N° d'ordre : 32/2012-M/MT

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Université des Sciences et de la Technologie Houari Boumediene

Faculté des Mathématiques

Présenté pour obtenir le dipmlôme de MAGISTERE en Mathématiques

Spécialité : Recherche Opérationnelle

Option : Mathémathique de Gestion

Par KHEDIMI Amina

Ordonnancement de type "Flow-Shop" de permutation Bi-Objectif

Résolution exacte sur une grille de calcul

Soutenu publiquement le 25 Janvier 2012 devant le jury composé de

M. OUAFI Rachid, Maître de Conférence à USTHB	Président
M.KHELLADI Abdelkader, Professeur à USTHB	Directeur du mémoire
M. BOUROUBI Sadek, Professeur à USTHB	Examinateur
M.me ELMAOUHAB Aouaouche, Chargée de Recherche au CERIST	Invitée

À mes très chers parents,
à mes frères et sœurs,
et à toute ma famille.

Remerciements

Je profite de ces quelque lignes pour remercier chaleureusement toutes les personnes qui m'ont aidé, de prés ou de loin, à l'élaboration de ce travail.

Je remercie mon directeur de mémoire, le professeur Abdelkader KHELLADI pour sa disponibilité et son assistance.

Je remercie tout spécialement Madame El-MAOUHAB, chef de la division Réseaux du CERIST pour la confiance qu'elle m'a fait en m'intégrant dans son équipe de recherche sur le Grid Computing.

Je tiens à remercier très vivement Monsieur CHERGUI Mohamed El-Amine et Mademoiselle Ait-Mehdi Meriem pour avoir toujours répondu présents à chacune de mes nombreuses demandes et interrogations.

Mes remerciements aux membres de jury pour avoir accepté de juger mon travail. A tous sincèrement.

Résumé

L'optimisation combinatoire regroupe une large classe de problèmes ayant des applications dans de nombreux domaines de l'industrie. Ces problèmes ont souvent été abordés comme des problèmes mono-objectifs alors que la plupart d'entre eux sont de nature multi-objectif. De plus, ils ont toujours été limités par les ressources d'une seule machine : soit par la puissance de calcul disponible. La résolution de problèmes d'optimisation combinatoire de grande taille, tels que les problèmes d'ordonnancement, constitue un vrai défi pour les applications qui s'exécutent sur les grilles de calcul. 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. En pratique, les problèmes d'ordonnancement nécessitent souvent la prise en compte de plusieurs critères. Ces problèmes ont fait l'objet de nombreuses études lorsqu'il s'agit d'optimiser un critère unique mais moins lorsqu'il s'agit de plusieurs critères. C'est dans cette optique que ce mémoire propose, à travers le problème d'ordonnancement de type flow-shop de permutation bi-objectif, une résolution par des méthodes exactes sur une grille de calcul. Cette résolution est optimisée par le traitement parallèle des étapes de l'algorithme d'une part et par le temps de traitement des calculs sur des ressources puissantes.

Mots clés : Optimisation multi-objectif, méthodes exactes, ordonnancement, flow-shop de permutation bi-objectif, grille de calcul, parallélisme.

Résumé

L'optimisation combinatoire regroupe une large classe de problèmes ayant des applications dans de nombreux domaines de l'industrie. Ces problèmes ont souvent été abordés comme des problèmes mono-objectifs alors que la plupart d'entre eux sont de nature multi-objectif. De plus, ils ont toujours été limités par les ressources d'une seule machine : soit par la puissance de calcul disponible. La résolution de problèmes d'optimisation combinatoire de grande taille, tels que les problèmes d'ordonnancement, constitue un vrai défi pour les applications qui s'exécutent sur les grilles de calcul. 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. En pratique, les problèmes d'ordonnancement nécessitent souvent la prise en compte de plusieurs critères. Ces problèmes ont fait l'objet de nombreuses études lorsqu'il s'agit d'optimiser un critère unique mais moins lorsqu'il s'agit de plusieurs critères. C'est dans cette optique que ce mémoire propose, à travers le problème d'ordonnancement de type flow-shop de permutation bi-objectif, une résolution par des méthodes exactes sur une grille de calcul. Cette résolution est optimisée par le traitement parallèle des étapes de l'algorithme d'une part et par le temps de traitement des calculs sur des ressources puissantes.

Mots clés : Optimisation multi-objectif, méthodes exactes, ordonnancement, flow-shop de permutation bi-objectif, grille de calcul, parallélisme.

Abstract

Combinatorial optimisation encompasses a large set of problem with numerous industrial applications. These problems have often been considered as mono-criterion ones while the multi-criterion approach would have been more relevant, on account of the multi-criterion nature inherent to most of them. Solving a large size of combinatorial optimization problems, such as scheduling problems, constitutes a real challenge for applications running on the grid. Indeed, it is necessary to rethink the main algorithms to take into account the characteristics of such environments, including their scale, heterogeneity and dynamic availability of resources. In practice, scheduling problems often require the consideration of several criteria. Yet they were the subject of many studies when it comes to optimizing a single criterion and a lot less when it comes to several criteria.
It is in this context, this paper proposes, through the bi-objective permutation flow-shop scheduling problem, a resolution by exact methods on the grid. This resolution is optimized by the parallelisation of the algorithm's steps on the one part and the execution time on powerful resources.

Keywords : Multicriteria optimisation, exact method, scheduling, bi-objectif permutation flow-shop, grid computing.

Table des matières

Liste des tableaux

Table des figures

Introduction

De nombreux problèmes rencontrés dans différents secteurs économiques, scientifiques et techniques, sont de nature combinatoire, selon le nombre de critères pris en compte, ces problèmes peuvent être mono ou multi-critères. En effet, l'optimisation combinatoire regroupe une large classe de problèmes ayant des applications dans de nombreux domaines de l'industrie, parmi lesquels nous citons :

- Design des systèmes dans les sciences d'ingénieurs (mécanique, aéronautique, chimie, etc.)
- Transport : design de réseaux de transport, tracé autoroutier, etc.
- Ordonnancement et affectation : ordonnancement en production, localisation d'usines.
- Finances : investissements financiers, etc.
- Planification de trajectoires des robots mobiles, etc.
- Agronomie : programme de production agricole, etc.
- Environnement : gestion de la qualité de l'air, distribution de l'eau, etc.
- Télécommunications : design d'antennes, affectation de fréquences, etc.

Ces problèmes issus du monde réel sont complexes et difficiles à résoudre, de plus ils sont rarement mono-objectif, et requièrent souvent la prise en compte de plu-

sieurs critères conflictuels, ce qui amplifie encore le degré de difficulté. En effet, de tels problèmes sont réputés pour être particulièrement des problèmes NP-Difficiles. Contrairement au cas mono-objectif, dans le cas multi-objectif, la notion de solution optimale n'a plus de sens et est remplacée par la notion de solution efficiente ou solution non dominée, dites Pareto optimales. Ainsi, l'optimisation multi-objectif s'intéresse aux particularités liées à l'existence de ces solutions optimales, et aux méthodes de résolution dédiées à ce type de problème.

La résolution d'un problème d'optimisation multi-objectif consiste à trouver la solution Pareto optimale qui répond au mieux aux préférences du décideur. Elle est donc étroitement liée au domaine de la prise de décision multi-critère. Selon la qualité exigée des solutions, les méthodes de résolution de ces problèmes se déclinent en deux grandes catégories : les méthodes approchées, appelées aussi heuristiques, et les méthodes exactes. Il existe également des méthodes hybrides combinant différentes méthodes aux comportements complémentaires.

La résolution de problèmes d'optimisation combinatoire de grande taille, tels que les problèmes d'ordonnancement, peut tirer profit des ressources partagées des grilles de calcul. 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. Les grilles informatiques s'avèrent un moyen efficace pour supporter le coût des méthodes d'optimisation, notamment des méthodes exactes et des méthodes hybrides. Une grille peut être vue comme un ensemble de ressources de calcul hétérogènes, volatiles, réparties sur plusieurs domaines d'administration autonomes et inter-connectés par un réseau à grande échelle.

Les problèmes d'ordonnancement sont présents dans tous les secteurs de l'économie et constituent une fonction importante en gestion de production. Un problème d'ordonnancement consiste à allouer dans le temps des jobs à des ressources existantes tout en satisfaisant un ensemble de contraintes. Le flow-shop est l'un des

problèmes d'ordonnancement les plus étudiés de la littérature. La majorité des travaux qui lui sont consacrés le considèrent sous une forme mono-objectif et vise généralement à minimiser la date de fin de l'ordonnancement ("le makespan"). Dans ce mémoire, l'étude porte principalement sur un problème d'ordonnancement bi-objectif. Les problèmes d'ordonnancement sont très répandus dans notre monde, et sont très souvent de nature multi-objectif [TB02]. Un problème d'ordonnancement consiste à allouer dans le temps des jobs à des ressources existantes tout en satisfaisant un ensemble de contraintes.

Le problème étudié porte plus particulièrement sur un problème de Flow-shop de Permutation Bi-objectif (BOFSP). Le but de ce problème d'optimisation est de définir l'ordre d'exécution d'une série de travaux à réaliser sur plusieurs machines afin de minimiser la date de fin de traitement, ainsi que la somme des retards pondérée accumulés par les différents travaux. Plusieurs méthodes d'optimisation exactes multi-objectifs ont été étudiées dans le cadre de ce travail pour résoudre notre problème. Une difficulté notable pour la résolution du BOFSP et des problèmes multi-objectif en général, est l'absence de relation d'ordre total entre les solutions. L'approche utilisée dans notre travail utilise le concept de dominance Pareto [Par86].

Bien que ce manuscrit soit rédigée en français, nous employons les termes anglais "*Flow-Shop*", "*Job-Shop*" et "*Open-Shop*" pour désigner les structures d'atelier que nous venons de décrire, étant donné qu'il n'existe pas d'équivalents français.

L'organisation de ce document sera la suivante :

Dans le premier chapitre, nous commençons par définir en général l'optimisation combinatoire . Nous décrivons ensuite les particularités de l'optimisation combinatoire Multi-Objectif, pour ceci nous décrivons les solutions composant l'ensemble Pareto. Nous parlons ensuite des approches de résolution, nous rappelons les méthodes exactes et les méthodes approchées et nous donnons un bref aperçu sur les méthodes existantes dans la littérature.

Dans le second chapitre, nous commençons par définir une série de concepts liés au domaine des technologies grilles, nous décrivons ensuite les composants d'une grille de calcul basée sur le l'intergiciel gLite [gLite] ainsi que son fonctionnement. Nous terminons le chapitre par présenter les avantages et les inconvénients d'une telle technologie.

Le troisième chapitre décrit le problème étudié dans ce mémoire : le flow-shop de permutation bi-objectif.
Nous présentons dans un premier lieu les différents problèmes d'ordonnancement existants dans la littérature. Nous rappelons la notation proposée par Graham et al [GLLR79] afin de classer les problèmes d'ordonnancement. Nous verrons ensuite de manière plus approfondie le problème de flow-shop bi-objectif. Nous proposons un modèle mathématique linéaire en nombres entiers. Nous présentons trois méthodes exactes pour résoudre notre problème.

Dans le quatrième chapitre, nous commençons par présenter un schéma de parallélisation des méthodes exactes dédiées pour la résolution de notre problème. Nous présentons l'environnement permettant la mise en œuvre des algorithmes étudiées : nous rappelons la grille d'expérimentation utilisée pour la validation de notre travail, nous parlons aussi de Matlab Parallel Computing toolbox et de Matlab Distributed Computing Server. Ce chapitre se termine par la présentation des expérimentations effectuées.

Nous finirons par donner les conclusions de ce travail. Nous rappelons les résultats obtenus au cours de ce document. Ceci nous amènera à proposer certaines perspectives.

CHAPITRE 1

Optimisation Combinatoire Multi-Objectifs

L'introduction des problèmes de la programmation linéaire multi-objectif à variables entière (MOILP :Multi Objective Integer Linear Programming) a permis d'élargir considérablement le champ d'application du monde réel de la programmation linéaire. En effet, très souvent, un seul critère ne décrit pas avec précision les objectifs fixés par le décideur. L'objectif de ce chapitre est de présenter les notions fondamentales de la programmation linéaire multi- objectif à variables entières, nous parlons ensuite du choix de la méthode d'aide à la décision et nous terminons par un bref aperçu sur les approches de résolution.

1.1 Introduction

De nombreux secteurs de l'industrie (mécanique, chimie, télécommunications, environnement, transport,...,etc) sont concernés par des problèmes complexes de grande dimension et multi-critères mettant en jeu des coûts financiers très importants et pour lesquels les décisions doivent être prises de façon optimale. En effet, les problèmes d'optimisation rencontrés en pratique sont rarement mono-objectif. Il y a généralement plusieurs critères contradictoires à satisfaire simultanément. L'optimisation multi-critère s'intéresse à la résolution de ce type de problèmes. Elle possède ses racines au 19ième siècle dans les travaux en économie de Edgeworth et Pareto [Par86]. Elle a été utilisée initialement en économie et dans les sciences de management, et graduellement appliquée aux sciences pour l'ingénieur. Résoudre un problème d'optimisation combinatoire Mono-Objectif consiste à trouver une solution, appartenant à l'ensemble des solutions réalisables, optimisant la fonction objectif. En optimisation Multi-Objectif, il faut optimiser plusieurs composantes d'une fonction coût où chacune des composantes représente un objectif. Dans ce cas, il n'existe pas une seule solution optimale mais un ensemble de solutions de meilleur compromis. En effet, une solution ayant un coût faible sur l'une des composantes pourra avoir un coût élevé sur les autres. Les solutions de meilleur compromis sont appelées solutions Pareto optimales (c'est-à-dire au sens de la dominance de Pareto [Par86]).

Les méthodes de résolution des problèmes d'optimisation combinatoire mono-objectifs ou multi-objectif peuvent être classées en deux grandes catégories : les méthodes heuristiques et les méthodes exactes. Les méthodes heuristiques ne permettent pas de trouver les solutions (Pareto) optimales exactes mais une approximation de celles-ci. Elle présentent l'avantage de pouvoir être employées sur de très grands problèmes. Les méthodes exactes permettent quand à elles de trouver l'intégralité des solutions (Pareto) optimales mais sont limitées par la taille des instances étudiées.

1.2 L'optimisation combinatoire

Un problème d'optimisation combinatoire est défini par un ensemble d'instances. A chaque instance du problème est associé un ensemble discret de solutions S, un sous-ensemble X de S représentant les solution réalisables (admissibles) et une fonction de coût f (ou fonction objectif) qui assigne à chaque solution $s \in X$ le nombre réel (ou entier) $f(s)$. Résoudre un tel problème (plus précisément une telle instance du problème) consiste à trouver une solution $s^* \in X$ optimisant la valeur de la fonction objectif f. Une telle solution s^* s'appelle une solution optimale ou un optimum global. Nous avons donc la définition suivante :

Une instance I d'un problème de minimisation est un couple (X, f) où $X \subseteq S$ est un ensemble fini de solutions admissibles, et f est une fonction de coût (ou objectif) à minimiser $f : X \to R$. Le problème consiste à trouver $s^* \in X$ tel que $f(s^*) \leq f(s)$ pour tout élément $s \in X$.

Notons que de manière similaire, on peut également définir les problèmes de maximisation en remplaçant simplement $\leq$ par $\geq$.

La difficulté majeure en optimisation combinatoire est due au phénomène d'explosion combinatoire (augmenter la taille de la donnée engendre également une augmentation exponentielle de la taille de l'espace de recherche). En effet, la plupart des problèmes étudiés appartiennent à la classe des problèmes NP-Difficiles. Un problème est "NP-Difficile" si les seules méthodes connues pour le résoudre exigent un temps de calcul exponentiel en fonction de la taille de l'instance. Autrement dit, c'est un problème pour lequel aucun algorithme de résolution polynomial en temps n'est connu (il est communément admis qu'il n'en existe pas, mais ceci n'a pas été prouvé).

Cette explosion combinatoire empêche donc d'explorer l'ensemble de l'espace de recherche en un temps raisonnable (quelques heurs, voir quelques jours).

Les méthodes utilisées en optimisation combinatoire peuvent être classées en deux grandes catégories :

- d'un côté, nous avons les méthodes exactes permettant de trouver la (ou les) solution(s) optimale(s).
- d'un autre côté, nous avons les méthodes approchées permettant de trouver une solution de bonne qualité mais non-optimale.

Les méthodes exactes, afin de résoudre ces problèmes de grandes complexité, coupent des parties non-intéressante de l'espace de recherche. Elles permettent de trouver la solution optimale mais sont limitées dans la taille des problèmes qu'elle résolvent.

Les méthodes approchées effectuent des recherches guidées à l'intérieur de l'espace de recherche afin de caractériser rapidement une solution de bonne qualité. Avec ces méthodes, la solution trouvée n'est pas optimale (ou si elle l'est, la preuve n'est pas faite).

D'une manière générale, il est à noter que les problèmes d'optimisation rencontrés sont de nature Multi-Objectif car il existe généralement plusieurs critères permettant d'évaluer une solution (coût de production, rapidité de production, retard des différentes tâches...). C'est pourquoi dans la suite de ce document, nous nous intéressons à l'optimisation combinatoire multi-objectif.

1.3 L'optimisation combinatoire Multi-Objectif

Dans cette section, nous verrons dans un premier temps quelques particularités de l'optimisation combinatoire Multi-objectif. Nous présenterons ensuite un résumé rapide des principales méthodes de résolution. Résoudre un problème d'optimisation combinatoire Multi-Objectif consiste à optimiser les composantes d'une fonction coût vectorielle dont chacune de ses composantes est un objectif. Un problème d'optimisation Multi-Objectif peut se définir par

$$\begin{cases} \min f(x) = (f_1(x), f_2(x), \ldots, f_k(x)) \\ x \in S \end{cases}$$

où k est le nombre des objectifs ($k \geq 2$), $x = (x_1, x_1, \ldots, x_n)$ est le vecteur représentant les variables de décision, S représente l'ensemble des solutions réalisables .

A chaque solution $x \in S$ est associé un vecteur objectif $z \in Z$ sur la base d'un vecteur de fonctions $f : S \to Z$ tel que $z = (z_1, z_2, ..., z_k) = f(x) = (f_1(x), f_2(x), \ldots, f_k(x))$, où $Z = f(S)$ représente l'ensemble des points réalisables de l'espace objectif (FIGURE 1.1). Chacune des fonctions $f_i(x), i = 1, \ldots, k$ est à optimiser, c'est-à-dire à minimiser ou à maximiser. Sans perte de généralité nous supposerons par la suite que nous considérons des problèmes de minimisation. La figure 1.1 montre un exemple avec deux variables de décision et trois objectif.

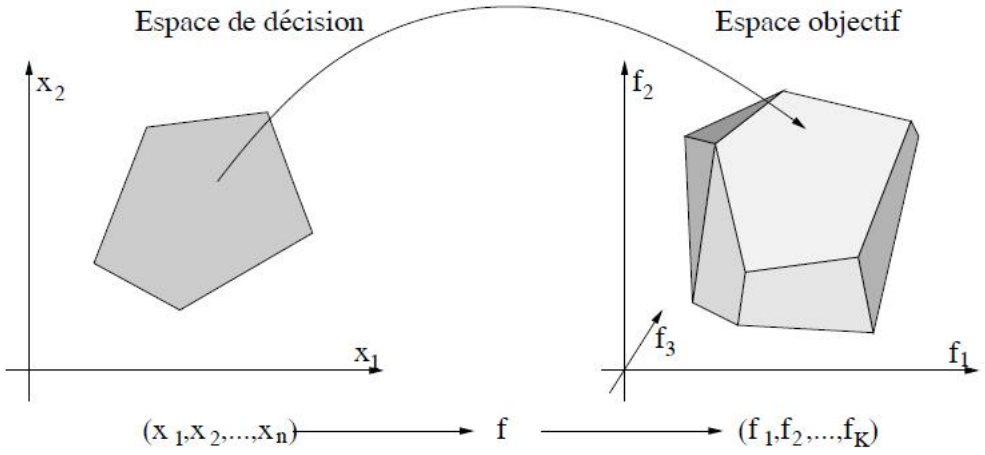

FIGURE 1.1 – Espace de décision et espace objectif d'un problème d'optimisation multiobjectif

1.3.1 Les Solutions d'un Problème Multi-Objectif

1.3.1.1 Solutions optimales et solutions de Pareto optimalité

Un problème d'optimisation Multi-Objectif n'admet pas une solution optimale unique mais un ensemble de solutions optimales dites de meilleures compromis. En effet, les solutions n'admettent pas une relation d'ordre totale, une solution pouvant être meilleure qu'une autre sur certains objectifs et moins bonne sur d'autres.
Il faut donc utiliser une notion d'optimalité permettant de définir un ensemble de solutions optimales. La notion la plus utilisée a été introduite par Edgworth [Pap76] puis généralisée par Pareto [Par86]. Nous parlerons donc d'optimum de Pareto et de solution Pareto optimale(i.e., optimale au sens de Pareto). Nous parlerons aussi de l'ensemble des solutions Pareto optimales. Il nous faut maintenant caractériser la notion de dominance entre deux solutions.

Définition 1.1. Dominance : Dans un problème de minimisation avec k objectifs, une solution x domine une solution $x^{'}$ si et seulement si :

$$\begin{cases} \forall i \in [1...k], f_i(x) \leq f_i(x^{'}) \\ \exists i \in [1...k], f_i(x) < f_i(x^{'}) \end{cases}$$

Nous notrons $x \leq x^{'}$

La figure 1.2 présente pour une solution x donnée, l'espace qui domine cette solution, l'espace dominé par cette solution et les deux espaces qui n'admettent pas de relation d'ordre avec x au sens de Pareto, pour un problème bi-objectif. Si pour deux solutions x et $x^{'}$, $x \nleq x^{'}$, et $x^{'} \nleq x$, alors nous noterons $x \sim x^{'}$.

