*

### 6.1 Conclusion

Les développements des réseaux et des technologies de communication ont permis l'émergence des grilles de calculs. Néanmoins, les nombreuses possibilités d'utilisation des grilles sont encore tempérées par une complexité de mise en œuvre importante. Dans le cadre de ressources distribuées à une échelle aussi grande qu'une grille, il convient de s'assurer une gestion la plus efficace possible des communications, et-ce particulièrement dans l'optique de communications parallèles massives. A l'heure actuelle, de nombreux projets de recherche visent à améliorer les performances et l'utilisation du réseau : nouveaux protocoles, caches, qualité de service,...

Nous avons pour notre part choisi de nous placer au niveau applicatif. Nous avons ainsi étudié dans cette thèse comment ordonnancer les communications lors de redistributions de données entre deux grappes d'ordinateurs reliées par un réseau à haut débit. Le travail d'ordonnancement et d'émission des communications peut ainsi être réalisé par l'application, une bibliothèque, ou encore l'intergiciel en charge des communications. Se placer au dessus de la couche réseau nous permet ainsi d'accéder à des informations inaccessibles à plus bas niveau : le motif global de données à redistribuer. Nous avons ainsi vu au chapitre 3 qu'un ordonnancement des communications peut avoir un intérêt y compris sur un réseau théorique parfait, distribuant le plus équitablement possible la bande passante et ce sans aucune perte ou ré-émission.

La première étape de nos travaux a consisté en la validation du modèle utilisé pour représenter les communications (par étapes ou non). Nous avons ainsi vérifié section 3.2.5 que les temps prédits par le modèle pour une redistribution ordonnancée ou non sont proches des temps obtenus dans la réalité.

Trois problèmes proches de redistribution sont étudiés dans cet ouvrage. Pour chacun d'entre eux, nous nous attachons à ne jamais dépasser les capacités du réseau par un ordonnancement efficace des communications.

1. le problème le plus simple, de redistribution entre deux grappes homogènes, modélisé par le problème KPBS, reposant sur des graphes représentant les communications ;

Nous avons proposé pour ce problème deux algorithmes d'approximation : GGP et OGGP, d'un ratio d'approximation de  $\frac{8}{3}$ , garantissant ainsi que tout résultat obtenu ne peut

prendre un temps plus de  $\frac{8}{3}$  fois plus élevé que le temps de redistribution minimal. Ces deux algorithmes ont de plus une complexité polynomiale relativement faible, les rendant ainsi utilisables en pratique. Lors de notre étude théorique de leurs caractéristiques, nous avons également prouvé qu'il n'est pas possible d'améliorer le ratio d'approximation de GGP à un facteur inférieur à 2, tandis qu'une telle possibilité reste ouverte pour l'algorithme OGGP.

Afin de comparer nos deux algorithmes entre eux, nous avons donc réalisé une série de simulations, sur des graphes aléatoires. GGP et OGGP ont été également comparés à un algorithme de redistribution par force brute, fonctionnant en réalisant toutes les communications simultanément, ainsi qu'à deux heuristiques gloutonnes d'ordonnancement. Les simulations nous ont permis de voir que OGGP fournit dans la plupart des cas de meilleurs résultats que GGP, et que les deux algorithmes fournissent dans les cas d'une latence faible par rapport aux temps de communications des ordonnancements de temps quasi-optimaux.

Nous avons également mis en pratique ces algorithmes lors de redistributions réelles réalisées entre deux grappes locales. Deux types de tests ont été réalisés, reposant sur la bibliothèque *MPI* ou uniquement sur la *libc*. Dans les deux cas, nous avons vérifié que l'ordonnancement des communications peut amener des gains de performances qui peuvent s'avérer non négligeables. Nous avons également proposé un algorithme permettant d'estimer le temps pris par une redistribution par force brute, permettant ainsi de choisir au cas par cas l'approche la plus performante.

