B&B@Grid : une approche efficace pour la gridification d'un algorithme Branch and Bound

Mohand Mezmaz, Nouredine Melab, El-Ghazali Talbi

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

B&B@Grid : une approche efficace pour la gridification d'un algorithme Branch and Bound (traduire)

Mohand Mezmaz, Nouredine Melab — El-Ghazali Talbi

N° 6937

Mai 2009

Thème NUM

ISSN 0249-6399 ISRN INRIA/RR--6937--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
LILLE - NORD EUROPE

B&B@Grid : une approche efficace pour la gridification d'un algorithme Branch and Bound (traduire)

Mohand Mezmaz, Nouredine Melab , El-Ghazali Talbi

Thème NUM — Systèmes numériques
Équipes-Projets Dolphin

Rapport de recherche n° 6937 — Mai 2009 — 32 pages

Abstract: The exact resolution of large instances of combinatorial optimization problems, such as scheduling problems, is a real challenge for grid computing. Indeed, it is necessary to reconsider the resolution algorithms and take into account the characteristics of such environments, especially large scale, heterogeneity and dynamic availability of resources, and their multi-domain administration.

In this paper, we propose an adaptation of the parallel *Branch and Bound* algorithm for computational grids. Such gridification is based on new ways to efficiently deal with some crucial issues, mainly dynamic adaptive load balancing, fault tolerance, global information sharing and termination detection of the algorithm. A new efficient coding of the work units (search sub-trees) distributed during the exploration of the search tree is proposed to optimize the involved communications.

This approach, much more efficient in terms of communication and checkpointing cost than the best known approaches in the literature, allowed the exact resolution on a nation-wide grid(Grid5000 and different educational networks of University of Lille1) of a standard Flow-Shop problem instance remained unresolved over the past fifteen years.

Key-words: Branch and Bound, Parallel Computing, Grid Computing, Flow-Shop Problem, Performance Evaluation.

Centre de recherche INRIA Lille – Nord Europe
Parc Scientifique de la Haute Borne
40, avenue Halley, 59650 Villeneuve d'Ascq
Téléphone : +33 3 59 57 78 00 — Télécopie : +33 3 59 57 78 50

B&B@Grid : une approche efficace pour la gridification d'un algorithme Branch and Bound

Résumé : La résolution exacte de problèmes d'optimisation combinatoire de grande taille, tels que les problèmes d'ordonnancement, constitue un vrai défi pour les grilles informatiques. En effet, il est nécessaire de repenser les algorithmes de résolution pour prendre en compte les caractéristiques de tels environnements, notamment leur grande échelle, l'hétérogénéité et la disponibilité dynamique de leurs ressources, et leur nature multi-domaine d'administration.

Dans cet article, nous proposons une nouvelle approche de passage sur grilles de calcul des méthodes exactes de type Branch-and-Bound appelée B&B@Grid. Cette approche est basée sur un codage des unités de travail (sous problèmes) sous forme d'intervalles permettant de minimiser le coût des communications induites par les opérations de régulation de charge, de tolérance aux pannes et de détection de la terminaison.

Cette approche, beaucoup plus performante en terme de coût de communication et de sauvegarde que les meilleures approches connues dans la littérature, a permis la résolution optimale sur la grille nationale Grid'5000 d'une instance standard du problème du Flow-Shop restée non résolue depuis une quinzaine d'années. Le Flow-Shop est l'un des problèmes d'ordonnancement les plus étudiés.

Mots-clés : Branch and Bound, Calcul Parallèle, Grilles de Calcul, Problème du Flow-Shop, Evaluation de performance.

Contents

1 Introduction

De nombreux problèmes, rencontrés dans différents secteurs économiques, sont de nature combinatoire. Ce type de problèmes requiert de trouver la ou les meilleure(s) configuration(s) parmi un ensemble fini mais très grand de configurations possibles. Chacune des configurations constitue une *solution* au problème considéré et appartient à un espace appelé *espace des solutions*. Le choix de la solution est fonction de son coût. Résoudre un *problème d'optimisation combinatoire* avec une méthode exacte revient à trouver la ou les solution(s) de coût *optimal*.

Les méthodes de résolution exactes utilisées en optimisation combinatoire sont, pour un bon nombre d'entre elles, de type B&B. Ces méthodes se déclinent essentiellement en trois variantes : B&B simple, le *Branch-and-Cut* (B&C), et le *Branch-and-Price* (B&P). Il existe d'autres variantes du B&B moins connues. Certains auteurs considèrent les algorithmes B&C et B&P, ainsi que les autres variantes, comme des algorithmes B&B distincts. Ces auteurs parlent alors de B&X pour désigner les algorithmes B&B, B&C, B&P, etc. Dans la suite du papier, l'algorithme B&B désigne B&B simple lui-même ou toute autre variante de cet algorithme.

Les algorithmes B&B reposent sur une exploration implicite de toutes les solutions du problème considéré. L'espace de ces solutions est exploré en construisant dynamiquement un arbre dont le sous problème racine représente la totalité de cet espace, les sous problèmes feuilles sont les différentes solutions possibles, et les sous problèmes internes sont des sous-espaces de l'espace global des solutions. La taille de ces sous-espaces est de plus en plus réduite à mesure qu'on s'approche des feuilles.

La construction d'un tel arbre et son exploration se font à l'aide de quatre opérateurs. Ce sont les opérateurs de *décomposition*, d'*évaluation*, de *sélection* et d'*élimination*. L'algorithme procède en plusieurs itérations, durant lesquelles la meilleure solution trouvée est conservée et améliorée au fur et à mesure de l'exploration. Les sous problèmes générés et non encore traités sont conservés dans une liste dont le contenu initial est le sous problème racine. A chaque itération de l'algorithme, l'*opérateur de sélection* choisit, selon une certaine stratégie, un sous problème, autrement dit un sous-espace, de cette liste. L'*opérateur de décomposition* le divise en plusieurs sous-espaces plus petits et deux à deux disjoints. L'*opérateur d'évaluation* calcule une borne des solutions de chaque sous-espace généré, et l'*opérateur d'élimination* élimine de la liste tout sous-espace dont l'évaluation a montré qu'il ne contient pas de solution pouvant améliorer la meilleure solution déjà trouvée.

Relativement à une énumération exhaustive de toutes les solutions, ces algorithmes réduisent considérablement la puissance de calcul nécessaire pour résoudre un problème d'optimisation combinatoire. Néanmoins, cette puissance reste considérable lorsqu'il s'agit de résoudre des instances de très grande taille. C'est pourquoi un recours aux grilles de calcul est nécessaire pour la résolution de telles instances. Les ressources d'une grille sont en général hétérogènes, volatiles, distribuées sur plusieurs domaines d'administration, et interconnectées avec un réseau de grande taille. Le passage des algorithmes B&B sur grilles de calcul nécessite la prise en compte de ces caractéristiques.

Pendant une résolution, l'irrégularité de l'arbre exploré par les méthodes exactes, l'hétérogénéité des grilles informatiques et leur volatilité impliquent

un nombre considérable d'opérations de régulation de charge et de sauvegarde/restauration. Toutes ces opérations se traduisent par des coûts de communication exorbitants pour le transfert, le stockage, et la restauration des unités de travail. Ces coûts sont accentués par l'échelle d'une grille et l'importance des délais de communication dans cet environnement. Il est donc primordial de proposer un codage de ces unités afin de réduire leur taille, et ainsi minimiser les coûts de communication. Il est également nécessaire de réduire le nombre de communications par le biais de leur mutualisation. Cela signifie qu'un message peut remplir plusieurs rôles à la fois.

Dans ce papier, nous proposons une méthode de passage sur grilles(gridification) de l'algorithme *Branch-and-Bound* (*B&B*) basée sur un codage d'une unité de travail, souvent composée d'une liste de sous problèmes, par un intervalle, défini avec seulement deux entiers naturels. Dans cette nouvelle approche, appelée B&B@Grid, l'information transférée, stockée et restaurée est un intervalle au lieu d'une liste de sous problèmes. Par conséquent, ce codage permet de minimiser le coût de communication, et rend les stratégies de régulation de charge, de tolérance aux pannes, et de détection de terminaison plus efficaces. De plus, à l'inverse des autres approches de régulation de charge publiées dans la littérature [3, 1, 11], la granularité des unités de travail dans B&B@Grid n'est pas fixe, mais varie selon les caractéristiques de la grille.

Cet article est organisé en quatre sections. La **section 2** résume les principaux travaux effectués sur la gridification des algorithmes B&B afin de les adapter aux caractéristiques des grilles. La **section 3** est consacrée à la présentation de B&B@Grid. Cette section décrit les concepts ainsi que les opérateurs définissant cette nouvelle approche. La **section 4** détaille ses stratégies de tolérance aux pannes, d'équilibrage de charge et de détection de terminaison. Dans la **section 5**, nous présentons les résultats expérimentaux obtenus. L'article se termine par les conclusions tirées de ces travaux ainsi que leurs perspectives.

2 Etat de l'art

Cette section présente les principales approches proposées dans la littérature pour réduire les délais de communication, partitionner la charge entre les processus B&B, gérer la tolérance aux pannes et prendre en charge le passage à l'échelle.

2.1 Optimisation du coût de communication

Les délais de communication dans une grille de calcul d'échelle nationale ou internationale sont plus longs que dans une grappe de machines. Les délais de communication sont l'une des sources majeures de perte d'efficacité dans toute approche parallèle du B&B. L'article [12] présente des résultats expérimentaux illustrant les coûts de communication lors de la parallilisation d'un algorithme B&B sur un réseau de grande échelle. Ces résultats sont comparés avec leurs équivalents dans une grappe de machines locale. La réduction de la taille des messages échangés est donc une des solutions qui améliore l'efficacité parallèle d'un B&B. Les messages communiqués dans un B&B parallèle sont constitués en grande partie de sous problèmes. L'approche de PICO est basée sur une

représentation des sous problèmes permettant de réduire leur taille. Dans cette approche, seulement les jetons sont communiqués au lieu des sous problèmes. Un jeton est une information permettant d'identifier et de localiser un seul sous problème. La taille d'un jeton est celle d'un entier, codé en général sur 48 bits, nettement inférieure donc à la taille complète d'un sous problème. Un jeton permet juste de localiser un sous problème pour le récupérer ensuite et le traiter. Il est impossible de déduire un sous problème à partir de son jeton. A notre connaissance, l'approche à jeton est celle qui réduit le plus la taille des sous problèmes communiqués.

Les résultats expérimentaux présentés dans cet article montrent que notre approche B&B@Grid réduit davantage la taille des sous problèmes communiqués. Contrairement à PICO, B&B@Grid peut réduire tout un ensemble de sous problèmes. Le codage de PICO concerne un sous problème à la fois. PICO attribue autant de codes que de sous problèmes, tandis que B&B@Grid peut codifier un ensemble de sous problèmes avec le même code.

2.2 Tolérance aux pannes

La dynamicité et volatilité des grilles requièrent, à toute approche de parallélisation du B&B, la définition d'une stratégie de tolérance aux pannes. Dans un algorithme B&B, les sous problèmes en cours de traitement représentent en grande partie l'état global d'une résolution. En les sauvegardant, il est toujours possible de réinitialiser une résolution en cas de panne et de la relancer à partir de son dernier point de sauvegarde. Réduire la taille de ces sous problèmes permet d'améliorer une stratégie de tolérance aux pannes. Les auteurs de [10] proposent une approche de codage des sous problèmes basée sur leur représentation en fonction de leur position dans l'arbre de B&B. L'intérêt de cette représentation est de permettre de retrouver un sous problème à partir de son codage. A notre connaissance, le codage proposé dans [10] est celui qui réduit le plus la taille des sous problèmes sauvegardés dans une stratégie de tolérance aux pannes.