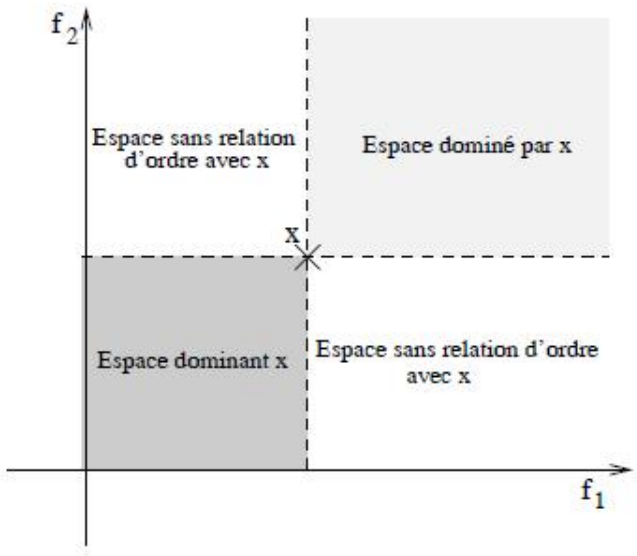

FIGURE 1.2 – Notions de dominance

Définition 1.2. Vecteur non-dominé : Un vecteur objectif $z \in Z$ est non-dominé si et seulement si il n'existe pas un vecteur $z^{'} \in Z$ tel que $z^{'}$ domine z.

Définition 1.3. Solution efficace ou Pareto optimale : Une solution x est dite efficace ou Pareto Optimale si est seulement si elle n'est dominée par aucune autre solution réalisable.

Définition 1.4. Ensemble Pareto optimal : Pour un problème d'optimisation donné, l'ensemble des solutions Pareto noté (P^*) est défini par :

$$P^* = \{x \in S | \nexists x^{'} \in S; x^{'} \leq x\}$$

La figure 1.3 donne un ensemble représentant l'image dans Y des solutions de l'ensemble Pareto (ensemble des solutions non-dominées) pour un problème bi-objectif, ceci est appelé Front Pareto.

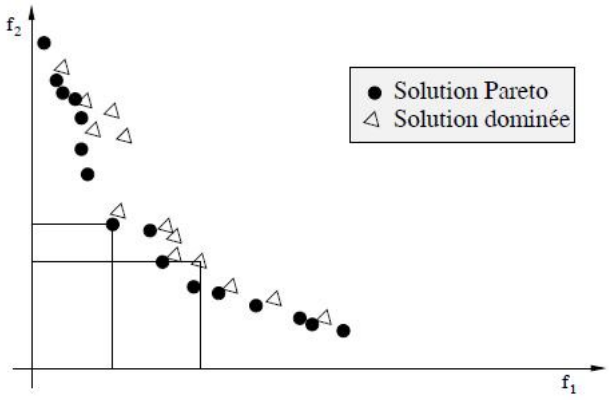

FIGURE 1.3 – Exemple de front Pareto Optimal

1.3.1.2 Points Particuliers

Il existe des points particuliers dans l'espace des objectifs qui permettent de borner l'ensemble Pareto ou encore de discuter de l'intérêt des solutions trouvées. Ces points peuvent être réalisables ou non (La figure 1.4 donne une visualisation des ces solutions).

1. Le point Idéal z^I, est défini par : $z^I = (f_1(z^I), ..., f_k(z^I))$, avec $f_q(z^I) = \min_{x \in S} f_q(x)$, $\forall q \in [1, \ldots, k]$.

   Ce point a pour chaque objectif la valeur optimale de cet objectif. Il nous donne donc une borne inférieure du front Pareto.

2. Le point Utopique, z^U, est défini par : $z^U = z^I - \epsilon U$, avec $\epsilon > 0$ et U le vecteur unitaire $U = (1, 1, ..., 1) \in R^k$, ce point sera donc un point dominant le point Idéal, il sera bien entendu non réalisable.

3. Le point Nadir, z^N, est défini par $z^N = (f_1(z^N), ..., f_K(z^N))$, avec $f_q(z^N) = \max_{x \in P^*} f_q(x)$, $\forall q \in [1, \ldots, k]$, avec P^* représente l'ensemble des solutions Pareto optimales.

   Cette solution représente pour chaque objectif la solution appartenant à l'ensemble Pareto ayant la valeur la plus mauvaise sur cet objectif. Elle nous donne donc une borne supérieure du front Pareto. Dans le cas Bi-objectif cette solu-

tion est définie par $f_1(z^N), f_2(z^N)$, avec $\forall i \in [1,2]\ f_j(z^N) = f_j(z^I)$, où $j \neq i$. Il apparaît donc que la recherche du point Nadir en Bi-Objectif ne demande aucun calcul supplémentaire si le point Idéal est connu.

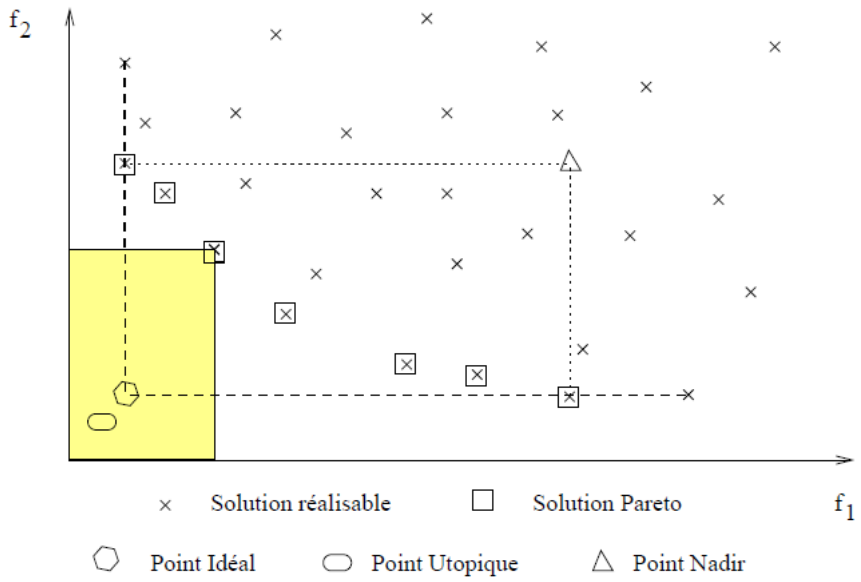

FIGURE 1.4 – Points particuliers du front Pareto

Nous voyons que le point Idéal et le point Nadir permettent de borner l'espace de recherche pertinent. Dans le cadre des méthodes exactes, le but est de limiter l'espace de recherche à ces seules parties pertinentes. Le point Idéal et le point Nadir seront donc primordiaux dans de nombreuses méthodes exactes.

1.3.2 Caractérisation du front Pareto

1.3.2.1 Solutions supportées/non supportées

Le front Pareto comporte plusieurs solutions de meilleur compromis appelées solutions Pareto optimales. Nous allons caractériser les différents types de solutions composant ce front. Deux types de solutions le composent.

Les solutions supportées : leurs images dans Y se trouvent sur l'enveloppe convexe

de Y. Chacune des ces solutions peut êtres trouvée en optimisant une agrégation linéaire des objectifs (voir le théorème de Geoffrion [Geo68]),

Les solutions non-supportées : Elle sont Pareto optimales mais leurs images dans Y ne sont pas situées sur l'enveloppe convexe de Y. Aucune des ces solutions ne peut être trouvée en optimisant une agrégation linéaire des objectifs (corollaire du théorème de Geoffrion [Geo68]).

Nous pouvons donc avoir des cas extrêmes dans lesquels la frontière Pareto est totalement convexe ou concave (voir la figure 1.5). Généralement, le front Pareto sera composé de solutions supportées et non supportées (voir la figure 1.6).

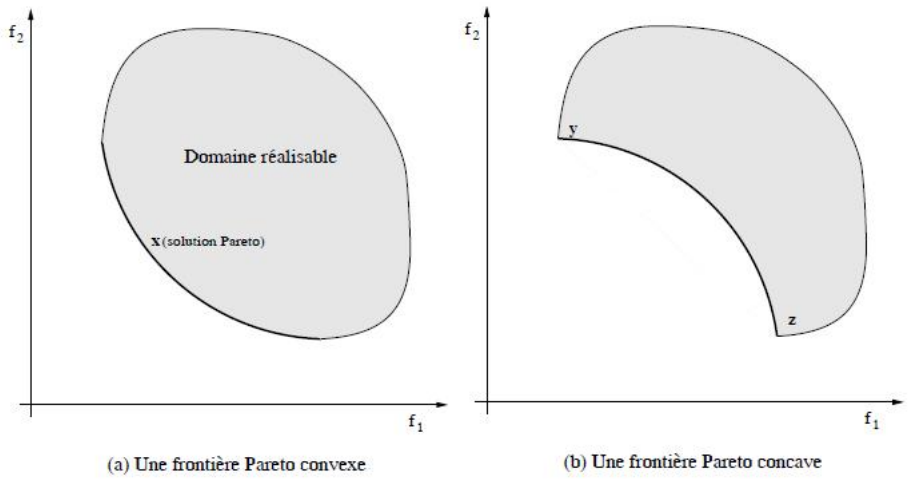

FIGURE 1.5 – Front Pareto Convexe/Concave

.

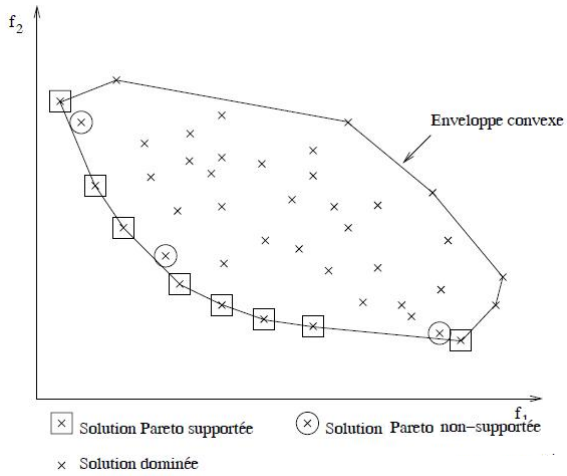

FIGURE 1.6 – Représentation des différents types de solutions en bi-objectif

Les solutions non-supportées ne se trouvent pas sur l'enveloppe convexe et ne sont la solution optimale d'aucune agrégation d'objectifs. Il peut donc paraître légitime de se demander si elle doivent être recherchées ou non.

1.3.2.2 Ensemble Pareto minimal complet/Ensemble Pareto maximal complet

Un ensemble Pareto est dit complet si toutes les valeurs Pareto pouvant être obtenues sont représentées parmi les solutions de l'ensemble. Certaines méthodes, ne permettent pas de trouver l'ensemble Pareto complet (ex : la méthode par agrégation).

Il est important de remarquer que le nombre de solution Pareto dépend de l'espace considéré (espace décisionnel/espace objectif). En effet, deux solutions différentes de l'espace décisionnel peuvent avoir le même vecteur coût et donc représenter le même point de l'espace des objectifs. L'ensemble Pareto de l'espace décisionnel est appelé ensemble Pareto maximal et celui de l'espace objectif est appelé ensemble Pareto minimal.

Il est donc nécessaire de définir l'ensemble recherché car cela fera varier le nombre

de solutions Pareto.

1.4 Choix de la méthode d'aide à la décision

Nous avons vu précédemment qu'un problème Multi-Objectif n'admet pas une solution optimale unique mais un ensemble de solution Pareto optimales. Il est donc nécessaire de faire intervenir l'humain à travers un décideur pour le choix de la solution finale à implementer.

Ainsi, avant de lancer la résolution d'un problème Multi-Objectif, il faut se poser la question du moment où le décideur intervient. Il existe trois types d'approches distinctes pour la résolution d'un problème Multi-Objectif :

1. *A priori* : Le décideur oriente la recherche dés le début en donnant des préférences (ex : agrégation) à chacun des objectifs. Avec cette méthode une solution unique est obtenue. Celle-ci ne convient pas forcément à ce qui était souhaité ; par exemple, les solutions non-supportées ne peuvent être atteintes par agrégation. En revanche, l'obtention du résultat est assez rapide.
2. *Interactive* :Le décideur doit interagir avec le programme. Dans cette méthode, le programme propose régulièrement une solution au décideur qui devra alors réorienter la recherche en fonction de ses objectifs. L'inconvénient de cette méthode réside dans le fait que le décideur doit attendre chaque étape avant d'obtenir à nouveau la main(ceci peut être long).
3. *A posteriori* : Le décideur obtient ici l'ensemble du front Pareto afin qu'il choisisse la solution la plus adaptée. L'obtention du front complet peut s'avérer très longue et le nombre des solution peut être très grand et entraîne de ce fait un problème de décision de la solution à choisir.

Nous avons vu que chacune de ces approches présente des inconvénients et des avantages. Dans ce document, nous nous plaçons dans une approche *a posteriori*qui permet au décideur de faire le choix le plus approprié. Ce type d'approche peut

se décomposer en deux étapes : une phase de recherche de l'intégralité du front Pareto (phase d'optimisation)et une phase de décision. Nous ne considérons par la suite que le problème d'obtention du front Pareto. Nous avons défini les problèmes Multi-Objectifs, voyons maintenant les approches de résolution existantes.

1.5 Approches de résolution

La résolution d'un problème multi-objectif consiste à déterminer soit l'ensemble des solutions efficaces dans l'espace des décisions noté *EFF*, soit l'ensemble des solutions non dominées dans l'espace des critères noté *SND*. Dans la littérature, l'accent est mis sur la recherche de l'ensemble des solutions non dominées compte tenu de son cardinal qui est moins import que celui des solutions efficaces ; plusieurs solutions efficaces pouvant donner lieu à un même vecteur non dominé [Che10]. Nous pouvons différencier deux grandes classes de résolution des problèmes d'optimisation et plus particulièrement des problèmes d'optimisation Multi-Objectif. D'un côté, les méthodes approchées qui ne permettent pas de trouver l'ensemble des solutions Pareto optimales (ou si elles les trouvent, ne peuvent prouver son optimalité) mais essaient de donner un ensemble de solutions de bonne qualité (notion convergence)et bien diversifiés (notion de diversité). Ces approches ne sont que très peu influencées par la taille des données traitées. D'un autre côté, les méthodes exactes qui permettent de trouver des solutions exactes avec preuve d'optimalité.

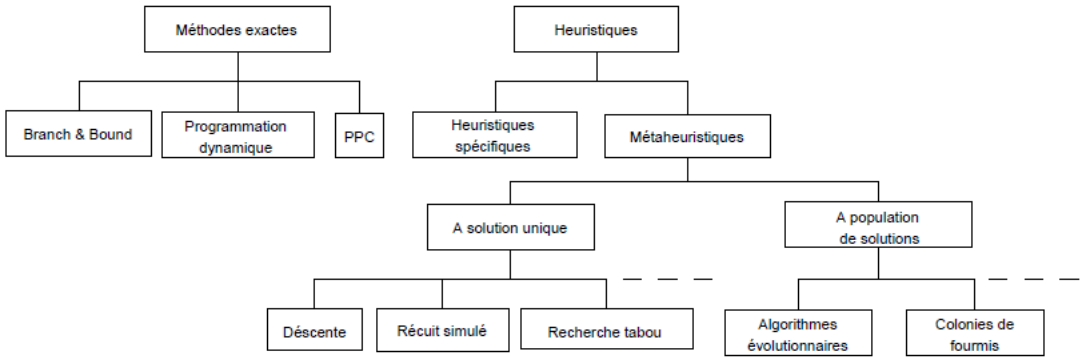

FIGURE 1.7 – Une taxinomie des méthodes de résolution en optimisation combinatoire

Dans ce document nous travaillons sur les méthodes exactes, il est toutefois important de présenter les méthodes approchées car elle apparaissent comme complémentaires aux méthodes exactes en vue de :

- soit d'améliorer le temps de recherche des méthodes exactes.
- ou d'améliorer la qualité des solutions de l'heuristique.

1.5.1 Les méthodes approchées

Parmi les différentes méthodes approchées, nous avons la classe des méthodes dites méta-heuristiques, parmi celles-ci deux classes peuvent être distinguées, les approches à solution unique et les approches à base de population.

Les approches à solution unique sont basées sur un algorithme de recherche de voisinage qui commence avec une solution initiale. Cette solution est ensuite améliorée pas à pas en choisissant une nouvelles solution dans son voisinage. La plus connue est la recherche locale [PS82][KGV83], qui à chaque pas de la recherche progresse vers une solution voisine de meilleure qualité. Cette méthode présente le désavantage d'être bloquée par les optima locaux. Pour pallier à ce problème d'autre méthodes comme le recuit simulé [KGV83] ou la recherche Tabou [Glo86][Han86] ont la par-

ticularité de pouvoir sélectionner une solution voisine dégradant le coût. Le recuit simulé utilise une probabilité permettant parfois de choisir une solution dégradant le coût, cette probabilité décroît avec le temps. Quant à la recherche tabou, elle n'est pas stochastique mais elle garde en mémoire la liste des solutions déjà visitées ou des mouvement déjà effectués afin d'éviter de faire des cycles.

Les approches à base de population de solutions améliorent, au fur et à mesure des itérations, une population de solutions. L'intérêt de ses méthodes est la diversité apportée par la population. Les plus connues sont les algorithmes génétiques [Hol75] qui sont basés sur des mécanismes biologiques tel que les lois de Mendel et sur le principe de sélection de Darwin [Dar]. Ces algorithmes se basent sur deux opérateurs : la mutation et le croisement. Les algorithmes génétiques sont réputés pour avoir une convergence (vers un ensemble Pareto) lente, c'est pour quoi ils sont souvent remplacés par les algorithmes mimétiques [Mos89][CDG]. Les algorithmes mimétiques sont classés par certains comme étant des algorithmes hybrides faisant coopérer recherche locale et algorithme évolutionnaire. Dans ces algorithmes l'opérateur de mutation est remplacé par un algorithme de recherche locale. Une autre approche connue est l'approche de colonies de fourmis [LTZ04][Fou85][UT95] qui est une méthode d'optimisation basée sur l'apprentissage par renforcement.

Les approche à base de population de solutions apparaissent comme bien adaptées pour les problèmes Multi-Objectif. En effet la notion de population de solutions est adaptée avec la notion d'ensemble de solutions Pareto optimales. De plus, la population des solutions permet de traiter efficacement un des deux buts des méthodes heuristiques Multi-Objectif, la diversité (l'autre étant la convergence).

Les méthodes heuristiques ne permettent pas de trouver les solutions exactes mais seulement une approximation de l'ensemble des solutions Pareto. Dans la section suivantes, nous verrons les méthodes exactes Multi-objectif qui, elles, per-

mettent de trouver l'intégralité de l'ensemble Pareto.

1.5.2 Les méthodes exactes

Concernant les méthodes exactes, plusieurs approches basées sur des procédures de séparation et évaluation (Branch and Bound) [UT95], sur l'algorithme $A\overline{*}$ [Ste91] et la programmation dynamique [CMM90] ont été proposées pour résoudre de petits problèmes à deux objectifs (problèmes bi-objectifs). Une approche particulière pour l'optimisation multi-objectif est le "goal programming" (programmation par but) [San94]. Dans ce type d'approche, le décideur indique une valeur cible (but) et l'objectif est de minimiser l'écart avec cette cible. Souvent la programmation par buts est vue comme une discipline en elle-même, différente de l'optimisation multi-objectif. Une approche intéressante a été proposée par B. Ulungu et J. Teghem [UT95] pour la recherche du front Pareto du problème bi-objectifs . Leur méthode en deux phases consiste dans un premier temps à rechercher l'ensemble des solutions Pareto supportées, puis dans un deuxième temps à chercher de façon indépendante les solutions non-supportées situées entre tous les couples de solutions supportées adjacentes. Nous aborderons plus en détails cette méthode dans le troisième chapitre. Généralement, les méthodes exactes Multi-Objectif se ramènent à la résolution de plusieurs problèmes Momno-Objectif. La méthode $\epsilon - contraintes$ est l'une des méthode qui se base sur ce principe.

1.5.2.1 La méthode $\epsilon - contraintes$

Cette méthode proposée par Laumanns et al [LTZ04] est une généralisation de la méthode $\epsilon - contraintes$ pour un nombre quelconque d'objectifs.
Elle est basée sur des recherches mono-objectif bornée(supérieurement et inférieurement). Ces recherches sont lexicographiques dans l'ordre des objectifs. D'abord, une recherche est effectuée dans l'espace total. Puis chaque solution trouvée permet de découper l'espace de recherche en sous-espaces de taille inférieure. Si aucune solution

n'est trouvée alors il est prouvé qu'il n'existe pas de solution Pareto dans un espace donné.

L'algorithme gère une hyper-grille de dimension $k-1$, qui découpe l'espace de recherche en hyper-rectangles parallèles aux axes représentés par les objectifs $f_2, ..., f_k$. La grille de départ est entre plus au moins l'infini, elle contient l'ensemble de l'espace objectif potentiel, $\mathbb{R}^k$. Puis, à chaque itération de l'algorithme une nouvelle solution Pareto est recherchée dans une case de la grille. Si une solution est trouvée alors la grille est découpée en fonction de cette solution, sinon la case suivante est visitée.

