Année 2009

THÉSE

présentée à

L'U.F.R. DES SCIENCES ET TECHNIQUES
DE L'UNIVERSITÉ DE FRANCHE-COMTÉ

pour obtenir

LE GRADE DE DOCTEUR DE L'UNIVERSITÉ DE FRANCHE-COMTÉ

Spécialité : Informatique

Modélisation et dimensionnement d'une plate-forme hétérogène de services

par

Hala Sabbah

Soutenue le 23 mars 2009 devant la Commission d'Examen :

Rapporteurs	Jean-Marc PIERSON	Professeur, Université Paul Sabatier, Toulouse
	Yves ROBERT	Professeur, Ecole Normale Supérieure, Lyon
Examinateurs	Fabrice BOUQUET	Professeur, Université de Franche Comté, Besançon
	Emmanuel JEANNOT	Chargé de Recherche HDR à l'INRIA, Nancy
	Jean-Marc NICOD	Maître de conférences HDR, Université de Franche-Comté, Besançon
	Laurent PHILIPPE	Professeur, Université de Franche-Comté, Besançon

Remerciements

À notre directeur de thèse, Monsieur le professeur Laurent PHILIPPE :

Nous vous remercions de nous avoir fait découvrir l'approche de l'informatique, de nous avoir enseigné et fait apprécier l'informatique dans le domaine des micro-techniques.

Grâce à la qualité de votre encadrement, à vos qualités humaines, à votre esprit novateur et créatif et à votre soutien, nous avons pu mener à bien ce travail. De tout notre cœur, nous vous disons merci.

À Monsieur Jean-Marc Nicod, Maître de Conférences HDR et co-directeur de cette thèse :

Nous vous remercions pour votre soutien, votre aide et votre contribution efficace à cette thèse. Grâce à votre compétence, à votre sens critique, à vos conseils judicieux et à vos qualités d'écoute, nous avons pu mener à bien ce travail. Nous vous remercions très sincèrement ; vos critiques sur ce travail mais surtout vos conseils pour l'avenir nous sont particulièrement précieux. Nous vous exprimons toute notre reconnaissance et notre profond respect.

À Monsieur le professeur Jean-Marc PIERSON :

Nous vous remercions chaleureusement d'avoir accepté de juger ce travail de thèse. Vos suggestions pertinentes ont été déterminantes. Votre travail dans le domaine de l'informatique et votre état d'esprit sont pour nous une référence. Soyez assurer de notre profonde et respectueuse reconnaissance.

À Monsieur le professeur Yves ROBERT :

C'est un très grand honneur d'être jugée par une personnalité scientifique. Nous vous remercions vivement de l'intérêt que vous avez porté à notre sujet d'étude et des remarques constructives que vous avez faites. Vos compétences reconnues de tous dans le domaine de l'informatique nous ont servi de références. Nous sommes heureux et honorés de votre participation à notre jury de thèse. Recevez ici le témoignage de notre reconnaissance et de notre respect.

À Monsieur le professeur Fabrice BOUQUET :

Votre présence à ce jury nous fait vraiment très plaisir et nous honore. Nous vous sommes très reconnaissante d'avoir accepter d'examiner ce travail. Cette marque de confiance nous touche et nous honore. Que ce travail soit l'expression de notre profonde reconnaissance.

À Monsieur Emmanuel JEANNOT :

Nous vous sommes très reconnaissante d'avoir accepté d'examiner ce travail. Nous vous remercions très sincèrement. Vos critiques sur ce travail mais surtout vos conseils pour l'avenir nous sont particulièrement précieux.

À Abed El Razak KADMIRI, Directeur de l'UFR des Sciences et Techniques de l'Université de Franche-Comté pour votre accueil dans votre établissement.

À Monsieur le professeur Jean-Christophe LAPAYRE, Directeur du laboratoire LIFC, à Monsieur le professeur Jacques Juliand, pour leur autorisation et leur soutien durant mon absence de mon travail et pour leur accueil.

À Loris MARCHAL pour sa participation à des groupes de travail sur le sujet de cette thèse.

Aux membres de l'équipe CARTOON du LIFC : merci pour leur accueil et votre collaboration.

À mes collègues doctorants du laboratoire et plus particulièrement ceux de l'équipe dirigée par Laurent PHILIPPE, Sékou et Alexandru : merci pour leur aide et leur encouragement.

Mes remerciements s'adressent tout particulièrement à mon époux Khalil EL MOKDAD qui m'a initié à poursuivre mes études et s'est occupé de notre aimable Malak durant mon absence et dans des situations souvent difficiles.

À ma plus belle fille Malak avec toute mon affection et mes excuses de l'avoir quitté pour des périodes plus ou moins longues pour pouvoir voyager et mener à bien mon travail de recherche.

À ma sœur, Docteur Ibtissam SABBAH, qui m'a initié à la réflexion et au travail rigoureux, qui a suivi pas à pas la progression de ce travail, et a dépensé sans compter son temps pour m'aider à le réaliser.

À la mémoire de mon père.

À maman.

À mes sœurs : Sikna, Rajaa, Docteur Sanaa et Najwa.

À mes frères : Docteur Rémy Praticien Hospitalier à l'hôpital Jean Minjoz à Besançon et Abbass.

À mes beaux-frères : Hani, Mohamed et Docteur Nabil.

À mes belles sœurs : Milona et Fadia.

À mes nièces et à mes neveux.

Avec toute mon affection.

Résumé

Les plates-formes distribuées sont hétérogènes dans un grand nombre de domaines, comme le calcul hétérogène sur la grille ou les usines reconfigurables dans l'industrie. Il est primordial de contrôler et d'optimiser le coût économique de telles plates-formes face à des objectifs de performance. Dans le cas de la grille de calcul, il convient de réserver des processeurs à moindre coût afin de traiter des flux de calculs comme le traitement d'images médicales. Dans l'industrie, une telle plate-forme est par exemple une micro-usine modulaire. Le problème est alors de concevoir cette micro-usine en choisissant les fonctions permettant la fabrication de lots de produits de taille micrométrique à moindre coût, à un débit donné. Cette thèse est une étude de cas dont le but est le dimensionnement, à moindre coût, de telles plates-formes hétérogènes, sous des contraintes de performances imposées en entrée. Dans certains cas, que nous caractérisons, il existe des configurations optimales. Des algorithmes issus de la programmation dynamique sont proposés pour calculer ces configurations. Lorsque les solutions optimales ne peuvent pas être données à cause de leur caractère combinatoire, des solutions heuristiques sont proposées.

Mots clés : hétérogène, dimmensionnement, optimisation.

Abstract

Distributed platforms become heterogeneous in more and more domains, as heterogeneous computing onto grids or reconfigurable factories in the industry. It is mandatory to control and optimize the economic cost of such a platform regarding performance objectives. In the case of grid computing, processors have to be booked, at lower cost, to perform workflows as medical image processing, for instance. In the industry, such a plate-form could be, for example, a modular micro-factory. Now, the problem is to design a micro-factory by choosing functions that allow the making of batches of micro metric products, at lower cost, for a given throughput. This PhD Thesis is a case study which aims at designing, at lower cost, such heterogeneous plate-forms under input performance constraints. In some cases, which are characterized, the corresponding configuration is optimal. Algorithms, inspired by the dynamic programing paradigm, are proposed to design such a configuration. When no optimal solution exists because of the combinatorial nature of the problem, heuristic solutions are proposed.

Key words : heterogeneous, dimension, optimization.

Table des matières

Table des figures

Liste des tableaux

Introduction

La nécessité de tenir compte de l'hétérogénéité des plates-formes distribuées va croissante pour répondre aux contraintes d'utilisation de ressources existantes et d'adaptabilité à moindre coût. Cette hétérogénéité provient de différentes origines en fonction des caractéristiques du contexte. Des parallèles, voire des modèles semblables, peuvent s'appliquer, qui permettent de traiter certains problèmes de manière identique quelque soit l'environnement applicatif. C'est en particulier le cas pour le calcul hétérogène sur les grilles et pour les usines reconfigurables dans l'industrie de la production.

Les grilles de calcul :
Dans de nombreux domaines de recherche, les besoins en matière de puissance de calcul croissent plus rapidement que la puissance des machines disponibles. Paradoxalement, les ressources de calcul sont souvent, si on se place à un niveau global, sous exploitées. En effet, de la simple station de travail jusqu'à la machine parallèle de grande taille, il est rare qu'une machine soit utilisée à son potentiel maximum en permanence. De cette constatation vient l'idée de grille de calcul [76]. Les grilles de calcul sont constituées de nombreuses ressources informatiques hétérogènes (ordinateurs séquentiels, grappes de calcul, super-calculateurs...), géographiquement éloignées, mises en réseau [59]. Or la taille de ces plates-formes devient plus en plus grande et maintenir l'homogénéité de l'architecture quand la dimension de la plate-forme est étendue engendre un coût élevé. De plus, ces plates-formes sont la plupart du temps construites en utilisant des processeurs ou des machines existantes, ce qui génère une hétérogénéité de fait.

Le Grid Computing, ou grille informatique, est par ailleurs de plus en plus connu dans l'univers des nouvelles technologies et se fait aussi progressivement connaître du grand public à travers des expériences scientifiques et industrielles largement relayées par les médias. Cette reconnaissance engendre un besoin d'ouverture des grilles vers de nouvelles technologies des systèmes distribués et l'approche SOA (Service Oriented Architecture) est prometteuse car elle permet de prendre en charge la partie administrative de cette hétérogénéité et la spécialisation des ressources. Le principe est de reconnaître les applications, ou services, disponibles sur chaque ressource pour les rendre accessibles aux utilisateurs. L'utilisation des ressources peut alors se concevoir en terme d'utilisation de services. Celles-ci sont alors vues en terme de machines ou serveurs sachant traiter un ensemble défini de services, chacun avec une rapidité qui est propre à la machine.

La grille de calcul supporte le paradigme de programmation parallèle et distribuée. Un enjeu pour les applications alors est le déploiement sur toute une plate-forme hétérogène, car suivant l'allocation des tâches de l'application à telles ou telles ressources, les performances peuvent être très différentes. Ce problème difficile pour lequel il n'existe pas de solution calculable

dans un temps raisonnable, inclut également des contraintes sur la minimisation des transferts, l'optimisation du stockage des résultats intermédiaires, etc. La question qui est alors posée est "*quel ordonnancement donne de bonnes performances pour cette application ?*". Une manière détournée de répondre à cette question peut être de prendre le problème du point de vue de l'utilisateur qui demande un certain niveau de performance. Le problème devient alors un problème de conception de l'architecture utile aux exigences de l'utilisateur. La question devient alors "*quelle architecture pour quelle performance ?*" sur la base des services accessibles par cet utilisateur sur la grille.

Les micro-usines reconfigurables :
La miniaturisation des produits devient obligatoire pour proposer de nouveaux produits dans les domaines tels que la mécanique, l'électronique, l'électromécanique ou l'optique. De nouvelles applications pourraient être considérées comme l'intégration et l'assemblage d'éléments optiques de très petite dimension (10 ou 100 μm) pour réaliser des bancs optiques. Dans ce cas, la réduction des systèmes industriels conduit à de bonnes solutions en diminuant l'utilisation de l'espace, le prix et la consommation d'énergie. La flexibilité dans les lignes industrielles et la reconfiguration dans les usines seront ainsi élevées car la taille des stations de production est telle que les contraintes de placement géographique ne sont plus une problématique prioritaire. En outre, les machines peuvent être placées au sol de l'usine, aux bureaux du design ou dans les classes, et peuvent être aussi distribuées aux petits laboratoires de production, même dans les régions résidentielles [80, 106]. Cependant la manipulation à cet ordre de grandeur pose plusieurs problèmes : l'équilibre des forces physiques en présence n'est plus le même que celui du monde "macro" et les capacités d'intervention humaine sont limitées. À l'heure actuelle, ce type de production est principalement réalisé par téléopération ce qui en limite évidemment le développement [106, 42] et les capacités. L'automatisation est par conséquent nécessaire pour considérer une production à plus grande échelle. Elle requiert la collaboration entre les roboticiens, les producticiens et les informaticiens sur des aspects tels que la modélisation, l'optimisation des plates-formes, la simulation ou l'ordonnancement.

Avec les micro-usines l'approche de conception de l'usine change, que ce soit pour la production de petites, moyennes ou grandes séries. En effet, dans l'espace macroscopique, la maîtrise de l'environnement est telle que l'approche de conception usuellement appliquée consiste à concevoir les outils adaptés à une production spécifique. Ces outils seront d'autant plus spécifiques que la production est grande. Dans le cas des micro-usines, la maîtrise de l'environnement est encore à un stade où les difficultés technologiques imposent les choix en terme d'outil. Dans ce cas l'approche est plus ascendante, c'est à dire que la conception d'une production est guidée par les outils disponibles ou réalisables. L'approche actuellement proposée pour la conception des micro-usines suppose donc de disposer d'un ensemble de stations de micro-manipulation ou de cellules regroupant des stations. Chaque station proposant des fonctions, la conception se fait alors sur la base de la combinaison d'actionneurs existants plus qu'en développant des éléments ou robots ad-hoc. Le problème devient alors un problème de conception de l'architecture utile aux exigences de la production. Pour ce qui est de la capacité de production, la question devient alors la même question que pour les grilles : "*quelle architecture pour quelle performance ?*".

Les similitudes :
Plusieurs similitudes peuvent être dégagées entre ces deux types de plates-formes. L'hétérogénéité y est centrale. Elle provient du mode de constitution des plates-formes qui repose sur

l'agrégation de ressources choisies sur la base de leurs fonctionnalités. Le modèle de constitution des grilles repose sur une réservation des ressources sur la base de services de type SOA. Pour les micro-usines le modèle suppose la constitution sur la base d'un "magasin", d'une base existante de cellules ou de stations.

En terme d'utilisation des ressources, les approches sont également semblables dans la mesure où c'est le type de fonctionnalité de la ressource et les performances liées à cette fonctionnalité qui prime pour le choix de la ressource d'exécution ou de production. Concevoir une plate-forme en accédant à des services dans un environnement SOA sur les grilles est très semblable à la conception d'une micro-usine afin de choisir les cellules offrant les fonctionnalités nécessaires à une production.

Pour finir le modèle peut être défini d'une manière identique dans les deux cas. Il s'agit de ressources, sélectionnées sur la base de leur fonctionnalités, interconnectées par un réseau et que l'on cherche à utiliser au mieux. L'objectif commun est la performance d'exécution ou de production. Si nous ajoutons à cela le fait que l'accès à un service de la grille peut s'avérer payant et que l'ajout d'une cellule dans une micro-usine a aussi un coût économique, qu'il recouvre son achat ou son utilisation, nous avons alors un critère d'optimisation qui est commun au deux problèmes.

Le contexte :
Le problème qui se dégage de ce que nous venons de décrire rapidement, et qui est abordé dans cette thèse, concerne donc la définition de plates-formes dans le but d'optimiser une production ou des calculs en fonction du coût des ressources composant la plate-forme. Nous allons maintenant mieux cerner notre sujet en regardant les cas d'application possibles.

Si nous voulons proposer une optimisation du problème, il faut en connaître les paramètres. Ceci est le cas pour l'utilisation des ressources : elles s'entendent dédiées aux applications ou productions que nous traitons. En effet, le traitement de plusieurs calculs en parallèle sur une même machine, par exemple, rend inutile toute volonté de choisir telle où telle machine puisque nous ne pourrions savoir quelle est la part qui nous est allouée et donc les performances d'exécution du service choisi sur cette machine. Le problème se pose moins dans les usines où il est moins facilement admis qu'une station de production soit utilisée en même temps pour plusieurs productions. Nous travaillons donc sur des ressources qui seront entièrement disponibles au traitement de nos travaux.

En ce qui concerne les applications, il nous paraît également utile de fixer le cadre de notre travail. Tout d'abord s'intéresser à des applications composées d'une seule tâche ou à des produits réalisés en une seule opération a un intérêt limité : il suffit pour cela de trouver les serveurs ou les cellules qui ont le meilleur ratio coût performance en fonction du nombre de traitements ou de produits. La portée de ce résultat ne concerne d'ailleurs qu'un petit nombres des applications de grille ou de production dans les usines reconfigurables. La réponse à cette question est, de plus, implicitement apportée par le traitement de la question plus générale qui concerne le cas d'applications composées de plusieurs tâches. Notre but est donc d'exécuter des travaux composés de plusieurs tâches, organisées en graphe, sur des ressources hétérogènes.

Si nous nous intéressons maintenant aux caractéristiques des applications visées, nous constatons que définir une architecture pour un ensemble d'applications ou de produits différents (caractérisés par des graphes différents) ne présente que peu d'intérêt en terme de définition d'une

architecture pour deux raisons. Tout d'abord chaque ensemble devra être traité indépendamment, deux ensembles successifs ne seront pas forcément identiques ; le calcul et la configuration d'une architecture devra être reproduit à chaque fois. Or ce calcul a un coût non nul et le reproduire à chaque nouvel ensemble signifie dégrader la performance d'exécution. Ensuite, il est sûrement plus adapté de traiter ce type de cas en prenant l'approche de définir son ordonnancement sur une plate-forme existante à l'aide d'une heuristique adaptée plutôt que de définir une architecture utilisée une seule fois. Nous nous intéressons donc à des lots de travaux qui, par leur répétition, justifient le besoin de définir une plate-forme adaptée à leur exécution.

Pour les mêmes raisons, cette problématique ne présente que peu d'intérêt si la taille de la production ou le nombre de calculs à réaliser est trop faible, donc les lots avec un petit nombre de travaux. Dans ce cas, il vaut également mieux s'intéresser à la manière d'utiliser des ressources existantes pour réaliser les travaux demandés à cause du coût de mise en place d'une plate-forme. Ceci exclut donc de notre étude les cas où la production est faible. Le cas d'une production continue, en régime permanent, ou pour des lots de grande taille, permet de mettre en évidence le besoin d'une plate-forme adaptée.

Performance

D'une manière classique, la performance est la vitesse d'exécution d'un système. Elle est caractérisée par des grandeurs appelées métriques. Pour une application donnée, la métrique usuelle est le temps d'exécution en fonction de la taille du problème – la taille des données en entrée de l'application. Pour les plates-formes, les métriques usuelles sont le débit et la latence. Le débit représente la quantité d'information traitée par unité de temps ; la latence est le temps minimal nécessaire pour réaliser le premier travail.

Nous cherchons ici à définir une plate-forme à même de supporter à moindre coût économique la production ou l'exécution d'un produit ou d'une application. Dans le contexte que nous venons de définir, nous nous sommes plus particulièrement intéressés à l'optimisation du coût de la plate-forme permettant d'atteindre un débit fixé à l'avance. Nous tentons donc de répondre à la question : "*quelle plate-forme, donc quel coût, pour quel débit d'exécution ou de production ?*".

Le choix d'une approche centrée sur le débit nous permet de nous rapprocher des travaux liés au régime permanent. L'ordonnancement en régime permanent permet de relaxer les problèmes d'ordonnancement du fait qu'il s'intéresse au critères de débit plutôt qu'au makespan [11]. Il suppose ici de négliger les phases d'initialisation et de terminaison inhérentes à tout ordonnancement fini.

Organisation de ce document

Ce document s'organise en deux parties. La première partie présente un état de l'art sous forme de trois chapitres : une présentation des plates-formes qu'elles soient des grilles de calcul ou bien des micro-usines. Leur domaine d'application, leurs contributions techniques et leurs utilisations dans différents environnements sont présentées dans le chapitre 1. Une introduction aux problèmes d'optimisation, leurs complexités et leurs méthodes de résolution est présentée dans le chapitre 2. Puis les problèmes d'ordonnancement sont présentés dans le chapitre 3. Selon la diversité des problèmes d'ordonnancement, nous avons classés ces problèmes selon trois champs $\alpha/\beta/\gamma$. Les différentes techniques d'ordonnancement sont aussi présentées dans le chapitre 3.

La deuxième partie présente le contexte de notre étude sous forme de deux chapitres : une formalisation et une classification des différents cas de notre problème sont présentées dans le chapitre 4. Nous apportons des éléments de réponse, voire des pistes de résolution, pour chacun des cas identifiés. Les résolutions algorithmiques que nous avons effectuées et les expérimentations numériques pour les différents cas sont présentées dans le chapitre 5.

Dans le dernier chapitre nous tirons les conclusions et les perspectives de notre étude.

Chapitre 1

Plates-formes

Dans de nombreux domaines de recherche, les besoins en terme de puissance de calcul ont toujours été un défi auquel la communauté informatique s'est confrontée. Même si les évolutions technologiques de ces dernières années ont permis d'aboutir à la création de machines de plus en plus puissantes, celles-ci ne fournissent pas suffisamment de puissance pour effectuer des calculs d'une complexité trop élevée, ou traitant un trop grand nombre de données. Ainsi, aussi bien dans le monde universitaire que dans le monde industriel, de nombreux travaux sont ralentis par l'impossibilité de traiter les données expérimentales ou d'effectuer des simulations nécessaires. Cependant l'apparition de nouvelles méthodes automatisées de traitement des données expérimentales permet d'envisager l'analyse de masses de données jusque là impossible à manipuler. Cette analyse conduit souvent à une campagne d'expériences de précision supérieure. Paradoxalement, les ressources de calcul sont souvent, si on se place au niveau global, sous exploitées [76].

Le calcul parallèle, ou distribué, est une réponse à ce problème. Il s'agit de répartir les calculs sur un ensemble de machines, reliées entre elles par des réseaux rapides, afin d'augmenter leurs performances d'exécution. L'avancement rapide de la technologie de la communication a changé le paysage de calcul. De nouveaux modèles de calcul, tels que le calcul à la demande, les services web , les réseaux pair-à-pair, et les grilles de calcul ont émergé pour exploiter le calcul distribué, les ressources du réseau et fournir des services puissants [58].

Le terme "grille" de calcul est le terme générique employé aujourd'hui pour désigner une plate-forme de ressources distribuées et hétérogènes dédiées aux calculs ou à la gestion des données. Cet aspect est développé dans le paragraphe 1 de ce chapitre. Historiquement le terme anglais *grid* désignait le réseau ou le maillage électrique (*grid power*). Ainsi, par analogie, le terme *"grid"* a été repris dans *grid-computing* pour traduire le fait que, pour un utilisateur final, il serait aussi simple et transparent d'obtenir de la puissance de calcul grâce à une prise réseau qu'il est possible d'obtenir de la puissance électrique avec une prise électrique. En effet, on ne se pose pas la question de savoir comment ni où est produit l'électricité lorsqu'on en a besoin. En théorie, utiliser la grille de calcul devrait s'opérer en totale transparence pour l'utilisateur : le partage, la gestion des ressources, comment, pourquoi et où se fait l'affectation d'un travail demandé sont des questions que l'utilisateur ne doit pas avoir à se poser.

Nous utilisons par la suite le terme grille pour désigner une plate-forme de ressources hétérogènes et distribuées, contrairement à la définition des clusters rappelée dans [47] où tous les nœuds sont censés être homogènes. Nous considérons ici les clusters comme des grappes de machines et les grilles désignent un agrégat de ressources potentiellement hétérogènes reliées par un réseau.

L'utilisation des ressources distribuées et homogènes n'est pas facile et l'hétérogénéité ajoute des problèmes supplémentaires. La gestion des ressources d'une grille concerne leur accès et le partage de leur capacité entre les différents utilisateurs. Des outils ont été développés afin d'améliorer la qualité de la gestion tout en la simplifiant. Choisis en fonction de la nature de la grille considérée, ils peuvent assurer de simples services de tests de présence jusqu'à proposer des environnements de travail complets et performants [28].

Dans un autre domaine, les industriels cherchent à augmenter la productivité de leurs usines tout en minimisant les coûts afférents (mise en œuvre, fabrication, utilisation des ressources, stockage, etc.) ou bien la taille de la production. Nous l'avons vu dans l'introduction, les micro-usines permettent de réaliser des produits micrométriques. Actuellement, leur développement en est encore au stade de la recherche mais ces contraintes doivent être considérées pour construire une micro-usine qui réponde aux demandes des industriels. Rappelons qu'une micro-usine est définie comme un ensemble des cellules ou des machines servant à la fabrication des micro-produits. La conception d'une micro-usine est proche de la conception de la grille de calcul et on peut la considérer comme un cas particulier de la grille de calcul. Pour cela, nous allons présenter dans ce chapitre deux plates-formes d'études : la grille de calcul et la micro-usine.

Ce chapitre est organisé comme suit : dans le paragraphe 1 nous introduisons le concept, les caractéristiques, la fonctionnalité et la problématique de la grille de calcul. Le paragraphe 2 présente la qualité de service de la grille en terme de disponibilité des ressources, de traitement parallèle et de contrôle non centralisé. Les différents modèles économiques pour la grille avec leurs caractéristiques et leurs objectifs sont présentés dans le paragraphe 3. Dans le paragraphe 4 nous présentons quelques outils et environnements de la grille (SETI@home, Condor, Globus, Diet, etc.). Dans le paragraphe 5 nous présentons l'utilité de l'allocation des tâches sur la grille. Dans le paragraphe 6 nous définissons la micro-usine en distinguant deux types d'industrie : l'industrie de fabrication et l'industrie de procédés. La spécificité de la micro-usine, la considération sur la taille, l'automatisation et le domaine d'application sont des notions décrites dans le paragraphe 6.1. Le contexte général de la thèse est défini dans le paragraphe 7.

1 La grille de calcul

1.1 Définition

Le terme "grille" a été inventé dans le milieu des années 90 pour décrire une infrastructure distribuée et une nouvelle façon d'envisager le calcul parallèle à grande échelle pour les sciences avancées [53]. Différentes définitions existent pour expliquer ce qu'est la grille informatique :

Définition 1 : Selon Foster [51], une grille de calcul est *une infrastructure matérielle et logicielle qui fournit un accès consistant et peu onéreux à des ressources informatiques.*

Définition 2 : Une grille informatique ou "grid" est *une infrastructure virtuelle constituée d'un ensemble de ressources informatiques potentiellement partagées, distribuées, hétérogènes, délocalisées et autonomes* [112].

Définition 3 : Une grille est *une infrastructure, c'est-à-dire des équipements techniques d'ordre matériel et logiciel. Cette infrastructure est qualifiée de virtuelle car les relations entre les entités qui la composent n'existent pas sur le plan matériel mais d'un point de vue logique* [112].

Définition 4 : Selon Serugendo [98], une grille de calcul est *un environnement persistant qui permet à des applications logicielles d'intégrer des instruments, des affichages, des données, des ressources de calcul et de stockages, gérées par différentes organisations dispersées à travers le monde.*

Définition 5 : Une grille de calcul est *un dispositif logiciel qui offre aux utilisateurs des puissances quasi illimitées de calcul ou de stockage de données, grâce à un accès transparent et facile à un vaste ensemble de ressources informatiques distribuées sur une grande échelle* [65].

Il est généralement reconnu que les domaines d'application des grilles de calcul se divisent essentiellement en cinq grandes classes [60] :

- Le calcul intensif distribué, où l'on utilise les grilles de calcul pour additionner les capacités de plusieurs machines afin de résoudre un problème donné. Ce peut être l'agrégation de plusieurs supercalculateurs ou simplement de stations de travail. Les applications typiques sont la simulation, que ce soit de situations complexes (manœuvres militaires par exemple) ou du déroulement précis de processus physiques ;
- Le calcul à haut débit où l'on utilise les grilles pour ordonnancer un grand nombre de tâches, peu ou pas couplées, sur des machines inutilisées ;
- Le calcul à la demande, qui permet une utilisation temporaire de ressources dont la possession permanente ne serait pas rentable : capacités de calcul, logiciels, bases de données, etc.. Ici, c'est le rapport qualité-prix qui est l'argument principal, plutôt que les performances absolues ;
- Le traitement intensif des données, où l'on produit de nouvelles informations à partir de données géographiquement distribuées ;
- Le calcul collaboratif, qui privilégie la mise en place d'interactions entre personnes humaines pour l'exploration conjointe de bases de données, des systèmes de réalité virtuelle à objectifs éducatifs ou de distraction.

Le but du concept de la grille est de permettre le support d'une nouvelle génération d'applications qui couvrent un grand nombre de disciplines et organisations, en fournissant des solutions plus efficaces à des problèmes scientifiques importants, d'ingénieries, des affaires et des problèmes du gouvernement. Ces applications comprennent essentiellement des simulations scientifiques et d'ingénierie des phénomènes physiques complexes qui combinent les calculs, les expériences, les observations, comme par exemple, la prédiction du climat, des tremblements de terre, la formation de galaxies, la circulation (flow) du pétrole souterrain. Une autre catégorie d'applications est aussi introduite afin de gérer, adapter et optimiser le contexte de notre vie, la gestion de crises, des applications médicales utilisant des capteurs in vivo et in vitro pour le suivi des patients. Enfin, dans le domaine des affaires, la grille est utilisée pour optimiser les profits en fonction du temps et du contexte [83].

D'après [112], les ressources de la grille sont qualifiées de :
- partagées si elles sont mises à la disposition des différents consommateurs de la grille et éventuellement pour différents usages applicatifs ;
- distribuées si elles sont situées dans des lieux géographiques différents ;
- hétérogènes si elles sont de toute nature différentes, par exemple par le système d'exploitation ou le système de gestion des fichiers ;
- coordonnées : les ressources sont organisées, connectées et gérées en fonction des besoins (objectifs) et des contraintes (environnements). Ces dispositions sont souvent assurées par un ou plusieurs agents, qu'ils soient :
  - centralisées ou réparties ;
  - non contrôlées (ou autonomes) : les ressources ne sont pas contrôlées par une unité commune. Contrairement à un cluster, les ressources sont hors de la portée d'un moniteur de contrôle ;
  - délocalisées : les ressources peuvent appartenir à plusieurs sites, organisations, réseaux et se situer à différents endroits géographiques.

Une grille peut être considérée comme une interconnexion dynamique et totalement hétérogène de réseaux locaux d'ordinateurs et de ressources de stockage affiliés à une ou plusieurs organisations indépendantes. Ainsi, dans [53], la définition des grilles se traduit plutôt sur la façon dont on peut la construire et quels composants, quels protocoles et quelles interfaces doivent être fournis. On y trouve une description de leurs rôles respectifs et de leurs interactions.

1.2 Caractéristiques des grilles

La notion de grille, si elle renvoie à l'image d'un agrégat de ressources, suggère souvent dans sa définition une idée sur le type des travaux et/ou le type de gestion qui va lui être appliqué. Ainsi, on parlera de :
- **Information Grid** pour une grille où les ressources de stockage et les échanges de données sont primordiaux dans l'efficacité qu'elle apporte à l'exécution des applications. La gestion de la distribution, la réplication et la fédération des données doivent entre autre être particulièrement bien conçues et optimisées. L'objectif ultime en terme de performance est de fournir la facilité et la rapidité d'accès aux données en donnant l'impression aux utilisateurs qu'ils manipulent des données locales ;
- **Desktop Grid** pour une grille fonctionnant selon l'approche dite technique de *pull*, c'est-à-dire que ce sont généralement les ressources de calcul qui demandent auprès des entités dont c'est la charge, le travail qu'elles doivent exécuter ;
- **Server Grid** pour une grille dont les ressources de calcul sont plus particulièrement mises en avant par rapport au réseau ou aux ressources de stockage. Ce type de grille est intéressant pour les applications gourmandes en temps de calcul et qui ne nécessitent pas ou peu de communications en terme de quantité de données à échanger. On classe généralement ce type d'applications sous l'appellation *Calcul Hautes Performances* (CHP). Dans cette approche, l'utilisateur contacte et demande aux ressources, selon l'avis de l'ordonnanceur, d'effectuer du travail (par exemple une opération algébrique ou la résolution d'un problème donné) : c'est ce que nous appelons la technique de *push*.

– **Global Grid** pour désigner une grille dont l'ensemble des ressources constitue un supercalculateur virtuel sur lequel l'utilisateur soumet ses applications utilisant des bibliothèques de fonctions spécifiquement conçues.

1.3 Fonctionnalités et problématiques

Les grilles de calcul à haute performance se généralisent et deviennent l'environnement de base de l'exécution de serveurs répartis ou d'applications parallèles de calcul scientifique [1, 59, 83]. Elles impliquent des centaines voir des milliers de processeurs reliés par un réseau de communication. Ces infrastructures fournissent une grande capacité de calcul et une haute extensibilité, permettant ainsi aux applications de répondre à une charge croissante (par exemple, le nombre de processeurs parallèles dans une application de calcul scientifique). Cependant, de par leur complexité, leur hétérogénéité, leur dispersion géographique et leur partage des applications concurrentes, l'exploitation de ces infrastructures présente de nombreux problèmes qui restent ouverts, parmi lesquels le déploiement des applications, c'est à dire la réservation des ressources nécessaires à l'exécution de l'application, l'installation si nécessaire du code et des logiciels sur les processeurs réservés, puis l'installation des composants exécutables de l'application sur ces processeurs (lancement des processus parallèles) [50].

Le schéma classique du déploiement d'une application sur une grille, par exemple sur l'infrastructure Grid'5000 [1], consiste en la réalisation des 4 opérations suivantes :

– **La réservation** : cette phase consiste à requérir les ressources nécessaires à l'exécution de l'application à déployer en spécifiant des critères tels que le nombre de processeurs requis, l'architecture matérielle requise pour l'exécution de l'application ;
– **L'installation** : cette phase a pour objectif l'installation de l'ensemble des logiciels nécessaires à l'exécution de l'application à déployer. Elle est nécessaire avant toute phase d'instanciation ;
– **L'instanciation** : il s'agit de la phase de lancement de l'application. Elle consiste à créer les composants exécutables de l'application et à les démarrer, comme par exemple les processus d'une application parallèle.
– **La configuration** : des opérations de configuration peuvent survenir à différentes étapes du déploiement d'une application comme, par exemple, la configuration d'un logiciel installé (sa version) ou la configuration des paramètres d'instanciation (nombre de processus dans une application parallèle).

Ce schéma de déploiement présente deux inconvénients majeurs : le déploiement est strict et statique.

Afin d'assurer la complète transparence de l'accès aux ressources de la grille, l'environnement doit aussi s'occuper des services comme :

– La gestion des ressources. Cela concerne l'enregistrement des ressources et de leurs capacités dans une base de données, la découverte ou le déploiement des services que pourront offrir les serveurs, l'ajout de nouveaux serveurs ou la migration de nouveaux services ;
– Le monitoring des ressources via des senseurs installés à différents endroits stratégiques de la plate-forme. Cela permet d'avoir des informations précises et pertinentes sur le comportement de chaque ressource (charge CPU, espace mémoire disponible, etc.) sans les altérer par la prise proprement dite des mesures ;

- La prédiction de performance. Elle découle généralement des données enregistrées par le monitoring et de l'identification des requêtes reçues ;
- L'ordonnancement et le placement des requêtes. Il s'agit du choix des ressources que l'environnement va mettre en œuvre pour un travail donné ;
- Le mécanisme de soumission. Il consiste entre autre en l'identification des problèmes demandés et la gestion des paramètres d'entrée/sortie pour une cohésion totale.

Le client peut avoir indirectement d'autres services. Ainsi, en fonction de la grille utilisée, l'environnement se doit de proposer des mécanismes de tolérance aux pannes : pour chacun des composants de l'environnement par leur réplication par exemple, mais cela peut vite devenir difficile à gérer ; pour chacun des services offerts grâce à des techniques *check points* incorporées dans les applications ou dans les messages, grâce à de la réplication de données ou encore avec de la réplication d'exécution.

De plus, certaines utilisations nécessitent un certain niveau de confidentialité pour l'accès aux données et aux logiciels. Des mécanismes de sécurité doivent donc être prévus. Le degré de sécurité peut varier en fonction du type d'environnement et du domaine sur lequel il est utilisé. Un environnement de résolution de problèmes doit gérer l'authentification de tous les acteurs du système afin que chacun puisse profiter de l'environnement fidèlement à ses droits : quelles applications accessibles, où et par qui. Ensuite, parce que tous les utilisateurs ne pourront bénéficier de tous les services ou de toutes les ressources, par exemple à cause de leur coût d'exploitation, des mécanismes doivent assurer les permissions accordées à chaque client par rapport aux ressources et aux applications, et à chaque application par rapport à chaque ressource. Enfin les données transférées entre les différentes composantes du système doivent être chiffrées afin d'éviter qu'elles ne soient interceptées ou altérées. Il est en revanche plus difficile de garantir, surtout avec les approches pair-à-pair sur les *desktops grid*, qu'il n'y ait pas espionnage des données directement sur les ressources de calcul. Les mécanismes mis en œuvre peuvent par exemple utiliser le protocole de communication Kerberos [1], fondé sur le chiffrement symétrique dont la sécurité est modulable, ou la bibliothèque OpenSSL [2] capable d'utiliser des chiffrements symétriques pour les messages échangés.

1.4 Caractéristiques du déploiement

Fontaine et al. [50] distinguent trois caractéristiques du déploiement de l'application sur une grille de calcul :

- Le déploiement strict. Un déploiement strict est un déploiement dont les paramètres de configuration représentent des contraintes fortes. Parmi les exemples de déploiement strict, un déploiement qui dépend d'une architecture matérielle donnée. Dans ce cas, tant qu'aucune ressource disponible dans une grille ne correspond à l'architecture requise, le déploiement de l'application est mis en attente. Un autre exemple de ce type de déploiment est un déploiement exigeant un nombre exact de processeurs pour l'application. Tant que le nombre de processeurs disponibles dans la grille n'est pas supérieur ou égal au nombre de processeurs demandés par le déploiement, le déploiement de l'application ne peut pas s'effectuer ;

1. `http://web.mit.edu/kerberos/www/`
2. `http://www.openssl.org/` et `http://wp.netscape.com/eng/ssl3/` pour le draft IETF

- Le déploiement statique. Un déploiement est dit statique s'il ne peut être effectué qu'une seule fois au cours de la vie d'une application. Ainsi, une fois l'application déployée, celle-ci ne peut être redéployée. Autrement dit, aucune de ses phases de déploiement ne peut être refaite. L'application ne peut donc pas être configurée ni réinstallée ;
- Le déploiement adaptatif. Un déploiement adaptatif est un déploiement qui n'est ni strict, ni statique. Un déploiement adaptatif est, en effet, un déploiement dynamique qui permet de redéployer une application au cours de son exécution pour adapter ses paramètres de configuration. Grâce à un déploiement adaptatif, il est possible de redéployer une application parallèle sur un nouveau type d'architecture matérielle, ou sur un nombre et un ensemble de processeurs différents.