L'approche [10] exige que tous les sous problèmes appartiennent au même sous-arbre pour être désignés avec le même code. Par contre, B&B@Grid peut représenter, avec un même code, un ensemble de sous problèmes même s'ils n'appartiennent pas au même sous-arbre. En outre, les expérimentations montrent que B&B@Grid réduit davantage la taille des sous problèmes sauvegardés que l'approche proposée dans [10].

2.3 Partitionnement et régulation de charge

La nature hétérogène et dynamique des grilles de calcul rend la définition de la répartition de la charge entre les processus travailleurs plus difficile. Les stratégies d'équilibrage de charge utilisées dans la parallélisation du B&B sont souvent basées sur le paradigme fermier-travailleur. Ces stratégies sont appliquées à différents problèmes d'optimisation combinatoire. L'approche publiée dans [3] est certainement l'une des plus abouties pour la parallélisation du B&B selon le paradigme fermier-travailleur. Les efficacités parallèles enregistrées dans cette approche (de 87% à 92%) sont nettement meilleures que celles enregistrées par beaucoup d'autres approches de la littérature. Une autre approche avec le

paradigme fermier-travailleur est publiée dans [1]. Dans cette approche, le fermier envoie à chaque processus travailleur un seul sous problème. Un processus travailleur opère alors au plus N décompositions pour générer un certain nombre de sous problèmes, élimine tout sous problème dont la borne indique qu'il ne peut améliorer la meilleure solution connue, et renvoie au fermier les sous problèmes non éliminés. BoB++ [11] peut également être utilisé avec le paradigme fermier-travailleur. Dans BoB++, un processus travailleur opère une seule décomposition du sous problème reçu et renvoie les sous problèmes obtenus au processus fermier.

L'inconvénient des approches proposées dans la littérature est la granularité fixe des unités de travail allouées. Or, dans une grille, la taille d'une unité de travail doit dépendre de la taille de la grille, de la nature hétérogène de ses ressources, et de leurs disponibilités variables. En effet, à l'inverse des autres stratégies, la taille d'une unité de travail dans B&B@Grid dépend du nombre de processeurs de la grille, de leur puissance et de leur disponibilité. La granularité du travail attribué à un processeur puissant est plus grande que celle d'un processeur moins puissant. Cette stratégie tient également compte de la disponibilité d'un processeur puisque B&B@Grid peut être utilisé avec le modèle de vol de cycles. Dans un tel modèle, les processeurs ne sont exploités que pendant la période où ils sont oisifs. Dans B&B@Grid, le travail est également réparti entre tous les processeurs quelque soit la taille de la grille utilisée. Les expérimentations montrent d'ailleurs que B&B@Grid exploite à plein régime les processeurs d'une grille.

2.4 Passage à l'échelle

Un des plus grands inconvénients de l'utilisation du paradigme fermier-travailleur dans une grille de calcul est le goulot d'étranglement que peut constituer le fermier lorsque le nombre de processus travailleurs devient important. Dans une grille, ce risque est d'autant plus accentué que cet environnement peut contenir un grand nombre de processeurs. Toutefois, une hiérarchisation des fermiers permet de pousser les limites de ce paradigme pour supporter davantage de processeurs travailleurs. C'est ce que propose les approches ALPS[16], PICO, BOB++ et celle décrite dans [2]. Le paradigme pair-à-pair peut supporter beaucoup plus de processeurs et pallier au goulot d'étranglement des approches centralisées. DIB [8] et l'approche publiée dans [10] font partie des travaux basés sur la parallélisation du B&B selon le paradigme pair-à-pair.

B&B@Grid peut être utilisée avec les paradigmes fermier-travailleur, hiérarchique et pair-à-pair. Grâce à la réduction de la taille des sous problèmes, B&B@Grid permet de repousser davantage les limites de ces paradigmes. En effet, les expérimentations décrites dans ce papier montrent que l'utilisation d'un simple paradigme centralisé, sans aucune hiérarchisation, permet l'exploitation de plusieurs milliers de processeurs sans goulot d'étranglement au niveau du fermier.

3 L'approche B&B@Grid

Cette section décrit les concepts et les opérateurs définissant B&B@Grid.

3.1 Principe de B&B@Grid

Cette nouvelle approche utilise la notion de liste de sous problèmes actifs. Les sous problèmes actifs d'un B&B sont ceux générés mais non encore traités. Au cours d'une résolution, cette liste évolue constamment et l'algorithme s'arrête lorsqu'elle devient vide. Comme l'indique la figure 1, une liste de sous problèmes actifs couvre un certain ensemble de sous problèmes de l'arbre. Cet ensemble est constitué de tous les sous problèmes pouvant être explorés à partir des sous problèmes de cette liste. Le nombre de sous problèmes couverts par les sous problèmes actifs diminue au fil d'une résolution, tandis que le nombre de sous problèmes implicitement ou explicitement explorés augmente. Un sous problème est implicitement exploré lorsqu'un algorithme B&B conclut, sans le visiter, qu'il ne peut pas contenir de solutions optimales. Par contre, un sous problème visité est dit explicitement exploré. Au début d'une résolution, l'ensemble des sous problèmes actifs ne contient que le problème racine, tous les autres sous problèmes de l'arbre sont couverts par le problème racine. A la fin d'un calcul, tous les sous problèmes de l'arbre du B&B appartiennent à l'ensemble des sous problèmes implicitement ou explicitement visités.

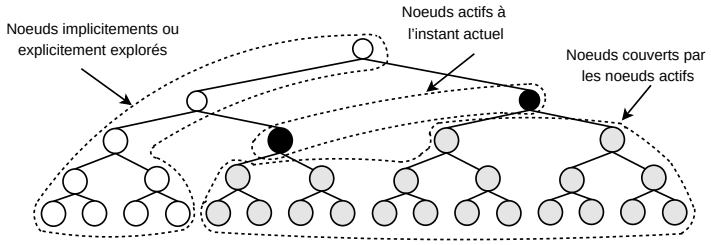

Figure 1: sous problèmes couverts par un ensemble de sous problèmes actifs.

Le principe de l'approche est basé sur l'assignation d'un numéro à chaque sous problème de l'arbre. Cette assignation est telle que tous les numéros de toute liste de sous problèmes actifs constitue un intervalle d'entiers. L'approche définit ainsi une relation d'équivalence entre le concept de liste de sous problèmes actifs et celui d'intervalle de numéros de sous problèmes. La connaissance de l'un doit permettre de déduire de façon unique l'autre. Etant donnée sa petite taille, l'intervalle est utilisé pour les communications et pour les sauvegardes, tandis que la liste des sous problèmes actifs est utilisée pour l'exploration. Afin de passer de l'un à l'autre des deux concepts, l'approche définit deux opérateurs supplémentaires de **pliage** et **dépliage** en complément des quatre opérateurs de base des algorithmes B&B. L'opérateur de pliage déduit un intervalle de numéros à partir d'une liste de sous problèmes actifs. L'opérateur de dépliage retrouve une liste de sous problèmes actifs à partir d'un intervalle. Pour définir ces deux opérateurs, trois nouveaux concepts sont introduits. Ce sont les concepts de **numéro**, de **poids** et de **portée** d'un sous problème.

Les opérateurs de *pliage* et *dépliage* sont définis à l'aide du concept de *portée* d'un sous problème. Les concepts de *numéro* et de *poids* d'un sous problème sont utilisés pour définir la *portée* d'un sous problème. Le concept de *poids* est employé pour définir le *numéro* d'un sous problème. Dans cette section, nous présentons successivement donc le *poids* d'un sous problème, son *numéro*, sa *portée*, l'opérateur de *pliage* et l'opérateur de *dépliage*. Chaque concept est

présenté indépendamment de la structure de l'arbre du B&B, et donc du problème combinatoire sous-jacent. Toutefois, une définition plus concise est donnée pour les arbres de certains problèmes d'optimisation combinatoire, tels que le QAP, le TSP ou le Flow-Shop. Ces problèmes sont très étudiés dans la littérature de l'optimisation combinatoire. En plus des deux opérateurs de pliage et dépliage, B&B@Grid définit également deux types de processus. Il s'agit du **processus coordinateur** et du **processus B&B**. Ces deux processus permettent de paralléliser un B&B avec tout paradigme. Cette section se termine donc par la présentation de ces deux processus.

3.2 Poids d'un sous problème

Le poids d'un sous problème n de l'arbre, noté *poids(n)*, est le nombre de feuilles du sous-arbre dont il est la racine. La formule (1) définit le poids d'un sous problème d'une façon récursive. Comme l'indique la formule (1), le poids d'une feuille est égal à 1, et le poids d'un sous problème interne est égal à la somme des poids de ses sous problèmes fils. Cette définition est à la fois générale et inapplicable directement. En effet, elle est générale puisque elle définit le poids d'un sous problème pour tout arbre, et inapplicable puisque la taille de l'arbre est exponentielle.

$$\text{poids}(n) = \begin{cases} 1 \;\; si \;\; \text{fils}(n) = \phi \\ \sum_{i \in \text{fils}(n)} \text{poids}(i) \quad \text{sinon} \end{cases} \tag{1}$$

Cependant, la connaissance de la structure d'un arbre permet de simplifier cette définition. Les formules (2) et (3) définissent, d'une façon plus simple, le poids d'un sous problème n pour respectivement un arbre binaire et un arbre de permutation. Dans ces deux définitions, la profondeur d'un sous problème n, notée *profondeur(n)*, est le nombre de sous problèmes se trouvant sur le même chemin qui le séparent de la racine de l'arbre. P est la profondeur associée aux feuilles de l'arbre. L'arbre binaire est celui où, hormis les feuilles, chaque sous problème a deux sous problèmes fils. L'arbre de permutation est celui associé aux problèmes où il est question de retrouver une permutation parmi un ensemble fini d'éléments.

$$\text{poids}(n) = 2^{(P - \text{profondeur}(n))} \tag{2}$$

$$\text{poids}(n) = (P - \text{profondeur}(n))! \tag{3}$$

Plusieurs problèmes d'optimisation combinatoire peuvent être représentés par un arbre de permutation. Dans ce type d'arbre, tout sous problème n, hormis le sous problème racine, respecte la condition (3). A l'instar de l'arbre binaire, de l'arbre de permutation, et de tout autre arbre de structure régulière, les sous problèmes de même profondeur ont le même poids. Par conséquent, au lieu d'associer les poids aux sous problèmes, il est plus simple de leur associer leurs profondeurs, et de déduire le poids d'un sous problème à partir de sa profondeur. Ainsi au début de l'algorithme B&B, un vecteur qui donne le poids associé à chaque profondeur est calculé. A l'aide de ce vecteur, il est possible de retrouver le poids d'un sous problème en connaissant sa profondeur. La

figure 2 montre un exemple des poids associés aux profondeurs dans un arbre de permutation.