Supposons un problème à k objectifs. Notons $(f, \epsilon, \epsilon^{'})$(où $\epsilon, \epsilon^{'} \in Re^{K-1}$) le problème suivant :

$$\left[\begin{array}{l} lex\min_{x\in\Omega} f(x) \\ avec \quad \epsilon_i < f_i(x) \leq \epsilon_i^{'}, \forall i \in 2, ..., k \end{array}\right]. \tag{1.1}$$

où *Lexmin* est une recherche lexicographique, rappelons que cette méthode proposée par Fourman [Fou85], classe le objectifs en fonction d'un ordre d'importance. En suite les fonctions objectif sont traitées dans cet ordre, c'est-à-dire que le premier objectif (dans l'ordre d'importance) est d'abord optimisé. Ensuite la valeur de cet objectif est conservée et le deuxième objectif est optimisé. Ceci est réalisé itérativement jusqu'à avoir optimisé tous les objectifs tout en ayant conservé les valeurs trouvées pour les précédents.

Notons $opt(f, \epsilon, \epsilon^{'})$, la solution optimale à ce problème. La figure 1.8 présente un exemple de trois objectifs. Les deux premières recherches permettent de trouver les solutions x^1 et x^2, qui permettent elles-mêmes de découper l'espace de recherche en 9 sous-espaces représentés à la figure 1.8.a.

Les cases de la grille correspondent à la projection de ces sous-espaces sur le plan formé par les objectifs f_2 et f_3, elles sont numérotées en fonction de leur ordre de visite. Les deux premières recherches, dans la case 0 (par la résolution du problème $(f, (5, 4), (\infty, \infty))$) et la case 1 (par la résolution du problème $(f, (5, 2), (\infty, 4))$, ne

donnent pas de nouvelles solutions (voir Fig 1.8.b). La recherche dans la case 2 (la résolution du problème $(f, (5, -\infty), (\infty, 2)))$ trouve la solution x^3. Les deux nouvelles contraintes sont alors rajoutées.

Nous pouvons remarquer que cet algorithme demande une recherche par point du front Pareto ce qui peut prendre un temps de calcul important, surtout si les objectifs sont NP-Difficiles dans le cas Mono-Objectif. De plus, l'algorithme demande de pouvoir réaliser des recherches bornées inférieurement et supérieurement. Lorsque la méthode Mono-Objectif utilisée n'est pas une méthode de programmation linéaire, par exemple lorsqu'il s'agit d'un Branch & Bound ou d'une méthode dédiée, ceci devient généralement impossible.

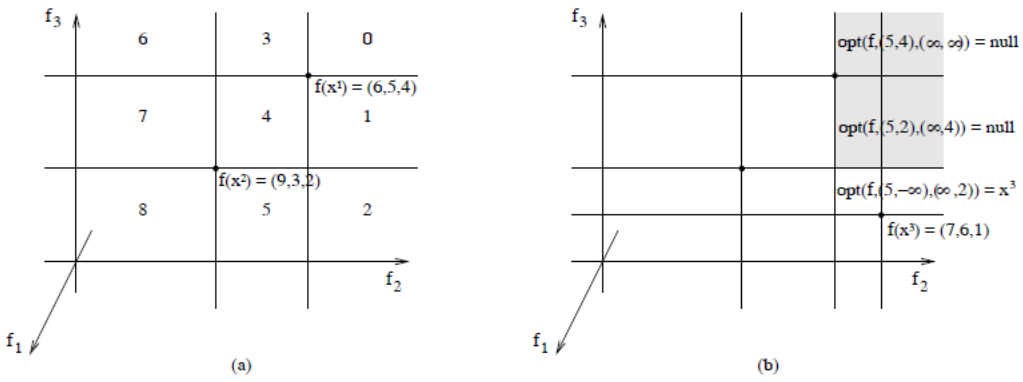

FIGURE 1.8 – La méthode $\epsilon - contraints$ à k objectifs

1.6 Conclusion

Les problèmes d'optimisation combinatoire Multi-Objectif sont réputés difficiles. En effet, il n'existe pas de relation d'ordre total sur le coût des solutions, et par conséquent, il existe un ensemble de solutions de meilleures compromis. Pour caractériser les solutions de meilleurs compromis, nous avons vu la notion de dominance de Pareto, nous parlons donc de solutions Pareto optimale. Nous avons vu les différents points particuliers permettant de caractériser le front Pareto, ainsi que le choix de la méthode d'aide à la décision à adopter.

Les méthodes exactes permettent de trouver l'intégralité des solutions Pareto optimales, de plus, la plupart de ces méthodes demandent une recherche par solution Pareto optimale. Si ces recherches sont elles-mêmes NP-Difficiles, alors, le temps pris pour trouver l'intégralité du front Pareto risque d'être très important.
En effet, la résolution de ce type d'instances nécessite une puissance de calcul considérable. Les grilles de calcul sont un moyen pour réunir une telle puissance de calcul et le recours à cette technologie est nécessaire pour la résolution de telles instances.

CHAPITRE 2

Les Grilles de calcul

Le besoin de puissance de calcul et de stockage de plus en plus important a guidé et guide encore l'évolution des architectures matérielles pour le calcul scientifique. Par besoin de puissance de calcul plus importante, il s'agit de l'exigence des applications scientifiques en terme de rapidité et d'efficacité de calcul. Ce chapitre présentera dans un premier temps le concept de grille ainsi que quelques bases théoriques qui l'entourent. Nous verrons ensuite les éléments principaux d'une grille basée sur gLite, support de notre travail expérimental, et comment soumettre une application sur une telle infrastructure. Enfin, nous terminerons par présenter les avantages et les inconvénients d'une telle technologie.

Au cours de ces deux dernières décennies, les systèmes distribués et parallèles ont connu une évolution significative sur les plans matériel et logiciel. En peu de temps, ce domaine est passé des super calculateurs aux architectures variées (vectorielles, à mémoire partagée ou distribuée) aux grilles informatiques, en passant par les réseaux (Networks of Workstations ou NOW) et grappes (Clusters of Workstations ou COW) de stations de travail et les méta-systèmes. L'engouement suscité aujourd'hui par les grilles est dû à plusieurs facteurs, notamment à : l'émergence et la popularité grandissante d'Internet et des technologies associées, et à l'évolution technologique et économique des ressources informatiques en termes de puissance de calcul, de capacité de stockage, de bande passante, et de prix de plus en plus abordables. Cet engouement a déjà conduit à la prolifération de projets sur les grilles avec différentes incarnations traduisant différentes approches : méta-calcul ou metacomputing, super-calcul virtuel ou virtual supercomputing, calcul global ou global/Internet computing, systèmes pair-à-pair ou peer-to-peer computing, etc.
Historiquement, l'idée des grilles est apparue en 1995 avec le projet I-WAY [DFPSK96] visant à construire un méta-système multi-site et une infrastructure logicielle permettant son exploitation pour le traitement d'applications haute-performance. Le méta-système inter-connecte des périphériques de visualisation (caves de réalité virtuelle immersive et instruments d'observation) et des super-calculateurs répartis sur 17 sites. L'idée a été étendue à tous types de ressources informatiques incluant les applications, le stockage et les masses de données. Il existe trois catégories de grilles :

- Les grilles d'information : elles permettent de partager la connaissance. Le web est un parfait exemple de grille d'information. Il donne aux internautes la possibilité de pouvoir accéder aux données de façon transparente sans avoir à se soucier de la situation géographique ni du type de support sur lequel elles sont stockées.
- Les grille de stockage : Ce type de grille permet le stockage de très grands volumes d'information. Elles sont souvent, mais pas toujours, combinées avec

les grilles de calcul.

– La grille de calcul : C'est sur la grille de calcul que s'exécutent les programmes, et celle à laquelle nous nous intéressons dans cette étude.

2.1 La Grille de Calcul

2.1.1 L'origine

L'origine de la terminologie *grid computing* provient du réseau électrique *electric power grid*. Au début du siècle dernier, chaque utilisateur produisait sa propre électricité pour la consommer sur place. Par la suite, la mise en place d'un réseau de distribution a permis une utilisation de la puissance électrique à la demande sans se soucier de son origine ni de la façon dont elle-ci était produite[Des06].
Par analogie avec ce réseau électrique, le concept de grille de calcul ou *grid computing* représente le fait de pouvoir disposer des ressources informatiques à la demande sans se soucier de leur situation géographique.
La notion de grille de calcul est née dans les dernière années du vingtième siècle. Grâce au succès de l'*Internet* et du protocole IP, nous pouvions alors compter sur l'infrastructure d'un réseau planétaire d'une densité phénoménale. Le monde académique, en manque de puissance informatique pour faire face aux besoins de la recherche, a commencé à imaginer de nouvelles pistes de solutions.

2.1.2 Définition

Il est assez difficile de définir de façon précise ce qu'est une grille de calcul tant il existe des définitions différentes dans la littérature actuelle. Parmi les nombreuses définitions, celle donnée dans l'ouvrage fondateur de I. Foster et C. Kesselnam [FK99].
grille de calcul (computing grid) :

– Infrastructure matérielle et logicielle qui fournit un accès fiable, cohérent, omniprésent et peu coûteux à des capacités de calcul haut de gamme.

Nous allons toutefois, proposer une définition qui reflète notre vision des grilles et qui soit en correspondance avec la description matérielle de l'infrastructure et avec son utilisation. : grille de calcul - Un système distribué composé de ressources informatiques partagées. Elle est le résultat d'une mise en commun de ressources hétérogènes appartenant à plusieurs organisations. Une grille offre à l'utilisateur une vision cohérente de ces ressources.

Du point de vue de la composition matérielle, nous considérons une grille comme des ressources hétérogènes reliées par des réseaux longue distance (Wide Area Network, WAN). Ces ressources sont situées dans des lieux géographiques différents (sites). Ils peuvent être des ordinateurs multi-processeurs, des grappes, des PCs individuels.

2.2 Caractéristiques des grilles de calcul

2.2.1 La grille est multi-domaine d'administration

Les ressources sont réparties sur plusieurs domaines d'administration et gérées par différentes organisations. Les utilisateurs et fournisseurs de ressources sont clairement identifiés atténuant ainsi les problèmes de sécurité.
Dans les systèmes de calcul global tels que XtremWeb [Fed03] basés sur le vol de cycles étendu à Internet, ce problème est résolu de manière naturelle puisque la communication est initiée par les machines volontaires "de l'intérieur d'un domaine d'administration". La résolution du problème n'est pas toujours aussi simple, et cela constitue souvent le défi des systèmes de metacomputing.

2.2.1.1 La grille est hétérogène

L'hétérogénéité des ressources matérielles et logicielles est accentuée par le nombre important de machines dans la grille et appartenant à différentes organisations. La

résolution de ce problème en utilisant les standards déchange récents tels que XML et SOAP constitue, entre autres, un facteur ayant fortement encouragé l'émergence des grilles. Néanmoins, l'hétérogénéité rend particulièrement plus difficile l'évaluation de performances des applications déployées, et constitue un thème de recherche à part entière [GWBG04][NGB04].

2.2.2 La grille a une large échelle

la grille a une échelle importante en termes de nombre de machines potentiellement disponibles et de la taille du réseau d'interconnexion de ces machines. Suivant l'échelle visée, on distingue souvent les systèmes de calcul global et pair-à-pair des grilles de calcul. Les premiers supportent pour le moment bien plus de machines que les grilles de calcul. Ils sont destinés en particulier aux applications basées sur le volontariat de milliers voire de millions de machines d'Internet.

2.2.3 La grille est dynamique

La disponibilité variable des ressources due à leurs pannes ou aux départs/retours de leurs propriétaires n'est pas une exception mais une règle dans la grille. En effet, avec un nombre si important de ressources, la probabilité de disparition des ressources est plus importante. Cette volatilité des ressources pose des problématiques de découverte dynamique de ressources, de tolérance aux pannes, de sauvegarde/restauration des données, de synchronisation, etc. Ces problématiques sont souvent difficiles à gérer de manière transparente et efficace dans les intergiciels(nous donnerons une définition de l'intergiciel dans la section 2.5). C'est pourquoi, leur prise en compte est parfois nécessaire au niveau applicatif.

2.3 Grilles de calcul et grappes de calcul

Si sur un plan conceptuel, il n'y pas de grandes différences entre les grilles de calcul et les grappes de calcul, les deux approches réalisent un système pouvant être vu comme une machine disposant d'une architecture parallèle. La réalisation de grilles de calcul est plus complexe pour les raisons suivantes : Dans une grappe de calcul, le nombre de machine participant est fini, seul les cas de panne et remplacement d'un élément de la grappe sont à prendre en compte. Par opposition dans une grille de calcul le nombre de machine est variable, les machines entrent et sortent de la grille selon les choix de l'utilisateur de la machine et non ceux de la grille de calcul. De plus le nombre de machines travaillant dans une grappe de calcul avoisine les deux-cents micro-ordinateurs alors que dans une grille de calcul le nombre dépasse le millier de machines. Les machines sont identiques, même capacité de traitements, même système dans une grappe de calcul. Dans la grille de calcul, les machines participantes sont hétérogènes aussi bien en terme de capacité de traitement que du système d'exploitation supportant la grille de calcul. Enfin dans une grappe de calcul, l'interconnexion des machines est très bien maîtrisée et dispose d'une grande capacité de transport mais aussi en général d'une gestion de la qualité de service, ce qui permet d'obtenir la puissance d'un calculateur à moindre coût. Une grille de calcul dispose d'un réseau très hétérogène en terme de capacité de transport (56Kbps et plus), sans possibilité de gestion de la qualité de service dans la plupart des cas.

2.4 Intérêts d'une grille de calcul

L'intérêt que l'on porte aux grilles de calcul à l'heure actuelle porte sur plusieurs points. Tout d'abord, avoir la possibilité d'effectuer des calculs plus important que sur un cluster isolé peut constituer un premier objectif. La quantité de RAM que contiennent les noeuds représente une limite dans l'utilisation des processeurs ; et

de ce fait, plus la simulation est de taille importante, plus le nombre de processeurs nécessaires à l'exécution est grand. Il est donc appréciable d'avoir accès à des ressources de calcul supérieures à celles que l'on peut généralement avoir dans les centres de recherche ou de production à un coût qui reste raisonnable. Relier des clusters de production indépendants entres eux semble plus avantageux que d'acquérir une machine contenant la puissance réunie de cette manière.

2.5 Les principes de base de la grille

2.5.1 Constitution d'une grille de calcul :

Une grille de calcul est toujours basée sur un principe d'échanges clients/serveurs. Une grille de calcul est alors composée de deux entités :

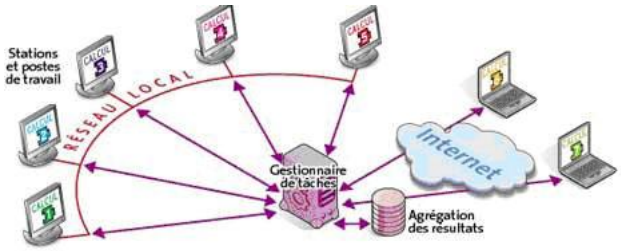

FIGURE 2.1 – Constitution d'une grille de calcul

- Un ou plusieurs serveurs,
- Un ou plusieurs clients.

Les serveurs fournissent des tâches à réaliser ou des données à traiter aux clients. Ils servent alors d'ordonnanceurs afin d'organiser le traitement et recomposent les résultats. L'agrégation des retours des clients permet la création d'un résultat final. Les clients, proposent leur puissance de calcul ou de stockage à la grille afin de créer une sorte de supercalculateur. En général, on retrouve des environnements hétérogènes : Ainsi, et contrairement au calcul parallélisé sur un cluster, les clients peuvent se retrouver avec des systèmes d'exploitation différents. Ils peuvent être aussi physiquement différents (Serveurs, clusters, PDA, calculatrices, bref, tout ce

qui a une puce de calcul et qui peut accéder à un réseau). Le principe du grid computing consiste en sa capacité à gérer des machines qui ne se trouvent pas dans un même lieu. Ces derniers peuvent très bien se trouver à des endroits différents, avec des connexions au réseau complètement différentes (par Internet, réseau local, VPN...).

2.5.2 Un intergiciel pour la grille

Pour construire une grille, on utilise un ensemble de logiciels appelé intergiciel, middleware en anglais. In intergiciel est un logiciel servant d'intermédiaire de communication entre plusieurs applications, généralement complexes ou distribuées sur un réseau informatique, il unifie l'accès à des ressources informatiques hétérogènes et se place entre les systèmes d'exploitation existants (pas de perturbation de ces systèmes) et l'utilisateur. Il masque à ce dernier l'hétérogénéité des divers systèmes installés et fournit un ensemble de routines (commandes et bibliothèque de programmation) indépendantes du système. Ces commandes et fonctions permettent par exemple d'exécuter une application sur une machine avec un langage de description de ressources (quantité de mémoire nécessaire, temps CPU,. . .) indépendant de la machine utilisée.

2.5.3 Quelque intergiciels

Parmi les exemples les plus connus, il convient de citer Globus Toolkit [Globus.T], gLite [Fed03], Unicore [Unicore]. Ces intergiciels libres, ne fonctionnant que dans le monde Unix, ont été développés autour de Java et XML. Ils offrent des interfaces permettant de dialoguer avec les ordonnanceurs locaux des divers clusters de la grille et l'authentification des clients et des nuds de la grille s'effectue via des certificats SSL.

L'intergiciel **Globus Toolkit** est développé par la **Globus Alliance**[Globus.A] et supporté par un consortium comprenant notamment IBM. Globus Toolkit comprend un ensemble d'outils plus ou moins indépendants les uns des autres. Lors du déploiement d'une grille basée sur cet intergiciel, il est possible de n'utiliser que certains modules. On y trouve en particulier, le module GSI (Grid Security Infrastructure) pour gérer la sécurité, le module GRAM (Grid Ressource Allocation and Management) pour gérer l'allocation et la supervision des tâches sur les nœuds de la grille, le module MDS (Monitoring and Discovery Service) pour répertorier en temps réel les nœuds de la grille disponibles et GridFTP pour transférer des données d'un site à un autre. Notons également que le client ne doit s'authentifier qu'une seule fois à une porte d'entrée d'une grille Globus (single sign on) ; par la suite, il sera connu de tous les autres noeuds via un proxy chargé de la délégation de l'authentification.

L'intergiciel **Unicore** (version 1.5) est le concurrent européen de Globus Toolkit. Il est supporté par un consortium d'industriels et par le gouvernement allemand. Il est entièrement écrit en Java et chaque module prend la forme d'un plugin. Le client de la grille Unicore peut charger les plugins souhaités à l'aide d'une interface graphique permettant de soumettre des tâches et de les suivre. Il est un peu moins complet que Globus, mais son installation s'avère plus aisée.

L'intergiciel gLite : L'intergiciel **gLite** est un intergiciel orienté "Web Services" développé dans le cadre du projet EGEE (Enabling Grids for E-sciencE)[EGEE]. Un des objectifs essentiels de gLite est l'interopérabilitée. Pour ce faire, l'intergiciel utilise un maximum de protocoles et interfaces standardisées ou en passe de l'être. GLite se veut le plus modulaire et souple possible. La granularité des services et sous-services est telle qu'on peut aisément choisir ceux que l'on désire faire cohabiter sur une même machine et ceux qui seront déployés sur des machines distinctes. Il a également pour objectif majeur d'ouvrir les portes du grid computing à la majorité des domaines de recherche scientifique. Pour ce faire, gLite développe de nouveaux

services et améliore les existants afin de pouvoir satisfaire toutes les exigences liées à ces différentes disciplines.

2.5.4 L'organisation Virtuelle

Une organisation virtuelle ou Virtual Organisation (VO) est un groupe dynamique d'entités qui décident de partager les ressources et de définir les conditions et les rôles de partage de celles-ci.
Craig Fellenstein et Josshy Joseph [FJ] définissent une organisation virtuelle comme étant :

- des entités logiques,
- limitées dans le temps,
- créees dynamiquement dans le but de résoudre un problème spécifique,
- en fournissant et en allouant des ressources à la demande.

L'organisation virtuelle constitue l'essence même du *grid computing* et représente un élément clé de celle-ci. Elle permet de définir de façon précise *qui* fait *quoi* à *quel moment* et avec *quelles ressources*.
L'organisation virtuelle prend en charge les aspects relatifs à la sécurité. Elle définit les conditions d'accès et la plitique d'utilisation des ressources disponibles sur la grille telles que les cycles CPU, les capacités de stockage, les logiciels accessibles, les périphériques, etc. Il peut être intéressant pour différents centres de recherche de réunir leurs ressources au sein d'une même entité virtuelle. Ceci permet d'augmenter le potentiel global du parc informatique et d'utiliser les ressources d'une manière plus rationnelle.
Les organisations virtuelles peuvent également être constituées juste le temps nécessaire à la résolution d'un problème spécifique. Un chercheur ne disposant d'aucune ressource hardware et ayant besoin d'une puissance de calcul importante peut contacter un fournisseur de services en grille et constituer une VO le temps nécessaire à la résolution de son problème. Cette solution évite au équipe de recherche d'investir

dans une infrastructure informatique lourde et coûteuse.

