REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE AMAR TELIDJI LAGHOUAT

Optimisation de l'allocation de ressources dans le Cloud Computing

Thése présentée

par

BOUZIDI Mohamed redha

En vue d'obtenir le titre de

Docteur en Informatique

Membres du jury

Mr. YAGOUBI Mohamed Bachir	Prof	UATL	President
Mr. BOUKRA Abdelmadjid	Prof	USTHB	Examinateur
Mr. BENCHAIBA Mahfoud	Prof	USTHB	Examinateur
Mr. BOUAKAZ Mustpha	M.C.A	UATL	Examinateur
Mr. GUELOUMA Younece	M.C.A	UATL	Examinateur
Mr. DAOUDI Mourade	M.C.A	USTHB	Directeur de thèse
Mr. BOUKHALFA Kamal	Prof	USTHB	Co-directeur de thèse

Remerciements

A l’issue de ce travail de thèse, je voudrais remercier toutes celles et tous ceux qui ont contribué à le rendre possible. Tout d’abord, mes deux directeurs de thèse monsieur DAOUDI Mourad et BOUKHALFA Kamel, pour avoir dirigé ce travail de recherche, avec leurs conseils, leurs patiences, et une constante exigence.

Je remercie très respectueusement l’ensemble des membres du jury qui m’ont fait l’honneur d’évaluer ce modeste travail.

Je remercie enfin tous ceux et celles qui ont contribué de près ou de loin dans l’accomplissement de ce modeste travail.

Dédicaces

Je dédie ce modeste à ceux qui sont le symbole de courage, ceux à qui je dois tout, ma mère et mon père.

Je le dédie aussi à :

Ma femme, mes fils,

À tous mes amis.

Sans oublier tous les enseignants qui ont contribué à ma formation, tous cycles confondus.

ملخص

تستخدم تتابع الأعمال بشكل متزايد في العديد من التطبيقات. وهي تصف رسميا سلسلة من الحسابات التي تسمح بتحليل البيانات بطريقة منظمة وموزعة. تحتوي تتابع الاعمال هذه على العديد من المهام وقيود التبعية بين هذه المهام. من ناحية أخرى ، توفر الحوسبة السحابية فرصا هائلة لحوسبة تتابع الأعمال واسعة النطاق والتطبيقات كثيفة الحوسبة. يعتمد أداء معالجة تتابع الأعمال بشكل كبير على كيفية جدولة المهام المختلفة على موارد السحابة. ولا يزال تخطيط تتابع الأعمال مشكلة صعبة تندرج في فئة المشاكل الصعبة، والتي يصعب إيجاد حل دقيق لها. وقد تم تكريس الكثير من الجهد البحثي لحل هذه المشكلة، بما في ذلك طريقة MOHEFT (متعددة الأهداف غير المتجانسة في وقت الانتهاء المبكر)، والتي تثبت أنها مرجع لقضايا جدولة تتابع الأعمال.

في حين أن إظهار القدرة على الحصول على نوعية جيدة جدا من النتائج المقدمة، MOHEFT يعاني من تعقيده الزمني الكبير عندما يتعلق الأمربتتابع الأعمال كبيرة. لعلاج هذا القصور، درسنا أولا تعقيد MOHEFT. ثم، لتسريع MOHEFT، حاولنا إعادة كتابة التعليمات البرمجية واستغلال البرمجة الموازية. هذه التقنيات الماضيين وجدت حاجز تصميم ناجم عن ازدحام قنوات الذاكرة، والتسارع الأقصى الذي تم التوصل إليه هو 3.11 مرات. ثانيا، اقترحنا FAMOBACH. هذا الإصدار المحسن من MOHEFT يلغي الحسابات الزائدة عن الحاجة باستخدام Checkpointing و Backtracking. تقييم أداء FAMOBACH يظهر أنه أسرع ب 9 مرات من MOHEFT.

Abstract

Workflows and workflow scheduling are increasingly used in many applications with many tasks and dependency constraints between these tasks. On the other hand, cloud computing offers huge opportunities to process large-scale workflows and data-intensive applications. The processing performance of a workflow is highly depending on how the different tasks are scheduled on the cloud's resources. Workflow scheduling is still a challenging issue that falls into NP-hard problems category, for which finding an exact solution is intractable. Much research effort has gone into solving this problem, including multi-objective heterogeneous earliest-finish-time (MOHEFT) method, which turns to be a reference for the workflow scheduling problem. While demonstrating very high efficiency at the provided results quality, MOHEFT suffers from its high time complexity when it comes to large-scale workflows. To address this shortcoming, we first investigate MOHEFT complexity. Then, to speed up MOHEFT, we tried code tuning and parallel processing ; the latter two techniques found an archetypal barrier caused by memory bus congestion, with a maximum speedup of 3.11 X. Secondly, we propose **FA**st workflow scheduling approach based **MO**HEFT using **BA**cktraking and **CH**eckpointing (FAMOBACH). The latter is an improved version that eliminates redundant calculations in MOHEFT using checkpointing and backtacking. FAMOBACH performance evaluation depicts it runs up to 9 times faster than the MOHEFT.

Keywords : MOHEFT, FAMOBACH, Workflow Scheduling, Cloud Computing.

Résumé

Les workflows sont de plus en plus utilisés dans de nombreuses applications. Ils décrivent d'une façon formelle une série de calculs qui permettent d'analyser des données de manière structurée et distribuée. Ces workflows contiennent de nombreuses tâches et des contraintes de dépendance entre ces tâches. D'autre part, le cloud computing offre d'énormes opportunités pour le calcul des workflows à grande échelle et des applications gourmandes en calcul. Les performances de traitement d'un workflow dépendent fortement de la façon dont les différentes tâches sont planifiées sur les ressources du cloud. La planification des workflows reste un problème difficile qui relève de la catégorie des problèmes NP-hard, pour lesquels il est difficile de trouver une solution exacte. Beaucoup d'efforts de recherche ont été consacrés à la résolution de ce problème, y compris la méthode MOHEFT (multi-objective hétérogène early-finish-time), qui s'avère être une référence pour la problématique de planification des workflows. Tout en démontrant l'aptitude d'avoir une très bonne qualité des résultats fournis, MOHEFT souffre de sa grande complexité temporelle lorsqu'il s'agit de large workflows. Pour remédier à cette lacune, premièrement nous avons étudié la complexité de MOHEFT. Ensuite, pour accélérer MOHEFT, nous avons essayé la réécriture du code et l'exploitation de la programmation parallèle ; ces deux dernières techniques ont trouvé une barrière architecturale causée par la congestion du bus mémoire, le maximum d'accélération atteint est de $3,11\times$. Deuxièmement, nous avons proposé FAMOBACH. Cette version améliorée de MOHEFT élimine les calculs redondants en utilisant le checkpointing et le backtacking. L'évaluation des performances de FAMOBACH montre qu'il est jusqu'à 9 fois plus rapide que MOHEFT.

Mots clés : MOHEFT, FAMOBACH, Workflow Scheduling, Cloud Computing.

Table des matières

Table des figures

Liste des tableaux

Liste des abbréviations

DM **D**ecision **M**aker
FAMOBACH **FA**ast **M**ulti **O**bjectif **BA**acKtracking and **CH**eckpointing
CL9 Column address strobe **L**atency 9
DAG **D**irected **A**cyclic **G**raph
HADOOP **H**igh **A**vailability **D**istributed **O**riented **P**latform
HEFT **H**eterogenous **E**arliest **F**inish **T**ime
IT **I**nformation **T**echnology
MODE **M**ulti **O**bjectif **D**ifferential **E**volution
MOAC **M**ulti **O**bjectif **A**nt **C**olony
MOELS **M**ulti **O**bjectif **L**ist **S**cheduling
MOHEFT **M**ulti **O**bjectif **H**eterogenous **E**arliest **F**inish **T**ime
MOP **M**ulti **O**bjectif **O**ptimisation
NP **N**on deterministic **P**olynomial time
NSGAII **N**on dominated **S**orting **G**enetic **A**lgorithmII
NSPSO **N**on dominated **P**article **S**warm **O**ptimisation
QOS **Q**uality **O**f **S**ervice
RAM **R**andom **A**ccess **M**emory
SLA **S**ervice **L**evel **A**greement
SPEAII **S**trength **P**areto **E**volutionary **A**lgorithmII
WAN **W**ide **A**rea **N**etwork
WEC **W**orkflow **E**xecution **C**ost
WET **W**orkflow **E**xecution **T**ime
XML **eX**tencible **M**arkup **L**anguage

Chapitre 1

Introduction

1.1 Contexte et motivations

Les workflows scientifiques sont des paradigmes couramment utilisés en informatique. Ils décrivent d'une façon formelle une série de calculs qui permettent d'analyser des données de manière structurée et distribuée. Les workflows scientifiques ont été utilisés avec succès pour réaliser des avancées scientifiques significatives dans divers domaines tels que la biologie, la physique, la médecine et l'astronomie. Leur importance est soulignée dans l'ère actuelle du big data, car ils offrent un moyen efficace de traiter et d'extraire des connaissances à partir des données toujours plus nombreuses produites par des outils de plus en plus puissants tels que les télescopes, les accélérateurs de particules et les détecteurs d'ondes gravitationnelles . L'émergence du cloud computing a apporté plusieurs avantages pour le déploiement et le calcul des workflows scientifiques à grande échelle. En particulier, les nuages IaaS (Infrastructure as à Service) offrent une infrastructure facilement accessible, flexible et évolutive pour le déploiement de ces applications. Les fournisseurs CLOUD offrent la possibilité de déployer les workflows scientifiques à faible coût et ceci sans avoir à posséder sa propre infrastructure, juste en louant des ressources informatiques virtualisées, ou machines virtuelles (VM). Les workflows scientifiques peuvent ainsi être facilement exécutés et déployés sur le cloud. D'où les systèmes de gestion des workflows ont accès à un pool virtuellement infini de machines virtuelles qui peuvent être acquises et libérées

de manière transparente. Le processus d'ordonnancement d'un workflow dans un système distribué consiste à associer les tâches du workflow aux ressources cloud et à arranger leurs exécutions de manière à préserver les dépendances entre elles. Le mappage des tâches du workflow aux ressources cloud est également effectué de manière à ce que différentes exigences de QoS définies par l'utilisateur soient satisfaites. Ces paramètres de qualité de service déterminent les objectifs d'ordonnancement et sont généralement définis en termes de mesures de performance, comme le temps d'exécution, le cout, la fiabilité et la consommation d'énergie... Les algorithmes d'ordonnancement de workflow sont essentiels dans le cloud, en général, ils permettent l'automatisation d'exécution des workflows scientifiques dans les environnements distribués.

1.2 Problèmes et objectifs

L'ordonnancement des workflows a toujours attiré l'attention des chercheurs industriels et universitaires. Les premières études se sont principalement focalisées sur l'optimisation mono-objectif, minimisant le temps d'exécution des workflows en utilisant des heuristiques et méta heuristiques . L'avènement du cloud computing a rendu possible l'exécution de workflows en louant des machines virtuelles à un certain coût, ce qui engendre de nouveaux objectifs à minimiser. Le problème d'ordonnancement des workflows est alors formulé comme un problème (NP hard) d'optimisation multi objective . Deux principales approches de résolution ont été considérées : la première approche combine les objectifs en un seul par l'utilisation d'une fonction d'agrégation en attribuant des poids différents à chaque objectif . En dehors de sa simplicité de mise en œuvre, trouver les bonnes valeurs des poids de la fonction d'agrégation reste un inconvénient important, car il n'est pas évident de trouver les valeurs adéquates reflétant exactement la préférence du décideur[1]. La seconde approche est basée sur le front Pareto qui offre un ensemble de solutions de compromis au décideur, afin qu'ils puissent sélectionner la solution adéquate à ses besoins. À cette fin, plusieurs méthodes ont été développées à base d'heuristique et de méta heuristique telle que NSPSO, MODE, NSGAII, SPEA2, etc.

Durant cette décennie l'ordonnancement des workflows dans le cloud a

constitué un open challenge. Les procédés d'ordonnancement les plus récent et efficace ont des gains et des améliorations plutôt minimes en termes de qualité de résultat, parfois de l'ordre de 1 % [2-4]. Les dernières contributions sont issues généralement d'hybridation d'heuristique et de méta heuristique. Une extension de l'heuristique bien connue HEFT appelée MOHEFT a été développée par Durillo et al ,elle est devenu la méthode de comparaison standard pour des méta heuristiques les plus récentes et efficaces, spécialement conçues pour le cloud computing [2-17].

Généralement dans la littérature, MOHEFT est cité pour sa généricité d'objectifs et la bonne qualité de ses résultats. Cependant, en raison de la complexité temporelle élevée plusieurs travaux ont conclu que MOHEFT ne peut pas être exécutée dans un temps acceptable, la note "N/A" non acceptable est placée dans le cas des larges workflows. MOHEFT souffre de sa grande complexité temporelle lorsqu'il s'agit de très larges workflows. À cet effet plusieurs études de la complexité temporelle ont été avancées, cependant leurs résultats sont différents et se contredisent [2-4].

Lors de l'évaluation de MOHEFT la plupart des recherches ont évoqué son paramètre d'input K. Ce paramètre K donne le nombre de solutions retourné, pour une grande valeur du paramètre K on obtient des résultats d'une meilleure qualité et un temps de calcul plus long. Lorsque le K est à l'infini, le MOHEFT conduit à une recherche exhaustive qui prend énormément de temps.

La réduction de la complexité temporelle de MOHEFT permet son utilisation avec de larges workflows ainsi que la possibilité d'utiliser de bonnes valeurs pour le paramètre K, soit des valeurs supérieure ou égale à cinquante. Dans cette thèse, le premier objectif est d'étudier et de valider la complexité temporelle de la méthode MOHEFT afin de comprendre quelles sont les parties ou les paramètres qui influent sur la vitesse d'exécution.
Le second objectif est l'étude des performances de l'utilisation et d'exploitation d'une architecture parallèle à usage général pour l'exécution de MOHEFT et d'identifier les différents critères architecturaux qui affectent l'efficacité de l'implémentation de MOHEFT. Le troisième objectif de cette thèse consiste à utiliser une nouvelle approche pour la réduction de la complexité temporelle de MOHEFT basé sur les mécanismes checkpointing et le backtra-

king, généralement ces deux mécanismes sont utilisés dans les systèmes informatiques distribués pour la reprise après une défaillance du système. Dans le cas de la méthode MOHEFT, le checkpointing et backtraking peuvent éliminer des calculs redondants.

1.3 Contributions

Sur la base de la définition de la problématique de recherche et de ses défis, cette thèse apporte les contributions clés suivantes :

• L'étude de la complexité de MOHEFT avec validation des résultats.

• Réduction de la complexité temporelle de MOHEFT et optimisation de l'accès à la mémoire.

• L'étude des performances de la parallélisation de MOHEFT.

• Une nouvelle approche d'optimisation basée sur l'intégration dans MOHEFT le Backtraking et le Checkpointing résultants une nouvelle méthode nommée FAMOBACH ; cette dernière est une version améliorée de MOHEFT qui élimine les calculs redondants.

1.4 Organisation du manuscrit

Après avoir introduit les travaux de cette thèse dans ce premier chapitre, la suite est divisée en quatre chapitres et d'une conclusion.
Les chapitres 2 et 3 présentent le contexte et l'état de l'art des domaines sur lesquels nous travaillons, à savoir : la compréhension générale du cloud computing d'une part, le workflow et l'ordonnancement de workflow dans le cloud avec optimisation multi objectif, d'autre part.
Le chapitre 4 et le chapitre 5 détaillent nos contributions, dans le chapitre 4 une étude de la complexité temporelle et l'évaluation d'une approche basée sur l'utilisation et l'implémentation de MOHEFT sur une architecture parallèle à usage général. Le chapitre 5 détaille l'approche FAMOBACH pour la réduction de la complexité temporelle de MOHEFT basé sur les mécanismes checkpointing et le backtraking.

Chapitre 2

Le Cloud Computing

2.1 Introduction

Durant les années 80, les systèmes plus puissants, connus sous le nom de superordinateurs, ont été créés en améliorant du matériel dédié. Cette solution est appelée mise à l'échelle « scaling-up ». La tendance actuelle est d'exécuter les calculs sur un grand nombre d'unités de traitement standard reliés entre elles, une solution appelée "scaling-out". En d'autres termes, le traitement d'un plus grand nombre de données consiste souvent, de nos jours, à utiliser un grand nombre de ressources soit le nombre des unités de calculs.

La parallélisation des calculs sur un grand nombre de machines pose de nombreux problèmes, tels que les communications entre ordinateurs, l'accès aux ressources partagées ou l'équilibre de la charge de travail. Les premières études n'ont cessé de fournir des solutions techniques à ces problèmes, dont certaines sont connues sous le nom d'architectures Grid Computing et Peer-to-Peer. Ces solutions ont donné naissance à des applications logicielles bien connues ou à des résultats qui ont un impact profond sur notre vie quotidienne, comme c'est le cas pour Google Search et les réseaux sociaux comme Facebook, etc.

Durant les deux dernières décennies, le nombre de sociétés de logiciels impliquées dans les calculs à grande échelle à augmenter. Cette situation a conduit à la création d'un nouveau marché économique des solutions de stockage et de calcul. Certains très grands acteurs du logiciel ont décidé de

fournir ces installations en tant que service commercial, permettant ainsi à de nouveaux acteurs d'externaliser leur solution de calcul, faisant du calcul et du stockage une facilité comme l'électricité l'est déjà. Ces nouveaux services commerciaux sont désignés sous le nom de "Cloud Computing".

Le Cloud Computing ou l'informatique en nuage fait référence à la fois aux applications fournies en tant que services sur Internet et au matériel et aux logiciels des centres de données qui fournissent ces services [18].

Du point de vue du consommateur, l'informatique en nuage permet à n'importe quel utilisateur de louer un grand nombre d'instances de calcul en quelques minutes pour effectuer des calculs intensifs et/ou stocker les données. Ces instances pouvant être dimensionnées de manière dynamique, elles permettent aux utilisateurs de répondre à des contraintes spécifiques de mise à l'échelle, c'est-à-dire d'être en mesure de s'adapter à une quantité croissante de travail. Par exemple, cette élasticité de mise à l'échelle permet aux consommateurs de faire face aux pics hebdomadaires de consommation de données. Les capacités de stockage et de calcul sont fournies en tant que service, selon le principe du "pay-as-you-go". Le Cloud Computing décharge donc les utilisateurs de l'investissement matériel.

Ce chapitre est un bref aperçu du Cloud Computing. Il est organisé comme suit : la section suivante donne une brève introduction aux origines et à l'historique du Cloud Computing. Les sections suivantes sont consacrées aux caractéristiques des environnements Cloud Computing et les types de services fournis cloud ainsi qu'une taxonomie générale des modèles de tarification.

2.2 Historique du Cloud Computing

2.2.1 HPC et ordinateur bon marché

Une grande partie des recherches et expériences en matière de calcul haute performance (HPC) est mise en œuvre sur du matériel HPC personnalisé, parfois appelé superordinateur. Si l'origine exacte des plates-formes de calcul haute performance personnalisées est plutôt incertaine, elle remonte au moins aux années 1960 et aux travaux de Seymour Cray. Ces plateformes de calcul ont été utilisées avec succès dans de nombreux domaines de recherche

tels que les prévisions météorologiques, la recherche aérodynamique, les simulations d'explosions nucléaires, la simulation de la dynamique moléculaire, la découverte de médicaments, etc.

Au cours des années 80 et 90, le nombre et la puissance des ordinateurs personnels dans le monde ont augmenté de manière très significative. Économiquement parlant, il est aujourd'hui souvent plus rentable d'empiler des unités centrales bon marché que d'acheter du matériel dédié coûteux. Parallèlement au développement du HPC sur les superordinateurs, des recherches ont donc été menées à distribuer les calculs sur du matériel de base standard. Vu cette situation il est important d'exploiter bien le matériel de base en défaveur d'un matériel spécialisé. De nos jours, la mise à l'échelle s'effectue le plus souvent horizontalement plutôt que verticalement[19].

Pour distribuer les calculs sur du matériel de base ou ordinateur bon marché, des solutions initiales ont été proposées pour mettre en réseau des ordinateurs bon marché distincts. Par exemple, le Framework PVM (Parallel Virtual Machine), développé en 1989, est un logiciel qui permet à un ensemble d'ordinateurs potentiellement hétérogènes de simuler un seul grand ordinateur. Il y a un deuxième Framework majeur, le MPI (Message Passing Interface) publié en 1994. MPI fournit une couche d'abstraction qui libère le développeur d'applications de la gestion manuelle des communications intermachines. Les systèmes matériels composés de plusieurs ordinateurs bon marché et mis en réseau par des logiciels tels que PVM ou MPI sont souvent appelés clusters Beowulf.

Dans le domaine du HPC il existe une solution à mi-chemin entre les superordinateurs et les clusters Beowulf. Cette solution combine la rentabilité de CPU standard avec le débit d'un matériel de communication dédié et personnalisé. Cette solution est donc composée de plusieurs CPU et de mémoires RAM de base, ainsi que des dispositifs d'alimentation, de carte mère et de connexions intermachines personnalisées et dédiées. Tous les différents composants sont fixés sur des racks et/ou des lames, puis rassemblés dans des armoires à rack.

Le Grid Computing est un terme général désignant les configurations matérielles et logicielles permettant de distribuer les calculs sur des CPU de base, que ce soit dans des clusters Beowulf ou dans des racks et armoires

personnalisés. La sous-section suivante décrit plus précisément le Grid Computing.

2.2.2 Grid Computing

Grid Computing est un système de calcul distribué composé de nombreux ordinateurs distincts en réseau (parfois appelés "nœuds"). Ces ordinateurs sont généralement des matériels commerciaux standard regroupés dans une plate-forme de Grid. Les différents composants sont, par conception, faiblement couplés, ce qui signifie que chaque ordinateur a une connaissance limitée de l'existence et de l'identité des autres. Une couche logicielle dédiée de bas niveau, appelée middleware, est utilisée pour surveiller et équilibrer la charge des tâches entre les nœuds. Contrairement aux superordinateurs, le réseau qui relie les différents dispositifs d'une grille peut être assez lent. Dans le cas des superordinateurs, tous les processeurs sont géographiquement très proches et la communication est assurée par un bus de données local efficace ou un réseau haute performance dédié. Dans le cas du calcul distribué, la communication passe par un réseau local (LAN) ou un réseau étendu (WAN), qui ne peut pas rivaliser avec le débit d'un bus local. Cette contrainte de communication sur le calcul distribué signifie que le calcul distribué est très bien adapté aux gros travaux par lots où les exigences de calcul sont intenses et les besoins de communication entre les machines sont faibles. Une des formes les plus connues de Grid Computing est apparue dans les années 2000, résultant de deux événements majeurs des années 90 : la démocratisation de l'ordinateur personnel et l'explosion d'Internet. Cette nouvelle forme de Grid Computing est connue sous le nom d'architecture Peer-to-peer, qui est une plate-forme de Grid Computing composée d'un ensemble collaboratif de machines réparties dans le monde entier et appelées pairs. Cette architecture supprime le modèle traditionnel client/serveur : il n'y a pas d'administration centrale et les pairs sont également privilégiés. Le modèle pair-à-pair a été popularisé par les systèmes de stockage distribués à très grande échelle, ce qui a donné lieu à des applications de partage de fichiers pair-à-pair comme Napster, Kazaa ou Freenet. Depuis lors, les architectures peer-to-peer ont également été appliquées avec succès à des tâches de calcul intensives. [19]

2.2.3 Les difficultés de la gestion d'une infrastructure

L'infrastructure informatique a évolué très rapidement, passant d'une seule machine aux grappes et aux grilles. Les grandes entreprises ont généralement besoin d'un grand nombre de machines pour héberger leurs applications professionnelles (par exemple : serveurs web, serveurs d'applications, serveurs de bases de données, serveurs de fichiers, serveurs de courrier électronique, services d'authentification, équilibreurs de charge, etc.) Ces serveurs doivent également être protégés contre les intrusions. D'autre part, les petites entreprises ont besoin de réduire l'investissement initial pour les IT afin de se concentrer sur leurs objectives. Ces exigences entraînent les difficultés suivantes pour maintenir et gérer l'infrastructure informatique de l'entreprise :

Coûts de la main-dœuvre : Les infrastructures complexes impliquent plus d'exigences en matière de déploiement, de configuration, de lancement et de reconfiguration au pendant l'exécution. Ces tâches sont soit exécutées manuellement, soit gérées automatiquement par le service informatique de l'entreprise, mais doivent encore être supervisées. Le premier déploiement de l'infrastructure (mise en place des serveurs, des réseaux, des systèmes de refroidissement) doit encore être fait manuellement. L'investissement en ressources humaines peut induire une des charges salariales conséquentes pour l'entreprise.

Coûts de la main-dœuvre : Après le déploiement, l'infrastructure doit être bien exploitée (c'est-à-dire qu'elle doit avoir une charge minimale active). Les machines inactives contribuent non seulement à l'investissement initial, mais gaspillent aussi de l'énergie. L'électricité est un élément important contribuant au coût total nécessaire au fonctionnement de l'ensemble de l'infrastructure informatique et de leurs systèmes de refroidissement. L'entreprise doit améliorer l'utilisation de l'infrastructure, afin de réduire le gaspillage des ressources et de l'énergie.

Difficultés de dimensionnement de l'infrastructure : La charge de travail de l'infrastructure ne peut pas être parfaitement prédite au moment du déploiement et d'allocation. Il existe des phases d'inactivité et des pics de charge (par exemple, respectivement pendant la nuit et les heures de travail). Si elle est surdimensionnée pour faire face aux pics de charge, l'infrastructure

sera davantage sous-utilisée pendant les périodes d'inactivité et contribuera au gaspillage des ressources. Par conséquent, elle doit avoir la capacité de s'adapter automatiquement à la charge actuelle en augmentant ou en diminuant le nombre de serveurs actifs pour répondre aux besoins des applications. Pour ce faire, la conception de l'infrastructure doit être suffisamment flexible pour faire face à ces différentes situations.

2.2.4 L'émergence du Cloud Computing

Au cours des années 2000, le boom de l'Internet a poussé certains mastodontes comme Google, Facebook, Microsoft, Amazon, etc. à relever des défis en matière de calcul et de stockage sur des ensembles de données d'une ampleur rarement rencontrée. Outre la mise à l'échelle de leurs systèmes, ils ont dû relever deux autres défis. Premièrement, ils devaient créer des abstractions sur le stockage et les calculs distribués pour faciliter l'utilisation de ces systèmes et permettre aux experts en informatique non distribuée d'exécuter des calculs à grande échelle sans s'impliquer dans le processus de parallélisation. Deuxièmement, ils étaient soucieux de minimiser le coût d'exploitation par machine dans leurs centres de données.

De telles contraintes impliquaient l'abandon partiel de l'ancien système de stockage de données ainsi que le développement de nouveaux Framework logiciels et matériels pour exprimer facilement les demandes et les besoins de calcul distribué.Les grandes entreprises ont construit des centres de données hébergeant des dizaines de milliers de machines de base et ont utilisé des technologies pour répondre à leurs besoins internes en matière de calcul et de stockage. Au départ, les centres de données ainsi que les nouveaux nouveaus systèmes de stockage et les logiciels pour les calculs distribués étaient conçus comme des outils exclusivement dédiés aux besoins internes des entreprises. Progressivement, certains de ces outils ont été transformés en produits commerciaux publics, accessibles à distance par le biais d'une interface de programmation d'applications (API) en tant que services payants. Les entreprises qui ont proposé ces offres commerciales sont devenues des fournisseurs de Cloud Computing.