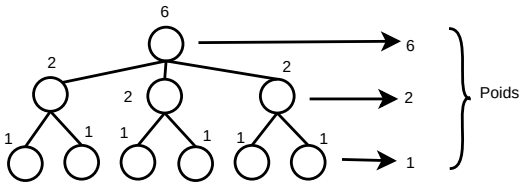

Figure 2: Poids d'un sous problème.

3.3 Numéro d'un sous problème

A chaque sous problème n de l'arbre, on associe un numéro noté *numéro(n)*. Comme l'indique la formule (4), le numéro d'un sous problème n peut être obtenu à l'aide des sous problèmes de son chemin. Le chemin d'un sous problème n, noté *chemin*(n), est l'ensemble des sous problèmes rencontrés en allant de la racine de l'arbre au sous problème n. Le sous problème n et la racine de l'arbre sont donc toujours inclus dans *chemin*(n). Pour retrouver le numéro d'un sous problème n, il suffit alors de connaître les précédents de chaque sous problème du chemin de n. Les précédents d'un sous problème n, noté *précédents(n),* est l'ensemble des sous problèmes frères de n qui sont générés avant n.

$$\text{numéro}(n) = \sum_{i \in \text{chemin}(n)} \sum_{j \in \text{précédents}(i)} \text{poids}(j) \tag{4}$$

La définition (4) s'applique à tout arbre indépendamment de sa structure. La formule (5) donne une définition plus simple pour les arbres de structure régulière tels que les arbres binaires ou les arbres de permutation. Cette définition repose sur le constat que, dans ce type d'arbre, les sous problèmes de même profondeur ont un poids similaire. Pour avoir le numéro d'un sous problème, il suffit donc de connaître son chemin et le rang de chaque sous problème de ce chemin. Le rang d'un sous problème n, noté *rang*(n), est la position de n parmi ses sous problèmes-frères. Ainsi, lors de la génération des sous problèmes-fils d'un sous problème quelconque, le rang du premier sous problème généré est égal à 0, le rang du deuxième sous problème généré est égal à 1, et ainsi de suite. La figure 4 montre un exemple illustrant les numéros obtenus dans un arbre de permutation.

$$\text{numéro}(n) = \sum_{i \in \text{chemin}(n)} \text{rang}(i) * \text{poids}(i) \tag{5}$$

3.4 Portée d'un sous problème

La portée d'un sous problème n, notée *portée*(n), est l'intervalle auquel appartiennent les numéros des sous problèmes du sous-arbre dont n est la racine. La figure 5 donne un exemple de la portée associée à chaque sous problème d'un arbre de permutation. Comme l'indique la formule (6), le début de la portée

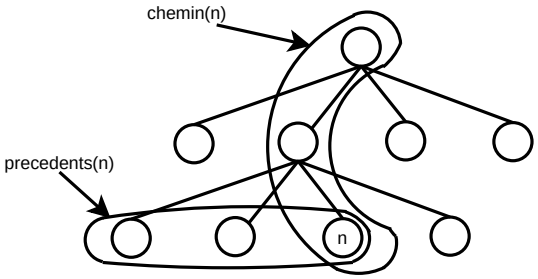

Figure 3: Chemin et précédents d'un sous problème.

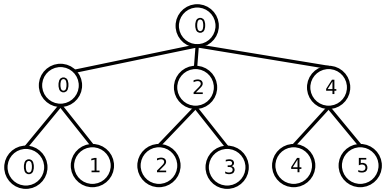

Figure 4: Numéro d'un sous problème.

d'un sous problème est égal à son numéro, et sa fin est égale à la somme de son numéro et de son poids.

$$\text{portée}(n) = \quad [\text{numéro}(n), \text{numéro}(n) + \text{poids}(n)[\tag{6}$$

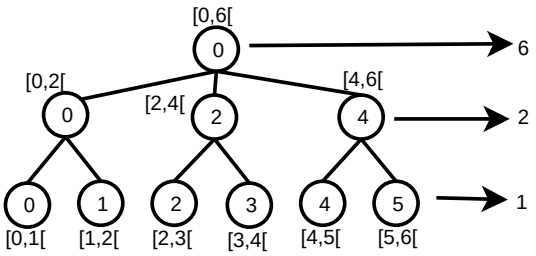

Figure 5: Portée d'un sous problème.

3.5 Opérateur de pliage

Le rôle de cet opérateur est de déduire, à partir d'une liste N de sous problèmes actifs, l'intervalle auquel appartiennent les numéros de sous problèmes pouvant être explorés en partant des sous problèmes de N. Cet intervalle est noté $Fold(N)$. Comme l'indique la formule (7), ceci revient à calculer l'union de toutes les portées des sous problèmes de N.

$$\text{Fold}(N) = \cup_{i \in N} \ \text{portée}(i) \tag{7}$$

Dans un B&B, la disposition des sous problèmes actifs dans une liste N dépend de la stratégie de parcours adoptée par l'opérateur de sélection. Soient

N_1, N_2 ... N_k les rangs par lesquels ces sous problèmes sont ordonnés, et [A_1,B_1[, [A_2,B_2[... [A_k,B_k[leurs portées respectives. La condition (8) est toujours vérifiée lorsque la stratégie de parcours est de type **profondeur d'abord**. Ainsi, *intervalle*(N) peut être retrouvé sans calculer toutes les portées des sous problèmes actifs. Comme l'indique la formule (9), il suffit de connaître les portées de A_1 et A_k, ou plus simplement, il suffit de connaître les numéros de ces deux sous problèmes et le poids de A_k. La figure 6 montre un exemple illustrant le passage d'une liste de sous problèmes actifs vers un intervalle en utilisant l'opérateur de pliage.

$$\forall\ i < k\ \ B_i = A_{i+1} \tag{8}$$

$$\text{Fold}(N) = \quad [\text{numéro}(A_1), \text{numéro}(A_k) + \text{poids}(A_k)[\tag{9}$$

3.6 Opérateur de dépliage

Cet opérateur est chargé de déduire, à partir d'un intervalle [A,B[, une liste de sous problèmes actifs notée $Unfold([A, B[)$. Comme l'indique la formule (10), *Unfold*([A,B[) est constitué des sous problèmes de l'arbre dont la portée est incluse dans [A,B[, et dont la portée de leur père n'est pas incluse dans [A,B[. Ces deux conditions garantissent que $Unfold([A, B[)$ est une liste unique et minimale. En effet, il est impossible de trouver une autre liste dont la cardinalité est inférieure à $Unfold([A, B[)$, et qui permet d'explorer les sous problèmes dont le numéro appartient à [A,B[. La cardinalité d'une liste est le nombre d'éléments qu'elle contient.

$$\text{Unfold}([A, B[) = \left\{ \begin{array}{c} n/ \\ \text{portée}(n) \subseteq [A, B[\\ \text{portée}(p) \not\subset [A, B[\\ p = \text{père}(n) \end{array} \right\} \tag{10}$$

$$\text{élimination}(n) = \left(\begin{array}{c} \text{portée}(n) \subseteq [A, B[\\ ou \\ (\text{portée}(n) \cap [A, B[) = \phi \end{array} \right) \tag{11}$$

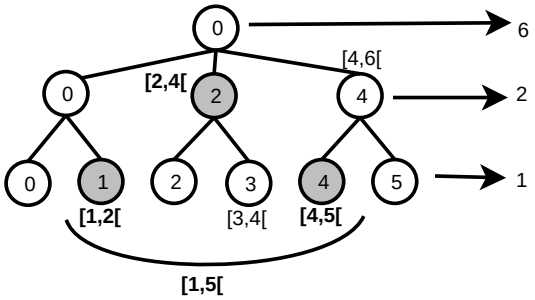

Figure 6: Correspondance entre ensemble de sous problèmes actifs et intervalle.

Le calcul de $Unfold([A, B[)$ peut être fait à l'aide d'un algorithme B&B dont les opérateurs, hormis l'opérateur d'élagage, sont les mêmes que ceux du B&B.

Cet algorithme est basé sur la portée d'un sous problème pour choisir entre un élagage ou une décomposition. Comme l'indique la formule (11), un sous problème n est élagué lorsque sa portée est incluse dans [A,B[, ou lorsque sa portée et [A,B[sont totalement disjoints. Dans le cas contraire, le sous problème n est décomposé. Dans un arbre de profondeur maximale P, cet algorithme B&B opère au plus P décompositions. Ceci garantit le faible coût de l'opérateur de dépliage. Comme l'indique la formule (12), la liste de Unfold([A,B[) est constituée alors par tous les sous problèmes élagués et dont l'intervalle est inclus dans [A,B[. La figure 6 donne un exemple illustrant le passage d'un intervalle vers une liste de sous problèmes actifs.

$$\text{Unfold}([A,B[) = \left\{ \begin{array}{c} n/ \\ \text{élagage}(n) \\ \text{portée}(n) \subseteq [A,B[\end{array} \right\} \tag{12}$$

4 Implémentation et déploiement de l'approche

Cette section présente la stratégie de tolérance aux pannes adoptée dans cette nouvelle approche, détaille la stratégie d'équilibrage de charge de B&B@Grid, et décrit la stratégie employée dans B&B@Grid pour gérer la détection de la terminaison.

4.1 Processus coordinateur et B&B

En plus des deux opérateurs de pliage et de dépliage, B&B@Grid utilise deux types de processus. Il s'agit du processus coordinateur et du processus B&B. Le rôle d'un processus B&B est de traiter un intervalle. L'intervalle est donc l'unité de travail allouée aux processus B&B. Le coordinateur gère principalement l'ensemble des intervalles. Ces intervalles sont stockés dans un ensemble appelé INTERVALLES. Dans la suite de la section, INTERVALLES est supposé être un ensemble d'intervalles non encore traités, même si, en fait, il peut être également un ensemble d'intervalles traités. Par contre, INTERVALLES ne peut pas contenir à la fois des intervalles traités et non traités.

Il est facile de passer de l'ensemble des intervalles traités à l'ensemble des intervalles non encore traités et vice versa. Il suffit de connaître l'ensemble de départ et de faire une opération de soustraction. Soit {[0,20[} l'ensemble initial des intervalles à explorer, et {[3,6[, [9,12[, [15,20[} l'ensemble des intervalles qui restent à explorer au bout d'un certain temps d'exploitation. L'ensemble des intervalles explorés depuis le début est donc égal à {[0,3[, [6,9[, [12,15[}. L'utilisation de B&B@Grid avec les approches fermier-travailleur proposées dans la littérature requiert la manipulation d'un ensemble d'intervalles non encore traités. Par contre, l'utilisation de B&B@Grid avec l'approche pair-à-pair proposée dans [10] requiert la manipulation d'un ensemble d'intervalles traités.

Le processus B&B ne gère qu'un seul intervalle à la fois. Il reçoit cet intervalle d'un processus coordinateur, utilise l'opérateur de dépliage pour déduire l'ensemble des sous problèmes à résoudre, traite une partie ou la totalité de ces sous problèmes à l'aide de l'algorithme B&B, utilise l'opérateur de pliage pour déduire l'intervalle qui correspond aux sous problèmes qui restent à explorer, et communique l'intervalle déduit à un processus coordinateur.