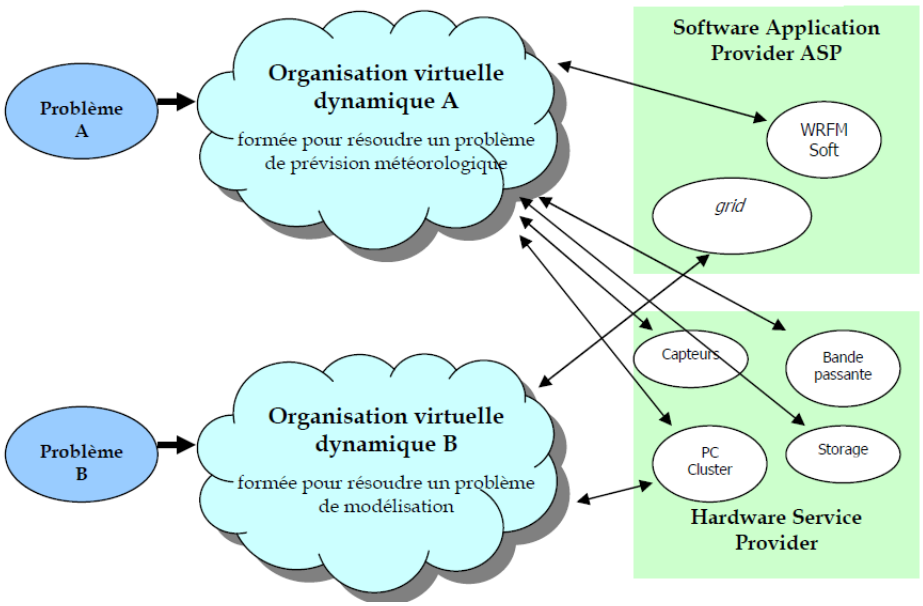

FIGURE 2.2 – Les entités virtuelles d'une grille de calcul

Dans l'exemple ci-dessus, deux organisations virtuelles ont été créées afin de répondre à deux besoins spécifiques ; la prévision météorologique d'une part et la modélisation de séries temporelles d'autre part. Chacune de ces organisations virtuelles utilise des ressources software et hardware durant la période de temps nécessaire à la résolution du problème.

Imagineons que pour résoudre le problème de prévision météorologique, nous ayons besoin du software spécialisé Weather Research and Forecasting Model, de différents capteurs aériens, d'une puissance informatique équivalente à 200.000 curs répartis sur 200 PC équipés de processeurs Intel, d'une capacité de stockage de 200 tera byte et d'une bande passante de 10 Gbits entre les noeuds afin d'assurer un transfert rapide de l'information.

Le second problème de modélisation aura besoin quant à lui du logiciel spécialisé Time Series Expert Grid pour la modélisation des séries temporelles et d'une puissance de calcul équivalente à 100.000 cœurs répartis sur 50 machines fonctionnant sur Unix.

Grâce aux organisations virtuelles, les utilisateurs se partagent les ressources communes. Lorsque les problèmes auront été résolus, les organisations virtuelles seront dissolues et les ressources seront à nouveau libérées pour d'autre utilisateurs.

2.6 Fonctionnement d'une grille

2.6.1 Les composants d'une grille de calcul basée sur gLite

Une Grille de calcul est composée des éléments principaux suivants :

2.6.1.1 Computing Element "CE"

Un Computing Element (CE) représente une ressource de calcul du point de vue de la grille ; plus exactement une ressource de calcul pour chaque file(s) de jobs définie(s) sur le CE. Typiquement, on crée différentes files selon les profils des jobs qu'elles vont accueillir. Par exemple, une file pour les jobs ne demandant pas plus de 10 heures d'utilisation processeur et une deuxième file pour les autres. Cela permet l'utilisation de techniques de type (tourniquet à étage) pour l'ordonnancement des jobs sur la ferme si nécessaire. On peut également décider de créer une file par organisation virtuelle pouvant utiliser le CE.

2.6.1.2 Worker Node "WN"

Les Worker Nodes (WN) sont les différents nœuds qui composent la ferme. Ce sont eux qui exécutent les jobs qui leur ont été transmis par leur Computing Element. Le client du système de gestion de jobs utilisé sur la ferme est donc installé sur ces machines. Notons que les WN ne doivent pas nécessairement être accessibles de l'extérieur du site, seul le CE doit l'être car c'est lui qui reçoit les jobs de la grille.

2.6.1.3 Storage Element "SE"

De même que le Computing Element fournit un accès unifié aux ressources de calcul, le Storage Element (SE) fournit un accès unifié aux ressources de stockage. Un SE peut s'interfacer avec plusieurs types de stockage tels que :

- Simples disques installés sur la machine faisant office de SE.
- Des systèmes de stockage de masse ; typiquement ils se présentent sous forme d'une batterie de disques au-dessus d'une collection de bandes magnétiques. Les migrations de fichiers entre les disques et les bandes sont effectuées par un logiciel spécialisé pour gérer ce type d'infrastructure.

2.6.1.4 Le LCG File Catalog "LFC"

Le LFC est une base de données qui contient les informations sur la localisation des fichiers sur la grille.

2.6.1.5 Le User Interface "UI"

Afin de pouvoir accéder aux ressources de la grille, l'utilisateur doit se connecter à une Interface Utilisateur(UI). Chaque site dispose généralement de son UI sur laquelle les personnes du site autorisées à utiliser la grille possèdent un compte (un simple accès SSH suffit). Chaque utilisateur a pris soin de copier son certificat d'authentification dans son répertoire personnel. L'utilisateur peut alors accéder aux ressources de la grille en utilisant les programmes installés sur l'interface utilisateur. Parmi les tâches qu'il peut effectuer à partir de cette UI citons :

- S'identifier afin de créer un proxy et pouvoir utiliser les ressources de la grille ;
- soumettre un job pour exécution ;
- répertorier toutes les ressources satisfaisant une série de contraintes ;
- gérer ses jobs en cours d'exécution : consulter leur état, les annuler ... ;
- récupérer les résultats de jobs terminés ;
- récupérer des copies de données disponibles sur la grille.

2.6.1.6 Berkeley Database Information Index "BDII"

C'est le responsable de la collecte de ces informations parmi les ressources dont il a la charge. Au sommet de la hiérarchie de la grille se trouve un autre serveur BDII chargé de récupérer régulièrement les informations stockées dans les différents BDII locaux.

2.6.1.7 Workload Management System "WMS"

Les jobs soumis par les utilisateurs (à l'aide de leur UI) arrivent au Workload Management System (WMS) qui a pour mission de déterminer le CE qui sera en charge de les accueillir pour exécution. Cette sélection se base sur plusieurs critères tels que les contraintes définies par l'utilisateur et l'état des différentes ressources au moment de la soumission du job. Pour ce faire, il récupère des informations sur les différentes ressources via le BDII global. Les services du WMS sont découpés en plusieurs grands composants :

- Le Ressource Broker (RB) en charge de la sélection des ressources pouvant satisfaire le job.
- Le Network Server (NS) qui reçoit les jobs des interfaces utilisateurs.
- Le Job Control Service (JCS) qui s'occupe de la soumission du job sur le CE sélectionné ; c'est aussi lui qui gérera l'annulation du job à la demande de l'utilisateur. Ces job sont décrit par le langage JDL (Job Description Language) qui précise, par exemple, l'exécutable à exécuter et ses paramètres, les fichiers à déplacer vers ou du WN où le job s'exécute, et toutes les exigences sur le CE et les WNs.
- Le proxy d'authentification d'un utilisateur a une durée de vie limitée. Or il peut arriver qu'un job nécessite tellement de calcul que le proxy expire avant la fin de son exécution. Dès lors, le job se verra refuser l'accès aux ressources car il ne sera plus identifié. Pour éviter cela, un composant du WMS, le Proxy Server (PS) également connu sous le nom de MyProxy va gérer automatiquement

l'initialisation et le renouvellement du proxy.

2.6.2 Schéma de prise en charge d'un job

Voici sous forme schématique, le cheminement d'un job exécuté sur une grille de calcul :

1. Aprés avoir obtenu un certificat depuis une autorité de certification, être enregistré dans une organisation virtuelle et avoir obtenu un certificat utilisateur, l'utilisateur est prés à utiliser la grille. L'utilisateur se loggue alors sur le "UI" depuis le quel il sera en mesure de lancer ses exécutions sur la grille.
2. L'utilisateur soumet le job au WMS via l'interface (UI).

   Le WMS recherche le CE pouvant prendre en charge l'exécution du job en consultant l'information System.

   L'utilisateur transmet ses fichiers d'entrée dans l'Input SandBox.
3. Le job ainsi que l'Input SandBox sont transférés au CE qui prend en charge le job dans la queue.
4. Le CE envoie le job sur un ou plusieurs WN disponibles.
5. Lorsque le job est terminé, les fichiers produits par celui-ci sont disponibles sur le LRMS (Local Resource Management System). Le WMS est averti que le job s'est terminé.
6. Le WMS récupère les fichiers de sortie dans l'OuputSandBox.
7. L'utilisateur peut interroger à tout moment l'état de son job par l'intermédiaire du Logging and Bookkeeping service (LB)

   Le LB conserve une trace de l'exécution des jobs.

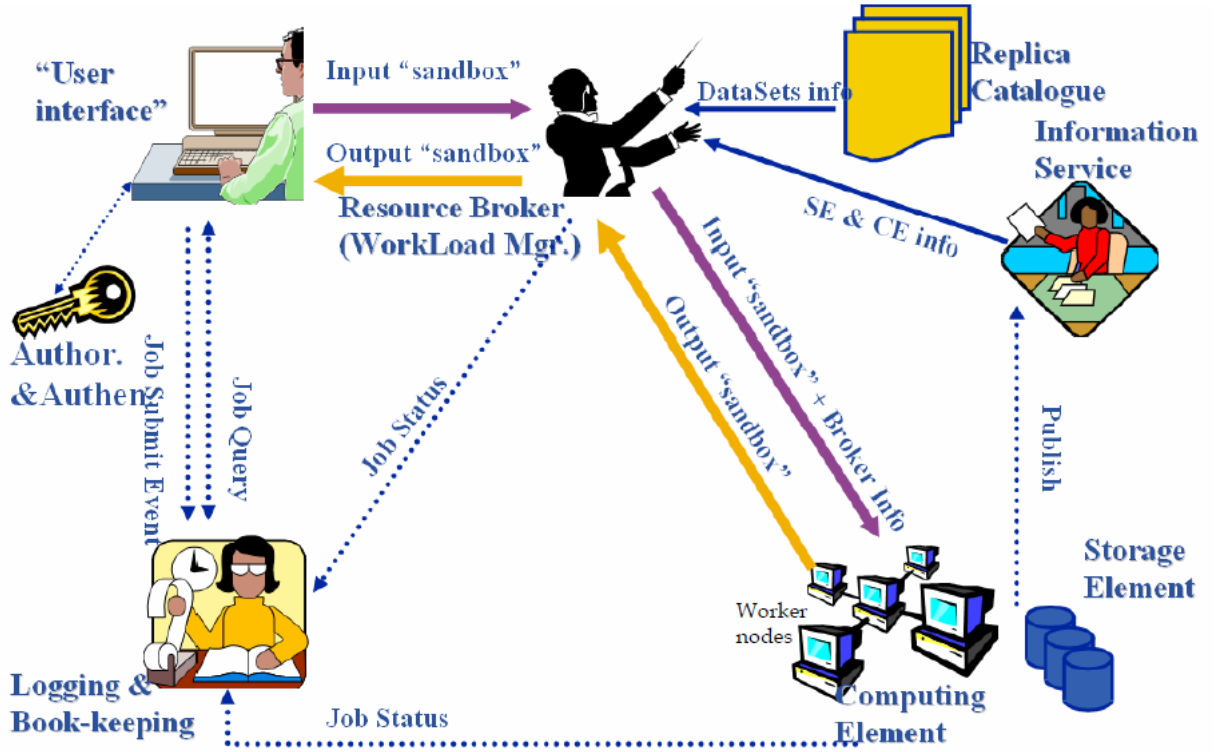

FIGURE 2.3 – Schéma d'exécution d'un job sur la grille- EGEE

2.6.3 Le langage JDL

Un job (fichier de description d'un job) est construit avec une séquence d'attributs dans un langage appelé JDL (Job Descrition Language).

Il y a deux grandes catégories d'attributs :

- Ceux concernant le job : définissent le job lui-même ;
- ceux concernant les ressources.

1. Ils sont utilisés par le RB pour déterminer les ressources nécessaires au job.
2. Ils permettent de préciser les caractéristiques de calcul requises (librairies disponibles, etc). Ces attributs sont définis à l'aide du préfixe 'other'.
3. Ils permettent de définir les caractéristiques liées aux données : données entrantes, SE utilisé pour les données, les protocoles d'accès aux

données,. . .

2.6.3.1 Attributs de job

- **Type** :Type des requêtes soumis. Des valeurs disponibles contiennent
  - Job : Une tache normale
  - DAG : graphe acyclique direct de jobs dépendants
  - Collection : Collections de jobs
- **JobType** :Type de jobs soumis. Des valeurs disponibles sont :
  - Normal
  - Interactive
  - MPICH
  - Parametric
- **Executable** (obligatoire) : Le nom de la commande à exécuter.
- **Arguments** (option) : Les arguments de la commande à exécuter.
- **StdInput**, **StdOutput**, **StdErr** (option) : La définition des entrées/sorties/erreurs standards.
- **Environment** (option) : Ensemble de valeurs lié à l'environnement d'exécution.
- **InputSandbox** (option) :Liste des fichiers se trouvant sur le UI et qui seront transférés avec le job. Ces fichiers seront copiés sur le CE cible.
- **OutputSandbox** (option) : Liste des fichiers générés par le job qui seront transférés sur le UI.

2.6.3.2 Attributs de ressources

- **Requirements** : Sont les besoins du job vis à vis des ressources de calcul. Ils sont spécifiés à partir des attributs qui sont définis dans le système d'information de la grille. S'ils ne sont pas définis dans le JDL, ce sont les valeurs définies par défaut dans le UI qui sont utilisées .

– **Rank** : C'est une préférence, concernant l'ordre de classement des ressources qui remplissent les « Requirements » S'il n'est pas spécifié, la valeur définie sur le UI sera utilisée. La valeur par défaut est (other.GlueCEStateFreeCPUs), i.e., le plus grand nombre de CPU libres.

2.6.3.3 Attributs de données

– **InputSandboxBaseURI** : Déterminer la location des ressources par le protocole gridFTP
– OutputSandboxDestURI : Détermine la place précise des résultats. Si on utilise ce paramètre, on ne peut pas obtenir par la commande glite-wms-job-output

2.6.4 Soumission d'un job

Afin de concrétiser toutes ces informations par un exemple simple, voici quelques copies d'écran correspondant à une soumission classique d'un job sur la grille de calcul. Le programme transmit à la grille est la méthode Branch and Bound implementée sous Scilab [Scilab].

– Création d'un certificat proxy d'une validité de 12 heures :

```
[amina@ui01 ~]$ voms-proxy-init --voms eumed
```

Sortie

```
Your identity: /C=IT/O=INFN/OU=Personal Certificate/L=DZ-eScience/CN=Khedimi Ami
na
Creating temporary proxy ..................... Done
Contacting  voms-02.pd.infn.it:15016 [/C=IT/O=INFN/OU=Host/L=Padova/CN=voms-02.p
d.infn.it] "eumed" Done
Creating proxy ..............................................................
Done
Your proxy is valid until Wed Apr 20 22:15:16 2011
```

– Contenu du fichier .jdl (job description language) :

```
Type = "Job";
JobType = "Normal";
Executable = "silp.sh";
Arguments = "silp.dem";
StdOutput = "scilab.out";
StdError = "scilab.err";
InputSandbox = {"silp.sh","silp.dem"};
OutputSandbox = {"scilab.err","scilab.out","sol.txt","obj.txt","time.txt"};
Requirements = Member("VO-eumed-SCILAB-5.2.2",other.GlueHostApplicationSoftwareRunTimeEnvironment)
```

– Soumission du job à la grille :

```
[amina@ui01 SILP]$ glite-wms-job-submit -a -r  ce01.grid.arn.dz:8443/cream-pbs-eumed  silp.jdl
```

Sortie :

```
==================== glite-wms-job-submit Success ====================

The job has been successfully submitted to the WMProxy
Your job identifier is:

https://prod-lb-01.pd.infn.it:9000/MJfJspTisgu_tlx0A6lwCw

======================================================================
```

– Interrogation sur l'état du job :

```
[amina@ui01 SILP]$ glite-wms-job-status https://prod-lb-01.pd.infn.it:9000/MJfJspTisgu_tlx0A6lwCw
```

Sortie(état du job= running)

```
==================== glite-wms-job-status Success ====================
BOOKKEEPING INFORMATION:

Status info for the Job : https://prod-lb-01.pd.infn.it:9000/MJfJspTisgu_tlx0A6lwCw
Current Status:     Running
Status Reason:      unavailable
Destination:        ce01.grid.arn.dz:8443/cream-pbs-eumed
Submitted:          Wed Apr 20 14:52:12 2011 CET
======================================================================
```

– Nouvelle interrogation sur l'état du job :

```
[amina@ui01 SILP]$ glite-wms-job-status https://prod-lb-01.pd.infn.it:9000/MJfJspTisgu_tlx0A6lwCw
```

Sortie(état du job done - success) :

```
====================== glite-wms-job-status Success =====================
BOOKKEEPING INFORMATION:

Status info for the Job : https://prod-lb-01.pd.infn.it:9000/MJfJspTisgu_tlx0A6lwCw
Current Status:     Done (Success)
Logged Reason(s):
    - job completed
    - Job Terminated Successfully
Exit code:          0
Status Reason:      Job Terminated Successfully
Destination:        ce01.grid.arn.dz:8443/cream-pbs-eumed
Submitted:          Wed Apr 20 14:52:12 2011 CET
==========================================================================
```

– Récupération des données en sortie :

```
[amina@ui01 SILP]$ glite-wms-job-output  https://prod-lb-01.pd.infn.it:9000/MJfJspTisgu_tlx0A6lwCw
```

Sortie

```
==========================================================================

                        JOB GET OUTPUT OUTCOME

Output sandbox files for the job:
https://prod-lb-01.pd.infn.it:9000/MJfJspTisgu_tlx0A6lwCw
have been successfully retrieved and stored in the directory:
/tmp/jobOutput/amina_MJfJspTisgu_tlx0A6lwCw

==========================================================================
```

– Contenu du fichier output :

```
[amina@ui01 SILP]$ cd /tmp/jobOutput/amina_MJfJspTisgu_tlx0A6lwCw
[amina@ui01 amina_MJfJspTisgu_tlx0A6lwCw]$ ls
obj.txt  scilab.err  scilab.out  sol.txt  time.txt
[amina@ui01 amina_MJfJspTisgu_tlx0A6lwCw]$ cat sol.txt
0.000000
2.000000
[amina@ui01 amina_MJfJspTisgu_tlx0A6lwCw]$ cat time.txt
0.010000
```

2.6.5 Accès aux fichiers

Une fois les fichiers localisés sur un SE à l'aide du catalogue de fichiers, l'utilisateur ou les applications exécutées sur la grille doivent pouvoir y accéder. Rappelons que les accès aux fichiers se font selon le principe du (write once, read man) ce qui implique qu'une fois écrit, les fichiers ne sont plus modifiés. On distingue deux grands types d'opérations sur les fichiers

– le transfert de fichiers : d'un SE vers un WN afin qu'il puisse les traiter, d'un SE vers un autre SE pour créer un répliqua, la suppression d'un répliqua...

– l'accès distant : typiquement une application voulant accéder en lecture à un fichier, sans forcement vouloir le rapatrier sur sa machine. Il faut aussi permettre aux applications de créer de nouveaux fichiers sur les SE.

2.6.6 Les différents états d'un job soumis à la grille

Un job soumis à la grille peut prendre les états suivant :

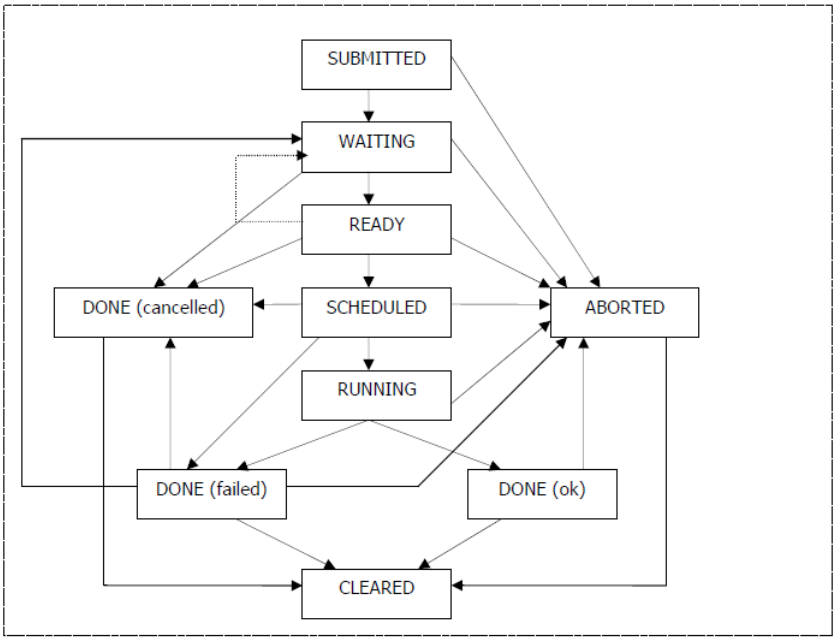

FIGURE 2.4 – Les état d'un job sur la grille- EGEE

SUBMITTED : le job été soumis pour exécution par l'utilisateur mais il n'a pas encore été traité par le serveur

WAITING : le job a été accepté par le serveur réseau mais n'a pas encore été pris en charge par le WMS.

READY : le job a été assigné à un CE mais il n'a pas encore été transmis à celui-ci.

SCHEDULED : le job est en liste d'attente sur le CE.

RUNNING : le job est en cours d'exécution sur le WN.

DONE : le job est terminé.

ABORTED : le job a été arrêté par le WMS (la durée d'exécution du job était trop longue, le certificat proxy est expiré, etc).

CANCELED : le job a été arrêté par l'utilisateur.