2.2.5 Les avantages du Cloud computing

Le Cloud Computing est similaire au grid computing en termes de déploiement et de gestion du matériel, mais il est différent par l'utilisation de la virtualisation pour fournir les ressources. Il s'agit de la principale technologie à l'origine de la gérabilité et de la portabilité des ressources de l'infrastructure Cloud Computing. L'utilisation du Cloud pour l'exécution d'application parallèle complexe présente certains avantages qui sont les suivants :

Allocation de ressources à la demande : Le mécanisme d'allocation de ressources à la demande dans le cloud peut améliorer l'utilisation des ressources et modifier l'expérience des utilisateurs finaux pour une meilleure réactivité. Les applications basées sur le cloud peuvent obtenir des ressources allouées en fonction du nombre de taches à chaque étape de l'exécution, au lieu de réserver un nombre fixe de machines à l'avance. Les applications exécutées dans le cloud peuvent évoluer de manière dynamique. Lorsque les applications ont besoin de plus de ressources au moment de l'exécution, le cloud est plus flexible que l'infrastructure Grid, avec le cloud, de nouvelles ressources peuvent être fournies instantanément à la demande.

Déploiement d'applications : Le déploiement des applications peut être rendu flexible et pratique. Avec des serveurs bare-metal physiques, il n'est pas facile de changer le déploiement des applications et la plateforme de support sous-jacente. Toutefois, avec la technologie de virtualisation dans une plate-forme Cloud, différents environnements applicatifs peuvent être soit préchargés dans des images de machines virtuelles (VM), soit déployées dynamiquement sur des instances VM et gérées par un logiciel de virtualisation.

Mise en commun des ressources et multi-location : Les ressources du cloud sont regroupées dans d'énormes centres de données et attribuées dynamiquement aux clients au moment de l'exécution. Les ressources sont partagées entre plusieurs locataires et attribuées à chaque locataire à la demande, afin d'augmenter l'utilisation des ressources et de réduire les coûts d'exploitation et réduire l'empreinte carbone.

Une élasticité rapide : L'environnement cloud computing s'adapte à l'évolution de ses charges de travail. Ainsi, les ressources du cloud sont

automatiquement réaffectées lorsque les demandes des clients varient (augmentation ou diminution).

2.3 Types de services cloud

Les clients des services Cloud doivent faire un compromis entre un contrôle accru du matériel et des systèmes qu'il utilise et un niveau plus élevé d'abstractions de programmation qui facilite le développement d'applications. Pour répondre aux différents choix des clients Cloud, chaque fournisseur Cloud a développé des stratégies et des API distinctes. Une classification générale des offres de Cloud Computing est donc basée sur le niveau d'abstraction. En fonction du degré d'abstraction et de contrôle qu'ils présentent à l'utilisateur (figure 2.1), les services de cloud computing sont classés en trois niveaux principaux de ce que l'on appelle souvent le Cloud Computing stack, chacun étant construit sur le précédent.

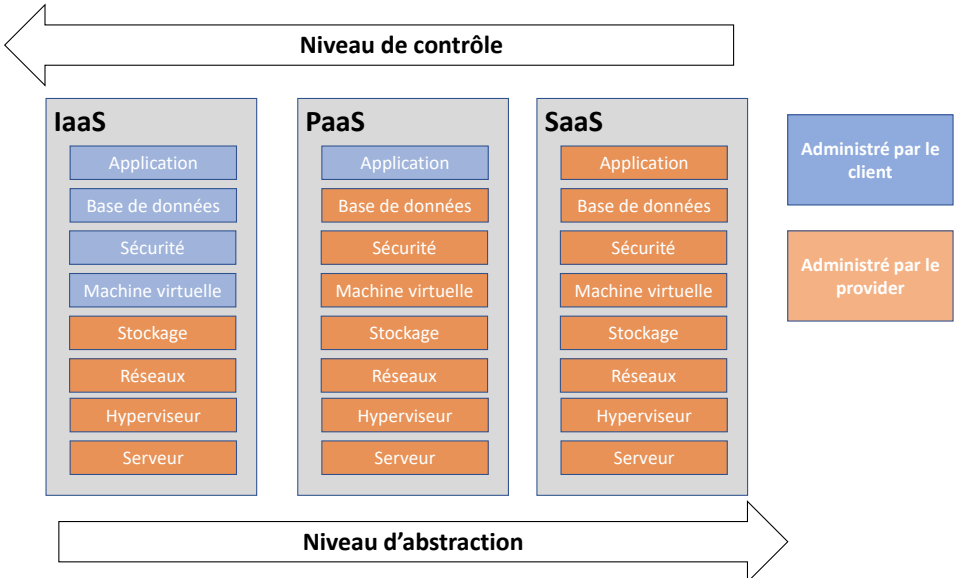


FIGURE 2.1 – Composants de chaque modèle du service cloud.

IaaS - Infrastructure as a Service : Il s'agit de la couche inférieure du service. Les utilisateurs ont la possibilité de créer et de gérer des composants bruts de traitement (machines virtuelles), de stockage ou de réseau,

mais aussi la responsabilité de toute configuration supplémentaire nécessaire à leur fonctionnement. Le fournisseur de cloud gère un catalogue de ressources d'infrastructure, telles que des images de systèmes d'exploitation, des disques ou des réseaux dédiés, à partir desquels les ressources peuvent être mises en commun. Bien qu'il existe des garanties en termes de disponibilité et de tolérance aux fautes, le fournisseur n'est pas responsable de l'interconnectivité des composants ou de leur bon fonctionnement avec une application donnée.

PaaS - Platform as a Service : À ce niveau, les utilisateurs ont accès à une variété de plateformes informatiques, dans lesquelles ils peuvent créer ou exécuter leurs propres applications. Ces plateformes sont généralement gérées par le fournisseur, dans le cadre de certaines garanties de sécurité et de disponibilité. Ces garanties sont assurées par certaines techniques de cryptage et/ou de réplication, mais de manière transparente pour les utilisateurs. Un exemple de ces services est HDInsight d'Azure [20], qui propose des outils pour créer des clusters azure pour la pile de traitement Big Data d'Apache (Pig, Hive, Spark, etc.).

SaaS - Software as a Service : Il présente le plus haut niveau d'abstraction pour les utilisateurs et nécessite une configuration minimale du backend. Les applications logicielles (parfois appelées services) sont hébergées sur des plateformes entièrement gérées et leurs infrastructures sous-jacentes. Ces applications sont généralement accessibles via des navigateurs web, des services web ou des interfaces personnalisés comme une application pour smartphone. Un exemple de SaaS se situant au-dessus des autres couches est Microsoft Office 365, qui utilise le PaaS du cloud Azure (Azure Active Directory) pour l'authentification [21] et l'IaaS (Azure Blobs) pour le stockage [20], d'une manière transparente pour ses utilisateurs.

2.4 Tarification des ressources dans le cloud

Les fournisseurs du Cloud Computing déterminent le prix de leurs services suivant différents facteurs, notamment les coûts d'exploitation, les bénéfices visés, la concurrence sur le marché, la satisfaction des consommateurs et la valeur perçue du service.

Différentes méthodes de tarification peuvent être appliquées en fonction

des critères du contrat (heures de pointe ou heures creuses, tarifs fixes ou variables, disponibilité des ressources). Il est important pour les fournisseurs de définir les stratégies de tarification appropriées pour réussir leur business, car c'est un facteur clé pour réguler l'offre et la demande, influer sur les comportements des clients et améliorer l'utilisation des ressources.

La tarification des ressources a été largement étudié par le monde universitaire et l'industrie informatique. Diverses stratégies de tarification ont été proposées dans la littérature, allant de modèles complexes à des modèles simples. Les fournisseurs de cloud existants utilisent leurs propres méthodes confidentielles d'évaluation et de tarification des services, ce qui conduit à une multitude de types et d'options de tarification parmi les fournisseurs.

Cette sous-section présente un aperçu des principaux modèles de tarification utilisés sur le marché du cloud aujourd'hui, en mettant l'accent sur les services IaaS[22].

La diversité des offres et des prix proposés par les fournisseurs a donné naissance à un secteur d'activité complexe. Une étape fondamentale pour les utilisateurs du cloud consiste à comprendre ces options de tarification, leurs avantages et leurs inconvénients, afin de sélectionner l'offre la mieux adaptée à leurs besoins et à leur budget.

D'une manière générale, la tarification des ressources est habituellement basée sur un modèle économique tel que le marché des matières premières, un taux forfaitaire ou variable. Une étude détaillée couvrant les méthodes de tarification et les paramètres appliqués par les fournisseurs IaaS a été réalisée par [23]. Cette étude souligne la grande diversité des modèles étudiés.

2.4.1 Types et modèles de tarification courants

Il existe deux types majeurs de tarification, à savoir le taux fixe et le taux variable qui évolue dans le temps en fonction des paramètres du marché.

A) Tarification fixe

Avec la tarification fixe, les fournisseurs de cloud computing fixent pour chaque service un prix de vente prédéterminé qui sera maintenu pendant une période prolongée. Les mécanismes de tarification fixe sont faciles à mettre

en œuvre et sont les plus populaires sur le marché du cloud. Les tarifications fixes les plus connues sont la tarification à la demande basée sur l'utilisation et la tarification par abonnement décrites ci-dessous :

— **La tarification à la demande basée sur l'utilisation :** Connue également sous le nom de "Pay-as-you-go", c'est le modèle de tarification le plus courant proposé par la majorité des fournisseurs IaaS (plus de 90 % selon [23]), dont Amazon [24], Google [25], Microsoft Azure [20], et bien d'autres. Ce modèle est basé sur le comptage de l'utilisation des ressources par les clients pour les facturer en conséquence. Les ressources sont quantifiées en tant qu'unités d'utilisation facturées à des prix fixes basés sur le temps (instance de VM par heure). Les clients acquièrent des ressources à la volée et ne paient que pour leur consommation, indépendamment de leur temps de demande.

  Du point de vue des utilisateurs, le modèle de tarification à la demande peut être une solution attrayante pour augmenter ou réduire rapidement les ressources, pour permettre l'expérimentation de services sans risque sans engagement à long terme et pour assurer un service garanti à un prix connu. Cependant, pour une utilisation à long terme, ce modèle de tarification peut ne pas convenir aux utilisateurs qui souhaitent minimiser leurs coûts d'approvisionnement. Pour satisfaire les clients, de nombreux fournisseurs (plus de 50% selon [23]) proposent de nouvelles alternatives de tarification permettant une utilisation plus rentable des ressources, comme détaillée ci-dessous.

— **Tarification par abonnement :** Il s'agit d'une tarification fixe basée sur le paiement d'un montant initial pour s'abonner à un service pendant une période d'engagement prédéfinie. Ce modèle de tarification est mis en œuvre par de nombreux fournisseurs IaaS (Amazon [24], Google [25], Microsoft Azure [20], etc.) avec différentes spécificités. Par exemple, Amazon propose le système de tarification "Instances réservées" [24] qui permet aux utilisateurs de réserver une instance d'une machine virtuelle pour un ou trois ans en payant un montant initial et de bénéficier en retour d'une réduction importante sur le prix d'utilisation horaire.

Certains fournisseurs cloud proposent le modèle "Prepaid VM", qui permet aux utilisateurs de ne payer qu'un abonnement pour bénéficier d'une utilisation gratuite illimitée pendant la durée du contrat.L'utilisation d'abonnements permet aux utilisateurs d'obtenir des prix plus bas tout en garantissant la disponibilité du service. Ce modèle de tarification est particulièrement rentable si l'utilisation des ressources peut être planifiée à l'avance afin d'utiliser largement les ressources réservées pendant la durée du contrat.

Ce modèle de tarification est également avantageux pour les fournisseurs, car il leur permet d'optimiser l'utilisation de leurs centres de données et d'obtenir un revenu assuré grâce aux frais d'abonnement. Cependant, ils doivent assurer la disponibilité des ressources réservées lorsqu'elles sont demandées afin de respecter les contrats SLA.

B) Tarification dynamique

La tarification dynamique consiste à se fixer un prix variable pour un même service en fonction des conditions du marché en temps réel, telles que les ressources disponibles ou la qualité de service attendue par les clients. La stratégie de tarification dynamique est le modèle le moins répandu sur le marché du cloud. Le Spot Pricing d'Amazon [24] est une politique dynamique mise en œuvre pour la vente de services IaaS. Cependant, cette stratégie de tarification a reçu la plus grande attention dans la littérature[26, 27] en raison de sa mise en œuvre complexe et de ses avantages prometteurs. Basé sur une Tarification d'instance au comptant via des enchères d'offres à prix variable, selon l'historique des prix d'Amazon, les utilisateurs peuvent acquérir des instances ponctuelles à des prix réduits de 50% à 93% par rapport aux instances à la demande. Une requête spot spécifie le type d'instance nécessaire, la zone de disponibilité, la durée de la réservation et surtout l'offre de l'utilisateur indiquant le prix horaire maximum qu'il est prêt à payer pour utiliser la ressource. Une fois envoyée, la demande reste en attente jusqu'à ce que son offre corresponde au prix spot actuel pour être satisfaite. Ce prix est fixé par le fournisseur et est censé être actualisé en fonction de l'offre et de la demande. Une fois la demande satisfaite, l'accès aux instances de la

machine virtuelle reste actif tant que le prix du marché est respecté, sinon ces instances se terminent instantanément.

Bien que les services spot ne soient pas garantis, cette tarification reste une alternative économique intéressante pour de nombreuses applications tolérant les interruptions, telles que l'exploration du Web et les tâches Map-Reduce. La tarification spot est également avantageuse pour les fournisseurs qui peuvent vendre les ressources inutilisées et gérer stratégiquement les demandes des clients en ajustant les prix.

C) Attributs de tarification et regroupement de ressources

Les fournisseurs IaaS actuels utilisent différents formats pour fournir leurs services aux clients, y compris des ressources informatiques personnalisables, des ensembles prédéfinis de ressources packagées ou entre les offres de services. quatre principaux niveaux de regroupement de ressources IaaS ont été identifiés dans [23], comme détaillé ci-dessous :

— **Stratégie de tarification regroupée :** fait référence à la pratique consistant à combiner plusieurs ressources informatiques telles que le processeur et la mémoire, en un seul package à vendre comme une ressource unique pour un seul tarif forfaitaire. Cette stratégie est pratiquée par la majorité des fournisseurs IaaS qui proposent un ensemble de bundles préconfigurés avec des capacités de ressources variées, généralement appelées instance de VM, classe de VM ou taille de VM. Différents niveaux de regroupement peuvent être utilisés, à savoir :

  — **VM Bundled :** offre des VM préconfigurées avec des capacités CPU, Mémoire et disque spécifiques, mais la bande passante est facturée séparément. De nombreux fournisseurs tels qu'Amazon[24], Microsoft Azure [20], Google [25] et IBM [28] proposent ce type de regroupement.

  — **Entièrement intégré :** offre des instances de VM avec un processeur, une mémoire, des capacités de disque prédéfinis et une bande passante de transfert de données illimitée. Serveur Dédié-Arsys Cloud [29] fournit ce type de tarification.

— **Stratégie de tarification dégroupée :**Cette Stratégie permet aux clients d'acheter des ressources informatiques individuelles à un niveau précis pour configurer eux-mêmes leurs machines virtuelles. Les ressources demandées sont facturées séparément par unité d'utilisation (par exemple 0,01875$ par CPU/heure, 0,04$ par Go de bande passante, 6,48$ pour 500 Mhz de CPU par mois, etc.. .). CloudSigma [30] et elastichosts [**elastichosts**] sont deux fournisseurs bien connus proposant ce type de tarification. Choisir entre des offres de services groupées ou dégroupées n'est pas une décision triviale et dépend principalement des caractéristiques et des exigences de la charge de travail de l'utilisateur.

Les services groupés sont favorables lorsque la plupart des ressources packagées sont nécessaires, sinon il est plus avantageux d'acheter des ressources dégroupées pour éviter de payer des capacités inutilisées et permettre une élasticité granulaire fine. Sans perte de généralité, notre étude d'optimisation suppose une stratégie de tarification groupée proposant des instances de VM avec une quantité prédéfinie de CPU et de mémoire. Pour les tarifs, des politiques fixes et dynamiques sont utilisées pour alimenter les algorithmes d'allocation. Alors que des prix fixes sont utilisés pour facturer les demandes des utilisateurs finaux, des prix dynamiques axés sur la demande sont appliqués pour l'échange de ressources au sein de la fédération.

2.5 Ordonancement et Optimisation de l'allocation de ressources dans le Cloud Computing

2.5.1 Objectifs d'ordonnancement

La prise en compte des coûts est généralement le point en commun de tous les algorithmes étudiés. Outre cet objectif, la plupart des algorithmes prennent également en compte certaines mesures de performance, comme le temps d'exécution total ou la balance de charge des tâches exécutées par le

système. De plus, certains algorithmes intègrent également la consommation d'énergie, la fiabilité et la sécurité dans leurs objectifs. Dans la partie suivante, la définition des objectifs d'ordonnancement inclut dans la Figure 2.2 qui sont abordés dans la littérature [1] :

Le coût : Les algorithmes conçus pour les plateformes cloud computing doivent tenir compte du coût de la location de l'infrastructure. Autrement, le coût de la location des VM, du transfert des données et de l'utilisation du stockage en cloud peut être considérablement élevé. Cet objectif est pris en compte dans les algorithmes d'ordonnancement soit en essayant le minimisant, soit en fixant un prix plafond à ne pas dépasser (c'est-à-dire le budget). Généralement tous les algorithmes étudiés équilibrent le coût avec d'autres objectifs liés aux performances ou à des exigences non fonctionnelles tels que la sécurité, la fiabilité et la consommation d'énergie. Par exemple, l'exigence de qualité de service la plus courante consiste à minimiser le coût total tout en respectant une contrainte de délai définie par l'utilisateur.

Makespan : La plupart des algorithmes étudiés s'intéressent au temps nécessaire à l'exécution du workflow, ou makespan. Comme pour le coût, il est inclus dans les objectifs de l'ordonnancement, soit en essayant de minimiser sa valeur, soit en définissant une limite de temps, ou date limite, pour l'exécution du workflow.

Maximisation de Workload : Les algorithmes développés pour ordonnancer les ensembles de taches visent généralement à maximiser la quantité de travail effectué, c'est-à-dire le nombre de workflows exécutés. Cet objectif est toujours associé à des contraintes telles que le budget ou le délai et, par conséquent, les stratégies de cette catégorie visent à exécuter autant de workflows que possible avec l'argent donné ou dans le délai spécifié.

Maximisation de l'utilisation de la VM : La plupart des algorithmes répondent indirectement à cet objectif en tenant compte des coûts. Les créneaux horaires inactifs dans les machines virtuelles louées sont considérés comme un gaspillage d'argent, car ils ont été payés, mais non utilisés, et les algorithmes essaient donc d'éviter ce cas dans leurs planifications. Cependant, il n'est pas rare que ces créneaux inutilisés proviennent de l'exécution d'un workflow, principalement en raison des dépendances entre les tâches et des exigences de performance. Certains algorithmes cherchent directement

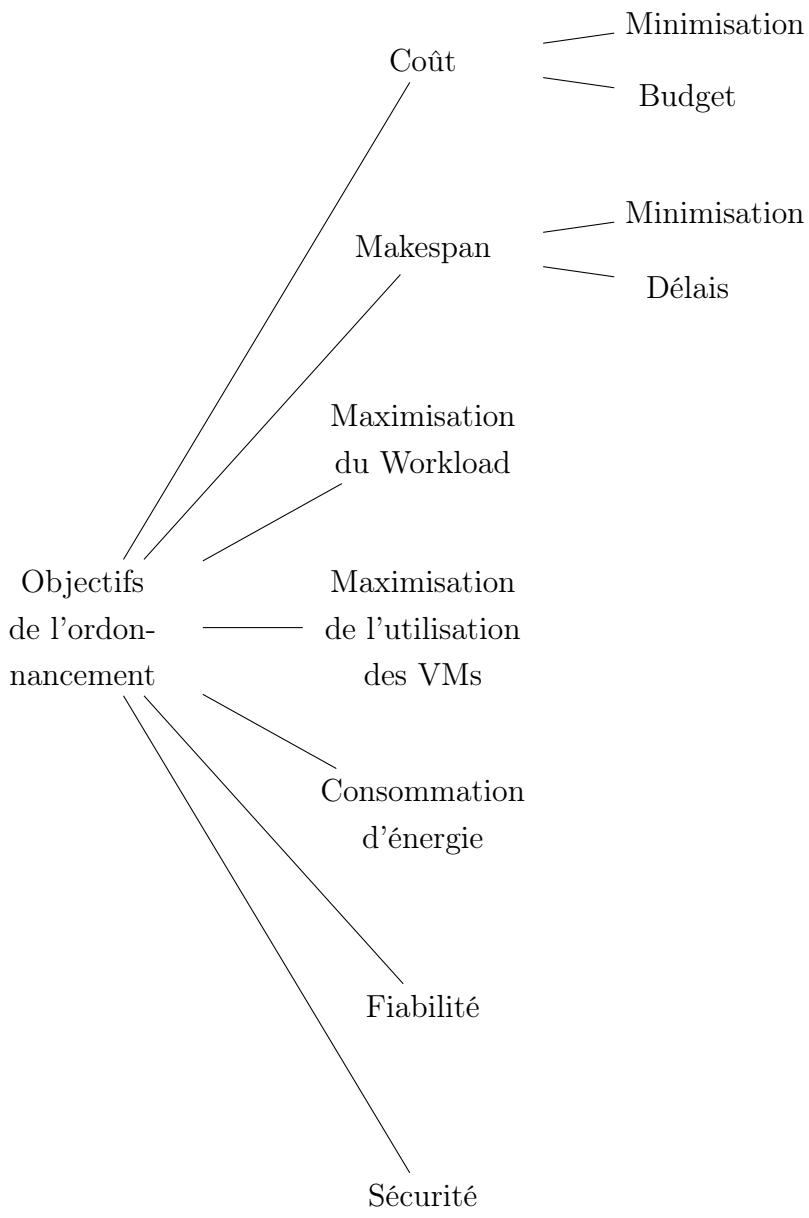


FIGURE 2.2 – Un schéma expliquant les objectifs d'optimisation.

à minimiser ces créneaux horaires inutilisés et à maximiser l'utilisation des ressources, ce qui présente des avantages pour les utilisateurs en termes de coût, et pour les fournisseurs en termes de consommation d'énergie, de profit et d'utilisation plus efficace des ressources.

Consommation d'énergie : Minimisation de la consommation d'éner-

gie. Les particuliers, les organisations et les gouvernements du monde entier se préoccupent de plus en plus de la réduction de l'empreinte carbone afin d'atténuer l'impact sur l'environnement. Bien qu'elle ne soit pas propre au cloud computing, cette préoccupation a également attiré l'attention des chercheurs dans ce domaine. Les algorithmes prenant en compte l'énergie consommée par l'exécution font un compromis entre la consommation d'énergie, les performances et le coût. De plus, la virtualisation et le manque de contrôle et de connaissance de l'infrastructure physique limitent leurs capacités et rendent le problème encore plus complexe.

la fiabilité : Les algorithmes qui considèrent la fiabilité comme faisant partie de leurs objectifs ont des mécanismes pour garantir que l'exécution du workflow est achevée dans le respect des contraintes de qualité de service des utilisateurs, même en cas d'échec des ressources ou des tâches. Certaines approches courantes consistent à répliquer les tâches critiques et à s'appuyer sur le checkpointing pour replanifier les tâches défaillantes. Cependant, les algorithmes doivent tenir compte des coûts supplémentaires associés à la réplication des tâches et au stockage des données pour les checkpointing.

la sécurité : Certaines applications scientifiques peuvent exiger que les données d'entrée ou de sortie soient traitées de manière sécurisée. Plus encore, certaines tâches peuvent être composées de calculs sensibles qui doivent être sécurisés. Les algorithmes concernés par ces questions de sécurité peuvent tirer parti de différents services de sécurité offerts par les fournisseurs IaaS. Ils peuvent gérer les données de manière sécurisée en les considérant comme inamovibles, ou gérer les tâches et les données sensibles de manière à ce que des ressources ou des fournisseurs ayant un niveau de sécurité plus élevé soient utilisés pour les exécuter et les stocker. La prise en compte de ces mesures de sécurité a un impact sur les décisions d'ordonnancement, car ils peuvent influer sur les estimations de temps et de coûts.

2.5.2 Principe de l'optimisation multi objectif

L'optimisation consiste à trouver une ou plusieurs solutions qui correspondent à la minimisation (ou à la maximisation) d'un ou de plusieurs objectifs spécifiés et qui satisfont toutes les contraintes (le cas échéant). Un

problème d'optimisation à objectif unique implique une seule fonction objective et aboutit généralement à une seule solution, appelée solution optimale. En revanche, un problème d'optimisation multi objectif prend en compte plusieurs objectifs conflictuels simultanément. Dans ce cas, il n'y a généralement pas de solution optimale unique, mais un ensemble d'alternatives avec différents compromis, appelées solutions optimales de Pareto ou solutions non dominées.

Malgré l'existence de multiples solutions optimales de Pareto, dans la pratique, une seule de ces solutions doit généralement être choisie. Ainsi, par rapport aux problèmes d'optimisation à objectif unique, l'optimisation multi objectif comporte au moins deux étapes d'égale importance : une étape d'optimisation pour trouver les solutions optimales de Pareto et une étape de prise de décision pour choisir une seule solution préférée. Cette dernière étape nécessite généralement des informations sur les préférences d'un décideur (DM).

Modélisation d'un problème d'optimisation

Avant toute optimisation, le problème doit d'abord être modélisé. En fait, la construction d'un modèle mathématique ou computationnel approprié pour un problème d'optimisation est aussi importante ou aussi critique que la tâche d'optimisation elle-même. Typiquement, la plupart des livres consacrés aux méthodes d'optimisation supposent implicitement que le problème a été correctement spécifié. Cependant, dans la pratique, ce n'est pas toujours le cas. La quantification et la discussion des aspects de modélisation dépendent largement du contexte réel du problème sous-jacent. Toutefois, nous souhaitons souligner les points suivants. Premièrement, la construction d'un modèle approprié (c'est-à-dire la formulation du problème d'optimisation avec la spécification des variables de décision, des objectifs, des contraintes et des limites de variables est une tâche importante. Deuxièmement, un algorithme d'optimisation (qu'il soit unique ou multiobjectif) trouve les optima du modèle du problème d'optimisation spécifié et non du véritable problème d'optimisation. En raison de ces raisons, les solutions optimales trouvées par un algorithme d'optimisation doivent toujours être analysées (par le biais d'une analyse de

post-optimalité) pour leur "adéquation" dans le contexte du problème. Cet aspect rend la tâche d'optimisation itérative dans le sens où, si l'analyse de post-optimalité révèle des divergences dans les solutions optimales obtenues, le modèle d'optimisation peut devoir être modifié et l'étape d'optimisation doit être exécutée à nouveau.