2. le problème d-KPBS, extension du problème KPBS pour les grappes hétérogènes ;

Nous avons montré que dans le cas de grappes disposant d'interfaces de communications hétérogènes, le problème KPBS nécessite d'être modifié en vue d'atteindre un temps optimal. La différence entre d-KPBS et KPBS réside dans l'autorisation de communications simultanées sur chaque interface réseau. Le problème d-KPBS étend donc notre problème initial en un problème plus complexe. Nous avons modifié l'algorithme GGP pour former l'algorithme DGGP, et prouvé que ce dernier est une 4-approximation pour le problème d-KPBS. Nous avons également montré qu'il est impossible de prouver un ratio d'approximation inférieur à 3 (pour n'importe quel algorithme) sans auparavant trouver de nouvelles bornes inférieures sur les temps de redistribution.

Enfin, DGGP a été validé à l'aide de simulations, montrant sa capacité à tirer parti de communications parallèles pour s'approcher, lui aussi, du temps minimal de redistribution pour les simulations de latence faible.

3. un problème de redistribution sur une topologie plus complexe, permettant l'utilisation de communications locales à chaque grappe.

Pour ce dernier problème, plus complexe, ne forme pas une extension du problème KPBS : en effet, afin de permettre une résolution, les délais d'établissement des communications sont considérés ici comme négligeables.

Nous avons montré comment l'ajout de liens locaux complexifie fortement le problème, en rajoutant la notion de routage. Nous avons proposé un algorithme calculant routage et ordonnancement des communications, permettant d'atteindre des temps de transferts asymptotiquement optimaux dans le cas de redistributions réalisées en régime permanent et formant une 3-approximation dans le cas où une seule redistribution est à réaliser. Cet algorithme bien que polynomial, est en revanche de complexité élevée.

## 6.2 Perspectives

Plusieurs perspectives sont ouvertes, pour chacun des différents travaux que nous avons menés.

Pour le problème de redistribution simple, modélisé par KPBS et résolu par les algorithmes GGP et OGGP, nous envisageons de modifier le problème pour traiter différents cas dynamiques :

- dans un premier temps, nous envisageons de considérer le problème où le motif de données à redistribuer n'est pas connu initialement, mais où les quantités de données à transmettre varient lors de la redistribution. Il est possible d'adapter simplement les algorithmes GGP et OGGP pour ce cas (en recalculant l'ordonnancement au fur et à mesure), mais l'estimation de l'efficacité d'une telle méthode reste encore du domaine de la recherche.
- dans une seconde étude, nous souhaitons examiner le comportement des redistributions par étape lorsque le réseau est sujet à des variations aléatoires de charge. Nous pourrions ainsi étudier le comportement de nos algorithmes dans ce cadre et les modifier pour chercher à obtenir des performances plus élevées par une adaptabilité plus grande aux changements de configurations.

Enfin, il est possible d'utiliser le modèle utilisé pour KPBS avec des problèmes plus classiques d'ordonnancement de tâches, notamment en utilisant les bornes inférieures sur les temps de communications comme moyen rapide d'estimer les temps de transferts des données. Il serait particulièrement intéressant de voir dans quelle mesure une estimation précise des temps de transferts pourrait impacter le positionnement des tâches ou le temps total d'exécution d'un ensemble de tâches interdépendantes.

Pour le problème d-KPBS, ainsi que le problème de redistribution incluant des communications locales, différents travaux sont possibles, d'ordre plus théorique que pour KPBS.

Pour d-KPBS, il serait intéressant de diminuer le ratio d'approximation de DGGP. Pour ce faire, la première étape consisterait à trouver de nouvelles bornes inférieures sur le temps de redistribution, plus proches du temps de redistribution minimal. Notons au passage que ce problème semble relativement complexe, les bornes actuelles étant atteintes dans de nombreux cas.

La plupart des travaux théoriques envisagés concernent donc le problème de redistribution avec communications locales. L'ensemble des travaux suivants sont ici envisageables :