CLEARED : l'Output Sandbox a été transféré vers l'interface utilisateur.

2.6.7 Les types de Job

2.6.7.1 Le job Collection

Une des fonctionnalité les plus utilisée sur la grille est la possibilité de soumettre des collections de jobs, définies comme un ensemble indépendants de jobs.

2.6.7.2 Le job DAG

Le Directed Acyclic Graph (DAG) est utilisé pour soumettre des job interdépendants, c'est-à-dire des jobs pour lesquels certains fichiers d'entrée proviennent de l'exécution d'autre jobs. Il doit être possible de représenter ce processus à l'aide d'un graphe orienté acyclique, où les sommets représentent les jobs et les arcs les dépendances.

2.6.7.3 Le job MPI

Le Message Passing Interface (MPI) est utilisé habituellement pour assurer la communication entre les tâches d'applications utilisant la parallélisation. Il existe deux versions de MPI, MPI-1 et MPI-2. La grille supporte deux implémentations de MPI-1 (LAM et MPICH) et deux implémentations de MPI-2 (OpenMPI et MPICH2). Un site grille peut choisir de supporter seulement un sous ensemble de ces implémentations, voir même aucune...

2.6.7.4 Le job Paramétrique

La réalisation d'études paramétriques est particulièrement adaptée à l'architecture distribuée de la grille. Il existe une classe de jobs spéciale pour lancer un job paramétrique extrêmement simplement. Exécuter un job paramétrique consiste à exécuter N jobs différents seulement par la valeur d'un paramètre PARAM dont on peut préciser les valeurs.

2.7 Les Avantages et les Inconvénients des Grilles de Calcul

2.7.1 Les Avantages

Les avantages d'utiliser une telle architecture sont multiples et indéniables. Nous pouvons citer les exemples suivants :

- Peut résoudre des problèmes plus large et plus complexes dans un temps plus court avec un meilleur usage du matériel existant.
- Cette technologie permet de collaborer avec d'autres organisations plus facilement.
- Les jobs peuvent être exécutés en utilisant les performances parallèles. Les environnements de grille sont extrêmement bien adaptés à exécuter les tâches qui peuvent être divisés en petits morceaux et s'exécuter simultanément sur plusieurs nœuds. En utilisant des choses comme MPI permettra le passage de messages à se produire parmi les ressources de calcul.
- Environnements de grille sont beaucoup plus modulaire et n'ont pas de points de défaillance uniques. Si l'un des serveurs ou postes de travail au sein de la grille échouent il y a beaucoup d'autres ressources en mesure de prendre la charge. Les jobs peuvent redémarrer automatiquement si une panne survient.
- Meilleure rentabilisation du matériel, il est évident qu'il y a une sous utilisation

des machines, et le grid computing présente la solution idéale, d'un point de vue économique pour les laboratoires de recherche et d'un point de vue pratique pour les utilisateurs, pour rentabiliser les ressources.

2.7.2 Les Inconvénients

Derrière cette idée de mutualiser les temps processeurs inutilisés pour exécuter des calculs distribués se cachent tout de même un certain nombre d'inconvénients et de contraintes :

- Les applications doivent être qualifiées et qualifiables pour tourner sur une telle architecture.
- Ce genre de système distribué possède une faible tolérance aux pannes.
- Un changement possible des méthodes de travail pour le lancement des calculs rend réticent les futurs utilisateurs.
- Cette technologie étant très récente, ces logiciels et ces standards sont encore en évolution.

Ces différentes faiblesses sont essentiellement dues la cohabitation de plusieurs standards incompatibles s'appuyant sur des langages et des protocoles différents.

2.8 Conclusion

Les technologies du Grid Computing sont les technologies de base de la future génération de l'Internet. D'un point de vue économique, l'avenir du grid computing est largement tracé. Mais seule l'évolution des technologies réseaux et systèmes en a permit son expansion.

On pourrait donc imaginer un avenir axé sur tout "le réseau", avec des utilisateurs n'ayant plus d'ordinateurs mais seulement un terminal graphique doté d'un minimum de puissance. Il sera ainsi possible de louer de la puissance CPU, de louer des espaces disques, d'augmenter sa mémoire vive en temps réel d'un simple clic.

CHAPITRE 3

Le "Flow-Shop" de permutation Bi-Objectif

Le flow-Shop est un problème classique d'ordonnancement qui a beaucoup d'applications en industrie [Kon09]. Les méthodes de résolution varient entre les méthodes exactes, les recherches heuristiques et les métaheuristiques, et la majorité des travaux traitent le problème dans sa forme mono-objectif avec pour objectif principal la minimisation du temps de fin des tâches(makespan). Pourtant l'intérêt de considérer plusieurs objectifs n'est plus à démontrer et pour avoir une vue générale des queleques travaux existants en ordonnancement multi-objectif, le lecteur peut se référencer au livre de T'kindt et Billaut[TB02]. Dans ce chapitre nous nous intéressons à une version bi-objectif du problème de flow-shop. Nous présentons dans un premier lieu les différents problèmes d'ordonnancement existants dans la littérature. Nous verrons ensuite de manière plus approfondie le problème de flow-shop bi-objectif et nous nous intéressons à la modélisation mathématique de ce problème ainsi qu'à sa résolution.

3.1 Introduction

Les problèmes d'ordonnancement sont présents dans tous le secteurs d'activités de l'économie depuis l'informatique jusqu'à l'industrie manufacturière. C'est pour cette raison qu'ils ont fait et continuent de faire l'objet de nombreux travaux de recherche [CCLL97].

Résoudre un problème d'ordonnancement consiste à ordonnancer, i.e., programmer ou planifier dans le temps l'exécution des tâches en leur attribuant les ressources nécessaires matérielles ou humaines, de manière à optimiser un ou plusieurs critères préalablement définis, tout en respectant les contraintes de réalisation [Got93][LR01].

Dans les problèmes d'ordonnancement d'atelier, les ressources sont généralement des machines qui ne peuvent réaliser qu'une tâche ou opération à la fois, i.e., disjonctive, et chaque travail à ordonnancer concerne un produit ou un lot de produit à fabriquer en respectant une gamme de fabrication. Ces problèmes sont pour la plupart très complexe, ce qui explique le grand intérêt porté par la recherche opérationnelle et l'optimisation sur ce type de problème. Par ailleurs, les applications industrielles sous-jacentes sont de nature Multi-Objectif (minimiser le temps de fin d'ordonnancement, minimiser le retard des tâches, minimiser le temps total passé sur l'atelier...). Généralement, ces objectifs sont de nature conflictuels et ne peuvent être atteints simultanément.

Nous avons estimé nécessaire de commencer par présenter les définitions et les concepts principaux liés à l'environnement de notre travail. Puis nous décrivons en détail le problème considéré dans ce mémoire.

3.2 Généralités sur les problèmes d'ordonnancement d'ateliers

Un problème d'ordonnancement consiste à "programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leurs dates d'exécution". Derrière cette définition générale se cache une multitude de problèmes, dont les modèles, les problématiques et les approches de résolution peuvent varier considérablement.

3.2.1 Définition

Il existe de nombreuses définitions des problèmes d'ordonnancement. Pinedo propose cette définition : " *L'ordonnancement concerne l'affectation de ressources limitées aux tâches dans le temps. C'est un processus de prise de décision dont le but est d'optimiser un ou plusieurs objectifs*" [Pin95]. Nous pouvons donc dire qu'un problème d'ordonnancement peut être vu de la manière suivante : connaissant les tâches à exécuter et les machines pouvant exécuter ces tâches, il convient de déterminer un ordre de passage des tâches sur les différentes machines et la date du début de chacune de ces tâches sur ces machines. Les tâches sont généralement désignées sous le nom d'opérations et regroupées en travaux (ou jobs).

3.2.2 Domaines d'application

Les problèmes d'ordonnancement sont rencontrés à tous les niveaux et dans tous les secteurs d'activité. En générale, nous pouvons distinguer entre ceux de la production manufacturière et ceux des systèmes informatiques ou de gestion de projet.

Nous rencontrons les problèmes d'ordonnancement dans les systèmes de fabrication flexible(FMS). Un FMS comprend trois éléments principaux : les machines de travail, un système de transport automatisé et un ordinateur de contrôle central

qui contrôle les mouvements et les flux des matières sur les machines. Ces problèmes sont largement couverts dans la littérature et le plus souvent dans une classe d'application bien définie. En outre, ils touchent d'autre problèmes liés à la planification des cellules robotisées et des Automated Guided Vehicles (AVG) [TB02][LM96].

Dans les lignes de production automobile, les problèmes d'ordonnancement sont rencontrées dans les magasins de rassemblement où certains équipements (ou options) doivent être assemblés dans les différents modèles de véhicules. Ces problèmes ont des contraintes et des particularités qui leurs sont propres. Connaissant la séquence de véhicules en cours de traitement, le problème est de déterminer le type du prochain véhicule programmé. Nous devons tenir compte d'un groupe de contraintes liés principalement aux options de montage de ces véhicules et au mouvement limité des outils de production le long de la ligne de production.

Dans les systèmes informatiques, les problèmes d'ordonnancement sont étudiés sous différentes formes en considérant les système mono ou multi-processeurs, avec la contrainte de la synchronisation des opérations et le partage des ressources. Dans ces problèmes, certaines opérations sont périodiques, d'autres ne le sont pas, certaines font l'objet d'une date due, d'autre d'un délais. L'objectif est souvent de trouver une solution efficace qui satisfait les contraintes [BEP96].

3.2.3 Notations

Il existe deux notations pour référencer les problèmes d'ordonnancement. La plus ancienne a été proposée par Conway et al [CMM67]. Cependant, la notation la plus utilisée a été introduite par Graham et al [GLLR79]. Cette notation comporte trois champs $\alpha/\beta/\gamma$

- Le champ α détermine la structure du problème d'ordonnancement traité.
- Le champ β définit l'ensemble des contraintes.
- Le champ γ indique le ou les objectifs à optimiser.

Nous pouvons retrouver une description très détaillée des différentes valeurs possible pour ces champs dans l'ouvrage "L'ordonnancement Multicritère" [TB02]. Nous ne décrirons donc que brièvement les valeurs possibles pour ces différents champs.

3.2.4 Les différentes structures des problèmes d'ordonnancement

Selon la structure de l'atelier, on distingue généralement les situations suivantes :

- **Atelier à machine unique** ($\alpha = \phi$) : il n'y a qu'une machine en exemplaire unique. Il s'agit d'un atelier de base ou d'un atelier dans lequel une seule machine pose un problème d'ordonnancement (ex : ordonnancer les opérations sur un processeur d'ordinateur).
- **Atelier de type Flow-Shop**($\alpha = \boldsymbol{F}$) : il y a plusieurs machines dans l'atelier. Dans ce type d'atelier les tâches doivent passer sur toutes les machines et utilisent les machines dans le même ordre (ex : chaîne de montage).
- **Atelier de type Job-Shop** ($\alpha = \boldsymbol{J}$) : il y a plusieurs machines dans l'atelier. Chaque tâche à un ordre de passage qui lui est propre sur les différentes machines (ex : montage de composant sur une carte mère d'ordinateur).
- **Atelier de type Open-Shop** ($\alpha = \boldsymbol{O}$) : il y a plusieurs machines dans l'atelier. Les tâches sont réalisées dans un ordre quelquonque sur les différentes machines (ex : coloration de tissu en industrie).
- **Atelier de type mixte** ($\alpha = \boldsymbol{X}$) : il y a plusieurs machines dans l'atelier. Certaines tâches ont un ordre de passage précis sur les machines et d'autres n'en ont pas.
- **Problèmes d'ordonnancement et d'affectation avec étage :** ici, les machines sont regroupés en étages, une machine n'appartient qu'à un étage. Les machines d'un étage sont capables de réaliser les mêmes opérations. A chaque étape, nous pouvons distinguer les configurations suivantes :
  - Les machines sont identiques (P) : les opération ont le même temps d'exé-

cution sur toutes les machines.

- Les machines sont uniformes (Q) : le temps d'exécution d'une opération $O_{i,j}$ sur la machine M_k est égal à $p_i, j, k = q_{i,j}/v_k$ où $q_{i,j}$ est par exemple le nombre de composant à traité dans l'opération $O_{i,j}$ et v_k le nombre de composants pouvant être traités par la machine M_k en unité de temps.
- Les machines sont indépendantes : le temps d'exécution de l'opération $O_{i,j}$ sur la machine M_k est égal à $p_i, j, k = q_{i,j}$, est une donnée du problème.

Globalement, les problèmes d'ordonnancement et d'affectation correspondent au configurations suivantes :

- Les machines parallèles ($\alpha = P/Q/R$) : il n'ont qu'un seul étage, et les jobs sont mono-opération.
- Flow-Shop hybride ($\alpha = FH$) : les gammes sont toutes identiques- comme dans tout flow-shop- et les machines sont groupées par étages.
- Job-Shop généralisé ($\alpha = JG$) : chaque job a sa propre gamme opératoire.
- Open-Shop généralisé ($\alpha = OG$) : les jobs n'ont pas un routage fixe.

- **Problème d'ordonnancement et d'affectation généralisée :** dans ce cas, chaque opération ne peut être exécutée que sur un ensemble de machines qui lui est propre. Il existe plusieurs cas possibles : les problèmes à machines parallèles avec affectation générale ($\alpha = PAG$ *ou* QAG *ou* RAG), les problèmes d'atelier avec affectation générale ($\alpha = SAG$) et les problèmes d'Open-Shop avec affectation générale ($\alpha = OAG$).

3.2.5 Les contraintes

Elles sont représentées dans le champ β de la notation de Graham et al [GLLR79]. Tous les problèmes d'ordonnancement ont un certain nombre de contraintes. Celle-ci peuvent être implicites ou explicites. Par exemple, pour un problème de Flow-Shop, il est implicite qu'une tâche ne peut commencer sur une machine si son exécution n'est pas terminée sur la machine précédente. En revanche, si des dates de disponibilité

des tâches sont rajoutées au problèmes, alors ceci sera une contrainte explicite. Nous reprenons maintenant les contraintes explicites les plus rencontrées (Une liste complète est proposée dans [TB02]) :

- **dates de fin/dates dues :** les tâches doivent se terminer à une date précise (appelée date due ou date de fin souhaitée). Dans le cadre des contraintes, cette date ne peut pas être dépassée (ex : date de péremption d'un produit).
- **date de début/date de disponibilité :** les tâches ne sont disponibles qu'à partir d'une date donnée.
- **permu :** seul les ordonnancements de permutation sont considérés. Si une tâche est à la j^{me}place sur la machine 1 elle sera à cette place sur toutes les machines. Cette contrainte ne s'applique qu'au seul cas du Flow-Shop.
- **pmnt :** la préemption est autorisée. Une opération peut être interrompue en cours d'exécution et reprise par la suite.
- **prec :** les tâches ont des contraintes de précédence.
- **no-wait :** les tâches se succèdent sur les machines sans attente.
- $p_i = p$ **:** les durées des tâches sont toutes identiques.
- $d_i = d$ **:** les dates dues des tâches sont toutes identiques.

3.2.6 Les objectifs

Ils font l'objet du champs γ de la notation de Graham et al [GLLR79]. Afin d'évaluer les ordonnancements, un certain nombre d'objectifs peuvent être optimisé. Nous pouvons considérer deux classes d'objectif : les objectifs de type *minmax* et ceux de type *minsum*. Nous présenterons chacun de ces deux types.

3.2.6.1 Les objectifs de type minmax

Il s'agit ici de la minimisation d'un maximum. Nous proposons, ici, un récapitulatif de ce que représentent les objectifs de type minmax les plus fréquents de la littérature :

- C_{max} **:** date de fin de l'ordonnancement, donc la date de fin maximum d'une tâches. C'est l'objectif le plus fréquemment rencontré. Cet objectif est appelé "Makespan".
- F_{max} **:** temps de présence maximum d'une tâche sur l'atelier.
- L_{max} **:** plus grand retard (valeur positive ou négative).
- T_{max} **:** plus grand retard (valeur positive uniquement).
- E_{max} **:** plus grande avance.

3.2.6.2 Les objectifs de types minsum

Il s'agit de la minimisation d'une somme. Ces objectifs sont généralement plus difficiles à optimiser que les objectifs de type "*minmax*". Voici la liste des objectifs de ce type les plus rencontrés :

- $\overline{C}$ **:** somme des dates de fin des tâches.
- $\overline{C}^{\omega}$ **:** somme pondérée des dates de fin des tâches.
- $\overline{T}$ **:** retard total des tâches.
- $\overline{T}^{\omega}$ **:** retard total pondéré.
- $\overline{U}$ **:** nombre de travaux en retard.
- $\overline{U}^{\omega}$ **:** nombre pondéré de travaux en retard.
- $\overline{E}$ **:** avance moyenne des tâches.
- $\overline{E}^{\omega}$ **:** avance moyenne pondérée.

3.3 Le flowshop de permutation bi-objectifs

Nous avons donné une classification des différents problèmes d'ordonnancement les plus rencontrés dans la littérature, nous allons maintenant présenter le problème étudié dans ce document. Nous commencerons par présenter le problème de Flow-Shop de permutation Bi-Objectif, puis nous proposerons un état de l'art des méthodes de résolution des problèmes de Flow-Shop Multi-objectif. Nous proposerons

ensuite une modélisation mathématique du problème.

3.3.1 Définition du problème

Le problème de Flow-Shop, noté $F/perm, d_i/(C_{max}, \sum w_i T_i)$ [GLLR79], se présente comme un ensemble de n jobs à ordonnancer sur m machines. Le premier champ de la notation désigne le type de problème, dans notre cas il s'agit d'un problème du flow-shop (F). Le deuxième champ désigne les éléments spécifiques à prendre en compte pour le problème étudié. Ici, "perm" indique qu'on étudie un flow-shop de permutation, c-à-d que les tâches doivent être ordonnées dans le même ordre sur toutes les machines. De plus d_i indique que chaque tâche a une date d_i avant laquelle elle doit s'achever. Enfin, le troisième champ désigne les critères à optimiser, deux objectifs seront optimisés. Le premier objectif est la date de fin de l'ordonnancement notée C_{max}, i.e., la date de fin de traitement de la dernière tâche sur la dernière machine. Le deuxième objectif est la somme des retard pondéré de chaque tâche (différence entre la date de complétude de la tâche et sa date de terminaison prévue d_i) notée $\sum w_i T_i$. Chaque tâche doit être exécutée sur chaque machine et chaque machine peut exécuter qu'une seule opération à la fois. Une tâche peut commencer sur la machine i seulement si son opération sur la machine $i-1$ a été effectuée. Les jobs sont indépendants et non préemptifs, i.e, la préemption n'est pas autorisée de telle sorte qu'une fois commencée, une opération ne peut pas être arrêtée et va jusqu'à son terme . Nous nous limitons à l'étude des problèmes de type Flow-Shop en présence du temps de transportation du job entre deux machines et qu'il existe un nombre illimité de transporteur : un système Multi-transporteur [NAJ09][SAZ08]. La figure 3.1 présente un exemple de Flow-Shop de permutation avec 3 tâches et 4 machines.

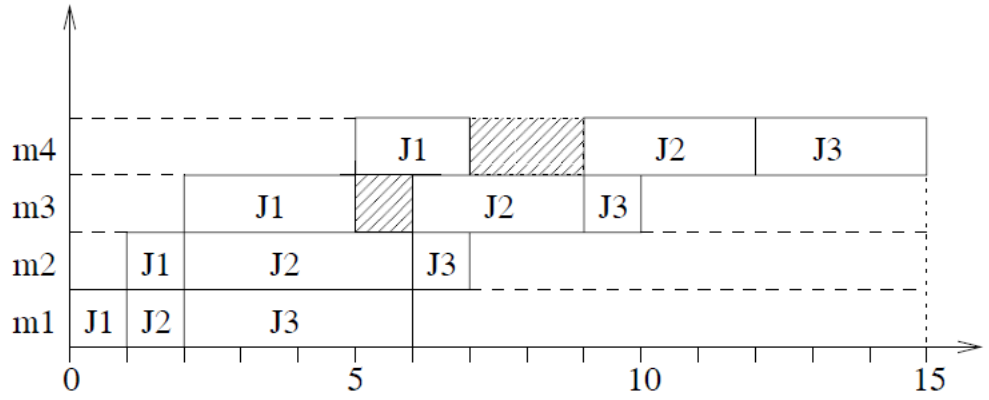

FIGURE 3.1 – Exemple de Flow-Shop de permutation avec 3 tâches et 4 machines

Il s'agit donc d'un problème d'optimisation discrète, puisque l'on suppose habituellement que toutes les variables (en particulier les dates d'exécution) sont des entiers. Face à de tels problèmes, la première question que l'on est amené à se poser concerne la complexité. L'objectif du makespan, pour le flow-shop de permutation est NP-difficile à partir de 3 machines [GJS76] ; cette objectif est très étudié. L'objectif du retard total est déjà NP-difficile pour le cas d'une machine [1] [DL90] ; cet objectif est très peu étudié pour m machines et réputé comme étant très difficile. Il apparaît donc que l'objectif du retard total sera certainement plus difficile à optimiser que l'objectif du makespan [Lem06].

3.3.2 État de l'art des méthodes de résolution du Flow-Shop Multi-Objectif

Les problèmes d'ordonnancement étant, pour la plupart, NP-Difficiles. Les problèmes du Flow-Shop sont divisés en deux ensembles : ceux se limitant à deux machines, et ceux avec un nombre de machines m variable selon l'instance du problème. Néanmoins, la majeur partie des méthodes Multi-Objectif proposées considère des problèmes n'ayant que deux machines. La liste des études citées n'est pas exhaus-

1. Notons qu'il existe un algorithme pseudo-polynomial [Law77] pour le cas à une machine

tive. Dans un premier temps, nous présentons les travaux traitant du makespan et d'un second objectif sur les dates de complétion, puis nous verrons ceux traitant du makespan et d'un objectif de type retard.