4.2 Déploiement fermier-travailleur à grande échelle

Les deux processus coordinateur et B&B peuvent être utilisés pour paralléliser un B&B avec n'importe quel paradigme parallèle. Dans une approche fermier-travailleur, par exemple, le fermier héberge un seul coordinateur et les processus travailleurs hébergent des processus B&B. Dans une approche pair-à-pair, un pair héberge un coordinateur et un processus B&B. L'approche pair-à-pair est celle qui permet d'exploiter le plus grand nombre de processeurs. Toutefois, afin de tester et de valider B&B@Grid, le paradigme fermier-travailleur est choisi dans nos travaux. Dans ce paradigme, un seul hôte joue le rôle de fermier, et tous les autres celui de travailleur. Ce paradigme est relativement simple à mettre en œuvre. Son inconvénient majeur est que le fermier peut constituer un goulot d'étranglement. Cependant, communiquer et manipuler des intervalles au lieu de listes de sous problèmes permet de réduire le coût des communications et la charge du fermier. Ce paradigme est donc choisi pour expérimenter cette approche. Le but est de montrer que B&B@Grid arrive à pousser les limites d'un paradigme aussi centralisé que celui du fermier-travailleur. Dans le paradigme fermier-travailleur adopté, les hôtes travailleurs hébergent autant de processus B&B qu'ils ont de processeurs, et le hôte fermier héberge le coordinateur. La figure 7 donne un exemple avec quatre processus B&B et un coordinateur. Dans cet exemple, quatre intervalles restent à explorer. Trois de ces intervalles sont en cours d'exploration, [1,5[, [9,19[, et [20,22[, tandis que le quatrième, [25,40[, est en attente d'un processus B&B.

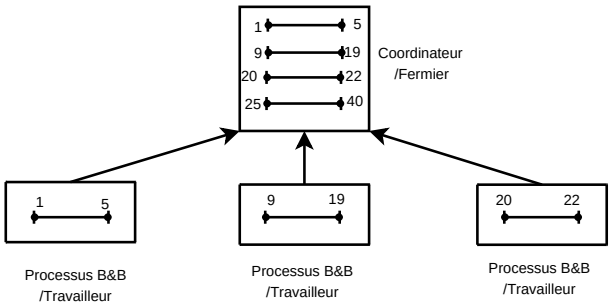

Figure 7: Exemple avec quatre processus B&B et un coordinateur.

4.3 Tolérance aux pannes

La tolérance aux pannes est l'aptitude d'un système à fonctionner malgré ses défaillances éventuelles. La tolérance aux pannes est cruciale pour les méthodes exactes. Une perte d'un (ou plusieurs) sous problème(s) risque d'entraîner une perte définitive d'une (ou plusieurs) solution(s) optimale(s). Par conséquent, il est indispensable de définir une stratégie de tolérance aux pannes pour toute méthode exacte destinée à exploiter un environnement de calcul parallèle volatile. Sur une grille, la tolérance aux pannes peut être faite au niveau de l'intergiciel ou au niveau applicatif. Au niveau intergiciel, les stratégies proposées sont souvent indépendantes des applications. A l'instar de $XtremWeb$[7], le processus en panne est souvent relancé depuis le début. Une telle stratégie

peut s'avérer inefficace pour les processus à longue durée de vie. Au niveau applicatif, les stratégies proposées sont souvent basées sur la sauvegarde et la restauration de l'état d'un processus (check-pointing). Le processus est alors relancé à partir de son dernier point de sauvegarde.

Il est donc important d'identifier les données à sauvegarder lors de la définition d'une stratégie de tolérance aux pannes. Dans le B&B, l'information sauvegardée par chaque processus contient principalement la (ou les) meilleure(s) solution(s) trouvée(s) et le ou le(s) sous problème(s) non encore traité(s). Dans B&B@Grid, le processus coordinateur gère une éventuelle panne de sa machine hôte en sauvegardant périodiquement dans un fichier le contenu de l'ensemble INTERVALLES et l'ensemble des solutions trouvées. En cas de panne du processeur fermier, le coordinateur est lancé en initialisant INTERVALLES par le contenu de ce fichier. Le coordinateur récupère également les solutions sauvegardées.

Un processus B&B utilise l'opérateur de pliage pour déduire l'intervalle auquel appartiennent leurs sous problèmes non encore explorés. Il gère les pannes du travailleur en actualisant régulièrement la copie de son intervalle dans INTERVALLES, et en informant le coordinateur de toute nouvelle solution trouvée. En cas de panne d'un travailleur, la dernière copie de son intervalle est soit affectée en totalité à un autre processus B&B, soit partagée entre plusieurs processus B&B. Un processus B&B met à jour la copie de son intervalle à l'aide d'une simple opération d'intersection d'intervalles. Cette opération actualise à la fois l'intervalle en cours de traitement et sa copie dans INTERVALLES. Soient [A,B[un intervalle en cours de traitement dans un processus B&B, et [A',B'[sa copie dans INTERVALLES au niveau du coordinateur. Comme l'indique la formule (13), l'intersection des deux intervalles s'obtient en considérant le maximum de leurs débuts et le minimum de leurs fins.

$$[A, B[\cap[A', B'[= [max(A, A'), min(B, B')[\tag{13}$$

Au cours d'une résolution, les deux intervalles [A,B[et [A',B'[évoluent constamment. En effet, un processus B&B, en traitant [A,B[, incrémente la valeur de A et laisse la valeur de B inchangée. Par contre, le mécanisme d'équilibrage de charge, expliqué dans la section suivante, décrémente la valeur de B' et laisse la valeur de A' inchangée. Le début d'un intervalle est susceptible également d'être incrémenté par plusieurs processus B&B. Ceci survient lorsque la stratégie d'équilibrage de charge attribut le même intervalle à plusieurs processus.

La figure 8 illustre la stratégie de tolérance aux pannes de B&B@Grid à l'aide d'un exemple. Au début, le coordinateur contient l'intervalle [0,6[. Cet intervalle peut correspondre, par exemple, à une instance du Flow-Shop de 6 tâches. L'intervalle [0,6[est attribué en entier au premier processus qui rejoint la grille. Lorsqu'un deuxième processus rejoint le calcul, le coordinateur lui accorde l'intervalle [3,6[. Chacun des deux processus traite l'intervalle reçu. Après une certaine période de temps, le premier processus contacte le coordinateur pour lui communiquer son intervalle. Cet intervalle représente l'ensemble des numéros des sous problèmes non traitées. Dans l'exemple illustré par la figure 8, l'intervalle restant au premier processus est égal à [1,6[, car les sous problèmes portant le numéro 0 sont explorés. Le coordinateur opère alors une intersection entre l'intervalle reçu et sa copie au niveau du coordinateur.

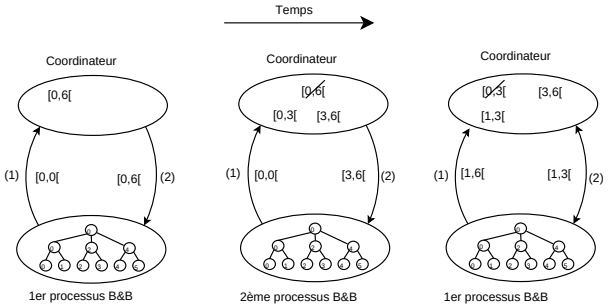

Figure 8: Stratégie de tolérance aux pannes de B&B@Grid.

Pour que la gestion de la tolérance aux pannes soit efficace, les intervalles de l'ensemble INTERVALLES doivent être ordonnés. Deux listes de références sont utilisées pour ordonner ces intervalles. Il s'agit des listes AJOUR et IDENTITE. Ces deux listes permettent respectivement de retrouver un intervalle dans INTERVALLES d'une façon plus efficace, et de détecter les processus en panne. Il est important de noter que INTERVALLES contient les copies des intervalles en cours de traitement, tandis que AJOUR et IDENTITE ne contiennent que des références vers les intervalles de INTERVALLES. La cardinalité des ensembles INTERVALLES, AJOUR et IDENTITE est toujours identique.

IDENTITE ordonne les intervalles selon leur identifiant. L'identifiant d'un intervalle est juste un numéro. Un compteur est utilisé par le coordinateur pour générer un nouvel identifiant à chaque fois qu'un nouvel intervalle est créé. Cet identifiant est alors attribué au nouvel intervalle. IDENTITE est utilisée lorsque un processus B&B contacte le coordinateur pour actualiser la copie de son intervalle. Le but est de retrouver rapidement la bonne copie de INTERVALLES à actualiser. Il suffit de lancer une recherche dans IDENTITE selon l'identifiant de l'intervalle reçu. Cette recherche peut être de type dichotomique, et peut donc être faite en $O(log(N))$, où N est le nombre d'intervalles de IDENTITE.

Comme indiqué auparavant, la liste AJOUR est utilisée pour détecter les processus en panne. Cette liste ordonne les intervalles selon leur dernier instant de mise à jour. A chaque fois qu'un intervalle est mis à jour, sa référence est retirée de la liste AJOUR, et remise au début de cette liste. Lorsqu'un intervalle est créé, sa référence est toujours placée au début de AJOUR. La référence du dernier intervalle mis à jour se trouve donc toujours à la fin de la liste. Il est alors facile de savoir si son processus est en panne. Il suffit de comparer sa dernière date de mise à jour avec le temps actuel. Si la période écoulée dépasse la période de mise à jour alors le processus chargé d'explorer cet intervalle est en panne. Par conséquent, cet intervalle peut être assigné à un autre processus.

Dans cette section, nous avons expliqué la stratégie de tolérance aux pannes de B&B@Grid à l'aide du paradigme fermier-travailleur. Cependant, le principe de cette stratégie reste valable pour les autres paradigmes. Très peu de modifications sont nécessaires pour utiliser cette stratégie dans l'approche de tolérance aux pannes pair-à-pair proposée dans [10]. Il suffit à chaque pair d'utiliser une liste d'intervalles à la place de la liste de sous problèmes résolus. Un pair intègre alors le fonctionnement du coordinateur décrit dans cette section. Cependant, l'ensemble INTERVALLES ne contient pas la liste des intervalles non traités,

mais ceux traités. Les listes IDENTITE et AJOUR doivent être également employées pour manipuler plus efficacement l'ensemble INTERVALLES. Néanmoins, le pair doit utiliser l'opération d'union d'intervalles au lieu d'utiliser l'opération d'intersection. Comme l'indique la formule (14), l'union de deux intervalles se fait en considérant le minimum de leurs débuts et le maximum de leurs fins. La formule (14) impose bien évidemment que les deux intervalles [A,B[et [A',B'[ne soient pas disjoints.

$$[A, B[\cup [A', B'[= [min(A, A'), max(B, B')[\tag{14}$$

4.4 Equilibrage de charge

Lors de la parallélisation de B&B, il est nécessaire de définir une politique de régulation de charge. L'objectif de cette politique est de minimiser l'occurrence de situations dans lesquelles des processus ont beaucoup de sous problèmes à traiter pendant que d'autres ont des listes vides. Cette politique initie, aux moments opportuns, des transferts de sous problèmes des machines en surcharge vers les machines en sous-charge.

Une stratégie d'équilibrage de charge peut être statique ou dynamique. Dans une stratégie d'équilibrage de charge statique [4, 5], l'allocation de toutes les tâches est décidée avant le début d'un calcul. Par contre, dans une stratégie d'équilibrage de charge dynamique [14], les tâches sont allouées et réallouées durant le calcul. Ceci est fait en fonction de l'état du système. L'équilibrage de charge statique est intéressant lorsque le coût de calcul des tâches est connu à l'avance. Dans un algorithme B&B, il est impossible de connaître le coût des tâches avant leurs traitements. Par conséquent, une stratégie d'équilibrage de charge pour un tel algorithme ne peut être que dynamique.