2 Qualité de Service (QoS)

Comment supporter la qualité de Service (QoS) d'une plate-forme pour satisfaire les demandes des utilisateurs ? La qualité a une signification différente suivant les utilisateurs et suivant l'environnement dans lequel ils évoluent. En général, la qualité est un caractère non fonctionnel ayant différentes dimensions telles que la performance, le coût, la sécurité, la précision, etc., ou une combinaison de ces différents facteurs. Dans un environnement distribué comme la grille, la qualité de service revêt d'autres aspects de ceux visibles dans un environnement centralisé. Ces questions, introduites par Xian-He-Sun and Miang Wu [58], sont les suivantes :

1. **la variation de disponibilité de la ressource.** Cette variation peut être due au partage de la ressource, au fait que le système soit dynamique, que les logiciels soient présents ou non sur le matériel distant, et tout autre facteur échappant au contrôle d'un utilisateur. L'incertitude de disponibilité de la ressource a un grand impact sur qualité de l'application.
2. **le traitement parallèle.** Le traitement d'une application déployée à grande échelle est divisée en sous-tâches inter-dépendantes allouées à différentes ressources pour profiter au mieux du parallélisme intrinsèque de l'application. Or, les ressources étant hétérogènes sur la grille, les résultats en terme de temps de traitement peuvent être très variables en fonction des allocations et des disponibilités des ressources.
3. **le contrôle non centralisé** qui ne permet pas la maîtrise de tous les paramètres.

3 Modèles économiques pour la gestion des ressources des grilles

Une approche sur les modèles économiques du marché réel peut être employée dans le cadre du calcul sur grille. Ces modèles économiques constituent une métaphore pour la gestion des ressources et le placement des applications, permettant ainsi de réguler l'offre et la demande au niveau des ressources de la grille, tout en garantissant une certaine qualité de service aux clients.

3.1 Objectifs des modèles économiques

L'objectif des modèles économiques est de permettre à la fois aux producteurs (les propriétaires des ressources) et aux consommateurs (les utilisateurs des ressources) d'atteindre leurs objectifs. Ils doivent ainsi inciter les producteurs à partager leurs ressources, en leur fournissant une contrepartie, et également inciter les consommateurs à réfléchir aux compromis entre temps de calcul et coût dépendant de la qualité de service désirée.

Ainsi, une approche basée sur les modèles économiques permet de mettre en relation les propriétaires des ressources et leurs utilisateurs potentiels, et place le pouvoir de décision entre leurs mains.

3.2 Les différents modèles économiques existants

Il existe cinq principaux modèles économiques, empruntés à l'économie du marché réel, qui peuvent être appliqués à la gestion des ressources des grille de calcul [27] :

- **Le modèle de marché** : Dans ce modèle, les fournisseurs de ressources fixent un prix, et les utilisateurs sont facturés à ce prix, proportionnellement à la quantité de ressources consommées ;
- **Le modèle de négociation** : Ce modèle permet à l'utilisateur d'agir sur le prix des ressources qu'il désire consommer. Ici, producteurs et consommateurs ont leurs propres objectifs, et doivent négocier pour les atteindre ;
- **Le modèle de l'appel d'offres** : Le modèle de l'appel d'offres est inspiré des mécanismes mis en place dans le monde des affaires pour gouverner les échanges de biens et de services. Il permet aux clients de trouver le fournisseur de service approprié pour travailler sur une tâche donnée ;
- **Le modèle de la vente aux enchères** : Ce modèle économique permet d'avoir une négociation entre un fournisseur et plusieurs consommateurs potentiels, ce qui permet d'aboutir à l'établissement d'un prix et au choix d'un consommateur.
- **Le modèle de partage proportionnel des ressources** : Selon ce modèle, une ressource peut être partagée entre plusieurs utilisateurs. La part de ressource allouée à chaque utilisateur par rapport aux autres est alors proportionnelle à son offre par rapport à celle des autres utilisateurs.

Différentes ressources peuvent être facturées à leurs utilisateurs, telles que le temps processeur utilisé, la quantité de mémoire consommée, la quantité de stockage occupée, la bande passante du réseau requise, les signaux reçus et les changements de contexte, ou bien les logiciels et librairies nécessaires à l'application.

4 Quelques outils et environnements pour les grilles de calcul

Dans ce paragraphe, nous présentons plus en détail certaines solutions utilisées actuellement, en insistant sur les aspects et les mécanismes mis en jeu. Puisque chaque solution est plus particulièrement adaptée à un type de grillesnous donnons aussi les domaines applicatifs concernés.

4.1 Système d'exploitation distribué : OpenMosix

OpenMosix est une extension du noyau Linux, pour une grappe dont les machines hétérogènes reposent sur la version patchée du noyau (nous parlons alors de *single image*). Cette solution n'est donc adaptée que pour des machines dans une même organisation. OpenMosix se veut capable d'interfacer les ressources de la grappe si elles composent un super-calculateur.

Il n'y a pas d'interface client puisque l'utilisateur exécute ses programmes comme sur sa propre station de travail. En effet, toutes les extensions openMosix sont directement incorporées aux noyaux. Les applications bénéficient automatiquement, et de façon transparente, de l'ensemble des ressources de la grappe. Le système gère l'ordonnancement des tâches exécutées et est capable de migrer un processus d'un nœud à un autre pour faire de l'équilibrage de charge [28]. Il cherche généralement à optimiser l'allocation des ressources. La solution embarque une technique de découverte des ressources : les capacités d'un nouveau nœud peuvent ainsi commencer à être exploitées dès son ajout à la plate-forme. La solution est extensible linéairement sur plus d'une centaine de nœuds qui peuvent être des machines SMP (*Symetric Multi-Processors*).

4.2 Globus Toolkit

Le projet Globus [6, 52] est le projet phare sur la grille de calcul. Il est mené par plusieurs universités et laboratoires américains. Son but est de rassembler les technologies et les protocoles nécessaires à la création d'une grille de calcul dans une boîte à outils libre d'utilisation (*toolkit open-source*). Globus utilise un ordonnanceur nommé GRAM (*Globus Ressources Allocation Manager*) qui s'occupe de l'exécution d'une tâche à distance et de la gestion de son statut, de l'envoi de requêtes d'exécution à l'hôte distant, de la création d'un gestionnaire pour les travaux sur ce dernier, de la surveillance de l'exécution et de l'envoi de statut de fin d'exécution.

4.3 SETI@home

Le projet SETI@home [99] se présente sous la forme d'un client que nous installons sur la machine sous la forme d'un économiseur d'écran, qui récupère les données par internet et qui utilise le CPU lorsque le système est inactif. Son objectif est la mutualisation des ressources informatiques connectées dans un réseau privé ou sur Internet. L'ordonnanceur considère par hypothèse que des ressources envoient sur le réseau des demandes d'exécution des services et que les autres ressources décident chacune localement si elles prennent en charge ou non l'exécution du service.

4.4 UNICORE

L'idée du projet UNICORE [96] est de supporter les utilisateurs en cachant le système et le site spécifique et en les aidant pour développer des applications distribuées. Les applications distribuées dans UNICORE sont définies comme des applications multi-parties où les parties différentes peuvent s'exécuter sur des systèmes informatiques d'une manière asynchrone ou séquentiellement synchronisée. Un travail (job) UNICORE contient une application multi-partie augmentée par des informations au sujet des systèmes de la destination, les exigences de la

ressource et les dépendances entre les parties différentes. D'un point de vue structurel, un travail UNICORE est un objet récursif qui contient des groupes du travail et des tâches. Le groupe de travail contient à son tour des groupes de travail et des tâches.

4.5 Les environnements construits sur le modèle maître/esclave

Le modèle maître/esclave fonctionne sur le principe suivant : une machine maître dispose de multiples tâches de travail à effectuer. Quand un des esclaves n'a plus de travail à faire, il contacte le maître pour en recevoir. La technique utilisée ici est le *pull* : ce sont les esclaves qui demandent à fournir du travail.

Condor Le concept de DRM (*Distributed Resource Management*) fondé sur la notion de maître/esclave, est déjà présent depuis 1988 pour des ressources hétérogènes distribuées grâce au projet Condor [46, 28] développé à l'université de Wisconcin-Madison. Il consiste à l'assignation de tâches de calcul soumises au préalable et stockées dans une queue, sur des ordinateurs inscrits pour faire profiter de leur capacité de calcul après un certain temps d'inactivité. Le retour impromptu de l'utilisateur met un terme à l'exécution distante de la tâche. Pour éviter la perte de temps de calcul, des mécanismes de préemption fondés sur des points de reprise (*checkpointing*) ont été mis en place. Ensuite, un travail change généralement de machine pour poursuivre son exécution [28].

Un travail Condor s'exécute sur une machine distante en ayant l'illusion d'être sur la machine locale : certains appels sont détournés, principalement ceux en relation avec les entrées/sorties. Pour cela, s'il n'y a pas de système de fichiers accessible par l'ensemble des ressources — comme NFS (*Network File System*) — un travail Condor doit être compilé avec la bibliothèque C modifiée et livrée avec Condor [28].

Un environnement du type Condor se rapproche des plates-formes de calcul pair-à-pair dans lesquelles chaque machine peut assurer indifféremment le rôle de client ou le rôle de serveur. Ces plates-formes se caractérisent par leur nature fortement distribuée et leur forte orientation vers les performances à grande échelle [76].

XtremWeb XtremWeb est un effort français pour la mise en place de services génériques pour le calcul haut débit (calcul global ou *desktop grid*) [76, 54]. Il est développé au LRI à l'université Paris Sud[3]. Son objectif est de fournir la gestion sécurisée d'une plate-forme de calcul intensif dont les ressources sont très volatiles et peuvent à un instant donné être très nombreuses [28]. Il est cependant destiné à être utilisé à l'échelle mondiale sur le réseau Internet et non principalement sur une configuration de réseau local. L'ensemble des machines volontaires est accessible comme un unique parc de nœuds de calcul. Les communications entre ces nœuds n'étant pas prises en charge, le système est ici également limité aux calculs pouvant être décomposés en plusieurs tâches indépendantes de petite taille.

XtremWeb s'appuie sur un ensemble de composants distribués. Le système est constitué des éléments suivants [76] :

3. `http://www.xtremweb.net/`

- un serveur gérant un sous-ensemble du parc de machines participant au système. Il est responsable de l'ordonnancement des tâches sur le parc ;
- un méta-serveur sur lequel tourne le répartiteur. Ce répartiteur reçoit les requêtes de calcul et les répartit sur les différents serveurs ;
- les machines des utilisateurs. Celles-ci peuvent utiliser XtremWeb de deux façon différentes :

  1. en mode collaborateur, un environnement complet permettant l'écriture d'applications distribuées et leur soumission au système est installé ;
  2. en mode utilisateur, la machine est simplement exploitée par le système lorsqu'elle est libre. Une application issue d'un poste collaborateur peut alors être exécutée sur cette machine.

Le système assure d'autre part la sécurité, en particulier lors du chargement d'un code sur une nouvelle machine, les points de contrôle, et le passage des pares-feux. Pour offrir cette dernière fonctionnalité, XtremWeb utilise une connexion systématique du client au serveur pour tout type de messages [76].

L'ordonnancement consiste à placer les tâches sur les machines volontaires, lorsque celles-ci demandent du travail. Cela se passe en deux étapes : il faut tout d'abord sélectionner quelles tâches vont être exécutées, puis les placer. La sélection des tâches s'effectue selon une priorité donnée aux applications en attribuant des proportions de tâches à accomplir par application. Puis l'ordonnanceur sélectionne la première tâche capable d'y être exécutée [28].

XtremWeb propose des solutions aux deux problèmes [76] :
- la spécification des systèmes. Une application comme SETI@home fait qu'une personne désireuse de donner des cycles pour un calcul est obligé d'installer un logiciel propriétaire qui ne saura allouer de la puissance que pour ce type de calcul précis. En effet, si de nombreux calculs sont adaptés à ce nouveau modèle, la quantité de cycles de calculs disponibles au niveau mondial devrait être équitablement répartie ;
- la centralisation du système. c'est une limite de Condor qui le rend plus adapté à une utilisation à l'échelle d'un réseau local. L'architecture XtremWeb prévoit la répartition de la tâche d'allocation des ressources sur deux niveaux (répartiteur et ordonnanceur) afin de permettre une distribution des traitements.

Ce type d'architecture, à la fois générique et distribuée, est un pas vers la construction de véritables boites à outils pour la construction d'applications de calcul scientifique utilisant l'approche du calcul haut-débit [76].

4.6 Environnement construits sur le modèle client/agent/serveur : le gridRPC

Dans un souci de normalisation de l'API des Appels de Procédure à Distance (RPC pour *Remote Procedure Call*) utilisés dans les NES, l'Open Grid Forum (OGF, ex-GGF), une organisation créée en 1999 afin de promouvoir les travaux dans le domaine des grilles, a confié au groupe de travail `GridRPC_WG` le soin de définir des standards architecturaux et de soumissions dans le cadre du projet APME (*Application, Programming Model and Environments*). L'approche résultante, expliquée par Laurent Philippe dans [88] et fonctionnant sur le mode *push*, est appelée GridRPC.

Cette approche propose comme architecture le modèle de référence illustré dans la figure ?? tirée de [88, 28] et fonctionne de la façon suivante : dans la phase initialisation de l'environnement, chaque serveur déclare ses services auprès de l'agent (appelé aussi *registry* ou encore RMS pour *Resource Management System*). Lorsqu'un utilisateur requiert un service, il contacte l'agent pour passer sa requête au système. Celui-ci lui communique en retour un identifiant donnant accès à l'application recherchée. Grâce à cet identifiant, le client contacte le serveur qui lui retournera les résultats du service exécuté.

Cette approche définit aussi le comportement de certains appels de fonction et leur contrôle, l'existence des modes synchrone et asynchrone, c'est-à-dire la capacité de laisser des résultats sur un serveur, et leur typage sont toujours à l'étude.

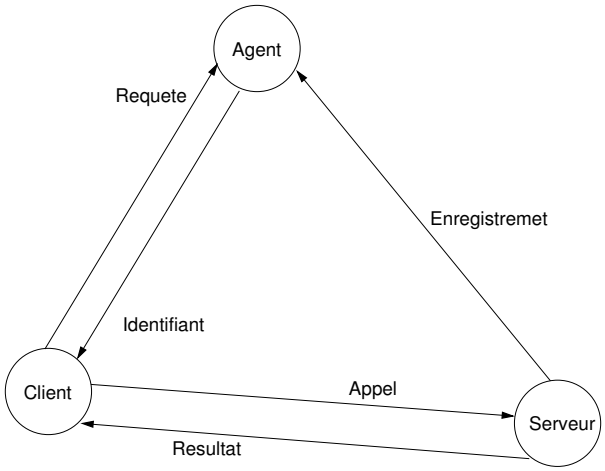

Figure 1.1 – Modèle de base GridRPC

NetSolve/GridSolve NetSolve/GridSolve [32] est un environnement *Server Grid* codé en C et développé à l'Université du Tennessee, Knoxville. Les membres de l'équipe de NetSolve ont participé activement à la formulation des standards pour l'interface GridRPC du groupe de travail du même nom, à l'OGF.

Dans l'approche NetSolve, aujourd'hui GridSolve, le déploiement de l'environnement et les mécanismes de soumission des requêtes correspondent à la norme GridRPC présentée adoptée par l'OGF.

DIET Le projet DIET (Distributed Interactive Engineering Toolbox) [108, 76, 28] développé initialement entre les laboratoires LIFC, LIP et LORIA dans le cadre du projet GASP [4], est largement distribué et maintenu aujourd'hui par le LIP à l'ENS de Lyon. L'équipe de DIET propose l'élaboration d'un standard pour le GridRPC pour la gestion des données (communication, persistance, etc.). L'objectif de DIET appartient à la catégorie des NES. DIET propose une architecture originale qui assure une bonne extensibilité à l'ensemble de l'environnement. Ainsi l'agent unique de NetSolve peut être remplacé par une hiérarchie d'agents afin d'en améliorer les performances en y ajoutant de surcroît une meilleure tolérance aux pannes. En effet, dans le cas de NetSolve, lorsque le nombre des requêtes envoyées à l'agent devient important, l'agent central

4. http ://graal.ens-lyon.fr/ gasp/

devient un goulet d'étranglement. Le but est donc ici de répartir la charge entre les différents agents. De plus, si la machine hébergeant l'agent de NetSolve tombe en panne, le système tombe à son tour. Dans DIET, les agents s'organisent alors pour le remplacer. On remarque ainsi que si DIET instancie une variante du modèle GridRPC, il existe aussi un prototype $DIET_{JXTA}$ [30] disposant d'une approche pair-à-pair que nous ne détaillons pas ici.

Les communications de l'environnement DIET reposent sur CORBA qui propose une interface de haut niveau pour la construction d'applications distribuées. DIET propose une approche hiérarchique pour l'ordonnancement des tâches. L'ordonnanceur de DIET choisit le serveur le plus rapide (et/ou le moins chargé) capable de résoudre le problème soumis par un client.

5 Grille et le point vue économique

La grille de calcul est quelque fois présentée comme un nouveau paradigme pour résoudre des problèmes de différentes natures telles que les problèmes scientifiques complexes, problèmes d'ingénierie ou problèmes commerciaux [25, 24, 114]. Les technologies de la grille permettent la création d'entreprises virtuelles qui partagent des ressources en associant logiquement un très grand nombre de ressources éparpillées à travers de multiples organisations géographiquement distribuées. Les utilisateurs, producteurs aussi appelés propriétaires de ressources et les consommateurs ont différents objectifs, stratégies et demandes (demand patterns). De manière générale, les utilisateurs et les ressources sont géographiquement distribués sur des fuseaux horaires différents. Pour gérer de tels environnements de grilles, les approches traditionnelles pour optimiser la performance des systèmes distribués ne peuvent pas être employés [26]. Les approches traditionnelles utilisent des politiques centralisées qui ont besoin d'informations de l'état complet du système et d'une politique commune pour la gestion des ressources partagées, ou des politiques décentralisées du concessus. Étant donnée la complexité de construction d'un tel environement de la grille, Buyya et al. proposent dans [26, 44] une approche économique pour gérer les ressources de la grille de calcul. Cela introduit plusieurs questions telles que l'autonomie des sites ou ressources, l'économie du calcul, la transparence de la grille, l'interaction hétérogène, le courtage de ressources, etc.

L'allocation efficace des tâches sur les grilles de calcul augmente la performance des calculs. Le système des grilles de calcul essaie de résoudre ces problèmes en allouant les travaux des utilisateurs sur les ressources oisives ou sur les ressources les moins chargées. Ces ressources ont des propriétaires différents. Ce faisant, les calculs peuvent être permis après un mécanisme de négociations automatisées par les contrôleurs de la grille. Cela peut être vu comme un marché régit par les lois de l'offre et de la demande appelé *Grid market*.

Un travail ou une application distribuée sur la grille exige habituellement un grand nombre de ressources appartenant éventuellement à plusieurs propriétaires/gestionnaires. Afin d'exploiter au mieux ces ressources, ces systèmes de grille de calcul doivent être capables d'assigner efficacement les travaux de plusieurs utilisateurs aux ressources de différents propriétaires pour profiter au mieux des ressources (par exemple inexploitées) [31]. Il s'agit alors d'équilibrer la charge des calculs afin d'améliorer la performance du système. Écrit autrement, ce problème peut être énoncé comme suit : soit un nombre de travaux de plusieurs grilles d'utilisateurs, trouver l'allocation de travaux aux ressources qui optimise une fonction objectif donnée (par

exemple, le coût total ou le temps de réalisation des travaux). Dans la plupart des travaux traitant le problème de l'équilibrage de charge, le coût des communications n'est pas pris en considération.

L'utilisation des mécanismes de marché pour l'allocation de la ressource dans les systèmes distribués n'est pas une idée nouvelle et a même été mise en avant par certains chercheurs (Shneidman, Ng et al. [100]).

Parfois, les applications conçues pour s'exécuter fréquemment sur les grilles de calcul exigent la co-allocation simultanée sur des ressources multiples pour satisfaire aux exigences de la performance. Par exemple, plusieurs ordinateurs et éléments du réseau peuvent être exigés pour accomplir la reconstruction de données expérimentales en temps réel, pendant qu'une grande simulation numérique peut exiger l'accès simultané à plusieurs clusters. Czajkowski et al. décrivent deux stratégies différentes pour négocier (dealing) avec le problème de la co-allocation [39]. La première approche est un méchanisme d'allocation atomique (*"Atomic Transaction Mechanisms and Strategy"*) qui recherche les ressources nécessaires à l'exécution de l'application. Si la requête échoue, une nouvelle requête peut alors être soumise. Cette stratégie ne donne pas de bons résultats en pratique étant donné son coté très statique. La deuxième approche est une stratégie itérative (*"Interactive Transaction Strategy and Mechanisms"*) qui répond aux inconvéniants de la première approche avec des requêtes d'allocation en deux temps, une phase d'allocation puis de modification de l'allocation en fonction des disponibilités des ressources.

5.1 Pourquoi utiliser des modèles économiques

D'après Furguson et al. [44], les modèles économiques fournissent plusieurs contributions intéressantes aux algorithmes de partage de ressources. La première est un ensemble d'outils pour limiter la complexité en décentralisant le contrôle des ressources. Une autre contribution est d'offrir un ensemble de modèles mathématiques qui décrivent la disponibilité des ressources qui partagent les bibilothèques résolvant ou contribuant à la résolution des mêmes problèmes. Dans une économie, la décentralisation est fournie par le fait que les modèles économiques consistent en agents qui égoïstement essaient d'accomplir leur but. Il y a deux types d'agents, les fournisseurs et les consommateurs. Une tentative du consommateur est d'optimiser ses critères de performance individuelle en obtenant les ressources qu'il exige, et ne s'inquiète pas des performances du système. Un fournisseur alloue ses ressources individuelles aux consommateurs. Le but d'un fournisseur est d'optimiser sa satisfaction individuelle (profit) dérivé de son choix d'allocations des ressources aux consommateurs.

La plupart des modèles économiques introduisent de l'argent et un coût (pricing) comme technique pour coordonner le comportement égoïste des agents. Chaque consommateur est doté d'argent qu'il utilise pour acheter les ressources exigées. Chaque producteur possède un ensemble de ressources qu'il fournit à des consommateurs. Le prix qu'un producteur impose pour une ressource est déterminé par la loi de l'offre et de la demande entre les agents. Le système du prix assure qu'une allocation réalisable de ressources est accomplie. Les critères de la performance du système sont déterminés par quelques combinaisons des critères de performance des agents individuels [44].

Penmasta et al. [86] proposent deux modèles de système pour gérer la grille suivant un modèle économique, en tenant compte cette fois des coûts de communication. Ces deux schémas

différents dans leur objectif, mais tous les deux sont considérés comme un problème de minimisation du coût. Le premier schéma essaie de minimiser le coût attendu par la communauté de la grille (i.e tous les utilisateurs) en prenant les travaux de tous les utilisateurs de la grille en considération. Cet aspect peut être bénéficiaire quand tous les utilisateurs de la grille appartiennent au même groupe social. Mais, la solution ne peut pas toujours être satisfaisante pour un utilisateur pris séparément des autres. Quelques utilisateurs peuvent être aménés à payer les ressources substantiellement plus chères que les autres. Le deuxième schéma, ou schéma non coopératif, essaie de fournir une solution utilisateur optimale pour fournir le meilleur prix à tous les utilisateurs de la grille. Ce schéma est formulé comme un jeu non coopératif parmi les utilisateurs de la grille qui essaient de minimiser indépendamment le prix attendu pour l'exécution de leurs propre travaux.

Une grille garantit des qualités de service non triviales, c'est-à-dire qu'elle se distingue des autres infrastructures dans son aptitude à répondre adéquatement à des exigences (accessibilité, disponibilité, fiabilité,etc) compte tenu de la puissance de calcul ou de stockage qu'elle peut fournir.

Nous voyons ici, que compte tenu des coûts liés aux serveurs de calcul, il existe différentes problématiques. L'une d'elles est le choix du ou des calculateurs nous permettant d'atteindre un niveau de performance acceptable par rapport à notre application sur la grille et par rapport à notre budget. Le problème est alors d'être capable de savoir quels serveurs nous devons réserver pour atteindre un niveau de performance donné à moindre coût au sens économique du terme.

Dans le paragraphe suivant, nous présentons un autre type de plate-forme dont certaines problématiques ne sont pas éloignées de celle de la grille, comme toutes autres plates-formes distribuées hétérogènes.

6 La Micro-Usine

De nos jours, les problèmes de gestion et d'optimisation de la production industrielle sont cruciaux — minimisation des coûts afférents à sa mise en œuvre, à la fabrication des produits, à l'utilisation des ressources, au stockage, au renouvellement des machines de production, etc. D'autre part, de nombreuses exigences métier, sanitaires ou commerciales se doivent d'être respectées. Pour cela, il est nécessaire de concevoir des outils d'optimisation et d'aide à la décision perfectionnés, adaptés aux difficultés que rencontrent les industriels vis-à-vis de l'organisation de leurs usines. Dans ce contexte, nous distinguons principalement deux catégories d'industries :

- **L'industrie de fabrication** : les usines comportent en général des chaînes d'assemblage de pièces, et visent à fabriquer des articles dont il est possible d'isoler une unité sans avoir à peser ou évaluer un volume (construction d'automobiles, de composants électroniques, d'équipements mécaniques).
- **L'industrie de process** : elle fait appel à des procédés de transformation pouvant être très complexes et délicats. L'élaboration des produits finis nécessite la manipulation de matières premières et des matériaux intermédiaires respectant des protocoles précis. Les industries chimiques, pharmaceutiques ou agro-alimentaires appartiennent à cette catégorie.

6.1 Spécificités de la micro-usine

Une micro-usine est une unité de production de petite taille pour fabriquer des produits de taille micrométrique (environ $50\mu m$). À cette échelle, les modèles macroscopiques ne peuvent plus être utilisés, tant au niveau des actionneurs qu'au niveau des objets manipulés. En effet, certaines forces négligées au niveau macro, deviennent prépondérantes au niveau micro. Ainsi, la micro-usine est une unité de production à inventer.

Une micro-usine se compose d'une ou plusieurs cellules de micro assemblage visant à opérer sur les micro-composants jusqu'à obtenir le produit final. La taille des objets manipulés ne permet plus une intervention humaine directe pour les assemblages à effectuer. L'automatisation d'une cellule est un problème ouvert pour de nombreuses opérations, ce qui explique que les cellules sont le plus souvent des cellules de micro-manipulation. Chacune de ces cellules permet de remplir une ou plusieurs fonctions unitaires. Ce faisant, une micro-usine est un regroupement de cellules de micro-assemblage coopérant pour réaliser une ou plusieurs productions, le tout sur une surface réduite (environ $1m^2$).

Une cellule est dotée d'un certain nombre de fonctions réalisées par des outils ou des groupes d'outils. Ainsi, après un niveau de configuration, une cellule offre un certain niveau de flexibilité.

Ce concept de micro-usine date des années 90 [48]. Il repose sur deux idées : d'une part réaliser une usine de très petite taille, qui peut tenir dans une valise [106], et d'autre part automatiser la production de pièces de taille micrométrique.

6.2 Les cellules de la micro-usine

Dans une recherche de flexibilité, une micro-usine s'oppose à la vision classique des usines du *macro*-monde par les dimensions et les fonctionnalités de ses robots. Les macro-robots sont complexes et capables de réaliser un grand nombre d'opérations. Les micro-robots sont plutôt de petite taille, centimétrique ou exceptionnellement millimétrique, et proposent des opérations élémentaires. Certains *macro*-robots ont un degré de précision suffisant pour manipuler des objets du monde *micro* mais leur précision n'est en aucun cas micrométrique. D'autre part, pour des raisons d'encombrement, de souplesse, de consommation énergique ou encore de temps de reconfiguration, ils ne sont pas les mieux adaptés au *micro*-monde.

Le choix de micro-robots à peu de degré de liberté et qui coopèrent permet d'atteindre l'objectif de taille, de précision et de réorganisation. Il implique une conception radicalement différente de l'usine en terme d'organisation et gestion de la production : chacun les micro-robots ne fournit qu'un ensemble limité d'opérations et il faut les regrouper pour fournir une fonction complexe. Ce regroupement est appelé cellule. Sa taille est petite, d'une dizaine de centimètres. La présence d'actionneurs indépendants permet, en fonction de la tâche à accomplir, une configuration à l'initialisation pour le calibrage de la cellule, et, dans une moindre mesure, une reconfiguration en intervenant dynamiquement pour changer la fonction réalisée. Ces modifications peuvent intervenir en actionnant/inhibant certains actionneurs ou en modifiant leur commande.

L'usine destinée à réaliser le produit final est alors constituée sur la base de cellules qui sont regroupées sur un même support. La taille de ce support est de l'ordre du mètre et l'organisation des cellules n'y est pas forcément linéaire comme dans la plupart des unités de production

puisqu'un même organe de transport, de type portique par exemple, peut satisfaire les besoins du transport entre les cellules. Nous avons donc une grande flexibilité dans la constitution de l'usine. La taille des pièces permet un stockage facile, ce qui permet de négliger ces déplacements dans le temps de production d'une séquence de micro-produits.

6.3 L'automatisation d'une production

L'automatisation de la production de pièces de taille micrométrique est un enjeu majeur de la micro-usine. Plusieurs facteurs agissent à ce niveau. Tout d'abord, il faut signaler que les forces en présence à cet ordre de grandeur ne permettent plus d'utiliser les modèles traditionnels de manipulation. Par exemple, la force de gravité n'est plus forcément la plus importante et certaines forces électrostatiques difficilement maîtrisables peuvent perturber la manipulation des pièces. De ce fait, la manière de les traiter doit évoluer pour faire place à une commande plus souple qui tient compte des risques accrus d'erreur. De plus, à cet ordre de grandeur, l'intervention humaine est difficile, voir impossible. Par exemple, il sera difficile de reprendre à la main des pièces qui ne peuvent être vue qu'avec un grossissement visuel (caméra ou microscope) et il n'est pas non plus possible d'impliquer directement un intervenant humain dans le processus de production. On privilégiera alors un traitement automatisé où les cas non gérés sont considérés comme des erreurs.

L'organisation est aussi un point intéressant dans l'étude de la gestion des micro-usines. En effet, le regroupement des actionneurs au sein de cellules permet d'identifier trois niveaux de contrôle pour l'automatisation. La commande des actionneurs est traitée de manière classique, c'est à dire des fonctions servent à l'asservir en positionnement et en temps. La coopération entre ces actionneurs doit aussi être traitée puisque la commande seule d'un actionneur n'est pas suffisante pour la réalisation d'une fonction sur le produit. Elle doit donc intégrer la synchronisation entre les actionneurs pour garantir l'exécution de la fonction. Elle est aussi soumise à de fortes contraintes temporelles et doit offrir des garanties temps-réel. À l'inverse, le niveau de coopération entre cellules n'est pas forcément synchronisé puisqu'il est possible de stocker les produits entre deux cellules : leur très petite taille permet d'envisager de stocker un grand nombre de produits sans difficulté. Dans ce cas, il n'y a plus de contraintes temps-réel et l'ordonnancement de la production entre les cellules peut-être considérée du point du vue du flux plutôt que du contrôle.

6.4 Les applications

Okazaki et al [80], Tanaka et al [106] ont présenté un concept de micro-usine, avec quelques prototypes d'outils de la machine extrêmement miniaturisés (i.e lathe, a milling machine and a pressing machine) une tour, une machine du métier de meunier et une machine pressante. Ces machines ont été développées et intégrées dans une micro-usine d'usinage portatif offrant la possibilité d'économiser de l'espace, de l'énergie, et d'avoir une haute performance d'usinage. Le dessin et le contrôle d'un micro-robot mobile omni-directionnel sont décrits pour un micro-montage dans une usine microscopique.

Zhenbo Li et al. [74] ont présenté des robots mobiles omni-directionnels qui ont été utilisés pour une variété large de tâches, dans les environnements dynamiques congestionnés et étroits,

tel que les centrales nucléaires, l'exploration dans l'espace, les environnements hasardeux et les usines. Ces micro-robots peuvent se déplacer rapidement ou pas à pas et satisfaire aux exigences d'applications différentes. Avec une méthode de contrôle spéciale, les micro-robots peuvent atteindre une plus haute précision.

6.5 Problématiques

L'usine est conçue pour la réalisation d'un lot de micro-produits. Un produit est réalisé à partir de composants auxquels sont appliqués des fonctions, par exemple un assemblage. L'application d'une fonction à un composant se fait sous la forme d'une tâche. L'ensemble des tâches d'un produit et leurs dépendances définissent un graphe de type DAG (Direct Acyclic Graph), appelé procédé. Nous le limitons à un *intree* (graphe sans fourche) puisqu'il n'est pas possible de dupliquer un composant. Différents procédés peuvent engendrer le même produit. Ces procédés sont quelque fois appelés "gammes" et décrivent bien la décomposition en plusieurs actions [94].

Ce contexte nous conduit à identifier différentes problématiques d'optimisation. Parmi celles que nous avons identifiées il y a la commande multiple des actionneurs. Par exemple lorsqu'un déplacement est basé sur deux actionneurs, l'un permettant de grands déplacements et l'autre de plus petits déplacements en fonction de la précision. Dans ce cas, quelle part attribuer à chacun ? Un autre type d'optimisation à effectuer est le choix du paramétrage des cellules pour obtenir un flux optimal de production, la réalisation optimale d'une fonction dans une cellule sur la base des actionneurs présents ou l'intégration des commandes entre les niveaux.

Dans le cadre de cette thèse, nous ne nous intéressons pas aux problèmes posés au niveau de la cellule elle-même. Nous considérons la cellule comme une boîte noire permettant de remplir une ou plusieurs fonctions pré-définies. Dès lors, deux problèmes se posent pour la gestion d'une micro-usine : (1) comment piloter la micro-usine en terme d'ordonnencement pour permettre de réaliser une production le plus rapide possible ? et (2) quelle micro-usine construire pour satisfaire une production rythmée à une cadence donnée ? Comme déjà mentionné, le but de cette thèse est de donner des élements de réponse à la deuxième question.

7 Contexte

Dans ce paragraphe, nous établissons le contexte commun aux deux plates-formes présentées précédemment, la grille et la micro-usine. Les applications visées sont celles qui consistent en la production d'une série de résultats élaborés tous de la même manière par les ressources de la plate-forme. Nous parlons alors de débit de calcul ou de production, c'est à dire du nombre de résultats ou de produits mis en œuvre par unité de temps. Les applications sur la micro-usine conçernent tout type de production d'un nombre donné de produits identiques. Sur la grille, cela équivaut à des applications du type flux de données exécutant autant d'instances du même graphe de tâches que de résultats à produire.

Afin de ne pas introduire trop de confusion dans la lecture du document, nous parlons maintenant :

- de graphes de tâches pour tout travail multi-activité que ce soit un work-flow, un procédé, une gamme ;

- d'instances de ce graphe pour désigner un graphe de tâches parmi tous les autres graphes identiques en cours de traitement ou à traiter sur la plate-forme ;
- de résultats pour toute sortie de la plate-forme pour chaque instance d'un graphe de taches ;
- de débit, c'est-à-dire du nombre de résultats en sortie par unité de temps ;
- de ressources pour qualifier les unités de traitement de la plate-forme, cellules ou processeurs.

Un graphe de tâches Un graphe de tâches décrit l'ensemble des tâches à réaliser et l'ordre des précédences à respecter pour qu'un résultat soit totalement traité par la plate-forme. Un graphe de tâches $G = (T, D)$ est défini par l'ensemble des tâches T (sommets du graphe) et par l'ensemble des relations de précédence entre les tâches D (arêtes du graphe).

Une tâche est définit par son type (T_i).

L'ensemble des tâches T est défini par I tâches, $\{T_1, \ldots, T_I\}$.

Rappelons qu'un résultat (calcul complet, produit fini) peut être produit par un ou plusieurs graphes de tâches.

Un résultat Un résultat est produit par la plate-forme en respectant un graphe de tâches et la précédence entre ces tâches. Ce résultat peut être obtenu en exécutant indifféremment J graphes différents. Dans cette thèse nous traitons les deux cas de résultats définis à l'aide d'un seul graphe, ou à l'aide de plusieurs graphes.

Les ressources L'exécution des activités de production est supportée par des ressources appartenant à la plate-forme. Dans cette thèse, nous considérons le cas de ressources spécialisées ou multi-spécialisées. Une ressource spécialisée ne sait sait réaliser qu'un seul type de traitement. Une réssource multi-spécialisée peut réaliser tout ou partie de traitement possible. Cela a une réalité à la fois dans la grille avec l'existance ou non de telles ou telles bibliothèques ou de telles ou telles architectures matérielles sur un serveur de calcul et à la fois dans la micro-usine avec l'existence de tels ou tels outils dans une cellule.

Une ressource est définie par le ou les types de tâches qu'elle est succeptible de savoir traiter, par une capacité et par un coût économique.

Les communications Les communications sont définies habituellement par le temps nécessaire pour acheminer un résultat intermédiaire d'une ressource à la suivante dans le respect du graphe de tâches. Cette durée dépendant des conditions dans lesquelles se fait cette communication, de la taille du transfert, de la distance, de l'occupation courante des liens de communication, etc. Étant donné le type d'application visée, il est raisonnable de considérer dans une première approximation que les temps de communication sont recouverts par les temps de traitement. Dans le cas de la grille, cela signifie que les objets manipulés nécessitent un ensemble de traitements suffisamment longs pour que les liens de communications existants ne soient pas le facteur limitant de l'application de type pipeline ou anti-arborescente. Dans le cas de la micro-usine, cela signifie que, étant donné la taille des composants manipulés, la taille des buffers en entrée

et en sortie des ressources est potentiellement infinie, et que de plus, les déplacements de ces composants peuvent se faire par lots très importants. Par conséquent, là encore, il est possible de négliger les communications.

Le débit Le débit est le nombre de résultats en sortie de la plate-forme par unité de temps. C'est l'objectif que doit respecter la plate-forme que l'on cherche à imaginer. Ce débit est donc fonction des performances des ressources de la plate-forme, c'est à dire des coûts en terme temps de traitement des ressources.

Coût d'une plate-forme Une plate-forme est totalement définie par les ressources qui la composent. Comme chaque ressource a un coût économique, une plate-forme possède également un coût économique. C'est de ce coût là que nous parlons au niveau d'une plate-forme.