Commençons par la résolution de manière lexicographique du problème de la somme des dates de complétion tout en ayant la valeur optimale du makespan pour un problème à deux machines (noté $F_2//Lex(C_{max}, \bar{C})$). Ce problème a été abordé à l'aide d'heuristique [Raj92, GNW01, GHW02, TGB03], d'un algorithme de recherche locale [TMTL02] (notons que cet algorithme coopératif combine une méthode exacte et une heuristique), ou encore de manière exacte à l'aide d'un Branch&Bound [TGB03]. L'agrégation des objectifs du makespan et de la somme des temps de fin de tâches (noté $F_2//\alpha C_{max} + \beta C$) a été résolue grâce à un Branch & Bound et à des heuristiques dans deux travaux [NHH95, SSU98], une résolution par branch&Bound a aussi été proposée par Yeh [Yeh99]. Yeh et Allaheverdi [YA04] proposent un Branch & Bound pour le problème à trois machines considérant l'agrégation des mêmes objectifs. Nous pouvons voir dans [SK99] une résolution retrouvant l'ensemble des solutions Pareto optimales pour le problème (C_{max}, C)) grâce à une application de la méthode des $\epsilon - contraintes$. Dans ce travail, les sous-problèmes sont résolus à l'aide de Branch&Bound. Varadharajan et Rajendran [VR05] proposent un algorithme de type " Multi-objective Simulated-annealing Algorithm" (MOSA) afin de résoudre le problème Bi-Objectif de minimisation du makespan et du flow total pour un nombre quelconque de machines.

Un Branch & Bound ainsi qu'une heuristique ont été présentés dans [DC90] pour le problème $(F_2/d_i/C_{max}, T_{max})$. Dans ce travail, les auteurs proposent aussi une heuristique pour approximer le front Pareto pour le problème $(F_2/permut, d_i/C_{max}, T_{max})$. Liano et al [LYJ97] étudient les problèmes $(F_2//(C_{max}, \bar{U}))$ et $(F_2/d_i/C_{max}, \bar{T})$ et présentent un Branch & Bound pour chacun de ces problèmes. Dans [GHW02], Gupta et al proposent aussi une résolution du problème d'optimisation du makes-

pan en objectif principal avec la somme des retards pondérée ($F_2/d_i/C_{max}, \bar{T^{\omega}}$) grâce à une heuristique. Le problème de l'agrégation du Makespan et du retard maximum ($F/d_i/\alpha C_{max} + \beta T_{max}$ est résolu par Allahverdi [ALL04] grâce à une heuristique. Toktas et al [TAk04] propose un Branch & Bound pour résoudre le problème de minimisation du makespan et de l'avance maximum d'une tâche ($F/permut, d_i/C_{max}, E_{max}$). Arroyo et Armento [AA05] proposent une résolution des problèmes ($F/d_i/C_{max}, T_{max}$) et ($F/d_i/C_{max}, \bar{T}$) grâce à un algorithme génétique, ils se compare aux Branch & Bound de Daniels et Chamvers [DC90] et de Liano et al [LYJ97] pour le cas à deux machines.

Chacune de ces méthodes possède bien entendu ces propres avantages et inconvénients, en termes de simplicité d'implémentation, de complexité temporelle, c'est-à-dire de temps d'exécution, et de qualité des solutions fournies. Notons que tous les travaux proposés sont des problèmes Bi-Objectif. Ceci s'explique par les difficultés liées à l'optimisation des problèmes d'ordonnancement et de l'optimisation Multi-Objectif.

3.3.3 Modélisation mathématique du problème

Comme nous l'avons déjà mentionner dans la section 3.3.1, nous nous intéressons au Flow-Shop de permutation Bi-Objectif avec temps de transport. Ne pas considérer l'hypothèse du temps de transportation pourraient être irréaliste dans le monde des industries. En effet, un job peut être réalisé immédiatement après sa terminaison sur la machine précédente, en d'autre terme, il n'y a pas le temps de transport, peut ne pas être toujours vrai, parce qu'il est nécessaire de mener un job d'une machine à une autre par un transporteur. Par conséquent, nous avons été motivés à étudier le Flow-Shop de permutation avec temps de transport entre les machines.
Après l'étude des modèles mathématiques proposé par [NAJ09] et [SAZ08], nous présentons dans ce qui suit le modèle mathématique de notre problème.

3.3.3.1 Les paramètres

n	Le nombre de Jobs
m	Le nombre de machines
i	Indices de job, $i \in \{1, 2, \ldots, n\}$
j	Indices de position, $j \in \{1, 2, \ldots, n\}$
k	Indice de machine,$i \in \{1, 2, \ldots, m\}$
$O_{i,k}$	L'opération du job i sur la machine k
$P_{i,k}$	Temps d'exécution de $O_{i,k}$
$T_{i,k}$	Le temps de transport du job i de la machine $k-1$ à la machine k
d_i	Date due du job i

3.3.3.2 Les variables

$x_{i,j}$	Variable bivalente vaut 1 si le job i occupe la position j
	0 sinon
$C_{i,j,k}$	Variable entière représente le temps de fin d'exécution du job i
	occupant la position j sur la machine k
D_i	Variable entière représente le retard du job i.

3.3.3.3 Le modèle mathématique

$$
(PL)\begin{cases}
Minimiser \qquad (C_{max}, \sum_{i=1}^{n} w_i D_i) & (1)\\
Sous \quad contraintes :\\
\sum_{j=1}^{n} x_{i,j} = 1 \qquad \forall i = 1, \ldots, n & (2)\\
\sum_{i=1}^{n} x_{i,j} = 1 \qquad \forall j = 1, \ldots, m & (3)\\
\sum_{i=1}^{n} (C_{i,j,k} - C_{i,j,k-1} - (P_{i,k} + T_{i,k}) x_{i,j}) \geq 0 \quad \forall j = 1, \ldots, n, k = 1, ..., m & (4)\\
\sum_{i=1}^{n} (C_{i,j,k} - C_{i,j-1,k} - (P_{i,k} x_{i,j})) \geq 0 \qquad \forall j > 1, k = 1, ..., m & (5)\\
C_{max} \geq \sum_{i=1}^{n} C_{i,n,m} & (6)\\
D_i - \sum_{j=1}^{n} C_{i,j,m} + d_i \geq 0 \qquad \forall i = 1, ..., n & (7)\\
C_{i,j,k} \leq M x_{i,j} \qquad \forall i = 1, ..., n, j = 1, ..., n, k = 1, ..., m & (8)\\
C_{i,j,k} \geq 0, x_{i,j} \in \{0, 1\}, D_i \geq 0, \qquad \forall i = 1, ..., n, k = 1, ..., m, j = 1, ..., n
\end{cases}
$$

Avec $C_{i,j,0} = 0$ et $C_{i,0,k} = 0$

- La fonction (1) exprime directement les objectifs de notre problème, i.e., la minimisation du makespan et la minimisation de la somme pondérée des retards.
- Les contraintes (2) et (3) sont des contraintes d'unicité des tâches sur la machine k et la position j, i.e., un job doit avoir exactement une seule position et les n positions doivent être occupées une et une seule fois.
- La contraintes (4) et (5) concerne la la date du début et de fin de l'opération $O_{i,k}$, i.e., le job ne peut pas commencer avant qu'il soit terminée sur la machine précédente et transféré à la machine k et que son prédécesseur soit terminé sur cette machine.
- Les contraintes (6) et (7) représentent le calcul de la valeur du makespan et le

retard du job j.

– La contrainte(8) lie le couple $(C_{i,j,k}, x_{i,j})$, si $(C_{i,j,k} > 0$, alors $x_{i,j} = 1)$, et M désigne un entier suffisamment grand par rapport aux autres données du problème.

3.3.4 Méthodes de résolutions

Afin de résoudre notre problème, nous avons fait une étude de trois méthodes exactes : la méthode de *Özlen* et *Azizoğlu* [OA09], la méthode des deux phases (TPM) élaborée par Ulungu et Teghem [UT95], et la méthode parallèle par partitions (PPM) proposée par Lemesre et al [LDT05b].

3.3.4.1 Méthode de *Özlen* et *Azizoğlu* [OA09]

Cette méthode est une amélioration de la méthode $\epsilon - contrainte$ (voir section 1.5.2.1, chapitre1) qui consiste à optimiser le problème sur l'un des objectifs en bornant tous les autres objectifs. Dans cette méthode, les valeurs ϵ_i sont diminuées d'une unité dans le cas de minimisation multi-objectif, alors que la méthode de *Özlen* et *Azizoğlu* diminue ces valeurs en tenant compte de la solution non dominée actuelle. En outre, la fonction objectif n'est plus l'un des critères, mais une combinaison pondérée et appropriée des critères du problème multi-objectif qui assure l'efficacité des solutions obtenues.

Dans cette méthode, on montre que le problème multi-objectif est équivalent (au sens des solutions efficaces) au problème suivant, obtenu en transformant le r^{me}

critère en contrainte :

$$(Q_{r-1}) \begin{cases} \min f_1(x) + \epsilon_r f_r(x) \\ \vdots \\ \min f_{r-1}(x) + \epsilon_r f_r(x) \\ f_r(x) \leq l_r \\ x \in S \end{cases} \tag{3.1}$$

où l_r est une borne inférieure de f_r. Ce problème est à son tour équivalent au problème obtenu en transformant le critère $r-1$ en contrainte et ainsi jusqu'à la résolution du programme linéaire en nombres entiers mono-objectif suivant :

$$(Q_1) \begin{cases} \min f_1(x) + \sum_{i=2}^{r} \omega_i f_i(x) \\ f_i(x) \leq l_i, \quad i = 2, ..., r \\ x \in S \end{cases} \tag{3.2}$$

où les poids ω_i sont calculés en fonction de f_i^{ub} borne supérieure et f_i^{lb} borne inférieure de f_i et la solution optimale de (Q_1) sera une solution efficace (ou non-dominée) de problème multi-objectif (Q_r). L'idée de cette méthode est pour générer l'ensemble non-dominé d'un problème k-objectif, on le ramène à un problème (k-1)-objectif en utilisant chaque fois la transformation $\epsilon - contrainte$. Ceci dit que pour résoudre notre problème (Le flow-shop bi-objectif), on le ramène à un problème mono-objectif. La méthode de *Özlen* et *Azizoğ*lu passe par les étapes suivante

1. **Étape 1** : Initialement l'ensemble des solutions non-dominées SND_r est vide. Cette étape consiste à trouver la bornes supérieure f_i^{ub}, et les bornes inférieure f_i^{lb} de tous les objectifs i avec $i = 2...r$, et calcule les valeurs ω_r tel que $\omega_r = \frac{1}{(f_2^{ub} - f_2^{lb})(f_3^{ub} - f_3^{lb})...(f_r^{ub} - f_r^{lb})}$ et l_r tel que $l_r = f_r^{ub}$
2. **Étape 2** : Cette étape consiste à résoudre le problème (Q_{r-1}) avec les $r-1$ premiers objectifs, si ce problème est non réalisable, l'algorithme de la méthode s'arrête sinon l'ensemble des solutions non-dominées du problème (Q_{r-1}) est

noté SND_{r-1}

3. **Étape 3** : Cette étape mit à jour l'ensemble des solutions non dominées du problème (Q_{r-1}), en posant $SND_r := SND_r \cup SND_{r-1}$ et calcule la nouvelle valeur de l_r avec $l_r := max\{f_r(x)|f(x) \in SND_{r-1}\} - 1$ et retourne à l'étape 1.

3.3.4.2 La Méthode des Deux Phases [UT95]

Le front de Pareto est constitué de deux types de solutions : solutions supportées (se trouvent dans l'envellope convexe des solutions) et solutions non-supportées. Basé sur ce principe, Ulungu et Teghem ont proposé une méthode de deux phases (TPM :Two Phase Method) pour résoudre un problème d'affectation Bi-Objectifs [UT95]. Comme son nom l'indique, cette méthode se décompose en deux phases, une première phase où sont recherchées les solutions supportées et une seconde où sont recherchées les solutions non-supportées.

1. La première phase consiste à trouver les solutions supportées avec une agrégation des deux objectifs f_1 et f_2 de la forme $\lambda_1 f_1 + \lambda_2 f_2$. Elle commence par trouver les deux solutions extrêmes qui sont des solutions supportées et qui permettent de borner l'espace de recherche (Voir Figure 3.2.(a)). Ensuite elle cherche l'existence des solutions supportées entre (dans l'espace objectif) deux solutions supportées déjà trouvées r et s (supposons que $f_r \prec f_s$), la direction de la recherche sera perpendiculaire à la ligne (rs), définie comme suit :$\lambda_1 = f_2(r) - f_2(s)$,$\lambda_2 = f_1(s) - f_1(r)$ (voir Figure 3.2.(b)). Avec cette recherche, s'il existe au moins une solution supportée t avec $f(t)$ entre $f(r)$ et $f(s)$ dans l'espace objectif, elle sera trouvée. Si une nouvelle solution est trouvée, elle génère deux nouvelle recherches entre $f(r)$ et $f(t)$ et $f(t)$ et $f(s)$ (voir Figure 3.2.c). A la fin de cette phase, toutes les solutions supportées sont trouvées (voir Figure 3.2.(d)).

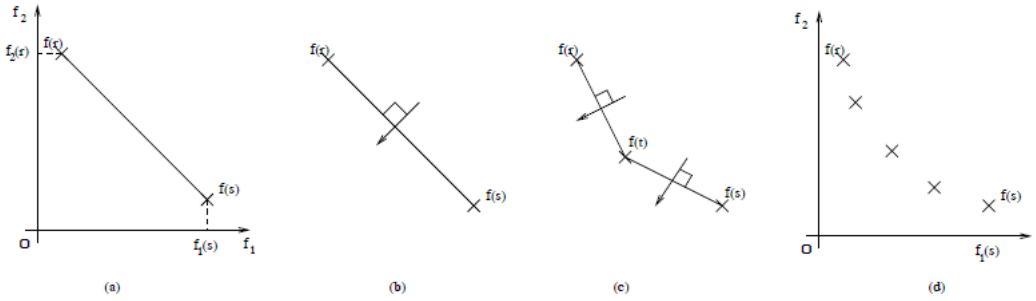

FIGURE 3.2 – La première phase de la méthode TPM

2. La deuxième phase consiste à trouver touts solutions non supportées du front Pareto entre chaque couple de solutions supportées. Ces solutions ne peuvent être trouvées avec une agrégation des objectifs. Ulungu et Teghem proposent alors d'utiliser les solutions supportées trouvées pour réduire l'espace de recherche. Soit une solution non-supportée u avec $f(u)$ entre deux solutions supportées $f(r)$ et $f(s)$ alors $f_1(r) < f_1(u) < f_1(s)$ et $f_2(r) > f_2(u) > f_2(s)$. Graphiquement, les solutions supportées entre $f(r)$ et $f(s)$ se situent forcément dans les triangles rectangles basés sur les solutions $f(r)$ et $f(s)$(voir Figure-3.3.(e))

   Entre chaque paire de solutions supportées, un triangle de recherche est construit. Tous les triangles sont alors explorés afin de trouver toutes les solutions non-supportées (voir Figure-3.3.(f)). A la fin de cette phase, toutes les solutions Pareto sont trouvées (voir Figure 3.3.(g)) [Lem06].

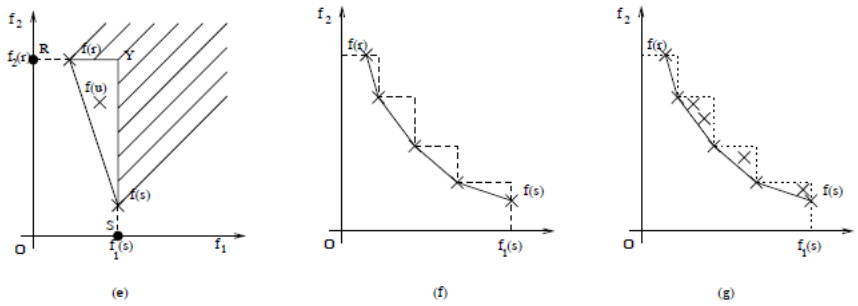

FIGURE 3.3 – La deuxième phase de la méthode TPM

3.3.4.3 Parallel Patitioning Method [LDT05b]

Lemesre et al [LDT05b] ont proposé certaines améliorations à la Méthode des Deux Phases. Cette amélioration les a conduit à proposer une nouvelle méthode exacte bi-objectif : la Méthode Parallèle par Partitions. Cette méthode permet de trouver l'intégralité du front Pareto. Elle est décomposée en trois phases :

1. **Phase 1 :** Recherche des solutions extrêmes (qui sont Pareto optimales) et partitionnement de l'espace de recherche Cette phase consiste à chercher les solutions optimale extrêmes pour chacun des critère. Ces deux solutions nous donnent une borne pertinente de l'espace de recherche car elles nous donnent la plus grande et la plus petite valeur possible, pour chacun des critères, pour toutes les solution Pareto. Ces solutions extrêmes sont recherchées en utilisant un ordre lexicographique sur les objectifs. Par exemple, pour rechercher la solution extrême du front Pareto pour un objectif f_1, cet objectif est tout dabord optimisé (sans tenir compte de l'objectif f_2 , puis en gardant la valeur obtenue pour f_1, le deuxième objectif est optimisé.

   L'espace contenu entre ces deux solutions est ensuite découpé de façon uniforme suivant l'un des objectifs et formant des sous espaces de taille égales.

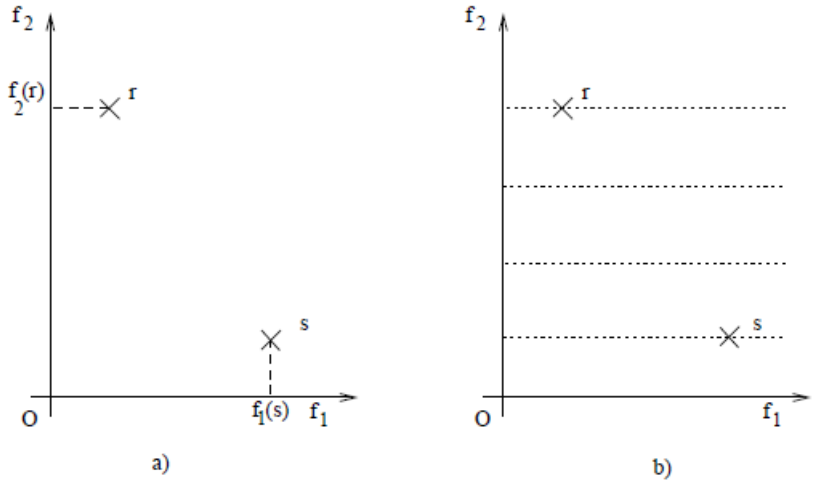

FIGURE 3.4 – La première phase de la méthode PPM

2. **Phase 2 :** Recherche d'une solution par partition

   Dans cette étape, une solution Pareto est calculée pour chaque sous-espace ou partition : celle qui appartient à la partition en question et qui optimise l'objectif non utilisé pour le découpage. Donc un des objectif est borné et le second est optimisé selon le concept d' $\epsilon - contrainte$. Des solutions supportées et non supportées sont trouvées au cours de cette phase, de plus, si les solutions sont bien réparties le long du front Pareto, alors le sous-ensemble de solutions trouvées est aussi bien réparti.

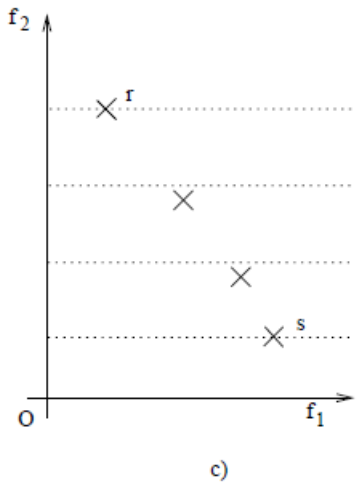

FIGURE 3.5 – La deuxième phase de la méthode PPM

3. **Phase 3 :** Recherche des autres solutions Pareto optimales Cette phase permet de trouver toutes les solutions du front Pareto non trouvées lors des deux premières phases. Cette étape est similaire à la seconde phase de la méthode TPM. Elle consiste à rechercher dans chacun des sous-espaces délimités par deux solutions adjacentes obtenues lors de la phase précédente, ce qui revient à faire une recherche dans les rectangles formés par chaque couple de solution adjacentes et l'origine du repère $SYRO$ où Y est le point $(f_1(s), f_2(r))$, R le point $(0, f_2(r))$ et S le point $(f_1(s), 0)$. En réalité, cette recherche peut se limiter au rectangle $f(r)Yf(s)Z$ où Z est le point $(f_2(s), f_1(r))$, si la méthode

permet de rajouter ces contraintes [Lem06].