- la première modification de l'algorithme envisagé consiste à trouver une nouvelle méthode de calcul du routage, de complexité moindre. Une possibilité pourrait consister à accepter de calculer un routage non optimal afin de permettre un temps de calcul moins élevé. Une fois cette étape franchie il deviendrait alors possible d'effectuer un ensemble de simulations.
- la seconde modification consisterait à autoriser les communications à destination des nœuds locaux, ainsi qu'à considérer le cas de grappes hétérogènes. Dans les deux cas, les modifications apportées influeraient uniquement sur la phase d'ordonnancement : sous ces hypothèses les graphes de communications à ordonner ne seraient pas forcément bipartis. Un algorithme d'ordonnancement serait ainsi plus complexe à mettre en œuvre. Il est possible d'envisager plusieurs approches basés sur des algorithmes de coloriage d'arête comme base de recherche.
- la notion de routage étant introduite, il est possible d'envisager le problème de redistribution sur des topologies plus complexes. A ce stade, il nous apparaît néanmoins qu'une telle extension relève plus du domaine de la recherche théorique que de la recherche appliquée.
- enfin, la prise en compte de la latence forme dans le cas présent un problème théorique difficile. Il est en effet délicat de calculer un routage optimal sans connaître au préalable le nombre de fois où une préemption des communications est nécessaire ; routage et or-

donnancement sont donc ici fortement liés. Il est par contre possible d'étudier l'impact de la latence sur l'algorithme proposé, notamment à l'aide de simulations. Une telle étude permettrait d'estimer comment choisir au mieux les paramètres de l'algorithme d'ordonnancement.

# Annexe A

## Notations

Les notations sont présentées ici dans l'ordre dans lequel elles ont été introduites au fil des chapitres.

<i>Notation</i>	<i>Description</i>	<i>page</i>
$b_1$	bande passante locale à la grappe émettrice	42
$b_2$	bande passante locale à la grappe réceptrice	42
$b_b$	bande passante de la dorsale	42
$\beta$	délai d'établissement d'une communication	42
$G = (V_1, V_2, E, w)$	graphe biparti	42
$V_1$	premier ensemble de sommets du graphe $G$	42
$V_2$	second ensemble de sommets du graphe $G$	42
$E \subseteq V_1 \times V_2$	ensemble des arêtes du graphe $G$	42
$w : E \rightarrow \mathbb{Q}$	fonction de valuation de $G$ associant un poids à chaque arête	42
$\mathcal{M}$	matrice de communication	42
$m =  E $	nombre d'arêtes de $E$	43
$n =  V_1 \cup V_2 $	nombre de sommets de $G$	43
$\Delta(G)$	degré maximal du graphe $G$	43
$D$	décomposition de $G$ en étapes	43
$M_i$	$i^{\text{ème}}$ couplage de la décomposition $D$	43
$k$	nombre maximal de communications parallèles autorisées	47
$p : V_1 \cup V_2 \rightarrow \mathbb{Q}$	fonction de valuation des sommets associe à chaque sommet la somme des poids $w$ des arêtes qui lui sont incidentes	47
$W(G)$	maximum des poids $p$ des sommets pour le graphe $G$	47
$P(G)$	la somme des poids de toutes les arêtes de $G$	47
$\eta_d(G)$	borne inférieure sur le temps de transferts des données	47
$\eta_s(G)$	borne inférieure sur le nombre d'étapes de redistribution	48
$\eta(G)$	borne inférieure sur le temps de redistribution	48
trans	fonction de transformation en multigraphe	73

<i>Notation</i>	<i>Description</i>	<i>page</i>
$\delta : V_1 \cup V_2 \rightarrow \mathbb{N}$	fonction associant à chaque sommet le nombre de communications autorisées en parallèles	100
$d : V_1 \cup V_2 \rightarrow \mathbb{N}$	fonction associant à chaque sommet son degré	101
$\eta'(G)$	borne inférieure sur le temps de redistribution pour le problème d-KPBS	101
$\eta'_d(G)$	borne inférieure sur le temps de transferts des données pour le problème d-KPBS	102
$\eta'_s(G)$	borne inférieure sur le nombre d'étapes de redistribution pour le problème d-KPBS	102

# Annexe B

## Programmes

### B.1 Exemple d'utilisation de la bibliothèque libkpbs

```

#include"kpbs.h"
#include<stdlib.h>

int main(int argc, char**argv) {

if (argc != 4) {
    cerr<<"usage : "<<argv[0]<<" communications k beta"<<endl;
    exit(1);
}

//ouverture de la matrice de communication
FILE *comms_file = fopen(argv[1], "r");
if (comms_file == NULL) {
    perror("impossible d'ouvrir le fichier contenant la matrice de communication");
    exit(1);
}
unsigned int senders, receivers;
fscanf(comms_file, "%dx%d\n", &senders, &receivers);

//construction du graphe biparti
bigraph g;

//ajout de sommets
for(unsigned int i = 0 ; i < senders ; i++)
    g.add_node(true);
for(unsigned int i = 0 ; i < receivers ; i++)
    g.add_node(false);

//lecture des arêtes
for(unsigned int i = 0 ; i < senders * receivers ; i++) {
    float value;