A notre connaissance, aucune stratégie, décrite dans la littérature pour équilibrer la charge entre processus B&B, ne prend en compte la nature hétérogène et volatile des systèmes de calcul distribués. Or, les processeurs d'une grille de calcul ont souvent des puissances variées, ne sont pas forcément dédiés au calcul, et leur nombre varie constamment. En plus d'être dynamique, une stratégie d'équilibrage de charge d'un B&B sur grilles de calcul doit donc être également adaptative. Autrement dit, la granularité du travail alloué aux processeurs doit s'adapter à leur puissance, leur disponibilité et leur nombre. B&B@Grid adapte la taille des intervalles à la puissance et la disponibilité du processeur demandeur de travail, ainsi qu'au nombre de processeurs participant au calcul. En effet, dans B&B@Grid, la granularité des unités de travail, autrement dit les tailles des intervalles, accordées par le(s) coordinateur(s) aux processus B&B dépendent toujours des puissances, des disponibilités et du nombre de processeurs intervenants dans le calcul.

Dans B&B@Grid, le travail d'un processus B&B consiste à explorer un intervalle. A la réception d'un intervalle, un processus B&B utilise l'opérateur de dépliage pour déduire la liste de sous problèmes à traiter. Un processus B&B sollicite un intervalle lorsqu'il rejoint le calcul pour la première fois, ou lorsqu'il termine l'exploration de son intervalle. Le rôle du coordinateur est alors de lui assigner un intervalle. Le coordinateur tente d'abord de sélectionner un intervalle libre. Un intervalle est considéré comme libre lorsqu'il n'est alloué à aucun processus B&B.

Durant une résolution, certains intervalles de l'ensemble INTERVALLES peuvent être ou devenir libres. Ceci survient notamment lorsque le processus qui traite un intervalle tombe en panne. Comme expliqué dans la section précédente, la consultation de l'ensemble AJOUR permet au coordinateur de détecter ce type d'intervalles libres. L'autre situation où des intervalles peuvent devenir libres est celle où le coordinateur tombe en panne. Après une panne, le coordinateur initialise, à son démarrage, l'ensemble INTERVALLES avec la dernière copie du fichier de sauvegarde. Le coordinateur attribue alors, à tous ces intervalles, la date de dernière mise à jour la moins récente, i.e. 1er janvier 1970 minuit, et initialise l'ensemble AJOUR à l'aide de cette date. La dernière situation où un intervalle est libre se présente au début d'une résolution. En effet, l'intervalle original est toujours libre au départ. L'intervalle original est celui qui contient l'ensemble des numéros de tous les sous problèmes de l'arbre de l'algorithme B&B. Cet intervalle représente la portée du problème original à résoudre. Au début, l'ensemble INTERVALLES ne contient que l'intervalle original. La référence de cet intervalle est également rajoutée aux ensembles IDENTITE et AJOUR. Au départ, la date de mise à jour de l'intervalle original est considérée comme étant le 1 janvier 1970 minuit. Le coordinateur le traite donc comme un intervalle en panne. Dans ces trois situations, la consultation de l'ensemble AJOUR permet toujours au coordinateur de déterminer les intervalles libres.

Le coordinateur tente donc d'abord de satisfaire une demande de travail par un intervalle libre. En absence d'intervalle libre, le coordinateur sélectionne le plus grand intervalle de INTERVALLES. Une fois sélectionné, l'intervalle est découpé et sa première partie est attribuée au processeur demandeur. Afin de retrouver efficacement un tel intervalle, le coordinateur gère une autre liste appelée TAILLE. Cette liste est également une liste de références vers les intervalles de INTERVALLES. Dans TAILLE, les intervalles sont ordonnés selon leur taille. La référence du plus grand intervalle se trouve toujours en tête de la liste, et la référence du plus petit intervalle se trouvant à la queue de la liste. Il suffit donc au coordinateur de prendre l'intervalle dont la référence est au début de TAILLE et de le partitionner.

Soit [A,B[l'intervalle sélectionné pour être partitionné. Le coordinateur découpe un intervalle [A,B[en deux intervalles [A,C[et [C,B[. Le processus B&B détenteur, celui à qui appartient [A,B[, garde [A,C[puisque il l'explore déjà en partant de A. Le processus B&B demandeur, celui qui sollicite un nouvel intervalle, obtient [C,B[. Au bout d'un certain temps, le processus B&B détenteur est également informé de son nouvel intervalle [A,C[. En effet, comme indiqué dans la section précédente, les processus B&B contactent régulièrement le coordinateur pour actualiser leurs intervalles ainsi que leurs copies dans INTERVALLES. Les deux intervalles [A,C[et [C,B[n'ont pas forcément la même longueur. En effet, les processus B&B demandeur et détenteur sont déployés dans un environnement où les hôtes sont souvent hétérogènes et non dédiés. Par conséquent, les longueurs des deux intervalles doivent être proportionnelles à la participation des deux processus au calcul. Comme l'indique la formule (15), le choix du point de partitionnement C dépend de la puissance et de la disponibilité des hôtes qui hébergent les deux processus détenteur et demandeur.

$$\frac{C-A}{B-C} = \frac{Puissance(pH) * Disponibilite(pH)}{Puissance(pR) * Disponibilite(pR)} \tag{15}$$

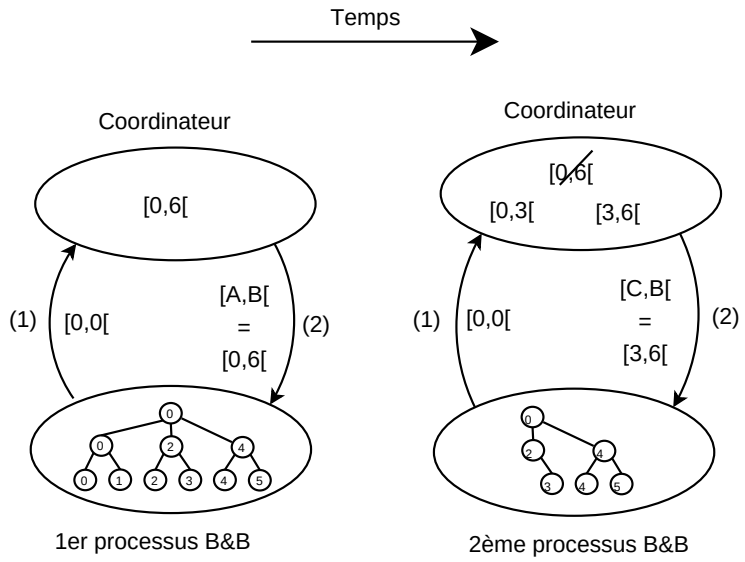

Figure 9: Stratégie d'équilibrage de charge de B&B@Grid.

Où:

- pH et pR: sont les processeurs qui hébergent les deux processus B&B respectivement détenteur et demandeur.
- $[A, B[$: l'intervalle à diviser.
- C : le point de division de l'intervalle [A,B[.
- $Puissance(p)$: la puissance d'un processeur p, pouvant s'exprimer en Million d'Instructions Par Seconde (MIPS) ou en FLOPS.
- $Disponibilite(p)$: la disponibilité d'un processeur p, i. e. le pourcentage de cycles CPU accordés par le système d'exploitation au processus B&B durant une période donnée. Dans B&B@Grid, cette période est égale à la période de mise à jour des intervalles définie par la stratégie de tolérance au pannes.

Après un découpage, les listes INTERVALLES, IDENTITE, AJOUR et TAILLE doivent être mises à jour.

- INTERVALLES est mis à jour en remplacent [A,B[par [A,C[, et en y insérant l'intervalle [C,B[,
- TAILLE est actualisée en supprimant la référence de l'intervalle [A,B[de la tête la liste, et en insérant les références de [A,C[et [C,B[à leurs bonnes positions. Ces deux insertions se font en $2 * log(N)$, où N est le nombre d'intervalles dans INTERVALLES.

- IDENTITE est mis à jour en insérant la référence de [C,B[à la tête de la liste IDENTITE. En effet, l'identifiant de [C,B[est le plus grand après ce découpage, tandis que [A,C[garde l'identifiant de [A,B[.
- Et AJOUR est actualisée en supprimant la référence de [A,B[et en insérant les deux références de [A,C[et [C,B[à la tête de la liste.

Dans la phase terminale de la résolution d'une instance d'un problème, le plus grand intervalle de INTERVALLES est tellement petit qu'il est préférable de ne pas le partitionner. En effet, la granularité d'une tâche sur une grille de calcul doit dépasser un certain seuil puisque la latence de ce type d'environnement n'est pas négligeable. Par conséquent, l'approche B&B@Grid est paramétrable avec un seuil ϵ. Un intervalle est dupliqué au lieu d'être découpé lorsque sa taille est inférieure à ϵ. Le coordinateur ne garde qu'une seule copie d'un intervalle dupliqué, même s'il est accordé à plusieurs processus.

La figure 9 illustre le fonctionnement de la stratégie d'équilibrage du B&B@Grid sur un exemple. Dans cet exemple, deux processus B&B interviennent dans le calcul. Le but est d'explorer l'intervalle [0,6[qui est la portée du problème original. Au départ, l'ensemble INTERVALLES contient l'intervalle libre [0,6[. Il l'attribue donc complètement au premier processus B&B. Au bout d'un certain temps, le deuxième processus B&B rejoint le calcul et demande à son tour du travail. Comme INTERVALLES ne contient plus d'intervalles libres, un intervalle non libre est sélectionné pour être découpé. Le plus grand et seul intervalle de la liste est [0,6[. Dans cet exemple, la participation des deux processus au calcul est supposée similaire. Par conséquent, le point de découpage de [0,6[est égal 3. Le processeur B&B détenteur garde l'intervalle [0,3[, tandis que le processeur B&B demandeur obtient l'intervalle [3,6[. Comme indiqué dans la section précédente, le premier processeur limite son exploration à l'intervalle [0,3[après une première opération de sauvegarde.

4.5 Détection de la terminaison

La gestion de la fin des traitements est une autre problématique qui se pose dans tout environnement parallèle, notamment les grilles de calcul. Dans notre approche, la résolution s'arrête lorsque l'ensemble INTERVALLES devient vide. Au départ, INTERVALLES contient un seul intervalle qui correspond à la totalité de l'arbre. Le début de cet intervalle est égal à zéro, le plus petit numéro de l'arbre, et sa fin est égale au plus grand numéro de l'arbre. Autrement dit, INTERVALLES est initialisé par la portée du problème racine de l'arbre.

La cardinalité de INTERVALLES est le nombre d'intervalles qu'il contient, et la taille de INTERVALLES est la somme des longueurs de ses intervalles. Au cours d'une résolution, la cardinalité de INTERVALLES est plus ou moins égale au nombre de processus B&B impliqués dans le calcul, tandis que sa taille diminue constamment. Cette taille correspond au nombre de solutions non encore explorées de l'espace de recherche. Dans B&B@Grid, le coordinateur supprime automatiquement tout intervalle vide de l'ensemble INTERVALLES. La résolution s'arrête lorsque INTERVALLES devient vide, et donc lorsque sa taille devient nulle. Lorsque INTERVALLES est vide, un processus B&B qui

contacte le coordinateur pour actualiser ou demander un intervalle reçoit l'ordre par le coordinateur de s'arrêter. Par conséquent, la détection de la terminaison est implicite et ne nécessite aucune communication explicite additionnelle.