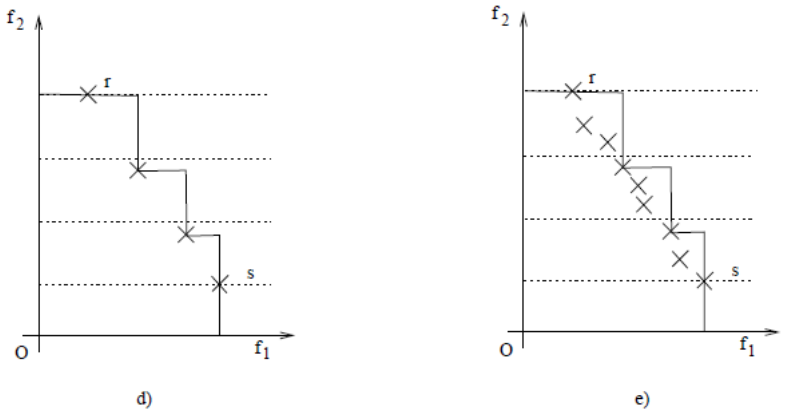

FIGURE 3.6 – La troisième phase de la méthode PPM

3.3.5 Discussion

Nous avons présenté trois méthodes exactes qui présentent un schéma de résolution très intéressant car très général et qui ne dépend pas du problème.
L'intérêt de ces méthodes réside dans une décomposition de l'espace de recherche et l'utilisation de méthodes mono-objectif pour les différentes résolutions successive (recherche des extrêmes, résolution des agrégation...). Appliquer chacune de ces méthodes pour la résolution d'un problème bi-objectif nécessite donc d'avoir une méthode mono-objectif efficace (si possible polynomiale). Cependant, lorsque la restriction du problème à un seul objectif génère un problème déjà NP-difficile, la non existence de méthode exacte efficace pouvant optimiser chaque objectif séparément peut compromettre l'intérêt de ces méthodes, qui peuvent être très coûteuse en temps et particulièrement la recherche des points extrêmes peut consommer beaucoup de temps. C'est le cas pour la méthode TPM de Ulungu et Teghem [UT95]. Si la méthode mono-objectif est efficace, alors TPM sera très performante, ceci est le cas pour le problème d'affectation linéaire sur lequel la méthode a été proposée. Cette méthode a été adaptée au problème de Flowshop bi-objectifs en utilisant l'al-

gorithme du Branch & Bound pour résoudre l'agrégation dans [Lem06]. Elle a été ensuite améliorée en fonctions des points particuliers de la classe des problèmes et les objectifs étudiés [Dha05]. Pour découper l'espace de recherche, la Méthode des Deux Phase utilise les solution supportées, elle implique de nombreuses recherches pour les retrouver. De plus certaines recherches n'amènent pas de nouvelles solutions, elles sont utilisées juste pour prouver qu'il n'existe pas d'autre solutions entre deux solutions supportées. Il est donc évident que le nombre de recherche mono-objectif peut être très grand avec cette méthode. La méthode de *Özlen* et *Azizoğlu* [OA09] elle aussi se ramène à résoudre un problème mono-objectif. Elle résout initialement $2r$ programmes linéaires en nombres entiers pour le calcul de f_i^{ub} et f_i^{lb} de tous les objectifs $i, i = 2, ..., r$, et que le programme mono-objectifs $(Q(1))$ est résolu autant de fois qu'il y a de solutions non dominées. Cependant, la dimension de $(Q(1))$ ne varie pas, le nombre de variables et de contraintes est le même durant tout le processus de révolution. Cette méthode dépend donc, du nombres d'objectifs et celui des solutions non dominées du problème multi-objectif.

La méthode Parallèle Par Partition [LDT05b] quand à elle, s'inspire de l'idée de découpage de l'espace de recherche de la méthode TPM et de la méthode $\epsilon-contrainte$ pour réaliser ce découpage. Mais contrairement à la méthode TPM qui utilise les solutions supportées pour découper l'espace de recherche, PPM s'appuie sur tout type de solutions (supportées ou non-supportées). Ainsi ces solutions ont plus de chance d'être bien réparties sur le front Pareto. Cette nouvelle méthode tient compte des différentes limites de la méthode des Deux Phase et essai d'y répondre. Une étude comparative des deux méthodes faite par Julien Lemesre [Lem06] a permet de caractériser les types de problèmes pour lesquels l'une ou l'autre des méthodes sera la plus efficace.

Suite à cette étude, nous avons choisi de résoudre notre problème par la méthode dédiée aux problèmes multi-objectif de *Özlen* et *Azizoğlu* et par la méthode Parallèle par Partition pour les problèmes bi-objectif.

3.4 Conclusion

Dans ce chapitre, nous avons présenté le problème de Flow-Shop de permutation Bi-Objectif noté $F/perm, d_i/(C_{max}, \sum w_i T_i)$.

Pour cela, nous avons commencé par une étude des problèmes d'ordonnancement et avons introduit la notation de Graham et al afin de classer ces problèmes. Nous avons décrit les différentes valeurs que peut prendre cette notation. Nous avons en suite vu en détail le problème traité dans ce document. Nous avons caractérisé les objectifs du makspan et le retard total pondéré. Nous avons en suite proposé un état de l'art des méthodes de résolution du Flow-Shop multi-objectif. Puis nous avons modélisé notre problème par programme linéaire en nombre entiers. Le modèle que nous proposons considère des temps de transportation des jobs d'une machine à une autre. En outre, nous nous limitons au cas où il existe un nombre infini de transporteurs. Pour résoudre notre problème, nous avons fait l'étude de trois méthodes exactes, La méthode de *Özlen* et *Azizoğlu*, la méthode des Deux Phases, et la méthode Parallèle Par Partition. La première méthode est dédiée aux problèmes Muti-Objectif, tandis ce que les deux autres sont dédiées aux problèmes Bi-Objectif. Pour finir, une discussion sur les trois méthodes proposées a été faite, montrant les avantages de chaque méthode. Suite à cette discussion, nous avons décidé de résoudre notre problème par les deux méthodes : la méthode PPM et la méthode *Özlen* et *Azizoğlu*.

CHAPITRE 4

Conception et Implémentation

Dans ce chapitre, nous nous intéressons à la parallélisation des méthodes que nous avons choisi pour résoudre notre problème. Nous verrons ensuite comment implémenter ces méthodes dans notre grille. Nous terminerons le chapitre par les résultats expérimentaux commentés .

4.1 Introduction

Les méthodes exactes permettent de trouver l'intégralité du front Pareto et aussi de prouver leur optimalité. Ceci peut prendre un temps de calcul très important. Le parallélisme à grande échelle, basé sur les grilles de calcul, s'avère un moyen indispensable pour supporter le coût des ces méthodes d'optimisation. L'exploitation de cet environnement nécessite la "gridification" des modèles parallèles en vue de leur adaptation aux caractéristiques des grilles de calcul.
Le parallélisme, par la force de calcul qu'il apporte, permet d'augmenter l'efficacité d'une méthode de résolution en terme de temps d'exécution. Il permet aussi de s'attaquer à des instances de grande taille.

4.2 Parallélisation de la méthode de *Özlen* et *Azizoğlu*

La méthode de *Özlen* et *Azizoğlu* recherche les solutions non-dominées une à une. A chaque étape, la solution trouvée précédemment est utilisée pour borner l'espace de recherche.

Cette méthode ne peut être parallélisée de façon efficace. En effet, la contrainte utilisée afin de trouver une solution non-dominée est fournie par la solution précédente de sorte que le procédé présente un schéma récurrent. Cette méthode peut donc être exécutée seulement de manière séquentielle.

4.3 Parallélisation de PPM

La méthode PPM se décompose en trois phases :

1. La recherche des deux solutions extrêmes (Pareto optimales) et le partitionnement de l'espace de recherche. Ces deux solutions indiquent les valeurs minimale et maximale du front Pareto pour chacun des objectifs.

Les calculs des deux solutions extrêmes sont indépendants, donc il peuvent être calculés simultanément. Deux processeurs vont être utilisés pour le calcul des solutions extrêmes.

2. Pour chaque partition ou "split", une solution Pareto est calculée, celle qui respecte le split et qui a la meilleure valeur de l'objectif non utilisé dans le découpage de l'espace de recherche, autrement dit, un objectif est borné, l'autre est optimisé.

   Lors de cette phase, les recherches des solutions dans chaque partition sont aussi indépendantes. Elles sont donc lancées simultanément. Ainsi pour pouvoir lancer les recherches bornées de la deuxième phase sur chaque partition en une seule étape, il faut que le nombre de processeurs soit suffisant, de plus, il faut définir une stratégie de parallélisation, autrement dit, est ce que les processeurs disponibles vont êtres utilisés pour trouver les autres solutions ou bien pour le lancement de la troisième phase.

3. Les solutions non trouvées durant les étapes précédentes sont recherchées lors de cette phase. Pour deux solutions adjacentes $f(s)$ et $f(r)$, une recherche est faite dans le rectangle $SYRO$ où $Y = (f_1(s), f_2(r))$, $R = (o, f_2(r))$, $S = f_1(s), o)$.

   Les recherches de solutions de la troisième étape ne sont, elles aussi, pas liées, elles peuvent être lancées en une seule étape. Comme pour la deuxième phase, le nombre de processeurs doit aussi être suffisant pour pouvoir effectuer cette phase en une seule étape.

Le modèle de parallélisme adopté est un modèle de type "maître-esclave", un processeur sert de maître et distribue aux autre processeurs les sous-problèmes.
Pour une parallélisation efficace de PPM, le nombre de partition doit être étudié, Lemesre et al [LDT05b] proposent de le fixer au nombre de processeurs.
Dans ce cas, seulement deux processeurs seront utilisés lors de la première phase, un seul processus sera inutile lors de la deuxième étape, car le nombre de recherches

effectués lors de cette phase est égale au nombre de partitions -1, et tous les processeurs seront utilisés lors de la troisième phase.

4.4 Mise en œuvre et expérimentation

4.4.1 Grille d'expérimentation de nos travaux

Les expérimentations présentées dans ce document ont été réalisées sur la grille de calcul basée sur le middleware gLite, elle est installée par l'équipe de recherche sur le Grid Computing de la division Réseaux du Centre de Recherche sur l'information Scientifique et Technique, CERIST, dans le cadre du projet d'extension EumedGrid Support [EumedGrid] qui consiste à supporter l'infrastructure EumedGrid déjà établi dans les pays Euro-Méditerranéens partenaires y compris l'Algérie.
Le cœur de l'approche eumedgrid est d'établir un réseau humain dans le secteur du "e-Science", élargissant et formant cette communauté et établissant une infrastructure grille pilote supportant le concept d'applications régionales.
Le logiciel utilisé pour la programmation et la mise en œuvre des méthode est le Parallel Matlab. La section suivante contient une description détaillée de "Parallel Matlab", technologie et produits, ainsi que son intégration avec les grilles de calcul [CGE08].

4.4.2 Matlab et le calcul parallèle

Les ingénieurs et les scientifiques ont besoin de pouvoir exploiter facilement des ressources de calcul hautement performantes afin d'exécuter des simulations à grande échelle et de générer le code plus rapidement [CGE08]. Le temps d'exécution de telle applications dépassent de loin les capacités des systèmes monoprocesseurs traditionnels. MathWorks a répondu à cette demande en commençant par supporter le calcul parallèle dans Matlab. Deux produits ont été réalisé en 2004 : Distributed Computing Toolbox (depuis renommé en Parallel Computing Toolbox) [PCT] et Matlab

Distributed Computing Engine (depuis renommé en MATLAB Distributed Computing Server) [MDCS]. Depuis lors, les utilisateurs de Matlab et Simulink ont tiré avantage de la programmation parallèle et l'exécution sur les ordinateurs multicurs, "workgroup" et des "clusters".

Matlab et Parallel Computing Toolbox (PCT) fournissent des environnements de développement et d'exécution pour les applications parallèles sur le poste d'un utilisateur. PCT peut lancer jusqu'à 4 processus MATLAB sur un poste de travail, qui permet à l'utilisateur de tester et de déboguer localement un programme parallèle. Matlab Distributed Computing Server (MDC) fournit l'environnement d'exécution pour des applications sur les "clusters" ou les grilles. Plusieurs Matlab "worker" peuvent être utilisé pour exécuter des applications en parallèle.

4.4.3 Le langage parallèle

Matlab fournit un ensemble de constructions qui peuvent être appliquées pour exploiter les différents types de parallélisme. L'utilisateur peut choisir un sous-ensemble spécifique de ces constructions dans le but d'exploiter le parallélisme dans leurs applications. La plupart des utilisateurs de Matlab choisissent des constructions de haut niveau, comme les boucles pour parallèle et les tableaux distribués, tandis que les programmeurs avancés pourraient utiliser des fonctions bas niveau de passage de messages. La boucle parallèle, "parfor", exploite le parallélisme des tâches. Un utilisateur exprime que les itérations d'une boucle "for" sont indépendantes afin d'annoter "for" comme un "parfor" . Pendant l'exécution du programme, les itérations de "parfor" sont distribués pour être exécuté sur les "Matlab worker" disponibles dans le "cluster" [CGE08].

```
% Original for loop
for itr = m : n
    % loop body
end
```

```
% Parallel for loop
parfor itr = m : n
    % loop body
end
```

FIGURE 4.1 – La boucle For parallèle

Les experts du calcul parallèle ont la possibilité d'utiliser les fonctionnalités de bas niveau pour exercer un meilleur contrôle sur leurs applications : les traitements par batch qui contiennent des tâches dépendantes ou indépendantes, fonctions "Matlab Message Passing" qui sont basées sur le MPI (MPICH2). Le parallélisme a été construit dans le langage MATLAB grâce à un ensemble de constructions fonctionnelles et les structures de données.

4.4.4 Parallel Matlab sur la grille

MATLAB et Grid computing sont reconnus comme des outils clés de la science et de l'ingénierie. L'intégration de ces technologies est une demande naturelle de la communauté scientifique et industrielle : les utilisateurs de MATLAB seront en mesure de résoudre des problèmes plus importants en exécutant leurs applications en parallèle sur les grilles de calcul, tandis que les utilisateurs Grid ont la capacité de développer des applications complexes dans MATLAB. Une exigence pour l'intégration de MATLAB et grille de calcul est que l'utilisateur sera en mesure de développer une application MATLAB sans avoir besoin de considérer la variété des environnements dans lesquels il pourrait être exécuté.

4.4.4.1 L'exécution parallèle d'un code Matlab sur la grille

Un utilisateur de MATLAB parallélise une application en utilisant des constructions de langage de haut niveau comme "parfor" ou à faible niveau comme le passage de message, MPI. L'utilisateur n'a pas besoin d'écrire aucun code spécial pour tenir

compte du fait que la demande sera exécuté sur une grille. L'exécution de l'application sur un système de grille comporte habituellement les étapes suivantes :

i L'utilisateur sélectionne un «local» de configuration distribuée et test le programme en parallèle avec les workers locaux de MATLAB sur un poste de travail.

ii Une fois le programme s'exécute correctement, l'utilisateur sélectionne une configuration pour le système de grille et exécute le programme à nouveau. Cette fois, sans aucune intervention ou la direction de l'utilisateur, MATLAB va transférer les données d'entrée et le code de la grille, exécuter le code en parallèle sur les travailleurs de MATLAB, et récupérer les données de sortie.

4.4.4.2 Intégration de Matlab avec gLite

Parallel Computing Toolbox et Matlab Distributed Computing Server sont totalement intégrés à gLite, grace à EGEE-Enabling Grids for E-sciencE. Avant cette intégration, il n'était pas possible de travailler avec Matlab dans un environnement de grille de calcul. Les utilisateurs de la grille peuvent maintenant accéder à la puissance de la grille directement à partir de leur ordinateur.
Lorsqu'un utilisateur exécute une application parallèle, MATLAB génère automatiquement des fichiers Job Definition Language et autres scripts spécifiques à gLite, copie les fichiers d'entrée à un élément de stockage (SE) et enregistre les fichiers avec un catalogue de fichiers, LFC. Les commandes de soumission sont ensuite appelées à soumettre les jobs à la gLite. L'utilisateur peut continuer à utiliser MATLAB pour faire un autre travail alors que les tâches sont exécutées sur la grille. Le Ressource Broker "RB" des sites locaux localise les sites qui ont des installations MDCS. Les Matlab Workers sont démarrés sur les worker nodes, WN, et l'application est exécutée en parallèle. Les données de sortie sont enregistrées sur le SE.

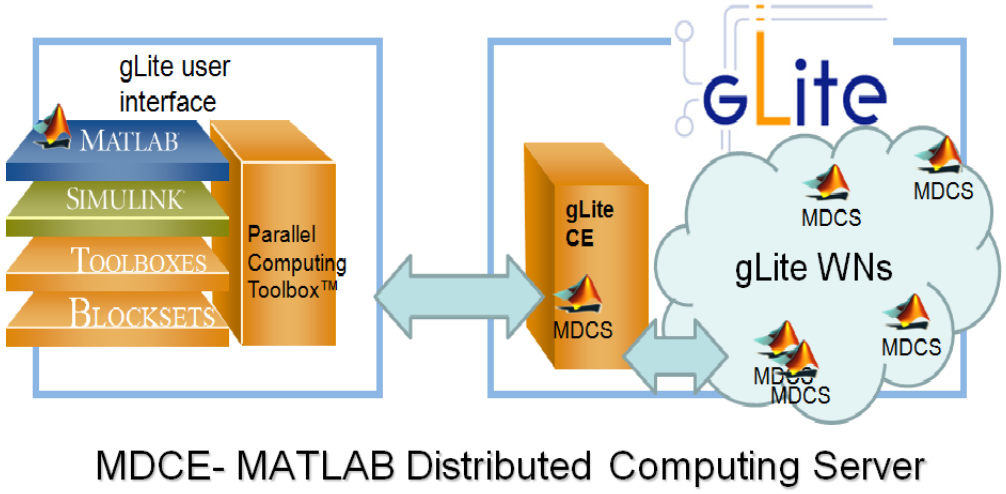

FIGURE 4.2 – Intégration de Matlab avec gLite

. [CGE08]

4.4.5 Exemple illustratif

Matlab et PCT sont installés sur un serveur g6, quadri-core, 4GB RAM avec système d'exploitation Scientific linux vs 5.4. MDCS est installé sur un cluster avec 4 serveurs en rack. Chaque serveur a 2 CPUS, chacun avec 4 curs, 1GB de Ram et 160GB de disque, système d'exploitation Scientific linux vs 5.4. L'accès distant aux cluster se fait avec SSH et PuTTY. Nous considérons le problème de Flow-shop de permutation où trois jobs doivent être ordonnancés sur deux machine, càd, $n = 3$, $m = 2$

Les temps d'exécution des jobs sur les deux machines sont tirés de la littérature et donnés par le tableau suivant :

Machine	Jobs		
	1	2	3
1	6	10	20
2	5	15	6

TABLE 4.1 – Temps d'exécution des jobs sur les machines

Le tableau suivant donne le temps de transportation des jobs d'une machine vers une autre

	Jobs		
	1	2	3
Temps de transport à la machine 2	4	3	5
Temps de retour à la machine 1	3	3	3

TABLE 4.2 – Temps d'exécution des jobs sur les machines

Et enfin, Les dates dues des jobs

D_i	21	27	15

TABLE 4.3 – Les dates dues des jobs

Pour cette exemple, les poids des retards sont tous égeaux à 1.
Deux solution ont été trouvées

– Première solution :

le job 1 occupe la position 3, le job 2 occupe la position 1, le job 3 occupe la position 2.

$C_max = 29$, $T = 50$.

– Deuxième solution :

le job 1 occupe la position 2, le job 2 occupe la position 1, le job 3 occupe la position 3.

$C_max = 49$, $T = 49$.

4.4.6 Expérimentations et résultats

Le tableau 4.4 présente les différents temps d'exécution nécessaires en fonction de la méthode utilisée, l'environnement d'exécution et le nombre de processeurs disponibles. Pour pouvoir effectuer les expérimentations, nous avons programmé

Instances		Temps d'exécution					
n	m	*Özlen&Azizoğlu* sur machine	*Özlen&Azizoğlu* sur cluster	PPM			
				P=1	P=2	P=3	P=4
2	3	21s	10.33s	15.5s	16.95s	17.45s	30 s
4	5	5 min 12s	3min 25s	4 min	3 min 45s	3 min 55s	3 min 56s
10	5	57 min 25 s	30 min 22 s	33 min	29 min	22 min	20 min 32 s
10	10	5 h 40 min	3 h 55 min	3 h 20 min	2h 50 min	2h 03 min	1h 45 min
20	10	8 h 39 min	4 h 40 min	4h 55 min	3h 44 min	2h 58 min	1h 49 min
20	15	non-résolu	11h 04 min	10h 55 min	8h 44 min	5h 23 min	3h 58 min
25	20	non-résolu	24h 04 min	24h 55 min	18h 44 min	15h 23 min	11h 58 min
30	20	non-résolu	non-résolu	non-résolu	non-résolu	non-résolu	non-résolu

TABLE 4.4 – Temps d'exécution des méthodes

la matrice des contraintes A et les vecteurs C et b représentant respectivement le vecteur coût et le vecteur second membre du modèle.

Nous avons généré des instances aléatoires de la manière suivante :
Nous avons pris $n = 2, 4, 5, 10, 20$ et $m = 3, 5, 10, 15$. Les temps d'exécutions, de transport et les dates dues des jobs sont générés aléatoirement à partir d'une distribution uniforme sur (1,99), (1,53) et (1,200), respectivement. Pour chaque valeur de n et m nous avons généré trois instances, les temps donnés dans le tableau représentent le temps de calcul moyen de ses instances.
Ce tableau illustre bien l'apport des grilles de calcul et le calcul parallèle en terme de gain de temps d'exécution. En effet, grâce à la puissance des ressources d'un tel environnement, le temps nécessaire pour trouver l'ensemble des solutions Pareto optimales par la méthode d'*Özlen&Azizoğlu* s'est réduit considérablement. Notons d'ailleurs que la grille permet la résolution de problèmes de taille 20x15, ce qui n'était pas le cas avec une machine simple.
Nous remarquons aussi que PPM permet un gains de temps important sur des instances de grande tailles à chaque fois que le nombre de processeurs augmente. Ce gain est plus au moins importante lorsqu'il s'agit des instances de petite taille. Ceci se traduit par le fait que la boucle "parfor" peut être plus lente que la boucle "for" pour les problèmes de petite taille.