Dans le cadre de nos travaux, ce coût n'est pas ramené à chaque résultat sortant de la plate-forme mais est un coût initial permettant d'atteindre un certain débit de par la configuration de la plat-forme, ou plus exactement du choix des ressources formant la plate-forme. Ce coût est l'objectif central de ce travail que nous cherchons à minimiser en fonction d'un débit à atteindre, du type de résultats à traiter et des ressources disponibles. La difficulté est par conséquent de bien choisir les ressources pour atteindre les objectifs de rendement que doit remplir la plate-forme.

7.1 Objectif

Compte tenu du contexte présenté ici, le but des travaux menés dans cette thèse est de proposer des solutions ou des pistes permettant de répondre à la question suivante en fonction de différents cas de figure : "*Quelle plate-forme pour quel débit ?*"

8 Synthèse

Nous avons présenté dans ce chapitre la genèse du concept des plates-formes (grille, micro-usine) et son utilitée face à des besoins de performances croissants. Nous l'avons définie comme l'agrégat de ressources hétérogènes de domaines administratifs différents. Nous avons classée les grilles en quatre grandes familles selon une caractéristique sur leur enjeux et leurs domaines d'applications respectifs. Nous avons proposé de regrouper les grilles sous les appellations : information grid, desktop grid, server grid, et global grid, et nous les avons brièvement décrites et commentées.

Dans le contexte des grilles de calcul, nous avons introduit l'approche des environnements de résolution de problèmes reposant sur le parallélisme de tâches des applications à soumettre. Après avoir présenté l'architecture globale des mécanismes mis en jeu, nous avons détaillé les fonctionnalités dont devait disposer un environnement de soumission de travail, qu'elles aient trait à la communication des données ou à la gestion de la plate-forme. Nous avons souligné certains problèmes (gestion des données, des ressources, tolérance aux pannes ou sécurité) que posait le déploiement d'un tel système sur un ensemble de ressources hétérogènes distribuées géographiquement et nous avons indiqué quelques solutions utilisées.

Par la suite, nous avons présenté les caractéristiques du déploiement de l'application sur la grille de calcul (déploiement strict, déploiement statique et déploiement adaptatif). Nous avons démontré que le déploiement adaptatif est le déploiement le plus flexible permettant de redéployer une application parallèle sur un nouveau type d'architecture matérielle ou sur un grand nombre ou un ensemble de processeurs différents.

Le terme "Qualité" a des significations différentes pour des utilisateurs différents sous environnements différents. D'une manière générale, nous avons défini la qualité de service sur une plate-forme comme un caractère non fonctionnel (performance, sécurité, coût, précision, etc.) ou une combinaison d'entre eux. Dans un environnement réseau partagé, comme la grille, nous avons défini la qualité de service en terme de disponibilité des ressources, de traitement parallèle dans le cadre d'un contrôle non centralisé.

Les modèles économiques empruntés au marché réel peuvent être appliqués à la gestion des ressources des grilles de calcul, en permettant à leurs propriétaires et à leurs utilisateurs de satisfaire leurs objectifs respectifs.

Nous avons sélectionné les outils présentés dans ce chapitre pour leurs caractéristiques et leur pertinence dans le domaine (originalité, référence, etc.). Nous avons présenté les modèles utilisés (mode *push* et *pull*, modèles maître-esclave et GridRPC) et leur fonctionnalité en établissant le parallèle entre les motivations du projet et les fonctionnalités théoriquement disponibles vues précédemment.

Dans le cadre d'utilisation efficace des différentes ressources de la grille, nous avons proposé une allocation des tâches sur les processeurs oisifs ou sur les processeurs les moins chargés.

Dans la partie micro-usine de ce chapitre, nous cherchons à construire une micro-usine qui réponde le mieux aux exigences des industriels. Pour cela il est nécessaire de concevoir des outils d'optimisation et d'aide à la décision perfectionnée, adaptés aux difficultés que rencontrent les industriels vis-à-vis de l'organisation de leurs usines (production, taille de l'usine, la taille de production, etc.).

La micro-usine que nous proposons dans cette étude repose sur deux idées : d'une part la réalisation d'une usine de très petite taille et d'autre part l'automatisation de la production de pièces de taille micrométrique. Pour identifier les niveaux de contrôle pour l'automatisation, nous proposons de regrouper les actionneurs au sein de cellules. Étant donné cette approche modulaire pour la construction d'une micro-usine, nous proposons le dimensionnement de cette unité de production en fonction d'un certain niveau de production. Cet ensemble modulaire formé d'un ensemble de cellules plus ou moins configurables ou reconfigurables, occupe une surface d'un ou deux mètres carré.

L'objectif de ce travail est de construire une micro-usine de petite taille tout en minimisant le coût économique total de traitement d'un ou plusieurs résultats pour un débit donné.

Enfin, nous avons montré les traits communs entre grille et micro-usine en proposant un langage commun. Ceci nous permet de proposer des solutions à la fois pour la grille pour minimiser le coût économique de la réservation de serveurs de calcul et pour la micro-usine pour l'achat de cellules permettant de garantir une certaine vitesse de production avec un coût matériel minimum. Ainsi, la micro-usine peut-elle être vue comme un cas particulier de la grille.

Dans la suite de ce travail, nous utilisons les termes génériques définis dans ce chapitre afin qu'il soit possible de les appliquer à l'un ou l'autre des deux domaines.

Le chapitre suivant est l'occasion de faire un survol des différentes techniques ou outils utilisés en optimisation.

Chapitre 2

Optimisation

1 Introduction

Parmi tous les problèmes rencontrés par l'ingénieur, le chercheur, le gestionnaire ou l'économiste, les problèmes d'optimisation et, en particulier, les problèmes discrets occupent une place de choix. Depuis une trentaine d'années, des méthodes très efficaces ont été proposées et sans cesse améliorées pour résoudre les problèmes d'optimisation linéaire ou quadratiqueconvexe comportant des variables entières. Par ailleurs, de nombreux logiciels solveurs mettant en œuvre ces méthodes sont actuellement disponibles (COIN-OR, CPLEX, OSL, Xpress-MP, etc.) ainsi que des langages de modélisation qui apportent une aide précieuse à l'utilisation de ces logiciels (AIMMS, AMPL, GAMS, Mosel, etc.). Toutefois, un grand nombre de problèmes d'optimisation discrets, pertinents d'un point de vue scientifique ou économique, mais difficiles à résoudre, ne se formulent pas directement de façon linéaire ou quadratique convexe.

Face à de tels problèmes, et compte tenu de la puissance croissante des ordinateurs, il est tentant d'essayer de les formuler sous la forme de problèmes linéaires ou quadratiques à variables entières. Dans certains cas, il est relativement facile d'obtenir ces formulations. C'est le cas des problèmes dont la nature est fondamentalement linéaire ou quadratique (convexe). Dans les autres cas, la modélisation est plus difficile à obtenir. Dans les deux cas, il reste la question délicate du choix d'une formulation, car il en existe souvent plusieurs. L'efficacité de la résolution dépend fortement de la formulation retenue.

Ce chapitre est organisé comme suit. Le paragraphe 2 présente les problèmes d'optimisation pour lesquels nous formalisons le problème d'optimisation discret et nous présentons quelques problèmes classiques d'optimisation. Dans le paragraphe 3, nous citons les différentes méthodes de résolution des problèmes d'optimisation que nous détaillons dans le paragraphe 8. Le paragraphe 4 présente une méthode de résolution d'un problème ne pouvant pas être résolu par une méthode exacte. Le paragraphe 5 définit le mot algorithme. Le paragraphe 6 définit la complexité d'un algorithme et les deux classes de complexité P et NP. Ainsi, le paragraphe 7 présente quelques problèmes qui sont connus comme des problèmes NP-Complets. Le paragraphe 9 décrit la simulation et l'optimisation. Nous concluons ce chapitre par une synthèse.

2 Les problèmes d'optimisation

Quiconque a jamais pris des décisions responsables de tout genre, a certainement rencontré des décisions discrètes : décisions parmi un ensemble fini d'alternatives mutuellement exclusives. Les choix rigides entre construire et ne pas construire, tourner à gauche ou tourner à droite, visiter la ville A au lieu de visiter la ville B, etc. Bien sûr, la discrétisation de l'espace de décision offre l'avantage d'avoir un aspect concret et les graphes élémentaires ou des illustrations semblables peuvent souvent, naturellement et intuitivement représenter la signification d'un choix particulier. La discrétisation alourdit la tâche de dimensionnement du problème. Si en plus, quelques choix sont faits, le décideur est face à une combinatoire regroupant toutes les possibilités, qui exige son évaluation. Il est impossible pour lui de prendre une décision dans cette situation étant donné le nombre exponentiel de cas.

Depuis le XVIII^e siècle, le paradoxe liant simplicité apparente du problème et complexité de la solution discrète existe dans de nombreux domaines comme la gestion et l'ingénierie. Ce faisant, de nombreux chercheurs se sont lancés dans l'étude de ces problèmes et la recherche de leurs solutions discrètes. L'intérêt a été multiplié avec le développement révolutionnaire des ordinateurs pendant la seconde moitié du XX^e siècle. Pour certains problèmes, des solutions élégantes ont été découvertes. Pour d'autres, des heuristiques et des algorithmes approchés ont été développés, et même si des progrès considérables ont été accomplis, aucune résolution complète de ces problèmes difficiles n'existe. De plus, certains problèmes apparemment très simples ont résisté à presque tous ces progrès [84].

2.1 Formalisation de problèmes

De manière générale, un problème consiste en une question à résoudre dans un contexte donné. En informatique, il est défini par un ensemble d'objets en entrée auxquels sont associées des variables, et un domaine d'arrivée soumis à des contraintes. Une solution est une instanciation de toutes les variables mises en jeu dans le problème. Elle est dite réalisable si elle satisfait l'ensemble des contraintes de ce problème. On note S_p l'ensemble des solutions réalisables du problème P, encore appelé espace des solutions de P.

Pour définir un problème d'optimisation, on se donne une troisième composante, la fonction objectif. Elle fournit une valeur, de sorte que la question à résoudre consiste à trouver une solution réalisable qui optimise cette fonction. Cette solution est appelée solution optimale (optimum notée f^*).

Un sous-ensemble de solutions réalisables est dit dominant pour un problème d'optimisation donné s'il contient nécessairement l'une des solutions optimales (s'il en existe au moins une).

On définit le problème de décision correspondant à un problème d'optimisation, comme le problème qui pour une valeur $\bar{f}$, pose la question de l'existence d'une solution réalisable de valeur inférieure à $\bar{f}$ si le problème est un problème d'optimisation. Un sous-ensemble de solution est dit dominant pour un problème de décision donné, s'il contient nécessairement l'une des solutions réalisables (s'il en existe au moins une).

Le problème P est dit continu si le domaine de définition de toutes ses variables est $\mathbb{R}$; il peut alors s'écrire de la manière suivante :

$$P \begin{cases} \min(f(x)) \\ s.c. \\ \qquad C(x) \geq 0 \\ \qquad x \in \mathbb{R}^n \end{cases}$$

dans lequel :
- x est le vecteur des variables réelles ou entières de dimension n ;
- C est l'ensemble des m fonctions de contraintes notées g_i
- f est la fonction objectif

Si, pour un problème d'optimisation continu, nous sommes confrontés à une contrainte d'égalité de type $g(x) = 0$, elle est intégrée au problème par le biais de deux inégalités : $g(x) \geq 0$ et $g(x) \leq 0$. Des bornes inférieures ou supérieures aux variables peuvent être interprétées comme des contraintes [40]. Un problème de maximisation peut être facilement obtenu en multipliant la fonction objectif par -1 : maximiser f revient à minimiser $-f$.

Dans la suite de ce chapitre, nous traitons que du problème de minimisation. Dans le cas d'un objectif unique un seul critère sera à minimiser, contrairement aux problèmes d'optimisation multi-critères, où il s'agit d'optimiser simultanément plusieurs objectifs. Notons enfin que les fonctions de contraintes et la fonction objectif seront toujours fixées au moment de la formalisation du problème et de manière déterministe.

Un problème d'optimisation combinatoire est un problème d'optimisation pour lequel une composante au moins de la solution recherchée est entière. D'un point de vue théorique, il est possible pour tout problème d'optimisation combinatoire, de définir un ensemble d'entités E fini (les arêtes d'un graphe, les cases d'un tableau, des sous-ensembles de sommets d'un graphe ayant des propriétés particulières, etc.), tel que toute solution réalisable S soit un sous-ensemble de E. Le vecteur variable x devient $x^S \in \{0,1\}^{|E|}$, un vecteur d'incidence de S vérifiant, pour tout $e \in E$: $x^S(e) = 1$ si $e \in S$, 0 sinon. Par exemple, pour le problème du voyageur de commerce, dans lequel on doit trouver un cycle hamiltonien minimal dans un graphe non orienté complet, une modélisation consiste à faire correspondre E à l'ensemble des arêtes. Le vecteur d'incidence représente alors les arêtes du cycle hamiltonien recherché.

L'ensemble des solutions réalisables S_p est donc inclus dans l'ensemble des sous-ensembles (ou parties) de E, ce qui en notation anglosaxonne est donné par $S \subset 2^E$.

Un problème d'optimisation combinatoire POC peut donc s'écrire de la manière suivante :

$$POC \begin{cases} minimiser \quad f(x) \\ s.c. \\ \qquad C(x) \geq 0 \\ \qquad x \in \{0,1\}^n \end{cases}$$

À moins de posséder des propriétés spécifiques, les problèmes d'optimisation combinatoire ont un espace des solutions exponentiel par rapport à la cardinalité de E. La recherche de la meilleure de solutions réalisables dans cet ensemble se révèle donc être très difficile dès que $\mid E \mid$ va prendre des valeurs assez grandes.

Dans ce qui sont, nous présentons des problèmes propres à l'optimisation discrète. Nous introduisons brièvement quelques uns de ces problèmes pour illustrer la grande diversité des modèles déjà étudiés dans la littérature. Si certains de ces problèmes ont été résolus, d'autres continuent à frustrer des chercheurs après deux siècles de recherches [84].

2.2 Problèmes classiques d'optimisation

Dans ce paragraphe, nous présentons et définissons quelques problèmes d'optimisation rencontrés dans la littérature pour illustrer la diversité énorme de modèles qui ont été étudiés [84]. Ces problèmes sont :

- **Problème du voyageur de commerce**. Étant donné un graphe (orienté ou non orienté) avec des poids spécifiés sur les arêtes, déterminer une traversée (un cycle) fermé qui inclut exactement une fois chaque sommet du graphe de façon à minimiser le poids total des arêtes.
- **Le problème de postier chinois (Chinese postman problem)**. Étant donné un graphe (orienté ou non orienté) avec des poids spécifiés sur les arêtes, déterminer une traversée (un cycle) fermée qui inclut au moins une fois chaque arête du graphe de façon à minimiser le poids total.
- **Le problème du sac à dos**. Étant donné un ensemble T de tâches, chacune avec un temps de traitement τ_j, $1 \leq j \leq \mid T \mid$, assigner chaque tâche à exactement une des m machines afin que le temps total de calcul de toutes les tâches soit minimisé.
- **Le chemin le plus court**. Pour un graphe donné (orienté ou non orienté) avec des poids ou des longueur spécifiés sur les arêtes, trouver le plus court chemin qui relie deux sommets donnés.
- **Le flux maximal**. Étant donné un graphe (orienté ou non orienté) avec des capacités spécifiées sur les arêtes (flux), trouver un flux maximal entre deux sommets spécifiques qui se conforment aux capacités des arêtes et ayant un flux total dans tous les autres sommets égal à la somme du flux sortant.
- **Le problème p-médian**. Étant donné un graphe (orienté ou non orienté) avec des poids spécifiés sur les arêtes, choisir un sommet p afin que la somme des distances de tous les sommets au sommet choisi p soit minimisé.
- **Le problème de la charge fixe**. Étant donné l'ensemble faisable S des activités non négatives ou niveaux de circulation, $x = (x_1, x_2, \ldots, x_n)$, ν_j le prix de revient pour employer x_j et les f_j des frais fixes répartis toutes les fois que le x_j est positif, choisir un coût total minimum avec $x \in S$.

3 Méthodes de résolution

Trouver une solution optimale dans un ensemble discret et fini est un problème facile à comprendre : il suffit d'essayer toutes les solutions et de comparer leurs qualités pour choisir la meilleure. Cependant, en pratique, l'énumération de toutes les solutions peut prendre trop de temps ; or le temps de recherche de la solution optimale est un facteur déterminant dans la résolution du problème. C'est à cause de celui-ci que les problèmes d'optimisation combinatoire

sont réputés si difficiles. La théorie de la complexité donne des outils pour mesurer ce temps de recherche.

Quelques problèmes d'optimisation peuvent être résolus de manière :

- exactes, en fournissant une solution optimale au prix d'un temps de calcul souvent important ;
- approchées, en fournissant une solution réalisable de bonne qualité (dont on peut garantir la qualité), en un temps de calcul raisonnable. Il est parfois possible de caractériser un certain type de garantie de performance. Dans le cas inverse, on parle de méthode heuristique simple.

Les méthodes heuristiques s'inspirent de la structure et des spécificités du problème traité. Dans la famille des méthodes heuristiques, on distingue les algorithmes gloutons dont le comportement vise à construire une solution en ajoutant à chaque étape l'élément susceptible de conduire à la solution optimale, aux vus des données dont on dispose à cette étape.

La plupart des problèmes d'optimisation peuvent être aussi résolus en formulant le problème comme un programme linéaire en variables réelles. Pour plus de détails, voir le paragraphe 8.

4 Bornes inférieures

Certains problèmes d'optimisation peuvent ne pas avoir de solution exacte. Cependant, dans certains cas, on peut leur associer une solution approchée. Cette solution n'est pas optimale, mais elle est généralement proche de la solution optimale. La méthode se décompose en deux phases. La première phase consiste à déterminer une solution approchée (meilleure solution) et la seconde consiste à déterminer une borne supérieure de la valeur optimale dans le cas de maximisation ou une borne inférieure de la solution optimale dans le cas de minimisation. Ceci permet de fournir un intervalle ou un ratio entre la valeur de la meilleure solution trouvée dans la première phase et la borne supérieure/inférieure.

4.1 Relaxation de contraintes

Dans le cadre de la minimisation, une borne inférieure correspond souvent à la solution optimale du problème pour lequel on a relâché certaines contraintes. En effet, une relaxation de contraintes consiste à élargir l'espace des solutions et donc à autoriser d'autres solutions, non réalisables, de valeur éventuellement meilleure. Cette problématique revient à relâcher les variables qui rendent le problème complexe.

La relaxation fractionnaire s'applique aux problèmes d'optimisation combinatoire, et elle consiste à supprimer les contraintes d'intégrité. Cette relaxation est dite linéaire pour les problèmes linéaires en nombres entiers. Dans la mesure où le problème est alors réduit à un programme linéaire en variables continues, il est soluble plus aisément.

La relaxation Lagrangienne revient également à rechercher certaines contraintes (contraintes de capacité) [33, 77], mais à en tenir compte dans la fonction d'objectif sous la forme d'une pénalité qui combine linéairement les contraintes relâchées.

5 Algorithmique

Le mot algorithme vient du nom du mathématicien arabe du IX^e siècle, Muhammed Ibn Al-Khawarizmi. Un algorithme peut être vue comme une méthode permettant de résoudre des classes de problèmes semblables. L'algorithme trouve un bon nombre de ses applications dans les problèmes d'optimisation. La complexité (en temps) d'un algorithme se mesure par son coût en temps de calcul en fonction de la taille de l'instance du problème considéré, c'est-à-dire par le nombre d'opérations élémentaires engendrées par l'exécution de l'algorithme. Ce nombre est exprimé en fonction de la taille d'instance du problème [17]. Par exemple, le nombre d'opérations élémentaires engendrées par l'algorithme classique permettant de calculer le produit de deux matrices carrées de dimensions n est inférieur ou égal à Cn^3 où C est une constante suffisamment grande. Dans cet exemple, la taille de l'instance est égale à n^2 pour les matrices carrées d'ordre n. On dit que l'algorithme est polynomial et que sa complexité est en $O(n^3)$.

6 Complexité

La plupart des problèmes sont résolus en temps polynomial $O(n^k)$, mais une question se pose : *tous les problèmes peuvent-ils être résolus en temps polynomial ?* La réponse est non. Par exemple, il existe des problèmes, tel les fameux **problème de l'arrêt d'une machine de Turing**, qui ne peuvent être résolus par aucun ordinateur, quel que soit le temps qu'il y passe. Il existe aussi des problèmes qui peuvent être résolus, mais pas en temps $O(n^k)$ pour une constante k quelconque. De la même manière, l'expérience montre que certains problèmes sont plus faciles que d'autres à résoudre. Une théorie de la complexité a été développée et permet mathématiquement de classer les problèmes faciles et difficiles en deux classes : la classe P et la classe NP. Dans la suite de cette partie, nous exposons les grands principes de la théorie de complexité [49, 16, 38, 84, 17].

6.1 Les classes P et NP

Pour pouvoir exposer la notion de classe de problèmes, il est tout d'abord nécessaire de distinguer les problèmes de décision des problèmes d'optimisation. Un problème de décision est un problème pour lequel la réponse est *oui* ou *non*. Il est possible d'associer à chaque problème d'optimisation, un problème de décision en introduisant un seuil K correspondant à la fonction objectif f. Le problème de décision devient : *existe-t-il un ordonnancement réalisable* (S) *tel que* $f(S) \leq K$ *?*

La classe P se compose des problèmes qui sont solubles en temps polynomial. Plus précisément, il existe des problèmes qui peuvent être résolus en temps $O(n^k)$ pour une certaine constante k (où n est la taille de l'entrée). Les algorithmes dont la complexité ne peut pas être bornée polynomialement sont qualifiés d'exponentiels.

La classe NP (*Non-deterministic polynomial*) se compose des problèmes qui sont solubles en temps polynomial, par un algorithme non déterministe [1]. Pour ces algorithmes, si à chaque instruction, le bon choix est effectué, le temps de calcul de la solution optimale est polynomial.

1. Algorithme qui comporte des instructions de choix.

Si au contraire, tous les choix sont énumérés, l'algorithme devient déterministe et son temps de calcul devient exponentiel.

Par conséquent, tout problème de la classe P appartient nécessairement à la classe NP : en effet, si un problème appartient à P, alors on peut le résoudre en temps polynomial même si l'on ne nous donne pas de solution. Donc, on peut dire $P \subseteq NP$.

Parmi les problèmes de la classes NP, une large classe de problèmes, les problèmes NP-complets, sont équivalents entre eux quant à l'existence d'un algorithme polynomial pour les résoudre.

La classe $NP-$complet possède une propriété surprenante : si un problème NP-complet peut être résolu en temps polynomial, alors tous les problèmes de la classe NP peuvent être résolus en temps polynomial, autrement dit $P = NP$. Pourtant, malgré des années de recherches, aucun algorithme polynomial n'a jamais été découvert pour quelque problème NP-complet que ce soit.

Les problèmes NP-Complets sont les problèmes les plus difficiles de NP. Afin de décrire cette classe d'équivalence, définissons tout d'abord la réduction polynomiale entre deux problèmes. Soit P_1 et P_2 deux problèmes de décision. On dit qu'un problème P_1 est réductible en temps polynomial à un problème P_2, ce qui s'écrit $P_1 \leq p\ P_2$, s'il existe un algorithme de résolution de P_1, qui fait appel à un algorithme de résolution de P_2, et qui est polynomial lorsque la résolution de P_2 est comptabilisée comme une opération élémentaire. On dit alors d'un problème de décision qu'il est NP-complet si tout problème de la classe NP se réduit polynomialement à lui. Les problèmes d'optimisation combinatoire dont le problème de décision associé est NP-complet sont qualifiés de NP-difficiles.

6.2 Preuves de NP-complétude d'un problème

D'après [49], la démonstration d'appartenance d'un problème de reconnaissance à la classe des problèmes NP-complets est très important puisque, une fois cette démonstration faite, on peut considérer comme peu vraisemblable l'existence d'un algorithme polynomial pour résoudre ce problème. Prouver qu'un problème de décision p est NP-complet consiste en les quatre étapes suivantes :

1. Montrer que p est dans NP ;
2. Choisir p', un problème NP-complet connu ;
3. Construire une transformation f de $p' \rightarrow p$;
4. Prouver que f est une réduction polynomiale.

Les premiers problèmes qui ont été prouvés comme étant NP-complets par [38] sont :

1. **Le problème de la satisfiabilité de circuit (SAT)** : Étant donné un circuit combinatoire booléen, le problème est dit satisfiable s'il existe un ensemble d'entrées booléennes qui fait que ce circuit donne en sortie la valeur 1.
   CIRCUIT-SAT : un circuit combinatoire booléen (C)
   Question : C est-elle satisfiable ?
2. **Problème de satisfaisabilité de formule (SAT)** : Étant donné une expression booléenne, le problème est dit satisfiable s'il existe une affectation en 0 et 1 de ses variables telles que l'évaluation de la formule donne 1.

SAT : formule booléenne ϕ ;
Question : ϕ est-elle satisfiable ?

3. **Satisfaisabilité** $3-CNF$ Étant donné une formule booléenne en forme normale conjonctive (CNF) d'ordre 3, construite à partir d'un ensemble fini de variables ;
Question : existe-t-il une affectation de ces variables qui satisfasse toutes les classes ?

7 Quelques problèmes NP-complets

Les problèmes NP-complets apparaissent dans des domaines variés : logique booléenne, graphes, arithmétique, séquencement et planification, automates et théorie des langages, etc. Dans ce paragraphe, nous présentons quelques autres problèmes NP-complets dont la complétude n'a pu être démontrée que grâce à l'existence des premiers problèmes NP-complets déjà cités dans le paragraphe précédent. Ce sont les problèmes de base exposés par [38] :

1. **Le problème de la clique**
Une clique dans un graphe $G = (S, A)$ est un sous-ensemble $S' \subseteq S$ de sommets, dont chaque paire est reliée par une arête de A. La taille de la clique est le nombre de sommets qu'elle contient. Le problème de la clique est le problème d'optimisation consistant à trouver une clique de taille maximale dans un graphe. En tant que problème de décision, on veut savoir si une clique de taille K donnée existe dans le graphe.
2. **Problème de la couverture de sommets**
Une couverture de sommets d'un graphe $G = (S, A)$ est un sous-ensemble $S' \subseteq S$ tel que si $(u, v) \in A$, alors $u \in S'$ ou $v \in S'$ (ou les deux). La taille d'une couverture de sommets est le nombre de sommets qui la composent. Le problème de la couverture de sommets consiste à trouver une couverture de sommet de taille minimale dans un graphe donné. Si ce problème d'optimisation est annoncé comme un problème de décision, il est souhaitable de déterminer si un graphe possède une couverture d'une taille K donnée.
3. **Cycle Hamiltonien**
Étant donné un graphe $G = (S, A)$, existe-t-il un cycle passant une fois et une seule par chacun des sommets S de G ?
4. **Problème du voyageur de commerce**
Dans le problème du voyageur de commerce le problème est modélisé comme un graphe complet à n sommets, le représentant souhaite faire une tournée en visitant chaque ville exactement une fois, et en terminant sa tournée dans la ville de départ avec un coût minimal.
5. **Problème de la somme de sous-ensembles**
Étant donné un ensemble fini $S \subseteq N$ et un objectif $t \in N$, existe-t-il un sous-ensemble $S' \subseteq S$ dont la somme des éléments est t.

8 Méthode classique de résolution

Nous avons vu précédemment qu'il existe des problèmes d'optimisation de complexité différente. Les problèmes appartenant à la classe P ont des algorithmes polynomiaux permettant

de les résoudre. Pour les problèmes appartenant à la classe NP, l'existence d'algorithmes polynomiaux semble peu réaliste. Ainsi, différentes méthodes de résolution (méthodes exactes ou heuristiques), sont largement utilisées pour appréhender les problèmes NP-difficiles. Nous exposons dans cette partie, les grandes lignes des méthodes les plus connues classées en trois catégories : les méthodes exactes, les heuristiques constructives et les méthodes amélioratives.

8.1 Méthodes exactes

Dans ce paragraphe, il est question de trois types de méthodes exactes : la programmation dynamique, la méthode de séparation et évaluation (Branch and Bound) et la résolution à partir d'une modélisation analytique (la programmation linéaire). Le temps de calcul de ces méthodes est exponentiel, ce qui explique qu'elles ne sont utilisables que sur des problèmes de petites tailles.

8.1.1 Programmation dynamique

La programmation dynamique résout les problèmes en combinant les solutions de sous-problèmes. La programmation dynamique est applicable lorsque les sous-problèmes ne sont pas indépendants, c'est-à-dire lorsque des sous-problèmes ont en commun des sous-sous-problèmes. Un algorithme de programmation dynamique résout chaque sous-sous-problème une seule fois, et mémorise son exécution dans un tableau, épargnant ainsi le recalcul de la solution chaque fois que le sous-sous-problème est rencontré [38].

La programmation dynamique est en général appliquée aux problèmes d'optimisation. Dans ce type de problème, il peut y avoir de nombreuses solutions possibles. Chaque solution est affectée d'une valeur, et on souhaite trouver une solution ayant la valeur optimale (minimisation ou maximisation). Une telle solution est une solution optimale du problème, et non pas la solution optimale, puisqu'il peut y avoir plusieurs solutions qui possèdent la valeur optimale.

Le développement d'un algorithme de programmation dynamique peut être planifié dans une séquence de quatre étapes :

1. Caractériser la structure d'une solution optimale.
2. Définir récursivement la valeur d'une solution optimale.
3. Calculer la valeur d'une solution optimale en remontant progressivement jusqu'à l'énoncé du problème initial.
4. Construire une solution optimale pour les informations calculées.

Les étapes $1-3$ forment la base d'une résolution en programmation dynamique. On peut omettre l'étape 4 si l'on a besoin que de la valeur d'une solution optimale. Pour plus de détail le lecteur peut consulter [38, 110].

Élément de la programmation dynamique . Dans ce paragraphe nous allons présenter les deux grandes caractéristiques que doit posséder un problème d'optimisation pour que la programmation dynamique soit applicable : sous structure optimale et chevauchement des sous problèmes.

a- Sous structure optimale

La première étape de la résolution d'un problème d'optimisation via la programmation dynamique est de caractériser la structure d'une solution optimale. Retenons qu'un problème exhibe une **sous structure optimale** si une solution optimale au problème contient en elle des solutions optimales de sous-problèmes. Chaque fois qu'un problème exhibe une sous structure optimale, c'est un bon indice de l'utilisabilité de la programmation dynamique. Avec la programmation dynamique, on construit une solution optimale du problème à partir de solutions optimales de sous-problèmes. Par conséquent, on doit penser à vérifier que la gamme de sous-problèmes que l'on considère inclut les sous-problèmes utilisés dans une solution optimale.

La découverte de la sous structure optimale obéit au schéma général suivant :

1. Nous montrons qu'une solution du problème consiste à faire un choix.
2. Nous supposons, que pour un problème donné, nous donnons le choix qui conduit à la solution optimale.
3. À partir de ce choix, nous déterminons quels sont les sous-problèmes qui en découlent et comment caractériser au mieux l'espace des sous-problèmes résultant.
4. Nous montrons que les solutions de sous-problèmes employés par la solution optimale du problème doivent elles-mêmes être optimales, et ce en utilisant la méthode du "*couper/coller*" : nous supposons que chacune des solutions des sous-problèmes n'est pas optimale et nous en déduisons une contradiction. En particulier, en "*coupant*" une solution de sous-problèmes non optimale et en la "*collant*" dans la solution optimale, nous montrons que nous obtenons une meilleure solution pour le problème initial, ce qui contre l'hypothèse que nous avons déjà une solution optimale. S'il y a plusieurs sous-problèmes, ils sont généralement similaires, de sorte que l'argument "*couper/coller*" utilisé pour l'un peut resservir pour les autres, moyennant une petite adaptation.

La sous structure optimale varie d'un domaine de problèmes à l'autre de deux façons : (1) le nombre de sous-problèmes qui est utilisé dans une solution optimale du problème initial, et (2) le nombre de choix que l'on a pour déterminer le(s) sous-problème(s) à utiliser dans une solution optimale.

De manière informelle, le temps d'exécution d'un algorithme de programmation dynamique dépend du produit de deux facteurs : le nombre de sous-problèmes et le nombre de choix envisagés pour chaque sous-problème.

La programmation dynamique utilise la sous structure optimale de manière ascendante (*bottom-up*) : on commence par trouver des solutions optimales pour les sous-problèmes de taille la plus petite, après quoi l'on trouve une solution optimale pour les sous problèmes de taille de plus en plus grande, jusqu'au problème initial. Trouver une solution optimale du problème implique de faire un choix entre des sous-problèmes : lesquels prendre pour résoudre le problème ? le coût de la solution du problème est généralement le cumul des coûts des sous-problèmes, plus un coût lié au choix lui même.

b- Chevauchement des sous-problèmes

La seconde caractéristique que doit avoir un problème d'optimisation pour que la programmation dynamique lui soit applicable est la suivante : l'espace des sous-problèmes doit être "*réduit*", au sens où un algorithme récursif pour le problème résout constamment

les mêmes sous-problèmes au lieu d'en engendrer toujours de nouveaux. En général, le nombre total de sous-problèmes distincts est polynomial par rapport à la taille de l'entrée. Quand un algorithme récursif repasse sur le même problème constamment, on dit que le problème d'optimisation contient des **sous-problèmes qui se chevauchent**. Les algorithmes de programmation dynamique tirent parti du chevauchement des sous-problèmes en résolvant chaque sous-problèmes une seule fois, puis en stockant la solution dans un tableau, ce qui permet ultérieurement de retrouver la solution avec un temps de recherche constant.

c- Reconstruction de la solution optimale
Concrètement, on stocke souvent dans un tableau le choix que l'on a fait pour résoudre chaque sous-problème : cela nous dispense de reconstruire cette donnée à partir des coûts que l'on a stockés, ce qui reviendrait à recalculer la solution.

d- Recensement
Il existe une variante de la programmation dynamique qui offre souvent la même efficacité que l'approche usuelle, tout en maintenant une stratégie descendante. Le principe est de "*recenser*" les actions naturelles, mais inefficaces, de l'algorithme récursif. Comme avec la programmation dynamique ordinaire, on conserve dans un tableau les solutions des sous-problèmes, mais la structure de contrôle pour le remplissage du tableau est plus proche de l'algorithme récursif.
Un algorithme récursif de recensement gère un élément du tableau pour la solution de chaque sous-problème. Chaque élément contient au départ une valeur spéciale, pour indiquer qu'il n'a pas encore été rempli. Lorsqu'un sous-problème est rencontré pour la première fois durant l'exécution de l'algorithme récursif, sa solution est calculée puis stockée dans le tableau. À chaque réapparition du sous-problème, la valeur stockée dans le tableau est tout simplement lue et retournée au programme principal [2].

8.1.2 Méthode de séparation et évaluation (Branch and Bound method)

D'après [49] la méthode de "*séparation et évaluation*" est basée sur une énumération implicite et intelligente de l'ensemble des solutions réalisables. Pour cela, la séparation consiste à décomposer le problème initial en plusieurs sous problèmes qui sont à leur tour décomposables. Ce processus peut se visualiser sous forme d'un arbre d'énumération. Pour chaque sous-problème (nœud de l'arbre), la procédure d'évaluation calcule (dans le cas d'un problème de minimisation) une borne inférieure de la solution obtenue à partir de ce sous-problème. Au préalable, une borne supérieure de la solution optimale a été calculée et est utilisée pour éviter l'exploration de nœuds dont la valeur de la borne inférieure est supérieure à la valeur de la borne supérieure. Cette borne supérieure est réactualisée lorsqu'une solution réalisable de valeur inférieure est atteinte.

Ainsi, l'exploration de certaines branches de l'arbre est coupée, ce qui permet de ne pas énumérer réellement toutes les solutions. Il faut donc remarquer que l'efficacité d'un algorithme de séparation et d'évaluation est déterminée par la qualité des bornes utilisées et par de bonnes conditions de branchement. La programmation dynamique est donc de complexité exponentielle.

2. Cette approche présuppose que l'ensemble de tous les paramètres possibles de sous-problème est connu, et que la relation entre positions dans le tableau et sous-problèmes est établi. Une autre approche consiste à recenser via hachage, en se servant des paramètres des sous-problèmes comme clés.

Pourtant, pour les problèmes NP-difficiles au sens faible, c'est-à-dire pour lesquels il existe des algorithmes de résolution pseudo-polynomiaux, il est souvent possible de construire un algorithme de programmation dynamique pseudo-polynomial, pouvant alors être utilisé pour des problèmes de tailles raisonnables.

8.1.3 Modélisation analytique et résolution

La modélisation analytique d'un problème permet, non seulement de mettre en évidence l'objectif et les différentes contraintes du problème, mais également, parfois de le résoudre. L'idéal est d'obtenir un programme linéaire dont les variables sont réelles. Dans ce cas, il existe un bon nombre de solveurs pouvant le résoudre. Dès que le problème comporte des coûts fixes, ou des décisions nécessitant l'utilisation de variables entières, les modèles deviennent plus difficiles à résoudre. Il en est de même lorsque le modèle n'est pas linéaire, mais quadratique par exemple.

La programmation mathématique du type linéaire en variables mixtes est décomposé en deux phases itératives :

- relaxation des contraintes d'intégrité et résolution par une méthode de simplexe ou de points intérieurs ;
- réintégration des contraintes d'intégrité et résolution par des méthodes arborescentes ou des méthodes de coupes.

8.2 Méthodes heuristiques constructives

Pour les problèmes de grandes tailles, les méthodes exactes ne sont pas envisageables de par leur temps de calcul. Il est dans ce cas possible d'utiliser des méthodes approximatives qui donnent des solutions certes sous optimale, mais obtenues rapidement. Ces solutions peuvent ensuite servir de solution initiale pour les méthodes amélioratrices. La performance de tels algorithmes est généralement calculée par le rapport entre la valeur de la solution calculée par l'heuristique et la valeur de la solution initiale : $R_A(I) = A(I)/OPT(I)$ pour le pire des cas. De plus, lorsque la solution optimale n'est pas calculable (parce que les problèmes sont de trop grande taille, par exemple), il est également possible d'étudier expérimentalement le comportement de l'heuristique en comparant ses performances soit avec les performances d'autres heuristiques, soit avec des bornes inférieures de la solution optimale.

Les méthodes par construction progressive sont des méthodes itératives où, à chaque itération, une solution partielle est complétée. La plupart de ces méthodes sont des algorithmes gloutons car elle considèrent les éléments (processeurs ou tâches, suivant les cas) dans un certain ordre sans jamais remettre en question un choix, une fois qu'il a été effectué. De principe très simple, ces algorithmes permettent de trouver une solution très rapidement.