Définition 2.1 *Problème d'optimisation multiobjectif.*

Un problème d'optimisation multiobjectif peut être défini comme suit :

$$\text{MOP } = \begin{cases} \min F(x) = (f_1(x), f_2(x), \ldots, f_n(x)) \\ \text{s.c. } x \in S \end{cases}$$

où $n(n \geq 2)$ est le nombre d'objectifs, $x = (x_1, \ldots, x_k)$ est le vecteur représentant les variables de décision, et S représente l'ensemble des solutions réalisables associées aux contraintes d'égalité et d'inégalité et aux limites explicites. $F(x) = (f_1(x), f_2(x), \ldots, f_n(x))$ est le vecteur d'objectifs à optimiser. L'espace de recherche S représente l'espace de décision ou l'espace de paramètre du MOP. L'espace auquel appartient le vecteur objectif est appelé espace objectif. Le vecteur F peut être défini comme une fonction de coût à partir de l'espace de décision dans l'espace objectif qui évalue la qualité de chaque solution $(x_1, \ldots, x_k)$ en attribuant un vecteur objectif $(y_1, \ldots, y_n)$, qui représente la qualité de la solution (ou fitness). Dans le domaine de l'optimisation multiobjective, le décideur l'utilise pour travailler en termes d'évaluation d'une solution sur chaque critère, et se place naturellement dans l'espace objectif. L'ensemble $Y = F(S)$ représente les points réalisables dans l'espace objectif, et $y = F(x) = (y_1, y_2, \ldots, y_n)$, où $y_i = f_i(x)$, est un point de l'espace objectif. Il n'est pas habituel d'avoir une solution x^*, associée à un vecteur variable de décision, où x^* est optimale pour tous les objectifs :

$$\forall x \in S, f_i(x^*) \leq f_i(x), \quad i = 1, 2, \ldots, n$$

Étant donné que cette situation n'est pas habituelle dans les MOP réels où les critères sont en conflit, d'autres concepts ont été établis pour tenir compte de l'optimalité. Une relation d'ordre partiel pourrait être définie, connue sous le nom de relation de dominance.

Définition 2.2 *Domination de Pareto.*

Un vecteur objectif $u = (u_1, \ldots, u_n)$ est dit dominer $v = (v_1, \ldots, v_n)$ (noté $u \prec v$) si et seulement si aucun composant de v n'est plus petit que le composant correspondant de u et qu'au moins un composant de u est strictement plus petit, c'est-à-dire,

$$\forall i \in \{1, \ldots, n\} : u_i \leq v_i \wedge \exists i \in \{1, \ldots, n\} : u_i < v_i$$

Le concept généralement utilisé est l'optimalité de Pareto. La définition de l'optimalité de Pareto provient directement du concept de dominance. Le concept a été initialement proposé par F.Y. Edgeworth en 1881 [1] et étendu par W. Pareto en 1896[1]. Une solution optimale de Pareto indique qu'il est impossible de trouver une solution qui améliore les performances sur un critère sans diminuer la qualité d'au moins un autre critère.

Définition 2.3 *Optimalité de Pareto.*

Une solution $x^* \in S$ est optimale de Pareto si pour chaque $x \in S, F(x)$ ne domine pas $F(x^*)$, c'est-à-dire $F(x)F(x^*)$.

Graphiquement, une solution x^* est optimale de Pareto s'il n'y a pas d'autre solution x telle que le point $F(x)$ se trouve dans le cône de dominance de $F(x^*)$ qui est la case définie par $F(x)$ avec ses projections sur les axes et l'origine (Figure 2.3). En général, la recherche dans un problème monoobjectif conduit à trouver une solution optimale globale unique. Un MOP peut avoir un ensemble de solutions connu sous le nom d'ensemble optimal de Pareto. L'image de cet ensemble dans l'espace objectif est désignée comme le front de Pareto.

Définition 2.4 *Ensemble optimal de Pareto.*

Pour un MOP (F, S)donné, l'ensemble optimal de Pareto est défini comme $\mathcal{P}^* = \{x \in S / x' \in S, F(x') \prec F(x)\}$.

Définition 2.5 *Front Pareto.*

Pour un MOP (F, S) donné et son ensemble optimal de Pareto $\mathcal{P}^*$, le front de Pareto est défini comme $\mathcal{PF}^* = \{F(x), x \in \mathcal{P}^*\}$. Le front de Pareto est l'image de l'ensemble optimal de Pareto dans l'espace objectif.

Obtenir le front de Pareto d'un MOP est l'objectif principal de l'optimisation multiobjective. Cependant, étant donné qu'un front de Pareto peut contenir un grand nombre de points, une bonne approximation du front de Pareto peut contenir un nombre limité de solutions de Pareto, qui doivent être aussi proches que possible du front de Pareto exact, ainsi que celles-ci doivent être uniformément réparties sur le front de Pareto. Sinon, l'approximation obtenue du front de Pareto ne serait pas très utile au décideur qui devrait disposer d'une information complète sur le front de Pareto.

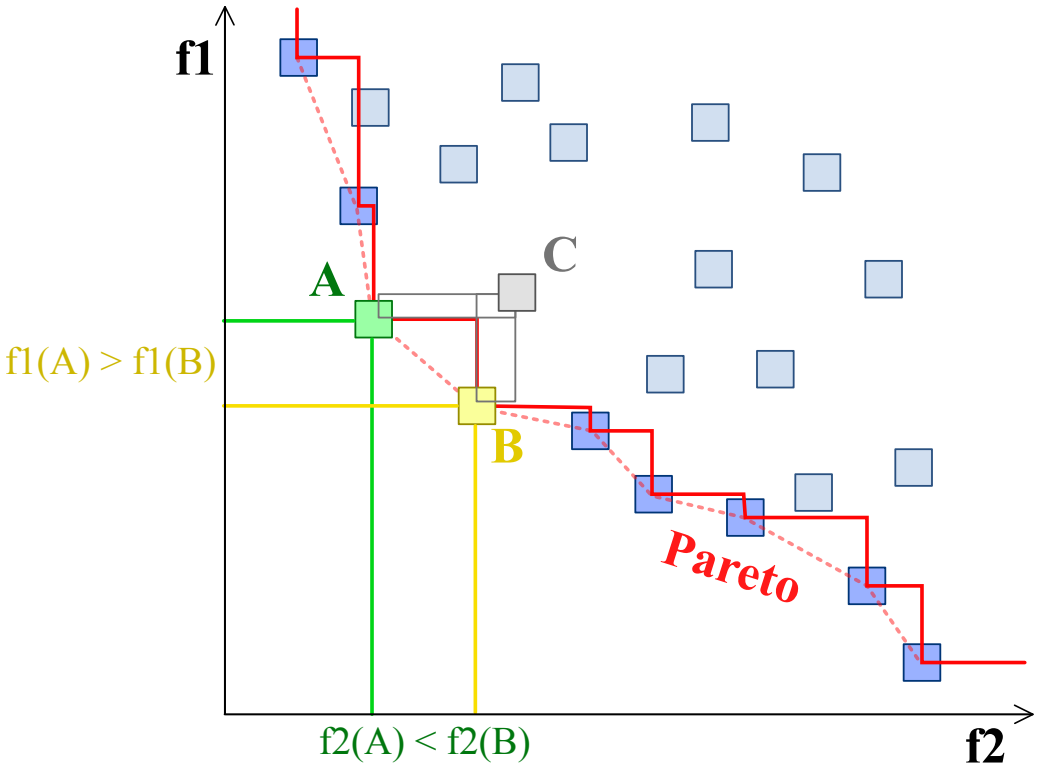


FIGURE 2.3 – Solutions non dominées dans l'espace d'objectifs.

Définition 2.6 *Vecteur idéal.*

Un point $y^* = (y_1^*, y_2^*, \ldots, y_n^*)$ est un vecteur idéal s'il minimise chaque fonction objective f_i dans $F(x)$, c'est-à-dire $y_i^* = \min(f_i(x)), x \in S, i \in [1, n]$

Le vecteur idéal est généralement une solution utopique dans le sens où il n'est pas une solution réalisable dans l'espace de décision. Dans certains cas, le décideur définit un vecteur de référence, exprimant l'objectif à atteindre pour chaque objectif. Cela généralise le concept de vecteur idéal. Le décideur peut spécifier certains niveaux d'aspiration $\bar{z}_i, i \in [1, n]$ à atteindre pour chaque fonction objective f_i. Les niveaux d'aspiration représentent des niveaux acceptables ou souhaitables dans l'espace objectif. Une solution optimale de Pareto satisfaisant tous les niveaux d'aspiration est appelée solution satisfaisante [1].

Définition 2.7 *Point de référence.*

Un point de référence $z^* = [\bar{z}_1, \bar{z}_2, \ldots, \bar{z}_n]$ est un vecteur qui définit le niveau d'aspiration (ou objectif) $\bar{z}_i$ à atteindre pour chaque objectif f_i.

Définition 2.8 *Nadir point.*

Un point $y^* = (y_1^*, y_2^*, \ldots, y_n^*)$ est le point nadir s'il maximise chaque fonction objective f_i de F sur l'ensemble de Pareto, c'est-à-dire $y_i^* = \max(f_i(x)), \quad x \in \mathcal{P}^*, i \in [1, n]$

Les points idéal et nadir donnent quelques informations sur les gammes de l'avant optimal de Pareto (Figure 2.4).

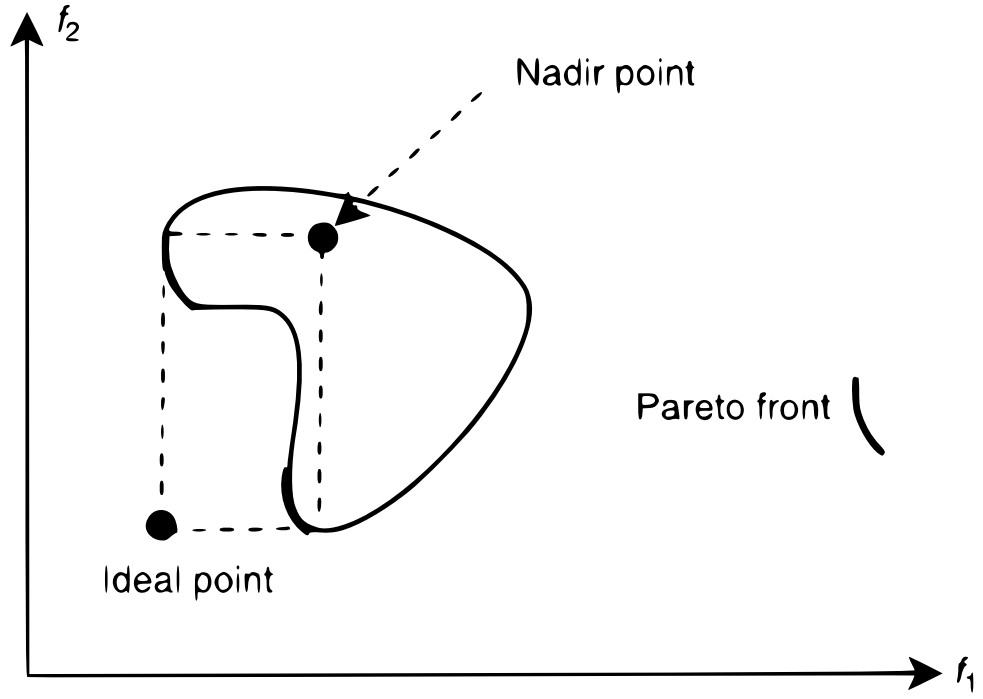


FIGURE 2.4 – Points Nadir et Idéal dans un MOP.

Définition 2.9 *Faible dominance.*

Un vecteur objectif $u = (u_1, \ldots, u_n)$ est dit dominer faiblement $v = (v_1, \ldots, v_n)$ (noté uv) si tous les composants de u sont inférieurs ou égaux aux composants correspondants de v, c'est-à-dire $\forall i \in \{1, \ldots, n\}, u_i \leq v_i$

Définition 2.10 *Dominance stricte.*

Un vecteur objectif $u = (u_1, \ldots, u_n)$ est dit dominer strictement $v = (v_1, \ldots, v_n)$ (noté $u \prec\prec v$) si tous les composants de u sont plus petits que les composants correspondants de v, c'est-à-dire $\forall i \in \{1, \ldots, n\}, u_i < v_i$

Définition 2.11 ϵ*-Dominance.*

Un vecteur objectif $u = (u_1, \ldots, u_n)$ est dit ϵ-dominate $v = (v_1, \ldots, v_n)$ (noté $u \prec_\epsilon v$) si et seulement si aucun composant de v n'est plus petit que le composant correspondant de $u - \epsilon$ et qu'au moins un composant de $u - \epsilon$ est strictement meilleur, c'est-à-dire $\forall i \in \{1, \ldots, n\} : u_i - \epsilon_i \leq v_i \wedge \exists i \in \{1, \ldots, n\} : u_i - \epsilon_i < v_i$

2.6 Conclusion

Ce chapitre, dans sa première partie, passe en revue les concepts de base, les caractéristiques, les modèles de service et les modèles de déploiement des environnements de cloud computing. Dans la deuxième partie, nous avons commencé par présenter brièvement quelques concepts sur l'optimisation puis nous avons défini les problèmes d'optimisation multiobjectifs et les éléments de base relatifs.

Nous consacrons le prochain chapitre à la présentation des Workflows et de leurs caractéristiques ainsi que l'état de l'Art de la planification des workflows dans le cloud.

Chapitre 3

Workflow

3.1 Introduction

Dans les entreprises commerciales, le concept de workflow est en tant qu'outil de modélisation des processus métier. Ces workflows métiers visent à automatiser et à optimiser les processus d'une organisation, vus comme une séquence ordonnée d'activités, et constituent un domaine de recherche mature mené par la Workflow Management Coalition (WfMC), fondée en 1993[31] Cette notion de workflow a été adaptée par la communauté scientifique où les workflows scientifiques sont utilisés pour prendre en charge des processus scientifiques complexes à grande échelle. Ils sont conçus pour mener des expériences et prouver des hypothèses scientifiques en gérant, analysant, simulant et visualisant des données scientifiques. La planification d'un workflow demeure toujours un problème difficile qui entre dans la catégorie des problèmes NP-difficiles auxquels trouver une solution exacte est inatteignable. Le problème de la planification des workflows est généralement formulé comme un problème d'optimisation multiobjectif (MOP) qui vise à optimiser au moins deux critères conflictuels.

3.2 Workflow Scientifique

Les workflows sont omni présents dans les différents domaines scientifiques, notamment la physique, l'astronomie, la biologie, la chimie[32, 33], la

science des tremblements de terre [34] et bien d'autres. Un workflow scientifique est une spécification de haut niveau d'un ensemble de tâches nécessaires à la gestion d'un processus de calcul scientifique ou d'ingénierie et les dépendances entre elles qui doivent être satisfaites afin d'atteindre un objectif spécifique. Aujourd'hui, les workflows scientifiques demandent de plus en plus de capacités de traitement de données et il est commun que les workflows sollicitent un traitement de quelques millions de tâches [32]. Par exemple, le workflow Montage Galactic Plane comporte environ 18 millions d'images d'entrée soit 2,5 To de données et 10,5 millions de tâches de calcul, il utilise environ 34 000 heures de CPU. Le workflow CyberShake [34] est une plateforme logicielle qui utilise la modélisation de formes d'ondes 3D pour l'analyse probabiliste des risques sismiques. CyberShake compte environ 840 000 tâches à s'exécuter qui sont nécessaires pour le calcul des risques sismiques d'un seul site géographique. La Californie du Sud compte environ 200 sites géographiques.

Un workflow peut être modélisé comme un graphe orienté acyclique (DAG). Chaque nœud du DAG représente une tâche du workflow, et les arêtes représentent les dépendances entre les tâches (t) qui contraignent l'ordre suivant lequel les tâches sont exécutées. Chaque tâche est un programme avec un ensemble de paramètres qui doit être exécuté. Les dépendances des taches du workflow représentent généralement les dépendances dans l'application, où les fichiers de sortie produits par une tâche sont nécessaires comme entrées d'une autre tâche.

3.2.1 Le cycle de vie d'un workflow scientifique

Le cycle de vie d'un workflow est classé en quatre phases principales [35, 36] : la composition (composition d'un résumé de workflow), la planification (mise en correspondance avec les ressources disponibles), l'exécution (exécution des tâches individuellement) et la récupération des résultats avec les données de provenance. Ces quatre phases sont présentées :

Composition : La composition du workflow permet à l'utilisateur de spécifier les phases et les dépendances entre les tâches d'une manière abstraite. Différents langages/outils de création, tels que le graphe orienté acy-

clique (DAG), où les nœuds représentent les tâches individuelles et les arêtes représentent les dépendances entre les tâches en XML (DAX), peuvent être utilisés pour composer un workflow. Le workflow composé à ce stade n'a pas d'informations sur ressources utilisées pour les calculs, elles sont spécifiées dans la phase de planification.

Planification : Dans cette phase, les taches d'un workflow sont mappées sur les ressources informatiques en vue de leurs exécutions. Les procédures de mise en correspondance et de planification transforment un workflow donné pour le rendre exécutable sur une infrastructure Grid ou Cloud.

Exécution : Dans cette phase, les jobs du workflow sont uploadés vers les ressources d'exécution en fonction des informations issues de la planification du workflow. Ces ressources d'exécution sont fournies par l'infrastructure Grid ou par le Cloud Computing. Durant cette phase, l'état d'un workflow et de ses tâches sont surveillés en permanence pour détecter l'achèvement ou l'échec de chaque tâche. Une fois qu'une tâche ou un job termine avec succès, sa sortie est récupérée pour l'utilisateur.

Provenance : Lors de la récupération de la sortie du workflow, sa provenance associée est également capturée. Les informations de provenance comprennent l'environnement d'exécution des tâches, les paramètres utilisés et les données d'entrée fournies à une tâche, les sorties intermédiaires et la sortie finale produite durant l'exécution du workflow. Ces informations sont importantes pour les scientifiques, car elles leur permettent de comprendre chaque phase du workflow et puis de modifier le workflows en conséquence. De plus, les informations sur la provenance fournissent un meilleur aperçu en cas d'échec d'un job pour un éventuel débogage. La provenance fournit l'historique des données, ce qui assure la traçabilité et l'authenticité des données consommées ou produites, ce qui est essentiel dans le Cloud.

3.2.2 Illustration Workflow scientifique

Dans les applications scientifiques, les workflows sont utilisés pour gérer des processus scientifiques complexes à grande échelle. Ils sont conçus pour mener des expériences et de prouver des hypothèses scientifiques en gérant, analysant, simulant et visualisant des données scientifiques. Cinq applications

du workflow scientifique sont citées cidessous :

a) LIGO : Les workflows de LIGO (Laser Interferometer Gravitational Wave Observatory) [32] sont utilisés pour rechercher des signatures d'ondes gravitationnelles dans les données collectées par des interféromètres à grande échelle. La mission de l'observatoire est de détecter et de mesurer les ondes gravitationnelles prédites par la théorie de la relativité générale d'Einstein, dans laquelle la gravité est décrite comme étant due à la courbure de l'espace-temps. Le traitement efficace de ces données a permis l'identification de ces ondes et "marque le début d'une nouvelle ère de l'astronomie des ondes gravitationnelles où les possibilités de découverte sont aussi riches et illimitées. Le workflow LIGO Inspiral est un workflow à forte demande de données. La figure 3.1 montre une version simplifiée de ce workflow. Le workflow LIGO Inspiral est séparé en plusieurs groupes de tâches interconnectées, appelés branches. Chaque branche peut toutefois comporter un nombre différent de pipelines.

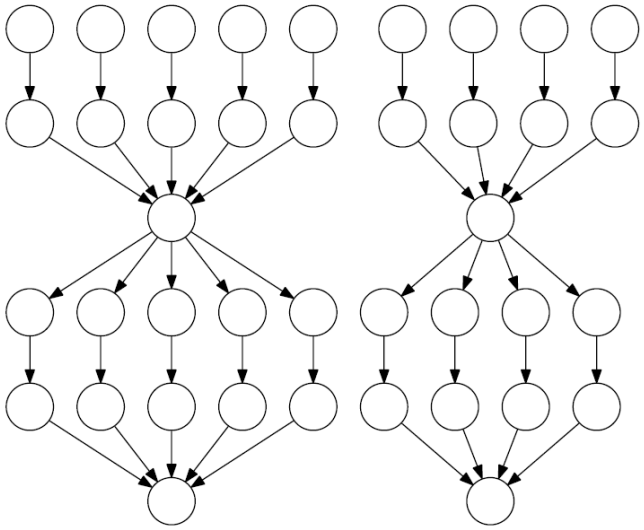


FIGURE 3.1 – Une visualisation simplifiée du workflow LIGO Inspiral.

b) Montage : Le workflow Montage [33] est une application d'astronomie caractérisée par son intensité d'E/S, utilisée pour créer des mosaïques du ciel à partir d'un ensemble d'images d'entrée. Il permet aux astronomes de générer une image composite d'une région du ciel qui est trop grande

pour être produite par des caméras astronomiques ou qui a été mesurée avec différentes longueurs d'onde et différents instruments. Durant l'exécution du worklfow, la géométrie de l'image de sortie est calculée à partir de celle des images d'entrée. Ensuite, les données d'entrée sont ré-projetées afin qu'elles aient la même échelle spatiale et la même rotation. Cette opération est suivie d'une normalisation de l'arrière-plan de toutes les images. Enfin, toutes les images d'entrée traitées sont fusionnées pour créer la mosaïque finale de la région du ciel étudié. La structure de ce workflow est illustrée à la figure 3.2.

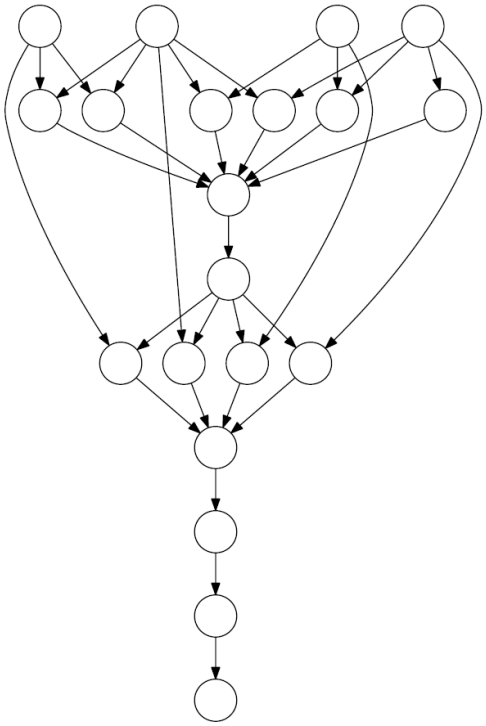


FIGURE 3.2 – Une visualisation simplifiée du workflow Montage.

c) Cybershake : Cybershake [34] est une application de sismologie qui calcule les courbes probabilistes de risque sismique pour des sites géographiques de la région de Californie du Sud. Elle identifie toutes les ruptures dans un rayon de 200 km du site d'intérêt et convertit la définition de la rupture en plusieurs variations de rupture avec différents emplacements d'hypocentre et des distributions de glissement. Il calcule ensuite des sis-

mogrammes synthétiques pour chaque variance de rupture, et les mesures d'intensité maximale sont ensuite extraites de ces synthétiques et combinées avec les probabilités de rupture originales pour produire des courbes probabilistes d'aléas sismiques pour le site. La figure 3.3 présente une illustration du workflow Cybershake.

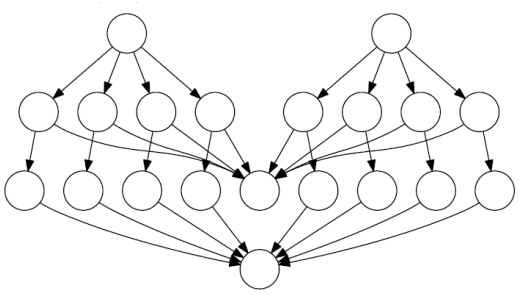


FIGURE 3.3 – Une visualisation simplifiée du workflow Cybershake.

d) Epigenomics : Le workflow Epigenomics [32] est une application à forte consommation de CPU. Les données initiales sont acquises auprès de l'analyseur génétique Illumina-Solexa sous la forme de séquences d'ADN. Chaque machine Solexa peut générer plusieurs séquences d'ADN. Ensuite, le workflow fait correspondre les séquences d'ADN aux emplacements corrects dans un génome de référence. Cela génère une carte qui affiche la densité des séquences montrant combien de fois une certaine séquence s'exprime à un endroit particulier du génome de référence. Une structure simplifiée de l'épigénomique est présentée à la figure 3.4.

e) SIPHT : Il est utilisé pour automatiser le processus de recherche des gènes encodés par les sRNA pour tous les réplicons bactériens dans la base de données du National Centre for Biotechnology Information [32]. La prédiction et l'annotation à grande échelle des gènes encodés pour les sRNA impliquent une variété de programmes individuels qui sont exécutés dans l'ordre approprié à l'aide de Pegasus [32]. Il s'agit de la prédiction des terminaisons transcriptionnelles indépendantes , des comparaisons BLAST (Basic Local Alignment Search Tools) des régions inter-géniques de différents réplicons et de l'annotation de tout sARNs trouvé. Une structure simplifiée workflow SIPHT est présentée à la figure 3.5.

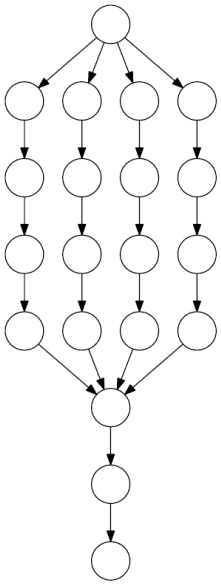


FIGURE 3.4 – Une visualisation simplifiée du workflow Epigenomics.

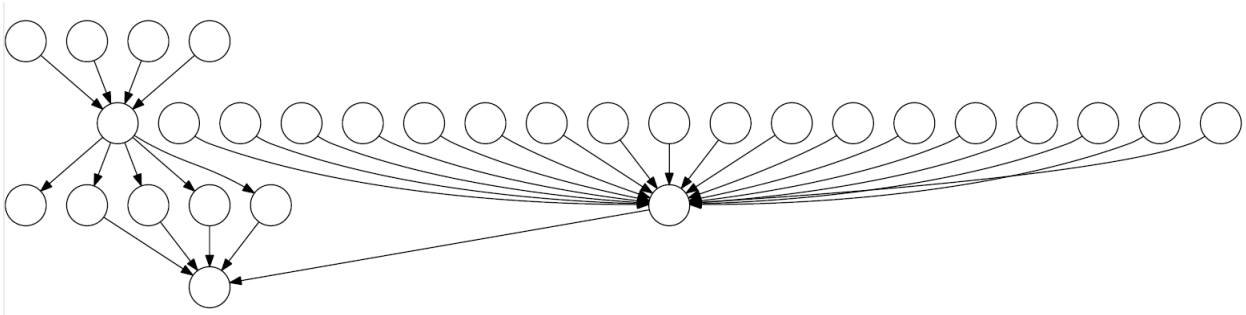


FIGURE 3.5 – Une visualisation simplifiée du workflow SIPHT.

3.2.3 Statistiques des workflows scientifiques

Le tableau 3.1 présente des statistiques sur les principales caractéristiques des workflows : nombre de tâches, volume moyen des données et temps d'exécution moyen des tâches pour les cinq workflows[32].

Workflow	Nombre de taches	Volume moyen de données	Temps d'exécution moyen
LIGO	800	5MB	228s
Montage	300	3MB	11s
CyberShake	700	148MB	23s
Epigenomics	165	355MB	2952s
SIPHT	1000	360KB	180s

TABLE 3.1 – Statistiques sur les caractéristiques des workflows scientifiques.