4.5 Conclusion

Dans ce chapitre, nous avons présenté un moyen pour réduire le temps de calcul requis pour trouver les exactes du problème : la parallélisation de la méthodes et leurs exécution sur une grille de calcul.

Dans un premier lieu, nous avons présenté la parallélisation des méthode exactes. Nous avons constaté que la méthode de *Özlen* et *Azizoğlu* ne peut pas être paralléliser, car les recherches des solutions ne sont pas indépendantes, par contre la méthode PPM représente un schéma de parallélisation très intéressant. Nous avons ensuite parlé des outils qui permettent la parallélisation. Nous avons donné une description détaillée de Parallel Matlab et son intégration avec la grille de calcul. Nous avons terminé le chapitre par les résultats.

Conclusion

Comme pour tout problème d'optimisation combinatoire mono-objectif, la résolution dans le contexte multi-objectif peut être faite par différents types d'approches.

Parmi ces approches, les méthodes exactes sont toujours intéressantes car elles permettent d'obtenir la solution optimale au problème. Le problème est que ces méthodes sont très souvent gourmandes en temps et elles ne sont pas utilisables pour des problèmes difficiles de grande taille. Ceci est d'autant plus vrai en multi-objectif où la présence de plusieurs objectifs augmente en général la difficulté du problème.

Dans ce mémoire, nous nous somme intéressés à la résolution exacte du problème d'ordonnancement de type Flow-Shop de permutation Bi-Objectif. Ce problème est connu pour être NP-difficile même dans le cas mono-objectif [GJS76]. Ce problème appartient à la classe des problèmes d'atelier.
Dans un premier temps nous avons présenté le formalise ainsi que les concepts propres aux problèmes d'optimisation combinatoire Multi-Objectif. Nous avons donné un état de l'art sur les méthodes exactes existantes. Ces méthodes réduisent considérablement la puissance de calcul nécessaire pour résoudre un problème d'optimi-

sation combinatoire. Néanmoins, cette puissance reste considérable lorsqu'il s'agit de résoudre des instances de très grande taille. C'est pourquoi un recours au calcul parallèle sur les grilles de calcul s'avère d'un grand secours pour la résolution de telles instances.

Une étude détaillée sur les grilles de calcul en général, et les grilles de calcul basées sur gLite a été faite. Le concept de la grille a été mis au point pour répondre aux différents besoin en optimisant au maximum l'utilisation des moyens de traitement et de stockage disponibles. Elles ont pour objectif de fournir, de manière transparente et sûre, à des communautés d'intérêt (organisation virtuelle) l'accès à des moyens de traitement et de stockage hétérogènes, individuellement ou incompatibles avec les propres moyens financiers d'une structure telles que des puissances de calcul de plusieurs CPUs ou des capacités de stockage de l'ordre de petabyte.

Nous nous sommes intéressés à la modélisation mathématique du problème de Flow-Shop de permutation Bi-Objectifs. Pour le résoudre, nous avons présenté trois méthodes exactes.

Le parallélisme apparaît comme un excellent moyen pour réduire les temps de résolution de ce type de problèmes moyennant un schéma de parallélisation adéquat. En effet, nous avons vu que ce n'est pas toutes les méthodes qui s'y prêtent au parallélisme compte tenu de la structure algorithmique utilisé. Nous avons ensuite montrer comment implémenter ces méthodes en utilisant Parallel Matlab sur la grille de calcul.

Les expérimentations sur la méthode d'*Özlen* et *Azizoğlu* et la méthode Parallèle par Partitions montrent l'intérêt d'utiliser les grilles de calcul pour résoudre les problèmes d'optimisation combinatoire. En effet, les résultats ont montré que l'exploitation d'un tel environnement amène un gain important en terme de temps de calcul grâce au calcul parallèle d'une part et la puissance de ces ressources.

Le domaine des grilles de calcul est récent et les perspectives sont donc nombreuses. Tout d'abord, la propositions des méthodes exactes pour les problème d'ordonnancement Multi-Objectifs pour plus de deux objectifs.

Une autre perspective intéressante concerne une meilleure exploitation du parallélisme. En effet, même si PPM optimise le parallélisme de haut niveau en rendant indépendantes chaque recherche dans chacune des phases, certaines recherches restent trop longues. Il serait donc intéressant de paralléliser chaque méthode de recherche (recherche des extrêmes, des solutions représentantes de chaque partition, des solutions Pareto au sein des triangles) de façon à les accélérer. Ceci est tout à fait réalisable surtout lorsque les problèmes associés sont difficiles et que les seules méthodes possibles sont basées sur des méthodes énumératives tel que la méthode Branch & Bound. La parallélisation de bas niveau de ce type de méthodes se fait à l'aide du langage MPI : Message Passing Interface.

Bibliographie

[AA05] J.E.C. Arroyo, V.A. Armentano. Generic local search for multi-objective flowshop scheduling problems. european Journal of operational Research, 167 : 717-738, 2005.

[ALL04] A. Allaverdi. Anew heuristic fom m-machine flowshop scheduling problem with bicriteria of makespan and maximum tardiness. Computer and operations research, 31 :157-180, 2004.

[BEP96] J. Blazewicz, K.H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. Scheduling Computer and Manufacturing Processes, Springer, Berlin, 1996.

[Bou08] H.L Bouziane. De l'Abstration des Modèles de Composants Logiciels pour la Programmation d'Applications Scientifiques Distribuées, Thèse de doctorat, Université de Rennes1, 2008.

[Bow76] V.J. Bowman. On the relationship of the tchebycheff norm and the efficient frontier of multiple-criteria objectives. In H. Thiriez & S. Zionts (eds), MCDM, Springer-Verlag, Berlin, pages 76-85, 1976.

[CMM90] R.L. Carraway, T.L. Morin, H. Moskowitz. Generalized dynamic programming for multicritaria optimization. European Journal of Operational Research,

44 : 95-104, 1990.

[Cha05] D. Chaabane. Optimisation multicritère en nombres entiers. Thèse de doctorat, Université des Sciences et de la Technologie Houari Boumediene, Algérie, 2005.

[CGE08] A.J. Chakravarti, S. Grad-Freilich, E. Laure. Enhancing e-Infrastructures with Advanced Technical Computing : Parallel MATLAB on the Grid. The Mathworks,Inc,Natick,MA,USA, EGEE-TR-2008-001.

[Che10] M.E.A. Chergui. Contribution à la programmation nonlinéaire. These de doctorat, Université des Sciences et de la technologie Houari Boumediane, Alger, 2010.

[CCLL97] P. Chrétienne, E.G. Coffman, J.K. Lenstra, Z. Liu. Scheduling Theory and its Application, John Wiley& Sons Ltd, 1997.

[CDG] D. Corne., M. Dorigo, and F. Glover, editors. New Idea in Optimization. McGraw-Hill, 1999.

[CMM67] R. Conway, W. Maxwell, and L. Miller, Theory of scheduling. Addision-Wesley, 1967.

[Dan] G.B. Dantzig. Linear programming and extensions. Princeton University Press, Princeton, 1963.

[Dar] C. Darwin. On the origin of species. John Murray, London, 1959.

[DC90] R.L. Daniels, R.J. Chambers. Multiobjective flow-shop scheduling. Naval research logidtics, 40 :85-101, 1990.

[DFPSK96] T.A. DeFanti, I.Foster, M.E. Papka, R. Stevens, and T. Kuhfuss. Overview of the I-WAY : Wide-Area Visual Supercomputing. The Ontel. Journal of Supercomputer Applications and High Performance Computing,page 123-131, 1996.

[Des06] G. Desmottes. Déploiement et configuration des intergiciels européens de grilles de calcul, Mémoire licence, Université Libre de Bruxelles, 2006.

[Dha05] C. Dhaenens-Flipo. Optimisation Combinatoire Multi-Objectif : Apport des Méthodes Coopératives et Contibution à l'Extraction de Connaissances, These HDR,Université des Sciences et Technologie de Lille, 2005.

[DL90] J. Du , J. Y-T. Leung. Minimizing total tardiness on one machine is NP-hard. Mathematics of Operations Research, 15 :483-495, 1990.

[Edg81] F.Y. Edgeworth. Mathematical physics. P.Keagan, London, England, 1981.

[Fed03] G. Fedak. XtremWen : une plate-forme pour l'étude expérimentale du calcul global pair-à-pair. These de doctorat, Université Paris XI, 2003.

[FJ] C. Fellenstein, J. Joseph : Grid Computing, IBM Press, ISBN-10-13-145660-1.

[FK97] I. Foster, C. Kesselman. Globus : A Metacomputing Infrastracture Toolkit. Journal of Supercomputer Applications,11(2, page 115-128, 1997).

[FK99] I. Foster, C. Kesselman. The Grid : Blueprint for a new Computing Infrastracture. Morgan-Kaufmann, 1999.

[Fou85] M.P. Fourman. Compaction of symbolic layout using genetic algorithms. in proceeding of the first international conference on genetic algorithms (ICGA), pages 141-153, 1985.

[Geo68] A.M. Geoffrion. Proper efficiency and the theory of vector maximization. Journal of Mathematical Analysis and Applications 22, pages 618-630, 1968.

[GJS76] M.R. Garey, D.S. Johnson, R. Sethi. The complexity of flow-shop and job-shop scheduling. Mathematics of Operations Research, 1 : 117-129, 1976.

[GWBG04] M. Gerndt, R. Wismuller, Z. Balaton, G. Gombas, Z. Németh, N. Podhorski, H.-L. Truong, T. Fahringer, M.Bubak, E.Laure, and T.Margalef. Performance Tools for the Grid : State of the Art and Future, chapter Volume 30 of LRR-TUM Research Report Series35, Shaker Verlag, Aachen,2004.

[Glo86] F. Glover. Future paths for integer programming and links to artificial intelligence. Computer and Operations research, 13(5) ; 533-549, 1986.

[Gom58] R.E. Gomory. Outline of an algorithm for integer solutions to linear programs. Bulletin of the American Mathematical Society 64, pages 275-278, 1958.

[Got93] GOThA. Les problèmes d'ordonnancement, RAIRO-Recherche Opérationnelle 27, pages 77-150, 1993.

[GLLR79] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy kan. Optimization and approximation in deterministic sequencing and dcheduling : a survey. In annals of Discrete mathematics, volune 5, pages 287-326. 1979.

[GM92] R. Gupta and R. Malhotra. Multi-criteria integer linear programming problem. Cahiers de CERO 34, pages 51-68, 1992.

[GHW02] J.N.D Gupta, K.Henning, F. Werner. Local search heuristics for tow-stage flow shop probleme with secondary criterion. Computer and Operations Research, 29 : 123-149, 2002.

[GNW01] J.N.D Gupta, V.R. Neppali, F. Werner. Minimizing total flow time in a two-machine flowshop problem with minimum makespan. International Journal ofProduction Economics, 69 : 323-338, 2001

[Han86] P. Hansen. The steepest ascent mildest descent heuristic for combinatorial programming. In Numerical methods in Combinatorail Optimization, Capri, Italy, 1986.

[Hol75] J.H. Holland. Adaptation in natural and artificial system. The University of Michigan press, Ann Arbor, MI, USA, 1975.

[KGV83] S. Kirkpatrick, C.D. Gellat, and M.P. Vecchi. Optimiszation by simulated annealing. Science, 220(4589) : 671-680, 1983.

[Kon09] O. Kone. Nouvelles approches pour la résolution du problème d'ordonnancement de projet à moyens limités. Thèse de doctorat. l'Université Toulouse III - Paul Sabatier, 2009.

[Law77] E.L. Lawler. A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. Annals of Discrete mathematics, 1 : 331-342, 1977.

[LTZ04] M. Laumanns, L. Thiele, and E. Zitzler. An adaptative scheme to generate the pareto front based on the epsilon-constraint method. Technical Report 199, Institut F Technische Informatik und Kommunikationsnetze, Zrich, Switzerland, 2004.

[Lem06] J. Lemesre, Méthodes Exactes pour l'Optimisation Combinatoir Multi-Objectif : Conception et Application, Thèse de Doctorat, Université des Sciences et Technologie de Lille, 2006.

[LDT05a] J. Lemesre, C. Dhaenens, and E-G. Talbi. An exact parallel method for a bi-objective permutation flowshop, European Journal of Operational Research, 2005.

[LDT05b] J. Lemesre, C. Dhaenens, and E-G. Talbi. Parallel Partitioning Method(PPM) : A new exact method to solve bi-objective problems, Computers and Operations research, 2005.

[Lem54] C.E. Lemke. The dual method for solving the linear programming problem. Naval Research Logistic Quarterly 1, pages 36-47, 1954.

[Lie09] A. Liefooghe. Méthaheuristiques pour l'Optimisation Multi-Objectif : Approches coopératives, prise en compte de l'incertitude et application en logistique, These de Doctorat, Université des Sciences et Technologie de Lille, 2009.

[LM96] J.L Liu, B.L MacCarthy. The classification of FMS scheduling problems. International Journal of Production Research, 34(3) : 647-656, 1996.

[LR01] P. Lopez, F. Roubellat. Ordonnancement de la production, Hermes Sciences, IC2 productique, 2001.

[LYJ97] C.J. Liao, C.J. Yu, C.B. Joe. Bicriterion scheduling in the two-machine flowshop. Journal of the Operational Research Society, 48 :929-935, 1997.

[MG04] M.B. McConnel and D.T. Galligan. The use of integer programming to select bulls across breeding companies with volume price discounts. J. Dairy Sci. 87, pages 3542-3549, 2004.

[Meu02] H. Meunier. Algorithmes Evolutionnaire parallèles Pour l'Optimisation Multi-Objectif de Réseaux de Télécommunications Mobiles, thse de doctorat, Université des Sciences et Technologie de Lille, 2002.

[Mos89] P. Moscato. on evolution, search, optimization, genitic algorithm and material arts : Towards memetic algorithms. Technical report 826, calefornia institute on technology, Pasadena, california, USA, 1989.

[Mou02] M. Moulaï. Optimisation multicritère fractionnaire linéaire en nombres entiers. Thèse de doctorat, Université des Sciences et de la Technologie Houari Boumediene, Algérie, 2002.

[NAJ09] B. Naderi, A.Ahmadi Javid, F.Jolai. Permutation flowshops with transportation time :mathematical models and solution methods. Springer-Verlag London Limited,46 :631-647,2009.

[NGB04] Z. Németh, G. Gombas, and Z. Balaton, performance Evaluation on Grids :Directions, Issues and Open Problems, In12th Euromicro PDP, pages 290-297, IEEE Service Center, 2004.

[NHH95] A. Nagar, S.S. Haragu, and J. Haddock, A Branch & Bound approach fot two-machine flowshop scheduling problem. Journal on the Operational Resaerch Society, 46 : 721-734, 1995.

[NW88] G.L. Nemhauser and L.A. Wolsey. Integer and combinatorial optimization. Wiley, New York, 1988.

[OA09] M. zlen, M. Azizo. Multi-Objective integer programming : A general approch for generating all non-dominated slutions. European Journal of Operational Research 199, pages 25-35, 2009.

[Pap76] C.H. Papadimitriou. The complexity of combinatorial optimization problems. These de doctorat, Princeton university, New Jersey, USA, 1976.

[PS82] C.H Papadimitriou, K. Steiglitz. Combinatorial Optimization : Algorithms and Complexity. Prentice-HALL, 1982.

[Par86] V. Pareto. Cours d'économie politique, Rouge, Lausanne, Switzerland 1986.

[Pin95] M. Pinedo. Scheduling - Theory, Algorithms, and Systems. Prentice Hall edition, 1995.

[Raj92] C. Rajendran. Two-stage flow shop scheduling probleme with bicriteria. Journal on the Operational Research Society, 43 :817-884, 1992.

[SAZ08] S.J Sadjadi, M.B Aryanezhad, M.Ziaee. The General Flowshop Scheduling Problem : mathematical Models. Journal of Applied Sciences,8(17) :3032-3037, 2008

[Sak84] M. Sakarovitch. Optimisation combinatoire, programmation discrète volume 2. Hermann, 1984.

[San94] E. Sandgren. Advances in design optimization. Chapman and Hall, chapter Multicritaria design optimization by goal programming, 1994.

[SK99] S. Sayin, S. Katabi. A bicriteria approch to the two machine flow shop scheduling problem. European Journal of Operational Research, 113 : 435-449, 1999

[SSU98] F.S. Sivrikaya-Serifouglu and G. Ulusoy. Abicriteria two machine permutation flowshop problem. European journal of Operational Resesrch, 107 : 414-430,1998.

[Stu85] R.E. Steuer. Multiple criteria optimization : theory, computation and application. John Wiley & Sons, New York, 1985.

[Ste91] B.S. Stewart, C.C. White. Multiobjectictive A*. Journal of the ACM, 38(4) : 775-814,1991.

[TAk04] B. Toktas, M. Azizoglu, S.K. Koosalan. Two-machine flow shop scheduling with two criteria : Maximum earliness and makespan. European Journal of Operational research, 157 : 286-295, 2004.

[TB02] V. T'Kindt, J.C Billaut. Multicritaria Scheduling : theory, models and algorithm. Springer-Verlag, 2002.

[TGB03] V. T'Kindt, J.N.D. Gupta and J.C. Billaut. Two-machine flowshop with a secondary criterion. Computers and Operationa Research, 30 : 505-526, 2003.

[TMTL02] V. T'Kinkt, N.Monmarché, F. Tecinet, and D. Laugt. An Ant colony optimisation algorithm to solve a 2-machine bicriteria flowshop scheduling problem. European Journal of operational Research, 142 :250-257, 2002.

[TS01] P.R. Tozer and J.R. Stokes. Using multiple objective programming in a dairy cow breeding program. J. Dairy Sci. 84, pages 2782-2788, 2001.

[UT95] E.L. Ulungu and J.Teghem. The two phases method : An efficient procedure to solve bi-objective combinatorial optimization problems. Foundation of computing and decision science, 20 : 149-156, 1995.

[VR05] T.K. Varadharajan, C. Rajendran. A multi-objective simulated-annealing algorithm for scheduling in flowshops to minimize the makespan and total flowtime of jobs. European Journal of Operational Research, 167 : 772-795, 2005.

[YA04] W.C. Yeh, A. Allahverdi. A branch-and-bound algorithm for the three-machine flowshop scheduling problem with bicriteria of makespan and total flowtime. Internation Transaction in Operational Research, 11 : 323, 2004.

[Yeh99] W.C. Yeh. A new branch-and-bound approach for the n/2/flowshop/$\alpha F + \beta C_{max}$ flowshop scheduling problem. Computer and Operations research, 26(13) : 1293-1310,1999.

Sites web :

[EGEE] EGEE :htpp ://www.eu-egee.org

[EumedGrid] EumedGrid Support :http ://www.eumedgrid.eu/

[gLite] gLite middleware site web officiel : http ://glite.cern.ch/

[Globus.A] Globus Alliance : http ://www.globus.org

[Globus.T] Globus Toolkit : http ://www.globus.org/toolkit

[GOThA] Groupe de recherche en Ordonnancement Théorique et Appliqué : http ://www.poleia.lip6.fr/sourd/gotha/livres.html

[MDCS] Matlab Distributed Computing Server System User's Guide :

[PCT] Parallel Computing ToolboxUser's Guide :

[Scilab] Scilab Le logiciel libre et gratuit de calcul numérique http ://www.scilab.org

[Unicore] Unicore : http ://www.unicore.eu

Ordonnancement de type "Flow-Shop" de permutation Bi-Objectif Résolution exacte sur une grille de calcul

Résumé

L'optimisation combinatoire regroupe une large classe de problèmes ayant des applications dans de nombreux domaines de l'industrie. Ces problèmes ont souvent été abordés comme des problèmes mono-objectifs alors que la plupart d'entre eux sont de nature multi-objectif. De plus, ils ont toujours été limités par les ressources d'une seule machine : soit par la puissance de calcul disponible. La résolution de problèmes d'optimisation combinatoire de grande taille, tels que les problèmes d'ordonnancement, constitue un vrai défi pour les applications qui s'exécutent sur les grilles de calcul. En effet, il est nécessaire de repenser les algorithmes de résolution pour prendre en compte les caractéristiques de tels environnements. C'est dans cette optique que ce mémoire propose, à travers le problème d'ordonnancement de type flow-shop de permutation bi-objectif, une résolution par des méthodes exactes sur une grille de calcul. Cette résolution est optimisée par le traitement parallèle des étapes de l'algorithme d'une part et par le temps de traitement des calculs sur des ressources puissantes.

Mots clés : Optimisation multi-objectif, méthodes exactes, ordonnancement, flow-shop de permutation bi-objectif, grille de calcul, parallélisme.

Bi-Objective permutation Flow-Shop Scheduling problem An exact resolution on the grid computing

Abstract

Combinatorial optimisation encompasses a large set of problem with numerous industrial applications. These problems have often been considered as mono-criterion ones while the multi-criterion approach would have been more relevant, on account of the multi-criterion nature inherent to most of them. Solving a large size of combinatorial optimization problems, such as scheduling problems, constitutes a real challenge for applications running on the grid. Indeed, it is necessary to rethink the main algorithms to take into account the characteristics of such environments.

It is in this context, this paper proposes, through the bi-objective permutation flow-shop scheduling problem, a resolution by exact methods on the grid. This resolution is optimized by the parallelisation of the algorithm's steps on the one part and the execution time on powerful resources.

Keywords : Multicriteria optimisation, exact method, scheduling, bi-objectif permutation flow-shop, grid computing.