8.3 Méthodes amélioratrices

Ces méthodes sont initialisées par une solution réalisable, calculée soit à l'aide d'une heuristique constructive exposées précédemment, et recherchent à chaque itération une amélioration de la solution courante par des modifications locales. Cet examen se poursuit jusqu'à ce qu'un critère d'arrêt soit satisfait. L'utilisation de ces heuristiques itératives suppose que l'on puisse

définir pour toute solution S, un voisinage de solutions $N(S)$, contenant les solutions voisines de S. La solution finale est générée en appliquant, plusieurs fois et de façons différente de petites transformations (échange, par exemple). À chaque solution S, nous associons le coût de cette solution $c(S)$. Dans la suite nous exposons les méthodes d'améliorations locales les plus connues.

8.3.1 Méthode de descente

C'est l'une des heuristiques de recherche locale les plus simples. Elle consiste à rechercher dans le voisinage courant, une solution de coûts plus faible. Elle procède ainsi jusqu'à arriver à un optimum local. Cette méthode a l'avantage d'être rapide, mais s'arrête dès qu'un optimum local est atteint, même si celui-ci n'est pas de bonne qualité par rapport à la vraie solution optimale du problème initial qui existe intrinsèquement.

Parmi ces méthodes, nous décrivons ici les méthodes d'échange de type $r-$opt. Ces méthodes d'optimisation ont été initialement proposées par Lin [75] pour résoudre le problème du voyageur de commerce, mais elles s'appliquent également à tout problème combinatoire dont la solution consiste en une permutation de r composantes parmi n.

Le terme $r-$optimal indique qu'une solution ne peut plus être améliorée en échangeant r éléments. La méthode consiste donc à sélectionner r composantes et à regarder si en interchangeant ces composantes, on obtient une meilleure solution. Nous remarquons donc qu'une solution $n-$optimale est une solution optimale au problème initial de taille n. Ainsi, plus r augmente, plus on se rapproche de la solution optimale, mais plus les calculs sont longs.

8.3.2 Recuit simulé

Le recuit simulé est une méthode de descente probabiliste.

L'algorithme choisit aléatoirement une solution initiale S. Une solution voisine Sc de cette solution est générée. On calcule l'écart des fonctions de coût $\delta = f(Sc) - f(S)$. Si on obtient une réduction de la fonction de coût, la solution courante est remplacée par la solution voisine. Dans le cas contraire, si on a une augmentation de la fonction coût, la solution voisine remplace la solution courante avec une probabilité d'acceptation donnée par : $e^{(-\delta/T)}$ où T est un paramètre, appelé température, qui contrôle le processus de recuit.

8.3.3 Recherche tabou

Le principe de l'algorithme est le suivant : à chaque itération le voisinage (complet ou un sous-ensemble du voisinage) de la solution courante est examiné et la solution minimisant l'augmentation de coût est sélectionnée. Pour éviter les phénomènes de cycle, il est interdit de revisiter une solution récemment visitée. Pour cela, une liste taboue contenant les dernières solutions visitées est maintenue à jour. Chaque nouvelle solution considérée chasse de cette liste la solution la plus ancienne. La longueur de la liste, paramètre à définir, détermine donc le nombre d'itérations pendant lesquelles une solution ayant été visitée ne pourra être reconsidérée. Ainsi, la recherche de la solution courante suivante se fait dans le voisinage de la solution courante actuelle sans considérer les solutions appartenant à la liste taboue. Tout au long de l'algorithme, la meilleure solution doit être conservée car l'arrêt se fait rarement sur la meilleure

solution. En effet, l'arrêt de cette méthode peut se faire soit après un certain nombre d'itérations, soit lorsqu'il n'y a pas eu d'amélioration de la meilleure solution depuis un certain nombre d'itérations.

Cette méthode donne de très bons résultats pratiques, malgré l'inexistence de résultats théoriques garantissant la convergence de l'algorithme vers la solution optimale.

8.3.4 Algorithmique génétique

Les algorithmes génétiques forment une famille intéressante d'algorithmes d'optimisation. Ils ont été proposés il y a vingtaine d'années par J.H Holland [61].

Cette classe d'algorithmes est basée sur une imitation des phénomènes d'adaptation des êtres vivants. Cette méthodes d'optimisation est utilisée dans plusieurs recherches. Pour plus de détails le lecteur peut consulter [35, 21, 49].

Les algorithmes génétiques fonctionnent sur une analogie avec la reproduction des êtres vivants. On part d'une population initiale (ensemble des solutions provisoires) sur laquelle des opérations de reproduction, de croisement ou de mutation vont être réalisées dans l'objectif d'exploiter au mieux les caractéristiques et propriétés de cette population. Ces opérations doivent mener à une amélioration (en terme de qualité des solutions) de l'ensemble de la population puisque les bonnes solutions sont encouragées à échanger par croisement leurs caractéristiques et à engendrer des solutions encore meilleures. Toutefois, des solutions de très mauvaises qualités peuvent aussi apparaître et permettent d'éviter de tomber trop rapidement dans un optimum local.

Les difficultés pour appliquer les algorithmes génétiques résident dans le besoin de coder les solutions, et dans les nombreux paramètres à fixer (taille de la population, coefficients de reproduction, probabilité de mutation, etc.). De plus, ces algorithmes demandent un effort de calcul très important.

9 Simulation et optimisation

La simulation est devenue une méthode importante d'étude de systèmes. Elle est utilisé largement dans les domaines de l'industrie et des sciences. Les zones d'application de cette méthode incluent les jeux militaires, l'industrie, et tout autre domaine complexe. La simulation permet d'exécuter le déroulement d'applications ou l'évolution d'environnements complexes impossibles à prévoir autrement, car rare est le cas où tout peut être modélisé par des équations mathématiques. Cela permet également d'offrir des scénarios reproductibles afin de comparer différentes stratégies d'exécution. La plupart du temps, la simulation a recourt à l'optimisation afin de faire des choix les plus judicieux, ces choix étant confirmés ou non par le résultat de la simulation. C'est le cas des simulations en ordonnancement où il s'agit de minimiser le temps d'exécution d'une série de tâches sur différentes machines. Différentes simulations peuvent par exemple être menées afin de comparer différentes stratégies d'allocation des tâches aux machines. Il y a plusieurs raisons qui motivent les interactions entre les modèles de simulation et d'optimisation :

- Simulation pour l'optimisation. Les simulations travaillent comme un générateur de scénarios qui fournit à l'optimisation un espace d'échantillon pour sa validation. Un des paramètres plus communs survient dans la programmation stochastique.
- Optimisation par simulation. Dans cette catégorie les composants de l'optimisation orchestrent la simulation d'une séquence de configurations du système, afin que la configuration du système obtenue finalement ait une solution optimale ou proche de l'optimale.

En plus de l'intégration de la simulation et des modèles d'optimisation, il y a besoin aussi d'intégrer respectivement des modèles de simulation ou des modèles d'optimisation eux-mêmes. Dans les systèmes de simulations à grande échelle, les modèles peuvent être divisés et peuvent être développés par des groupes différents. De la même façon dans l'optimisation à grande échelle, les problèmes sont souvent décomposés afin de permettre leur résolution. Le calcul distribué joue un rôle de plus en plus important pour la simulation et l'optimisation, alors que l'action du programmeur se limite à la mise en œuvre personnalisée pour des applications spécifiques. L'effort de développement d'un système de calcul distribué pour un problème spécifique est difficile et grand consommateur de temps de développement. Par conséquent, il est nécessaire de développer des architectures génériques dans ce but.

Y. Xu et S. Sen proposent une architecture de programmation distribuée pour les développement en recherche opérationnelle. Cette architecture est une partie d'une plate-forme de simulation pour l'expérimentation et l'évaluation des systèmes de calculs distribués (*Simulation Platform for Experimentation and Evaluation of Distributed Computing Systems SPEED-CS*) [113]. Cette architecture concerne des modèles d'optimisation et des modèles des simulations d'événements discrets. L'architecture proposée est un système multi-couches qui est composée des éléments suivants :

- une couche de logiciel de configuration (serveur d'objets (ORBE)) fournit les services de communications distribuées ;
- une couche de gestion des composants : le service principal fourni par cette couche est de diriger les composants du modèle ;
- une couche composant : maintient une collection de composants du modèle de simulation et de composants du modèle de décision. En spécifiant les noms et en associant des relations des composants, les utilisateurs peuvent construire un modèle composé ;
- interface utilisateur : cette couche fournit une interface de modélisation afin que l'utilisateur construise ses modèles.

Les auteurs font la démonstration de leur approche et discutent de l'interaction entre optimisation et simulation. Les exemples donnés incluent les mises en œuvre de chaque couche, les modèles pour chaque couche, et les convertisseurs entre la programmation linéaire (Lineare Programming (LP)) et la simulation. Le système est capable de fournir des services pour faciliter le calcul distribué, la gestion des évènements, les services d'attribution de noms, et la gestion des composants. Ils utilisent XML comme format standard de données pour les composants. Un autre trait important est que les ensembles des composants peuvent être mis à jour et agrandis avec des modèles différents. Les modèles peuvent être vu comme des modèles à évèvements discrets.

10 Synthèse

Nous avons présentés dans ce chapitre une introduction à l'optimisation en définissant tous ses composants : les contraintes, l'ensemble des solutions réalisables et la fonction objectif. Les problèmes d'optimisation sont des anciens problèmes et ils sont souvent utilisés dans des différents domaines, tel que les domaines industriels, les domaines de la recherche opérationnelle ou bien dans les domaines mathématiques. Nous avons présenté les différentes méthodes de résolution des problèmes d'optimisation, maximisation ou minimisation, ainsi nous avons fait un rappel sur la complexité en temps des algorithmes de résolution.Ainsi certains problèmes sont résolus en temps polynomial ainsi que des autres problèmes peuvent être résolus en temps exponentiel ou pseudo polynomial. La théorie de la complexité a été développée et permet de classer les problèmes faciles et difficiles en deux classes P et NP. Nous avons également présenté des stratégies de résolution sous-optimales pour les problèmes les plus difficiles. Enfin, nous avons introduit la notion de simulation permettant d'assister les stratégies d'optimisation ou de les valider.

Dans le chapitre suivant nous présentons le concept relatif à l'ordonnancement. Nous dressons un tour d'horizon des techniques d'optimisation visant à servir l'ordonnancement face à différents objectifs et différents contextes.

Chapitre 3

Ordonnancement et Optimisation

L'ordonnancement est largement utilisé pour résoudre les problèmes d'optimisation que ce soit minimisation ou maximisation.

Les différentes configurations de machines (ou processeurs) pourraient avoir plusieurs complexités. Tout d'abord, la configuration la plus simple est composée des machines identiques, où toutes les machines peuvent exécuter toutes les tâches. Une tâche i a alors une durée opératoire P_i quelle que soit la machine qui l'exécute.

Ensuite, les machines uniformes sont souvent polyvalentes et possèdent des vitesses différentes. La durée d'exécution de la tâche i sur la machine m est $P_{im} = \frac{P_i}{S_m}$ où S_m est la vitesse de la machine m. Enfin, il existe un cas plus difficile, où les machines sont non liées et la durée d'exécution P_{im} d'une tâche i dépend à la fois de la tâche et de la machine m sur laquelle elle est exécutée. Cependant, il est possible de spécifier qu'une machine ne peut exécuter certaines tâches en leur donnant des durées opératoires infinies (∞).

Notons que considérer des machines non liées complique fortement les problèmes dans certains cas. L'étude de cette configuration est justifiée car elle permet de rester proche de la réalité, où il existe plusieurs machines pouvant réaliser une même opération avec des temps et des coûts différents en fonction de la génération technologique de la machine, de ses spécificités propres et de ses outils disponibles, etc.

Ce chapitre est consacré à un état de l'art de ce domaine.

Le chapitre est divisé en quatre paragraphes. L'objet du premier est de présenter les notions fondamentales concernant les problèmes d'ordonnancement, de montrer la diversité de ces problèmes en indiquant quelques domaines d'application. Dans le paragraphe suivant, nous présentons les techniques d'ordonnancement des tâches sur une plate-forme hétérogène. Le troisième paragraphe traite des problèmes d'ordonnancement des tâches avec minimisation de temps total d'exécution (C_{max}), minimisation de la somme des dates de fin d'exécution ($\sum C_i$), maximisation de débit et la minimisation du coût total d'exécution. Nous concluons ce chapitre par une synthèse.

1 Introduction à l'ordonnancement

L'ordonnancement est un champ d'investigation qui a connu un essor important ces dernières années, tant par les nombreux problèmes identifiés que par l'utilisation et le développement de nombreuses techniques de résolution. Les problèmes d'ordonnancement se rencontrent souvent dans le milieu industriel. Il s'agit de répartir un ensemble de travaux sur des machines ou ateliers de production en respectant au mieux un ensemble de contraintes (technologiques, temporelles, etc.) et en cherchant à optimiser un ou plusieurs objectifs (cadence de production, délais, coûts, etc.). En informatique, on est également confronté aux problèmes d'ordonnancement pour allouer des processeurs à l'exécution des programmes. Nous citons brièvement d'autres domaines d'application des problèmes d'ordonnancement : génie civil (suivi de projets), administration (gestion du personnel, emploi du temps) ou toute autre structure.

1.1 De l'ordonnancement en général

1.1.1 Terminologie et domaine d'utilisation

Dans un problème d'ordonnancement, quatre notions fondamentales interviennent. Ce sont les travaux (ou *jobs*), les ressources, les contraintes et les objectifs. Un travail (on dit aussi *tâche*) est défini par un ensemble d'opérations qui doivent être exécutées. Une ressource est un moyen matériel (machine) ou humain intervenant dans la réalisation d'un travail. Les contraintes représentent les limites imposées par l'environnement ou les ressources, tandis que les objectifs sont les critères à optimiser. La résolution d'un problème d'ordonnancement consiste à déterminer [56] :

- le placement des travaux dans l'espace, c'est-à-dire sur les processeurs ;
- le placement des travaux dans le temps, c'est-à-dire les instances de début d'exécution de chaque travail sur chacune des ressources qui participent à sa réalisation. Ce placement découle de l'ordre de prise en charge des travaux par les ressources.

Dans de nombreux problèmes d'ordonnancement, deux hypothèses de base sont généralement respectées :

- À chaque instant, une machine ne peut exécuter qu'un seul travail ;
- À chaque instant, un travail peut être exécuté par une machine au plus.

Cependant, il y a des problèmes d'ordonnancement plus spécifiques :

- l'ordonnancement par lots (ou *batch*) [95, 94], où une machine peut exécuter plusieurs travaux simultanément.
- l'ordonnancement avec chevauchements, où les opérations d'un même travail peuvent être en cours d'exécution sur plusieurs machines à un même instant.

Aussi, deux modes d'exécution sont possibles :

- avec préemption : l'exécution d'une opération peut être interrompue puis reprise sur une des machines ;
- sans préemption : si une opération a commencé, elle doit être menée jusqu'à son terme sur la même machine, sans interruption.

Enfin, les problèmes d'ordonnancement se divisent en deux grandes catégories, selon le nombre d'opérations nécessaires à la réalisation de chaque travail. La première regroupe les

problèmes pour lesquels chaque travail nécessite une seule opération, et la deuxième regroupe ceux pour lesquels chaque travail requiert plusieurs opérations.

Cependant les ordonnancements sont communément représentés par des diagrammes de *Gantt*. Ils représentent simplement une solution et sont composés des deux graphiques. Le premier montre les exécutions des tâches en fonction du temps 3.1 et le second nous informe de l'état des ressources qui sont consommées en fonction du temps 3.2 (et bien entendu en fonctions des tâches). Les figures 3.1 et 3.2 illustrent l'ordonnancement des quatre tâches sur deux processeurs.

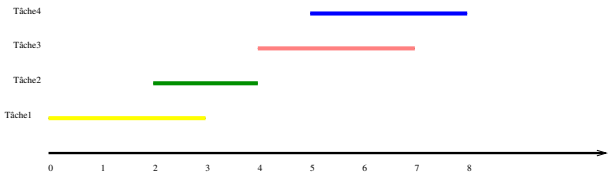

Figure 3.1 – Exemple de diagramme de Gantt (tâches)

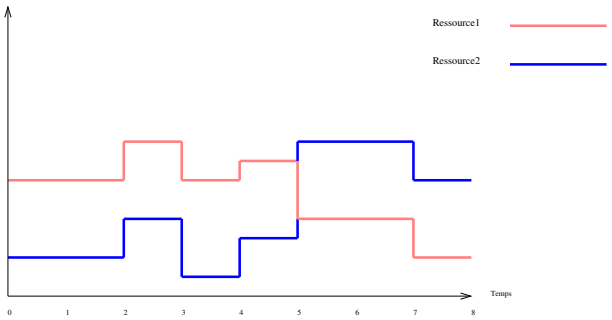

Figure 3.2 – Exemple de diagramme de Gantt (ressources)

1.1.2 Configuration des machines

La configuration de machines considérées dans l'ordonnancement pourrait être [22] :

- machine unique : tous les travaux sont appelés à être exécutés sur une même machine ;
- machines dédiées : plusieurs machines, chacune étant spécialisée pour l'exécution de certains travaux.

  Dans ce cas, on distingue trois modèles différents de machines en fonction des vitesses de celles-ci :
  - machines identiques : toutes les machines présentent la même vitesse d'exécution quel que soit le travail ;
  - machines uniformes : la vitesse d'une machine diffère d'une autre par un coefficient de proportionnalité ;
  - machines indépendantes (on dit aussi non liées) : chaque machine présente une vitesse particulière pour chaque travail ; la durée opératoire d'un travail dépend du travail et de la machine qui l'exécute.

Les problèmes de la deuxième catégorie (machines dédiées) sont dits *problèmes d'atelier* du fait de la nécessité du passage de chaque travail sur deux ou plusieurs machines dédiées. Ils sont généralement spécifiés par la donnée de m machines et de n travaux composés chacun de m

opérations. Chaque opération devrait être exécutée par une machine différente. Selon le mode de passage des opérations sur les différentes machines, nous différencions trois sous-classes de problèmes :

- Open-shop : l'ordre de passage des opérations sur les machines est libre.
- Flow-shop : l'ordre de passage des opérations sur les machines est le même pour tous les travaux.
- Job-shop : chaque travail a un ordre de passage des opérations sur les machines.

1.1.3 Classes d'ordonnancement

En général, les classes d'ordonnancement les plus utilisées sont :

- **Ordonnancement Statique** [85, 107] : les algorithmes statiques sont applicables pour des programmes dans lesquels il est possible d'obtenir la description de tous les calculs à exécuter avant le démarrage. Une analyse de cette description permet de calculer à l'avance le processeur et la date à laquelle chaque tâche sera exécutée.
- **Ordonnancement Dynamique** [105] : dans le cas où les tâches ne sont connues que lors de l'exécution du programme, l'ordonnancement applique des stratégies dynamiques. Ce type d'algorithme prévoit une activation à chaque changement d'état du programme en exécution, par exemple au moment de la création ou de la terminaison d'une tâche. Un surcoût d'ordonnancement est donc ajouté au coût total d'exécution du programme. Bien que les algorithmes dynamiques ne puissent pas analyser la description complète du problème, ils peuvent suivre l'évolution des taux d'utilisation des ressources de la machine.

1.2 Classification

Étant donné la diversité des problèmes d'ordonnancement, nous utilisons couramment un formalisme issu des travaux de Conway et al. [37], Rinnoy Kan [93], Blazewicz et al. [18], et par Demeulemeester et al. [41]. Ces travaux permettent de distinguer les problèmes d'ordonnancement entre eux et de les classer. Ce formalisme, comporte trois champs $\alpha/\beta/\gamma$ permettant de décrire les différentes entités d'un problème d'ordonnancement.

1.2.1 Champ α : organisation des ressources

Le champ α est composé de deux sous-champs $\alpha_1\alpha_2$. Les ressources peuvent être parallèles (chaque machine peut exécuter chaque tâche) ou dédiées (spécialisées à l'exécution d'une ou plusieurs tâches). Dans le cas des machines parallèles, elles peuvent être soit identiques et avoir la même vitesse pour toutes les tâches ; soit uniformes, chaque machine a alors sa propre vitesse qui ne dépend pas de la tâche exécutée ; soit non liées, cas plus général, où la vitesse d'exécution dépend de la machine et de la tâche exécutée.

Le système peut être plus ou moins simple. L'exécution d'une tâche peut être réalisée soit en une seule opération, soit en plusieurs opérations successives exécutées sur des machines différentes. Dans ce dernier cas, plusieurs structures sont définies : les problèmes de flow-shop, lorsque chaque tâche comporte le même enchaînement d'opérations, les problèmes d'open-shop, où l'ordre des opérations est quelconque et les problèmes de Job-shop, où chaque tâche a une gamme d'opération qui lui est propre.

Le paramètre $\alpha_1 \in \{\emptyset, P, Q, R, F, O, J\}$ représente le type de ressources principales (machines/processeurs) utilisées :

$\alpha_1 = \emptyset$: une seule machine. C'est le cas le plus simple qui peut être vu comme un cas particulier des autres configurations.

$\alpha_1 = P$: machines parallèles identiques. Les ressources sont composées de machines de même vitesse, disposées en parallèle pouvant exécuter toutes les tâches.

$\alpha_1 = Q$: machines parallèles uniformes. Les vitesses des machines sont différentes, mais restent indépendantes des tâches.

$\alpha_1 = R$: machines parallèles non liées. Les vitesses des machines sont différentes et dépendent des tâches exécutées.

$\alpha_1 = F$: flow-shop. Les tâches sont décomposées en plusieurs opérations qui doivent être exécutées sur l'ensemble des machines, disposées en série, selon un même routage.

$\alpha_1 = O$: Open-shop. Les opérations des tâches doivent être exécutées sur certaines machines sans restriction sur le routage des tâches.

$\alpha_1 = J$: Job-shop. Les opérations des tâches doivent être exécutées sur l'ensemble des machines disposées en série, mais peuvent avoir des routages différents.

Le paramètre $\alpha_2 \in \{\emptyset, m\}$ dénote le nombre des machines composant le système. Lorsque $\alpha_2 = \emptyset$, le nombre de machines est variable, tandis que lorsque $\alpha_2 = m$ le nombre de machines est égal à m ($m > 0$).

1.2.2 Le champ β : contraintes et caractéristiques du système

Ce champ $\beta = \beta_1\beta_2\beta_3\beta_4\beta_5\beta_6\beta_7\beta_8$ décrit les caractéristiques des ressources et des tâches. Pour répondre à des problèmes industriels des plus divers, la variété de ces caractéristiques est très importante.

Tout d'abord, différents modes d'exécution sont possibles : soit avec préemption (l'exécution d'une opération peut être interrompue puis reprise sur la même machine ou sur une autre), soit sans préemption (lorsqu'une opération est commencée, elle doit être terminée avant de pouvoir exécuter une autre opération sur la même machine). Le paramètre $\beta_1 \in \{\emptyset, pmtn\}$ indique cette possibilité de préemption ($\beta_1 = pmtn$). Si $\beta_1 = \emptyset$, la préemption n'est pas autorisée.

Le paramètre $\beta_2 \in \{\emptyset, res\}$ caractérise les ressources supplémentaires nécessaires à l'exécution d'une tâche (outils, ressources de transport). $\beta_2 = \emptyset$ indique qu'il n'y a pas de ressource complémentaire. $\beta_2 = res$ spécifie les ressources complémentaires.

Le paramètre $\beta_3 \in \{\emptyset, prec, tree, chain\}$ reflète les contraintes de précédence entre tâches spécifiant qu'une tâche doit être exécutée avant une autre. Les valeurs $\emptyset, prec, tree, chain$ représentent respectivement des tâches indépendantes, des relations de précédence générales, des relations de précédence particulières sous forme d'arbre ou de chaîne.

Le paramètre $\beta_4 \in \{\emptyset, r_j\}$ décrit les dates d'arrivée (ou dates de disponibilité, ou dates au plus tôt) des différentes tâches dans le système. Ces dates peuvent être identiques et égales à zéro pour toutes les tâches ($\beta_4 = \emptyset$) ou différentes suivant les tâches ($\beta_4 = r_j$).

Le paramètre $\beta_5 \in \{\emptyset, p_j = p\}$ détaille les temps d'exécution des différentes tâches. Ces temps peuvent être ou ne pas être fonction de la machine.

Le paramètre $\beta_6 \in \{\emptyset, d_j, \overline{d_j}\}$ indique les éventuelles dates d'échéance (ou dates au plus tard) des tâches, dates pour lesquelles les tâches doivent être terminées.

Le paramètre $\beta_7 \in \{\emptyset, s, s_i, s_{ij}\}$ permet de spécifier des contraintes sur les enchaînements de tâches très souvent introduites pour représenter au mieux les problèmes réels. Il est parfois nécessaire de considérer un temps de changement entre l'exécution de deux tâches différentes sur une même machine pour représenter les changements et réglages d'outils. Ce temp de changement peut être constant (s), fonction de la nouvelle tâche (s_i) ou bien fonction de l'enchaînement de deux tâches (s_{ij}).

Enfin le paramètre $\beta_8 \in \{\emptyset, M_j\}$ indique, dans le cas de machines parallèles, des restrictions sur la polyvalence des machines. L'ensemble M_j représente l'ensemble des machines capables de réaliser la tâche j. Lorsque β_8 est vide, toutes les machines sont capables d'exécuter toutes les tâches.

1.2.3 Champ γ : critère d'optimisation

Le troisième champ, le champ γ, concerne les critères d'évaluation d'un ordonnancement. Les objectifs visés sont liés à une bonne utilisation des ressources, une minimisation du délai global ou encore le respect d'un maximum de contraintes. Nous donnons ici les critères les plus couramment utilisés :

$\gamma = C_{max}$: makespan. Ce critère vise à minimiser la date de sortie du système de la dernière tâche ($C_{max} = \max_j C_j$) [1]

$\gamma = \sum w_j C_j$: somme pondérée des dates de fin d'exécution [2]. Ce critère représente la somme des encours des tâches dans le système.

$\gamma = L_{max}$: décalage temporel maximal. Ce critère mesure la plus grande violation des dates d'échéance souhaitées ($L_{max} = \max_j(L_j) = \max_j(C_j - d_j)$).

$\gamma = \sum w_j T_j$: somme pondérée des retards ($T_j = \max_j(C_j - d_j, 0)$), qui permet d'évaluer les pénalités dues aux retards.

$\gamma = \sum w_j U_j$: somme pondérée du nombre de tâches en retard [3].

$\gamma = \sum x_{im} C_{im}$: coût total d'exécution d'un lot r [4] [5].

2 Techniques d'ordonnancement

Les problèmes d'ordonnancement d'un graphe de tâche se retrouvent souvent dans plusieurs domaines et ont été montré comme étant $NP-$Complet. Plusieurs heuristiques ont été proposés dans la literature que nous allons présenté dans la suite de ce paragraphe.

1. C_j : date de fin d'exécution de la tâche j
2. w_j : poids de la tâche j
3. $U_j = 1$ si la tâche j est en retard, 0 sinon
4. x_{im} : le nombre des machines de type m nécessaires à l'exécution d'une tâche de type i
5. C_{im} : le coût d'exécution d'une tâche de type i sur une machine de type m

2.1 Ordonnancement d'un graphe des tâches sur une plate-forme hétérogène

L'ensemble de différentes ressources interconnectées entre elles avec un réseaux haut débit constitue une nouvelle plate-forme de calcul appelée système hétérogène. Ce système peut assurer deux types de calcul qui sont le calcul parallèle et le calcul distribué [109].

Le calcul hétérogène a pour objectif d'attribuer chaque tâche à l'une des machines du système tout en minimisant le temps total de calcul et le temps de communication inter-machines.

Les problèmes d'ordonnancement d'un graphe de tâches sur une plate-forme hétérogène ont été largement étudiés dans la littérature et ont été démontrés comme étant NP-complet [8, 97, 19, 109, 45]. Divers heuristiques ont été proposées. Ces heuristiques sont classées selon différentes catégories comme les algorithmes d'ordonnancement de liste, les algorithmes de regroupement (clustering algorithms), les algorithmes de duplication des tâches, les algorithmes génétiques, etc. Dans la suite de ce paragraphe nous présentons certains algorithmes largement utilisés pour ordonnancer un ensemble des tâches sur une plate-forme hétérogène, leurs étapes (idées) et leurs complexités. Finalement, nous présentons un algorithme pour ordonnancer les tâches sur une plate-forme homogène

2.1.1 Algorithmes d'ordonnancement de liste.

Les algorithmes d'ordonnancement de liste sont bien connus pour ordonnancer un ensemble des tâches sur une plate-forme hétérogène. Ils sont généralement les plus usuels car ils fournissent une bonne qualité d'ordonnancement avec une basse complexité [109, 19, 45].

L'idée de ces algorithmes est la suivante. Une liste ordonnée des tâches est structurée en attribuant une priorité pour chaque tâche et les tâches sont sélectionnées à l'exécution selon leurs priorités [45]. Parmi ces algorithmes nous citons :

Mapping Heuristic (MH) [45] Cet algorithme comprend trois étapes :

1. Le niveau de chaque nœud de graphe des tâches est calculé et utilisé comme une priorité de chaque nœud (dans le cas d'échéance le nœud qui a le plus grand nombre de successeurs immédiats est sélectionné. Si cela n'élimine pas l'échéance un nœud au hasard est choisi). Une liste des évènements est initialisée en insérant l'évènement "tâche est prête" au temps zéro pour chaque nœud qui n'a pas de prédécesseurs immédiats. Les évènements sont ordonnés en ordre décroissant en fonction du niveau des priorités de leurs tâches. En d'autres termes, la tâche ayant la plus haute priorité est en premier.
2. Ensuite, tant que la liste des évènements n'est pas vide :
   (a) Un évènement est obtenu de la liste.
   (b) Si l'évènement indique que *"la tâche T est prête"*, un processeur est sélectionné pour l'exécuter. La tâche sélectionnée est attribuée au processeur déjà sélectionné. Quand l'exécution de la tâche est terminée, l'évènement *"tâche est accomplie"* est ajouté à la liste d'évènements.
   (c) Si l'évènement indique *"tâche est accomplie"*, le statut des successeurs immédiats de la tâche terminée est modifié. Ainsi, lorsque la tâche T est exécutée, le nombre des conditions qui empêchent l'un de ses successeurs immédiats d'être exécutée est

diminué d'une unité. Lorsque le nombre de conditions liées à un successeur devient zéro alors le nœud successeur peut être ordonnancé.

3. L'étape 2 est répétée jusqu'à ce que tous les nœuds du graphe de tâches soient affectés à un processeur.

L'algorithme MH n'ordonnance pas de tâche dans une période libre située entre deux tâches pour lesquelles les dates de début d'exécution ont déjà été calculées. La complexité en temps de cet algorithme prenant en compte la contention est $O(v^2 \times q^3)$, si v est le nombre de tâches et q le nombre de ressources. Sinon sa complexité devient $O(v^2 \times q)$ [109].

Levelized Min Time (LMT) [66, 109], est un algorithme constitué de deux phases. La méthode utilisée pour la première phase est une technique utilisée pour ranger les nœuds basée sur leurs contraintes de précédence, appelé *"Level Sorting"*. Cette technique regroupe les tâches qui peuvent être exécutées en parallèle, c'est à dire les tâches de même niveau n'ayant pas de contraintes de précédence entre elles [66]. La deuxième phase de cet algorithme pourrait assigner les tâches niveau par niveau, en utilisant une heuristique d'affectation qui n'utilise pas l'information de précédence. Cette phase est une méthode gloutonne qui attribue à chaque tâche le processeur disponible le plus rapide. L'algorithme fonctionne selon les étapes suivantes :

1. Calcul du temps d'exécution moyen de chaque tâche sur tous les processeurs disponibles.
2. Si le nombre de tâches dans un même niveau est supérieur au nombre de processeurs disponibles, les tâches de petite taille sont fusionnées en des tâches de plus grande taille jusqu'à ce que le nombre de tâches soit égal au nombre des processeurs.
3. Les tâches sont ordonnées en ordre inverse de leurs tailles (les plus grandes tâches en premier), ceci étant basé sur la moyenne du temps de calcul.
4. Finalement, en partant de la plus grande tâche, chaque tâche est confiée au processeur qui minimise la somme du coût de calcul de la tâche et les coûts de communication avec les tâches dans les couches précédentes. Elle n'a pas de dépendance avec aucune tâche ordonnancée au même niveau.

La complexité de cet algorithme pour un graphe $G(V, E)$ complètement connecté est égale à $O(v^2 \times p^2)$, où v est le nombre des tâches et p est le nombre des processeurs [109].

La figure 3.3 représente un exemple d'algorithme Level Sorting.

Heterogeneous Earliest Finish Time (HEFT) [109]. L'algorithme HEFT est un algorithme d'ordonnancement d'une application sur un nombre limité des processeurs hétérogènes. Cet algorithme se compose de deux phases principales :

- Une phase d'identification des priorités : cette phase exige que la priorité de chaque tâche soit placée selon la valeur du rang montante $rank_u$ (*upward rank value*). Elle est basée sur le coût moyen de calcul et le coût moyen de communication. La liste des tâches est produite en assortissant les tâches en ordre décroissant de $rank_u$.
- Une phase de sélection d'un processeur : les tâches sont sélectionnées selon leurs ordre de priorité. Ensuite, chacune des tâches sélectionnées est ordonnancée sur le processeur, qui minimise le temps d'achèvement de la tâche.

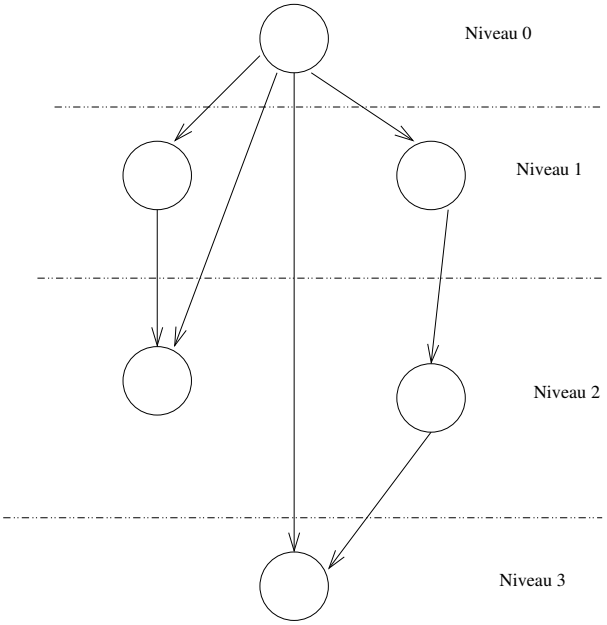

Figure 3.3 – Exemple de Level Sorting

L'algorithme HEFT permet l'insertion d'une tâche dans la plus proche période libre encadrée par deux tâches déjà ordonnancées sur le processeur.

La complexité de cet algorithme est égale à $O(v^2 \times p)$ où v est le nombre des tâches dans un graphe dense et p est le nombre des processeurs.

Critical Path On Processor (CPOP) [109]. Cet algorithme ressemble à l'algorithme HEFT. Il est constitué de deux phases :

- Une phase d'identification des priorités : la priorité de chaque tâche est assignée par l'addition des rangs montants et descendants ($rank_u + rank_d$). À chaque instant la tâche prête est insérée dans une file (queue) de priorité. À chaque étape, l'algorithme CPOP sélectionne la tâche ayant la plus grande valeur de $rank_u + rank_d$ de la file de priorité.
- Une phase de sélection des processeurs : Le processeur de chemin critique, PCP (Processor Critical-Path), est celui qui minimise le coût de calcul cumulé des tâches sur le chemin critique. Si la tâche sélectionnée est sur le chemin critique, alors elle est ordonnancée sur le processeur de chemin critique. Autrement, elle est assignée à un processeur qui minimise le temps de fin au plus tôt de l'exécution d'une tâche.

La complexité de cet algorithme est égale à $O(v^2 \times p)$ où v est le nombre des tâches dans un graphe dense et p est le nombre des processeurs.

L'algorithme PETS (Performance Effective Task Scheduling) [64]. Le PETS algorithme comporte trois phases :

1. Une phase de tri de niveau : dans cette phase le DAG (Direct Acyclic Graph) est traversé de haut en bas pour trier les tâches de chaque niveau, en vue de regrouper les tâches indépendantes les unes avec les autres. Comme résultat, les tâches d'un même niveau peuvent être exécutées en parallèle.

2. Une phase d'attribution de priorité : dans cette phase la priorité de chaque tâche est identifiée. Afin d'attribuer une priorité à une tâche, Ilvarsan et al. définissent trois attributs : le coût moyen de calcul (ACC), le coût de transfert des données (DTC) et le rang des tâches des prédécesseurs (RPT). La priorité est attribuée à toutes les tâches à chaque niveau en se basant sur sa valeur de rang. En d'autres termes, la tâche avec la plus haute valeur de rang reçoit la plus haute priorité suivie de la tâche suivante ayant la plus haute valeur de rang et ainsi de suite. En cas d'échéance, la tâche avec la valeur minimale de ACC reçoit une priorité plus élevée.
3. Une phase de sélection des processeurs : Dans cette phase la tâche est attribuée au processeur ayant un EFT (Earliest Finish Time) minimum. Ilvarsan et al. montrent que la complexité de leur algorithme est égale à $O(e(p + \log(v))$ où v, e et p représentent respectivement le nombre de tâches, le nombre d'arêtes et le nombre de processeurs.

2.1.2 Algorithmes de regroupement (Clustering algorithms)

Les algorithmes de regroupement ou `clustering` sont composés typiquement de deux étapes. Durant la première étape, les tâches sont regroupées ensemble avec leurs prédécesseurs, afin de réduire les coûts de communication. La deuxième étape alloue les groupes résultants aux processeurs disponibles du système [19]. Parmi ces algorithmes nous citons :

- Task Duplication based scheduling Scheme (TDS), qui ordonnance un DAG sur des processeurs hétérogènes interconnectés par des liens homogènes de communication. L'heuristique est composée de trois étapes. Initialement, pour chaque tâche du graphe les temps de début et de fin au plus tôt sont calculés en ordre de manière à identifier le prédécesseur qui devrait être ordonnancé avec la tâche et le processeur qui réduisent au maximum son temps de fin. La prochaine étape consiste à ordonnancer les tâches (dans l'ordre de leur niveau), basée sur le prédécesseur et le processeur favori, et sur les temps de début et de fin au plus tard de la tâche sélectionnée. Dans la dernière étape, une procédure de duplication tente de reproduire le prédécesseur favori de la tâche sélectionnée sur son processeur si le makespan diminue [19, 92].
- Regroupement des tâches pour des processeurs hétérogènes (CTHP (Clustering Tasks onto Heterogeneous Processors)). Cet algorithme est constitué de deux étapes. Initialement la stratégie suppose un environnement virtuel homogène tout en créant des groupes de tâches en utilisant la structure de l'algorithme de regroupement. Dans une deuxième étape, un sous-ensemble des clusters déjà générés sont plaqués à l'environnement hétérogène, compte tenu du coût d'exécution des tâches sur les processeurs et des caractéristiques de transmission sur les liens de communication [19].