3.3 Déploiement des Workflows sur Cloud

La planification des tâches de workflow dans les plates-formes distribuées a été largement étudiée au fil des ans. Les chercheurs ont développé des algorithmes adaptés à différents type d'environnements ; des clusters homogènes avec un ensemble limité de ressources aux grilles communautaires à grande échelle, en passant par le paradigme le plus récent, le Cloud basée sur des services publics, hétérogène avec une grande abondance en ressources. Dans cette section, nous nous intéressons à État de l'art des algorithmes de planification des workflows scientifique sur les environnements IaaS. Comme illustré à la figure 3.6,il s'agit d'algorithmes dans lesquels le mappage de tâche vers la machine virtuelle VM est produit à l'avance et exécuté une seule fois.

Un tel plan n'est pas modifié pendant l'exécution et le moteur de workflow doit s'y conformer quel que soit l'état des tâches et des ressources. La modification de planification ou le retard dans l'exécution d'une tâche aura un effet domino sur le reste des taches, ainsi particulièrement influant sur l'exécution des taches descendantes.

Le principal avantage des planificateurs est leur capacité à générer des planings de bonne qualité en utilisant des techniques d'optimisation globales au niveau du workflow et à comparer différentes solutions avant de choisir la mieux adaptée.

La grande majorité des algorithmes se concentrent sur la génération de solutions approximatives ou quasi optimales. Pour la catégorie sous-optimale, nous identifions trois différentes classes. Les deux premières sont des approches heuristiques et méta-heuristiques telles que définies par Yu et al. [37]. Nous ajoutons à ces classes une troisième catégorie hybride pour in-

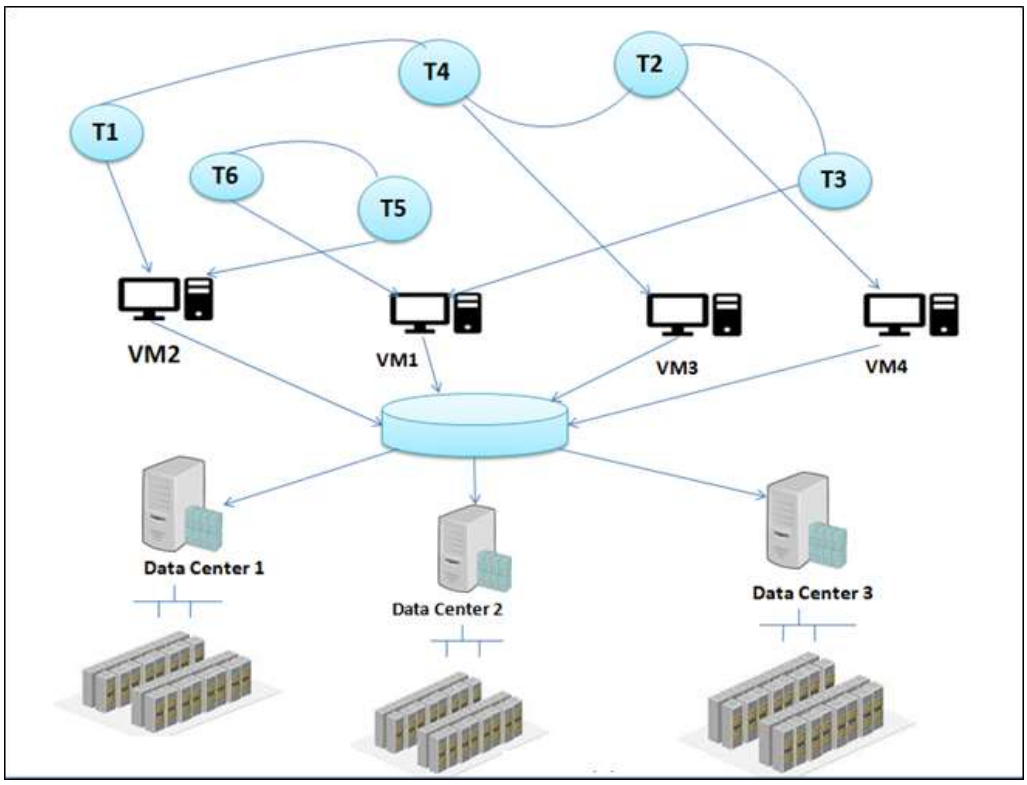


FIGURE 3.6 – Un schéma expliquant l'utilisation du cloud pour l'exécution d'un workflow.

clure des algorithmes combinant leurs différentes stratégies heuristiques et méta-heuristiques(voir Figure3.7).

Heuristique : En général, une heuristique est un ensemble de règles qui visent à trouver une solution à un problème particulier [1]. De telles règles sont spécifiques au problème et sont conçues pour qu'une solution approximative soit trouvée dans un délai acceptable. Pour le scénario de planification discuté ici, une approche heuristique utilise les connaissances sur les caractéristiques du cloud ainsi que l'application de workflow afin de trouver une planification qui répond aux exigences de QoS de l'utilisateur. Le principal avantage des heuristiques d'ordonnancement est leur efficacité en termes de performances ; ils ont tendance à trouver des solutions satisfaisantes dans un

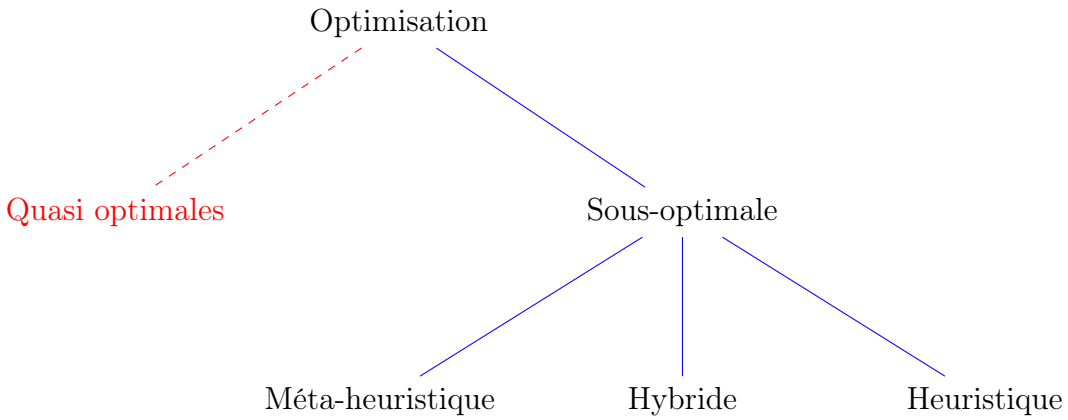


FIGURE 3.7 – Un schéma expliquant les types d'optimisation.

laps de temps adéquat. Elles sont également plus faciles à mettre en œuvre et plus prévisibles que les méthodes basées sur des méta-heuristiques.

Méta-heuristiques : Alors que les heuristiques sont conçues pour fonctionner au mieux sur un problème spécifique, les métaheuristiques sont des algorithmes à usage général conçus pour résoudre des problèmes d'optimisation [1]. Ce sont des stratégies de niveau supérieur qui appliquent des heuristiques spécifiques à un problème afin de trouver une solution quasi optimale à un problème. Comparées aux heuristiques, les approches méta-heuristiques sont généralement plus gourmandes en calculs et prennent plus de temps à s'exécuter ; Cependant, ils ont également tendance à trouver de meilleures planification lorsqu'ils explorent différentes solutions à l'aide d'une recherche guidée.

Hybride : Les algorithmes utilisant une approche hybride peuvent utiliser des méta-heuristiques pour optimiser la planification d'un ensemble de tâches de workflow tout en les combinant avec des heuristiques. De cette façon, les algorithmes peuvent être en mesure de trouver de meilleures solutions d'optimisation qu'une simple méta-heuristiques tout en réduisant le temps de calcul en considérant un espace de problème plus petit.

3.4 Ordonnancement des taches de workflow : Etat de l'art

L'ordonnancement des Workflows attire toujours l'attention des chercheurs industriels et universitaires. Les premières études se sont principalement concentrées sur le problème d'optimisation mono-objectif, minimisant le temps d'exécution du Workflow en utilisant des heuristiques et des métaheuristiques [38].

L'avènement du cloud computing a rendu possible l'exécution de workflows en louant des machines virtuelles à un certain coût, qui apparaît comme un objectif conflictuel avec le temps d'exécution. Le problème d'ordonnancement est depuis formulé comme un problème d'optimisation multiobjectif (MOP). Les deux principales approches de résolution suivantes sont envisagées :

La première approche combine les objectifs en un seul objectif par l'utilisation d'une fonction d'agrégation attribuant un poids pour chaque objectif [39-41]. Outre sa simplicité de mise en œuvre, trouver les bonnes valeurs des poids de la fonction d'agrégation reste un inconvénient important, car il n'est pas évident de trouver la valeur adéquate reflétant exactement la préférence du décideur.

Alors que la première approche généralement fournit une solution unique, la seconde approche est basée sur le front de Pareto qui offre un ensemble de solutions de compromis aux utilisateurs, afin qu'ils puissent sélectionner la solution adéquate à leurs besoins. À cette fin, plusieurs méthodes optimisant le cout et le makespan ont été développées :

— NSGAII a été adapté par Jia Yu et al [42] pour l'ordonnancement des workflows en utilisant deux critères de sélection efficaces, le tri par non-dominance de Pareto et la distance de crowding, pour guider la recherche vers le front optimal. Le tri de non-dominance est utilisé pour diviser les individus en plusieurs fronts non dominés classés en fonction de leurs relations de dominance. La distance de crowding est utilisée pour estimer la densité des individus dans une population pour effectuer la sélection d'individus sur un front.

— SPEA2 est un algorithme génétique proposé par Zitzler et al [43] qui commence avec une population de solutions candidates et les recombine itérativement dans le but d'en produire de meilleures. SPEA2*[42] est une version personnalisée hybride de SPEA2 ou elle est initialisé avec une solution quasi-optimale en termes de makespan (calculé en utilisant HEFT) et de coût financier (en utilisant les ressources les moins chères).

— MODE (Multiobjective Differential Evolution), les auteurs de [44] ont proposé une approche basée sur l'évolution différentielle multi objectif pour planifier un workflow. L'idée de base de MODE est similaire à celle des algorithmes génétiques ; c'est-à-dire que l'opération de mutation génére de nouveaux individus, puis les opérations de croisement et de sélection sont effectuées par une évolution itérative constante pour trouver la solution optimale globale. L'algorithme MODE met en œuvre l'opération de mutation en utilisant la stratégie de différence, qui diffère de l'AG en améliorant la capacité de l'algorithme à effectuer des recherches sur la base des caractéristiques de la population.

— NSPSO, une optimisation par essaims de particules utilisant le tri non dominance est utilisée dans [45], pour planifier les workflow. PSO est une technique d'optimisation par recherche globale auto-adaptative introduite par Kennedy et Eberhart [46] ,elle repose sur le comportement social des particules. À chaque génération, chaque particule ajuste sa trajectoire en fonction de sa meilleure position (meilleure position locale) et de la position de la meilleure particule (meilleure position globale) de toute la population.

— PSO est une version de PSO utilisant la dominance floue est implémentée dans [47] pour l'ordonnancement dans les environnements de grille hétérogènes traditionnels. Par la suite, ils ont été adaptés aux environnements Cloud AMAZON EC2 avec ses modèles de tarification particuliers dans la plateforme IaaS.

— EMS-C est un algorithme basé sur l'optimisation évolutive multiobjective (EMO) est proposé par Zhu et *al* [2] exclusivement adapté pour une plateforme IaaS. De nouveaux schémas pour l'encodage spécifique au problème et l'initialisation de la population, l'évaluation du fitness

et les opérateurs génétiques sont proposés dans cet algorithme. Ils ont utilisé deux opérateurs pour explorer la recherche de l'espace entier et pour exploiter simultanément des régions qui ont été explorées précédemment.

— MOAC [3] est un algorithme hybride de colonies de fourmis multiobjectif basé sur un cadre co-évolutionnaire de populations multiples pour des objectifs multiples est proposé. Ils adoptent deux colonies pour traiter deux objectifs. De plus, l'approche proposée incorpore trois nouvelles conceptions pour traiter efficacement les défis multiobjectifs : 1) une nouvelle règle de mise à jour de la phéromone basée sur un ensemble de solutions non dominées provenant d'une archive globale pour guider chaque colonie à rechercher suffisamment son objectif d'optimisation ; 2) une heuristique complémentaire pour éviter qu'une colonie ne se concentre uniquement sur son seul objectif d'optimisation, coopérant avec la règle de mise à jour de la phéromone pour équilibrer la recherche des deux objectifs ; et 3) une stratégie d'étude de l'élite pour améliorer la qualité de la solution de l'archive globale afin d'aider à se rapprocher davantage du front de Pareto global.

— MOELS est un algorithme d'ordonnancement par liste évolutionnaire multiobjectif et hybride[4]. Il intègre l'ordonnancement par liste classique dans un puissant algorithme évolutionnaire multiobjectif (MOEA). Un génome est représenté par une séquence d'ordonnancement pondéré par un poids de préférence et interprété comme une solution via une heuristique d'ordonnancement par liste spécialement conçue, et les génomes de la population évoluent grâce à des opérateurs génétiques adaptés.

— Durillo *et al* [48] ont développé MOHEFT qui est une extension multi objectifs et générique de l'heuristique bien connue HEFT [49]. MOHEFT semble être une méthode d'énumération, car elle fait le mappage de toutes les ressources existantes sur toutes les tâches. Cependant, elle utilise la distance de Cronwding et la non-dominance de Pareto pour l'élagage dans chaque étape de la construction des solutions et évite ainsi l'explosion combinatoire en ne conservant que k planifica-

tion pendant la construction des solutions pour chaque tâche. Ainsi, à la dernière étape (tâche), on obtient exactement k solutions.

MOHEFT est devenu la méthode de comparaison standard pour divers travaux scientifiques [2-16] ou ils ont mis l'accent sur le paramètre d'entrée k qui permet d'avoir k solutions en sortie. Les études menées se sont accordé que MOHEFT est une excellente méthode. En plus, la qualité des résultats retournés s'améliore au fur et à mesure que la valeur du paramètre k augmente, conduisant à une complexité temporelle plus grande [3, 8].

Workflow	EMS-C	MOAC	MOELS
Montage 25	+**0.09%**	-0.39%	-
Montage 50	+**0.03%**	-0.27%	-0.60%
Montage 100	+**0.46%**	-0.12%	-0.60%
Montage 1000	**N/A**	**N/A**	-
Epigenomics 24	-7.22%	-1.41%	-
Epigenomics 46	-1.20%	-18.64%	-
Epigenomics 100	+**1.37%**	-31.26%	-5.00%
Epigenomics 997	**N/A**	**N/A**	-
CyberShake 30	-1.95%	-0.64%	-
CyberShake 50	-3.14%	-0.43%	-
CyberShake 100	-0.76%	-1.07%	-
CyberShake 1000	**N/A**	**N/A**	-
sipht 30	+**0.05%**	-0.092%	-
sipht 60	+**0.08%**	-0.78%	-
sipht 100	+**0.16%**	+**0.026%**	-2.10%
sipht 1000	**N/A**	**N/A**	-
Inspiral 30	-5.86%	-1.55%	-
Inspiral 50	-1.30%	-3.90%	-6.00%
Inspiral 100	-6.66%	-1.03%	-3.70%
Inspiral 1000	**N/A**	**N/A**	-

TABLE 3.2 – Les performances HV [2-4] de MOHEFT par rapport aux algorithmes pairs sur les workflows du monde réel.

Le tableau 3.2 montre les performances de MOHEFT par rapport à des métaheuristiques efficaces et récentes, spécialement conçues pour le cloud computing, EMS-C [2], MOAC [3] et MOELS [4]. La métrique de performance est basée sur l'indicateur d'hyper-volume et les valeurs de chaque colonne du tableau sont extraites de l'article original décrivant la métaheuristique

en question. Les performances de MOHEFT sont très proches de celles de EMS-C, MOAC et MOELS. Bien que les grands benchmarks n'aient pas été testés, ils ont conclu que MOHEFT ne peut pas être achevé dans un temps acceptable, et la note "N/A" non acceptable est mis dans les cases de 1000 tâches [2, 3].

Dans MOELS [4], une marque **" -"** a été mise pour noter que le benchmark n'a pas été utilisé, en particulier les workflows de 1000 taches .

Les métaheuristiques [2-4] sont spécialement conçues pour le Cloud avec des objectifs particuliers or que MOHEFT reste générique,il peut être utilisé sur une plateforme Grid ou cloud avec n'importe quels objectifs ,ce qui lui donne une bonne réputation [50].

3.5 Conclusion

L'ordonnancement des workflows dans le cloud a constitué un open challenge. Les procédés d'ordonnancement les plus récent et efficace ont comparé dans ce chapitre, un bon mappage des taches de workflow sur les ressources du cloud est crucial. Une comparaison des différents travaux a mis en exergue la complexité ainsi que de la qualité des résultats de MOHEFT. En raison de la sous-exploitation de MOHEFT sur les larges workflows, dans le chapitre suivant nous étudierons la complexité temporelle en détail et nous évaluerons une version parallèle de l'algorithme MOHEFT.

Chapitre 4

MOHEFT

4.1 Introduction

La flexibilité offerte par MOHEFT en tant qu'algorithme multiobjectif générique est très attrayante. En outre, le front Pareto est un outil efficace d'aide à la décision, car il permet aux utilisateurs de sélectionner la solution la plus appropriée en fonction de leurs besoins. Le principal inconvénient de MOHEFT est son temps d'exécution qui le place loin derrière les autres méthodes lorsqu'il est utilisé pour les larges workflows. Dans ce chapitre, nous nous intéressons tout particulièrement à l'accélération de MOHEFT en le rendant plus rapide avec l'exploitation dela parallélisation et de tous les cœurs du CPU. Avant de présenter notre approche, nous revisitons MOHEFT et son fonctionnement afin d'évaluer sa complexité et essayer de comprendre où réside le manque de performance en termes de temps d'exécution.

4.2 Etude de MOHEFT

4.2.1 Modélisation de workflow

Les graphes acycliques directs (DAG) sont couramment utilisés dans la modélisation des applications de workflow. Un DAG peut être défini comme un graphe G = (N,E), où N = $\{T_0, T_1, ..., T_n\}$ est un ensemble de n tâches dans un workflow, et E =$\{(T_i, T_j, Data_{ij}) \mid (T_i, T_j) \in N \times N\}$ est un en-

semble d'arêtes évaluées (T_i, T_j) représentant les dépendances de priorité entre les tâches T_i et T_j. La valeur $Data_{ij}$ représente la taille des données qui doivent être transférées de la tâche T_i vers T_j. Soit $pred(T_i)$ désignant l'ensemble des tâches à accomplir avant de commencer T_i, comme : $pred(T_i) = T_j|(T_j, T_i) \in E$ et nous supposons que pour chaque tâche T_i, la charge de travail de calcul est connue et le nombre d'instructions machine à exécuter est donné.

Modèle de ressource

La plateforme IaaS (Infrastructure as a Service) fournit ses ressources via des machines virtuelles. Il est courant pour une plate-forme IaaS de fournir un large éventail de types d'instances avec différentes combinaisons de CPU et de bande passante réseau. Comme défini dans [48, 51], la plate-forme matérielle se compose d'un ensemble de ressources hétérogènes m R=$\bigcup_{i=1}^{m} R_i$. La vitesse de chaque ressource est connue et mesurée par le nombre d'instructions par seconde. De plus, chaque ressource suit un modèle de prix composé des quatre composantes suivantes :

- PE_{R_i} est le prix par seconde de l'utilisation d'une ressource donnée.
- PS_{R_i} est le prix par seconde du stockage des données sur cette ressource en Mo.
- PI_{R_i} est le prix de la réception de données sur la base de la bande passante réseau entrante en Mo.
- PO_{R_i} est le prix de l'envoi de données sur la base de la bande passante réseau sortante en Mo.

Énoncé du problème

Notre problématique est de planifier l'exécution des tâches de workflow sur les ressources cloud, afin de minimiser le makespan (WET) et le coût (WEC). Désignons sched (T_i) la ressource sur laquelle la tâche 'T_i' est planifiée pour être exécutée. Nous présentons ci-dessous comment les deux objectifs d'intérêt sont calculés.

Temps d'exécution du workflow (WET) Pour calculer le WET, le temps d'exécution d'une activité T_i sur une ressource R_j = sched (T_i) noté $t_{(T_i,R_j)}$

est définie comme la somme du temps nécessaire pour terminer T_i sur R_j et du temps nécessaire pour transférer les données d'entrée les plus importantes à partir de n'importe quel $T_p \in \text{pred}(T_p)$

$$t_{(T_i,R_j)} = \max_{T_p \in \text{pred}(T_i)} \left\{ \frac{Data_{pi}}{b_{pj}} \right\} + \frac{\text{workload}(T_i)}{s_j} \tag{4.1}$$

où Data_{pi} est la quantité de données à transférer entre T_p et T_i, b_{pj} est la vitesse de liaison entre la ressource où la tâche T_p a été effectuée et R_j resource, workload(T_i) est la longueur de la tâche T_i en termes d'instructions machine, et s_j la vitesse de la ressource R_j exprimée en nombre d'instructions informatiques par seconde. Ensuite, nous pouvons calculer le temps d'achèvement CT_{T_i} de la tâche T_i en tenant compte de son temps d'exécution et de celui de ses prédécesseurs comme suit :

$$CT_{T_i} = \begin{cases} t_{(T_i,sched(T_l))}, & \text{pred}(T_i) = \emptyset \\ \max_{T_p \in \text{pred}(T_i)} \left\{ CT_{T_p} + t_{(T_i,\text{sched}(T_i))} \right\} & \\ \text{pred}(T_i) \neq \emptyset & \end{cases} \tag{4.2}$$

WET est finalement décrit comme le temps d'achèvement maximal de l'ensemble des activités de workflow :

$$WET_W = \max_{i \in [1,n]} \left\{ CT_{(T_i,sched(T_i))} \right\} \tag{4.3}$$

Coût d'exécution du workflow (WEC) Le coût d'exécution du workflow WEC dépend du coût de calcul, du transfert de données et du coût de stockage, désignés respectivement par cost $C^{(comp)}$, $C^{(\text{data})}$. Nous définissons $C^{(\text{data})}_{(T_i,R_j)}$ comme les coûts de transfert $\text{In}(T_i)$ et $\text{Out}(T_i)$ et storage $\text{Data}(T_i)$ résultant de l'exécution de l'activité T_i sur la ressource R_j :

$$C^{(\text{data})}_{(T_i,R_j)} = \text{Data}(T_i) \cdot t_{(T_i,R_j)} \cdot PS_{R_i} + \text{In}(T_i) \cdot PI_{R_i} + \text{Out}(T_i) \cdot PO_{R_i} \tag{4.4}$$

Lors de la définition de cost $C_{R_j}^{(\text{comp})}$ de l'utilisation d'une ressource R_j, nous supposons que nous enregistrons deux horodatages pour chaque tâche T_i effectuée sur R_j : $t_{T_i}^{(\text{start })}$ au démarrage de l'activité et $t_{T_i}^{(end)}$ une fois l'activité terminée. La valeur $t_{T_i}^{(end)}$ est calculable comme $t_{T_i}^{(\text{start})} + t_{(T_i,R_j)} + \max_{T_i \in \text{pred}(T_p)} \left\{ \frac{\text{Data}_{ip}}{b_{jp}} \right\}$.

Nous considérons également les heures auxquelles les données d'entrée $\text{In}\,(T_i)$ sont transférées et $\text{Out}\,(T_i)$ les données de sortie sont incluses dans l'intervalle de temps $t_{T_i}^{(\text{start})}$ et $t_{T_i}^{(end)}$.

En d'autres termes, ces horodatages représentent la période pendant laquelle la ressource R_j sera active en conséquence directe de l'exécution de la T_i. Maintenant, trouvons la collection de toutes les activités p planifiées pour la ressource R_j notée $\{J_1, \ldots, J_p\}$, où $p < n$ et $\text{sched}\,(J_i) = R_j, i \in [1, p]$. Trier en fonction de leurs heures de début tamp : $t_{j_1}^{(\text{start})} < \cdots < t_{j_p}^{(\text{start})}$· Sur la base de cet ordre, ces activités sont regroupées dans $q \leq p$ différents groupes $G_k^{(j)}, 1 \leq k \leq q$ afin que toutes les activités d'un groupe soient effectuées consécutivement sans libérer la ressource. La ressource est libérée une fois que l'activité a terminé l'horodatage de début le plus important du groupe.

Nous créons le premier groupe $G_1^{(j)} = \{J_1, \ldots, J_r\}, r \leq p$ sur la base des trois règles suivantes :

1. La première activité J_1 fait partie du premier groupe : $J_1 \in G_1^{(j)}$
2. Chaque activité $J_i \in G_1^{(j)}, 2 \leq i \leq r$ se termine avant la libération de la ressource. Cela signifie que J_i commence lorsque la ressource est toujours louée en raison de l'exécution de J_{i-1} :

$$t_{J_i}^{(start)} < t_{f_1}^{(start)} + \left[\frac{t_{i-1}^{(end)} - t_{j_1}^{(start)}}{3600} \right] \cdot 3600 \tag{4.5}$$

Pour arrondir le temps converti en heures en le divisant par 3600, nous utilisons l'opérateur de plafond pour compléter les heures de calcul. Il est clair que la ressource sera louée pour autant d'heures que nécessaire pour mener à bien toutes les activités au sein de ce groupe.

3. L'activité suivante (ne faisant pas partie du groupe précédent) J_{r+1} $G_1^{(j)}, r+1 \leq p$ démarre en un instant de $t_{J_r+1}^{\text{start}}$ si la ressource est déjà libérée, c'est-à-dire que la tâche J_r a terminé son exécution, la dernière période louée d'une heure pour l'exécution de J_r a expiré, et la ressource R_j n'était pas nécessaire entre $t_{J_r}^{\text{end}}$ et $t_{J_{r+1}}^{\text{start}}$. Mathématiquement, cela peut être exprimé comme :

$$t_{J_1}^{(\text{start})} + \left[\frac{t_{J_r}^{(\text{end})} - t_{J_1}^{(\text{star})}}{3600}\right] \cdot 3600 < t_{J_{r+1}}^{(\text{start})} \tag{4.6}$$

Les groupes successifs sont construits jusqu'à ce que la dernière activité J_p soit affectée à un groupe. Le deuxième groupe $G_2^{(j)}$ doit être construit de la même manière en commençant par la tâche J_{r+1} au lieu de J_1. La même stratégie est appliquée à tous les groupes restants. Une fois tous les groupes configurés, le coût $C_{R_j}^{(comp)}$ de l'utilisation de la ressource Rj est défini comme le nombre d'heures nécessaires à l'exécution de tous les groupes multiplié par le coût par heure.

$$C_{R_j}^{(comp)} = PE_{R_j} \cdot \sum_{k=1}^{q} \left\lceil \frac{\sum_{T_i \in G_{R_j}^{(k)}} t_{\left(T_i, R_j\right)}}{3600} \right\rceil \tag{4.7}$$

Nous mesurons que le coût de l'ensemble du workflow G = (N,E) est le coût de l'exécution sur toutes les ressources m plus le coût du transfert et du stockage des données :

$$WEC_G = \sum_{j=1}^{m} C_{R_j}^{(comp)} + \sum_{(T_i, T_j, Data_{ij}) \in D} C_{(T_i, R_j)}^{(data)} \tag{4.8}$$

Outils d'optimisation multiobjectifs

Dans cette thèse, nous traitons un problème d'optimisation particulier avec deux objectifs WET et WEC. Deux fonctions sont utilisées pour mesurer la qualité des résultats et pour l'élagage dans MOHEFT, la distance de Crowding et Hypervolume :

La distance de Crowding reflète la densité de l'environnement d'un individu donné au sein d'une population. Cela peut être représenté par les individus environnants comme la somme du plus grand rectangle et de la plus grande largeur (Figure 4.1).