La gestion de la fin des traitements est plus difficile dans un paradigme complètement décentralisé tel que le paradigme pair-à-pair. Comme indiqué auparavant, B&B@Grid peut être utilisée avec l'approche présentée dans [10]. Dans ce cas, le coordinateur garde, dans l'ensemble INTERVALLES, les intervalles déjà traités. Au départ, INTERVALLES est vide. Au cours d'une résolution, la cardinalité de INTERVALLES est constamment plus ou moins égale au nombre de processus B&B impliqués dans le calcul. Cependant, la taille de INTERVALLES augmente constamment. Cette taille correspond au nombre de solutions explorées de l'espace de recherche. Le calcul s'arrête une fois que INTERVALLES contient l'intervalle global.

5 Expérimentations

Cette section commence par une description de la grille d'expérimentation utilisée pour valider nos travaux ansi que du problème du Flow-Shop. Elle présente ensuite les expérimentations effectuées pour évaluer les performances de B&B@Grid et la comparer avec les meilleures approches de la littérature. La section se termine par la présentation des expérimentations effectuées pour résoudre un problème du Flow-Shop connue dans la littérature et resté non résolu pendant une quinzaine d'années.

5.1 Grille d'expérimentation de nos travaux

Les expérimentations présentées dans ce document ont été réalisées sur la grille détaillée dans le tableau Table 1. Comme l'indique la figure 10, notre grille expérimentale est composée de grappes de *Grid'5000*[1] et d'établissements de l'Université des Sciences et Technologies de Lille (USTL[2]). La grille expérimentale utilisée dans nos travaux est conforme aux caractéristiques des grilles.

En effet, cette grille est d'une échelle importante en termes du nombre de processeurs et de la taille du réseau d'interconnexion de ces machines. Notre grille d'expérimentation est composée d'environ 2000 processeurs répartis sur neuf grappes. Six grappes appartiennent à la grille expérimentale Grid'5000, et trois grappes à trois établissements de l'USTL. A la différence des machines universitaires qui sont mono-processeur, tous les nœuds de calcul de Grid'5000 utilisés sont des bi-processeurs. Grid'5000 apporte les trois quarts du nombre de processeurs de la grille expérimentale utilisée dans nos travaux. Comme l'indique la figure 10, les processeurs de cette grille sont géographiquement répartis.

Les ressources de calcul de notre grille expérimentale sont gérées par différentes organisations. Chacun des sites de *Grid'5000* est administré par un ingénieur différent. Cependant, les sites de cette grille ne sont pas protégés entre eux par des pare-feux. Autrement dit, tout processeur de *Grid'5000* peut joindre tout autre processeur de cette grille. Par contre, les machines des grappes universitaires se trouvent derrière des pare-feux. Les grappes universitaires sont

[1] https://www.*Grid5000*.fr/
[2] http://www.univ-lille1.fr/

CPU (GHz)	Domaine(Grappe)	Nombre
P4 1.70	IEEA-FIL (USTL)	24
P4 2,40		48
P4 2,80		59
P4 3,00		27
AMD 1,30	Polytech'Lille (USTL)	14
Celeron 2,40		35
Celeron 0,80		14
Celeron 2,00		13
Celeron 2,20		28
P3 1,20		12
P4 3,20		12
P4 1,60	IUT-A (USTL)	22
P4 2,00		18
P4 2,80		45
P4 2,66		57
P4 3,00		41
AMD 2,2	Bordeaux (Grid'5000)	2x47
AMD 2,2	Lille (Grid'5000)	2x54
Xeon 2,4	Rennes (Grid'5000)	2x64
AMD 2,2		2x64
AMD 2,0		2x100
AMD 2,0	Sophia (Grid'5000)	2x107
AMD 2,2	Toulouse (Grid'5000)	2x58
AMD 2	Orsay (Grid'5000)	2x216
Total		**1889**

Table 1: Détails de la grille utilisée.

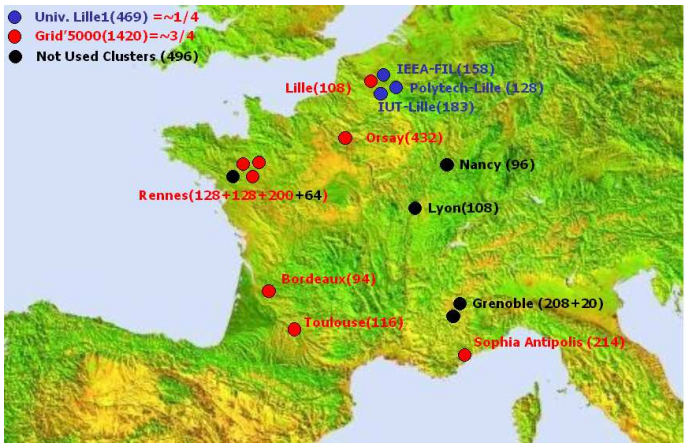

Figure 10: Une grille géographiquement répartie.

administrées par des équipes réseau distinctes. Chaque grappe est protégée par un pare-feu, et appartient à un domaine d'administration distinct. Les machines

universitaires ne sont donc pas joignables en dehors des machines de la même grappe.

La grille expérimentale utilisée est un environnement dynamique. En effet, les processeurs de Grid'5000 doivent être réservés avant leur utilisation. Une réservation peut se faire selon trois modes. Il s'agit d'une réservation normale, d'une réservation avec déploiement du système d'exploitation, et d'une réservation en *besteffort*. Les expérimentations présentées dans nos travaux sont faites en *besteffort*. Une réservation d'un processeur en *besteffort* serait annulée si un autre utilisateur demande ce processeur. Le système de réservation n'informe pas les processus lancés de la fin d'une réservation *besteffort*. Un processeur perdu, à cause d'une réservation d'un autre utilisateur, peut donc être considéré comme un processeur tombé en panne. Pendant nos expérimentations, tout processeur libre de *Grid'5000* est réservé en *besteffort*. De plus, les machines des grappes universitaires sont principalement dédiées à l'enseignement et ne peuvent donc être exploitées que lorsqu'elles sont oisives. Par conséquent, notre grille expérimentale constitue un environnement hautement dynamique où les ressources de calcul sont exploitées en vol de cycles. Ceci permet de n'exploiter que la puissance de calcul perdue, et de ne pas pénaliser les autres utilisateurs de *Grid'5000* et les étudiants de l'USTL.

L'hétérogénéité des ressources matérielles et logicielles caractérisent également notre grille expérimentale. Comme l'indique le tableau Table 1, les processeurs des différentes grappes ne sont pas similaires. De plus, certaines grappes contiennent même des processeurs différents. Par ailleurs, les grappes sont interconnectées différemment. En effet, les grappes universitaires sont reliées en 1 GigaBit, tandis que les grappes de Grid'5000 sont interconnectées en 2,5 GigaBits par le réseau national RENATER3. En plus de cette hétérogénéité matérielle, la grille expérimentale se caractérise également par une hétérogénéité dans les distributions Linux installées sur les grappes. Ceci accentue l'hétérogénéité des outils logiciels se trouvant sur chaque grappe.

5.2 Le problème du Flow-Shop

Afin de valider l'approche B&B@Grid, les expérimentations décrites sont faites sur un problème de Flow-Shop. Ce problème fait partie des problèmes d'ordonnancement. Un problème d'ordonnancement est défini par un ensemble de tâches et de ressources. Le Flow-Shop est un problème multi-opération où chaque opération est le traitement d'une tâche par une machine. Dans le Flow-Shop, les ressources sont considérées comme des machines se trouvant dans un atelier de production. Ces machines traitent les tâches selon le principe de la chaîne de production. Les machines sont donc disposées dans un certain ordre. Ainsi, une machine ne peut commencer le traitement d'une tâche que lorsque toutes les machines, qui se trouvent en amont, ont fini leur traitement. Une durée est associée à chaque opération, celle-ci correspond au temps nécessaire à la machine pour finir son traitement. Une opération ne peut pas être interrompue, et les machines sont des ressources critiques, car une machine ne peut traiter qu'une tâche à la fois. La figure 11 montre un exemple d'une solution d'une instance d'un problème de Flow-Shop définie par 6 tâches et 3 machines. Le *makespan* d'une solution correspond à la date de fin de sa dernière tâche sur la dernière machine. L'objectif est donc de trouver la solution qui minimise le *makespan*.

Dans [9], il a été montré que la minimisation du *makespan* est NP-difficile à partir de trois machines.

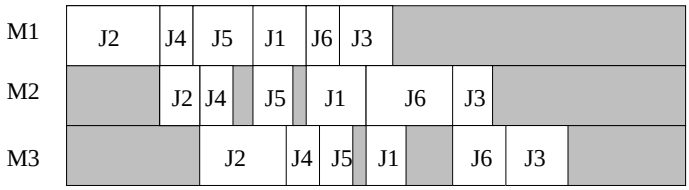

Figure 11: Instance du problème du Flow-Shop avec 6 tâches et 3 machines.

5.3 Efficacité de B&B@Grid

Dix expérimentations avec B&B@Grid ont été effectuées pour évaluer la capacité de cette nouvelle approche à optimiser les communications des sous problèmes, et la comparer avec les autres approches connues dans la littérature. Chaque expérimentation utilise une des dix instances définies par 50 tâches et 20 machines et publiées dans [15]. Les solutions optimales de ces instances ne sont pas encore connues. Le site [6] publie les meilleures solutions trouvées pour chaque instance. Ces instances sont notées *Ta051*, *Ta052*, *Ta053*, *Ta054*, *Ta055*, *Ta056*, *Ta057*, *Ta058*, *Ta059* et *Ta060*.

Un seul processus coordinateur est utilisé et un processeur B&B est lancé sur chaque processeur libre de notre grille. Les dix expérimentations sont faites selon l'approche fermier-travailleur et utilisent les stratégies de tolérance aux pannes et d'équilibrage de charge présentées respectivement dans les sections 4.3 et 4.4. Afin de sauvegarder leurs intervalles, les processus B&B communiquent leurs intervalles au coordinateur toutes les 3 minutes. Chaque expérimentation a été initialisée par la meilleure solution connue de l'instance utilisée. Les expérimentations ont été arrêtées après une heure de calcul puisque l'objectif n'était pas de les résoudre.

5.3.1 Optimisation des communications

Le tableau Table 2 permet de comparer les taux de compression des sous problèmes communiqués entre B&B@Grid et l'approche à jeton de PICO [6]. Le taux de compression d'une approche, généralement exprimé en pourcentage, est défini comme le rapport de la taille des données après compression sur leur taille initiale avant compression. Pour obtenir le taux de compression de chaque approche, notre algorithme calcule la taille de chaque intervalle communiqué, détermine le codage associé à cet intervalle avec l'approches à jetons, calcule la taille de chaque codage, et compare les deux tailles ainsi obtenues avec la taille des sous problèmes associés à l'intervalle communiqué. Les deux premières colonnes du tableau désignent respectivement le nom de l'instance expérimentée et le nombre moyen de processeurs exploités pendant l'expérimentation. Les autres colonnes donnent le taux de compression de chaque approche. La

[6] http://ina2.eivd.ch/Collaborateurs/etd/problemes.dir/ordonnancement.dir/ordonnancement.html

dernière ligne donne le nombre moyen de processeurs, exploités durant les 10 expérimentations, et le taux de compression moyen de chaque approche.