```

```
char tmp;
fscanf(comms_file, "%f", &value);
fread(&tmp, 1, 1, comms_file);
unsigned int node2 = i % receivers;
unsigned int node1 = (i - node2) / receivers;
if (value != 0) g.add_edge(g.get_node(node1, true), g.get_node(node2, false), (double)
value);
}

fclose(comms_file);

g.set_beta(atoi(argv[3]));
g.set_k(atoi(argv[2]));
cout<<"Optimal bound : "<<g.get_optimal_bound()<<" ";

//lancement de OGGP
kpbs_approximation app1 (&g);
app1.OGGP();

//affichage des résultats
kpbs_approximation app1 (&g);
cout<<"oggp : "<<app1.get_cost()<<endl;
app1.display();
}
```

# Annexe C

## Bibliographie



# Bibliographie

- [1] Cray x-mp. [http://en.wikipedia.org/wiki/Cray\\_X-MP](http://en.wikipedia.org/wiki/Cray_X-MP).
- [2] DIET (Distributed Interactive Engineering Toolbox). <http://graal.ens-lyon.fr/DIET>.
- [3] Doxygen : génération automatique de documentation. <http://www.doxygen.org>.
- [4] Le projet hydrogrid. <http://www-rocq.inria.fr/~kern/HydroGrid/HydroGrid.html>.
- [5] Le projet manhattan. [http://en.wikipedia.org/wiki/Manhattan\\_Project](http://en.wikipedia.org/wiki/Manhattan_Project).
- [6] Le réseau vthd. <http://www.vthd.org/?wpid=5296>.
- [7] rshaper. <http://ar.linux.it/software/#rshaper>.
- [8] Z3. <http://en.wikipedia.org/wiki/Z3>.
- [9] *3rd IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, 12-15 May 2003, Tokyo, Japan. IEEE Computer Society, 2003.
- [10] F. Afrati, T. Aslanidis, E. Bampis, and I. Milis. Scheduling in switching networks with set-up delays. In *AlgoTel 2002*, Mèze, France, May 2002.
- [11] K.R. Baker. *Introduction to sequencing and scheduling*. John Wiley & Sons, New-York, 1974.
- [12] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Optimizing the steady-state throughput of broadcasts on heterogeneous platforms. Technical Report 2003-34, LIP, jun 2003.
- [13] M. A. Bender, S. Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *SODA : ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1998.
- [14] C. Berge. *The Theory of Graphs*. (Dover, New York), 2001.
- [15] F. Bertrand, R. Bramley, K. B. Damevski, J. A. Kohl, D. E. Bernholdt, J. W. Larson, and A. Sussman. Data redistribution and remote method invocation in parallel component architectures. In *Proceedings of the 19th International Parallel and Distributed Processing Symposium : IPDPS 2005*, 2005. to appear ; Best Paper Award.
- [16] P. B. Bhat, V. K. Prasanna, and C. S. Raghavendra. Block-cyclic redistribution over heterogeneous networks. *Cluster Computing*, 3(1) :25–34, 2000.
- [17] L. S. Blackford, J. Choi, A. Cleary, E. D’Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users’ Guide*. SIAM, 1997.
- [18] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W. Su. Myrinet : A gigabit-per-second local area network. *IEEE Micro*, 15(1) :29–36, 1995.
- [19] G. Bongiovanni, D. Coppersmith, and C. K. Wong. An Optimum Time Slot Assignment Algorithm for an SS/TDMA System with Variable Number of Transponders. *IEEE Transactions on Computers*, 29(5) :721–726, 1981.