L'hypervolume est un moyen de mesurer la qualité d'un ensemble de solutions d'un front. Étant donné un ensemble de solutions X, l'hypervolume HV(X) mesure la zone comprise entre les points de X et un point de référence W (voir Figure 4.1), généralement choisi comme la valeur objective maximale (par exemple, le délai et le coût économique les plus élevés). Dans la figure 4.1, l'ensemble contenant les points ronds pleins est meilleur que l'ensemble contenant les solutions présentées par des formes carrées ; par conséquent, l'hypervolume de l'ensemble contenant tous les points ronds pleins est plus élevé que l'ensemble contenant les solutions présentées par des formes carrées.

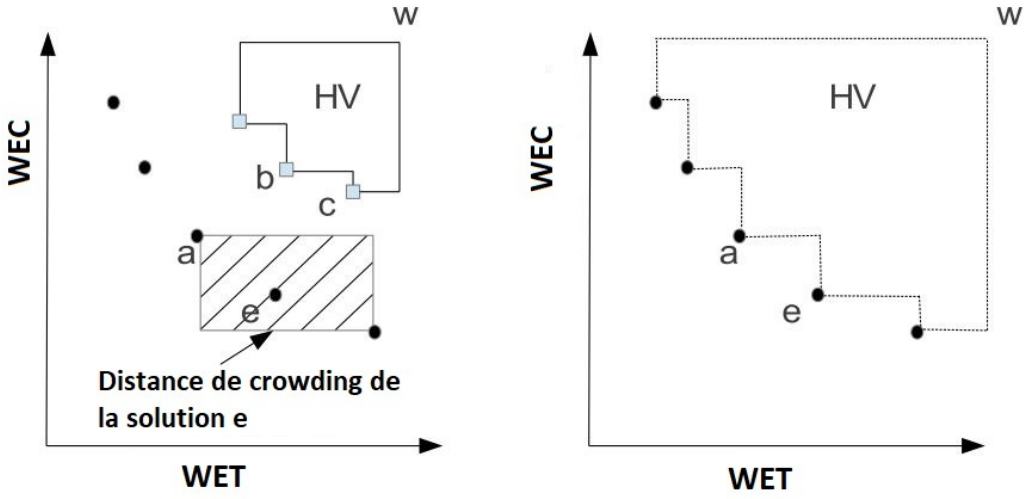


FIGURE 4.1 – Comparaison de solutions multiobjectifs.

4.2.2 Algorithme MOHEFT

Le pseudo-code de MOHEFT est présenté dans Algorithm 1,MOHEFT reçoit trois paramètres : le DAG du workflow à planifier, le pool de ressources et un nombre k qui exprime le nombre souhaité de résultats qui sera retourné en sortie de la planification du workflow.

MOHEFT classe en premier les tâches en utilisant la fonction de rang B (ligne 2), elle trie les tâches en fonction du chemin le plus long vers la tâche de sortie[49].
Dans une deuxième phase MOHEFT crée un ensemble $\mathcal{S}ol^{'}$ de $k$solutions vides (lignes 3 à 5). Ensuite, la phase de mappage de MOHEFT commence de la ligne 6 à ligne 16. L'algorithme itéré d'abord sur la liste des tâches (ligne 6) triées par ordre d'exécution selon le B-rank.

Algorithme 1 : MOHEFT

Input : W = (N,E), N = $\bigcup_{i=1}^{n} T_i$; // Workflow DAG

Input : $\mathcal{RS}$= $\bigcup_{i=1}^{m} R_i$; // $\mathcal{R}$essource $\mathcal{S}$et

Input : k; // Number of optimal Pareto solutions

Output : $\mathcal{S}ol = \bigcup_{i=1}^{k} schedW$; // Set of K scheduling solutions

1 **begin**

2 $\mathcal{B}_{Activities} \leftarrow$ UP-Rank(N) ; // Order activities according to B-rank

3 **foreach** $s \in \mathcal{S}ol$; // Create K empty Solution

4 **do**

5 s $\leftarrow \phi$

/* from this point MOHEFT is setup and the mapping should begin */

6 **foreach** $T_i \in \mathcal{B}_{Activities}$; // Iterate through all the K ranked activities

7 **do**

8 $\mathcal{S}ol^{'} \leftarrow \phi$

9 **foreach** $R_j \in \mathcal{RS}$; // Iterate over all m resources

10 **do**

11 **foreach** $s_k \in \mathcal{S}ol$; // Iterate over all K schedules in $\mathcal{S}ol$

12 **do**

13 $s^{'} \leftarrow \phi$ $s^{'} \leftarrow s_k$; // Copying schedule

14 $\mathcal{S}ol^{'} \leftarrow \mathcal{S}ol^{'} \cup \{ s^{'} \cup (T_i, R_j) \}$; // Add new mapping to intermediate schedule

/* All the T_i of the $(k \times n)$ $s^{'}$ solutions in $\mathcal{S}ol^{'}$ are mapped at the end of these two nested loops. */

15 $\mathcal{S}ol^{'} \leftarrow SortCrowdDist(\mathcal{S}ol^{'})$; // Sorts $\mathcal{S}ol^{'}$ according to crowding distance

16 $\mathcal{S}ol \leftarrow First(\mathcal{S}ol^{'}$,k); // Choose the best k solutions from sorted $\mathcal{S}ol^{'}$

17 **Return** ($\mathcal{S}ol$)

L'idée est d'étendre chaque solution dans $\mathcal{S}ol$ en mappant la tâche à planifier T_i sur toutes les ressources possibles et de les stocker dans un ensemble temporaire $\mathcal{S}ol^{'}$ qui est initialement vide (ligne 8).
Pour créer ces nouvelles solutions, nous itérons sur l'ensemble des ressources $\mathcal{R}S$ (ligne 9) et l'ensemble $\mathcal{S}ol$ (ligne 11), et nous ajoutons les nouvelles planifications intermédiaires étendues au nouvel ensemble $\mathcal{S}ol^{'}$ (ligne 15). Cette stratégie débouche sur une recherche exhaustive et une explosion combinatoire. Par conséquent, nous sélectionnons que les k meilleures solutions de l'ensemble temporaire $\mathcal{S}ol^{'}$ grâce à la fonction $SortCrowdDist$. Cette fonction sauvegarde les solutions sélectionnées dans l'ensemble$\mathcal{S}ol$ (lignes 16 à 17).
La fonction $SortCrowdDist$ utilise le tri par rang de dominance pour sélectionner. En plus pour préserver la diversité de l'ensemble des solutions de $\mathcal{S}ol$, la fonction $SortCrowdDist$ utilise la distance de crowding représenté graphiquement dans Figure 4.1 ; elle donne une mesure de la zone entourant une solution où aucune autre solution du front n'est placée. Ce critère de sélection favorise les solutions avec les distances de crowding les plus élevées, car cela signifie que le front est plus large avec diverses solutions.
Finalement après le traitement de toutes les tâches (ligne 18), l'algorithme retourne l'ensemble des k meilleures solutions.

4.2.3 Complexité de MOHEFT

La complexité temporelle de l'algorithme MOHEFT a fait l'objet de plusieurs études dans différents travaux scientifiques. Les résultats de la complexité différents d'une étude à l'autre.

En effet, les auteurs de MOHEFT ont avancé dans [48] que l'algorithme a une complexité de $O\ (n \times m \times k)$, où n est le nombre de tâches, m est le nombre de ressources et k est le nombre de solutions retourné. Une autre étude dans [52] donne la complexité de $O(n \times k \times (n-1+m))$. Les auteurs de [14] n'ont pas calculé la complexité, mais l'ont largement attribuée au calcul de la distance de Crowding. De la même manière, en impliquant la distance de Crowding, une autre analyse dans [2] donne la complexité de $O(n^3 \times k^2 \times m^2)$. Une analyse plus approfondie dans [4] a évoqué les calculs de la dominance

en plus de la distance de crowding résultant en $O(n \times k^2 \times (n+m)^2)$.

Avant d'avancer une complexité pour l'algorithme MOHEFT, nous avons mesuré le temps d'exécution de chaque instruction (ligne) dans l'algorithme 1. Les résultats obtenus montrent que plus de 99 % du temps d'exécution est pris par la fonction *SortCrowdDist* à la ligne 16. Cependant, avant de conclure que ce temps d'exécution est dû aux calculs de la distance de Crowding et au tri par rang de non dominance, nous devons examiner en profondeur cette fonction illustrée dans Algorithm 2.

Algorithme 2 : *SortCrowdDist Function*

Input : $\mathcal{S}ol = \bigcup_{i=1}^{k} schedW$; `// Set of K scheduling solutions`

Output : $\mathcal{S}ol' = \bigcup_{i=1}^{k} schedW$; `// Set of K scheduling solutions`

1 **begin**
2 **foreach** $s \in \mathcal{S}ol$ **do**
3 $\mathcal{I}NIT\_SYM\_ENV(s)$; `// Initiating the Simulation environment for` s
4 $\mathcal{F}ITNESS(s)$; `// Solution evaluation function`
5 $\mathcal{R}ankListSol \leftarrow \mathcal{C}alculateDominance(\ \mathcal{S}ol\ )$
6 $\mathcal{S}ol' \leftarrow \phi$
7 **foreach** $ListSol \in \mathcal{R}ankListSol$ **do**
 `// Iterate over all Dominance Rank`
8 $\mathcal{S}ol' \leftarrow \mathcal{S}ol' \bigcup \mathcal{C}alculateCrowdingDistance(\ ListSol\ )$
9 **Return** ($\mathcal{S}ol'$)

Les mesures du temps d'exécution de chaque ligne de la fonction $SortCrowdDist(\mathcal{S}ol')$ ont été effectuées.

Le temps pris par le calcul de la distance de crowding et le tri par rang de non dominance (Ligne 5-8) représente moins de 0,03%. Ceci est négligeable par rapport au temps nécessaire à l'exécution de la boucle de la ligne 2 à la ligne 4, dans laquelle se fait l'initialisation de la simulation d'exécution du workflow et le calcul de la fonction FITNESS qui prend la totalité du temps d'exécution dans l'algorithme MOHEFT.

Pour confirmer nos résultats, nous avons utilisé une autre approche qui consiste à simuler l'exécution de MOHEFT sur un workflow de 1000 tâches avec $k = 50$ et $m = 9$ ressources. Pour une telle simulation, il faut effectuer 1000 fois le calcul du tri par non dominance et le calcul de distance de crouwding de 450 solutions générées aléatoirement.

En plus, on calcul également 450000 appels de la fonction $\mathcal{FITNESS}$ pour un workflow de 1000 tâches avec des solutions générées aléatoirement et qui représente le nombre d'appels effectué dans MOHEFT ($n = 1000, m = 9, k = 50$).

Le temps nécessaire à l'exécution des distances de crowding et des tris par non dominance, représente $0.7e^{-6}$ % du temps pris par les 450000 runs de la fonction $\mathcal{FITNESS}$. De plus, le temps d'exécution de 450000 $\mathcal{FITNESS}$ est quatre fois plus long qu'une exécution de MOHEFT. Cet écart peut s'expliquer par le fait que la simulation et l'évaluation des solutions pour les premières itérations de la boucle de mappage dans MOHEFT sont effectuées sur un nombre réduit de tâches (uniquement les tâches déjà effectuées).

La complexité du MOHEFT dépend principalement de deux facteurs (i) le nombre d'appels à la fonction $\mathcal{FITNESS}$ ($n \times m \times k$) et (ii) la complexité de cette fonction. En règle générale, la fonction $\mathcal{FITNESS}$ a deux boucles imbriquées, la première boucle s'exécute jusqu'à ce qu'il n'y ait plus de tâche non traitée et la deuxième boucle s'exécute jusqu'à ce que toutes les tâches prêtes à être traitées soient terminées.

Cependant, nous estimons que la complexité de $\mathcal{FITNESS}$ est de $O\,(n^2)$, vu que le nombre de ressources m et le nombre de solutions k n'affectent pas le runtime de $\mathcal{FITNESS}$ fonction.

Pour valider notre estimation, nous avons mis à l'échelle le nombre de tâches et nous avons obtenu une courbe polynomiale d'ordre 2 comme illustré dans la figure 4.2.

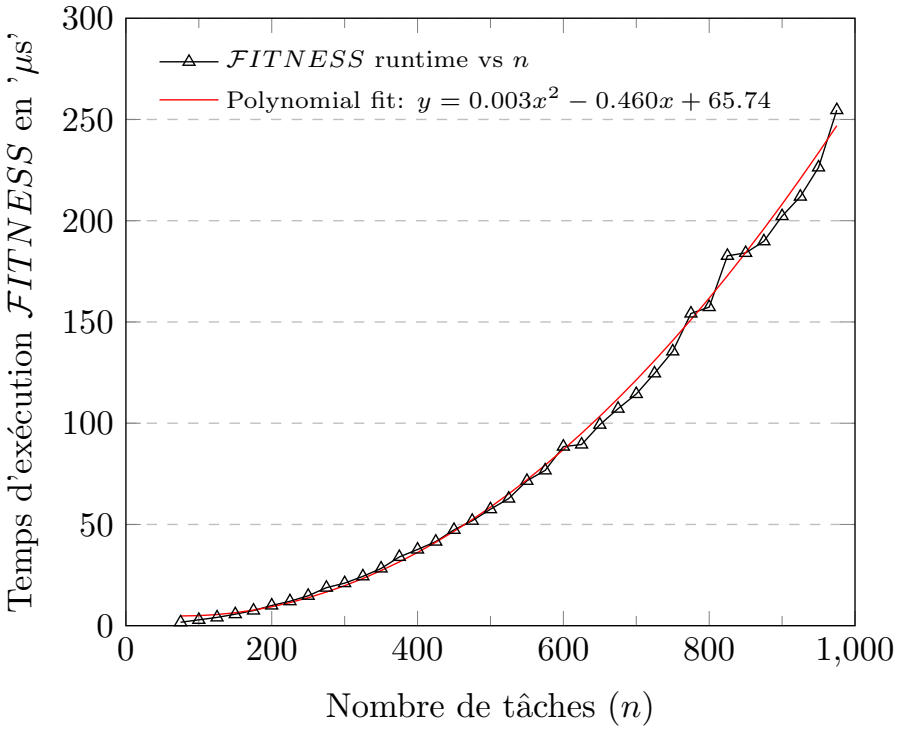


FIGURE 4.2 – Runtime de la $\mathcal{FITNESS}$ en fonction du nombre de tâches n.

MOHEFT a une complexité de O ($n \times k \times m \times$ O($\mathcal{FITNESS}$)) et vu que la complexité de la fonction $\mathcal{FITNESS}$ est O (n^2), la complexité de MOHEFT ne pourrait être que O ($n^3 \times k \times m$). Afin de valider cette hypothèse, nous avons mis à l'échelle les trois paramètres n,m et k.

Figure 4.3 et Figure 4.4 représentent le runtime MOHEFT avec la mise à l'échelle du nombre de ressources m et de la valeur de k respectivement. Ils ont tous deux à une courbe linéaire.

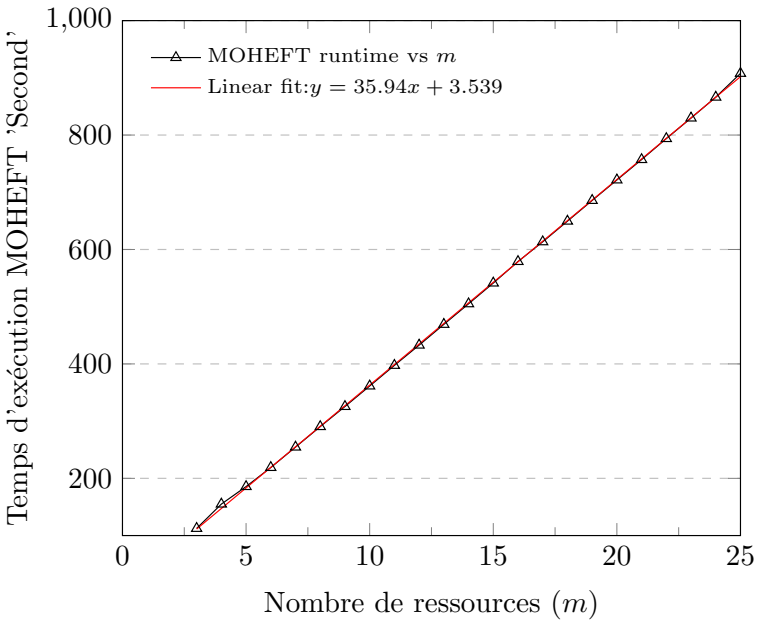


FIGURE 4.3 – Runtime du MOHEFT en fonction du nombre de ressources m.

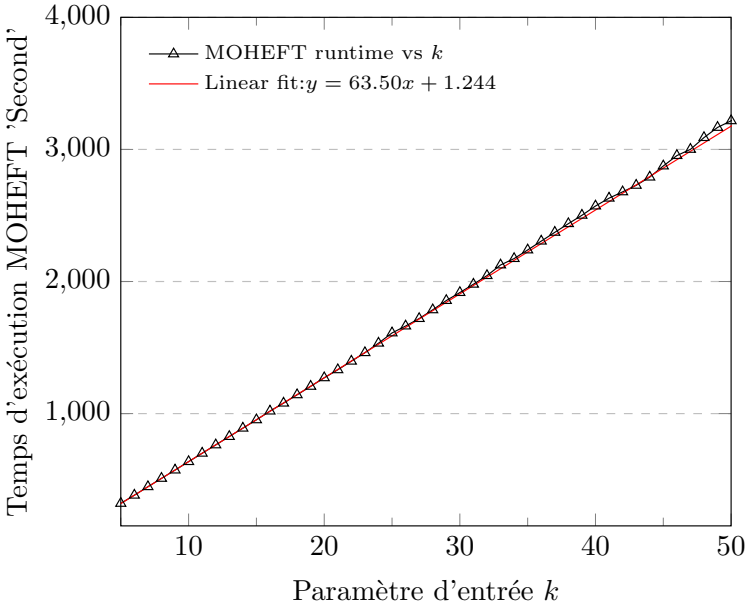


FIGURE 4.4 – Runtime du MOHEFT en fonction du nombre k .

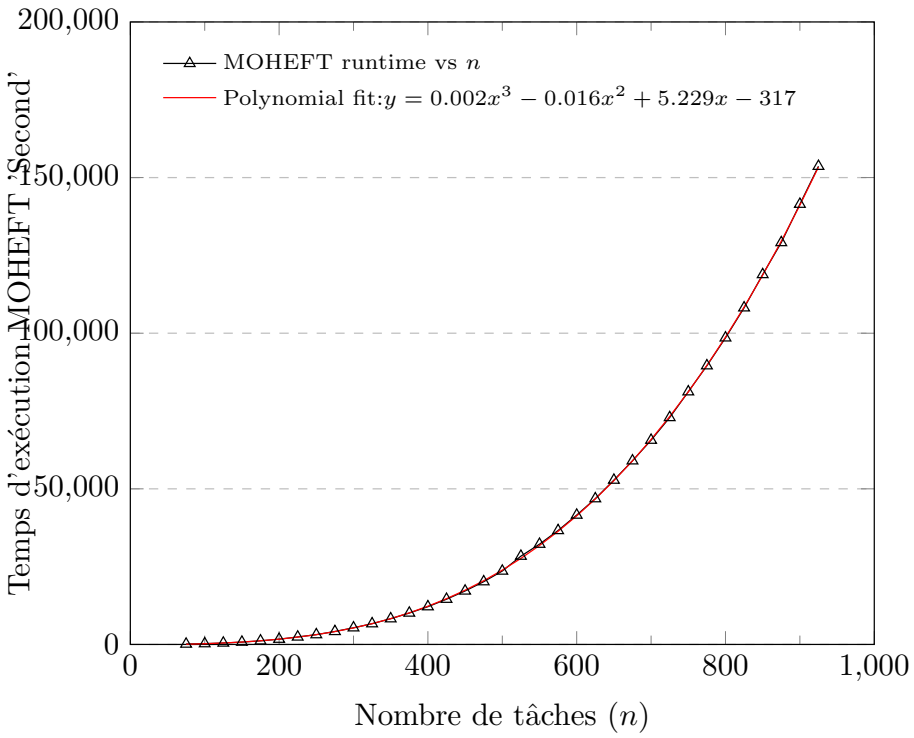


FIGURE 4.5 – Runtime du MOHEFT en fonction du nombre de tâches n.

Figure 4.5 représente le runtime MOHEFT avec une variation du nombre de tâches n. elle à une courbe polynomiale du troisième degré avec $R^2 = 1$.

4.3 Parallélisation et accélération de MOHEFT

L'optimisation du MOHEFT consiste dans la modification du code visant à améliorer sa qualité et son efficacité. Un programme peut être optimisé de manière à ce qu'il devienne plus petit, consomme moins de mémoire ou, dans notre cas, s'exécute plus rapidement. Bien que le mot « optimisation » partage la même racine que « optimal », il est rare que le processus d'optimisation produise un système vraiment optimal.

On peut classifier l'optimisation du code en deux classes :

Optimisation de bas niveau (Automatique) :

les optimisations de bas niveau sont effectuées au niveau où le code source est compilé en un ensemble d'instructions-machine, et c'est à ce stade que l'optimisation automatisée est généralement employée. Les programmeurs assembleurs estiment cependant qu'aucune machine, aussi parfaite soit-elle, ne peut faire mieux qu'un programmeur compétent[53].

Optimisations de haut niveau (Manuelle) :

les optimisations de haut niveau sont généralement effectuées par le programmeur, qui manipule les différentes fonctions, procédures, classes, etc., et garde toujours les mêmes résultats retournés par le code ; cette optimisation opère au niveau de la conception du programme. Les optimisations effectuées au niveau des blocs structurels élémentaires du code source soit boucles ou branches sont généralement désignées comme des optimisations de haut niveau.

Généralement les modifications opérées sur les structures de données et les boucles par l'optimisation dite de haut niveau visent deux aspects d'optimisation qui sont les suivants :

— Optimisation d'accès à la mémoire.

— La parallélisation du code.

Dans les sous-sections suivantes nous nous intéressent à l'optimisation de haut niveau avec ses deux aspects (voir figure 4.6).

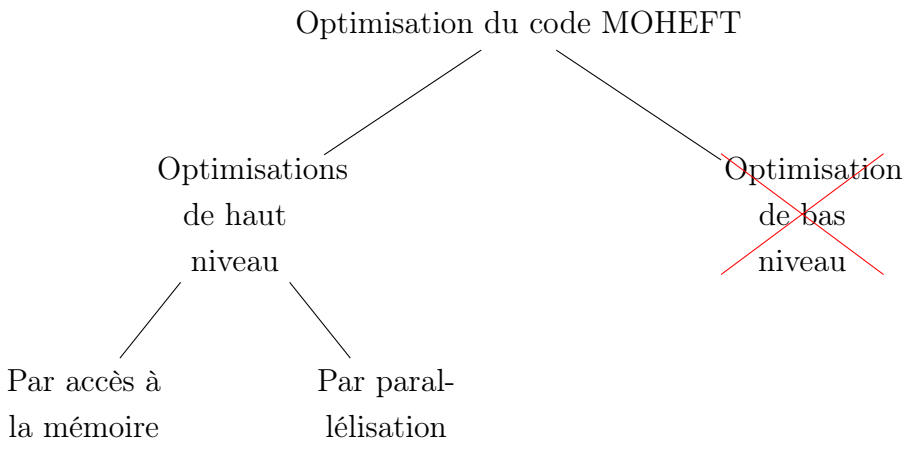


FIGURE 4.6 – Les approches standard de l'optimisation de MOHEFT.

4.3.1 Optimisation de MOHEFT par réduction d'accès à la mémoire

En architecture informatique, la hiérarchie de la mémoire sépare le stockage informatique en une hiérarchie basée sur le temps de réponse. Étant donné que le temps de réponse, la complexité et la capacité sont liés, les niveaux peuvent également être distingués par leurs performances et leurs technologies. Il existe deux principaux niveaux de mémoire volatile :

— Mémoire interne : Les registres du processeur et le cache avec ces différents niveaux.

— Mémoire principal : La mémoire vive du système, la RAM.

Il s'agit d'une structuration générale de la hiérarchie de la mémoire. D'autres structures peuvent être utilisées. Par exemple, un algorithme de pagination peut être considéré comme un niveau de mémoire virtuelle lors de la conception d'une architecture informatique.

Une hiérarchie de mémoire utilisée efficacement est d'une importance primordiale pour l'optimisation du transfert et du stockage des données. Pour exploiter une telle hiérarchie mémoire, le code doit offrir des possibilités maximales de réutilisation des données.
Les transformations de code peuvent être appliquées manuellement par les

développeurs pour exploiter les différentes mémoires efficacement. Cependant, cela impose un suivi et une lecture minimiseuse du code source, car pour appliquer une transformation d'un code spécifique, il faut comprendre si elle est permise et si elle est rentable.
Habituellement, pour savoir si une transformation est permise, il faut que les résultats du code transformé et le code original soient identiques.

Généralement l'analyse du gain d'une transformation n'est pas facile ; les développeurs inexpérimentés ne peuvent pas saisir l'impact d'une transformation du code à moins de l'appliquer manuellement et de mesurer directement son impact. Dans la pratique, le profilage du code est également utilisé pour décider si on peut appliquer une transformation d'un code. Dans certains cas, les améliorations apportées par une transformation du code dépendent de l'ensemble de données et de l'environnement d'exécution. Dans ce cas, les développeurs peuvent avoir besoin d'essayer plusieurs versions du code, chacune résultant de l'application de différentes transformations de code et des optimisations du compilateur, et de choisir au moment de l'exécution laquelle doit être sélectionné.

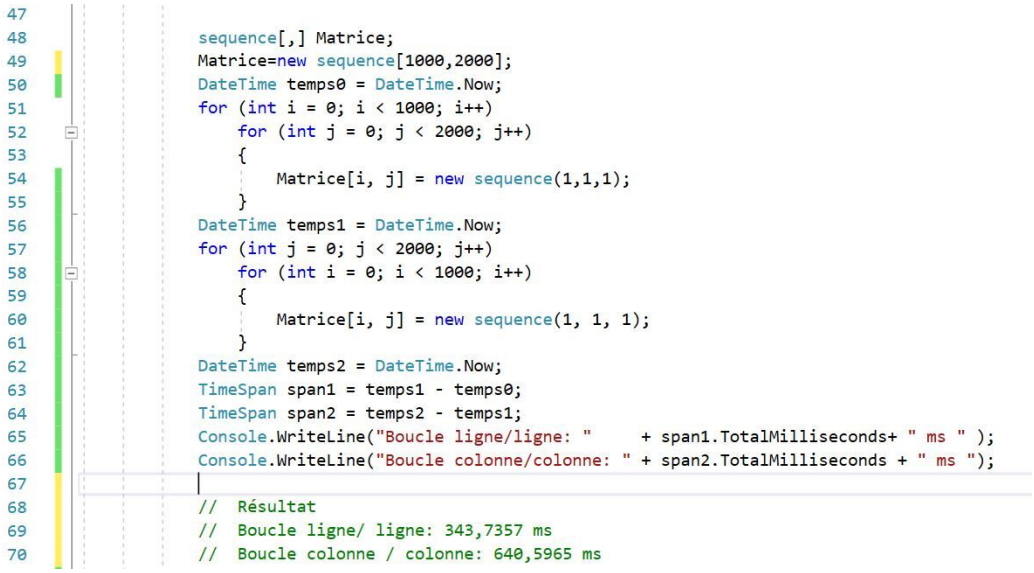
Les différentes approches de transformations du code sont présentées dans [53], nous allons citer celles qu'on a utilisées dans le code MOHEFT :

— **Les transformations basées sur la structure des données :** Dans la plupart des cas, un programme peut être optimisé en termes de performance, d'énergie ou de puissance si les structures de données sont modifiées et/ou transformées pour changer la façon dont elles sont stockées. Un exemple est l'utilisation d'un tableau au lieu d'une liste chaînée. Dans certains cas, les développeurs de logiciels ne prennent pas toujours la bonne décision et appropriée concernant la structure de données à utiliser ce qui peut entraîner des performances plus réduites. La structure de données choisie entraîne des besoins en mémoire différents,ce qui implique des coûts différents pour accéder à ses éléments et pour ajouter ou supprimer des enregistrements. Ces opérations de structure de données ont des coûts différents et selon leur fréquence d'utilisation, donc on peut choisir une structure de données la plus efficace selon le cas d'utilisation.