2.1.3 Algorithmes de duplication des tâches.

Ce genre d'algorithme réplique les tâches critiques sur plusieurs processeurs afin de réduire le nombre des opérations de communication inter-tâches [4]. Dans la littérature, un certain nombre des algorithmes utilisant cette technique pour un système hétérogène est rencontré, citons par exemple

- CPFD (Critical Path Fast Duplication) [4] : Dans cet article, un nouvel algorithme a été proposé, qui en premier identifie le chemin critique de la liste dominante des nœuds

d'un DAG et sa performance est comparée à cinq autres algorithmes existants lorsque un nombre limité et illimité de processeurs sont supposés être disponibles. Cet algorithme permet de fournir un ordonnancement optimal si le nœud racine est dupliqué à chaque processeur.

– HCNF (Heterogeneous Critical Node First) et TANH (Task duplication Algorithm for Network of Heterogeneous system) [8]. Cet algorithme consiste en quatre phases. Dans la première étape de cet algorithme, le graphe est traversé de manière Top-Down pour calculer les dates au plus tôt de début et de fin d'exécution. Puis au niveau de chaque nœud, le processeur favori et le prédécesseur favori sont déterminés pour les tâches. Dans une deuxième étape, les temps de début et de fin au plus tard sont déterminés de façon *bottom-up*. Le temps de fin au plus tard d'un nœud est utilisé pour calculer le temps de début au plus tard d'un successeur de ce nœud. Les autres étapes de l'algorithme sont la génération des grappes et le choix des processeurs pour leur exécution.

2.1.4 Algorithmes génétiques

Les algorithmes génétiques (GA) consititue une approche heuristique pour trouver des solutions sous optimales dans des grands espaces de la recherche. En résumé, les pas qui sont pris pour rendre effectif un algorithme génétique pour un problème donné de l'optimisation sont les suivants : (1) un codage, (2) la création d'une population initiale, (3) une évaluation des individus de la population grâce à une fonction de l'aptitude particulière *(fitness)*, (4) un mécanisme de sélection, (5) un mécanisme de croisement, (6) un mécanisme de mutation, et (7) un ensemble de critère d'arrêt [111]. Il existe un grande variété d'approches pour les algorithmes génétiques. Nous citons deux approches :

– Techniques de recherche pour les problèmes d'ordonnancement des tâches. Parmi ces algorithmes *"matching and scheduling algorithm"* [111], les caractéristiques de cet algorithme inclues la séparation des représentations du placement et d'ordonnancement, l'indépendance de la structure des chromosomes (individus) des détails de communication, et la prise en compte du recouvrement calcul communication entre tâches dépendantes.
– PSGA (Problem Space Genetic Algorithm) [43], est une technique évolutionnaire qui combine la capacité de recherche des algorithmes génétiques à une heuristique *"fast-problem-specific"* pour provoquer une meilleure solution possible au problème d'une manière efficace comme étant comparée à d'autres heuristiques probables. C'est un algorithme robuste, dans le sens qu'il exploite un espace de solutions large et difficile en un temps minimum de CPU qui n'utilise qu'un faible volume d'espace de la mémoire, comparé aux algorithmes génétiques traditionnels. L'algorithme *"Push-Pull"* [97] commence par le meilleur ordonnancement de tâches trouvé par un algorithme déterministe, et ensuite essaie itérativement d'améliorer la meilleure solution courante grâce à une méthode déterministe de recherche guidée.

2.2 Ordonnancement d'un graphe de tâches sur une plate-forme homogène

Dans [103] l'auteur introduit une nouvelle approche d'un algorithme génétique pour le problème d'ordonnancement d'un graphe de tâches sur un ensemble de processeurs homogènes. Cet algorithme utilise l'approche génétique, sauf que l'algorithme proposé par l'auteur se diffèren-

cie en deux points d'un algorithme génétique classique. D'abord, il utilise une représentation flexible, qui permet à GA de faire évoluer à la fois la structure et la valeur de la solution. Ensuite l'algorithme utilise une incrémentation dynamique de la fonction de *fitness* qui donne une grande satisfaction pour des buts les plus simples. Il augmente graduellement la difficulté de la valeur désirée de *fitness* jusqu'à ce que la solution finale soit atteinte.

3 Objectifs d'ordonnancement

Le problème d'ordonnancement consiste à organiser dans le temps la réalisation des tâches, compte tenu de contraintes temporelles (délais, contraintes d'enchaînement) et de contraintes portant sur la disponibilité des ressources requises pour satisfaire un ou plusieurs objectifs tels que la minimisation du *makespan*, du coût total d'exécution, etc. Ce paragraphe est consacré à un état de l'art sur la complexité des problèmes de machines hétérogènes et les méthodes existantes. Bien sûr, étant donné le très grand nombre de travaux réalisés dans ce domaine, nous n'avons pas pu avoir connaissance de tout et il peut y avoir des oublis. Nous avons malgré tout, essayés d'en rapporter un grand nombre.

3.1 Minimisation du C_{max}

On appele C_{max} le temps final d'exécution d'un certain lot appelé en anglais *makespan*.

3.1.1 $P//C_{max}$

Le problème consiste à ordonnancer un ensemble de tâches sur un ensemble de processeurs (parallèles, hétérogènes) sans contraintes supplémentaires, tout en cherchant à minimiser la durée totale d'exécution (makespan C_{max}). Ce problème est NP-difficile même si le problème est restreint à deux machines [104] ou bien à trois machines [55].

Cependant, malgré l'impossibilité de trouver un algorithme polynomial qui permette de résoudre ce problème, des heuristiques de résolution sont proposées dans la littérature nous citons :

- Les heuristiques composées d'algorithmes de liste. Elles affectent les tâches aux machines en suivant l'ordre d'une liste établie à partir d'un certain critère [57, 79]. Les performances [6] de ces algorithmes, appliqués aux problèmes $P//C_{max}$, est égale à $R_{Liste} = 2 - 1/m$.
- L'heuristique LPT (Longest Processing Time) est la plus connue pour le problème $P//C_{max}$. Elle classe les tâches selon l'ordre décroissant du temps d'exécution. Cette heuristique a une complexité de $O(n \log n)$, et une performance de $R_{LPT} = 4/3 - 1/3m$ [40]. Ultérieurement, Ibarra et Kim [63] prouvent que $R_{LPT} = 1 + 2(m-1)/n$ pour $n \geq 2(m-1)\pi$ avec $\pi = \max_i(p_i)/\min_i(p_i)$.
- L'algorithme génétique utilise l'heuristique de liste d'ordonnancement. Cet algorithme ordonnance d'une manière dynamique un ensemble des tâches hétérogènes sur une plateforme hétérogène distribuée, et d'efficacité élevée [81].

6. Performance d'une heuristique : $R_H = Z^H/Z^*$ où, Z^H est la valeur du critère de la solution donnée par l'heuristique et Z^* est la valeur du critère pour une solution optimale

- Kubiak et Timkovsky [69] étudient le problème d'ordonnancement des tâches minimisant le temps total d'exécution restreint à deux machines ayant des durées opératoires unitaires ($J_2|P_{ij} = 1|\sum C_j$). Les tâches sont indexées selon des indices entiers consécutifs. Les opérations indexées selon les indices impaires sont exécutées sur une machines et les autres indexées par les indices paires sur une autre. L'algorithme (Shortest Remaining First (SFR)) ainsi développé est exécuté en temps polynomial en respectant le nombre succinct d'instances du problème, le nombre de *bits* nécessaire pour encoder une tâche de k opérations est égale à $O(\log(k+1))$.
- Foto Afrati et al. [3] introduisent le premier PTASs (Polynomial Time Approximation Schemes) pour ordonnancer n tâches chargées d'un poids ($\sum w_i C_i$) avec des dates de parution, le but est de minimiser le temps moyen de leur achèvement.
  Il montre que le temps d'exécution de l'algorithme dans le cas des machines parallèles est polynomial en m (nombre de machines) alors qu'il est exponentiel en m pour les machines non liées.
- Abdekhodaee et al. [2] traitent le problème d'ordonnancement des tâches non-préemptives à deux opérations sur deux machines identiques. Un seul serveur convenable pour réaliser la première opération (*setup operation*), la deuxième opération est exécutée automatiquement sur la même machine, sans le serveur. Un des algorithmes utilisés dans cet article est l'algorithme de Glimore-Gomory qui s'exécute en $O(n \log n)$.

D'autres heuristiques sont cités [21] où l'ordonnancement est supposé statique et les tâches sont indépendantes. Citons :

- **Équilibrage opportuniste de la charge** (Opportunistic Load Balancing (OLB)) consiste à assigner les tâches, en ordre arbitraire, aux processeurs disponibles, sans tenir compte du temps d'exécution de la tâche anticipée sur cette machine. Pour cette raison le makespan est minimisé le plus possible.
- **Temps d'exécution minimal** (Minimum Execution Time (MET)) Consiste à assigner chaque tâche, en ordre arbitraire, à la machine avec le meilleur temps anticipé pour cette tâche, sans égard (indépendamment) de la disponibilité de la machine. L'idée de cette heuristique est d'assigner chaque tâche à son meilleur processeur. Cela pourrait causer un déséquilibre de charge entre les processeurs. En général, cette heuristique n'est pas applicable à des environnements hétérogènes de calcul caractérisés par des matrices du temps attendu du calcul (expected to compute matrices (ETC)).
- **Temps de l'achèvement minimum** (Minimum completion time (MCT)) assigne chaque tâche, en ordre arbitraire, à la machine avec la date de l'achèvement de la tâche la plus petite. Cela induit des tâches assignées à des machines qui ne possèdent pas le temps de l'achèvement minimum. MCT combine les avantages de OLB et de MET en évitant les circonstances dans lesquelles OLB et MET s'exécutent d'une manière médiocre.

Pour le cas NP-difficile spécifique d'un ordonnancement acyclique où toutes les opérations ont une unité de longueur, Goldberg et al [55] démontre qu'un ordonnancement avec un makespan en $O(P_{max} + \pi_{max})$ existe toujours, avec P_{max} le temps d'exécution maximum nécessaire pour une tâche et π_{max} le temps maximal d'exécution d'une opération pour une machine. L'algorithme ρ-approximation est définit comme un algorithme en temps polynomial qui donne toujours un ordonnancement faisable avec un makespan d'au plus $\rho-$temps optimal. ρ est appelé garantie d'approximation.

Jansen et al [67] considère les deux variantes d'ordonnancement préemptif et non-préemptif. Les algorithmes pour ces deux variantes calculent des solutions approximatives pour n'importe quelle exactitude ε positive et s'exécutent en temps $O(n)$ pour des valeurs déterminées de m (nombre des machines) μ (nombre des opérations par tâches) et ε. Enfin l'auteur montre que le makespan d'ordonnancement final est au plus $(1+\varepsilon)$ fois l'optimal.

Les problèmes d'ordonnancement de Job-shop et les problèmes de circulation de paquets sont des problèmes fondamentaux dans la recherche opérationnelle et en informatique. Bertsimas et Gamarnik [15] proposent des algorithmes d'ordonnancement asymptotiquement optimaux minimisant le makespan. Pour le problème de circulation de paquets, ils proposent une relaxation par programmation linéaire qui donne une borne inférieure du C_{max} et construit un ordonnancement avec une valeur objective du makespan de $C_{max} + O(\sqrt{C_{max}})$.

Coll et al. [36] considère le problème d'ordonnancement des tâches restreintes à des contraintes de précédence sur un ensemble de processeurs, afin est de minimiser le makespan. Il développe une nouvelle formalisation d'un programme linéaire en variables entières.

Dans le cas des processeurs identiques Abdekhodaee et al. [2] considère le problème d'ordonnancement des tâches non-préemptives à deux opérations (*setup* et *processing*). Un seul serveur est convenable pour réaliser la première opération, la deuxième opération est automatiquement réalisée sans le serveur.

3.1.2 $P/Pmnt/C_{max}$

Dans le cadre des problèmes d'ordonnancement des charges indépendantes et divisibles qui partage des fichiers en entrée, Beaumont et al. [13] envisagent deux méthodes :

1. une méthode pour la distribution optimale du travail aux processeurs. L'approche se base sur la comparaison des quantités de travail exécutées par les deux premiers processeurs esclaves. Comme il y a deux processeurs qui sont sélectionnés, il existe donc deux ordres possibles. Pour chaque ordre, les auteurs déterminent le nombres total des tâches $\alpha_1 + \alpha_2$ traitées par T unités de temps, et l'occupation total t_2 de la moyenne des communications durant cet intervalle de temps. Ensuite les auteurs trouvent des expressions grâce auxquelles ils trouvent des distributions optimales dans quelques cas particuliers :
   - les processeurs sont triés tels que $G_1 \leq \ldots \leq G_p$ lorsque toutes les latences g_i des communication sont nulles ($g_i = 0$). Par conséquent, l'ordre selon lequel les tâches sont envoyées aux processeurs $p_i, 1 \leq i \leq p$ est optimal.
   - les P processeurs sont triés tels que $g_1 w_1 \leq g_2 w_2 \leq \ldots \leq g_p w_p$ lorsque tous les temps élémentaires de transfert $G_i = G$. Par conséquent, l'ordre selon lequel les tâches sont envoyées à $P_1, \ldots, P_p$ est optimal.
2. une méthode qui sélectionne automatiquement les ressources à utiliser. En raison de sa périodicité, il est plus facile à mettre en œuvre et plus robuste aux variations de charge des processeurs ou à ses liens de communications.

Phillips et al. [89] considère le problème d'ordonnancement de n tâches qui sont lancées dans le temps et soumises à des dates de début pour minimiser le temps moyen de leur achèvement sur un système des machines parallèles $(P|r_j, pmnt|\sum C_j)$. L'algorithme utilisé est l'algorithme SPT

(Shortest remaining Processing Time). L'auteur montre que cet algorithme est un algorithme 2-approximé (2-approximation algorithm).

3.1.3 $P/Prec/C_{max}$

Dans le cadre d'ordonnancement des tâches non-préemptives avec contraintes de précédence Pezella et Merelli dans [87] présentent une heuristique efficace minimisant le makespan. Cette méthode de recherche locale est basée sur la combinaison de recherche tabou et la procédure de mouvement de goulet d'étranglement utilisée pour générer la solution initiale et raffiner les prochaines solutions courantes. La solution initiale donnée en changeant le goulet d'étranglement, la structure spéciale de voisinage et la liste dynamique proposée autorisent d'obtenir des résultats intéressants.

3.2 Minimisation de la somme des dates de fin d'exécution ($\sum C_i$)

3.2.1 Sans préemption

Yuan et al. [115] considère le problème d'ordonnancement sur une seule machine minimisant le temps total d'exécution des tâches fixées restreintes à des contraintes de précédence et des dates de fins. Il montre que le problème admet une n-approximation en temps linéaire, mais pas en temps pseudo-polynomial. Il existe un algorithme en $(1-\delta)n$-approximation même si les dates de fins sont toutes nulles, pour tout $\delta > 0$, avec $P \neq NP$, et où n est le nombre des tâches. Pour le cas où les tâches ne sont pas restreintes à des contraintes de précédence et à des dates de fins, l'auteur montre que le problème n'admet pas d'algorithme en $(2-\delta)$-approximation en temps pseudo-polynomial. Pour le cas où les tâches sont pondérées, le problème n'a ni un algorithme en $2^{p(n)}$ approximation en temps polynomial et ni un algorithme en $q(n)$ approximation en temps pseudo-polynomial, où $q(n)$ est un polynôme donné de n. Horn [62], et Bruno et al [23] représentent $R//\sum C_i$ par un programme linéaire en nombre entiers. La structure de ce programme consiste à ce que le problème soit résolu en un temps polynomial. $R//\sum C_i$ étant polynomial, il en est de même pour $P//\sum C_i$ et $Q//\sum C_i$. Il est possible de trouver des réductions polynomiales de ces problèmes vers $R//\sum C_i$, et la méthode utilisée pour $R//\sum C_i$ est utilisable pour $P//\sum C_i$ et $Q//\sum C_i$. Néanmoins, des méthodes plus simples sont développées pour les machines parallèles identiques et uniformes. Par contre, dès que l'on associe des poids aux tâches (correspondants à une priorité, une urgence, ...) le problème devient rapidement NP-difficile. En effet Bruno et al. [23] ont mis en évidence une réduction du problème au problème du sac à dos (connu pour être NP-complet) au problème de décision associé au problème $P_2//\sum w_i C_i$.

Dans le cas de la recherche d'un ordonnancement d'un ensemble des tâches sur un ensemble des machines parallèles identiques minimisant la somme des dates de fin d'exécution des tâches. La généralisation de l'algorithme SPT (Shortest Processing Time), développé pour le problème $1//\sum C_i$, est optimal. Cet algorithme de liste classe les tâches par ordre croissant de leur durée d'exécution et les affecte, dans cet ordre, à la première machine libre. La complexité de cet algorithme est $O(n \log n)$

3.2.2 Avec préemption

Yuan et al. [115] considère le problème d'ordonnancement sur une seule machine minimisant le temps total d'exécution des tâches indépendantes, préemptives et avec des dates de fins différentes. Le problème pourrait être résolu par un algorithme de complexité $O(n \log n)$.

Pour un système de processeurs parallèles non liés René Sitters [101] montre que le problème de minimisation de la somme de date de fin est NP-difficile dans le sens dur. De plus le problème $(R|Pmtn|\sum C_j)$ peut être résolu en temps polynomial, et la préemption ne peut pas réduire la valeur optimal de l'objectif. La complexité de ce problème reste ouverte pour un nombre fixé de machines même si $m = 2$.

3.3 Maximisation du débit

3.3.1 Ordonnancement en régime permanent des tâches indépendantes et sans contraintes de précédence

Le problème d'optimisation du débit en régime permanent est en général NP-complet. Beaumont et al. [11] dérivent une borne sur le débit optimal qui peut être accompli en régime permanent, en utilisant une approche par la programmation linéaire. Ils construisent un ordonnancement périodique qui accomplit ce débit. L'algorithme proposé dans ce travail est constitué par les étapes suivantes :

1. Résoudre le programme linéaire ;
2. Décomposer la solution dans une somme d'allocations et construire le graphe des communications induites par $s(P_i \rightarrow P_j, e_{k,l})$.
3. Transformer le graphe des communications en un graphe biparti chargé d'un poids B.
4. Décomposer le graphe B en un somme pondéré de correspondances $B = \sum \alpha_c X_c$ telle que $\sum \alpha_c \leq 1$.
5. Définir la période T_p
6. Utiliser la correspondance entre les différents modèles et l'ordonnancement périodique, les auteurs dérivent un ordonnancement périodique désiré avec un débit asymptotiquement optimal.

Arnaud Legrand et al. [72] s'intéressent aux communications qui ont lieu lors de l'exécution d'une application complexe distribuée sur un environnement hétérogène de type grille de calcul. Ils cherchent à optimiser le débit d'une diffusion de données à travers le réseau d'interconnexion en régime permanent, en supposant qu'un grand nombre de messages doivent être diffusés successivement. La plate-forme hétérogène visée, est donc modélisée par un graphe où les différentes ressources (calcul ou communication) ont des vitesses différentes. Le calcul du débit optimal d'une diffusion et de la construction d'un ordonnancement périodique qui réalise ce débit, se base sur les équations de programmations linéaires en régime permanent.

Arnaud Legrand et al. [71] et Beaumont et al. [12] considèrent plusieurs communications collectives, comme scatter ou diffusion personnalisée, Personnalisé all-to-all, opérations de réduction, avec l'objectif d'optimiser le débit qui peut être accompli, en régime permanent, en envoyant un grand nombre d'opérations. Un meilleur ordonnancement en temps polynomial est déterminé explicitement pour ces types d'opérations ainsi le débit optimal se retrouve en

se basant sur les équations de programmation linéaires. Pour le problème de séries des opérations réduites, les auteurs utilisent des arbres de réductions pour décrire un ordonnancement polynomial. Enfin on peut signaler que les algorithmes d'ordonnancement concrets basés sur l'opération en régime permanent sont asymptotiquement optimales, dans la classe de tous les ordonnancements possibles.

3.3.2 Ordonnancement en régime permanent des tâches indépendante avec contraintes de précédence

Beaumont et al. [10] proposent un modèle d'ordonnancement en régime permanent des tâches indépendantes sur une plate-forme hétérogène de typologie maître-esclave. Il introduit les contraintes de précédence entre les tâches. Les équations de la programmation linéaire sont dérivées. La solution du système des équations linéaires donne une solution optimale. Comme le problème de la programmation linéaire est en nombre rationnel, alors toutes les variables obtenues sont aussi rationnelles en temps polynomial. Pour obtenir la solution optimale il prend le plus petit commun multiple pour les dénominateurs et donc il dérive une période entière pour les opérations en régime permanent.

3.4 Minimisation des coûts

Dont le but de la minimisation des coûts de production ou d'exécution d'une tâche, que nous allons traiter dans la suite de cette thèse, a fait l'objet de nombreux travaux que nous citons dans la suite de ce paragraphe.

3.4.1 Ordonnancement des tâches sur une seule machine

Motivé par les applications industrielles Baptise et Le Pape [9] ont étudiés le problèmes d'ordonnancement d'un ensemble de tâches sujettes à une date de début r_i(*release date*) et à une date de fin d_i (*deadline date*) sur une seule machine. L'objectif est de minimiser le coût total d'achèvement $\sum_i f_i(C_i)$ où $f_i(C_i)$ correspond au coût de l'achèvement de la tâche T_i au temps $C_i \in [r_i+p_i, d_i]$, avec p_i est le temps de traitement de la tâche i sur le processeur p. La procédure ainsi utilisée est la procédure Branch and Bound. L'algorithme s'exécute en $O(n \log n)$.

Benyoucef et al. [14] ont récemment introduit un nouveau mode d'approvisionnement, appelé livraison synchrone, dans un environnement de production et de livraison de type *"juste-à-temps"*. Il considère un problème d'ordonnancement sur une seule machine où il cherche à minimiser le coût de changement de production total. Les produits sont de différents types et un coût de changement est généré à chaque fois que la machine change de type de produit en production. Pour chaque produit, le temps de production est unitaire et un délai de livraison est à respecter. Principalement dû au contexte de la livraison synchrone, les délais de livraison sont différents.

Ensuite Benyoussef et al. [14] présente une autre approche optimale basée sur la programmation dynamique dédiée au problème du coût de changement dépendant du produit. Cette approche construit un graphe d'état allant du temps T au temps 0 et qui tient compte du produit plus que des tâches. Les auteurs de cet article présentent également un algorithme optimal et deux heuristiques pour le résoudre. La complexité de sa solution est $(O(\pi n_i))$ tandis

que la complexité des deux heuristiques présentées par l'auteur est $(O(NT))$. Des expériences numériques, comparant les deux heuristiques à la solution optimale donnée par la méthode de programmation dynamique montre que ces deux heuristiques sont efficaces.

3.4.2 Ordonnancement des tâches sur plusieurs machines

Mingyuan Chen [34] développe un modèle de programmation linéaire en variables entières pour minimiser une fonction de coût constituée de trois termes : coût de la machine, coût de traitement et coût de reconfiguration de cellule pour une période de planification. L'environnement considéré est un environnement dynamique manufacturier. La résolution de ce problème est NP-complet. L'approche utilisé pour trouver la solution du problème initial est la programmation dynamique.

Pervaiz Ahmad et al. [5] développe et utilise un algorithme génétique, simulated annealing (SA) et recherche tabou pour un système cellulaire manufacturier dans un environnement dynamique, avec comme objectif de minimiser le coût de production. Il présente ainsi un programme non linéaire en variables entières mixtes.

Yumei Hou et al. [116] considère un problème de planification de production pour le Job-shop sur des machines non reliées produisant un nombre de produits. Il existe des bornes inférieures et supérieures pour les produits intermédiaires et une borne supérieure pour les produits finaux. Les capacités des machines sont modélisées comme des chaînes de Markov finies. L'objectif est de choisir la quantité de production qui minimise le coût total escompté (*discounted cost*) de production et le coût de l'inventaire (*inventory cost*). Cependant la présence d'une politique optimale de contrôle pour ce problème est difficile, une approximation asymptotique est donc développée en laissant la quantité d'échange de machines tendre vers l'infini. L'avantage majeur de cette approche est la réduction de la complexité du système.

Cao et al. [29] traite du problème de sélection et d'ordonnancement simultané des machines parallèles pour minimiser la somme du coût d'utilisation de la machine et du coût du retard des travaux. Il développe un modèle d'optimisation combinatoire pour cet objectif. Pour résoudre ce problème, l'auteur developpe un algorithme de recherche heureustique, basé sur la recherche tabou, pour trouver une solution optimale ou proche de la solution optimale. Selon les notations des auteurs, l'équation de la fonction objectif est représentée par la formule suivante :

$$\min T_c = \sum_{k=1}^{K} \beta_k z_k + \sum_{i=1}^{N} w_i \max\{0, C_i - d_i\}$$

avec

- T_c : le coût total ;
- β_k : le coût de la machine utilisée ;
- z_k : une variable bivalente. $z_k = 1$ si la machine est utilisée ; $z_k = 0$ sinon ;
- w_i : le poids associé à la pénalité de retard pour la tâche i ;
- C_i : la date de fin réelle de la tâche i ;
- d_i : la date de fin prévue de la tâche i.
- k : l'indice des machines.

Le premier terme ($\sum_{k=1}^{K} \beta_k z_k$) de la fonction objectif est le coût de la machine et le second terme ($\sum_{i=1}^{N} w_i \max\{0, C_i - d_i\}$) est la pénalité de retard . Cette fonction reflète l'équilibre entre le coût du système et le coût au retard pris sur la réalisation des tâches.

Pierre Boulet et al. [20] présente un algorithme de programmation dynamique pour ordonnancer un ensemble des tâches sur une plate-forme hétérogène avec comme un objectif de minimiser le coût d'une allocation. L'algorithme ainsi présenté dans cet article est de complexité de $O(pB)$ où p est le nombre de processeurs et B est la borne supérieure du nombre des tâches.

Chen et al. [33] considère un problème d'ordonnancement de I tâches sur M machines afin de minimiser le coût total d'exécution ($CT = \sum_{i=1}^{I} CT(i)$ où $CT(i)$ est le coût d'exécution de la tâche i. En effet, chaque tâche est constituée d'une suite d'opérations non préemptives, chaque opération est exécutée sur une seule machine. Les contraintes de précédence entre les tâches sont représentées par la structure *chain/join*. Les auteurs utilisent les multiplicateurs lagrangiens pour relaxer les contraintes de capacité et proposent un algorithme de programmation dynamique pseudo-polynomial pour les niveaux de sous-problèmes des tâches relaxées.

Plesnik, dans l'article [90], prouve que la résolution du problème de $COV^{=}$ (recherche d'un sous-ensemble d'arêtes de coût minimum qui couvrent exactement un nombre donné de sommets) peut se faire en temps polynomial, quand tous les coûts des arêtes dans un graphe $G = (V, E)$ sont positives. Plesnik fait alors les suppositions suivantes :

- la fonction de coût est positive ($c(e) \geq 0$) pour chaque arête e du graphe G ($e \in E(G)$) ;
- le graphe G ne contient pas de sommet isolé (autrement de tels nœuds sont supprimés)

Il pose un entier k tel que $2 \leq k \leq n = |V(G)|$. L'objectif est alors de trouver un k-recouvrement de poids minimal, i.e. un ensemble $S \subseteq E(G)$ d'un coût total minimal sujet à la contrainte que S couvre exactement k sommets de G. L'algorithme présenté est de complexité $O(n^3)$.

Jaudlbauer [68] considère le problème d'ordonnancement pour minimiser le coût total, dans un système de production multi-objets avec une demande dynamique. La fonction objectif inclue le coût d'installation et de traitement de la machine. Il formule un modèle de programmation non-linéaire. La solution trouvée par le solveur standard est meilleure que celle trouvée dans la formulation EPL (Economic Production Lot).

Sohn [102] présente une approche pour déterminer le coût minimal pour lequel la charge doit être divisée entre les processeurs accédés par les clients. Il montre que le temps de calcul minimal peut être obtenu lors de l'arrangement d'une séquence de processeurs, d'une manière à ce que les processeurs les moins chers reçoivent la tâche plus tôt que les processeurs plus chers. Le temps de la finition $T_f = \alpha_i w_i T_{cp}$ peut être réduit en étudiant le rapport entre le temps de la finition de traitement (T_{cp}) et l'inverse de la vitesse w_i du processeur. Donc le temps de finition du traitement est minimum quand w_i est le plus petit (donc, le processeur P_i a la plus haute vitesse) dans l'ensemble $s(w) = \{w_1, w_2, .., w_N\}$. Par conséquent, pour minimiser le coût total de calcul, les processeurs sont ordonnés, les moins chers doivent être utilisés plus. Cela conduit à un classement spécial pour la séquence des processeurs $c_1 w_1 \leq c_2 w_2 \leq \ldots \leq c_N w_N$ (ou la séquence de la distribution de charge). Le coût total de calcul pour le travail entier est donné par l'expression suivante :

$$C_{Total} = \sum_{i=1}^{N} (\alpha_i C_i w_i T_{cp})$$

avec

- α_i est la fraction du travail entier associée au i^{me} processeur P_i ;
- C_i est le coût de calcul du processeur P_i ;
- w_i l'inverse de vitesse du processeur P_i ;
- T_{cp} Le temps nécessaire pour que le processeur P_i traite le travail entier quand $w_i = 1$.

Le coût total de calcul C_{Total} peut être réduit à la valeur la plus basse possible comme :

$$C_{Total} = \min_{i=1,2,\ldots,N}(C_i w_i T_{cp})$$

Dans [102] les auteurs présentent deux algorithmes :

1. **Minimisation de coût** (Cost Minimiser)
   La fonction objectif est la suivante : $\min^{new}_{T_f \leq T_f} Cost$. Cet algorithme alloue de la charge de travail à chaque processeur (du moins cher au plus cher en terme de rapport *coût* sur *charge* en respectant la borne supérieure sur le temps de fin) jusqu'à que toute la charge de travail soit allouée. Donc, la contrainte de temps de la fin des travaux est satisfaite tout en minimisant le coût.
2. **Minimisation de temps de la fin** (Finish time minimizer)
   Objectif $\min_{C \leq C_{given}} T_f$. Cet algorithme consiste à trier les processeurs par ordre croissant du coût de calcul par charge de travail $(c_1 w_1, c_2 w_2, \ldots, c_N w_N)$. Le temps de terminaison est une fontion décroissante sur le sous-ensemble des processeurs $i = 1, 2, ..r$. Enfin, une heuristique classique du type recherche dichotomique est utilisée pour trouver la valeur maximale de r telle que le coût total soit inférieur à la valeur d'un coût donné C_{given}.

3.5 Divers critères

3.5.1 Minimisation du flux pondéré

Papadaki et Powel [82] illustrent l'importance de l'utilisation des résultats structurels dans les algorithmes de programmation dynamique. Les auteurs cherchent à résoudre le problème d'optimisation du flux des clients à une station service de façon à se rapporcher le plus possible des solutions optimales. Les clients arrivent aléatoirement au poste et attendent d'être servis. Le coût pour eux est la somme du coût d'attente et du coût de service. Le service de clients est exécuté en groupe d'une capacité du service fixe. Les auteurs exposent une approche théorique des fonctions de coût et montrent le caratère monotone des valeurs de ces fonctions objectif. Les auteurs ont recours à un algorithme monotone adaptatif de programmation dynamique qui conserve la monotonie des évaluations à chaque itérations, pour se rapprocher des valeurs de la fonction de coût optimal.

Ramabhatta et al. [91] présente un programme linéaire en variables entières mixtes pour trouver les machines constituant les cellules telles que le flux intercellulaire des produits entre les machines dans des cellules différentes soit minimal.

Arnaud Legrand et al. [73] s'intéressent au problème de l'ordonnancement de requête de comparaisons de motifs et de bases de données biologiques. Ils montrent expérimentalement que ce problème peut être traité comme un problème de tâches divisibles. Dans ce contexte, ils proposent un algorithme en temps polynomial qui produit un ordonnancement minimisant le flot pondéré maximal sur un ensemble de machines de caractéristiques non corrélées, et ce quand les dates d'arrivée des tâches sont connues à l'avance (modèle off-line).

3.5.2 Réduction de la consommation d'énergie

Actuellement, la communauté de la recherche produit un effort important pour les problèmes de conception des systèmes consommant peu d'énergie. Réduire la consommation d'énergie d'un système informatique revêt des aspects nécessairement multiples impliquant des composants séparés comme le CPU, la mémoire et les Entrées/Sorties. Une cible évidente pour réduction d'énergie est le processeur.

La structure d'échelle de voltage dynamique (Dynamic Voltage Scaling (DVS)) vise à ajuster dynamiquement le voltage et la fréquence du processeur afin de réduire sa consommation d'énergie. Ce domaine est devenu récemment un enjeu de recherche majeur.

Aydin et al. [7] adressent l'ordonnancement *"power-aware"* des tâches périodiques pour réduire la consommation d'énergie de l'UCT (Unité Central de Traitement) dans les systèmes en temps réel durs à travers un DVS. Leur solution pour l'ordonnancement inter-tâches inclus trois niveaux :

1. Une solution statique (off-line) suppose que chaque tâche présente son travail dans le pire cas au processeur, à chaque instant (pour chaque arrivée). Dans ce cas, les auteurs montrent que la vitesse qui minimise la consommation totale de l'énergie correspond à l'utilisation du système ($U_{tot} = \sum_i^n \frac{C_i}{P_i} \leq 1$) sujet à la contrainte de la vitesse de CPU minimum.
2. Un mécanisme de réduction de la vitesse dynamique (on-line) pour réguler l'énergie en l'adaptant au travail courant. Cet algorithme est basé sur la détection des premières dates de fin des tâches et ajuste la vitesse des tâches rapides pour fournir des économies d'énergie tout en respectant la date d'échéance (deadline time).
3. Un mécanisme adaptatif et spéculatif de l'ajustement dynamique de la vitesse (on-line) pour anticiper les premières dates de fin des futures exécutions en utilisant l'information de la moyenne cas de d'exécution.

Toutes ces solutions assurent que toutes les échéances sont respectées.

3.5.3 Minimisation de nombre des tâches en retard

Sitters [101] traite le problème de minimisation du nombre de tâches en retard sur un système de processeurs parallèles non liés. L'auteur formule le problème comme ($R|pmtn|\sum U_j$) et montre que ce problème a une solution pseudo-polynomiale.

4 Synthèse

Nous avons défini dans ce chapitre l'ordonnancement comme la répartition d'un ensemble de travaux sur des machines ou des ateliers de production en respectant au mieux un ensemble de contraintes et en cherchant à optimiser un ou plusieurs objectifs. Ainsi nous avons décrit les différents domaines d'application de l'ordonnancement (dans l'informatique, dans l'industrie, dans les systèmes temps réel). En tenant compte de la diversité des problèmes d'ordonnancement, nous avons classés ces problèmes selon les trois champs $\alpha/\beta/\gamma$.

Les algorithmes d'ordonnancement sont très souvent montrés comme NP-complets. Parmi les algorithmes d'ordonnancement, les algorithmes d'ordonnancement de liste sont très souvent

employés et provoquent une bonne qualité d'ordonnancement avec une faible complexité. Ainsi, nous avons présenté quelques algorithmes d'ordonnancement de l'état de l'art, pour optimiser divers critères (makespan, coût total d'exécution, etc.).

Dans la suite de ce travail nous présentons la problématique et les hypothèses de notre travail de recherche.

Chapitre 4

Formalisation du problème

Ce chapitre est consacré à la formalisation d'un problème d'optimisation de la constitution d'une plate-forme hétérogène. Il s'agit du problème qui a motivé les travaux de cette thèse, c'est-à-dire l'optimisation de la taille de la plate-forme en terme de coût économique total sous la contrainte d'un certain niveau de performance exprimé sous la forme d'un débit s. Dans une première partie nous décrivons le modèle de la plate-forme utilisée dans ce travail. Dans une deuxième partie nous introduisons les notations utilisées dans la suite de cette thèse. Nous introduisons la problématique générale du problème d'optimisation dans une troisième partie. Dans la quatrième partie de ce chapitre, nous identifions les différents cas que nous allons étudier pour résoudre notre problème. Avant de conclure en fin de chapitre, nous donnons la manière dont la plate-forme doit être piloter pour atteindre le niveau de performance pour lequel elle est conçue.

1 Architecture et Modèle

Comme nous l'avons vu au chapitre 1, les contextes de grille et de micro-usine peuvent s'avérer similaires en terme de modèle pour la résolution de notre problème d'optimisation.

La plate-forme que nous présentons dans ce travail est une plate-forme hétérogène composée de P ressources communiquantes entre elles par des liens de communication. Le graphe $G_p = (P, E)$ (figure 4.1) représente la plate-forme, donc les ressources sur lesquelles seront exécutées les travaux à réaliser. Les nœuds représentent les ressources P_m de la plate-forme. Les arrêtes de E représentent les liens de communication entre les ressources. La micro-usine est représentée par la même description avec des cellules en lieu et place des ressources et des organes de transport pour les liens de communication.

La plate-forme traite un lot de I tâches identiques non préemptives. Le lot est défini comme un ensemble de N instances $G_A^{(N)}$ d'un même travail G_A. Le travail G_A est un graphe orienté acyclique (Directed Acyclic Graph (DAG)) composé de tâches et de dépendances, qui constituent l'ensemble des arcs entre les tâches. Or, comme les bibliothèques utiles à l'exécution des tâches ne sont pas présentes sur toute la plate-forme, une tâche ne peut s'exécuter que sur certaines ressources. Dans ce cas, les ressources sont dites dédiées ou mono-tâche.

Toutes les tâches de G_A peuvent être toutes de même type ou de types différents. L'exemple de la figure 4.2 montre le graphe de tâches. Un exemple de plate-forme est décrit par le tableau 4.1.