- 
- [20] Y. Caniou and E. Jeannot. New Scheduling Heuristics in the Client-Agent-Server Model. In *IEEE Heterogeneous Computing Workshop (HCW'03)*, Nice, France, April 2003.
- [21] H. Casanova and J. Dongarra. NetSolve : A network-enabled server for solving computational science problems. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3) :212–223, Fall 1997.
- [22] H. Casanova and L. Marchal. A network model for simulation of grid application. Research Report RR-2002-40, LIP, ENS Lyon, France, October 2002. Also available as INRIA Research Report RR-4596.
- [23] B. Chapman, P. Mehrotra, H. Moritsch, and H. Zima. Dynamic data distributions in Vienna fortran. Technical Report TR-93-92, 1993.
- [24] H. Choi, H.-A. Choi, and M. Azizoglu. Efficient Scheduling of Transmissions in Optical Broadcast Networks. *IEEE/ACM Transaction on Networking*, 4(6) :913–920, December 1996.
- [25] J. Cohen, E. Jeannot, and N. Padoy. Messages Scheduling for Data Redistribution between Clusters. In *Algorithms, models and tools for parallel computing on heterogeneous networks (HeteroPar'03) workshop of SIAM PPAM 2003, LNCS 3019*, pages 896–906, Czestochowa, Poland, September 2003.
- [26] P. Crescenzi, D. Xiaotie, and C. H. Papadimitriou. On Approximating a Scheduling Problem. 5 :287–297, 2001.
- [27] A. Denis, C. Pérez, T. Priol, and A. Ribes. Parallel corba objects for programming computational grids. *Distributed Systems Online*, 4(2), February 2003. Electronic journal.
- [28] F. Desprez, J. Dongarra, A. Petitet, C. Randriamaro, and Y. Robert. Scheduling block-cyclic array redistribution. In E. H. D'Hollander, G. R. Joubert, F. J. Peters, and U. Trottenberg, editors, *Parallel Computing : Fundamentals, Applications and New Directions, Proceedings of the Conference ParCo'97, 19-22 September 1997, Bonn, Germany*, volume 12, pages 227–234, Amsterdam, 1998. Elsevier, North-Holland.
- [29] V. Van Dongen, C. Bonello, and G. R. Gao. Data parallelism with high performance c. In *CASCON '94 : Proceedings of the 1994 conference of the Centre for Advanced Studies on Collaborative research*, page 69. IBM Press, 1994.
- [30] N. Doss, W. Gropp, E. Lusk, and A. Skjellum. A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. Technical report, Argonne National Laboratory, 1996.
- [31] G. Egan. *La cité des permutants*. Livre de poche, 1999.
- [32] A. Esnard. Modèle pour la redistribution de données complexes. 16ème Rencontres francophones du parallélisme (RenPar'16), 2005.
- [33] W. Feng, M. Fisk, M. K. Gardner, and E. Weigle. Dynamic right-sizing : An automated, lightweight, and scalable technique for enhancing grid performance. In *PIHSN '02 : Proceedings of the 7th IFIP/IEEE International Workshop on Protocols for High Speed Networks*, pages 69–83, London, UK, 2002. Springer-Verlag.
- [34] M. Fisk and W. Feng. Dynamic adjustment of tcp window sizes, 2000.
- [35] I. Foster and C. Kesselman. The Globus project : A progress report. In *Heterogeneous Computing Workshop*, March 1998.
- [36] I. Foster and C. Kesselman. *The Grid : Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1998.