— **Réarrangement des codes :** Dans de nombreux cas, le réarrangement du code peut avoir un impact significatif sur les performances/énergie/puissance. En termes d'instructions, il est courant de se baser sur le réarrangement effectué par le compilateur en fonction des dépendances de données, par exemple lors de l'ordonnancement des instructions. Cependant, dans certains cas, les développeurs peuvent avoir besoin de réarranger les instructions manuellement. Le cas le plus fréquent dans le code (le cas le plus commun) peut être déplacé vers le haut en tant que premier cas, cela minimise le nombre d'opérations de test et de saut.

— **Réutilisation des données :** Il existe des transformations de code qui ont pour l'objectif de maximiser la réutilisation des données. La réutilisation des données est très importante, car elle réduit le nombre d'accès à une mémoire lente en utilisant à la place un stockage plus rapide. Les techniques de réutilisation des données éliminent les accès répétés aux mêmes données, en sauvegardant ces données dans des registres internes et/ou dans des mémoires à accès plus rapide que celles où les données sont stockées à l'origine. Les données fréquemment consultées sont enregistrées dans des registres lors de leur première utilisation, puis réutilisées lors des accès ultérieurs. Cela permet à augmenter la disponibilité des données et d'éviter les accès à la mémoire principale.

Dans l'exemple suivant, nous illustrons la transformation de code et de voir le gain qu'on peut obtenir en réarrangeant un code ou une boucle. La Figure 4.7 représente une petite partie du code source de MOHEFT ; cette partie du code initie une matrice de 1000 lignes et de 2000 colonnes d'objets de type sequence, chaque objet contient trois attributs entiers.



```
47
48          sequence[,] Matrice;
49          Matrice=new sequence[1000,2000];
50          DateTime temps0 = DateTime.Now;
51          for (int i = 0; i < 1000; i++)
52              for (int j = 0; j < 2000; j++)
53              {
54                  Matrice[i, j] = new sequence(1,1,1);
55              }
56          DateTime temps1 = DateTime.Now;
57          for (int j = 0; j < 2000; j++)
58              for (int i = 0; i < 1000; i++)
59              {
60                  Matrice[i, j] = new sequence(1, 1, 1);
61              }
62          DateTime temps2 = DateTime.Now;
63          TimeSpan span1 = temps1 - temps0;
64          TimeSpan span2 = temps2 - temps1;
65          Console.WriteLine("Boucle ligne/ligne: "     + span1.TotalMilliseconds+ " ms " );
66          Console.WriteLine("Boucle colonne/colonne: " + span2.TotalMilliseconds + " ms ");
67
68          //  Résultat
69          //  Boucle ligne/ ligne: 343,7357 ms
70          //  Boucle colonne / colonne: 640,5965 ms
```

FIGURE 4.7 – Exemple de transformation de code source.

Après avoir sauvegardé le temps de l'horloge système dans la variable nommée temps1 à la ligne 50 nous initions la matrice avec de nouvelles instances d'objet créé ; de la ligne 51 jusqu'à la ligne 55 deux boucles imbriquées fixant la première ligne de la matrice et initialisant colonne par colonne les différentes cases, une fois la première ligne terminée on boucle sur la deuxième ligne et cela jusqu'à la dernière ligne.
À la ligne 56 nous sauvegardons le temps de l'horloge système dans une deuxième variable nommée temps2.
À partir de la ligne 57 jusqu'à la ligne 61 nous réinitialisons la matrice avec les mêmes valeurs, mais au lieu de l'initialiser ligne par ligne nous l'initialisons colonne par colonne. À la ligne 58 nous sauvegardons le temps de l'horloge système pour une troisième et dernière fois.

L'initiation de la matrice colonne par colonne a pris plus de 86% de temps en plus par rapport à l'initiation de la matrice ligne par ligne. Cela est due aux défaut de page lors de l'accès à la mémoire.

En effet, comme mentionné dans la sous-section 4.2.3, le calcul des différents appels à la fonction $\mathcal{F}ITNESS$ constitue 99% du temps d'exécution de MOHEFT. Par conséquent, l'optimisation de la fonction $\mathcal{F}ITNESS$ réduira le temps d'exécution de MOHEFT.

L'optimisation du codage MOHEFT consiste à minimiser la quantité de données à transférer de et vers la mémoire centrale lors du déroulement de la fonction $\mathcal{F}ITNESS$. La réduction de l'accès à la mémoire diminue le taux défaut de cache et diminue ainsi le temps d'exécution.

Le Tableau 4.1 est une comparaison du temps d'exécution de MOHEFT optimisé par transformation de code comparé avec MOHEFT[14]. Les temps d'exécution de MOHEFT[14] sont publiés dans [14].

Workflow	Temps d'exécutions en seconde (s)	
	MOHEFT[14]	MOHEFT
CyberShake 30	9.55	0,44
CyberShake 50	43.51	1,16
CyberShake 100	555.31	5,59
CyberShake 1000	-	3179,98
Inspiral 30	11.93	0,53
Inspiral 50	50.56	1,57
Inspiral 100	622.71	8,02
Inspiral 1000	-	4803,35
Montage 25	5.60	0,41
Montage 50	46.30	1,47
Montage 100	552.32	7,02
Montage 1000	-	3799,27
sipht 30	10.52	0,50
sipht 60	74.72	2,27
sipht 100	518.55	7,47
sipht 1000	-	4195,43

TABLE 4.1 – comparaison des temps d'exécutions de MOHEFT[14] et MOHEFT optimisé

Les résultats du temps d'exécution de MOHEFT optimisé avec les workflows de 100 tâches sont jusqu'à 100 fois plus rapide que celui de MOHEFT[14]. Même si les deux expériences ont été effectuées sur des machines presque équivalentes en termes de puissance de calcul la différence est énorme. L'explication logique de cette différence est que les auteurs MOHEFT[14] ont utilisé un simulateur lent pour le calcul du cout et du makespan de chaque solution. L'optimisation de MOHEFT par transformation du code et réduction d'accès à la mémoire a donné de bons résultats pour les workflows ne dépassant pas

les 100 tâches. Cette technique de transformation de code reste insuffisante avec les larges workflows avoisinant les 1000 tâches qui dépassent une heure de temps d'exécution.

4.3.2 Optimisation de MOHEFT par parallélisation

La fréquence maximale à laquelle un processeur peut fonctionner a atteint ses limites depuis plus d'une décennie. Cette limite, environ 4 Ghz, est due physiquement aux augmentations intolérables de la consommation d'énergie par le CPU suite à l'augmentation des fréquences d'horloge de ce dernier. Les concepteurs de processeurs ont augmenté le nombre de cœurs afin d'exploiter le parallélisme, plutôt que de se concentrer sur les performances d'un processeur à cœur unique.

Une autre façon d'améliorer constamment les performances d'un processeur est donc de prendre en charge plusieurs cœurs sur un processeur et plusieurs processeurs sur une plate-forme.

Méthodologie de parallélisation :

La principale raison de concevoir un algorithme parallèle est de minimiser le temps d'exécution. Il existe essentiellement deux méthodes pour concevoir un algorithme parallèle, la première consiste à détecter et à exploiter le parallélisme inhérent à un algorithme séquentiel déjà existant, la deuxième approche vise à construire un nouvel algorithme dédié au problème donné.

La conception d'un algorithme parallèle pour un problème donné est beaucoup plus complexe que celle d'un algorithme séquentiel. Plusieurs facteurs devront être pris en compte, entre autres, la partie du programme qui peut être traitée en parallèle, la manière de distribuer les données, les dépendances entre les données, la répartition des charges entre les processeurs, les synchronisations entre les processeurs, etc.

Métrique de performance :

Considérons un algorithme parallèle P_A à implémenter sur une machine multiprocesseur M composé de p processeurs identiques. Soit T_p le temps

d'exécution de P_A sur les processeurs p et T_1 le temps d'exécution sur seulement 1 processeur. Nous définissons ensuite :

— Le cout $C_p = pT_p$

— Le Speedup $S_p = \frac{T_1}{T_p}$

— L'efficacité $E_p = \frac{S_p}{p} = \frac{T_1}{pT_p} = \frac{T_1}{C_p}$

La notion de coût d'un algorithme est destinée à prendre en compte à la fois le temps d'exécution et le nombre de processeurs utilisés. Pour une même durée d'exécution, l'ajout de processeurs en plus est toujours coûteux. Un algorithme qui nécessite moins de processeurs pour une même durée d'exécution qu'un autre est donc toujours préférable.

L'efficacité d'un algorithme parallèle est généralement inférieure à 1 (ou 100% si elle est définie par un pourcentage). Plus l'efficacité est proche de 1, plus la qualité de l'algorithme parallèle est bonne. Le facteur le plus important dans la diminution de l'efficacité est le coût de la communication.

Si l'on considère la droite d'équation $y = p$, l'algorithme parallèle est d'autant plus efficace que la courbe des variations est proche cette droite. Dans ce cas, nous parlerons d'accélération linéaire. Pour avoir de bons speedups, le volume de communication entre les threads doit être minimal. Il est également nécessaire de minimiser les temps d'inactivité des processeurs pour éviter un déséquilibre dans la répartition des charges entre les processeurs. Dans certains cas, le déséquilibre dans la répartition est inévitable, notamment en raison des priorités entre les tâches ou de l'hétérogénéité des processeurs.

Description parallélisation de MOHEFT

L'approche la plus facile pour rendre le code de MOHEFT parallèle est de détecter et exploiter le parallélisme inhérent du code MOHEFT existant (séquentielle). De plus dans la sous-section 4.2.3, la partie la plus gloutonne en calcul dans MOHEFT a été isolée et identifiée. Cette partie est $SortCrowdDist(\mathcal{S}ol^{'})$,plus de 99% du temps d'exécution. Donc pour une parallélisation efficace, de MOHEFT, la fonction $SortCrowdDist(\mathcal{S}ol^{'})$ doit être parallélisé .

Une étude minutieuse de la fonction $SortCrowdDist(\mathcal{S}ol^{'})$ dans la sous-section 4.2.3 a permis d'isoler la boucle gloutonne, de la ligne 2 à la ligne 4, dans l'algorithme 2 (plus de 99% du temps d'exécution). Dans cette boucle se fait l'initialisation de la simulation d'exécution du workflow et le calcul des appels de fonction FITNESS.

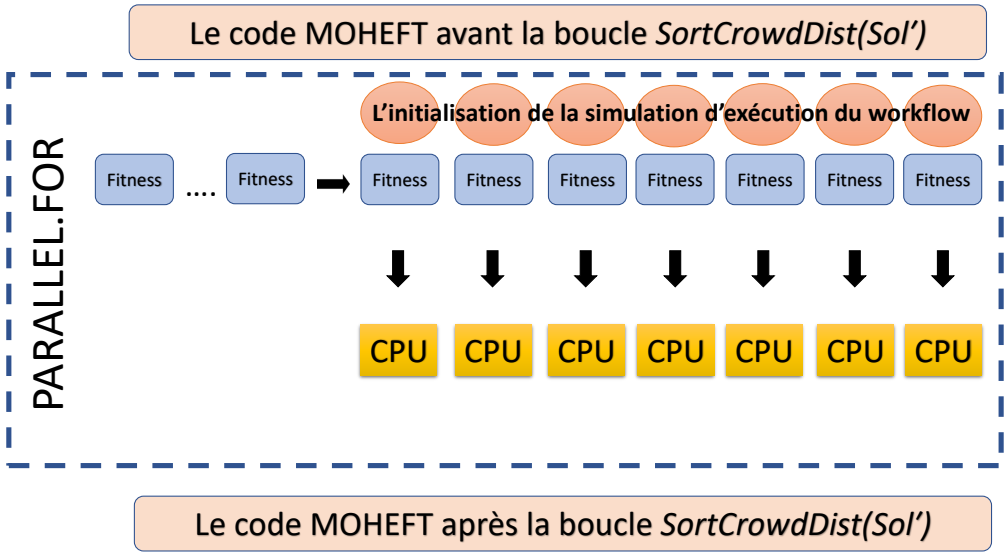


FIGURE 4.8 – La parallélisation de MOHEFT

La parallélisation de $SortCrowdDist(\mathcal{S}ol^{'})$ consiste à paralléliser uniquement la boucle des appels de fonction FITNESS ; dans la figure 4.8 le schéma expliquant l'approche de parallélisation de MOHEFT.

Nous utilisons, pour effectuer la parallélisation de MOHEFT, une architecture Multi Processeurs Symétrique qui est composée de plusieurs processeurs identiques partageant la même mémoire. Le temps d'accès à la mémoire est identique entre les processeurs.

L'utilisation du GPU pour le type du code MOHEFT parait inenvisageable selon Brodtkorb et al ont [54], la Figure 4.9 illustre le principe de la divergence dans le cas de l'utilisation d'un fragment de code If-Else sur un GPU.

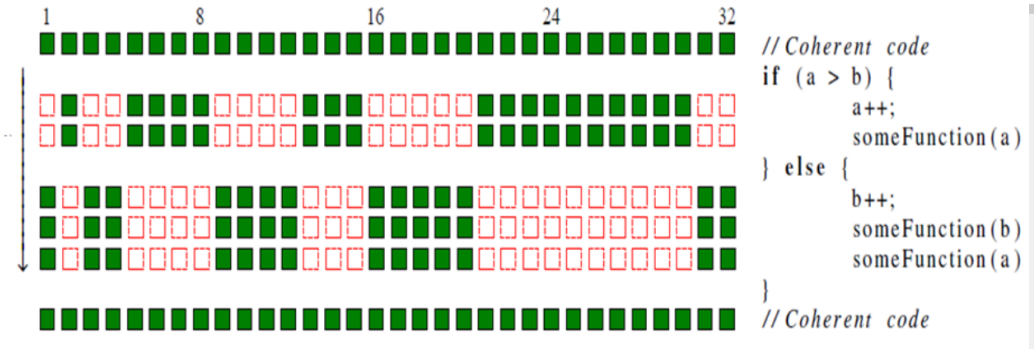


FIGURE 4.9 – Divergence d'exécution [54].

Nous testons MOHEFT parallèle sur des benchmarks les plus communément utilisés dans la littérature. Les benchmarks sont des workflows scientifiques du monde réel, publiés par le projet Pegasus et initiés par l'Université de Californie du Sud. Les caractéristiques des workflows (nombre de nœuds et nombre d'arêtes) sont résumées dans le Tableau 4.2.

Workflow	Nombre de nœuds	Nombre d'arêtes
Montage 25	25	95
Montage 50	50	206
Montage 100	100	433
Montage 1000	1000	4485
Epigenomics 24	24	75
Epigenomics 46	46	148
Epigenomics 100	100	322
Epigenomics 997	997	3228
CyberShake 30	30	112
CyberShake 50	50	188
CyberShake 100	100	380
CyberShake 1000	1000	3988
sipht 30	30	91
sipht 60	60	198
sipht 100	100	335
sipht 1000	1000	3528
Inspiral 30	30	95
Inspiral 50	50	160
Inspiral 100	100	319
Inspiral 1000	1000	3246

TABLE 4.2 – Caractéristiques des Workflows

Le nombre de solutions retourné par MOHEFT est 50 solutions (k =50) ; le paramètre restant est le pool de ressource, neuf instances de ressources sont utilisées(m=9), les mêmes ressources que ceux utilisés pour la comparaison dans [4, 14].
Les caractéristiques de l'ordinateur utilisé pour l'exécution de MOHEFT sont un CPU Intel(R) Core(TM) i7-3820 CPU @ 3.60GHz , 16 Go de RAM (2x 8Go) DDR3 1333 MHz CL9 avec un système d'exploitation Windows 10.

Algorithme 3 : Parallel *SortCrowdDist Function*

Input : $\mathcal{S}ol = \bigcup_{i=1}^{k} schedW$; `// Set of K scheduling solutions`

Output : $\mathcal{S}ol' = \bigcup_{i=1}^{k} schedW$; `// Set of K scheduling solutions`

1 **begin**
2 **foreach** $s \in \mathcal{S}ol$ ***in*** *Parallel* **do**
3 $\mathcal{I}NIT\_SYM\_ENV(s)$; `// Initiating the Simulation environment for` s
4 $\mathcal{F}ITNESS(s)$; `// Solution evaluation function`
5 $\mathcal{R}ankListSol \leftarrow \mathcal{C}alculateDominance(\ \mathcal{S}ol\ )\ \mathcal{S}ol' \leftarrow \phi$ **foreach** $ListSol \in \mathcal{R}ankListSol$ **do**
 `// Iterate over all Dominance Rank`
6 $\mathcal{S}ol' \leftarrow \mathcal{S}ol' \bigcup \mathcal{C}alculateCrowdingDistance(\ ListSol\ )$
7 **Return** ($\mathcal{S}ol'$)

Pour la version MOHEFT parallèle aucune ligne de l'algorithme MOHEFT séquentiel n'est modifiée, juste la fonction *SortCrowdDist* (algorithme 2). Cette modification est illustrée dans l'algorithme 3 où nous modifions la ligne 2 et nous ajoutons «in Parallel» à cette dernière.

Résultats de parallélisation MOHEFT

Les résultats en termes de temps d'exécution de MOHEFT séquentiels et MOHEFT parallèle sont regroupés dans le Tableau 4.3, de plus nous avons calculé la métrique d'efficacité de parallélisation E_p qui est présenté dans la troisième colonne de ce dernier. Les résultats du tableau 4.3 sont triés en 4 catégories, chaque catégorie est représentée par la taille approximative du workflow soit 25,50,100 et 1000.

Nous observons que l'efficacité E_p de la parallélisation de MOHEFT augmente à mesure que la taille des workflows augmente. D'une moyenne E_p de 16% pour les workflows de taille 25 tâches elle augmente jusqu'à une moyenne E_p de 35.6% pour les workflows de taille 1000 tâches.

Les moyennes de l'efficacité E_p observé avec les workflows de taille ap-

proximative de 50 et 100 tâches sont respectivement 18.5 % et 23.2 %. L'efficacité E_p maximal observé sur l'exécution de MOHEFT parallèle est de 38,95% et cela sur le workflow Epigenomics 997.

Workflow	Temps d'exécution en séquentiel (MS)	Temps d'exécution en parallèle (MS)	Efficacité Ep
CyberShake 30	468,7267	421,857	13,89 %
Epigenomics 24	406,2357	312,4819	16,25 %
Inspiral 30	640,6065	484,3362	16,53 %
Montage 25	453,1043	328,1163	17,26 %
Sipht 29	562,4726	426,4731	16,49 %
CyberShake 50	1349,6827	968,7002	17,42 %
Inspiral 50	1749,9165	1156,1909	18,92 %
Epigenomics 47	1598,3954	1046,8225	19,09 %
Montage 50	1609,2896	1127,3815	17,84 %
Sipht 57	2515,5037	1609,2916	19,54 %
CyberShake 100	6345,1363	4027,3582	19,69 %
Epigenomics 100	8534,2498	4205,527	25,37 %
Inspiral 100	8582,71	4222,2455	25,41 %
Montage 100	7468,3755	4062,2549	22,98 %
Sipht 97	7902,1231	4346,4315	22,73 %
CyberShake 1000	3484871,36	1398928,81	31,14 %
Epigenomics 997	5383744,52	1727822,04	38,95 %
Inspiral 1000	5339609,89	1770204,158	37,70 %
Montage 1000	4100390,69	1474775,73	34,75 %
Sipht 968	4524812,46	1582918,89	35,73 %

TABLE 4.3 – Temps d'exécution de MOHEFT en séquentiel et sur 8 coeurs en parallèle.

Le Speedup S_p de la parallélisation de MOHEFT est calculée et illustrée dans la Figure 4.10 ;cette figure est composée de quatre graphes où chaque graphe regroupe une catégorie de workflow. Les quatre catégories sont représentées par leurs tailles approximatives des workflows soit 25,50,100 et 1000 tâches.

Même s'il n'y a aucune communication entre les différents processus parallèles (appel à la fonction fitness), les résultats montrent que l'architecture parallèle n'est pas bien exploitée (moins de 40%).

Le Speedup S_p de MOHEFT parallèle sur le workflow de la catégorie 1000

tâches est plus important que ceux des workflows de catégorie de moins de 1000 tâches ; le S_p diminue à mesure que la taille des workflows diminue. Le meilleur S_p est de 3.11× pour le workflow Epigenomics 997.

L'hypothèse qui peut expliquer ces résultats est que l'overhead de la parallélisation s'estompe à mesure que le workflow s'élargit. Ce overhead est le temps de calcul indirect nécessaire au système d'exploitation afin d'exécuter des threads parallèles.

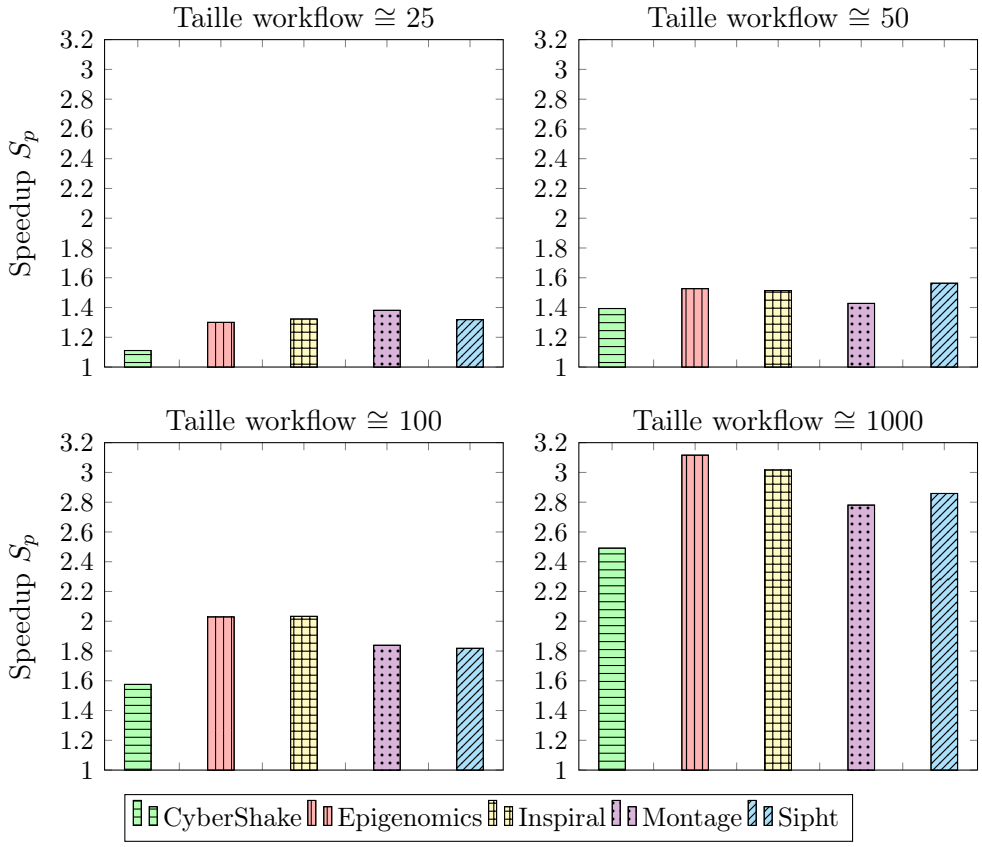


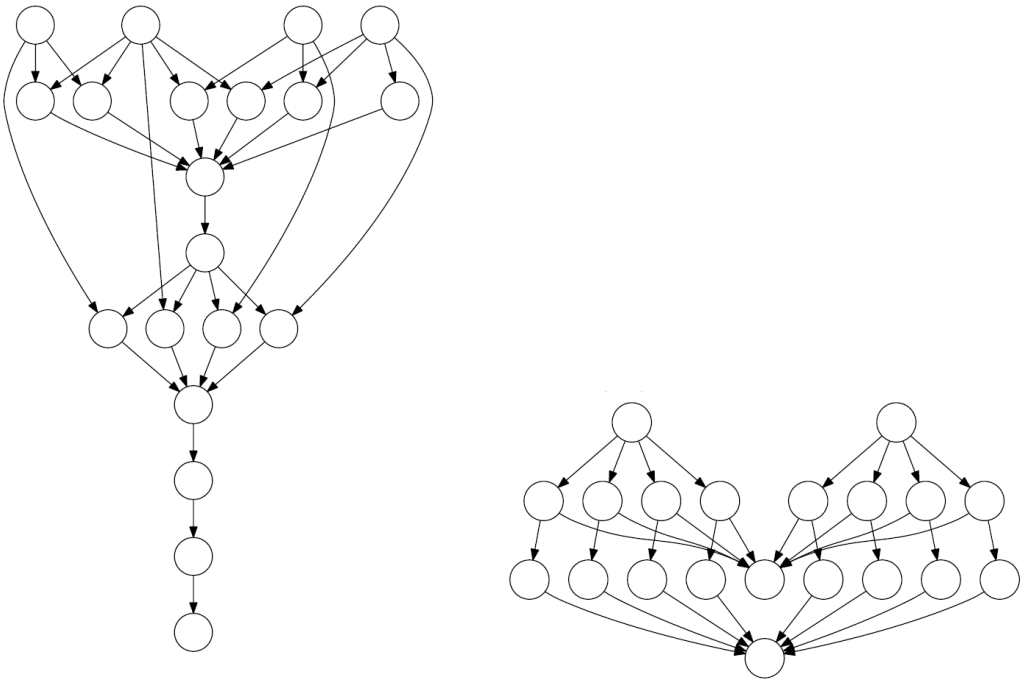
FIGURE 4.10 – Speedup S_p de MOHEFT sur 8 coeurs

L'architecture parallèle d'un ordinateur standard est relativement bien exploitée avec l'ordonnancement d'Epigenomics par MOHEFT parallèle. Le classement des workflows selon les performances Sp de la parallélisation de MOHEFT est comme suit, en premier le workflow Epigenomics puis Inspiral,Sipht,Montage et en dernier CyberShake.

Le facteur nombre d'arrêtes divisé par le nombre de sommets pour ces workflows est respectivement 3.23 , 3.24 , 3.48 , 4.44 et 3.95. Il y a une convolution entre le Sp et le facteur nombre d'arrêtes divisé par le nombre de sommets.