Le tableau Table 2 montre que, dans les dix expérimentations, le taux de compression de B&B@Grid est toujours meilleur que celui de l'approche à jetons. Le taux de compression moyen obtenu par B&B@Grid est d'environ $0,13\%$. Il signifie que cette approche réduit, en moyenne, d'environ 766 (100/0.1289 = 775.7951) fois la taille des sous problèmes communiqués. Les rapports entre les taux de compression moyens permettent de comparer les deux approches entre elles. Ainsi le rapport du taux de compression moyen de B&B@Grid avec celui de l'approche à jetons montre que le codage de B&B@Grid est en moyenne 92 (11.9527/0.1289 = 92.7284) fois plus petit que celui de l'approche à jetons.

Inst.	**Proc.**	**App. Jet.(%)**	**B&B@Grid(%)**
Ta051	0692,0	11,9278	00,1386
Ta052	1276,3	11,9170	00,1295
Ta053	1055,9	11,7572	00,1264
Ta054	1236,7	11,9346	00,1256
Ta055	1312,3	11,9537	00,1285
Ta056	1213,0	12,0936	00,1300
Ta057	1442,6	12,0155	00,1291
Ta058	1432,2	11,9850	00,1248
Ta059	1448,2	11,9016	00,1293
Ta060	1398.7	12,0405	00,1267
Moy.	**1111**	**11,9527**	**00,1289**

Table 2: Taux de compression des sous problèmes communiqués.

5.3.2 Tolérance aux pannes

Le tableau Table 3 permet de comparer B&B@Grid et l'approche à variables de [10] pendant les dix expérimentations. Cette comparaison est faite en terme de la taille globale des sous problèmes stockés par le coordinateur. Ces sous problèmes représentent l'état global d'une résolution. En les sauvegardant, il est toujours possible de réinitialiser une résolution et de la relancer à partir de son point de sauvegarde. Pour chaque expérimentation, la première colonne de ce tableau indique le nom de l'instance utilisée, et la deuxième colonne le nombre moyen de processeurs exploités. Les troisième, quatrième et cinquième colonnes donnent le taux de compression de chaque approche. Le tableau Table 3 indique principalement que B&B@Grid réduit la taille des sous problèmes du coordinateur, et améliore d'environ 465 (60.7704/0.1306 = 465.3169) fois le taux de compression de l'approche à variables.

5.3.3 Equilibrage de charge

Pendant une résolution, un processus B&B passe son temps à déplier les intervalles reçus du coordinateur, résoudre des sous problèmes, plier les sous problèmes non résolus, et communiquer au coordinateur les intervalles obtenus après

Inst.	**Proc.**	**App. Var.(%)**	**B&B@Grid(%)**
Ta051	0692,0	62,2347	00,1414
Ta052	1276,3	60,2902	00,1319
Ta053	1055,9	60,6217	00,1295
Ta054	1236,7	61,6560	00,1273
Ta055	1312,3	60,1562	00,1302
Ta056	1213,0	60,2080	00,1300
Ta057	1442,6	60,1562	00,1302
Ta058	1432,2	61,5869	00,1259
Ta059	1448,2	59,8684	0,1315
Ta060	1398.7	60,9254	00,1285
Moy.	**1111**	**60,7704**	**00,1306**

Table 3: Taux de compression de sous problèmes du coordinateur.

pliage. Le tableau Table 4 détaille la répartition du temps entre ces quatre différentes fonctions pendant les dix expérimentations. La première colonne donne le nom de l'instance expérimentée et la deuxième colonne le nombre moyen de processeurs utilisés pendant une expérimentation. Les troisième, quatrième, cinquième et sixième colonnes indiquent respectivement le temps consacré au dépliage, à la résolution de sous problèmes, au pliage, et à la communication. Comme le temps de pliage est le plus faible et pour plus de lisibilité, les temps de chaque expérimentation sont normalisés par rapport au temps du pliage. Par exemple, pendant l'expérimentation faite avec l'instance *Ta051*, le temps du dépliage est 77 fois plus grand que celui du pliage. La dernière colonne est la plus importante du tableau, elle donne le taux d'utilisation des processeurs par les processus B&B pour faire de la résolution de sous problèmes. Ce taux est calculé en faisant le rapport entre le temps de résolution, d'un côté, et la somme des temps de pliage, de dépliage, de résolution et de communication, de l'autre. Comme l'indique cette colonne, les processus B&B passent plus de 99% de leur temps à faire de la résolution de sous problèmes. Ceci montre que la répartition de la charge entre les processus B&B est quasiment optimale.

Inst.	**Proc.**	**Unfold**	**Résol.**	**Fold**	**Com.**	**Résol.(%)**
Ta051	0692,0	77	92 249	1	20	99,91
Ta052	1276.3	31	93 207	1	20	99,94
Ta053	1055.9	124	78 819	1	21	99,99
Ta054	1236.7	47	98 281	1	18	99,93
Ta055	1312.3	31	103 612	1	15	99,95
Ta056	1213.0	67	96 592	1	18	99,91
Ta057	1442.6	34	101 406	1	15	99,95
Ta058	1432.2	43	99 692	1	15	99,94
Ta059	1448.2	32	102 144	1	14	99,95
Ta060	1398.7	58	95 209	1	15	99,92
Moy.	**1111**	**54**	**96 121**	**1**	**17**	**99,94**

Table 4: Taux d'utilisation du processeur par les processus B&B.

5.3.4 Passage à l'échelle

Le tableau Table 5 indique les taux d'utilisation CPU du coordinateur pendant les dix expérimentations. La première colonne donne le nom de l'instance utilisée, la deuxième colonne le nombre moyen de processeurs exploités pendant chaque expérimentation, et la troisième colonne le taux d'utilisation CPU du processeur qui héberge le coordinateur. Comme l'indique ce tableau, malgré le nombre considérable de processeurs exploités, le taux d'utilisation du processeur par le coordinateur reste faible. Environ 1111 processus B&B sont lancés en moyenne et le taux d'utilisation CPU du processeur par le coordinateur est égal en moyenne à $1,29\%$. Ce taux est un bon indicateur de la capacité de passage à l'échelle de B&B@Grid. Dans un paradigme centralisé, tel que celui du fermier-travailleur, le goulot d'étranglement survient lorsque le processeur du fermier utilise le processeur à 100%. Par ailleurs, il faut noter que le passage à l'échelle dépend également de la taille de l'information reçue par le coordinateur. En effet, le réseau ne doit pas être saturé. Comme indiqué auparavant, B&B@Grid réduit la taille de l'information communiquée. Par conséquent, le taux de réduction de la taille des sous problèmes et le taux d'utilisation CPU du coordinateur permettent à B&B@Grid de passer à l'échelle.

Inst.	**Proc.**	**Taux. Util. CPU Coord.(%)**
Ta051	0692,0	00,39
Ta052	1276.3	01,29
Ta053	1055.9	01,52
Ta054	1236.7	01,43
Ta055	1312.3	01,27
Ta056	1213.0	01,26
Ta057	1442.6	01,44
Ta058	1432.2	01,33
Ta059	1448.2	01,59
Ta060	1398.7	01,47
Moy.	**1111**	**01,29**

Table 5: Taux d'utilisation CPU par le processus coordinateur.

5.4 Résolution exacte de l'instance *Ta056*

Deux autres expérimentations ont été réalisées pour valider B&B@Grid. Dans ces expérimentations, nous avons considéré l'instance $Ta056$ qui est une instance publiée dans [15]. La solution optimale de cette instance n'était pas encore connue. Cette instance est définie par 50 tâches et 20 machines. La meilleure solution connue pour $Ta056$ a un coût de 3681. Elle a été trouvée dans [13] à l'aide d'une méta-heuristique gloutonne itérative. En plus de la validation de B&B@Grid, le défi de cette première expérimentation est de résoudre de façon exacte cette instance. Par contre, la deuxième expérimentation vise à prouver l'efficacité de B&B@Grid.

Le défi de la première expérimentation est de trouver la solution exacte de $Ta056$. Cette instance nécessite une puissance de calcul considérable. L'initialisation d'un algorithme B&B avec une solution approchée permet de réduire, d'une

façon significative, la puissance de calcul nécessaire à une résolution. Par conséquent, l'algorithme est initialisé avec la meilleure solution connue de $Ta056$. Au début de l'expérimentation, seules les trois grappes universitaires et une seule grappe Grid'5000, celle de Lille, ont été exploitées. Les autres grappes de Grid'5000 ne sont intégrées à cette expérimentation que trois jours avant la fin de la résolution. L'expérience a permis de trouver la solution optimale au bout d'un mois et trois semaines environ. Le pic de processeurs enregistré est de 1245. La solution optimale de $Ta056$ a un coût égal à 3679. Dans cette solution, les tâches doivent être ordonnancées selon l'ordre suivant : (14, 37, 3, 18, 8, 50, 5, 42, 33, 40, 4, 45, 17, 27, 20, 21, 49, 13, 43, 11, 10, 41, 24, 15, 16, 19, 44, 32, 26, 28, 46, 1, 36, 39, 47, 25, 30, 7, 2, 31, 23, 6, 48, 22, 29, 34, 9, 35, 38, 12).

Le but de la deuxième expérience est de récupérer davantage de statistiques durant la résolution de $Ta056$. Ces statistiques permettent de mieux connaître le comportement de B&B@Grid, et d'évaluer ses performances. Dans cette expérience, la totalité des machines disponibles dans les grappes universitaires et Grid'5000 ont été exploitées. L'algorithme est initialisé avec une borne égale à 3680. Cette borne est choisie pour réduire le temps de résolution de $Ta056$, par rapport à la première expérimentation, tout en permettant au B&B@Grid d'améliorer cette solution initiale. Le tableau Table 6 résume les statistiques enregistrées durant l'expérimentation. Comme l'indique ce tableau, la résolution a duré environ 25 jours avec un temps de résolution cumulé d'environ 22 ans. En moyenne, 328 processeurs sont utilisés. La figure 12 montre l'évolution du nombre de processeurs utilisés pendant la résolution. Le pic de processeurs enregistré est de 1.195.

Temps de résolution	25 jours
Temps cumulé	22 ans
Nombre moyen de travailleurs	328
Nombre maximal de travailleurs	1.195
Utilisation des processeurs des travailleurs	97%
Utilisation du processeur du fermier	1,7%
Opérations de sauvegarde	4.094.176
Allocations d'unités de travail	129.958
Nœuds explorés	6,50874 e+12
Nœuds redondants	0,39%

Table 6: Statistiques enregistrées durant la deuxième expérimentation.

Durant cette expérimentation, les processus B&B ont sollicité environ 130000 fois le mécanisme d'équilibrage de charge du coordinateur, et effectué plus de 2 millions d'opérations de sauvegarde. Ceci démontre le fonctionnement robuste de ces deux mécanismes. Le coordinateur opère des sauvegardes toutes les 30 minutes. Plus de 6 milliards de nœuds ont été explorés. Certains nœuds sont explorés de manière redondante, cela survient en phase de terminaison de la résolution. En effet, s'il ne reste plus que des intervalles d'une taille inférieure à un certain seuil fixé, ces derniers sont dupliqués en réponse aux requêtes de demande de travail.