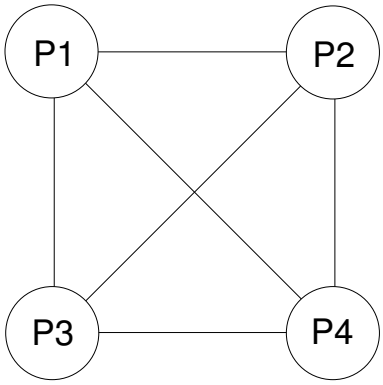

Figure 4.1 – Graphe de plate-forme

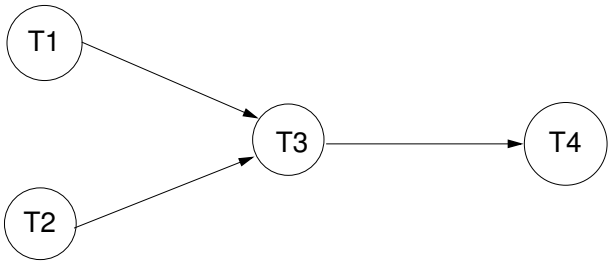

Figure 4.2 – Graphe de tâches

		p_1	p_2	p_3	p_4
type	T_1	200	∞	∞	∞
	T_2	∞	100	∞	∞
	T_3	∞	∞	200	∞
	T_4	∞	∞	∞	300

Tableau 4.1 – Matrice de description de la plate-forme donnée à la figure 4.1

Tous les nœuds de la plate-forme ne sont pas toujours capables d'exécuter tous les types de tâches. Chaque tâche est alors traitée seulement par un sous-ensemble des ressources. Les conditions d'exécution de chaque type de tâche sont données dans la matrice de coût d'exécution décrivant la plate-forme, comme montré dans le tableau 4.1. Dans ce tableau, les valeurs représentent la quantité de tâches exécutées par unité de temps sur la ressource courante. Le signe infini (∞) indique que la tâche décrite dans la ligne correspondante ne peut pas être exécutée sur la ressource de la colonne.

Comme mentionné précédemment, cette architecture et ce modèle sont valides dans le contexte de la micro-usine et de la grille de calcul.

NB : Étant donné les temps de calcul ou de fabrication, et de la taille des pièces à transporter, nous faisons l'hypothèse forte, mais raisonnable, que les temps de communication sont largement recouverts par les temps de calcul ou de fabrication.

2 Notations

Soient les notations suivantes :
- PT : ensemble des types des ressources ;
- M : nombre des ressources, $M = |PT|$;
- m : indice du type de ressources, $m = 1, \ldots, M$;
- K : nombre des résultats de sortie ;
- k : numéro du résultat calculé, $k = 1 \ldots K$
- J_k : nombre de graphes différents utilisés pour calculer le résultat k ;
- j_k : numéro du graphe qui calcule le résultat k, $j = 1 \ldots J_k$;
- $T_{j,k}$: ensemble des types de tâches du graphe j pour le résultat k ;
- $I_{j,k}$: nombre de tâches de type différents utilisées dans le graphe j pour le produit k ;
- $i_{j,k}$: indice des types des tâches $i_{j,k} = 1, \ldots, I_{j,k}$;
- $T_{i_j,k}$: la tâche de type i du graphe j définissant le résultat k.
- $\sharp i_{j,k}$: nombre d'occurrences des tâches de type $i_{j,k}$ dans le graphe j pour le produit k ;
- $r_{i_{j,k},m}$: nombre de tâches de type $i_{j,k}$ par une unité de temps exécutées sur la machine m (débit de la machine m pour une tâche $i_{j,k}$ si celle-ci calcule la tâche $i_{j,k}$ à 100%) ;
- s_k : débit de sortie des résultats k ou le nombre d'instances des J_k graphes conduisant à un résultat k, exécutés par unité de temps.
- $s_{j,k}$: débit du graphe j qui participe au débit final s_k, nombre de résultats k par unité de temps ;
- c_m : coût économique de la ressource m ;
- $r^m_{i_{j,k}}$: quantité de tâches de type $i_{j,k}$ du graphe j_k du résultat k exécutées sur la ressource de type m, lorsqu'il s'agit de ressources partagées. Lorsque la tâche ne peut être exécutée que sur un seul type de ressources, on parle alors du débit $r_{i_{j,k}}$ de la tâche $i_{j,k}$.
- $CT(s)$: coût économique total de la plate-forme d'exécution d'un débit s.

Dans un souci de clarté, lorsqu'un résultat n'est décrit que par un seul graphe de tâches et ou qu'un seul résultat n'est calculé à la sortie, nous omettons respectivement les indices j et k dans les notations proposées dans ce paragraphe. Dans ce cas, les notations sont précisées en début des paragraphes suivants, si nécessaire.

3 Problématique

Les contraintes pesant sur les ressources outils d'usinage ont beaucoup augmenté tant de point de vue de la réduction du makespan, de la réduction de la consommation d'énergie, de la réduction de coût d'exécution, de la miniaturisation de la production et de la réduction de taille des plates-formes [104, 55, 57, 81, 69, 3, 2, 21, 67, 15, 36, 2, 13, 89, 87, 9, 34, 5, 116, 29, 14, 20, 33, 90, 68, 102, 7]. Certaines études montrent que le coût de la résolution d'un tel problème dépend essentiellement de sa taille et de la forme et de la dureté de ses contraintes [70]. Cependant, notre travail est le calcul du nombre x_m des ressources de chaque type utilisées sur

la plate-forme pour mener à bien des résultats respectant un débit d'exécution donné s afin de minimiser le coût économique de la plate-forme. Le coût économique des ressources est un paramètre de la fonction objectif exprimée par la formule suivante :

$$CT(s) = \sum_{m=1}^{i=M} c_m \cdot x_m$$

Notre travail ici est de trouver les x_m en fonction des cas de figure exposés dans la suite de ce chapitre. Nous cherchons par conséquent à répondre à la question suivante : *"Quelle plate-forme (hétérogène) pour quel débit et quelle(s) exécution(s) ?"*

Pour simplifier la lecture de la suite de notre travail, nous utilisons une terminologie valable à la fois pour la micro-usine et pour la grille. Ainsi nous parlons désormais de ressources (processeurs/cellules) et de résultats (calculs/produits).

En effet, chaque résultat est décrit par un ou plusieurs graphes de tâches (DAG). Notre hypothèse est la suivante : les ressources existent et peuvent exécuter les opérations nécessaires pour les tâches auxquelles elles sont dédiées. Donc, chaque type de tâches peut être fait par au moins un type quelconque de ressources. Dans ces conditions, savoir comment concevoir une plate-forme consiste à choisir combien de ressources de chaque type doivent être utilisées afin d'autoriser la mise en œuvre de l'exécution complète de l'ensemble des travaux sous contraintes de performance (débit).

La réponse à la question précédente consiste alors à garantir un niveau de performance en nombre de résultats finis par unité du temps pour un coût économique minimum.

La construction de la plate-forme dans ce contexte est par conséquent un problème d'optimisation pour lequelle nous proposons des pistes pour sa résolution dans des cas différents. Mais, avant de traiter le cas le plus général, il est important d'exposer des cas intermédiaires pour lesquels les solutions optimales peuvent exister. Dans les autres cas, les solutions sous optimales doivent être envisagées.

Ce qui influence le niveau de généralités du problème de l'optimisation est le fait qu'une ressource est mono ou multi-tâches. Attention nos ressources, comme souvent en calcul haute performance et dans l'industrie, réalisent leurs tâches séquentiellement, les unes après les autres. Nous entendons par mono-tâche, une ressource spécialisée pour un seul type de tâches. Cette ressource fait toujours le même type de travail. Par contre, une ressource multi-tâches peut traiter des travaux de types différents, dans un ordre arbitraire. Nous négligeons dans cette étude les temps de reconfiguration ou de changement de contexte permettant à la ressource de traiter une tâche d'un type puis d'un autre type. De plus, les résultats en sortie de la plate-forme sont définis avec un ou plusieurs graphes de tâches et il est possible d'imaginer que des résultats différents soient exécutés sur la même plate-forme, chacun soumis à des contraintes de débit en sortie. Il y a ainsi quatre paramètres binaires soit 16 combinaisons différentes (2^4). Nous présentons dans la suite ces différents cas, classés tout d'abord en quatre groupes :

1. un type de tâches par type de ressources (mono-tâche ou dédiée), un type de ressources par type de tâches, exécution d'un seul ou de plusieurs résultats, chacun d'eux défini selon un seul ou plusieurs graphes (voir tableau 4.2).

2. un type de tâches par type de ressources (mono-tâche), plusieurs ressources par type de tâches, exécution d'un seul ou de plusieurs résultats, chacun d'eux défini selon un seul ou plusieurs graphes (voir tableau 4.3).
3. plusieurs types de tâches par type de ressources (multi-tâches), un type de ressources par type de tâches, exécution d'un seul ou de plusieurs résultats, chacun d'eux défini selon un seul ou plusieurs graphes (voir tableau 4.4).
4. plusieurs types de tâches par type de ressources (multi-tâches), plusieurs ressources par type de tâches, exécution d'un seul ou de plusieurs résultats, chacun d'eux défini selon un seul ou plusieurs graphes (voir tableau 4.5).

Notons que Les "1" dans les tableaux indiquent la satisfaction des critères. Nous discutons 8 des 16 cas représentés dans les quatre tableaux dans le paragraphe 4.

Une ressource ne sait faire qu'un seul type de tâche		
ressource mono-tâche	procédé mono-graphe	un seul produit
1	1	1
1	1	0
1	0	1
1	0	0

Tableau 4.2 – Matrice d'identification des cas GI

Différentes ressources savent faire la même tâche		
ressource mono-tâche	procédé mono-graphe	un seul produit
1	1	1
1	1	0
1	0	1
1	0	0

Tableau 4.3 – Matrice d'identification des cas GII

Une ressource ne sait faire qu'un seul type de tâche		
ressource multi-tâches	procédé mono-graphe	un seul produit
1	1	1
1	1	0
1	0	1
1	0	0

Tableau 4.4 – Matrice d'identification des cas GIII

Différentes ressources savent faire la même tâche		
ressource multi-tâches	procédé mono-graphe	un seul produit
1	1	1
1	1	0
1	0	1
1	0	0

Tableau 4.5 – Matrice d'identification des cas GIV

4 Présentation des cas

Comme nous avons déjà mentionné dans le paragraphe 3, toutes les combinaisons que nous avons identifiées représentent 16 cas. Cependant, dans la suite de ce chapitre, nous n'examinons que 8 d'entre eux. Il s'agit des cas qui imposent à une ressource de ne traiter qu'un seul type de tâche. Cela se justifie pleinement dans le cas de la micro-usine dans laquelle tout changement de traitement de type de tâche entraîne un temps de reconfiguration incompatible avec les temps de traitement des tâches. Ce faisant, nous considérons que les ressources sont spécialisées. De plus, résoudre le problème initial avec la possibilité donnée à toutes les ressources de pouvoir traiter toutes les tâches est une généralisation qui complexifie grandement sa résolution. Une formalisation est faite en fin de chapitre pour un des cas autorisant les ressources à traiter plusieurs types de tâches. Cette configuration nous conduit à décrire le problème d'optimisation associé avec un programme non-linéaire en variables mixtes.

Comme nous voyons dans la suite, les 8 cas ne conduisent pas à 8 cas réellement différents. En effet, certains cas peuvent être résolus en s'inspirant des solutions déjà proposées.

4.1 Présentation et objectifs de 8 cas

Dans ce paragraphe nous présentons les 8 cas pour lesquels les ressources sont dédiées à une seul type de tâches, les tâches peuvent s'exécuter sur une ou plusieurs ressources dans le but de produire un ou plusieurs résultats différents, chacun décrit par un ou plusieurs DAG. Ceci conduit bien à définir 8 cas différents. Comme nous l'avons déjà expliqué, notre objectif est de minimiser le coût économique total de la plate-forme tout en assurant d'un débit donné s en sortie. Nous exprimons, pour chacun des cas, une fonction objectif qui représente le coût à minimiser, c'est à dire la somme des coûts unitaires des ressources strictement nécessaires.

4.1.1 Cas 1. Ressources dédiées, une tâche par ressource, un seul résultat décrit par un seul graphe

Nous cherchons une plate-forme pour laquelle les ressources sont dédiées à l'exécution d'un seul type des tâches, chaque type de tâches n'étant exécuté que par un seul type de ressources. Le seul résultat en sortie n'est décrit que par un seul graphe de tâches ($k = 1$ et $i = m$ pour simplifier).

La plate-forme à construire est constituée de ressources, toutes interconnectées entre elles (graphe complet de ressources). Le nombre de type de tâches est par conséquent égal au nombre de type de ressources ($M = I$).

Coût optimal de plate-forme : Soit x_i le nombre de ressources utiles à l'exécution d'une tâche T_i de type i que nous pouvons exprimer par la formule suivante, sachant qu'une tâche de type i peut être présente $\sharp i$ fois dans le DAG et que la ressource traite r_i de la tâche de type i par unité de temps :

$$x_i = \left\lceil \frac{\sharp i}{r_i} \cdot s \right\rceil$$

Soit c_i le coût économique de la ressource pouvant exécuter une tâche de type i. Le coût d'exécution de chaque tâche ($CT_i(s)$) est égal à la multiplication du nombre de ressources utiles à l'exécution par le coût de la ressource :

$$CT_i(s) = \left\lceil \frac{\sharp i}{r_i} \cdot s \right\rceil \times c_i$$

En additionnant sur toutes les tâches, nous obtenons le coût total d'exécution d'un débit s qui est donc exprimé selon la formule suivante :

$$CT(s) = \sum_{i=1}^{i=I} CT_i(s) = \sum_{i=1}^{I} \left\lceil \frac{\sharp i}{r_i} \cdot s \right\rceil \cdot c_i$$

Tous les termes de l'expression sont des données en entrées de notre problème. Le coût $CT(s)$ est obtenu en temps linéaire en terme de nombre de types de tâches $O(I)$.

Débit maximisant l'utilisation de la plate-forme : Il est aussi possible de trouver le débit r_{opt} qui maximise l'utilisation d'une plate-forme exécutant ce type de résultat. Ce débit est celui qui permet à toutes les ressources d'être à 100% d'utilisation. Il faut donc que $\lceil \frac{\sharp i}{r_i} \cdot r_{opt} \rceil = \frac{\sharp i}{r_i} \cdot r_{opt}$ pour tout i.

On pose k_i un entier tel que $\forall i, \frac{\sharp i}{r_i} \cdot r = k_i$. Ainsi, k_i peut alors écrire $\frac{a_i}{b_i} \times r = k_i$ avec a_i et b_i entiers tels que la fraction soit irréductible. On cherche donc la plus petite valeur pour r telle que $\frac{a_i}{b_i} \times r$ soit entier.

Or $\frac{\sharp i}{r_i} = \frac{a_i}{b_i}$. Il faut trouver la fraction irréductible pour $\frac{\sharp i}{r_i}$. Pour ce faire on doit diviser son numérateur et son dénominateur par le $PGCD$ de ces deux nombres. Nous avons donc la relation suivante :

$$\frac{\sharp i}{r_i} = \frac{\frac{\sharp i}{PGCD(r_i, \sharp i)}}{\frac{r_i}{PGCD(r_i, \sharp i)}} = \frac{a_i}{b_i}$$

On a alors les relations suivantes :

$$\sharp i = a_i \cdot PGCD(r_i, \sharp i)$$

$$r_i = b_i \cdot PGCD(r_i, \sharp i)$$

Pour un i donné on pose $r_{opt_i} = b_i$. Il s'agit de la plus petite valeur du débit pour la tâche i telle que l'utilisation des ressources utiles est à 100%. Pour tout débit, multiple entier de r_{opt_i}, les ressources traitant les tâches de type i sont utilisées également à 100%.

$$r_{opt_i} = \frac{r_i}{PGCD(r_i, \sharp i)}$$

Ainsi, pour l'ensemble des tâches i de 1 à I on a donc :

$$r_{opt} = PPCM_{1 \leq i \leq I}\Big(\frac{r_i}{PGCD(r_i, \sharp i)}\Big)$$

r_{opt} est par conséquent le plus petit débit en sortie permettant une utilisation optimale de la plate-forme. Tout multiple entier de ce débit permet également une utilisation optimale de la plate-forme [78].

4.1.2 Cas 2. Ressources dédiées, une tâche par ressource, plusieurs résultats, un graphe par résultat

Nous cherchons une plate-forme pour laquelle les ressources sont dédiées à l'exécution d'un seul type des tâches, chaque type de tâches n'étant exécuté que par un seul type de ressources. Cette plate-forme est amenée à produire K résultats ($K > 1$) différents, chacun d'eux n'étant décrit que par un seul graphe (DAG).

Comme précédemment le graphe complet de ressources regroupe autant de types de ressources M que de types de tâches I contenus dans l'union des graphes de tâches décrivant les K travaux. La plate-forme à construire est faite de ressources permettant de garantir un débit en sortie s_k pour tout k entre 1 et K. Chaque ressource i a un coût économique c_i et permet un débit d'exécution de r_i tâches de type i par unité de temps.

Soit x_{i_k} le nombre de ressources utiles à l'exécution d'une tâche de type i le résultat k à exécuter. Ce nombre s'exprime par la formule :

$$x_{i_k} = \left\lceil \frac{\sharp i_k}{r_i} \cdot s_k \right\rceil$$

Or les ressources peuvent être partagées entre les J graphes décrivant les K résultats à exécuter sur la plate-forme. Le calcul du nombre de ressource ne doit pas être calculé graphe par graphe, résultat par résultat. Ce faisant, x_i s'exprime comme suit :

$$x_i = \left\lceil \sum_{k=1}^{k=K} (\frac{\sharp i_k}{r_i} \cdot s_k) \right\rceil$$

Le coût d'exécution de chaque tâche ($CT_i(s)$) dans chaque produit est égal à la multiplication du nombre de ressources utiles par leur coût :

$$CT_i(s = s_1, s_2, ..., s_k) = \Big(\left\lceil \frac{1}{r_i} \sum_{k=1}^{k=K} \sharp i_k \cdot s_k \right\rceil \Big) c_i$$

où $\sharp i_j$ est le nombre d'occurrences des tâches de type i dans le graphe décrivant le résultat k.

En sommant sur toutes les tâches, nous obtenons le coût total d'exécution d'un débit s s'exprime selon la formule suivante :

$$CT(s) = \sum_{i=1}^{i=I} \Big(\left\lceil \frac{1}{r_i} \sum_{k=1}^{k=K} \sharp i_k \cdot s_k \right\rceil \Big) c_i$$

La complexité est ici également polynomiale en $O(IK)$ [78].

4.1.3 Cas 3. Ressources dédiées, une tâche sur différentes ressources, un seul résultat décrit par un seul graphe

Nous cherchons la plate-forme pour laquelle les ressources sont dédiées à l'exécution d'un seul type de tâche, une tâche pouvant être exécutée par différents types de ressources, chacune avec une vitesse d'exécution et un coût économique différent. Ici, un seul résultat est produit en sortie, ce résultat n'étant décrit que par un seul graphe (DAG).

Le contexte de ce problème est le suivant :

- l'ensemble des ressources dont nous disposons est ($P = \{p_m, m = 1 \dots M\}$) pour lequel chaque ressource a un coût économique c_m, et un débit r_m pour tout type de tâches i, $i = 1 \dots I$.
- l'ensemble des tâches ($T = \{T_1, T_2, ..., T_I\}$) s'exécute sur les M ressources.
- Une tâche de type i peut s'exécuter sur des ressources m de type différent, chacune avec un coût économique différent et une vitesse d'exécution différente en fonction également du type de la tâche à exécuter.

Notre but est de minimiser le coût total d'exécution de la plate-forme pour un certain débit s donné.

Soit x_{im} le nombre de ressources de type m utilisées sur la plate-forme pour exécuter une tâche de type i. Soit $CT_i(s)$ le coût associé à l'exécution de la tâche de type i sur toutes les ressources nécessaires :

$$CT_i(r_i) = \sum_{m=1}^{m=M} x_{im} c_m$$

avec r_i le débit associé aux tâches de type i :

$$r_i = \sharp i \cdot s = \sum_{m=1}^{m=M} x_{im} r_{im}$$

Nous pouvons alors décomposer le problème en sous problèmes. On trouve alors le meilleur coût économique d'exécution de chaque type de tâches sur l'ensemble des ressources, les ressources n'étant pas partagées entre les types de tâches :

$$CT_i(r_i) = \min_{1 \leq m \leq M} \Big(c_m + CT_i(w(r_i - r_{im})) \Big)$$

Pour la simplicité nous supposons que $r_i - r_{im} = y$, nous vérifions que $r_i - r_{im}$ est bien positif. Dans le cas contraire, la valeur doit être nulle. Pour cela nous introduisons la notation suivante : $w(y) = 0$ si $y \leq 0$ et $w(y) = y$ sinon.

Le coût total d'exécution d'un débit s est égal à la somme des coûts économiques pour chaque type i du graphe des tâches :

$$CT(s) = \sum_{i=1}^{I} CT_i(\sharp i \cdot s)$$

Un programme dynamique peut calculer la meilleure configuration pour la plate-forme, type par type, afin de trouver la solution optimale du coût de la plate-forme. L'algorithme de ce programme est donné dans le chapitre suivant. Sa complexité est en $O(sM)$ ou M est le nombre de ressources et s le débit donné. Il s'agit d'un algorithme de complexité pseudo-polynomiale car il dépend de s dont la valeur est codé en binaire avec un nombre logarithmique de bits [78].

4.1.4 Cas 4. Ressources dédiées, une tâche par ressource, un seul résultat décrit par différents graphes

Nous cherchons la plate-forme pour laquelle les ressources sont des ressources dédiées à l'exécution d'un seul type de tâches, chaque type de tâches n'étant exécuté que par un seul type de ressources. Ici, un seul résultat ($K = 1$) est décrit par J graphes ($J > 1$) différents.

La plate-forme à construire est faite de ressources, toutes interconnectées entre elles (graphe complet de ressources) et le nombre de type de tâches est par conséquent égal au nombre de type de ressources ($M = I_J$).

Notre but est de minimiser le coût économique total de la plate-forme pour un certain débit s donné.

Avant de chercher à résoudre le problème général, nous faisons deux approximations permettant d'alléger le problème général.

Cas 4.1 Une première approximation consiste à considérer que chaque graphe j est une boîte noire, en fait la ressource j dotée un débit r_j et d'un coût c_j, avec $j = 1 \ldots J$ (J=M).

Le problème est de trouver le nombre x_j de ressources j pour que le débit cible s soit atteint à moindre coût. On pose s_j le débit disponible pour effectuer des tâches de type j par les ressources j : $s_j = x_j \cdot r_j$. Étant donné que x_j est entier, $s \leq s_1 + \ldots + s_j + \ldots + s_J$.

Ce problème peut s'écrire sous la forme du programme linéaire suivant :

$$\begin{cases} \text{minimiser} & CT(s) = \sum_{j=1}^{J} x_j \cdot c_j \\ \text{sous la contrainte} & \sum_{j=1}^{J} x_j \cdot r_j = \sum_{j=1}^{J} s_j \geq s \end{cases}$$

Nous pouvons réécrire ce programme linéaire comme suit :

$$\begin{cases} \text{maximiser} & CT(-s) = \sum_{j=1}^{J} x_j \cdot (-c_j) \\ \text{sous la contrainte} & \sum_{j=1}^{J} x_j \cdot (-r_j) = \sum_{j=1}^{J} (-s_j) \leq (-s) \end{cases}$$

Ce problème est en fait une manière d'écrire le problème du sac à dos avec ici un poids négatif $(-s)$ avec des objets j dotés d'un poids $(-s_j)$ et d'une valeur $(-c_j)$. En conséquence, le problème initiale (cas 4.1) est un problème NP-Complet pour lequel il existe une solution qu'il est possible de calculer grâce un programme dynamique de complexité pseudo-polynômiale. Il est en effet possible de construire une solution optimale du problème du sac à dos à j variables à partir du problème à $j-1$ variables.

$$CT(r) = \min_{j=1..J} \Big(CT(w(r - r_j)) + c_j \Big)$$

avec $r = 1 \ldots s$ et la fonction $w(y) = y$ si $y > 0$, 0 sinon.

Le coût minimal de la plate-forme est donc $CT(s)$. L'algorithme de programmation dynamique fait varier r de 1 à s et une structure de données annexe permet de reconstruire la solution optimale. Cet algorithme est de complexité pseudo-polynomiale en $O(sJ)$ car il est dépendant de la valeur s, codée en binaire, c'est à dire avec un nombre logarithmique de bits.

Cas 4.2 Une deuxième approximation consiste à considérer les J graphes j définissant le résultat final. Chaque graphe j est formé de I_j tâches différentes. Toute augmentation du débit cible peut se traduire par l'ajout d'une ressource i_j pour le graphe j. Là encore, aucune tâche n'est partagée entre les graphes. Chaque ressource ajoutée est une ressource bloquante. La question est alors de savoir quelle ressource ajouter pour augmenter le débit jusqu'au débit cible. Là encore, il est possible d'imaginer un algorithme de programmation dynamique donnant la configuration optimale pour un débit donné.

Le contexte de ce problème est le suivant :

- l'ensemble des ressources dont nous disposons est $(\{P_m, m = 1 \ldots M\})$ pour lequel chaque ressource a un coût économique c_m et un débit r_m. Or toutes les tâches sont différentes. Ces tâches sont numérotées par i_j avec $i_j = 1 \ldots I_j$. Les ressources étant dédiées à un type de tâche, à tout m correspond un indice $m = i_j$ différent ;
- l'ensemble des tâches $(T = \{t_{i_j}, i = 1 \ldots I_j$ et $j = 1 \ldots J)$. La tâche t_{i_j} s'exécute sur la ressource $m = i_j$ de débit r_{i_j} et de coût c_{i_j} ;
- le débit en sortie est s ;
- le nombre d'occurrences de la tâche de type i_j dans le graphe j, est $\sharp i_j$.

Soit x_{i_j} le nombre de ressources de type $m = i_j$ permettant d'exécuter la tâche t_{i_j} tel que le débit du graphe j atteigne le débit s_j.

Le coût d'exécution d'une tâche de type i_j est égale à la multiplication de nombre de ressources utiles par le coût économique de la ressource :

$$CT_{i_j}(s_j) = x_{i_j} \cdot c_{i_j}$$

En sommant sur toutes les tâches de tous les graphes, nous obtenons le coût total d'exécution de la plate-forme :

$$CT(s) = \sum_{j=1}^{j=J} \sum_{i=1}^{i=I} CT_{i_j}(s_j) = \sum_{j=1}^{J} \sum_{i=1}^{I_j} x_{i_j} \cdot c_{i_j}$$

Le débit du graphe j ne doit pas dépasser la capacité de ressources pouvant l'exécuter, ainsi, soit la contrainte suivante :

$$\forall j = 1 \dots J \; et \; i = 1 \dots I_j \quad s_j \leq \frac{r_{i_j} \cdot x_{i_j}}{\sharp i_j}$$

Le débit total de production est limité par la somme de débit des J graphes :

$$\sum_{j=1}^{J} s_j \geq s$$

Cela correspond au programme linéaire en entier suivant :

Objectif, minimiser $CT(s)$ avec :

$$CT(s) = \sum_{j=1}^{j=J} \sum_{i}^{I_j} x_{i_j} \cdot c_{i_j}$$

sous les contraintes :

$$\begin{cases} s_j \leq \frac{r_{i_j} \cdot x_{i_j}}{\sharp i_j}, & \forall i = 1 \dots I_j, \forall j = 1 \dots J \\ \sum_{j=1}^{J} s_j \geq s \\ x_{i_j} \in \mathbb{N} \end{cases}$$

Cependant il n'y a pas une solution en temps polynomial, même avec un logiciel de résolution efficace comme CPLEX. C'est un programme linéaire mixte avec variables entières. C'est également un problème $NP-$Complet, d'autant qu'il est plus général que le problème 4.1.

Il est cependant possible de trouver encore une fois un programme dynamique permettant de calculer la solution optimale avec une complexité pseudo-polynômiale.

Idée de l'algorithme :

Pour chaque graphe $j = 1 \dots J$, il faut construire la courbe des $CT_j(s_j)$ pour s_j de 1 à s :

$$CT_j(s_j) = \sum_{i_j=1}^{I_j} \left\lceil \frac{\sharp i_j \cdot s_j}{r_{i_j}} \right\rceil \cdot c_{i_j} \tag{4.1}$$

avec $\sharp i_j$ le nombre de tâches de type i_j dans le graphe j et r_{i_j}. Ainsi il faut vérifier si le débit de la ressource $m = i_j$ lorsqu'elle exécute la tâche de type i_j est de 100% de son temps (ce qui est toujours le cas ici).

Soit deux graphes j' et j'' différents avec $j', j'' \leq J$. L'idée est de calculer pour tout débit $s_{j',j''}$ de 0 à s le coût de la plate-forme atteignant ce débit avec $s_{j',j''} = s_{j'} + s_{j''}$, $s_{j'}$ et $s_{j''}$ allant de 0 à s. Pour cela, il faut parcourir la partie triangulaire supérieure d'un espace 2D de taille $(s+1)^2$ qui couvre toutes les combinaisons pour les valeurs respectives de $s_{j'}$ et $s_{j''}$. Cet espace est parcouru selon des diagonales correspondant à $s_{j',j''}$ constant et au cours duquel sont calculés les coûts pour toutes les configurations $s_{j'}$ et $s_{j''}$ possibles pour ce débit $s_{j',j''}$. Seul le coût minimal est conservé dans un tableau $CT_{j',j''}[s_{j',j''}]$ car il est le seul intéressant pour notre problème. Le débit $s_{j'}$ est également conservé afin de permettre la reconstruction des solutions optimales ainsi calculées. Cela peut être fait également dans un tableau 1D $S_{j',j''}[s_{j',j''}]$. La valeur de $s_{j''}$ peut être recalculée facilement. L'existence des tableaux $CT_{j',j''}$ et $s_{j',j''}$ permettra de considérer les configurations avec d'autres graphes par la suite.

On peut effectuer cette opération deux graphes par deux graphes, puis quatre graphes par quatre graphes jusqu'au nombre de final de graphes, en réutilisant les valeurs des tableaux $CT_{j',j''}$ calculés précédemment. Il est également possible d'associer un seul graphe avec deux graphes, etc. On trouve ainsi le coût optimal de la plate-forme permettant à J graphes différents de cohabiter. L'ordre de traitement des graphes n'a pas d'importance à cause de la propriété d'associativité de la fonction *min()*.

La complexité associée est en $O(s^2J + sJ)$ avec s le débit à atteindre et J le nombre de graphes utilisables. Il s'agit bien d'une complexité pseudo-polynômiale.

Cas 4.3 C'est le cas général de notre problème, où les tâches ne sont pas liées aux graphes. Ici, les graphes partagent les tâches. Une ressource de type m exécute toutes les tâches de type i de tous les graphes [78].

Le contexte de ce problème est le suivant :
- L'ensemble des ressources dont nous disposons est $(\{P_m, m = 1 \dots M\})$ pour lequel chaque ressource a un coût économique c_m et un débit r_m pour tout type de tâches $i = 1 \dots |T|$.
- L'ensemble des tâches $T = \cup_{j=1}^{J} T_j = \{t_i, i = 1 \dots |T|\}$. Pour simplifier on pose i l'indice de la ressource traitant la tâche i.
- Le débit en sortie est s.
- Le nombre d'occurrences des tâches de type i dans le graphe j est $\sharp i_j$.

On pose x_i le nombre des ressources pouvant exécuter une tâche de type i (x_i est un entier) :

$$x_i = \left\lceil \frac{1}{r_i} \sum_{j=1}^{J} \sharp i_j \cdot s_j \right\rceil$$

soit $CT_i(s)$ le coût d'exécution de la tâche de type i sur la ressource $m = i$. Ce coût est égale à la multiplication du nombre des ressources utiles par le coût économique de la ressource :

$$CT_i(s = s_1, s_2, ..., s_k) = x_i \cdot c_i$$

En additionnant sur toutes les tâches, nous obtenons le coût total d'exécution d'un débit s :

$$CT(s) = \sum_{i=1}^{|T|} \left\lceil \sum_{j=1}^{J} \frac{\sharp i_j \cdot s_j}{r_i} \right\rceil \cdot c_i \tag{4.2}$$

Or, on ne connaît pas les s_j tel que $s = \sum_j s_j$ avec $CT(s)$ minimal.

Le problème s'écrit là encore sous la forme du programme linéaire mixte en entier suivant :

Objectif, minimiser :

$$CT(s) = \sum_{i=1}^{|T|} x_i \cdot c_i$$

sous les contraintes :

$$\begin{cases} \forall i \ x_i \cdot r_i \geq \sum_{j=1}^{J} (\sharp i_j \cdot s_j) \\ \sum_{j=1}^{J} s_j \geq s \\ x_{i_j} \in \mathbb{N} \end{cases}$$

On peut résoudre ce programme linéaire en entier avec CPLEX, sans gage de succès. Par contre, il est possible de proposer des solutions heuristiques sous optimales. Une possibilité est le recours à l'algorithmique génétique pour recherche la meilleure décomposition $s = s_1 + s_2 + \ldots + s_j + \ldots + s_J$ minimisant le coût de la plate-forme.

4.1.5 Cas 5. Ressources dédiées, une tâche par ressource, plusieurs résultats chacun défini par différents graphes

Nous cherchons une plate-forme hétérogène pour laquelle les ressources sont des ressources dédiées (mono-tâche) au traitement d'un seul type de tâches, une tâche ne peut pas être exécutée que sur un seul type de ressource. Cette plate-forme est amenée à produire K résultats ($K > 1$) différents, chacun d'eux étant décrit par J_k graphes différents ($J > 1$ et $k = 1 \ldots K$) .

Le contexte de ce problème est le suivant :

- $\sharp i_{j,k}$: le nombre d'occurrences des tâches de type $i_{j,k}$ dans le graphe j pour le résultat k ;
- $s_{j,k}$: le débit du graphe j pour le produit k ;
- s_k : le débit total de production du produit k ;

Cela revient donc à calculer la quantité de ressources optimale pour chacun des K résultats, ce qui nous ramène au cas 4 étudié précédemment. Comme dans ce cas précédent, la difficulté est de gérer l'absence d'équité entre les graphes définissant un même résultat. Par contre, les débits des résultats sont donnés, ce qui permet de connaître les quantités de travail dans certains

cas. Il convient là encore de faire les mêmes hypothèses simplificatrices données pour les cas 4.1 et 4.2. Dans ces deux cas, aucun partage ne se fait entre les traitements de type différents. Les quantités de travail peuvent donc être calculées de la même manière que dans le cas 4, le fait d'ajouter des résultats supplémentaires à traiter en parallèle ne change rien. Ils peuvent être traités indépendemment les uns des autres.

Par contre, dans le cas le plus général, lorsque des tâches peuvent être partagées entre différents graphes et même ici entre différents résultats, une solution heuristique peut être envisagée, pour les mêmes raisons que celle évoquée dans le cas 4.3.

4.1.6 Cas 6. Ressources dédiées, une tâche sur différentes ressources, un seul résultat, chacun défini par différents graphes

Nous cherchons la plate-forme pour laquelle les ressources sont dédiées à l'exécution d'un seul type de tâches, une tâche pouvant être exécutée sur différentes ressources, chacune avec une vitesse d'exécution et un coût économique différents. Ici, un seul résultat est produit en sortie, ce résultat est décrit par J différents graphes.

Le contexte de ce problème est le suivant :

- l'ensemble des ressources dont nous disposons est $(P = \{p_m, m = 1 \dots M\})$ pour lequel chaque ressource a un coût économique c_m, et un débit r_{im} pour tout type de tâches i_j, $i_j = 1 \dots I_J$ et $j = 1 \dots J$.
- l'ensemble des tâches $(T = \bigcup_{i=1}^{i=I} \{T_1, T_2, ..., T_I\})$ s'exécute sur les M ressources.
- une tâche de type i peut s'exécuter sur des ressources m de type différents, chacune avec un coût économique différent et une vitesse d'exécution différente en fonction également du type de la tâche à exécuter.

Ce cas est à nouveau très semblable au cas 4 que nous venons de présenter. Cependant, il existe une différence importante car nous ne savons pas quelle ressource traite quelle tâche. Or le cas 3 nous aide pour connaître la composition des ressources optimisant le coût économique pour un débit donné et pour une tâche donnée. Ce faisant, ce cas est à rapprocher du cas 4 en reprenant les mêmes étapes mais en modifiant le calcul du coût.

Comme avec les hypothèses du cas 4.1, si tous les graphes sont des boîtes noires, mais que chacune peut être traitée par différentes ressources, cela conduit à trouver la meilleure configuration grâce à un programme dynamique similaire à celui imaginé pour résoudre le cas 3. Cette remarque vaut aussi pour les cas correspondant aux hypothèses données dans le cadre des cas 4.2 et 4.3. Ainsi, l'évaluation du coût d'une tâche ne se fait plus en multipliant le nombre de ressources par leur coût unitaire, mais en recherchant la meilleure configuration par un programme dynamique, type cas 3. Si la solution heuristique est faite grâce à l'algorithmique génétique, l'évaluation des individus devra se faire de la même manière.

4.1.7 Cas 7. Ressources dédiées, une tâche sur différentes ressources, plusieurs résultats chacun défini par différents graphes

Nous cherchons la plate-forme pour laquelle les ressources sont dédiées à l'exécution d'un seul type des tâches, une tâche pouvant être exécutée sur différentes ressources, chacune avec

une vitesse d'exécution et un coût économique différent. Cette plate-forme est amenée à produire K résultats ($K > 1$) différents. Chacun d'eux est décrit par J_k graphes différents ($J_k > 1$).

Le contexte de ce problème est le suivant :

- l'ensemble des ressources dont nous disposons est ($P = \{p_m, m = 1 \dots M\}$) pour lequel chaque ressource étant dotée par un coût économique c_m, par un débit r_{im} pour tout type de tâches i, $i = 1 \dots I$.
- l'ensemble des tâches ($T = \bigcup_{k=1}^{k=K} \bigcup_{j=1}^{j=J} \{T_1, T_2, ..., T_{I_{J,K}}\}$) s'exécute $T_{i_{j,k}}$ sur les M ressources.
- une tâche de type i peut s'exécuter sur des ressources m de type différents, chacune avec un coût économique différent et une vitesse d'exécution différente en fonction également du type de la tâche à exécuter.