À l'exception Montage et CyberShake, plus le facteur arrête/sommets augmente plus le S_p diminue.

L'hypothèse qui peut expliquer ces résultats, est que le nombre d'arrêtes élevé implique plus de lecture de la mémoire centrale, et une intense sollicitation du bus de données de la RAM qui crée un goulot d'étranglement.



(a) montage (b) Cyber Shake

FIGURE 4.11 – Différence de forme entre Montage et CyberShake

Pour le cas d'exception de Montage et CyberShake la Figure 4.11 illustre la différence de forme entre ces deux workflows ; cette différence montre que les arrêtes de CyberShake sont plus régulière entre les diffèrent niveaux du workflow que ceux de Montage.

Cette régularité a pour effet lors de la simulation du workflow de diminuer l'accès à la mémoire centrale et par conséquent même si CyberShake a un facteur arrête/sommets supérieurs à celui de Montage, le S_p de MOHEFT parallèle sur ce dernier est inférieur à celui de MOHEFT parallèle sur CyberShake.

Pour réconforter les deux hypothèses avancées auparavant, celle de l'overhead de la parallélisation et du goulot d'étranglement sur le bus de données mémoire, nous procédons à deux expériences qui sont notamment les suivantes.

— **Expérience 1 :**

Ici l'objectif est double, le premier est de mesurer l'impact du désengorgement du bus de données de la mémoire centrale sur le speedup et le deuxième objectif est de mesurer l'impact du nombre de tâches d'un workflow sur l'overhead de la parallélisation.
Les étapes de cette expérience consistent dans la parallélisation pour chaque type de workflow plusieurs boucles de 450 appels à la fonction fitness avec des solutions générées aléatoirement et déjà stockées en mémoire centrale.
Le nombre de tâches des workflows évalué dans une boucle de 450 appels à la fonction fitness est le même, par contre le nombre de tâches d'une boucle à une autre varie de 100 tâches jusqu'à 1000.
Pour cette expérience tous les processus, même ceux du système d'exploitation non indispensable, ont été stoppés, et la priorité du processus père a été élevé au maximum soit « temps réel ».
Le schéma expliquant cette démarche pour une seule boucle est illustré dans la Figure 4.12.

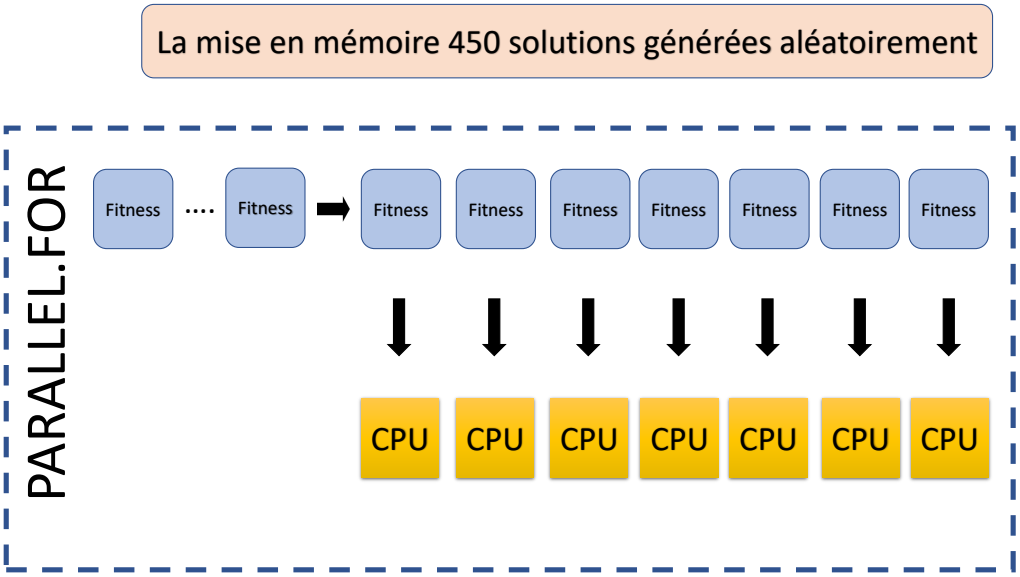
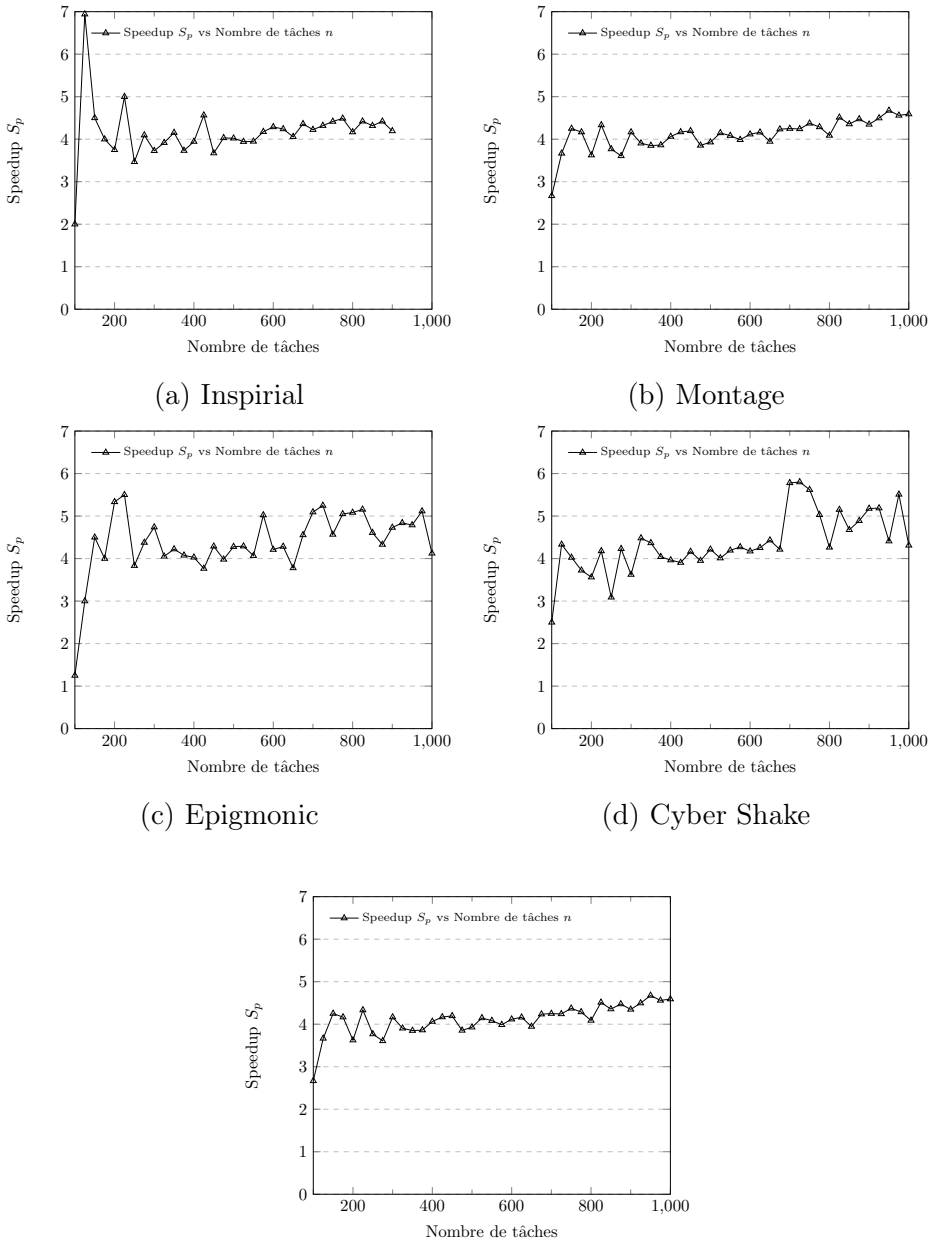


FIGURE 4.12 – La parallélisation de 450 appels de la fonction fitness avec des solutions prégénérés aléatoirement.

— **Résultats expérience 1 :**

La Figure 4.13 comprend cinq graphes, chaque graphe représente un type de workflow auquel est illustré le speedup S_p de l'évaluation en parallèle de 450 solutions générées aléatoirement et cela en fonction du nombre de tâches n. D'une façon générale les S_p ont augmenté d'une moyenne de 50% par rapport aux résultats de MOHEFT parallèle sur 8 cœurs. Ce gain peut s'expliquer par le désengorgement du bus mémoire puisque le temps d'exécution ne comprend pas la génération des solutions ni leurs chargements en mémoire centrale.
Le S_p augmente linéairement à mesure que nombre de tâches augmentent. Cette augmentation est due à l'estampillage de l'overhead de la parallélisation par apport à l'évaluation des workflows. Plus le workflow est large plus le temps de l'overhead s'estompe et devient négligeable.



(a) Inspirial

(b) Montage

(c) Epigmonic

(d) Cyber Shake

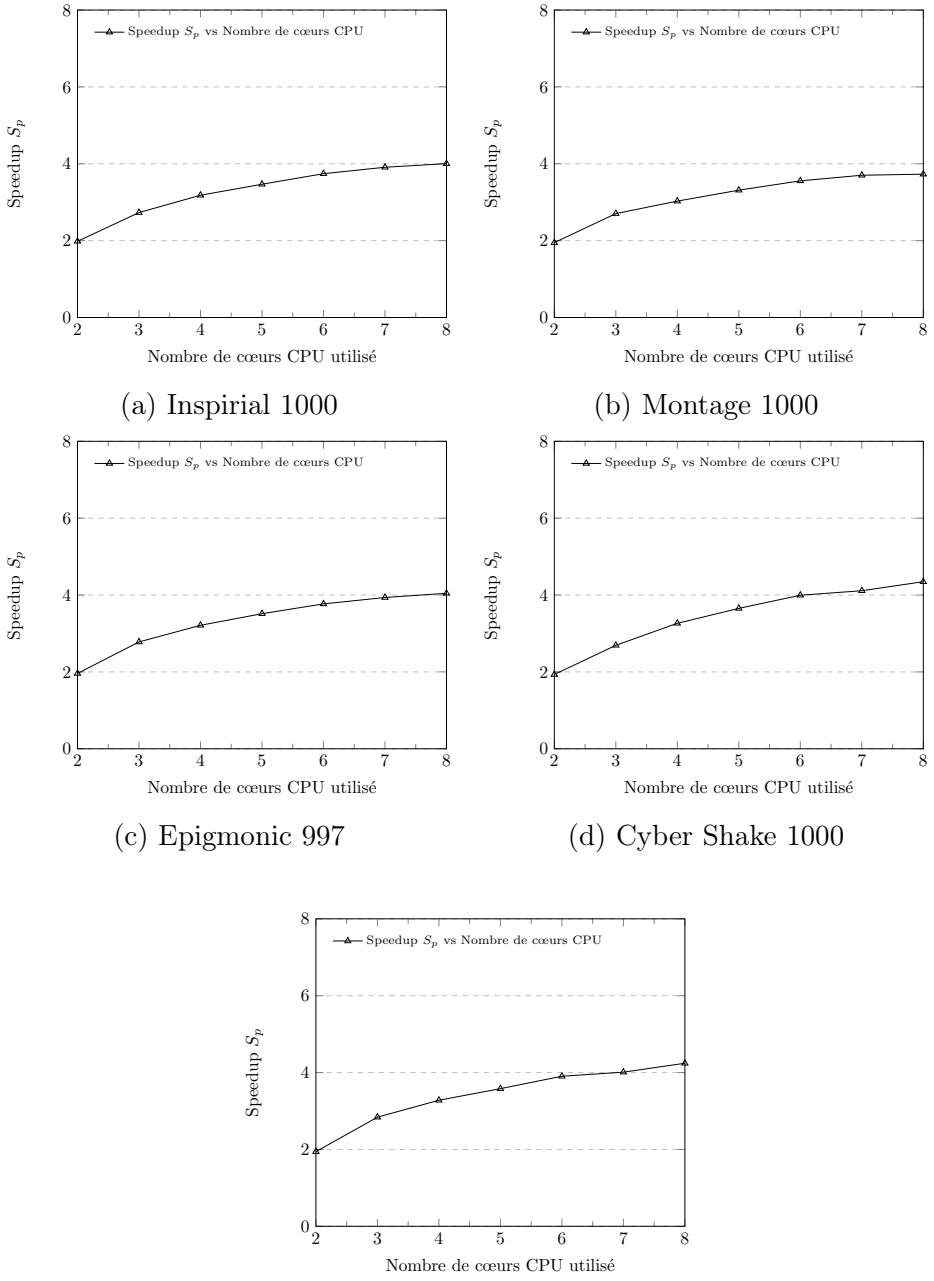
(e) Sipht

FIGURE 4.13 – Speedup de l'exécution d'un ensemble de 450 appels de la fonction fitness avec des solutions prégénérés aléatoirement.

— **Expérience 2 :**

La deuxième expérience est semblable à la précédente, mis à part le nombre des tâches du workflow qui est fixé à 1000 tâches.
Vu que l'overhead de la parallélisation devient négligeable relativement au calcul de la fonction fitness avec des workflows de 1000 tâches ; l'objectif ici est d'étudier l'efficacité de la parallélisation d'une boucle de 450 appels à la fonction fitness après le désengorgement du bus de la mémoire ; ce désengorgement se fait par limitation du nombre des cœurs utilisé pour la parallélisation. Le nombre de cœurs varie dans cette expérience, de 2 cœurs à 8 cœurs.

La Figure 4.14 englobe cinq graphes, chaque graphe représente un type de workflow auquel est illustré le speedup S_p de l'évaluation en parallèle de 450 solutions aléatoirement générées pour des workflow de 1000 tâches et cela en fonction du nombre de cœurs utilisé pour la parallélisation.
Pour tous les types de workflows l'accélération avec 2 cœurs est carrément le double puis le S_p diminue légèrement avec l'ajout d'un troisième cœur, à partir de 4 cœurs, l'accélération tend et se stabilise vers $4\times$.
La Figure 4.15 englobe cinq graphes, chaque graphe représente un type de workflow auquel est illustré l'efficacité E_p de l'évaluation en parallèle de 450 solutions aléatoirement générées pour des workflows de 1000 tâches et cela en fonction du nombre de cœurs utilisé pour la parallélisation.



(a) Inspirial 1000

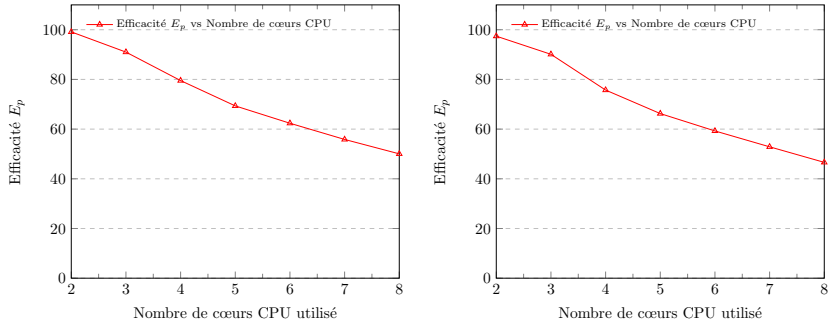
(b) Montage 1000

(c) Epigmonic 997

(d) Cyber Shake 1000

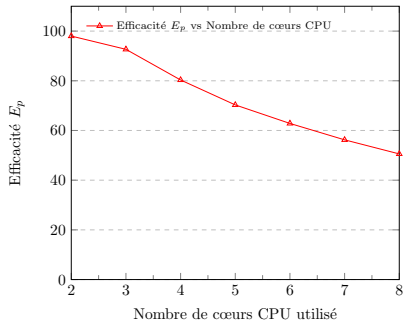
(e) Sipht 968

FIGURE 4.14 – Speedup de l'exécution d'un ensemble de 450 appels de la fonction fitness sur un nombre de cœurs CPU variable.

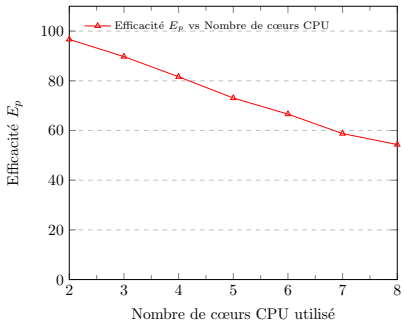


(a) Inspirial 1000

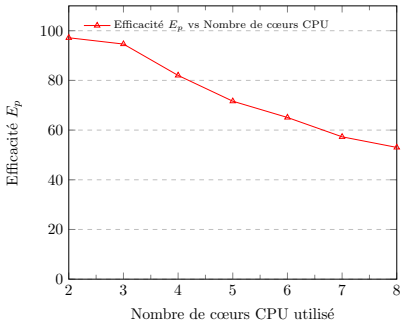
(b) Montage 1000



(c) Epigmonic 997



(d) Cyber Shake 1000



(e) Sipht 968

FIGURE 4.15 – Efficacité de parallélisation d'un ensemble de 450 appels de la fonction fitness sur un nombre de cœurs CPU variable.

L'efficacité de la parallélisation des boucles d'évaluation en parallèle de 450 solutions décline linéairement.
De 100 % avec deux cœurs, l'efficacité E_p décline jusqu'à 50% avec 8 cœurs.
Ces résultats réconfortent l'expérience précédente en deux-points :
Le premier est que l'overhead est négligeable par rapport l'évaluation de

workflow de 1000 tâches vu que l'utilisation de deux cœurs nous donne 100% d'efficacité.
Le deuxième point est la quantification de l'engorgement du bus de données de la mémoire centrale. À partir de la parallélisation avec 3 cœurs, le bus commence à être saturé, ce qui nous donne une idée sur l'énorme quantité d'information transférée entre le CPU et la RAM lors d'une évaluation avec la fonction fitness.

4.4 Conclusion

L'étude de MOHEFT et de sa complexité a permis d'identifier la partie gloutonne de ce dernier qui consiste dans la fonction fitness. Deux approches complémentaires d'optimisation ont été utilisées, la première approche use de la transformation de code pour bien exploiter l'architecture matérielle et sa mémoire centrale, cette approche a donné de bons résultats.

La deuxième approche consiste à exploiter le parallélisme inhérent dans MOHEFT pour bénéficier pleinement de la puissance des nouvelles architectures et d'utiliser tous les cœurs du CPU. L'efficacité de la parallélisation de MOHEFT a été limitée à un speedup maximal de 3.11 $\times$ à cause de l'engorgement du bus de données de la mémoire centrale.

Dans le chapitre suivant, nous nous intéressons à une nouvelle approche d'optimisation séquentielle (FAMOBACH) en utilisant le backtraking et le chekpounting afin d'éviter d'éventuelle redondance de calcul dans MOHEFT.

Chapitre 5

FAMOBACH

5.1 Introduction

Les limites architecturales matérielles des ordinateurs standard restreignent l'optimisation de MOHEFT par parallélisation. Ces limitations sont dues principalement à la vitesse du bus de données mémoires et deuxièmement au nombre de cœurs efficacement utilisé.

Dans ce chapitre, nous nous intéressons tout particulièrement à l'accélération de MOHEFT en développant une nouvelle version nommée FAMOBACH [55] (**FA**st workflow scheduling approach based **MO**HEFT using **BA**cktraking and **CH**eckpointing). Cette nouvelle version de MOHEFT est séquentielle et elle utilise le Checkpointing et le Backtracking. Ces deux mécanismes permettent d'éliminer des redondances de calculs dans la fonction fitness et accélérer le temps d'exécution de MOHEFT.

5.2 Etude de FAMOBACH

En informatique, le Checkpointing est une technique qui assure la tolérance aux pannes des systèmes informatiques. Elle consiste essentiellement à sauvegarder un instantané de l'état de l'application, de sorte que les applications puissent redémarrer à partir de ce point en cas de défaillance.

Le Backtracking est un algorithme utilisé pour les problèmes de satisfaction de contraintes, qui construit de manière incrémentielle des candidats à traiter, et abandonne un candidat « Backtrack » dès qu'il détermine que le candidat ne peut constituer une solution valide.

FAMOBACK est une version améliorée de MOHEFT ou la complexité temporelle est réduite par l'intégration des deux mécanismes le Checkpointing et le Backtracking ; c'est une méthode déterministe et ses résultats sont identiques à ceux de MOHEFT.

Dans FAMOBACK le Checkpointing permet de stocker l'état d'un calcul de la fonction fitness afin de pouvoir le récupérer ultérieurement et le poursuivre. Le stockage d'un état de calcul fitness consiste à sauvegarder les horodatages des sommets du workflow (Les $t_{T_i}^{(\text{start})}$ et les $t_{T_i}^{(end)}$) voir la sous-section 4.2.1. Le Backtracking dans FAMOBACK considère que les sommets du workflow sont des candidats, et tous les sommets avec des horodatages sauvegardé et non sujet aux changements d'horodatages sont éliminés.

Le calcul des horodatages de toutes les sommets d'un workflow constitue la majorité du temps d'exécution de la fonction $\mathcal{F}ITNESS$. La combinaison du Backtraking et du Checkpointing évite un grand nombre de recalcul des horodatages déjà établis auparavant. La réduction du nombre de calcul des horodatages implique forcement la réduction du temps d'exécution de MOHEFT.

Algorithme 4 : MOHEFT using backtracking with checkpointing

Input : W = (N,E), N = $\bigcup_{i=1}^{n} T_i$; // Workflow DAG

Input : $\mathcal{RS}$= $\bigcup_{i=1}^{m} R_i$; // $\mathcal{R}$essource $\mathcal{S}$et

Input : k; // Number of optimal Pareto solutions

Output : $\mathcal{S}ol = \bigcup_{i=1}^{k} schedW$; // Set of K scheduling solutions

```
1  begin
2      B_Activities ← UP-Rank(N) ;   // Order activities according to B-rank
3      foreach s ∈ Sol;              // Create K empty Solution
4      do
5          s ← φ
       /* from this point MOHEFT is setup and the mapping should begin */
6      foreach T_i ∈ B_Activities;   // Iterate through all the K ranked activities
7      do
8          Sol' ← φ
9          foreach R_j ∈ RS ;        // Iterate over all m resources
10         do
11             foreach s_k ∈ Sol;    // Iterate over all K schedules in Sol
12             do
13                 s' ← φ
14                 s' ← s_k ; // Copying schedule and checkpointing data
15                 Sol' ← Sol' ∪ { s' ∪ (T_i, R_j) }; // Add new mapping to intermediate schedule
           /* All the T_i of the (k × n) s' solutions in Sol' are mapped at the end of these two nested loops. */
16         Sol' ← SortCrowdDist(Sol'); // Sorts Sol' according to crowding distance
17         Sol ← First(Sol',k);   // Choose the best k solutions from sorted Sol'
18     Return ( Sol )
```

L'insertion des mécanismes Backtraking et Checkpointing est illustrée dans l'algorithme MOHEFT 4. Le seul changement dans MOHEFT, qui est mentionné en rouge, est une structure de donnée appropriée pour enregistrer les données Checkpointing collectées à partir de la fonction $\mathcal{F}ITNESS$. Le code de la fonction $\mathcal{F}ITNESS$ est totalement réécrit pour permettre de faire la collecte des données relatives à l'état d'un calcul (les horodatages) et de faire des backtrack.

Afin d'illustrer la redondance des recalculs des horodatages, prenons l'exemple de Figure 5.1 représentant un workflow de 20 tâches.

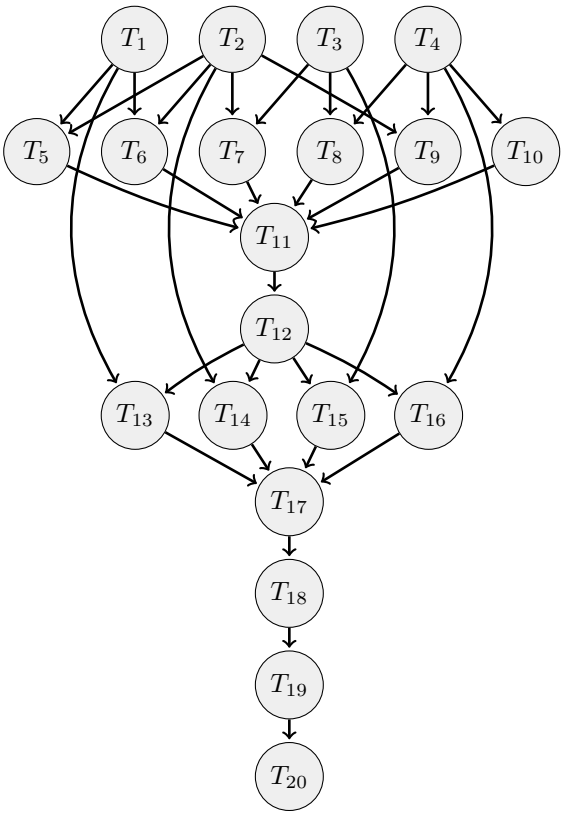


FIGURE 5.1 – Montage Workflow.

;

Le déroulement de MOHEFT sur ce workflow crée k solution, les solutions sont construites étape par étape ; à chaque étape un nouveau sommet du workflow est mappé sur une ressource. Les sommets sont traités selon l'ordre de B-rank.
Dans cet exemple, nous considérons que l'ordre des sommets défini par le B-rank est le même que le numéro de tâche ; ce qui signifie que la première tâche de la liste à planifier est T_1 puis T_2 jusqu'à T_{20}.

Toutes les solutions retenues et retournées par MOHEFT sont évaluées 20 fois chacune par la fonction $\mathcal{F}ITNESS$; c'est-à-dire a chaque nouveau mappage d'un sommet.
Lorsque la nouvelle tâche à mapper est T_{19}, avant de calculer $t_{T_{19}}^{(\text{start})}$ et $t_{T_{19}}^{(end)}$, la fonction $\mathcal{F}ITNESS$ recalcule les deux horodatages $t_{T_i}^{(\text{start})}$ et $t_{T_i}^{(end)}$ pour chaque tâche T_i avec $i = \{1, 2, ..., 18\}$.
Dans ce cas, nous n'avons pas besoin de recalculer les 18 premières tâches puisque T_{19} est leur successeur. Le $t_{T_{19}}^{(\text{start})}$ ne peut être que le $t_{T_{18}}^{(end)}$. Le calcul des deux horodatages des 18 premiers sommets est une redondance et une perte de temps de calcul.

Le Tableau 5.1 répertorie tous les calculs des horodatages d'une seule solution durant sa construction itérative. Chaque ligne représente un sommet. La deuxième colonne indique les sommets dont leurs horodatages seront calculés avant le sommet de la ligne en question. La troisième colonne représente le nombre de calculs des horodatages pour le sommet de cette ligne dans toute la construction. Le nombre total de calculs des horodatages pour une seule solution retenue par MOHEFT dans notre exemple est de 420.