- 
- [37] G. Fox, S. Hiranandani, K. Kennedy, C. Koebel, U. Kremer, C.-W. Tseng, and M.-Y. Wu. Fortran D language specification. Technical Report CRPC-TR90079, Houston, TX, December 1990.
- [38] A. Ganz and Y. Gao. A time-wavelength assignment algorithm for a wdm star network. In *INFOCOM*, pages 2144–2150, 1992.
- [39] G.A. Geist, J.A. Kohl, and P.M. Papadopoulos. CUMULVS : Providing Fault-Tolerance, Visualization and Steering of Parallel Applications. *International Journal of High Performance Computing Applications*, 11(3) :224–236, August 1997.
- [40] T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *J. ACM*, 23 :665–679, 1976.
- [41] I.S. Gopal, G. Bongiovanni, M.A. Bonuccelli, D.T. Tang, and C.K. Wong. An Optimal Switching Algorithm for Multibeam Satellite Systems with Variable Bandwidth Beams. *IEEE Transactions on Computers*, COM-30(11) :2475–2481, November 1982.
- [42] M. Guo and I. Nakata. A framework for efficient data redistribution on distributed memory multicomputers. *J. Supercomput.*, 20(3) :243–265, 2001.
- [43] S. Sekiguchi H. Nakada, M. Sato. Design and Implementations of Ninf : Towards a Global Computing Infrastructure. *Future Generation Computing Systems, Metacomputing Issue*, 15 :649–658, 1999.
- [44] High Performance Fortran Forum. High Performance Fortran language specification, version 1.0. Technical Report CRPC-TR92225, Houston, Tex., 1993.
- [45] S. Hiranandani, K. Kennedy, and C. Tseng. Evaluation of compiler optimizations for fortran d on mimd distributed memory machines. In *ICS*, pages 1–14, 1992.
- [46] C. Hsu, Y. Chung, D. Yang, and C. Dow. A generalized processor mapping technique for array redistribution. *IEEE Trans. Parallel Distrib. Syst.*, 12(7) :743–757, 2001.
- [47] E. Jeannot. Improving Middleware Performance with AdOC : an Adaptive Online Compression Library for Data Transfer. In *International Parallel and Distributed Processing Symposium 2005 (IPDPS'05)*, Denver, Colorado, USA, April 2005.
- [48] E. Jeannot and F. Wagner. Message scheduling for data redistribution through high performance networks. Technical Report RR-5077, INRIA, 2004.
- [49] K. Keahey, P. Fasel, and S. Mniszewski. PAWS : Collective Interactions and Data Transfers. In *10th IEEE Intl. Symp. on High Perf. Dist. Comp. (HPDC-10'01)*, pages 47–54, August 2001.
- [50] N. P. Kronenberg, H. M. Levy, and W. D. Strecker. Vaxcluster : a closely-coupled distributed system. *ACM Trans. Comput. Syst.*, 4(2) :130–146, 1986.
- [51] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2 :83–97, 1955.
- [52] Oak Ridge National Labs. Mxn. <http://www.csm.ornl.gov/cca/mxn>.
- [53] T. Lakshman and U. Madhow. The performance of tcp/ip for networks with high bandwidth-delay products and random loss, 1997.
- [54] D. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. Sequencing and scheduling : Algorithms and complexity. In S. C. Graves, A. H. G. Rinnooy Kan, and P. H. Zipkin, editors, *Handbooks in Operations Research and Management Science, Vol 4., Logistics of Production and Inventory*, pages 445–522. North-Holland, 1993.

- [55] A. Legrand, L. Marchal, and Y. Robert. Optimizing the steady-state throughput of scatter and reduce operations on heterogeneous platforms. *J. Parallel and Distributed Computing*, 2005. to appear.
- [56] V. K. Naik, S. Sivasubramanian, D. F. Bantz, and S. Krishnan. Harmony : A desktop grid for delivering enterprise computations. In *GRID*, pages 25–33, 2003.
- [57] S. Ogura, S. Matsuoka, and H. Nakada. Evaluation of the inter-cluster data transfer on grid environment. In *CCGRID* [9], pages 374–381.
- [58] N. Park, V. K. Prasanna, and C. Raghavendra. Efficient algorithms for block-cyclic array redistribution between processor sets. In *Supercomputing '98 : Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM)*, pages 1–13, Washington, DC, USA, 1998. IEEE Computer Society.
- [59] L. Prylli and B. Tourancheau. Efficient block cyclic data redistribution. In *Euro-Par, Vol. I*, pages 155–164, 1996.
- [60] A. Ribes. *Contribution à la conception d'un modèle de programmation parallèle et distribué et sa mise en oeuvre au sein de plates-formes orientées objet et composant*. Thèse de doctorat, IRISA, Université de Rennes 1, IRISA, Rennes, France, December 2004.
- [61] A. Schrijver. Bipartite edge coloring in  $o(\Delta m)$  time. *SIAM J. Comput.*, 28(3) :841–846, 1998.
- [62] S. Shankaran, J. J. Alonso, M. Liou, N. Liu, and R. Davis. A multi-code-coupling interface for combustor/turbomachinery simulations.
- [63] H. Sivakumar, S. Bailey, and R. L. Grossman. Psockets : the case for application-level network striping for data intensive applications using high speed wide area networks. In *Supercomputing '00 : Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, page 37, Washington, DC, USA, 2000. IEEE Computer Society.
- [64] V. A. Strusevich. A greedy open shop heuristic with job priorities. *Ann. Oper. Res.*, 83 :253–270.
- [65] D. Thain and M. Livny. Building reliable clients and servers. In Ian Foster and Carl Kesselman, editors, *The Grid : Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2003.
- [66] P. M. Vaidya. An algorithm for linear programming which requires  $o((m+n)nz+(m+n)ln)$  arithmetic operations. In *STOC '87 : Proceedings of the nineteenth annual ACM conference on Theory of computing*, pages 29–38, New York, NY, USA, 1987. ACM Press.
- [67] D. W. Walker and S. W. Otto. Redistribution of block-cyclic data distributions using MPI. *Concurrency : Practice and Experience*, 8(9) :707–728, 1996.
- [68] D. Werra and J. Blazewicz. Some preemptive open shop scheduling problems with a renewable or a nonrenewable resource. *Discrete Applied Mathematics*, 43(1) :103,104, 1993.
- [69] D. Williamson, L. Hall, J. Hoogeveen, C. A. J. Hurkens, J. Lenstra, and D. Shmoys. Short shop schedules, 1993.
- [70] R. Wolski, N. T. Spring, and J. Hayes. The network weather service : a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5–6) :757–768, 1999.