Là encore, nous sommes dans un cas de figure qu'il est possible de résoudre en reprenant les principes avancés par le cas 6 précédent, lui même basé sur le cas 4. Lorsque nous faisons des hypothèses sur l'absence de partage (cas 4.1 et 4.2). Le fait de produire un ou plusieurs résultats ne change pas le problème. Dans le cas du partage des ressources entre les différents résultats et les différents graphes, une méthode heuristique est nécessaire et s'accompagne, pour son évaluation, du calcul des configurations optimales comme dans le cas 3, à chaque fois qu'un coût est à calculer pour un type de tâches donné à un débit donné. Nous sommes donc bien dans la même situation que dans le cas 6 précédent et par conséquent dans la situation du cas 4.

4.1.8 Cas 8. Ressources dédiées, une tâche sur différentes ressources, différents résultats décrits par un seul graphe

Nous cherchons la plate-forme pour laquelle les ressources sont dédiées à l'exécution d'un seul type de tâche, une tâche pouvant être exécutée par différents types de ressources, chacune avec une vitesse d'exécution pour un coût économique différent. Cette plate-forme est amenée à produire K résultats différents ($K > 1$) , chacun d'eux étant décrit par un seul graphe (DAG) ($J = 1$).

Pour résoudre ce cas de figure, un débit s_k est imposé pour chacun des K résultats ($k = 1 \dots K$). Que les différents graphes décrivant les résultats partagent des tâches ou non, il est possible de connaître la quantité de travail associé à chaque type de tâche. Il est alors possible de résoudre ce cas de figure comme nous l'avons fait pour le cas 3, en examinant tous les types de tâches et en sommant les coûts des ressources associées. La meilleure configuration pour chaque type de tâche en fonction des ressources capables de la traiter est calculée par le même type de programme dynamique que dans le cas 3.

4.2 Généralisation à un cas utilisant des ressources non dédiées

4.2.1 Cas 9. Ressources non dédiées, une tâche sur différentes ressources, un seul résultat décrit par un seul graphe

Nous cherchons la plate-forme pour laquelle les ressources ne sont pas dédiées à l'exécution d'un seul type de tâche, une tâche pouvant être exécutée par différents types de ressources, chacune avec une vitesse d'exécution et un coût économique différents. Ici, un seul résultat

est produit en sortie, ce résultat n'étant décrit que par un seul graphe (DAG). Les ressources partagent donc leur performance entre plusieurs types de tâches.

Le contexte de ce problème est le suivant :
- l'ensemble des types de ressources dont nous disposons est ($P = \{p_m, m = 1 \dots M\}$) pour lequel chaque ressource a un coût économique c_m, un débit r_{im} pour tout type de tâches i, $i = 1 \dots I$.
- l'ensemble des types de tâches ($T = \{T_1, T_2, ..., T_I\}$) s'exécute sur les M ressources.
- une tâche de type i peut s'exécuter sur des ressources de type différents, chacune avec un coût économique différent et une vitesse d'exécution différente en fonction également du type de la tâche à exécuter.

Notre but est de minimiser le coût total de la plate-forme pour un certain débit s donné.

Coût optimal de la plate-forme Soit x_m le nombre de ressources de type m utilisées sur la plate-forme pour exécuter un ou plusieurs types de tâches. Soit $CT(s)$ le coût d'une telle plate-forme :

$$CT(s) = \sum_{m}^{M} x_m \cdot c_m$$

Caractérisons les contraintes associées à l'exécution des instances du graphe de tâches au débit s donné. Le nombre de tâches de type i à traiter par unité de temps est r_i :

$$r_i = \sharp i \cdot s$$

avec $\sharp i$ le nombre de tâches de type i dans le graphe de tâches.

Or si r_i^m est le nombre moyen de tâches de type i traitées par unité de temps par les ressources de type m, sachant qu'on en compte x_m, le débit r_i peut alors d'écrire :

$$r_i = \sum_{m=1}^{M} x_m \cdot r_i^m$$

De plus, pour toutes les ressources m, le taux d'utilisation ne doit pas dépasser 100%. On a alors la contrainte suivante, sachant que r_{im} est le débit observé sur la ressource m pour le traitement des tâches de type i, 100% du temps :

$$\forall m = 1 \dots M : \sum_{i=1}^{I} \frac{r_i^m}{r_{im}} \leq 1$$

Ceci nous conduit à l'écriture du programme non-linéaire, avec variables entières suivant :

$$Objectif : CT(s) = min \sum_{m} x_m \cdot c_m$$

Sous les contraintes

$$\begin{cases} \forall m = 1 \dots M \sum_{i=1}^{I} \frac{r_i^m}{r_{im}} \leq 1 \\ \forall i = 1 \dots I \sum_{m=1}^{M} x_m \cdot r_i^m = \sharp i \cdot s \\ avec \quad r_i^m \geq 0 \\ x_m \in \mathbb{N} \end{cases}$$

Comme le coût de communication et le temps de changement sont négligeables, on suppose une répartition en moyenne sur toutes les ressources de même type de l'exécution des tâches de même type.

La résolution d'un tel système ne peut se faire directement. Il conviendrait de le transformer, sans que des solutions puissent être apportées en temps raisonnable, étant donné le fait que ce problème est plus général que des problèmes déjà difficiles. Des solutions heuristiques approchées devront être imaginées.

5 Pilotage de la plate-forme

Nous abordons dans ce chapitre les aspects liés au dimensionnement de la plate-forme permettant d'atteindre un certain niveau de performance, débit en sortie. L'ordonnancement des instances de toutes les tâches sur la plate-forme permettant que le débit cible soit atteint n'est pas un problème difficile.

En effet, dans le cas où les types de tâches ne peuvent s'exécuter que sur un type de ressource et qu'un type de ressource ne peut traiter qu'un seul type de tâche, un ordonnancement de type "*round-robin*" doit garantir le bon débit en sortie étant donné que le nombre de ressource a été prévue en conséquence.

Dans le cas où les types de tâches ne peuvent s'exécuter que sur plusieurs types de ressource et qu'un type de ressource ne peut traiter qu'un seul type de tâche, un ordonnancement de type liste "*EFT*" doit également garantir le niveau de performance pour lequel elle a été conçue.

6 Synthèse

Dans ce chapitre nous avons décrit la plate-forme et le graphe d'application utilisés dans cette étude. Notre plate-forme est une plate-forme hétérogène dont les nœuds représentent les ressources et les arêtes représentent les liens de communications entre les ressources. Le graphe d'application est un graphe acyclique ("*intree*"), dont les nœuds représentent les tâches à exécuter et les arêtes représentent les dépendances entre les tâches.

Nous avons présenté la problématique de ce travail, qui consiste à répondre à la question : *quelle micro-usine pour quelle production ?*. Dans ce travail nous avons considéré 4 variables qui influencent le niveau de généralité du problème d'optimisation. Ces variables sont : les ressources dédiées ou non, Les différentes ressources ne savent exécuter qu'un seul type de tâche ou savent en faire plusieurs, un calcul selon un seul ou plusieurs graphes, un ou plusieurs résultats simultanément.

De toutes les combinaisons de ces variables, nous avons identifié seize cas d'étude. Pour des raisons liées au réalisme d'un des cas d'utilisation de ce travail, la micro-usine, seul 8 cas sont détaillés. Ces 8 cas correspondent au cas où les ressources sont dédiées au traitement d'un seul type de tâches. Cependant, un neuvième cas plus général a été formalisé même si nous ne donnons pas de solution pour le traiter, étant donné sa difficulté.

Dans le chapitre suivant, nous écrivons les algorithmes proposés ici.

Chapitre 5

Résolution des cas principaux

Ce chapitre est consacré à la mise en œuvre algorithmique des quatre cas mis en évidence dans le chapitre précédent. En effet, dans le chapitre 4, nous avons caractérisé les 4 cas permettant de résoudre le problème du dimensionnement optimal d'une plate-forme hétérogène lorsque les ressources sont dédiées au traitement d'un type de tâches donné. Ici, nous présentons les algorithmes permettant de calculer le coût associé à ces 4 types de dimensionnement. Des pistes sont données dans le cas où le calcul du dimensionnement est heuristique.

Les quatre premiers paragraphes présentent ces quatre cas dans l'ordre dans lequel ils ont été décrits dans le chapitre précédent. Nous concluons dans la cinquième section.

1 Calcul de coût de la plate-forme dans le cas 1

Dans ce cas, nous cherchons une plate-forme pour laquelle les ressources sont dédiées au traitement d'un seul type de tâches (mono-tâche), chaque type de tâches n'étant traité que par un seul type de ressources. Le seul résultat en sortie n'est décrit que par un seul graphe de tâches.

Dans ce cadre nous conduisons deux expériences :

Expériences 1 : Dans cette expérience nous prenons les paramètres suivants :

- Le graphe d'application est constitué de quatre types des tâches (T_1, T_2, T_3, T_4) ;
- Le nombre d'occurrences de chaque type des tâches (T_1, T_2, T_3, T_4) dans le graphe est respectivement de 3, 2, 2 et 3 (voir la figure 5.1) ;
- La plate-forme est constituée de quatre types de ressources (P_1, P_2, P_3, P_4) ;
- Nous rappelons l'hypothèse de ce cas selon laquelle chaque ressource ne traite qu'un seul type de tâches. Les vitesses de traitement sont différentes et données par le tableau 5.1 ;
- Pour chaque ressource (P_1, P_2, P_3, P_4), le coût économique est respectivement de $c_1 = 15$€, $c_2 = 20$€, $c_3 = 25$€, et $c_4 = 45$€ ;
- Le débit total de production est de $s = 500$ résultats par unité de temps.

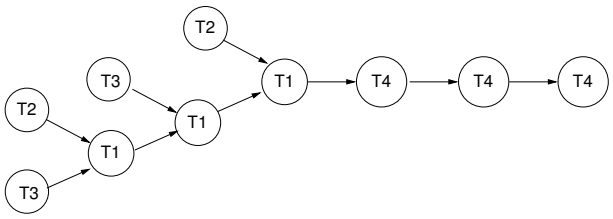

Figure 5.1 – Cas 1 : graphe de tâches

		p_1	p_2	p_3	p_4
Type	T_1	200	∞	∞	∞
	T_2	∞	100	∞	∞
	T_3	∞	∞	200	∞
	T_4	∞	∞	∞	300

Tableau 5.1 – Matrice de performance des ressources, expérience 1, cas 1

Notre but est de minimiser le coût économique $CT(s)$ d'une plate-forme hétérogène produisant s résultats par unité de temps. Comme précédemment, ce coût est donné par la formule suivante :

$$CT(s) = \sum_{i=1}^{I} \left\lceil \frac{\sharp i}{r_i} \cdot s \right\rceil \cdot c_i$$

Les résultats de l'expérience montrent que, le coût total d'exécution d'un débit s de 500 tâches par unité de temps est $CT(500) = 670$€. De plus, la plate-forme est constituée de 28 unités de ressources. Parmi elles, 8 ressources de type P_1, 10 ressources de type P_2, 5 ressources de type P_3 et 5 ressources de type P_4.

Le débit maximal (r_{opt}) qui optimise l'utilisation de la plate-forme est égal à 200 tâches par unité de temps.

Expérience 2 : Dans cette expérience nous prenons les paramètres suivants :

- Le graphe d'application est constitué de quatre types des tâches (T_1, T_2, T_3, T_4) ;
- Le nombre d'occurrences de chaque type des tâches (T_1, T_2, T_3, T_4) dans le graphe est respectivement de 3, 2, 2 et 3 (voir la figure 5.1) ;
- La plate-forme est constituée de quatre types de ressources (P_1, P_2, P_3, P_4) ;
- Comme précédemment chaque ressource ne traite qu'un seul type de tâches. Les vitesses de traitement sont différentes et données par le tableau 5.2 ;
- Pour chaque ressource (P_1, P_2, P_3, P_4), le coût économique est respectivement de $c_1 = 15$€, $c_2 = 20$€, $c_3 = 25$€, et $c_4 = 45$€ ;
- Le débit total de production est de $s = 500$ résultats par unité de temps.

Notre but est de minimiser le coût économique $CT(s)$ d'une plate-forme hétérogène produisant s résultats par unité de temps. Ce coût est donné par la formule suivante :

$$CT(s) = \sum_{i=1}^{I} \left\lceil \frac{\sharp i}{r_i} \cdot s \right\rceil \cdot c_i$$

		p_1	p_2	p_3	p_4
Types	T_1	350	∞	∞	∞
	T_2	∞	200	∞	∞
	T_3	∞	∞	400	∞
	T_4	∞	∞	∞	350

Tableau 5.2 – Matrice de performance des ressources, expérience 2, cas 1

Les résultats de l'expérience montrent que, le coût total d'exécution d'un débit $s = 500$ tâches par unité de temps est $CT(s) = 475$€.

La plate-forme est cette fois constituée de 18 unité de ressources. Parmi elles nous comptons 5 ressources de type P_1, 5 de type P_2, 3 de type P_3 et 5 de type P_4.

Le débit optimal (r_{opt}) est maintenant de 1400 tâches par unité de temps.

Résultats : Pour rendre compte de l'évolution du coût de la plate-forme en fonction du débit en sortie, nous traçons le coût total d'exécution en fonction de s sur l'intervalle de 0 à 500. La figure 5.2 montre cette courbe pour les expériences 1 et 2. Cette courbe montre bien une progression en escalier à cause de la présence de partie entière dans la définition de CT.

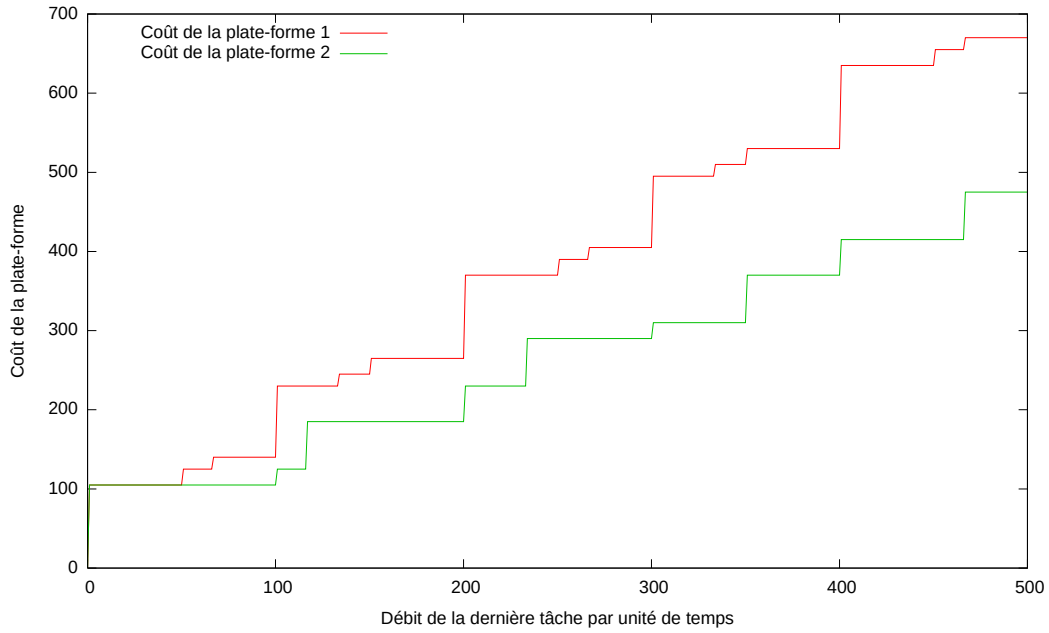

Figure 5.2 – Cas 1 : coût de deux plates-formes avec deux graphes
Fig. 5.1

Algorithme 1 : `CoutTotalCas1`(r_i : tableau d'entiers, n : integer) : integer

I : nombre de tâches différentes dans le graphe
$r_i[i]$: tableau des performances des unités de ressources.
s : le débit en sortie de la plate-forme
$coutRessources[i]$: tableau des coûts économiques des ressources
$occurTache[i]$: le nombre d'occurrences des tâches de type i dans le graphe
i : indice des tâches
$CT(s)$: le coût total d'exécution d'un débit s
début
  pour $i = 1$ *à* I **faire**
    $CT(s)+ \leftarrow \lceil (occurTache[i]/r_i[i] \cdot s) \rceil \cdot coutRessources[i]$
  finpour
  retourner $CT(s)$
fin

Algorithme 2 : `CalculdeRoptCas1`(r_i : tableau d'entiers, occurTache : integer) : integer

I : nombre de types des tâches i dans le graphe
$r_i[i]$: tableau d'entiers à partir duquel on calcul le pgcd.
$occurTache[i]$: occurrence des tâches de type i dans le graphe
$res[i]$: débit qui optimise les ressources pour la tâche de type i
début
  pour $i = 1$ *à* I **faire**
    $res[i] \leftarrow r_i[i]/pgcd(r_i[i], occurTache[i])$
  finpour
  $ropt \leftarrow ppcm(res[1..I])$
  retourner $ropt$
fin

Les algorithmes 1 et 2 expliquent respectivement le calcul du coût total d'exécution d'un débit s, et le calcul du débit optimisant le coût économique des ressources de la plate-forme.

2 Calcul de coût de la plate-forme dans le cas 2

Ce paragraphe est consacrée à une étude expérimentale du deuxième cas. Ici, nous cherchons une plate-forme pour laquelle les ressources sont dédiées à l'exécution d'un seul type de tâches (mono-tâche), chaque type de tâches n'étant exécuté que par un seul type de ressources. Par contre, cette plate-forme est amenée à produire K résultats ($K > 1$) différents, chacun d'eux n'étant décrit que par un seul graphe (DAG) ($J = 1$).

Comme dans le cas 1, nous faisons deux expériences pour lesquelles nous donnons les valeurs de $CT(s)$ correspondant à la plate-forme produisant s résultats par unité de temps à moindre coût. Nous décrivons les deux plates-formes considérées pour ces expériences.

Nous rappelons que CT est donné par la formule suivante :

$$CT(s) = \sum_i \Big(\left\lceil \frac{1}{r_i} \sum_j \sharp i_j \cdot r_j \right\rceil \Big) c_i$$

Pour accomplir notre expérience nous considérons les données suivantes :

– Le graphe d'application est constitué de trois types des tâches (T_1, T_2, T_3) ;

- La plate-forme est constituée de trois types de ressources (P_1, P_2, P_3).
- Le coût économique des unités de ressources (P_1, P_2, P_3) est respectivement 8€, 5€et 10€.
- Nous désirons calculer deux résultats selon deux graphes. Le débit du premier graphe est $s_1 = 100$ résultats par unité de temps et le débit du deuxième graphe est $s_2 = 150$ résultats par unité de temps.
- Le nombre d'occurrences des tâches (T_1, T_2, T_3) dans le premier graphe est respectivement : 3, 2 , et 2. Ce graphe est représenté sur la figure 5.3.
- Le nombre d'occurrences des tâches (T_1, T_2, T_3) dans le deuxième graphe est respectivement : 2, 2 , et 3. Ce graphe est représenté sur la figure 5.4.
- P_1 traite la tâche T_1, P_2 traite T_2 et P_3 T_3.

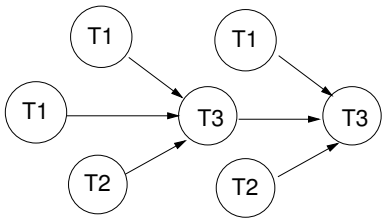

Figure 5.3 – Graphe 1 des tâches (cas 2)

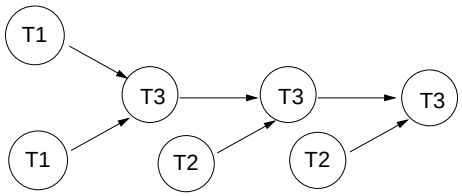

Figure 5.4 – Graphe 2 des tâches (cas 2)

Nous considérons deux plates-formes différentes :

La première plate-forme (tableau 5.3) est constituée des ressources ayant comme performances : 150 tâches par unité de temps pour P_1, 100 tâches par unité de temps pour P_2 et 200 tâches par unité de temps pour P_3.

		p_1	p_2	p_3
Type	T_1	150	∞	∞
	T_2	∞	100	∞
	T_3	∞	∞	200

Tableau 5.3 – Matrice de performance des ressources, expérience 1, cas 2

La deuxième plate-forme (tableau 5.4) est constituée des ressources ayant comme performances : 300 tâches par unité de temps pour P_1, 200 tâches par unité de temps pour P_2 et 300 tâches par unité de temps pour P_3.

Pour les deux plates-formes et avec les données d'entrées décrites ci-dessus, nous obtenons les résultats suivants :

Expérience 1 : Pour la première expérience nous obtenons les résultats suivants :

		p_1	p_2	p_3
Type	T_1	300	∞	∞
	T_2	∞	200	∞
	T_3	∞	∞	300

Tableau 5.4 – Matrice de performance des ressources, expérience 2, cas 2

- Le coût total d'une plate-forme permettant les débits s_1 et s_2, respectivement pour la production des deux résultats décrits par les deux graphes donnés précédemment, est égal à 97€ ;
- La taille de la plate-forme constituée de 13 unités de ressources : 4 unités de type P_1, 5 de type P_2 et 4 unités de type P_3 ;

Expérience 2 : Pour la deuxième expérience nous obtenons les résultats suivants :

- Le coût total d'une plate-forme permettant les débits s_1 et s_2, respectivement pour la production des deux résultats décrits par les deux graphes donnés précédemment, est égal à 61€ ;
- La taille de la plate-forme constituée de 8 unités de ressources : 2 unités de type P_1, 3 de type P_2 et 3 unités de type P_3 ;

3 Résolution du cas 3

Ce paragraphe est consacré à la résolution du cas numéro 3. Dans ce cas, les ressources sont toujours dédiées à l'exécution d'un seul type de tâches. Une tâche peut être exécutée sur des ressources de différents types avec différentes vitesses et différents coûts. Ici, un seul type de résultats est traité sur la plate-forme.

Dans les expériences que nous menons pour le cas 3, la plate-forme considérée est constituée de neuf types de ressources (P_{11}, P_{12}, P_{13}, P_{21}, P_{22}, P_{23}, P_{31}, P_{32}, P_{33}). Le coût économique de chaque type de ressources P_{i1}, P_{i2}, et P_{i3} est respectivement 10€, 20€ et 40€.

Le graphe d'application avec lequel nous produisons notre résultat est constitué de trois types de tâches (T_1, T_2, T_3). Les tâches T_i pouvant être exécutées sur les trois types de ressources P_{i1}, P_{i2} et P_{i3}. Le nombre d'occurrences T_1, T_2 et T_3 dans le graphe est respectivement 2, 3 et 2 (figure 5.5).

Le tableau 5.5 illustre le coût d'exécution d'une tâche de type i sur la ressource de type $P_{i,j}$ avec $j = 1 \ldots 3$.

Tableau 5.5 – Matrice de performance des ressources : r_{im}

		P_{11}	P_{12}	P_{13}	P_{21}	P_{22}	P_{23}	P_{31}	P_{32}	P_{33}
Types	T_1	10	50	130	∞	∞	∞	∞	∞	∞
	T_2	∞	∞	∞	20	30	100	∞	∞	∞
	T_3	∞	∞	∞	∞	∞	∞	30	50	120

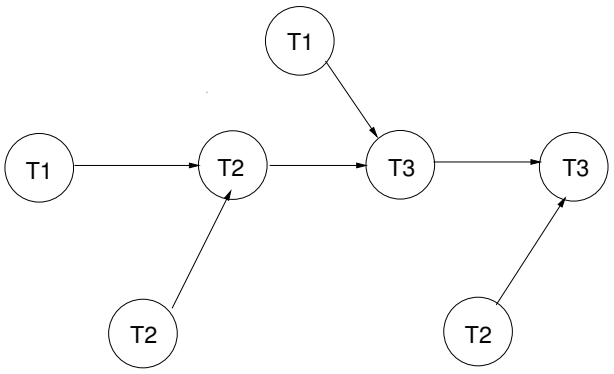

Figure 5.5 – Graphe des tâches

Lors de la sélection d'une ressource pour une tâche donnée, cette ressource doit être assignée à cette tâche. Par conséquent la plate-forme est constituée par les ressources utiles à chacun des types de tâches à traiter. Nous calculons donc la configuration de la plate-forme pour le traitement de chaque tâche i et dont le coût est $C_i(r_i)$ avec $r_i = \sharp i \cdot s$. Le coût minimal pour la plate-forme entière est donc la somme des coûts minimaux pour tous les types de tâches : $CT(s) = \sum_{i=1}^{I} C_i(\sharp i \cdot s)$.

Comme nous l'avons vu dans le chapitre précédent, nous recourons à un algorithme de programmation dynamique pour résoudre ce problème. Grâce à une expression récursive, le coût d'exécution de chaque tâche de type i est le suivant en reprenant les notations du chapitre précédent :

$$C_i(r_i) = \min_{1 \leq m \leq M} \Big(c_m + C_i(w(r_i - r_{im})) \Big)$$

Pour la raison de simplicité, nous donnons dans ce paragraphe des expériences numériques pour la meilleure configuration pour les ressources de traitement de la première tâche.

L'algorithme 3 permet de trouver la taille du plus grand pas d'itérations possible ainsi que le nombre minimal d'itérations nécessaires pour l'algorithme recherchant le coût optimal des ressources sur l'intervalle de 0 à r_i pour l'exécution de tâches de type i.

L'algorithme 4 calcule le coût minimal pour le traitement de la tâche i en fonction du débit de traitement à atteindre pour cette tâche. Comme nous l'avons expliqué précédemment, cet algorithme est un algorithme de programmation dynamique.

Algorithme 3 : `cas3Parameters`(r_{im} : tableau d'entiers, M : integer, r_i :integer) : integer, integer

$r_i[m]$: tableau d'entiers à partir duquel on calcule le pgcd.
M : taille de $r_i[m]$
r : le débit à atteindre
k : le nombre d'itérations utilisées dans l'algorithme de calcul de $C_i(r)$
d : la plus grande étape pour calculer la configuration de la plate-forme (algorithme 4)

```
begin
    d ← pgcdN(r_im, M)/* pgcd des M entiers de r_i[m]                    */
    k ← r/d
    if (d%r) > 0 then
        k ← r/d
    else
        k ← r/d + 1
    endif
    return d, k
end
```

Algorithme 4 : `CoutTotalCas3`(p, d, r_i, C : integer)

$w(x)$: la fonction qui retourne x si $x > 0$ et 0 sinon
$r_i[m]$: Le débit disponible sur la ressource m pour le traitement d'une tâche de type i
k : le nombre d'itérations
d : la plus grande valeur de pas qui nous permet de couvrir les valeurs possibles de débit avant $r = d \times k$
$C[m]$: coût de l'unité de ressource m
$cost[j]$: coût total de la plate-forme pour un débit $j \times d$
$addedResource[j]$: ressource ajoutée à l'étape j pour la solution optimale avec le débit courant $j \times d$
$resourceToAdd$: La ressource à ajouter à cette étape pour que la plate-forme atteigne le meilleur coût pour le débit courant

```
début
    /* pour tous les débits                                                    */
    pour j = 1 à k faire
        min ← cost[w(j − r_i[0]/d)] + C[0]
        resourceToAdd ← 0
        /* pour toutes les machines                                            */
        pour m = 1 à M − 1 faire
            si (cost[w(j − r_i[m]/d] + C[m]) < min) alors
                min ← cost[w(j − r_i[m]/d)] + C[m]
                resourceToAdd ← m
            finsi
        finpour
        cost[j] ← min
        addedResource[j] ← resourceToAdd ;
    finpour
    /* reconstruction de la solution à ce problème d'optimisation              */
    print "débit" k × d "avec un coût de" cost[k]
    j ← k
    tant que j > 0 faire
        print ("ressource" addedResource[j], "débit suivant" j × d − r_i[addedResource[j]])
        j ← j − r_i[addedResource[j]]/d
    fintq
    Résultat : addedResource, cost[k]
fin
```

Après l'exécution de ces deux algorithmes avec deux débits différents 300 et 450 tâches par unité de temps, les résultats sont les suivants pour la configuration concernant la tâche T_1 :

- Pour un débit de 450 taches par unité de temps, on obtient les résultats suivants : Nous comptons un pas de 10 pour calculer le coût, le nombre d'itérations est égal à 45, le coût optimal de la plate-forme est 150 €. Pour ce faire, il faut 5 unités de ressources : 1 de type P_{11}, 1 de type P_{12} et 3 de type P_{13}.
- Pour un débit de 300 tâches par unité de temps, on obtient les résultats suivant : le pas d'itération vaut 10 également. Le nombre d'itérations est donc égal à 30. Le coût optimal de la plate-forme est 100 €. Pour ce faire, il faut 2 unités de ressources seulement : 1 de type P_{12} et 1 de type P_{13}.

La figure 5.6 illustre le coût optimal de la plate-forme pour l'exécution de la tâche T_1 dans les conditions expérimentales décrites plus haut.

Le tableau 5.6 montre le nombre optimal de ressources utilisées sur la plate-forme pour chaque valeur cible de débit r_1 de la tâche T_1 afin d'optimiser le coût de la plate-forme.

Nous avons remplacé la deuxième ressource de la plate-forme décrite ci-dessus par une ressource plus rapide. La ressource P_{12} peut traiter 250 tâches par unité de temps. Le coût

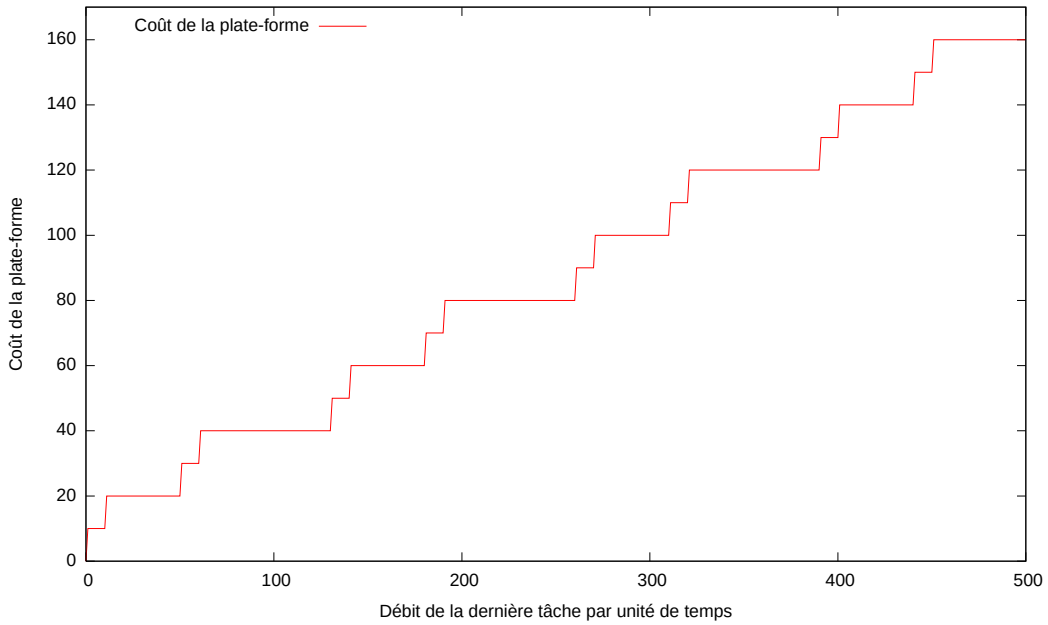

Figure 5.6 – Coût de l'ensemble des processeurs exécutant la tâche T_1 pour le cas 3

économique de cette ressource est aussi plus élevé que celui utilisée précédemment (50 €). Le graphe d'application décrit précédemment est le même.

Pour un débit de 500 tâches par unité de temps, l'expérience montre que le coût total d'exécution est de 100 €. La taille de la plate-forme est de 2 unités de ressources de type P_{12}.

La figure 5.7 montre le coût optimal de la plate-forme pour le traitement de la tâche T_1 dans les conditions expérimentales décrites plus haut.

Les résultats montrent, que le coût total d'exécution augmente en escalier avec le débit et l'utilisation des ressources plus performantes réduit la taille de la plate-forme. Ceci est bien conforme à une exécution classique d'un programme dynamique, la solution optimale correspondant à un certain débit s'appuie sur les solutions optimales des sous-problèmes, c'est à dire des solutions optimales considérant des débits plus faibles. C'est pourquoi, ces solutions peuvent être remises en cause. L'avantage de cet algorithme est d'éviter que tout calcul intermédiaire ne soit refait plusieurs fois comme cela serait le cas avec une programmation descendante. Cependant, la complexité de la solution reste pseudo-polynomiale car elle dépend de la valeur numérique du débit cible (codé en binaire).

Tableau 5.6 – Nombre optimal d'unités de ressources P_{11}, P_{12} et P_{13} pour chaque débit r_1

		nombre de p_i		
		P_{11}	P_{12}	P_{13}
débit r_1	50	0	1	0
	100	0	2	0
	150	2	0	1
	200	2	1	1
	250	0	0	2
	300	0	1	2
	350	0	2	2
	400	1	0	3
	450	1	1	3
	500	0	0	4

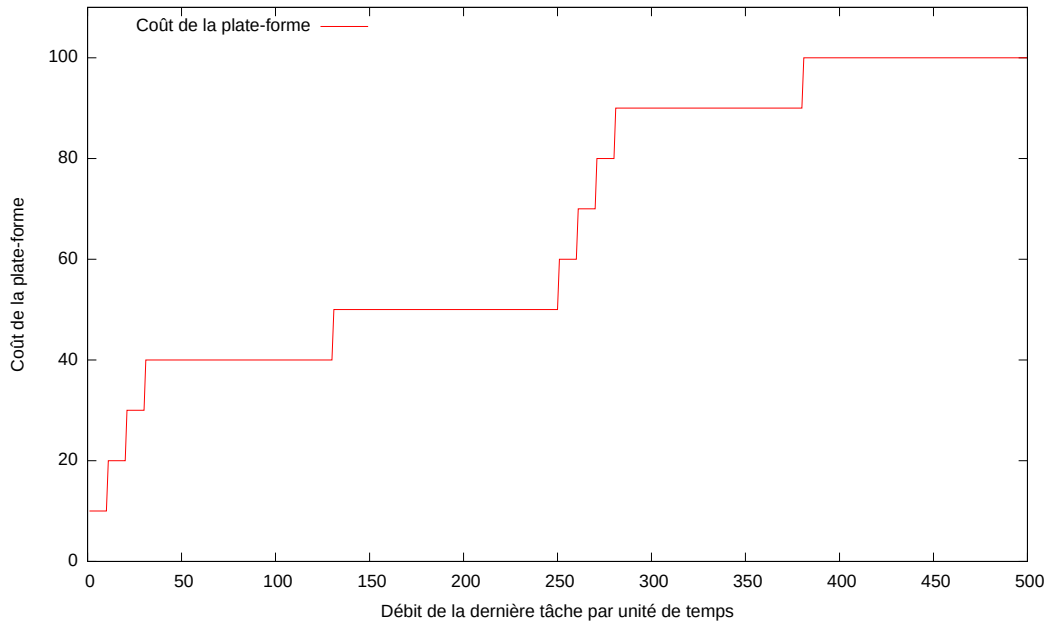

Figure 5.7 – Coût de l'ensemble des ressources traitant la tâche T_1 pour le cas 3

4 Résolution du cas 4

Dans ce paragraphe, nous donnons des algorithmes dans le cas où les ressources de la plate-forme sont dédiées à l'exécution d'un seul type de tâches, chaque type de tâches n'étant exécuté que par un seul type de ressources. Ici, un seul résultat produit ($K = 1$) est décrit par J graphes différents ($J > 1$).

Comme nous avons déjà écrit dans le paragraphe 4.1.4, nous faisons deux approximations avant de présenter le cas général du cas 4.

Cas 4.1 La première approximation consiste à considérer que les graphes, décrivant chacune des manières différentes de produire le résultat, ne partagent pas de tâches de même type. Ces DAGs sont vus comme des boites noires. Les ressources associées à ces boîtes noires sont ajoutées en groupe, en fonction du débit cible à atteindre à moindre coût. Par conséquent, ces boîtes représentent chacune une ressource à part entière. Dans ce cas, le nombre de types de ressources doit être au moins égal au nombre de graphes, c'est-à-dire, si le résultats est décrit par K graphes nous devons avoir au moins K types de ressources sur la plate-forme. En fin de compte, la plate-forme sera composée de plusieurs instances de chacun des types m avec $m = 1 \dots M$.

La plate-forme considérée pour cette expérience est constituée de trois types de ressources (P_1, P_2, P_3). Chacune a un coût de 10, 20 et 40€ respectivement. P_1 a une performance de 10 tâches par unité de temps, P_2 50 tâches par unité de temps et P_3 130 tâches par unité de temps. Pour résoudre cette approximation nous avons développé un programme dynamique directement issu de la formalisation de ce cas donné dans le paragraphe 4.1.4.

La figure 5.8 montre le coût optimal de la plate-forme pour l'exécution de la tâche T_1 dans les conditions expérimentales décrites plus haut.

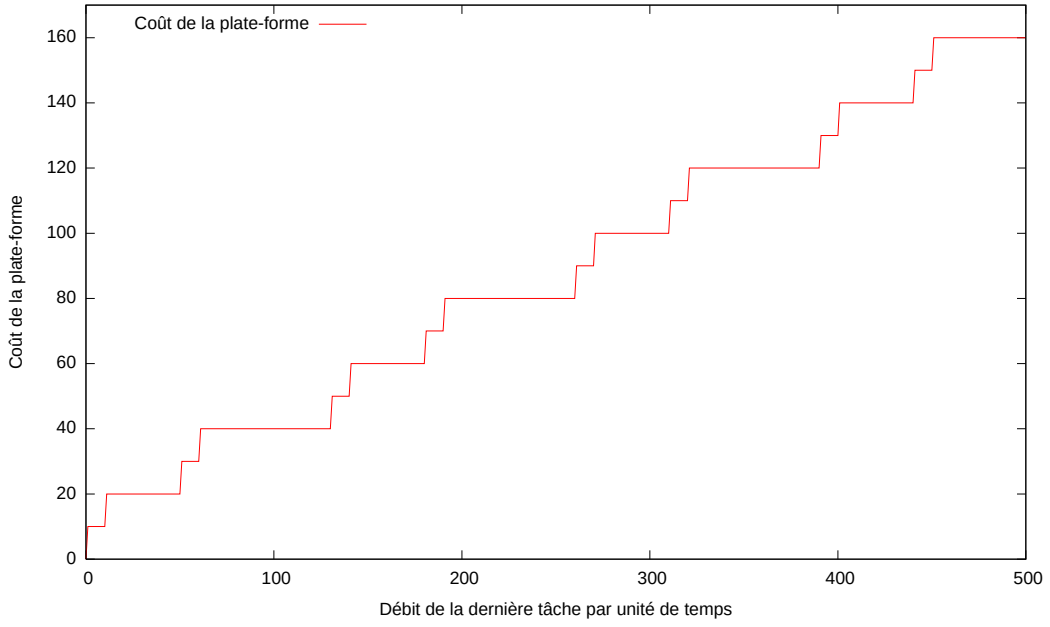

Figure 5.8 – Coût total d'exécution d'un débit $s \leq 500$ tâches par unité de temps pour le cas 4.1 (expérience 1)

L'algorithme 3 donné pour le cas 3 peut être utilisé ici pour éviter également de traiter toutes les valeurs du débit de 1 à s. Il est en effet possible de calculer un pas d'itération égal au *pgcd* des débits de chaque ressource. On en déduit ensuite le nombre d'itérations. Il s'agit d'une optimisation qui ne change pas la complexité algorithmique de la solution donnée ici.