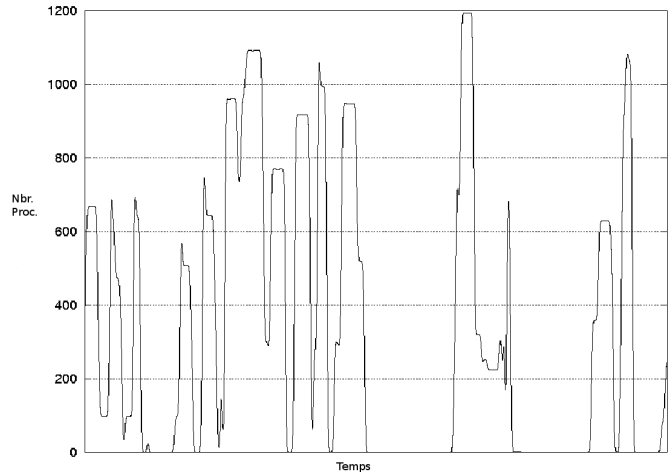

Figure 12: Evolution du nombre de processeurs utilisés durant la deuxième expérience.

Les statistiques enregistrées montrent que le pourcentage de nœuds redondant est faible (inférieur à $0,4\%$). Les processeurs travailleurs sont exploités en moyenne à 97%, tandis que le processeur hébergeant le fermier est exploité à $1,7\%$. Ces deux pourcentages sont de bons indicateurs sur l'efficacité parallèle de l'approche et de sa qualité quant au passage à l'échelle. Dans le paradigme fermier-travailleur, une bonne approche parallèle doit maximiser l'utilisation des processeurs des travailleurs, et minimiser l'utilisation du processus fermier.

En terme de puissance de calcul utilisée, cette expérience se place en deuxième position des expérimentations à grande échelle menées sur les problèmes d'optimisation combinatoire. A notre connaissance, une seule autre expérience[12], a mobilisé davantage de ressources de calcul. Le tableau Table 7 montre les principaux défis réalisés en optimisation combinatoire. $Sw24978$ est la seule instance qui a demandé plus de puissance de calcul. Il s'agit d'une instance du problème de voyageur de commerce qui consiste à trouver le plus court chemin pour visiter 24978 villes suédoises. La puissance de calcul utilisée pour résoudre cette instance est équivalente à environ 84 ans de calcul sur un processeur Intel Xenon 2.8 Ghz.

Instance	**Puiss. de calcul**
1)$Sw24978$(TSP de 24978 villes de Suède)	84 ans/Intel 2.8 GHz
2)$Ta056$(FS de 50 tâches sur 20 machines)	22 ans
3)$D15112$(TSP de 15.112 villes d'Allemagne)	22 ans/ Alpha 500 MHz
4)$Nug30$(QAP de taille 30)	7 ans/HP-C3000 400MHz
5)$Usa13509$(TSP de 13.509 villes des USA)	4 ans

Table 7: Principaux défis en optimisation combinatoire.

[12]http://www.tsp.gatech.edu/world/swlog.html

6 Conclusions et perspectives

Dans ce papier, nous avons présenté une nouvelle approche, appelée B&B@Grid, pour la gridification des méthodes exactes utilisées pour la résolution des problèmes d'optimisation combinatoire. La gridification d'une méthode de résolution, telle qu'elle est définie dans nos travaux, signifie la prise en compte des caractéristiques des grilles, notamment l'hétérogénéité, la volatilité, la nature multi-domaine d'administration et la grande échelle, en vue de l'adaptation de cette méthode aux grilles informatiques. L'approche B&B@Grid est basée sur une stratégie de parcours de type profondeur d'abord. Cette approche assigne un numéro à chaque sous problème de l'arbre exploré, et définit une relation d'équivalence entre le concept d'intervalle de numéros de sous problème, d'une part, et celui d'ensemble de sous problèmes, d'autre part. Afin de passer d'un concept à l'autre, B&B@Grid étend les algorithmes B&B par deux opérateurs de pliage et dépliage. L'approche B&B@Grid peut être utilisée pour la gridification des méthodes exactes de type B&B avec différents paradigmes. Dans ce papier, nous avons également proposé des stratégies de tolérance aux pannes, d'équilibrage de charge, de détection de terminaison, et de partage des solutions trouvées. Toutefois, l'apport le plus significatif de B&B@Grid concerne la stratégie de répartition de la charge entre les processus B&B. Contrairement aux autres approches, cette stratégie est la seule, à notre connaissance, à prendre en compte la nature hétérogène et volatile des grilles de calcul, ainsi que leur échelle.

Dix expérimentations, d'une heure chacune sur le problème du Flow-Shop avec une grille de 1111 processeurs en moyenne, ont été effectuées pour valider B&B@Grid. Les résultats montrent que le codage de B&B@Grid réduit de 766 fois en moyenne la taille des sous problèmes communiqués. Ces expérimentations ont également montré que le codage des sous problèmes communiqués, réalisé à l'aide de B&B@Grid, est en moyenne 92 fois plus petit que celui de l'approche à jetons de PICO [6]. Ces expérimentations prouvent l'intérêt d'utiliser B&B@Grid dans les grilles pour réduire la taille de l'information communiquée, et par conséquent les délais de communication. Par ailleurs, la sauvegarde de l'ensemble des sous problèmes en cours d'exploration permet à une stratégie de tolérance aux pannes de relancer une résolution en cas de panne. Les résultats démontrent que B&B@Grid améliore d'environ 465 les taux de compression de l'approches à variables publiée dans [10]. Ceci prouve l'efficacité de la stratégie de tolérance aux pannes de B&B@Grid. En outre, malgré le nombre considérable de processeurs de notre grille expérimentale, i. e. 1111 en moyenne, les expérimentations, réalisées avec le paradigme *fermier-travailleur*, ont montré que le fermier exploite son processeur à $1,29\%$ en moyenne, et que les travailleurs passent en moyenne $99,94\%$ de leur temps à résoudre des sous problèmes. Ces deux taux sont de bons indicateurs sur la qualité de la stratégie d'équilibrage de charge de B&B@Grid et sur sa capacité quant au passage à l'échelle. B&B@Grid a été utilisée pour résoudre une instance connue sous le nom de *Ta056* (50 tâches et 20 machines) et publiée dans [15] n'ayant jamais été résolue de manière optimale. L'expérimentation a permis de trouver la solution optimale de *Ta056* avec preuve d'optimalité. En terme de puissance de calcul utilisée, la résolution de *Ta056* se classe en deuxième position parmi les défis à grande échelle réalisés sur les problèmes d'optimisation combinatoire. En

moyenne, 328 processeurs ont été utilisés pendant plus de 25 jours, et un pic d'environ 1200 processeurs a été enregistré pendant cette résolution.

Une des principales perspectives de nos travaux est de trouver des approches permettant d'estimer le nombre de sous problèmes restant à traiter, et d'affiner cette estimation tout au long d'une expérimentation. Au cours d'une résolution, de nombreuses informations utiles sont collectées, notamment le nombre de sous problèmes traités, leurs positions dans l'arbre exploré, et le nombre de solutions explicitement ou implicitement visitées, l'objectif est d'exploiter ces informations pour estimer la durée d'une résolution. Une telle approche permettrait de poursuivre ou d'arrêter une résolution en fonction de cette estimation. D'autres perspectives concernent notamment l'utilisation et l'étude de B&BGrid avec d'autres (1) paradigmes parallèles comme les paradigmes fermier-travailleur hiérarchique et pair-à-pair, (2) algorithmes de type B&X et (3) d'autres problèmes de permutation tels que le problème Q3AP.

Acknowledgments

Les expérimentations présentées dans cet article ont été réalisées sur une grille composée de la plate-forme expérimentale Grid5000 et des machines d'enseignement de l'Université de Lille1 (Polytech'Lille, département FIL de l'IEEA, IUT A). Grid5000 est supportée par l'Action de Développement Technologique Aladdin de l'INRIA ainsi que le CNRS, Renater et certaines universités (voir https://www.grid5000.fr). Merci aux équipes techniques de Grid5000 et des établissements d'enseignement cités.

References

[1] K. Aida and Y. Futakata. High-performance parallel and distributed computing for the BMIeigenvalue problem. Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM, pages 71-78, 2002.

[2] K. Aida, Wataru N. and Y. Futakata. Distributed computing with hierarchical master-worker paradigm for parallel branch and bound algorithm, Proceedings oft the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003), pages 156-163, 2003.

[3] K. Anstreicher, N. Brixius, J.P. Goux and J. Linderoth. Solving large quadratic assignment problems on computational grids. Mathematical Programming, 91(3): 563-588, 2002.

[4] C.A. Bohn and G.B. Lamont. Load balancing for heterogeneous clusters of PCs. Future Generation Computer Systems, 18(3): 389-400, 2002.

[5] P. Bouvry, J.C. de Kergommeaux and D. Trystram. Efficient Solutions for Mapping Parallel Programs. European Conference on Parallel Processing, pages 379-390, 1995.

[6] J. Eckstein, C. A. Phillips, and W. E. Hart. PICO : An object-oriented framework for parallel branch-and-bound. Research Report 40–2000, RUTCOR, 2000.

[7] G. Fedak. XtremWeb : une plate-forme pour l'étude expérimentale du calcul global pair-à-pair. PhD thesis, Université Paris XI, 2003.

[8] R. Finkel and U. Manber. DIB-a distributed implementation of backtracking. ACM Transactions on Programming Languages and Systems (TOPLAS), 9(2): 235-256, 1987.

[9] M.R. Garey, D.S. Johnson and R. Sethi. The complexity of flow-shop and job-shop scheduling. Mathematics of Operations Research, 1: 117-129, 1976.

[10] A. Iamnitchi and I. Foster. A Problem-Specific Fault-Tolerance Mechanism for Asynchronous, Distributed Systems. 29th International Conference on Parallel Processing (ICPP), Toronto, Canada, August, pages 21-24, 2000.

[11] B. Le Cun and C. Roucairol. BOB: A unified platform for implementing branch-and-bound like algorithms. Research Report 95/16, PRiSM Laboratory, University of Versailles - St. Quentin en Yvelines, May 1995.

[12] R. Moe and T. SØREVIK. Parallel Branch and Bound Algorithms on Internet Connected Workstations. High Performance Computing and Communcations, 3726/2005: 768–775, 2005.

[13] R. Ruiz and T. Stutzle. A Simple and Effective Iterative Greedy Algorithm for the Flowshop Scheduling Problem. Technical Report, submitted to European Journal of Operational Research, 2004.

[14] D. Sinclair. The GST load balancing algorithm for parallel and distributed systems. International Journal of Approximate Reasoning, 19(1-2): 39-56, 1998.

[15] E. Taillard. Benchmarks for basic scheduling problems. European Journal of Operational Research, 64: 278-285, 1993.

[16] Y. Xu, T..K. Ralphs, L. Ladányi, and M.J. Saltzman. Alps : A framework for implementing parallel search algorithms. In The Proceedings of the Ninth INFORMS Computing Society Conference, 2005.

Centre de recherche INRIA Lille – Nord Europe
Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399