Sommets	Les sommets ayant leurs horodatages recalculés	Le nombre de calculs des horodatages pour le sommet en question
T_1	–	**40**
T_2	T_1	**38**
T_3	$T_1 - T_2$	**36**
T_4	$T_1 - T_2 - T_3$	**34**
T_5	$T_1 - T_2 - T_3 - T_4$	**32**
T_6	$T_1 - T_2 - ..... - T_5$	**30**
T_7	$T_1 - T_2 - ..... - T_6$	**28**
T_8	$T_1 - T_2 - ..... - T_7$	**26**
T_9	$T_1 - T_2 - ..... - T_8$	**24**
T_{10}	$T_1 - T_2 - ..... - T_9$	**22**
T_{11}	$T_1 - T_2 - ..... - T_{10}$	**20**
T_{12}	$T_1 - T_2 - ..... - T_{11}$	**18**
T_{13}	$T_1 - T_2 - ..... - T_{12}$	**16**
T_{14}	$T_1 - T_2 - ..... - T_{13}$	**14**
T_{15}	$T_1 - T_2 - ..... - T_{14}$	**12**
T_{16}	$T_1 - T_2 - ..... - T_{15}$	**10**
T_{17}	$T_1 - T_2 - ..... - T_{16}$	**8**
T_{18}	$T_1 - T_2 - ..... - T_{17}$	**6**
T_{19}	$T_1 - T_2 - ..... - T_{18}$	**4**
T_{20}	$T_1 - T_2 - ..... - T_{19}$	**2**
		$\sum = \mathbf{420}$

TABLE 5.1 – Le nombre de calculs des horodatages pour une solution dans MOHEFT

Sommets	Les sommets ayant leurs horodatages recalculés	Le nombre de calculs des horodatages pour le sommet en question
T_1	–	**16**
T_2	T_1	**10**
T_3	$T_1 - T_2$	**12**
T_4	$T_1 - T_2 - T_3$	**8**
T_5	$T_3 - T_4$	**12**
T_6	$T_3 - T_4 - T_5$	**10**
T_7	$T_1 - T_4 - T_5 - T_6$	**8**
T_8	$T_1 - T_2 - T_5 - T_6$ T_7	**6**
T_9	$T_1 - T_3 - T_5 - T_6$ $T_7 - T_8$	**4**
T_{10}	$T_1 - T_2 - T_3 - T_5$ $T_6 - T_7 - T_8 - T_9$	**2**
T_{11}	–	**2**
T_{12}	–	**2**
T_{13}	–	**8**
T_{14}	T_{13}	**6**
T_{15}	$T_{13} - T_{14}$	**4**
T_{16}	$T_{13} - T_{14} - T_{15}$	**2**
T_{17}	–	**2**
T_{18}	–	**2**
T_{19}	–	**2**
T_{20}	–	**2**
		$\sum = \mathbf{120}$

TABLE 5.2 – Le nombre de calculs des horodatages pour une solution dans MOHEFT sans redondances

Dans le tableau 5.2 est répertorié tous les calculs des horodatages sans redondance pour une solution. Pour chaque nouveau mappage d'un sommet tout calcul des sommets prédécesseurs ou des sommets qui ne sont pas en concurrence avec le dernier sommet ajouté est considéré comme redondant. Le nombre total de calculs des horodatages pour une seule solution retenue par MOHEFT est réduit de 420 à 120.

Le nombre de calculs des sommets est de 1715 millions durant le déroulement de MOHEFT pour obtenir 50 solutions sur le workflow Montage 1000 tâches avec neuf ressources ($n = 1000, m = 9, k = 50$).
L'utilisation de l'algorithme 4 réduit le nombre de calculs des sommets à 221 millions grâce aux Checkpointing et le Backtracking , soit, théoriquement, par une accélération de 7,7 $\times$.

À l'inverse des espérances d'accélération (théoriquement calculé), le temps d'exécution de MOHEFT n'a pas été réduit, mais, au contraire, augmenté de 30% minimum. Pour comprendre ce phénomène, nous avons procédé à la mesure du temps d'exécution de chaque ligne dans Algorithme 4 pour identifier la partie gloutonne du code.
Les résultats obtenus indiquent que plus de 79% du temps d'exécution est pris par la ligne 14 et qui consiste à copier les solutions avec les donneés du Checkpointing vers les nouvelles solutions intermédiaires. Le volume des donneés Checkpointing engorge le bus de donnée mémoire.

FAMOBACK est illustré dans l'algorithme 5 qui est une version améliorée de l'Aghoritme 4 par minimisation d'accès à la mémoire. Cette minimisation est effectuée par l'introduction du principe de solution légère.
À l'inverse d'une solution ordinaire contenant des données de mappages et de Checkpointing, la solution légère à l'exception du mappage du dernier sommet ajouté dans la boucle 6-17, elle ne contient aucun mappage ni de données du Checkpointing pour les sommets préalablement traités.
Une solution légère doit impérativement être liée à une et seulement à une seule solution ordinaire. Cette liaison doit permettre a la fonction $\mathcal{FITNESS}$ d'accéder aux mappages et aux données Checkpointing des sommets prédécesseurs à partir d'une solution ordinaire.
À la ligne 14, les données Checkpointing ne sont pas copiées, elles sont seulement remplacées par une référence à ces données. De plus, le code qui génère les données Checkpointing est supprimé de la fonction $\mathcal{FITNESS}$ et placé dans la fonction $First()$ (ligne 17). Au lieu de $k \times m$ seuls les k solutions sélectionnées par la fonction $First()$ auront droit à la génération des données Checkpointing.

Algorithme 5 : MOHEFT using backtracking with checkpointing (FAMOBACH)

Input : W = (N,E), N = $\bigcup_{i=1}^{n} T_i$; // Workflow DAG
Input : $\mathcal{RS}$= $\bigcup_{i=1}^{m} R_i$; // $\mathcal{R}$essource $\mathcal{S}$et
Input : k; // Number of optimal Pareto solutions
Output : $\mathcal{S}ol = \bigcup_{i=1}^{k} schedW$; // Set of K scheduling solutions

1 **begin**
2 $\mathcal{B}_{Activities} \leftarrow$ UP-Rank(N) ; // Order activities according to B-rank
3 **foreach** $s \in \mathcal{S}ol$; // Create K empty Solution
4 **do**
5 s $\leftarrow \phi$
/* from this point MOHEFT is setup and the mapping should begin */
6 **foreach** $T_i \in \mathcal{B}_{Activities}$; // Iterate through all the K ranked activities
7 **do**
8 $\mathcal{S}ol^{'} \leftarrow \phi$
9 **foreach** $R_j \in \mathcal{RS}$; // Iterate over all m resources
10 **do**
11 **foreach** $s_k \in \mathcal{S}ol$; // Iterate over all K schedules in $\mathcal{S}ol$
12 **do**
13 $s^{'} \leftarrow \phi$
14 $s^{'} \leftarrow s_k$; // Referring to schedule with checkpointing data
15 $\mathcal{S}ol^{'} \leftarrow \mathcal{S}ol^{'} \cup \{ s^{'} \cup (T_i, R_j) \}$; // Add mapping to light solution
/* All the T_i of the $(k \times n)$ $s^{'}$ solutions in $\mathcal{S}ol^{'}$ are mapped at the end of these two nested loops. */
16 $\mathcal{S}ol^{'} \leftarrow SortCrowdDist(\mathcal{S}ol^{'})$; // Sorts $\mathcal{S}ol^{'}$ according to crowding distance
17 $\mathcal{S}ol \leftarrow First(\mathcal{S}ol^{'}$,k); // Choose the best k light solutions from sorted $\mathcal{S}ol^{'}$ and return k solutions containing checkpointing data
18 **Return** ($\mathcal{S}ol$)

5.3 Evaluation des performances de FAMOBACK

Pour évaluer FAMOBACK, nous avons effectué des planifications de workflow à l'aide de MOHEFT optimisé. Ensuite, nous avons implémenté la métaheuristique NSGAII avec le même paramétrage que celui de [2], la taille de la population est de 50 individus et le nombre d'itérations est de 1000.
Mis à part les travaux de Xiumin Zhou et al [14], la plupart des comparaisons du temps d'exécution dans la littérature n'ont pas mentionné le temps, sauf un ratio. Pour une comparaison équitable du temps d'exécution, nous avons lancé 50 000 exécutions de la fonction fitness avec des solutions aléatoirement et préalablement générées, le temps d'exécution total est noté 50K_fitness(). 50000 appels a la fonction fitness est le nombre minimum d'appels fitness dans la littérature [2, 3, 14].
Nous utilisons les benchmarks les plus communément utilisés dans la littérature. Les benchmarks sont des workflows scientifiques du monde réel, publiés par le projet Pegasus et initiés par l'Université de Californie du Sud. Les caractéristiques des workflows (nombre de nœuds et nombre d'arêtes) sont résumées dans le Chapitre 4, Tableau 4.2.
Le nombre de solutions retourné par MOHEFT et FAMOBACK est de 50 solutions (k =50) ; neuf instances de ressources sont utilisées(m=9), les mêmes ressources que ceux utilisés pour la comparaison dans [4, 14].
Les caractéristiques de l'ordinateur utilisé pour toutes les méthodes sont un CPU Intel(R) Core(TM) i7-3820 CPU @ 3.60GHz , 16 Go de RAM (2x 8Go) DDR3 1333 MHz CL9 avec un système d'exploitation Windows 10.

Workflow	50K_fitness()	NSGAII	MOHEFT	FAMOBACH
CyberShake 30	2,37	5,63	0,44	0,12
CyberShake 50	5,08	12,05	1,16	0,27
CyberShake 100	15,38	37,03	5,59	1,41
CyberShake 1000	1179,47	2865,04	3179,98	969,53
Epigenomics 24	3,09	6,93	0,37	0,06
Epigenomics 47	9,31	20,39	1,44	0,21
Epigenomics 100	26,79	59,54	7,60	1,36
Epigenomics 997	2218,13	4846,03	4562,17	1113,72
Inspiral 30	3,22	7,44	0,53	0,09
Inspiral 50	6,71	15,39	1,57	0,22
Inspiral 100	20,14	46,75	8,02	0,89
Inspiral 1000	1502,66	3490,56	4803,35	625,37
Montage 25	3,95	8,82	0,41	0,06
Montage 50	11,21	24,51	1,47	0,23
Montage 100	33,42	75,24	7,02	1,11
Montage 1000	2333,10	5232,47	3799,27	758,47
Sipht 29	3,19	7,32	0,50	0,12
Sipht 57	8,73	20,00	2,27	0,58
Sipht 97	19,85	46,21	7,47	1,95
Sipht 968	**1415,18**	3343,50	4195,43	**1476,53**

TABLE 5.3 – Temps d'exécution des quatre algorithmes en seconde

Le Tableau 5.3 représente le temps d'exécution (temps moyen pour 25 exécutions) de 50K_fitness(),NSGAII, MOHEFT et FAMOBACH appliqué sur les différents workflows.

Pour les larges workflows MOHEFT est devenu équivalent en terme de temps d'exécution avec NSGAII. Par conséquent, il peut être utilisé pour les larges workflows.
FAMOBACK est plus rapide que toutes les métaheuristiques auquel MOHEFT a été comparé dans la littérature.

Workflow	50K_fitness()	NSGAII	MOHEFT
CyberShake 30	18,94	45,03	3,50
CyberShake 50	19,13	45,34	4,38
CyberShake 100	10,94	26,35	3,98
CyberShake 1000	1,22	2,96	3,28
Epigenomics 24	49,41	110,94	6,00
Epigenomics 47	44,41	97,19	6,85
Epigenomics 100	19,68	43,73	5,59
Epigenomics 997	1,99	4,35	4,10
Inspiral 30	34,24	79,24	5,64
Inspiral 50	30,41	69,72	7,11
Inspiral 100	22,73	52,77	9,05
Inspiral 1000	2,40	5,58	7,68
Montage 25	**63,14**	141,07	6,50
Montage 50	47,84	104,59	6,28
Montage 100	30,13	67,82	6,32
Montage 1000	3,08	6,90	5,01
Sipht 29	25,52	58,59	4,02
Sipht 57	15,09	34,59	3,92
Sipht 97	10,16	23,66	3,83
Sipht 968	**0,96**	2,26	2,84

TABLE 5.4 – Ratio des temps d'exécution de trois algorithmes par rapport à FAMOBACH

Le tableau 5.4 représente le ratio du temps d'exécution de FAMOBACK sur 50_K Fitness(), NSGAII et MOHEFT.
FAMOBACK a permis de réduire le temps d'exécution de la version optimisée de MOHEFT d'au moins 3,5 ×. La forme du workflow a une influence sur les performances de FAMOBACH. Par ordre décroissant, les meilleures performances de FAMOBACK sont obtenues sur les workflows montage puis Inspirial, Epigmonics, Cyber Shake, et enfin Sipht.
La forme du workflow Sipht est la moins favorable pour le FAMOBACH. Avec Sipht 968, le temps d'exécution de FAMOBACH est légèrement plus long que le 50K_fitness() (4% en plus).
Nous notons que les mesures des temps d'exécution de 50K_fitness() ne prennent pas en compte les différents opérateurs utilisés par les metaheuristique pour générer les solutions. En conséquence, FAMOBACH est la mé-

thode la plus rapide en termes de temps d'exécution.

5.4 Conclusion

L'algorithme déterministe MOHEFT nécessite généralement un grand nombre d'évaluations (de solutions) pour clore le mappage des solutions résultantes , ce qui peut être très lent et coûteux à évaluer.
L'utilisation et la combinaison du chekpounting et du backtraking permettent d'éliminer les redondances dans les évaluations des solutions. Ces deux mécanismes ont accéléré la version MOHEFT de 3.5 $\times$ jusqu'à 9 $\times$.

Conclusion et perspectives

Le cloud computing fournit une infrastructure permettant de résoudre des problèmes de calculs scientifiques à grande échelle. Le modèle de facturation "pay-as-you-go" permet aux fournisseurs de cloud computing de les facturer sur la base de l'utilisation, comme d'autres services publics tels que l'électricité, le gaz et l'eau. Cependant, le déploiement efficace des workflows scientifiques sur ces plateformes dynamiques et hétérogènes est un problème difficile. Cette thèse aborde le problème de l'ordonnancement des workflows scientifiques dans les environnements de cloud computing avec des exigences de QoS. Un grand nombre de méthodes de planification des tâches de workflow sur les ressources cloud ont été développées. L'optimisation de l'efficacité de l'ordonnancement des tâches d'un workflow sur le cloud computing est une fonction essentielle pour les administrateurs système.

Dans cette thèse, nous avons mis en exergue la qualité des résultats de la méthode MOHEFT, représentés sous forme d'un front Pareto, ils ont une très bonne réputation. De plus MOHEFT est une méthode déterministe générique, où l'on peut l'utiliser avec n'importe quels objectifs.
Ce pendant, la méthode MOHEFT n'est pas exploitée avec les workflows volumineux en raison de son temps d'exécution élevé. En effet, nous avons étudié la complexité de MOHEFT qui a été le sujet de divergence entre différents chercheurs. Nous avons validé les résultats de notre analyse empiriquement. L'inefficacité de MOHEFT sur les larges workflows en termes de temps d'exécution est principalement due au nombre d'appels à la fonction fitness et à la quantité d'information transféré vers la RAM par cette dernière.

Le deuxième et principal point abordé dans cette thèse est la réduction de la complexité temporelle de MOHEFT qui permet son utilisation avec les larges workflows. L'accélération de MOHEFT permet aussi l'augmentation de la quantité et de la qualité des résultats retournés. Les contributions d'accélération de MOHEFT se résument en trois approches, les deux pre-

mières sont complémentaires :(i) optimisation par réécriture du code et (ii) l'exploitation des cœurs de la machine pour une parallélisation. Une barrière architecturale a limité les deux premières approches, l'accélération maximale atteinte est de 3.11×.
La troisième approche(iii) est nommée FAMOBACH, c'est une approche de planification de workflow basée sur MOHEFT et utilisant le backtacking et le checkpointing. L'évaluation de FAMOBACK a donné de bons résultats même dans le cas des larges workflows. L'accélération maximale atteinte est de 9×.

Suite aux travaux présentés dans cette thèse, de nouvelles activités de recherche peuvent être lancées afin d'améliorer le travaux présentés. Les perspectives que nous proposons peuvent donc s'orienter vers les directions suivantes :

— L'utilisation de FAMOBACH avec plus de deux objectifs en même temps, pour cela nous prévoyons d'autres objectifs tels que la consommation d'énergie, l'équilibrage de charge, l'utilisation des ressources, la fiabilité et la disponibilité.

— L'utilisation d'autres opérateurs pour l'élagage et la sélection des solutions dans la phase de sélection de FAMOBACH.

— L'hybridation de FAMOBACK avec d'autres heuristiques pour prendre en considération tous les types de tarification dynamique.

References

[1] "Metaheuristics for Multiobjective Optimization". In : *Metaheuristics*. John Wiley Sons, Ltd, 2009. Chap. 4, p. 308-384. eprint : `https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470496916.ch4`. URL : `https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470496916.ch4`.

[2] Zhaomeng Zhu et al. "Evolutionary Multi-Objective Workflow Scheduling in Cloud". In : *IEEE Transactions on Parallel and Distributed Systems* 27.5 (2016), p. 1344-1357.

[3] Z. Chen et al. "Multiobjective Cloud Workflow Scheduling : A Multiple Populations Ant Colony System Approach". In : *IEEE Transactions on Cybernetics* 49.8 (2019), p. 2912-2926.

[4] Quanwang Wu et al. "MOELS : Multiobjective Evolutionary List Scheduling for Cloud Workflows". In : *IEEE Transactions on Automation Science and Engineering* 17.1 (2020), p. 166-176.

[5] María Arsuaga-Ríos et Miguel A. Vega-Rodríguez. "Multiobjective small-world optimization for energy saving in grid environments". In : *Computer Journal* 58.3 (2015), p. 432-447.

[6] Zhicheng Cai et al. "A delay-based dynamic scheduling algorithm for bag-of-task workflows with stochastic task execution times in clouds". In : *Future Generation Computer Systems* 71 (2017), p. 57-72.

[7] Xiaoping Li et Zhicheng Cai. "Elastic Resource Provisioning for Cloud Workflow Applications". In : *IEEE Transactions on Automation Science and Engineering* 14.2 (2017), p. 1195-1210.

[8] Longxin Zhang et al. “Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems”. In : *Information Sciences* 379 (2017), p. 241-256.

[9] Guangshun Yao et al. “Endocrine-based coevolutionary multi-swarm for multi-objective workflow scheduling in a cloud system”. In : *Soft Computing* 21.15 (2017), p. 4309-4322.

[10] Jianfeng Chen et Tim Menzies. “RIOT : A Stochastic-Based Method for Workflow Scheduling in the Cloud”. In : *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)* (2018), p. 318-325. url : https://ieeexplore.ieee.org/document/8457815/.

[11] Ilia Pietri, Yannis Chronis et Yannis Ioannidis. “Multi-objective optimization of scheduling dataflows on heterogeneous cloud resources”. In : *Proceedings - 2017 IEEE International Conference on Big Data, Big Data 2017* 2018-Janua (2018), p. 361-368.

[12] Haiyang Hu et al. “Multi-objective scheduling for scientific workflow in multicloud environment”. In : *Journal of Network and Computer Applications* 114 (2018), p. 108-122. url : https://doi.org/10.1016/j.jnca.2018.03.028.

[13] César Gómez-Martín et Miguel A. Vega-Rodríguez. “Optimization of resources in parallel systems using a multiobjective artificial bee colony algorithm”. In : *Journal of Supercomputing* 74.8 (2018), p. 4019-4036. url : https://doi.org/10.1007/s11227-018-2407-5.

[14] Xiumin Zhou et al. “Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT”. In : *Future Generation Computer Systems* 93 (2019), p. 278-289. url : https://doi.org/10.1016/j.future.2018.10.046.

[15] Somayeh Mohammadi, Latif PourKarimi et Hossein Pedram. “Integer linear programming-based multi-objective scheduling for scientific workflows in multi-cloud environments”. In : *Journal of Supercomputing* 75.10 (2019), p. 6683-6709. url : https://doi.org/10.1007/s11227-019-02877-8.

[16] Ali Asghari, Mohammad Karim Sohrabi et Farzin Yaghmaee. “learning agents and genetic algorithm”. In : *The Journal of Supercomputing* 0123456789 (2020). url : https://doi.org/10.1007/s11227-020-03364-1.

[17] Mohammed Ridha Bouzidi et al. “New Search Based Methods to Solve Workflow Scheduling Problem in Cloud Computing”. In : *2018 5th International Conference on Control, Decision and Information Technologies, CoDIT 2018* (2018), p. 647-652.

[18] Michael Armbrust et al. *Above the Clouds : A Berkeley View of Cloud Computing*. Rapp. tech. UCB/EECS-2009-28. EECS Department, University of California, Berkeley, fév. 2009. URL : `http://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html`.

[19] Ulrich Drepper. *What Every Programmer Should Know About Memory*. 2007.

[20] microsoft. *The Azure cloud platform*. URL : `https://azure.microsoft.com/en-us/` (visité le 30/09/2021).

[21] microsoft. *Azure Active Directory (Azure AD)*. URL : `https://azure.microsoft.com/en-us/services/active-directory/` (visité le 30/09/2021).

[22] May Al-Roomi et al. “Cloud Computing Pricing Models : A Survey”. In : *International Journal of Grid and Distributed Computing* 6 (2013), p. 93-106.

[23] Caesar Wu, R. Buyya et K. Ramamohanarao. “Cloud Pricing Models”. In : *ACM Computing Surveys (CSUR)* 52 (2020), p. 1-36.

[24] Amazon. *Amazon Web Services*. URL : `https://aws.amazon.com/` (visité le 30/09/2021).

[25] Google. *Google Cloud*. URL : `https://cloud.google.com/` (visité le 30/09/2021).

[26] G. George et al. “Analyzing AWS Spot Instance Pricing”. In : *2019 IEEE International Conference on Cloud Engineering (IC2E)* (2019), p. 222-228.

[27] Wenqiang Liu et al. “Cloud spot instance price prediction using kNN regression”. In : *Human-centric Computing and Information Sciences* 10 (2020).

[28] IBM. *IBM Cloud*. URL : `https://www.ibm.com/cloud` (visité le 30/09/2021).

[29] arsys. *arsys Cloud*. URL : `https://www.arsys.net/servers/cloud` (visité le 30/09/2021).

[30] CLOUDSIGMA. URL : https://fra.cloudsigma.com/ (visité le 30/09/2021).

[31] Ustun YILDIZ, Adnene GUABTNI et Anne H.H. NGU. "Business versus Scientific Workflows : A Comparative Study". In : *2009 Congress on Services - I.* 2009, p. 340-343.

[32] PEGASUS. URL : https://pegasus.isi.edu/ (visité le 30/09/2021).

[33] Joseph C. JACOB et al. "Montage : A Grid Portal and Software Toolkit for Science-Grade Astronomical Image Mosaicking". In : *Int. J. Comput. Sci. Eng.* 4.2 (juil. 2009), p. 73-87. URL : https://doi.org/10.1504/IJCSE.2009.026999.

[34] R. GRAVES et al. "CyberShake : A Physics-Based Seismic Hazard Model for Southern California". In : *Pure and Applied Geophysics* 168 (2011), p. 367-381.

[35] Ewa DEELMAN et al. "Workflows and e-Science : An overview of workflow system features and capabilities". In : *Future Generation Computer Systems* 25.5 (2009), p. 528-540. URL : https://www.sciencedirect.com/science/article/pii/S0167739X08000861.

[36] Ewa DEELMAN et Yolanda GIL. "Managing Large-Scale Scientific Workflows in Distributed Environments : Experiences and Challenges". In : *2006 Second IEEE International Conference on e-Science and Grid Computing (e-Science'06)*. 2006, p. 144-144.

[37] Jia YU, R. BUYYA et K. RAMAMOHANARAO. "Workflow scheduling algorithms for grid computing". In : 2008.

[38] Tracy D BRAUN et al. "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems". In : *J. Parallel Distrib. Comput.* 61.6 (juin 2001), p. 810-837. URL : https://doi.org/10.1006/jpdc.2000.1714.

[39] O. UDOMKASEMSUB, LI XIAORONG et T. ACHALAKUL. "A multiple-objective workflow scheduling framework for cloud data analytics". In : *2012 Ninth International Conference on Computer Science and Software Engineering (JCSSE)*. 2012, p. 391-398.

[40] Amandeep VERMA et Sakshi KAUSHAL. "Cost-Time Efficient Scheduling Plan for Executing Workflows in the Cloud". In : *Journal of Grid Computing* 13.4 (2015), p. 495-506.

[41] Saeid Abrishami, Mahmoud Naghibzadeh et Dick H.J. Epema. “Deadline-Constrained Workflow Scheduling Algorithms for Infrastructure as a Service Clouds”. In : *Future Gener. Comput. Syst.* 29.1 (jan. 2013), p. 158-169. URL : https://doi.org/10.1016/j.future.2012.05.004.

[42] J. Yu, M. Kirley et R. Buyya. “Multi-objective planning for workflow execution on Grids”. In : *2007 8th IEEE/ACM International Conference on Grid Computing*. 2007, p. 10-17.

[43] E. Zitzler, M. Laumanns et L. Thiele. “SPEA2 : Improving the strength pareto evolutionary algorithm”. In : 2001.

[44] L. Adhianto et al. “HPCTOOLKIT : Tools for performance analysis of optimized parallel programs”. In : *Concurrency Computation Practice and Experience* 22.6 (2010), p. 685-701.

[45] Ritu Garg et Awadhesh Kumar Singh. “Multi-objective workflow grid scheduling based on discrete particle swarm optimization”. In : *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7076 LNCS.PART 1 (2011), p. 183-190.

[46] J. Kennedy et R. Eberhart. “Particle swarm optimization”. In : *Proceedings of ICNN’95 - International Conference on Neural Networks* 4 (1995), 1942-1948 vol.4.

[47] Ritu Garg et Awadhesh Kumar Singh. “Multi-objective workflow grid scheduling using varepsilon -fuzzy dominance sort based discrete particle swarm optimization”. In : *The Journal of Supercomputing* 68.2 (2014), p. 709-732.

[48] Juan J. Durillo, Radu Prodan et Weicheng Huang. “Workflow scheduling in Amazon EC2”. In : *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8374 LNCS.February 2013 (2014), p. 374-383.

[49] H. Topcuoglu, S. Hariri et Min-You Wu. “Performance-effective and low-complexity task scheduling for heterogeneous computing”. In : *IEEE Transactions on Parallel and Distributed Systems* 13.3 (2002), p. 260-274.

[50] Maria Alejandra Rodriguez. “RESEARCH ARTICLE A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments”. In : October (2016), p. 1-23.

[51] Juan J. Durillo, Radu Prodan et Jorge G. Barbosa. “Pareto tradeoff scheduling of workflows on federated commercial Clouds”. In : *Simulation Modelling Practice and Theory* 58.February (2015), p. 95-111. url : http://dx.doi.org/10.1016/j.simpat.2015.07.001.

[52] Hamid Mohammadi Fard, Sasko Ristov et Radu Prodan. “Handling the Uncertainty in Resource Performance for Executing Workflow Applications in Clouds”. In : *IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC)* (2016), p. 89-98.

[53] João M.P. Cardoso, José Gabriel F. Coutinho et Pedro C. Diniz. “Chapter 5 - Source code transformations and optimizations”. In : *Embedded Computing for High Performance*. Sous la dir. de João M.P. Cardoso, José Gabriel F. Coutinho et Pedro C. Diniz. Boston : Morgan Kaufmann, 2017, p. 137-183. url : http://www.sciencedirect.com/science/article/pii/B9780128041895000053.

[54] André R. Brodtkorb et al. “GPU computing in discrete optimization. Part I : Introduction to the GPU”. In : *EURO Journal on Transportation and Logistics* 2.1 (2013), p. 129-157. url : https://www.sciencedirect.com/science/article/pii/S2192437620600267.

[55] Mohammed Redha Bouzidi et al. “FAMOBACH : A fast and survivable workflow scheduling approach based MOHEFT using backtacking and checkpointing”. In : *Comput. Commun.* 171 (2021), p. 16-27.