Le calcul du coût optimal de la plate-forme est donné par l'algorithme 5. Il a une complexité $O(Js)$ comme déjà mentionné dans le chapitre précédent.

Nous avons exécuté ces deux algorithmes avec deux débits différents, respectivement 320 et 450 tâches par unité de temps et nous avons obtenu :

- Pour un débit de 450 taches par unité de temps, on obtient les résultats suivants : le pas d'itération est égal à 10, le nombre d'itérations est donc égal à 45 pour un coût total de la plate-forme de 150 €. Cette plate-forme est formée de 5 unités de ressources : 1 de type P_1, 1 de type P_2 et 3 de type P_3.
- Pour un débit de 320 tâches par unité de temps, on obtient les résultats suivants : le pas d'itération est toujours de 10, le nombre d'itérations est donc égal à 32. Le coût total de la plate-forme est maintenant de 110 €, pour 4 unités de ressources : 1 de type P_1, 1 de type P_2 et 2 de type P_3.

Les résultats montrent que le coût d'exécution augmente avec le débit.

Nous répétons l'expérience pour un débit de 320 tâches par unité de temps, en remplaçant la ressource P_2 par une autre plus rapide, qui peut traiter 300 tâches par unité de temps, mais plus coûteuse (80 €). Nous obtenons les résultats suivants après 32 itérations (pas de 10) : la plate-forme optimale dans ce cas à un coût de 100 € pour 3 unités de ressource, 2 de type P_1, 0 de type P_2 et 1 de type P_3.

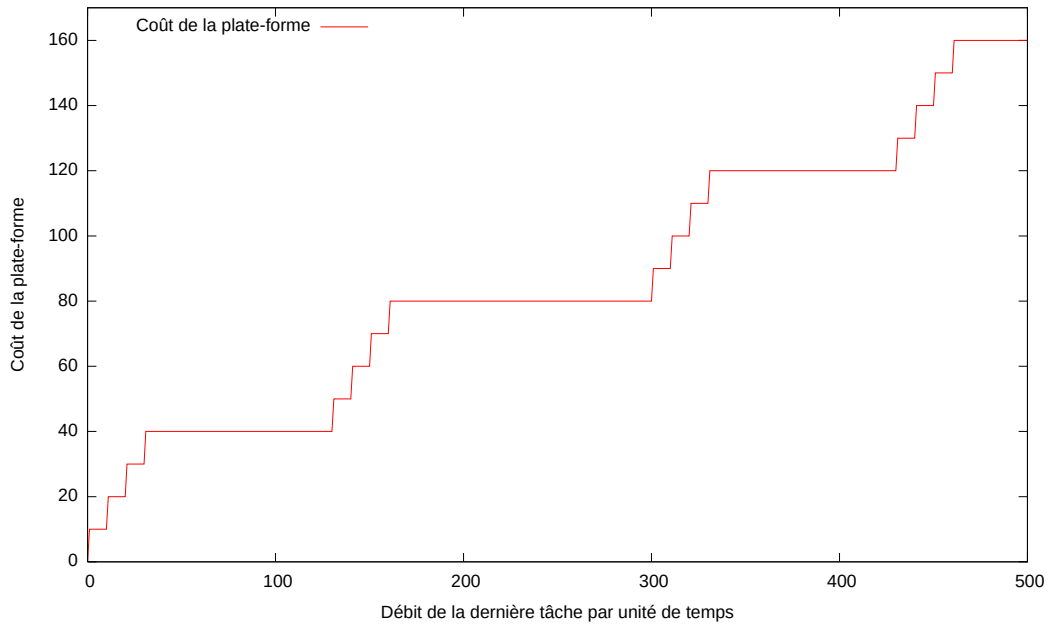

Figure 5.9 – Coût total d'exécution d'un débit $s \leq 500$ tâches par unité de temps pour le cas 4.1 (expérience 2)

Algorithme 5 : `CoutTotalCas4-1`(p, $nbpas$, r, C : integer)

$w(x)$: la fonction qui retourne x si $x > 0$ et 0 sinon
$r[m]$: le débit disponible sur la ressource m
$C[m]$: coût de l'unité de ressource m
$nbPas$: le nombre des itérations
p : pas d'itération
$ressAajouter[i]$: ressource à ajouter à l'étape i pour une solution optimale avec un débit courant $i * p$
$cout[i]$: le coût optimal de la plate-forme pour un débit $i * p$
s : débit cible des résultats en sortie
sc : débit courant
mc : dernière ressource ajoutée pour arriver à un débit $p * i$
$sFutur$: prochain débit examiné
$Debit$: tableau de débits cumulés des machines de chaque type
$ress[mc]$: tableau du nombre des ressources de chaque type

```
début
    pour i = 1 à nbPas faire
        min ← cout[w(i − r[0]/p)] + C[0]
        ressAajouter ← 0
        pour m = 1 à M − 1 faire
            si (cout[w(i − r[m]/p] + C[m]) < min) alors
                min ← cout[w(i − r[m]/p)] + C[m]
                ressAajouter ← m
            finsi
        finpour
        cout[i] ← min
        ressAajouter[i] ← ressAajouter ;
    finpour
    /* reconstruction de la solution pour ce problème d'optimisation                    */
    sc ← p × nbPas
    print "débit" sc "avec un coût" cout[i]
    i ← nbPas
    tant que i > 0 faire
        mc ← ressAajouter[i]
        print ("ressource" mc, "débit" i × p)
        ress[mc] ← ress[mc] + 1
        sFutur ← sc − r[mc]
        si sFutur < 0 alors
            Debit[mc] ← Debit[mc] + sc
        sinon
            Debit[mc] ← Debit[mc] + r[mc]
        finsi
        i ← w(i − r[mc]/p) /* prochain débit à examiner                                 */
        sc ← sFutur
    fintq
    /* décomposition de la somme                                                        */
    mc ← 0
    tant que i < M faire
        print ress[i] "ressources de type" i "pour un débit de" Debit[i]
    fintq
fin
```

La figure 5.9 montre le coût optimal de la plate-forme pour le traitement de la tâche T_1 dans les conditions expérimentales décrites plus haut.

Les résultats montrent que l'utilisation des ressources plus puissantes réduit la taille de la plate-forme.

Cas 4.2 Il s'agit de la deuxième approximation pour le cas 4. Nous donnons ici l'algorithme dont l'idée a été formulée au chapitre précédent. Par rapport au principe donné précédemment, nous prenons une notation voisine afin de tenir compte de tous les graphes permettant de construire le résultat.

- Soit $CT_{i..j}[r]$ le tableau qui donne la valeur optimale du coût pour tous les débits r de 0 à s si les résultats finaux utilisent les graphes i, $i+1$, ..., j. $CT_{i..i}[r]$ est le coût optimal pour le calcul du résultat en n'utilisant que le graphe i pour le débit r.
- Soit $S_{i..j}[r]$ le tableau qui donne s_1 pour la décomposition $r = s_1 + s_2$ conduisant à la solution optimale pour r. Comme la solution se construit en coupant l'espace $i..j$ en deux, ce tableau permet de reconstruire la solution optimale à partir de l'énoncé initial.

On appelle $\texttt{minCT}(s, i, j)$ la fonction qui permet le calcul des tableaux $CT_{i..j}[r]$ et $S_{i..j}[r]$ avec $r = 0 \ldots s$. Cette notation a été choisie afin de ne pas surcharger l'écriture de l'algorithme. Les tableaux CT et S seraient alors des tableaux 3D. Il s'agit de détails à prendre en compte au niveau de l'implémentation dans une langage de programmation.

L'algorithme 6 montre la construction des tableaux définis ci-avant obéissant à une programmation ascendante, même si un découpage récursif de l'espace des graphes (DAG) par deux est fait dans un premier temps. Le calcul se faire dans la phase de remontée des appels, sachant que les données calculées dans une branche ne servent pas à une branche voisine. Ainsi, au sein d'une même branche, les calculs ne sont fait qu'une et une seule fois. L'appel à la fonction se fait de la manière suivante : $\texttt{minCT}_{1,\texttt{J}}(s)$. La valeur optimale de la plate-forme est donnée par $CT_{1..J}[s]$.

L'algorithme 7 permet l'affichage de la solution en montrant la décomposition de s en la somme des J débits partiels effectués par chacun des J DAGs permettant, chacun à leur manière,

de produire le résultat. La somme de ces débits est s. L'appel à la fonction se fait de la manière suivante après que la fonction précédente ait été exécutée : `afficheSij(S, s, 1, J)`.

Algorithme 6 : `minCT`$(s, i, j) : CT_{i..j} : integer, S_{i..j:integer}$

i : indice du premier DAG de l'ensemble des DAGs considérés
j : indice du dernier DAG de l'ensemble des DAGs considérés
l : indice du DAG avec lequel on coupe l'ensemble initial en 2
r : le débit courant de 0 à s
s_1 et s_2 : les débits tels que $r = s_1 + s_2$
$CT_i(r)$: la valeur de la plate-forme avec le seul DAG i pour atteindre un débit r en sortie (voir équation 4.1)

```
début
    si i = j alors
        pour r = 0 à s faire
            CT_{i..j}[r] ← CT_i(r)
            S_{i..j}[r] ← r
        finpour
    sinon
        l ← (i + j)/2
        CT_{i..l}[0..s], S_{i..l}[0..s] ← minCT(s, i, l)
        CT_{(l+1)..j}[0..s], S_{(l+1)..j}[0..s] ← minCT(s, l + 1, j)
        CT_{i..j}[0] ← 0
        S_{i..j}[0] ← 0
        pour r = 1 à s faire
            s_1 ← 0
            CT_{i..j}[r] ← CT_{(l+1)..j}[r]
            pour s_2 = r à 0 (pas -1) faire
                si CT_{i..j}[r] > CT_{i..l}[s_1] + CT_{(l+1)..j}[s_2] alors
                    CT_{i..j}[r] ← CT_{i..l}[s_1] + CT_{(l+1)..j}[s_2]
                    S_{i..j}[r] ← s_1
                finsi
                s_1 + +
            finpour
        finpour
    finsi
fin
```

Résultat : $CT_{i..j}, S_{i..j}$

Algorithme 7 : `afficheSij`$(S, r, i, j : integer)$

i : indice du premier DAG de l'ensemble des DAGs considéré
j : indice du dernier DAG de l'ensemble des DAGs considéré

```
début
    si i = j alors
        print "s" i " = " r
    sinon
        afficheSij(S, S_{i..j}[r], i, (i + j)/2)
        afficheSij(S, r − S_{i..j}[r], (i + j)/2 + 1, j)
    finsi
fin
```

Cas 4.3 Nous avons vu que ce cas conduit à l'écriture d'un programme linéaire en entier pour lequel nous n'avons pas trouvé d'algorithme efficace permettant de dégager une solution optimale calculable en un temps raisonnable. Une solution heuristique est à envisager.

Une heuristique possible est le recours à l'algorithmique génétique. La population de départ serait un ensemble de vecteurs S tels que les composantes de ces vecteurs seraient les valeurs des débits partiels $S[i] = s_i$ des DAGs i. La contrainte est que la somme des $S[i]$ soit égale au débit s donné au départ.

Chacun de ces vecteurs peut être évalué grâce à la formule 4.2 donnant le coût de la plate-forme connaissant les valeurs des débits partiels s_i.

Il faut itérer un certain nombre de fois sur cette population en utilisant les opérations classiques en génétique : sélection, croisement, mutation. Attention, ces opérateurs doivent être adaptés à notre contexte car dans tous les cas, la somme des composantes des vecteurs doit être égale à s. On pourra évaluer comment peut être exploiter le fait de partir de bons individus pour en obtenir de meilleurs.

Une évaluation de cette technique fera l'objet de travaux futurs à mener pour compléter notre contribution dans le registre des solutions sous optimales au problème posé en début de document.

5 Synthèse

Dans ce chapitre, nous avons présenté les résultats d'études algorithmiques et expérimentales des cas 1 à 4 directement liés aux principes des solutions proposées dans le chapitre précédent. Ces cas sont à la base de la résolution des 8 cas mis en évident dans le chapitre 4. Les solutions algorithmiques proposées ici permettent de donner le coût optimal d'une plate-forme hétérogène à ressources dédiées avec contraintes de performance sur le nombre de résultats à sortir par unité de temps. Dans tous les cas, la reconstruction des solutions est possible et est formulée.

Nous avons décrit l'architecture de la plate-forme utilisée, qui est formée de M processeurs hétérogènes. Nous avons défini aussi toutes les données d'entrées du problème telles que le coût économique et les performances des ressources, le nombre d'occurrences de chaque tâche dans le graphe d'application (DAG), et le débit total à traiter en sortie.

Les cas 1 et 2 sont résolus par l'exécution d'une formule. Pour résoudre respectivement le cas 3 et les deux approximations du cas 4, nous avons développés des algorithmes issus de la programmation dynamique et dont la complexité est pseudo-polynomiale. Nous ne proposons pas de solution calculable pour le cas 4.3 mais des pistes en direction d'une heuristique issue de l'algorithmique génétique.

Conclusion et perspectives

Conclusion

Depuis la révolution qu'a représenté l'apparition de l'informatique pour le calcul scientifique, les besoins en matière de puissance de calcul n'ont cessé de croître. La puissance de calcul disponible conditionne toujours la taille maximale des calculs et le débit avec lequel ils peuvent être effectués. Cette puissance connaît une forte augmentation avec l'amélioration des performances des processeurs, de la mémoire, etc [76]. La plupart des plates-formes existantes proposent en particulier des services centralisés basés sur un modèle client/serveur classique.

Ce document présente un instantané de mes activités de recherche sur l'optimisation d'une plate-forme hétérogène de services. La plate-forme ainsi utilisée est constituée de m ressources hétérogènes distribuées interconnectées par des liens de communication de haute performance. Toutefois, en tenant compte de l'existence de telle ou telle bibliothèque, de telle ou telle architecture, nous classifions les ressources qui constituent la plate-forme en ressources spécialisées ou multi-spécialisées.

Une plate-forme est totalement définie par les ressources qui la composent. Comme chaque ressource a un coût économique, une plate-forme possède également un coût économique. C'est de ce coût là que nous parlons au niveau d'une plate-forme.

Dans le cadre de nos travaux, ce coût n'est pas ramené à chaque résultat sortant de la plate-forme mais est un coût initial permettant d'atteindre un certain débit de par la configuration de la plate-forme, ou plus exactement du choix des ressources formant la plate-forme. Ce coût est l'objectif central de ce travail que nous cherchons à minimiser en fonction d'un débit à atteindre, du type de résultats à traiter et des ressources disponibles. La difficulté est par conséquent de bien choisir les ressources pour atteindre les objectifs de rendement que doit remplir la plate-forme.

Notre travail consiste à mener une étude de cas permettant de faire varier 3 paramètres. Les ressources de la plate-forme à construire sont dédiées à un type de tâche uniquement, les tâches peuvent s'exécuter sur une ou plusieurs ressources dans le but de produire un ou plusieurs résultats différents, chacun décrit par un ou plusieurs DAG. Nous avons isolé les 8 cas correspondants et montré que 4 cas sont vraiment différents. Un cas plus général avec des ressources non dédiée a été formalisé par un programme non-linéaire à variables mixtes.

Les 2 premiers cas permettent de calculer la plate-forme optimale grâce à l'évaluation de formules que nous avons développées dans cette étude. Par contre, pour les cas 3 et 4, ce sont des algorithmes qui permettent la construction de l'architecture de la plate-forme. Pour

cela, des algorithmes de programmation dynamique nous ont permis de résoudre trois de nos problèmes d'optimisation (cas 3, cas 4.1 et cas 4.2) alors que les problèmes sont connus pour être combinatoires. Les algorithmes présentés permettent de trouver les solutions optimales de ces problèmes avec des complexités pseudo-polynomiales. Avec des valeurs raisonnables du débit en sortie, les algorithmes sont efficaces.

Perspectives de recherche

Les cas permettant d'exhiber une solution optimale pour la plate-forme ont été montrés dans cette thèse. Une perspective intéressante à mener suite à ce travail serait d'étendre notre étude aux cas nous obligeant à proposer des solutions sous optimale par une approche heuristique. Dans ce cas, différentes approches heuristiques pourraient être proposées et comparées. Nous avons d'ailleurs proposé une telle approche pour la résolution du cas 4.3. La formalisation du cas pour lequel les ressources ne sont plus dédiées au traitement d'un seul type de tâches nous a montré également l'intérêt d'une telle approche. Ainsi, la totalité des 16 cas représentatifs du problème le plus général serait cernée.

Bibliographie

[1] Grid 5000. Grid'5000 project. http//www.grid5000.org/.

[2] Amir H. Abdekhodaee, Andrew Wirth, and Heng-Soon Gan. Scheduling two parallel machines with a single server : the general case. *Computers and operations research*, 33 :994 – 1009, 2006.

[3] Foto Afrati, Evripidis Bampis, David Karger, Claire Kenyon, Sanjeev Khanna, Ioanis Milis, Maurice Queyranne, Martin Skutella, Cliff Stein, and Maxim Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. *Journal of Scheduling*, 1 :846– 857, 1973.

[4] I. Ahmad and K. Yu-Kwong. On exploiting task duplication in parallel program scheduling. *IEEE Transactions on Parallel and Distributed Computing*, 9(9) :872–892, September 1998.

[5] Pervaiz Ahmed, Reza Tavakkoli-Moghaddam, and Nadim Safaei. A comparison of heuristic methods for solving a cellular manufacturing model in a dynamic environment. *Working Paper Series*, 2004.

[6] The Globus Alliance. Le site w3 de globus. http ://www.globus.org.

[7] Hakan Aydin, Rami Melhem, Daniel Mossé, and Pedro Mejia-Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Transactions on Computers*, 53(5), May 2004.

[8] Rashmi Bajaj and Dharma P. Agrawal. Improving scheduling of tasks in a heterogeneous environment. *IEEE Transactions on Parallel and Distributed Systems*, 15(2), February 2004.

[9] Philippe Baptise and Claude Le Pape. Scheduling a single machine to minimize a regular objective function under setup constraints. *Discrete optimization*, 2 :83–99, 2005.

[10] Olivier Beaumont, Arnaud Legrand, Loris Marchal, and Yves Robert. Scheduling strategies for mixed data and task parallelism on heterogeneous clusters. *Parallel Processing Letters*, 13(2) :225–244, 2003.

[11] Olivier Beaumont, Arnaud Legrand, Loris Marchal, and Yves Robert. Assessing the impact and limits of steady-state scheduling for mixed task and data parallelism on heterogeneous platforms. In *HeteroPar'2004 : International Conference on Heterogeneous Computing, jointly published with ISPDC'2004 : International Symposium on Parallel and Distributed Computing*, pages 296–302. IEEE Computer Society Press, 2004.

[12] Olivier Beaumont, Arnaud Legrand, Loris Marchal, and Yves Robert. Steady-state scheduling on heterogeneous clusters : why and how ? In *6th Workshop on Advances in Paral-*

lel and Distributed Computational Models APDCM 2004. IEEE Computer Society Press, 2004.

[13] Olivier Beaumont, Arnaud Legrand, and Yves Robert. Optimal algorithms for scheduling divisible workloads on heterogeneous systems. In *HCW'2003, the 12th Heterogeneous Computing Workshop.* IEEE Computer Society Press, 2003.

[14] Lyés Benyoucef and Bernard Penz. Just-in-time changeover cost minimization problem : Dynamic programming approach and complexity results. Rapport de recherche 4727, INRIA, February 2003.

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

[16] Jean Bétréma. Classes de complexité p et np, chapitre 8. `http://www.labri.fr/perso/betrema/MC/MC8.html`, September 2008.

[17] Alain Billionnet. *Optimisation discrète. De la modélisation à la résolution par des logiciels de programmation mathématique*, chapter 1. Dunod, 2007.

[18] J. Blazewicz, K Ecker, E. Pesch, G. Schmidt, and J. Weglarz. *Scheduling computer and manufacturing processes.* Berlin-Springer, 2001.

[19] Cristina Boeres, José Filho, and Vinod E.F. Rebello. A cluster strategy for scheduling task on heterogeneous processors. *IEEE Computer Society, Proc 16th Symp. on Computer Architecture and High Performance Computing*, 2004.

[20] Pierre Boulet, Jack Dongarra, Fabrice Rastello, Yves Robert, and Frédéric Vivien. Algorithmic issues on heterogeneous computing platforms. *Parallel Processing Letters*, 9(2) :179–213, 1999.

[21] Tracy D. Braun, Howard Jay Siegel, and Noah Beck. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Academic Press*, pages 810– 837, 2001.

[22] Peter Brucker. Complex scheduling problems. *Journal of Scheduling*, 1999.

[23] J. Bruno, E. Coffman, and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Communications of the A.C.M*, 17(7) :382–387, 1974.

[24] R. Buyya, D. Abramson, and J. Giddy. A case for economy architecture for service oriented grid computing. `http://gridbus.org/papers/ecogrid.pdf`.

[25] R. Buyya, D. Abramson, and J. Giddy. An economy grid architecture for service-oriented grid computing. In *10th IEEE International Heterogeneous Computing Workshop (HCW2001), with IPDPS 2001, SF, California, USA*, April 2001.

[26] R. Buyya, H. Stockinger, J. Giddy, and D. Abramson. Economic models for management of resources in grid computing. `http://arxiV.org/ftp/cs/papers/0106/010602.pdf`.

[27] R. Buyya, H. Stockinger, J. Giddy, and D. Abramson. Economic models for management of ressources in peer-to-peer and grid computing. In *SPIE International Conference on Commercial Applications for High Performance Computing*, Denver, USA, August 2001.

[28] Yves Caniou. *Ordonnancement sur une plate-forme de métacomputing.* PhD thesis, Ecole doctorale IAEM Lorraine, Nancy, France, 2004.

[29] Dong Cao, Mingyuan Chen, and Guohua Wan. Parallel machine selection and job scheduling to minimize machine cost and job tardiness. *Computers and Operations Research*, (32) :1995– 2012, 2005.

[30] Eddy Caron and Fréderic Desprez. Diet : tour d'horizon. *GridUse-2004, École thématique sur la globalisation des ressources informatiques et des données : Utilisation et Services, Campus de Metz de supélec*, June 2004.

[31] Ernemann Carsten, Hamscher Volker, and Yahyapour Ramin. Economic scheduling in grid computing. `http://www.ds-technik.uni-dortmund.de/~yahya/papers/cei_jsspp2002.pdf`, 2002.

[32] Henri Casanova and Jack Dongarra. Netsolve : A network solver for solving computational science problems. In *Super-Computing Pittsburg*, 1999.

[33] Haoxum Chen, Chengbin Chu, and Jean-Marie Proth. Job shop scheduling using lagrangian relaxation. Technical Report RR-2505, INRIA, September 1995.

[34] Mingyuan Chen. A mathematical programming model for system configuration in a dynamic cellular manufacturing environment. *Annals of Operations Research*, 77 :109–128, 1998.

[35] Shen-Yeh Chen. A robust genetic algorithm for structural optimization. *FEA-Opt Technology*, 2001.

[36] Pablo E. Coll, Celso C. Ribeiro, and Cid C. de Souza. Multiprocessor scheduling under precedence constraints : polyhedral results. *Discrete Applied Mathematics*, 154 :770 –801, 2006.

[37] R. Conway, W. Maxwell, and L. Miller. *Theorie of scheduling.* Courier Dover Publications, 2003.

[38] Thomas H. Cormen, Charles E. Leiserson, Ronald L.Rivest, and Clifford stein. *Introduction à l'algorithmique, cours et exercices*, chapter 34. DUNOD, 2e edition edition, 2004.

[39] K. Czajkowski, I. Foster, and C. Kesselman. Ressource Co-Allocation in Computational Grids. In *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8)*, pages 219–228, 1999.

[40] Jean Damay. *Techniques de résolution basées sur la programmation linéaire pour l'ordonnancement de projet.* PhD thesis, Université Blaise Pascal-Cermont II, Ecole Doctorale sciences pour l'ingénieur de Clermant-Ferrand, 2005. N° d'ordre : D.U.1620.

[41] L. Erik Demeulemeester and Willy Herroelen. *Project scheduling : A research handbook*, chapter 3. Springer, 2e edition edition, 2002.

[42] E. Descourvières and All. Towards automatic control for microfactories. In *5th Int. Conf. on Industrial automation*, Montréal, Québec, Canada, June 2007. ISBN 978-2-9802946-4-8.

[43] Muhammad K. Dhodhi, Imtiaz Ahmad, Anwar Yatama, and Ishfaq Ahmad. An integrated technique for task matching and scheduling onto distributed heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 62 :1338–1361, 2002.

[44] Furguson F. Donald, Nicholaou Christos, Sairamech Jakka, and Yemini Yechiam. Economic models for allocatating ressources in computer systems. In *Market based of distributed systems, world scientific*, 1996.

[45] H. EL-Rewini and T.G Lewis. Scheduling parallel program tasks onto arbitrary target machines. *Journal of Parallel and Distributed Computing*, 9 :138–153, 1990.

[46] D. H. J Epema, Miron Linvy, R. van Dantzig, X. Evers, and Jim Pruyne. A worldwide klock of condors : Load sharing among workstation clusters. *Journal of Future Generations of Computer Systems*, 12, 1996.

[47] M. Essaidi. *Echange des données pour le parrallélisme à gros grain.* PhD thesis, Université Henri Poincaré, Nancy, France, February 2004.

[48] Antoine Ferreira. Vers les micro-usines automatisées du futur. *J'automatise*, 18 :47–51, 2001.

[49] Clarisse Flipo-Dhaenens. *Optimisation d'un réseau de production et de distribution.* PhD thesis, Institut National Polytechnique de grenoble, 1998.

[50] Stéphane Fontaine, Christophe Taton, Sara Bouchenak, and Thierry Gautier. Administration autonome d'applications réparties sur grilles. *RenPar'17/SympA'2006/CFSE'5/JC'2006*, October 2006.

[51] Ian Foster. What is the grid ? a three point checklist. *GridToday*, July 2002.

[52] Ian Foster and Carl Kesselman. Globus : A metacomputing infrastructure toolkit. http ://www.globus.org.

[53] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid : Enabling scalable virtual organization. *Int. J. High perform. Comput. Appl.*, 15(3) :200–222, 2001.

[54] Fedak Gilles, Germain Cécile, Néri Vincent, and Cappello Franck. Xtremweb : A generic global computing system. In *IEEE press, editor, CCGrid2000, workshop on Global Computing on Personal Devices*, May 2001.

[55] Leslie Ann Goldberg, Mike Paterson, Aravind Srinivasan, and Elizabeth Sweedyk. Better approximation guarentees for job shop scheduling. *ACM-SIAM Symposium on Discrete Algorithms*, pages 599–608, 1997.

[56] GOTHA. Les problèmes d'ordonnancement. *R.A.I.R.O.-O.R.27,1*, 27(1) :77–150, 1993.

[57] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell Systems Technical Journal*, 45 :1563–1581, 1966.

[58] Xian He and Ming Wu. Quality of service of grid computing :sharing resources. *The six international conference on Grid and cooperative computing (GCC 2007) IEEE-Computer Society*, 2007.

[59] Fabien Hermenier, Xavier Lorca, Hadrien Cambazard, Jean-Marc Menaud, and Narendra Jussien. Reconfiguration dynanique du placement dans les grilles de calcul dirigé e par des contraintes. *RenPar'18/SympA'2008/CFSE'6*, 2008.

[60] Fabio Hernandez, Nicolas Jacq, and Sophie Nicoud. Datagrid, projet européen de grille de calcul. `http://www.urec.cnrs.fr/IMG/pdf/articles01.JRES01.Datagrid.pdf`.

[61] Henry John Holland. *Adaptation in natural and artificial systems*, chapter 1. The MIT Press, 2e edition edition, 1992.

[62] H. Horn. Minimizing average flow time with parallel machines. *Operations research*, 21, 1973.

[63] Oscar H. Ibarra and Chul E. Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the Association for Computing Machinery*, 24(2) :280–289, April 1977.

[64] E. Ilvarasan and P. Thambidurai. Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. *Journal of Computer Sciences*, 3(2) :94–103, 2007.

[65] IN2P3. Grille de calcul : l'internet du calcul intensif. `http://www.in2p3.fr`.

[66] M. Iverson, F. Ozguner, and G. Follen. Parallelizing existing applications in a distributed heterogeneous environments. *Proc. Heterogeneous computing Workshop*, pages 93–100, 1995.

[67] Klaus Jansen and Lorant Porkolab. Polynomial time approximation schemes for general multiprocessor job shop scheduling. *Journal of Algorithms*, 45 :167–191, 2002.

[68] Herbert Jaudlbauer. An approach for integrated scheduling and lot-sizing. *European journal of operational research*, 172 :386–400, 2006.

[69] Wieslaw Kubiak and Vadim Timkovsky. Total completion time minimization in two-machine job shop with unit-time operations. *European journal of operational research*, 94 :310–320, 1996.

[70] Yassine Lassoued. Etude du paramétrage d'un algorithme d'optimisation hybride. Master's thesis, rapport de DEA Programmation et Systèmes, 2000.

[71] A. Legrand, L. Marchal, and Y. Robert. Optimizing the steady-state throughput of scatter and reduce operations on heterogeneous platforms. *Journal of Parallel and Distributed Computing*, 2005.

[72] Arnaud Legrand, Olivier Beaumont, Loris Marchal, and Yves Robert. Optimizing the steady-state throughput of broadcasts on heterogeneous platforms. Rapport de Recherche RR-4871, INRIA, July 2003.

[73] Arnaud Legrand, Alan Su, and Frédéric Vivien. Off-line scheduling of divisible requests on an heterogeneous collection of databanks. Technical Report RR-5386, INRIA, November 2004.

[74] Zhenbo Li, Jiapin Chen, and Jianzhi Feng. Design for an omni-directional mobil microrobot(ommr-i) for a micro-factury with 2mm electromagnetic micromotors. *Robotica*, 23 :45–49, 2005.

[75] S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44 :2245–2269, 1965.

[76] Frédric Lombard. *Spécification et mise en œuvre d'une plate-forme de métacomputing multi-agents*. PhD thesis, L'UFR des Sciences et Techniques de L'Université de Franche-Comté, Besançon, France, December 2002.

[77] P. B. Luh, D. J. Hoitomt, E. Max, and K. P. Pattipati. Scheduling generation and reconfiguration for parallel machines. *IEEE Transaction on Robotics and Automate*, 6 :687–696, 1990.

[78] Jean-Marc Nicod, Laurent Philippe, and Hala Sabbah. Optimizing the cost of an heteregeneous distributed platform. In *In DFMA'08, 4th IEEE Int. Conf. on Distributed Frameworks for Multimedia Applications, Penang, Malysia*, pages 60–67, 2008.

[79] Tchimou N'takpé. Algorithmes d'ordonnancement de graphes de tâches parallèles sur plates-formes hétérogènes en deux étapes. *RenPar'17/sympA'2006/CFSE'5/JC'2006, Canet en Roussillon, 4 au 6 octobre 2006*, 2006.

[80] Yuichi Okazaki, Nozomu Mishima, and Kiwamu Ashida. Microfactory and micro machine tools. In *Reported in the first Korea-Japon Conference on Positionning Technology, Daejeon, Korea*, 2002.

[81] Andrew J. Page and Thomas J. Naughton. Framework for task scheduling in heterogeneous distributed computing using genetic algorithms. *Artificial Intelligence Review*, 24 :415 – 429, 2005.

[82] Katerina P. Papadaki and Warren B. Powell. Exploiting structure in adaptive dynamic programming algorithms for a stochastic batch service problem. *European Journal of Operationnal Research*, (142) :108–127, 2002.

[83] Manish Parashar and James C. Browne. Conceptual and implementation models for the grid. In *IEEE special issue on Grid Computing*, pages 653–668, 2005.

[84] R. Gray Parker and Ronald L. Rardin. *Discrete optimization.* Academic Press, Werner Rheinbolt and Daniel Siewiorek edition, 1988.

[85] François Pelligrini and Jean Roman. Experimental analysis of the dual recursive bipartitioning algorithm for static mapping. Technical Report RR 1138-96, Labri, URA CNRS 1304, 1996.

[86] Satich Penmasta and T. Anthony Chronopoulos. Job allocation schemes in computational grid based on cost optimization. In *19th IEEE Int. Parallel and Distributed Processing Symposium*, 2005. ISBN 1530-2075.

[87] Ferdinando Pezella and Emanuela Merelli. A tabou search method guided by shifting bottelneck for the job shop scheduling problem. *European Journal of Operationnal Research*, 120 :297–310, 2000.

[88] Laurent Philippe. Gridrpc normalisation des api d'accès aux applications de grille. *GridUse-2004*, June 2004.

[89] Cynthia Phillips, Clifford Stein, and Joel Wein. Minimizing Average Completion Time in the Presence of Release Dates. *Mathematical Programming B*, 82(1-2) :199–224, June 1998.

[90] Jan Plesnik. Minimum cost edge covering exactly k vertices of a graph. *Journal of Combinatorial Optimization*, 5 :275–286, 2001.

[91] Vishwanath Ramabhatta and Rakesh Nagi. An integrated formulation of manufacturing cell formation with capacity planning and multiple routings. *Annals of Operations Research*, 77 :79–95, 1998.

[92] S. Ranaweera and D. Agrawal. A task duplication based scheduling algorithm for heterogeneous systems. *IEEE Computer Society*, May 2000.

[93] A. Kan Rinnoy. *Machine scheduling problem : classification, complexity and computations.* Springer, 1976.

[94] Anna Robert. *Optimisation des batches de production.* PhD thesis, Université Pierre et Marie Curie, September 2007.

[95] Anna Robert, Claude Le Pape, Frédéric Paulin, and Francis Sourd. Une nouvelle approche de l'intégration de la planification de production et de l'ordonnancement. 7^{me} *rencontre des jeunes chercheurs en intelligence artificielle, RJCIA 2005*, pages 15–28, 2005.

[96] Mathilde Romberg. The unicore architecture seamless access to distributed ressources. In *8th IEEE Int. Sym. on High Performance Distributed Computing*, 1999. ISBN 0-7803-5624-5.

[97] Cheol Kim Sang and Lee Sunggu. Push-pull : Guided search dag scheduling for heterogeneous clusters. *Proceeding of the 2005 international conference on parallel processing (ICCP'05)*, 2005.

[98] Giovanna Di Marzo Serugendo. Grille de calcul et reseau pair-a-pair. `http://cui.unige.ch/~dimarzo/seminars/ist-05-02.pdf`.

[99] SETI@home. Le site w3 de seti@home. http ://setiathome.ssl.beckeley.edu.

[100] Jeffrey Shneidman, Chaki Ng, C. David Parkes, Au Alvin Young, C. Alex Snoeren, Amin Vahdat, and Brent Chun. Why market could (but don't currently) solve ressource allocation problems in systems. In *Usinex'05 : Proceedings of the 10th USENIX workshop on hot topics in operating systems*, 2005.

[101] René Sitters. Complexity of Preemptive Minsum Scheduling on Unrelated Machines. *ACM*, 17 :382–394, 1974.

[102] Jeeho Sohn, Thomas G. Robertazzi, and Serge Luryi. Optimizing computing costs using divisible load analysis. *IEEE Transactions on Parallel and Distributed Systems*, 9(3), March 1998.

[103] Annie S.Wu, Shiyuan Jin, Kuo-Chi Lin, and Guy Schiavone. An incremental genetic algorithm approach to multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 15(9), September 2004.

[104] Djamel Nait Tahar, Farouk Yalaoui, Chengbin Chu, and Lionel Amodeo. A linear programming approach for identical parallel machine scheduling with job splitting and sequence dependent setup times. *International journal of production economics*, 99 :63 –73, 2006.

[105] E.G. Talbi. Une taxonomie des algorithmes d'allocation dynamique de processus dans les systèmes parallèles et distribués. In *ISYPAR'97 2ème ecole d'informatique des systèmes parallèles et répartis*, pages 137–164, March 1997.

[106] Makoto Tanaka. Development of desktop machining microfactory. *Journal RIKEN Rev*, 34 :46–49, April 2001. ISSN :0919-3405.

[107] Yang Tao and Gerasoulis Apostolos. Pyrros : static task scheduling and code generation for message passing multiprocessors. In *6th ACM International conference on supercomputing*, pages 428–437, 2001.

[108] The Diet Team. Le site w3 de diet. http ://graal.ens-lyon.fr/DIET.

[109] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. Performance-effective and low complexity task scheduling for heterogeneous computing. *IEEE Transactions On Parallel and Distributend Systems*, 13(3), March 2002.

[110] Harvey M. Wagner. *Principles of Operations Research with Applications to Managerial Decisions*, chapter 8, Introduction to Dynamic Optimization Models, pages 263–300. Prentice Hall ; 2nd edition, second edition, September 1975.

[111] Lee Wang, Howard Jay Siegel, Vwani P. Roychowdhury, and Anthony A. Maciejewski. Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. *Journal of Parallel and Distributed Computing*, 47 :8–22, 1997.

[112] Wikipédia. Grille informatique. `http://fr.wikipedia.org/wiki/Grille_de_calcul`, September 2008.

[113] Yijia Xu and Suvrajeet Sen. A distributed computing architecture for simulation and optimization. *Proceedings of the 2005 Winter simulation conference*, 2005.

[114] Jia Yu and Rajkumar Buyya. Grid market directory : A web and web services based grid service publication directory. Technical report, Department of computer science and software engineering, University of Melbourne, 2002.

[115] J. J Yuan, Y. X. Lin, C. T. Ng, and T. C. E. Cheng. Approximation of single scheduling with fixed jobs to minimize total completion time. *European journal of operational research*, 2006.

[116] Hou Yumei, P. Sethi Suresh, Zhang Hanqin, and Zhang Qing. Asymptotically optimal production policies in dynamic stochastic jobshops with limited buffers. *Mathematical analysis and applications*, 317 :390–428, 2006.