## Résumé

Nous considérons ici le problème où deux programmes différents situés sur deux grappes d'ordinateurs distantes, reliées par un réseau à haut débit, forment un couplage de code et échangent régulièrement des données. Un tel échange s'effectue par une redistribution de données. Nous étudions comment effectuer une telle redistribution le plus efficacement possible en minimisant temps de communication et congestion du réseau.

Nous utilisons pour ce faire, une modélisation du problème à l'aide de graphes bipartis. Le modèle choisi permet une prise en compte du délai d'initialisation des communications, des différentes bandes passantes et impose une limite d'une communication simultanée par interface réseau (modèle 1-port) et de  $k$  communications simultanées sur la dorsale. Nous effectuons une validation expérimentale du modèle puis l'utilisons pour développer deux algorithmes d'ordonnancement des communications. Nous montrons que chacun d'entre eux est un algorithme d'approximation garantissant un temps d'exécution dans le pire des cas  $\frac{8}{3}$  fois plus élevé que le temps optimal. Nous concluons l'étude de ces algorithmes par une série d'expériences démontrant de bonnes performances en pratique.

Enfin, nous étendons le problème initial au cas de grappes hétérogènes : ce cas imposant de sortir du modèle 1-port, nous montrons comment modifier nos algorithmes pour en tirer parti. Nous étudions également le cas de redistributions exécutées en régime permanent sur un réseau d'une topologie plus complexe autorisant les communications locales.

**Mots-clés:** redistribution, ordonnancement de messages, PBS, couplage de code, algorithmes d'approximation

## Abstract

We consider here the case of a code-coupling application consisting of two different programs, located on two different clusters linked by a high speed network, which regularly exchange data through a data redistribution. We study how to achieve this redistribution in an efficient way while minimizing transfer time and network congestion.

We use to reach this goal a modeling of the problem using bipartite graphs. The chosen model allows us to take into account communications setup delays, the different available bandwidths and impose a limit of one simultaneous communication on each network interface (1-port model) and  $k$  simultaneous communications on the backbone. We execute an experimental validation of this model and use it to develop two messages scheduling algorithms. We show that each of them is an approximation algorithm thus guaranteeing results with execution times no worse than  $\frac{8}{3}$  times optimal times. We conclude the study of these algorithms by experiments showing good performances in practice.

Finally, we extend the initial problem to the case of heterogeneous clusters : we show that this case imposes us to lift the one-port constraint in order to achieve performance and we show how to adapt our algorithms accordingly. We also study the case of redistributions executed under steady state on networks with more complex topologies allowing local communications.

**Keywords:** redistribution, messages scheduling, PBS, code coupling, approximation algorithms

